MET CS 777 – Big Data Analytics
Term Project
Professor Dimitar Trajanov
Boston University
Spring 2 2022
Muniba Shaikh
04/24/2022

Sentiment Analysis of IMDB dataset

Data Set Description:
IMDB dataset is obtained from Kaggle. IMDB dataset have 50K highly polar movie reviews. Each row in the data has a review and a sentiment (positive and negative) about a movie.
The dataset has two features: review and sentiment (positive and negative).

Data Set Link https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?select=IMDB+Dataset.csv

Classification Problem:
My goal is to build a classifier that can predict whether a review about a movie is positive or negative based on the review text only.

Features:
There are two features in the dataset:
1) **review:** text
2) **sentiment:** "positive" or "negative"

Besides the above two features, I added a class label for sentiment feature (0: negative sentiment, 1: positive sentiment).

review	sentiment	label
One of the other ...	positive	1
A wonderful littl...	positive	1
I thought this wa...	positive	1
Basically there's...	negative	0
Petter Mattei's "...	positive	1
Probably my all-t...	positive	1
I sure would like...	positive	1
This show was an ...	negative	0
Encouraged by the...	negative	0
If you like origi...	positive	1

Split the dataset into training and test set:
IMDB dataset is split into train and test set using 70/30 split size.

Training Data

label	count
1	17450
0	17543

Test Data

label	count
1	7550
0	7457

Data Preprocessing:
Following data preprocessing tasks are performed on the *IMDB* data using pipeline functionality of MlLib Spark's machine learning (ML) library for text mining purposes.

Tokenization:

- All non-letter characters are removed from the review text
- Review text is converted to lower case.
- Review is tokenized into individual words.

Removing stop words:

- Stop words such as *a, the, is, are, etc.* are removed from the review text because they appear frequently and don't carry as much meaning.
- Some stop words such as *"br", 'm', 've', 're', 'll', 'd'*, are determined through eye-balling and removed from the review text.

```
|[turkish, bath, sequence, film, noir, located, new, york, 50, must, hint, something, something, curiously, previous, comments, one,
pointed, seems, essential, understanding, movie, turkish, baths, sequence, back, street, night, entrance, sleazy, sauna, scalise, wrapped,
sheet, getting, thighs, massaged, steve, masseur, young, rough, boxer, beefcake, type, another, guy, bodyguard, finishes, dressing, dixon,
obviously, hates, sees, gets, rough, right, away, know, reputation, roughing, suspects, good, cop, getting, control, easy, hates, much,
hates, part, inherited, father, dark, side, lead, right, end, sidewalk, gutter, dark, side, lurked, within, closet, remember, whenever,
dixon, meets, scalise, 3, times, guy, lying, bed, men, around, company, irony, girls, poster, pinned, wall, near, bed, scalise, acts,
```

Bag of Words:
A vocabulary of words is extracted from reviews collections and the top 5000 words ordered by their term frequency across the corpus are selected using CountVectorizerModel.

```
+-----------------------+
|Top 10 vocabulary words|
+-----------------------+
|movie                  |
|film                   |
|one                    |
|like                   |
|good                   |
|time                   |
|even                   |
|story                  |
|really                 |
|see                    |
+-----------------------+
```

Feature vectors:
Vocabulary of 5000 words are then used to vectorize the review text into feature vectors using CountVectorizerModel.

IDF:
Then IDF Model is applied to feature vectors to down-weight features which appear frequently in a corpus.

```
+---------+-----+--------------------+
|sentiment|label|            features|
+---------+-----+--------------------+
| positive|    1|(5000,[0,1,2,3,4,...|
| negative|    0|(5000,[0,1,2,3,6,...|
| negative|    0|(5000,[0,3,4,7,11...|
| negative|    0|(5000,[1,3,5,6,17...|
| negative|    0|(5000,[1,2,9,11,1...|
| negative|    0|(5000,[0,1,3,7,15...|
| positive|    1|(5000,[0,1,3,7,8,...|
| positive|    1|(5000,[1,2,3,5,7,...|
| negative|    0|(5000,[0,8,16,18,...|
| negative|    0|(5000,[1,3,10,25,...|
+---------+-----+--------------------+
```

Chi-Squared feature selection:
ChiSqSelector uses the Chi-Squared test of independence to decide which features to choose. It operates on labeled data with categorical features. Top 500 features are determined by Chi-Squared feature selection method to be used in the classification of reviews. Now the data is ready for the applying classifiers and predicting sentiment labels for reviews.

Classifiers used on the dataset:
SVM and *logistic regression* classifiers are trained on the train-set and then labels are predicted for the test-set.

Support Vector Machine Classifier:

First 10 beta coefficients: [0.00980576 0.06211804 -0.06753853 0.01698218 0.13457261 0.04795967 -0.21343299 0.27376441 0.09719346 -0.03554475]

Intercept: 0.13717365823289823

Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	6467	828
Actual Positive	990	6722

Performance Metrics

Analytics	Value
Accuracy	0.878857
Precision (Class 0)	0.867238
Recall (Class 0)	0.886498
Precision (Class 1)	0.890331
Recall (Class 1)	0.871629
F1-Measure	0.880881

Computation Time

Operation	Time (s)
Model Training	67.907753
Testing Model	0.26406

Performance Metrics	24.151414
Total Time	92.323227

Logistic Regression Classifier:

First 10 beta coefficients: [9.45340869e-05 7.23486328e-02 -1.05165477e-01 2.59919535e-02 1.83762811e-01 6.27347557e-02 -2.95955687e-01 3.80198763e-01 1.36041935e-01 -5.62965598e-02]

Intercept: 0.13953873111771362

Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	6491	844
Actual Positive	966	6706

Performance Metrics

Analytics	Value
Accuracy	0.8793896
Precision (Class 0)	0.870457
Recall (Class 0)	0.884935
Precision (Class 1)	0.888212
Recall (Class 1)	0.874088
F1-Measure	0.881093

Computation Time

Operation	Time (s)
Model Training	75.9268169
Testing Model	0.364875
Performance Metrics	24.945547
Total Time	101.237239

Comparison of performance measures between classifiers

Model	**Recall (Class 0)**	**Recall (Class 1)**	**Accuracy%**	**Computation Time(s)**
SVM	0.886498	0.871629	87.8	92.323227
Logistic Regression	0.884935	0.874088	87.9	101.237239

Accuracy of prediction and TPR is the same for both the classifiers. However, SVM is faster than Logistic Regression.

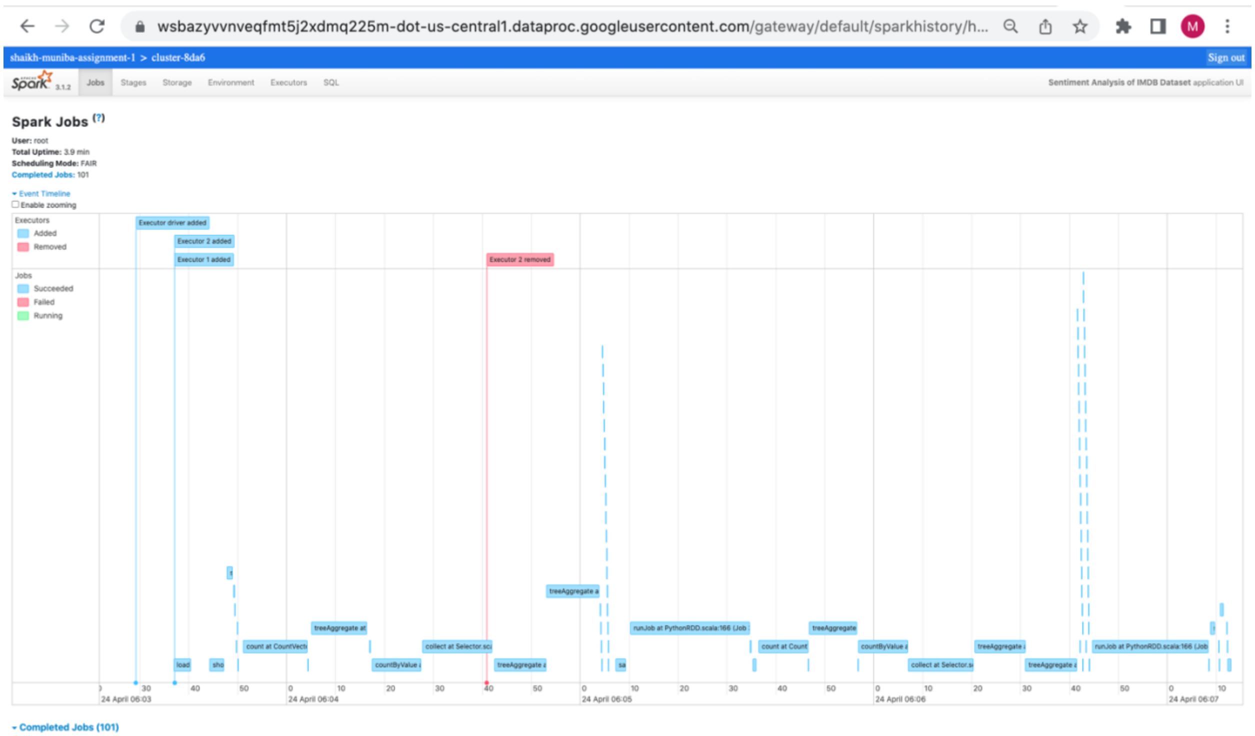
Predicting sentiment on new data using SVM model:

```
Prediction using SVM model:
+--------------------------------------------------------+----------+
|review                                                  |prediction|
+--------------------------------------------------------+----------+
|This movie was horrible, plot was boring, acting was okay.|0.0       |
|The film really sucked. I want my money back            |0.0       |
|What a beautiful movie. Great plot, great acting.       |1.0       |
|Harry Potter was a good movie.                          |1.0       |
+--------------------------------------------------------+----------+
```

Predicting sentiment on new data using Logistic Regression model:

```
Prediction using Logistic Regression model:
+-----------------------------------------------------------+----------+
|review                                                     |prediction|
+-----------------------------------------------------------+----------+
|This movie was horrible, plot was boring, acting was okay.|0.0       |
|The film really sucked. I want my money back               |0.0       |
|What a beautiful movie. Great plot, great acting.          |1.0       |
|Harry Potter was a good movie.                             |1.0       |
+-----------------------------------------------------------+----------+
```

Spark History:



Google Cloud link to PySpark Script:
gs://muniba-met-cs-777/CS777-Term-Project-Sentiment-Analysis/run_Term_Project_Sentiment_Analysis.py

Google Cloud link to results:
gs:// muniba-met-cs-777/CS777-Term-Project-Sentiment-Analysis/output/

Google Cloud link to dataset:
gs://muniba-met-cs-777/CS777-Term-Project-Sentiment-Analysis/IMDB_Dataset.csv

Appendix A - *run_Term_Project_Sentiment_Analysis.py*

```
from __future__ import print_function
import os
import sys
import requests
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql import functions as F
from pyspark.sql.functions import *
from pyspark.sql.types import StringType, IntegerType
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.ml.feature import StopWordsRemover
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.feature import IDF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import LinearSVC
from pyspark.ml import Pipeline
from pyspark.ml.feature import ChiSqSelector
from pyspark.mllib.evaluation import MulticlassMetrics
import time

# Function to predict the sentiment(positive, negative) of text for different classifiers
def getPrediction(text, model):
    # Check if text is one review or a list of reviews
    if (isinstance(text, str)):
        review = [text]
    else:
        review = text

    # Create a dataframe of review list
    df_new_data = spark.createDataFrame(review, StringType())

    # Rename the dataframe column
    df_new_data = df_new_data.withColumnRenamed("value", "review")

    # Predict sentiment using the SVM model
    predict = model.transform(df_new_data)
    predict = predict.select("review", "prediction")
    return predict

# main() starts here
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: wordcount <file> <output> ", file=sys.stderr)
        exit(-1)

    # Create spark context
    spark = SparkSession \
        .builder \
        .appName("Sentiment Analysis of IMDB Dataset") \
        .getOrCreate()

    # Set your file path here
    # Google cloud path
```

```
    data_file = "gs://muniba-met-cs-777/CS777-Term-Project-Sentiment-Analysis/IMDB_Dataset.csv"

    #data_file =
"/Users/munibasiddiqi/Desktop/BUCS777/Homework_Assignments/Term_Project/IMDB_Dataset.csv"

    # Colab path
    #data_file = "IMDB_Dataset.csv"

    # Upload data into a dataframe
    spark_df = spark.read.format("csv").option("header",
"true").option("escape","\"").option("multiLine","true").load(data_file)

    # Add label column to dataframe (Positive sentiment = 1, Negative sentiment = 0)
    df = spark_df.withColumn("label", F.when(F.col("sentiment")=="positive",1).otherwise(0)).cache()
    df.show(10)

    # Split the dataset into training and test set
    df_train, df_test = df.randomSplit(weights=[0.7, 0.3], seed=100)

    # Check if the dataset is balanced or imbalanced
    print("Taining Data")
    df_train.groupby("label").count().show()
    print("Test Data")
    df_test.groupby("label").count().show()

    ################################ Data Preprocessing ######################################

    # Converts text to lowercase and split text on non-word character
    regexTokenizer = RegexTokenizer(inputCol="review", outputCol="words", pattern="\\W")

    # Remove stopwords
    remover = StopWordsRemover(inputCol = regexTokenizer.getOutputCol(), outputCol="filtered")

    # Remove stopWords=["br", 'm', 've', 're', 'll', 'd']
    remover2 = StopWordsRemover(inputCol= remover.getOutputCol(), outputCol="token", stopWords=["br",
'm', 've', 're', 'll', 'd'])

    # Extracts a vocabulary from document collections and generates a CountVectorizerModel
    # During the fitting process, CountVectorizer will select the top vocabSize words ordered
    # by term frequency across the corpus.
    countVectorizer = CountVectorizer(inputCol= remover2.getOutputCol(), outputCol="rawFeatures",
vocabSize=5000)

    # The IDF Model takes feature vectors and scales each feature.
    # Intuitively, it down-weights features which appear frequently in a corpus
    idf = IDF(inputCol= countVectorizer.getOutputCol(), outputCol="featuresIDF")

    # Chi-Squared feature selection. It operates on labeled data with categorical features.
    # ChiSqSelector uses the Chi-Squared test of independence to decide which features to choose.
    selector = ChiSqSelector(numTopFeatures=500, featuresCol=idf.getOutputCol(),
                    outputCol="features", labelCol="label")

################################ LOGISTIC REGRESSION ####################################

    #################### Training Model ####################
```

```
    # Start time
    begin_time = time.time()

    # LogisticRegression classifier
    classifier_logreg = LogisticRegression(maxIter=20)

    # Chain indexers and classifier_logreg in a Pipeline
    pipeline_logreg = Pipeline(stages=[regexTokenizer, remover, remover2, countVectorizer, idf, selector,
classifier_logreg])

    # Train model.
    model_logreg = pipeline_logreg.fit(df_train)

    # Print the coefficients and intercept for linear SVC
    print("Logistic Regression Model")
    print("First 10 Coefficients: " + str(model_logreg.stages[6].coefficients[:10]))
    print("Intercept: " + str(model_logreg.stages[6].intercept))

    # Top 20 vocabulary words
    #pipeline_logreg.getStages()
    vocabulary = model_logreg.stages[3].vocabulary
    print("Top twenty vocabulary words", vocabulary[0:20])

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to train logistic regression model on the train data: ", end_time)

    # Create a dataframe of top 20 vocabulary words to save as csv file
    df_top20 = spark.createDataFrame(vocabulary[0:20], StringType())

    # Store this result in a single file on the cluster
    df_top20.coalesce(1).write.format("csv").option("header",True).save(sys.argv[1]+'.top20_words_IDF')

    #################### Model Testing ####################

    # Start time
    begin_time = time.time()

    # Make predictions.
    predictions_logreg = model_logreg.transform(df_test).cache()

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to test logistic regression model on the test data: ", end_time)

    #################### Model evaluation ####################

    # Start time
    begin_time = time.time()

    # Covert dataframe to RDD for Model evaluation
    predictionAndLabels_logreg = predictions_logreg.select("label",  "prediction").rdd.map(lambda x :
(float(x[0]), float(x[1]))).cache()

    # Instantiate metrics object
    metrics_logreg = MulticlassMetrics(predictionAndLabels_logreg)
```

```
    # Statistics by class
    #labels = data.map(lambda lp: lp.label).distinct().collect()
    print("Summary statistics for Logistic regression classifier")
    labels = [0.0, 1.0]
    for label in sorted(labels):
        print("Class %s precision = %s" % (label, metrics_logreg.precision(label)))
        print("Class %s recall = %s" % (label, metrics_logreg.recall(label)))
        print("Class %s F1 Measure = %s" % (label, metrics_logreg.fMeasure(label, beta=1.0)))

    print("Accuracy = %s" % metrics_logreg.accuracy)
    print("Confusion Matrix")
    print(metrics_logreg.confusionMatrix().toArray().astype(int))

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to evalute the performance of logistic regression model on test data: ", end_time)

    # Create a dataframe to store the summary of results
    data = [("Accuracy", str(metrics_logreg.accuracy)),("Confusion
Matrix",str(metrics_logreg.confusionMatrix().toArray()))]
    df = spark.createDataFrame(data)

    # Store this result in a single file on the cluster
    df.coalesce(1).write.format("csv").option("header",True).save(sys.argv[1]+'.logreg_statistics')



 ############################## SUPPORT VECTOR MACHINE ################################

    #################### Training Model ###################

    # Start time
    begin_time = time.time()

    # SVM classifier
    classifier_lsvc = LinearSVC(maxIter=20)

    # Fit the model
    #lsvcModel = classifier.fit(df_train)

    # Chain indexers and classifier_lsvc in a Pipeline
    pipeline_lsvc = Pipeline(stages=[regexTokenizer, remover, remover2, countVectorizer, idf, selector,
classifier_lsvc])

    # Train model.
    model_lsvc = pipeline_lsvc.fit(df_train)

    # Print the coefficients and intercept for linear SVC
    print("Support Vector Machine Model")
    print("First 10 Coefficients: " + str(model_lsvc.stages[6].coefficients[:10]))
    print("Intercept: " + str(model_lsvc.stages[6].intercept))

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to train SVM model on the train data: ", end_time)
```

```
    #################### Model Testing ###################

    # Start time
    begin_time = time.time()

    # Make predictions.
    predictions_lsvc = model_lsvc.transform(df_test).cache()

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to test SVM model on the test data: ", end_time)

    #################### Model evaluation ###################

    # Start time
    begin_time = time.time()

    # Covert dataframe to RDD for Model evaluation
    predictionAndLabels_lsvc = predictions_lsvc.select("label",  "prediction").rdd.map(lambda x : (float(x[0]),
float(x[1]))).cache()

    # Instantiate metrics object
    metrics_lsvc = MulticlassMetrics(predictionAndLabels_lsvc)


    # Statistics by class
    #labels = data.map(lambda lp: lp.label).distinct().collect()
    labels = [0.0, 1.0]
    for label in sorted(labels):
        print("Class %s precision = %s" % (label, metrics_lsvc.precision(label)))
        print("Class %s recall = %s" % (label, metrics_lsvc.recall(label)))
        print("Class %s F1 Measure = %s" % (label, metrics_lsvc.fMeasure(label, beta=1.0)))

    print("Accuracy = %s" % metrics_lsvc.accuracy)
    print(metrics_lsvc.confusionMatrix().toArray().astype(int))

    # End time
    end_time = time.time() - begin_time
    print("Total execution time to evalute the performance of SVM model on test data: ", end_time)

    # Create a dataframe to store the summary of results
    data = [("Accuracy", str(metrics_lsvc.accuracy)),("Confusion
Matrix",str(metrics_lsvc.confusionMatrix().toArray()))]
    df = spark.createDataFrame(data)

    # Store this result in a single file on the cluster
    df.coalesce(1).write.format("csv").option("header",True).save(sys.argv[1]+'.SVM_statistics')

######################### Predicting Sentiments on New Data (Reviews)  #########################

   # A list of reviews
    new_data = ['This movie was horrible, plot was boring, acting was okay.',
               'The film really sucked. I want my money back',
               'What a beautiful movie. Great plot, great acting.',
               'Harry Potter was a good movie.'
```

```
        ]

################# Prediction using logistic regression model ###############

# Call to getPrediction function with a list of reviews and logistic regression model
predict = getPrediction(new_data, model_logreg)
print("Prediction using Logistic Regression model:")
predict.show(truncate=0)

# Store this result in a single file on the cluster
predict.coalesce(1).write.format("csv").option("header",True).save(sys.argv[1]+'.logreg_prediction')

##################### Prediction using SVM model ####################

# Call to getPrediction function with a list of reviews and SVM model
predict = getPrediction(new_data, model_lsvc)
print("Prediction using SVM model:")
predict.show(truncate=0)

# Store this result in a single file on the cluster
predict.coalesce(1).write.format("csv").option("header",True).save(sys.argv[1]+'.SVM_prediction')

# Stop spark context
spark.stop()
```