1. Info

CS777 Big Data Analytics Term Project

Yelp Reviews Sentiment Analysis

Assiya Karatay, Euiyoung Lee

2. Description

In this project we will demonstrate a supervised learning model for classification of sentiments with a sample of Yelp reviews data and vector labels over two types of sentiments.

3. Import libraries

In [1]:
```
!pip install pyspark==3.1.2
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
heels/public/simple/
Collecting pyspark==3.1.2
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |████████████████████████████████| 212.4 MB 61 kB/s
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 10.9 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=
212880769 sha256=135e4ba3cabd49cd19b1bfc5af06a36d125dc920465d685bb6c835fb0660a
709
  Stored in directory: /root/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863
dcd846daaa95f598744e71b02c77
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.2
```

In [2]:
```
#import libraries
from pyspark import SparkContext
from pyspark.sql import SparkSession ,Row
from pyspark.sql.functions import col
from pyspark.sql import SQLContext
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.types import StructType,StructField,IntegerType,StringType,Flo
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
from google.colab import drive
import os
```

```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:
```
spark = SparkSession.builder\
          .appName("SentimentAnalysis")\
          .getOrCreate()

schema = StructType([
    StructField("text", StringType(), True),
    StructField("target", IntegerType(), True)])

project_folder = '/content/drive/MyDrive/CS777_BigDataAnalytics/term_project/'

dfTextTarget = spark.read.csv(project_folder + 'small_preprocessed_review', \
                              header=False, schema=schema)
dfTextTarget = dfTextTarget.dropna()
dfTextTarget.printSchema()
```

```
root
 |-- text: string (nullable = true)
 |-- target: integer (nullable = true)
```

In [5]:
```
dfTextTarget.show(5)
```

```
+--------------------+------+
|                text|target|
+--------------------+------+
|horrible experien...|     0|
|i went to the fre...|     0|
|my phone dies at ...|     0|
|another terrific ...|     1|
|called on monday ...|     1|
+--------------------+------+
only showing top 5 rows
```

In [6]:
```
dfTextTarget.count()
```

Out[6]:
```
69998
```

Data Split

In [8]:
```
(train_set, test_set) = dfTextTarget.randomSplit([0.8, 0.2], seed = 2000)
test_set.count()
```

Out[8]:
```
13940
```

In [9]:
```
train_set.count()
```

Out[9]:
```
56058
```

Hashing TF - IDF -Logistic Regression

In [10]:
```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer, CountVectorizer
```

```
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

In [11]:

```
%%time
def eval_model(model_name,model):

  tokenizer = Tokenizer(inputCol="text", outputCol="words")
  hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
  idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: ren
  label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
  pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx,model])
  pipelineFit = pipeline.fit(train_set)

  predictions_train = pipelineFit.transform(train_set)
  predictions_test = pipelineFit.transform(test_set)

  train_accuracy = predictions_train.filter(predictions_train.label == predicti
  test_accuracy = predictions_test.filter(predictions_test.label == predictions

  evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
  train_roc_auc = evaluator.evaluate(predictions_train)
  test_roc_auc = evaluator.evaluate(predictions_test)
  metricsList = [(model_name,train_accuracy,test_accuracy,train_roc_auc,test_rc
  return metricsList
```

```
CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 10 µs
```

In [12]:

```
schemaMetrics = StructType([\
  StructField('model', StringType(), True),\
  StructField('train_accuracy', FloatType(), True),\
  StructField('test_accuracy', FloatType(), True),\
  StructField('train_ROC_AUC', FloatType(), True),\
  StructField('test_ROC_AUC', FloatType(), True)])
metrics = spark.createDataFrame([], schemaMetrics)
```

In [13]:

```
%%time
lr = LogisticRegression(maxIter=100)
logreg_metricsList = eval_model('LogReg', lr)
# spark.createDataFrame(logreg_metricsList).write.csv(project_folder+'metrics')
logreg = spark.createDataFrame(logreg_metricsList, schemaMetrics)
metrics = metrics.union(logreg)
metrics.show()
```

```
+------+--------------+-------------+-------------+------------+
| model|train_accuracy|test_accuracy|train_ROC_AUC|test_ROC_AUC|
+------+--------------+-------------+-------------+------------+
|LogReg|     0.9999643|    0.8167862|    0.9999995|   0.8584581|
+------+--------------+-------------+-------------+------------+

CPU times: user 900 ms, sys: 93.4 ms, total: 993 ms
Wall time: 2min 3s
```

TFIDF + LInear SVC

In [14]:

```
%%time
```

```
from pyspark.ml.classification import LinearSVC
lsvc = LinearSVC(maxIter=10, regParam=0.1)
lsvc_metricsList = eval_model('LinearSVC', lsvc)
lsvc_metricsDF = spark.createDataFrame(lsvc_metricsList, schemaMetrics)
metrics = metrics.union(lsvc_metricsDF)
metrics.show()
```

```
+---------+--------------+-------------+-------------+------------+
|    model|train_accuracy|test_accuracy|train_ROC_AUC|test_ROC_AUC|
+---------+--------------+-------------+-------------+------------+
|   LogReg|     0.9999643|    0.8167862|    0.9999995|   0.8584581|
|LinearSVC|     0.9407043|   0.90164995|   0.97813654|  0.94934887|
+---------+--------------+-------------+-------------+------------+

CPU times: user 525 ms, sys: 62 ms, total: 587 ms
Wall time: 1min 7s
```

TFIDF + Decision Tree

In [15]:
```
%%time
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt_metricsList = eval_model('DecisionTree', dt)
dt_metricsDF = spark.createDataFrame(dt_metricsList, schemaMetrics)
metrics = metrics.union(dt_metricsDF)
metrics.show()
```

```
+------------+--------------+-------------+-------------+------------+
|       model|train_accuracy|test_accuracy|train_ROC_AUC|test_ROC_AUC|
+------------+--------------+-------------+-------------+------------+
|      LogReg|     0.9999643|    0.8167862|    0.9999995|   0.8584581|
|   LinearSVC|     0.9407043|   0.90164995|   0.97813654|  0.94934887|
|DecisionTree|    0.76986337|   0.77245337|    0.6773023|   0.6837623|
+------------+--------------+-------------+-------------+------------+

CPU times: user 2.17 s, sys: 263 ms, total: 2.43 s
Wall time: 6min 39s
```

CountVectorizer + IDF + Logistic Regression

There's another way that you can get term frequecy for IDF (Inverse Document Freqeuncy) calculation. It is CountVectorizer in SparkML. Apart from the reversibility of the features (vocabularies), there is an important difference in how each of them filters top features. In case of HashingTF it is dimensionality reduction with possible collisions. CountVectorizer discards infrequent tokens.

Let's see if performance changes if we use Countvectorizer instead of HashingTF.

In [16]:
```
%%time
def count_vectorizer_model(model_name,model):

  tokenizer = Tokenizer(inputCol="text", outputCol="words")
  cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
  # hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
  idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5) #minDocFreq: rem
```

```
    label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
    pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx,model])
    pipelineFit = pipeline.fit(train_set)

    predictions_train = pipelineFit.transform(train_set)
    predictions_test = pipelineFit.transform(test_set)

    train_accuracy = predictions_train.filter(predictions_train.label == predicti
    test_accuracy = predictions_test.filter(predictions_test.label == predictions

    evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
    train_roc_auc = evaluator.evaluate(predictions_train)
    test_roc_auc = evaluator.evaluate(predictions_test)
    metricsList = [(model_name,train_accuracy,test_accuracy,train_roc_auc,test_rc

    return metricsList
```

```
CPU times: user 0 ns, sys: 8 µs, total: 8 µs
Wall time: 13.6 µs
```

In [17]:

```
%%time
metricsList = count_vectorizer_model('CVIDF_LogReg', lr)
metricsDF = spark.createDataFrame(metricsList, schemaMetrics)
metrics = metrics.union(metricsDF)
metrics.show()
```

```
+------------+--------------+-------------+-------------+------------+
|       model|train_accuracy|test_accuracy|train_ROC_AUC|test_ROC_AUC|
+------------+--------------+-------------+-------------+------------+
|      LogReg|     0.9999643|    0.8167862|    0.9999995|   0.8584581|
|   LinearSVC|     0.9407043|   0.90164995|   0.97813654|  0.94934887|
|DecisionTree|    0.76986337|   0.77245337|    0.6773023|   0.6837623|
|CVIDF_LogReg|     0.9999643|    0.8235294|    0.9999992|   0.8653524|
+------------+--------------+-------------+-------------+------------+

CPU times: user 341 ms, sys: 48.9 ms, total: 390 ms
Wall time: 1min 14s
```

N-gram Implementation with Chi Squared Selector

Spark does not automatically combine features from different n-grams, so I had to use VectorAssembler in the pipeline, to combine the features I get from each n-grams.

I first tried to extract around 16,000 features from unigram, bigram, trigram. This means I will get around 48,000 features in total. Then I implemented Chi Squared feature selection to reduce the features back to 16,000 in total.

In [18]:

```
from pyspark.ml.feature import NGram, VectorAssembler
from pyspark.ml.feature import ChiSqSelector

def build_trigrams(inputCol=["text","target"], n=3):
    tokenizer = [Tokenizer(inputCol="text", outputCol="words")]
    ngrams = [
        NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
        for i in range(1, n + 1)
    ]
```

```
    cv = [
        CountVectorizer(vocabSize=2**14,inputCol="{0}_grams".format(i),
            outputCol="{0}_tf".format(i))
        for i in range(1, n + 1)
    ]
    idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), mi

    assembler = [VectorAssembler(
        inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
        outputCol="rawFeatures"
    )]
    label_stringIdx = [StringIndexer(inputCol = "target", outputCol = "label")]
    selector = [ChiSqSelector(numTopFeatures=2**14,featuresCol='rawFeatures', c
    lr = [LogisticRegression(maxIter=100)]
    return Pipeline(stages=tokenizer + ngrams + cv + idf+ assembler + label_str
```

In [19]:

```
%%time
trigram_pipelineFit = build_trigrams().fit(train_set)
# predictions = trigram_pipelineFit.transform(val_set)
# accuracy = predictions.filter(predictions.label == predictions.prediction).co
# roc_auc = evaluator.evaluate(predictions)

# # print accuracy, roc_auc
# print("Accuracy Score: {0:.4f}".format(accuracy))
# print("ROC-AUC: {0:.4f}".format(roc_auc))
```

```
CPU times: user 2.37 s, sys: 296 ms, total: 2.66 s
Wall time: 6min 15s
```

In [20]:

```
%%time
predictions_train = trigram_pipelineFit.transform(train_set)
predictions_test = trigram_pipelineFit.transform(test_set)

train_accuracy = predictions_train.filter(predictions_train.label == predictior
test_accuracy = predictions_test.filter(predictions_test.label == predictions_t

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
train_roc_auc = evaluator.evaluate(predictions_train)
test_roc_auc = evaluator.evaluate(predictions_test)
metricsList = [('Ngrams_ChiSqSelector',train_accuracy,test_accuracy,train_roc_a
```

```
CPU times: user 845 ms, sys: 137 ms, total: 982 ms
Wall time: 1min 39s
```

In [21]:

```
metricsDF = spark.createDataFrame(metricsList, schemaMetrics)
metrics = metrics.union(metricsDF)
metrics.show()
```

```
+--------------------+--------------+-------------+-------------+------------+
|               model|train_accuracy|test_accuracy|train_ROC_AUC|test_ROC_AUC|
+--------------------+--------------+-------------+-------------+------------+
|              LogReg|     0.9999643|    0.8167862|    0.9999995|   0.8584581|
|           LinearSVC|     0.9407043|   0.90164995|   0.97813654|  0.94934887|
|        DecisionTree|    0.76986337|   0.77245337|    0.6773023|   0.6837623|
|         CVIDF_LogReg|     0.9999643|    0.8235294|    0.9999992|   0.8653524|
|Ngrams_ChiSqSelector|     0.9999465|    0.8776901|   0.99999934|  0.92885864|
+--------------------+--------------+-------------+-------------+------------+
```

N-gram Implementation without Chi Squared Selector

In [22]:

```
from pyspark.ml.feature import NGram, VectorAssembler

def build_ngrams_wocs(inputCol=["text","target"], n=3):
    tokenizer = [Tokenizer(inputCol="text", outputCol="words")]
    ngrams = [
        NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
        for i in range(1, n + 1)
    ]

    cv = [
        CountVectorizer(vocabSize=5460,inputCol="{0}_grams".format(i),
            outputCol="{0}_tf".format(i))
        for i in range(1, n + 1)
    ]
    idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), mi

    assembler = [VectorAssembler(
        inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
        outputCol="features"
    )]
    label_stringIdx = [StringIndexer(inputCol = "target", outputCol = "label")]
    lr = [LogisticRegression(maxIter=100)]
    return Pipeline(stages=tokenizer + ngrams + cv + idf+ assembler + label_str
```

In [23]:

```
%%time

trigramwocs_pipelineFit = build_ngrams_wocs().fit(train_set)
predictions_train = trigramwocs_pipelineFit.transform(train_set)
predictions_test = trigramwocs_pipelineFit.transform(test_set)

train_accuracy = predictions_train.filter(predictions_train.label == predictior
test_accuracy = predictions_test.filter(predictions_test.label == predictions_t

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
train_roc_auc = evaluator.evaluate(predictions_train)
test_roc_auc = evaluator.evaluate(predictions_test)
metricsList = [('Ngrams_WithOut_ChiSq',train_accuracy,test_accuracy,train_roc_a
```

```
CPU times: user 2.09 s, sys: 276 ms, total: 2.36 s
Wall time: 4min 55s
```

In [24]:

```
metricsDF = spark.createDataFrame(metricsList, schemaMetrics)
metrics = metrics.union(metricsDF)
metrics.show()
```

model	train_accuracy	test_accuracy	train_ROC_AUC	test_ROC_AUC
LogReg	0.9999643	0.8167862	0.9999995	0.8584581
LinearSVC	0.9407043	0.90164995	0.97813654	0.94934887
DecisionTree	0.76986337	0.77245337	0.6773023	0.6837623
CVIDF_LogReg	0.9999643	0.8235294	0.9999992	0.8653524
Ngrams_ChiSqSelector	0.9999465	0.8776901	0.99999934	0.92885864
Ngrams_WithOut_ChiSq	0.9999822	0.87439024	0.99999946	0.9283303

In [39]:
```
metrics.toPandas().to_csv(project_folder+'metrics.csv')
```

The end.