CSE587 Assignment 3

Predictive Analytics with Spark

Team members:

Name	UBIT id
Astrid Gomes	ASTRIDYV
Mansi Wagh	MANSIWAG
Rasita Pai	RASITASH

We had to run the code on Google Colab as we were having issues running on the VM. Below are the steps:

1. To run the code on google colab, install Pyspak and Java8

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.5/spark-2.4.5-bin-hadoop2.7.tgz
!tar -xvf spark-2.4.5-bin-hadoop2.7.tgz
!pip install -q findspark
```

Set the environment variables:

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.5-bin-hadoop2.7"
```

2. Start a Spark session and load the test and train csv files into pyspark dataframes.
3. Load the mapping.csv file and rename the empty header of labels to *label* and header of genres as *genre* [1] and store the genres as a list [2], which is used further for one-hot-encoding the string of genres.
4. Preprocessing performed on the data
   a. We were initially running into error as some entries of train data had null values, so we created another copy of genre column and select only the entries which had no null values.
5. We created an one-hot-encoder function to generate a string consisting of 1's and 0's (length 20), here we split the genre string on comma and remove the quotes and braces at the beginning and end of string as part of preprocessing each string and replace the genre location with 1 if present otherwise replace with 0. This is done using user-defined-function.

Task 1: - Basic Model

Here we have created a basic pipeline and passed it to a machine learning model (we used Logistic Regression)

1. Transformed the input column ‘plot’ to ‘tokens’ using the *regexTokenizer* feature transformer [3].
2. Used the *countVectorizer* transformer and fitted the train data to convert a collection of tokens to the vectors of token counts. CountVectorizer converts text documents to vectors of term counts. This forms our features for this model [4].
3. Once the feature vectors are generated, we transform the test data also for this same pipeline.
4. Now for each feature vector formed, we split the one hot encoded string and are creating a genre column for each of the 20 genres present.
5. We are using the Logistic Regression machine learning model to predict the labels of our test data using the feature vectors generated and labels as input.
6. Since we have 20 genres, we fit the logistic regression model 20 times.
7. Once the logistic regression model is fitted the labels and the features are transformed so the predictions can be generated.
8. Then we perform left outer join to get the data from test data and its generated genre labels in the same sequence as in test data.
9. To convert our generated predictions to required format of output, we typecast the predicted labels to integer type and form a new column containing the string formed by concatenating all the predicted labels separated by space between them [5].
10. We finally write the modified dataframe to csv file to be uploaded to Kaggle.

Macro F1 score- 97.334

Output file in Kaggle- op1.csv

Task 2: - Use TF-IDF to improve the model

Using the Logistic Regression model generated in the previous step we applied Term frequency-inverse document frequency (TF-IDF) to improve the pipelined model. The term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus [6].

TF: We have used the CountVectorizer to generate the term frequency vectors. For this we have used the features obtained from the task 1.
IDF: IDF is an estimator which is fit on a dataset and produces the IDFmodel. The IDFModel takes feature vectors created from CountVectorizer and scales each column. Intuitively, it down-weights columns which appear frequently in a corpus.

1. We use the idf to generate the TF-IDF features from the features obtained in the task 1.
2. The idfmodel is fitted using the training data.
3. This model is then used to transform the train data to generate the raw tfidf features.
4. Once the feature vectors are generated, we transform the test data also for this same pipeline.

5. Now for each feature vector formed, we split the one hot encoded string and are creating a genre column for each of the 20 genres present.
6. We are using the Logistic Regression machine learning model to predict the labels of our test data using the feature vectors generated and labels as input.
7. Since we have 20 genres, we fit the logistic regression model 20 times.
8. Once the logistic regression model is fitted the labels and the features are transformed so the predictions can be generated.
9. Then we perform left outer join to get the data from test data and its generated genre labels in the same sequence as in test data [7].
10. To convert our generated predictions to required format of output, we typecast the predicted labels to integer type and form a new column containing the string formed by concatenating all the predicted labels separated by space between them.
11. We finally write the modified dataframe to csv file to be uploaded to Kaggle.

Macro F1 score- 98.156

Output file in Kaggle- op2.csv

Task 3: - Use Custom feature Engineering to improve the model

1. Transformed the input column ‘plot’ to ‘tokens’ using the *regexTokenizer* feature transformer and used the stopWordRemover to remove stop-words from the tokens formed.
2. The feature engineering technique used is the hashingTF to generate features.
3. On this transformed data, we used the *MaxAbsScaler* to further scale the features.
4. This model is then used to transform the train data to generate the features.
5. Once the feature vectors are generated, we transform the test data also for this same pipeline.
6. Now for each feature vector formed, we split the one hot encoded string and are creating a genre column for each of the 20 genres present.
7. We are using the Logistic Regression machine learning model to predict the labels of our test data using the feature vectors generated and labels as input.
8. Since we have 20 genres, we fit the logistic regression model 20 times.
9. Once the logistic regression model is fitted the labels and the features are transformed so the predictions can be generated.
10. Then we perform left outer join to get the data from test data and its generated genre labels in the same sequence as in test data [7].
11. To convert our generated predictions to required format of output, we typecast the predicted labels to integer type and form a new column containing the string formed by concatenating all the predicted labels separated by space between them.
12. We finally write the modified dataframe to csv file to be uploaded to Kaggle.

Macro F1 score- 97.506

Output file in Kaggle- op3.csv

References:

[1] https://sparkbyexamples.com/pyspark/pyspark-rename-dataframe-column/
[2] https://backtobazics.com/big-data/spark/apache-spark-flatmap-example/
[3] https://spark.apache.org/docs/latest/ml-features#tokenizer
[4] https://spark.apache.org/docs/latest/ml-features#countvectorizer
[5]https://stackoverflow.com/questions/31450846/concatenate-columns-in-apache-spark-dataframe
[6] https://spark.apache.org/docs/latest/ml-features#tf-idf
[7] http://www.learnbymarketing.com/1100/pyspark-joins-by-example/

Other references:

https://docs.python.org/3.1/library/re.html
https://pythonspot.com/tokenizing-words-and-sentences-with-nltk/
https://www.kaggle.com/guilherme93/nlp-movie-genre-prediction-from-plot
https://github.com/databricks/spark-training/blob/master/website/movie-recommendation-with-mllib.md
https://towardsdatascience.com/multi-class-text-classification-with-pyspark-7d78d022ed35
https://davefernig.com/2016/05/07/building-a-term-document-matrix-in-spark/
https://benalexkeen.com/multiclass-text-classification-with-pyspark/
https://towardsdatascience.com/countvectorizer-hashingtf-e66f169e2d4e
https://spark.apache.org/docs/latest/ml-features
http://www.learnbymarketing.com/1100/pyspark-joins-by-example/
https://stackoverflow.com/questions/53579444/efficient-text-preprocessing-using-pyspark-clean-tokenize-stopwords-stemming
https://medium.com/@dhiraj.p.rai/logistic-regression-in-spark-ml-8a95b5f5434c
https://spark.apache.org/docs/latest/mllib-clustering.html#latent-dirichlet-allocation-lda
https://medium.com/@kunalgupta4595/predicting-movie-genres-based-on-plot-summaries-bae646e70e04
https://www.kaggle.com/dhirajrai87/feature-engineering-with-pyspark#Feature-selection
https://www.kaggle.com/guilherme93/nlp-movie-genre-prediction-from-plots