Assignment No. 4
Sentiment Analysis on Twitter Dataset using PySpark

1. Introduction

This report presents sentiment analysis performed on a Twitter dataset using PySpark. The dataset includes columns such as text and label, and the goal is to classify tweets as either positive (1) or negative (0) based on their content. Unnecessary columns were removed to focus on relevant features.

2. Dataset Description

The dataset used for this analysis is the kazanova/sentiment140.csv dataset, which contains tweets along with sentiment labels. The following columns were considered:

- **label**: Binary sentiment labels (0 = Negative, 1 = Positive)
- **text**: The actual tweet content

The dataset was preprocessed to clean the text, remove stopwords, and convert it into a format suitable for machine learning.

3. Methodology

The sentiment analysis followed the below PySpark workflow:

1. Data Preprocessing:

- Converted text to lowercase.
- Removed special characters and punctuation.
- Tokenized the text.
- Removed stopwords.

2. Feature Engineering:

- Used TF-IDF (Term Frequency - Inverse Document Frequency) to convert text into numerical feature vectors.

3. Model Training:

- Logistic Regression was used for classification.
- The dataset was split into an 80%-20% training-test split.

4. Evaluation Metrics:

- Accuracy was measured using PySpark's MulticlassClassificationEvaluator.
- Precision, Recall, and F1 Score were also computed for further assessment.

4. Results

The model achieved the following performance metrics:

- **Accuracy**: 59.51%

- **Precision**: 59.67%
- **Recall**: 59.51%
- **F1 Score**: 59.35%

Below are some sample predictions:

text	label (Actual)	prediction
great pic im j…	0	1.0
nah do it under…	0	0.0
all the old show…	0	0.0
damn im doing so…	0	0.0
haha well i wish…	0	0.0
i cant dude i ha…	0	0.0
im right here	0	0.0
mmm that looks g…	1	1.0
thats the worst …	1	1.0
the studio for t…	1	0.0

5. Interpretation of Results

The model achieved an accuracy of 59.51%, indicating moderate performance. Some reasons for misclassification could be:

- **Contextual meaning**: Sarcasm and slang are difficult to detect.
- **Imbalanced dataset**: If there are significantly more negative or positive tweets, the model may be biased.
- **Data preprocessing limitations**: Some meaningful words might have been removed as stopwords.

7. Conclusion

This project successfully demonstrated sentiment analysis on Twitter data using PySpark. The Logistic Regression model achieved an accuracy of 59.51%, which provides a baseline for further improvements. Future enhancements can involve advanced NLP techniques and fine-tuning to achieve better classification accuracy.

sentiment-analysis

March 18, 2025

[1]:
```
from pyspark.sql import SparkSession


spark = SparkSession.builder.appName("sentiment").getOrCreate()
```

[3]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("kazanova/sentiment140")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version (installed:
0.3.7), please consider upgrading to the latest version (0.3.10).
Path to dataset files:
C:\Users\Harshal\.cache\kagglehub\datasets\kazanova\sentiment140\versions\2
```

[4]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf, regexp_replace, lower
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline

# Initialize Spark Session
spark = SparkSession.builder.appName("SentimentAnalysis").getOrCreate()
```

[6]:
```
df = spark.read.csv("C:/Users/Harshal/.cache/kagglehub/datasets/kazanova/
 ↪sentiment140/versions/2/training.1600000.processed.noemoticon.csv",␣
 ↪header=False, inferSchema=True)
```

[7]:
```
df = df.selectExpr("_c0 as label", "_c5 as text")

# Convert Labels (4 -> 1 for positive sentiment)
df = df.withColumn("label", (col("label") / 4).cast(IntegerType()))

# Sample 2000 records
df_sample = df.sample(fraction=2000/1600000, seed=42)
```

```
# Text Cleaning: Remove URLs, mentions, hashtags, and special characters
df_sample = df_sample.withColumn("text", lower(col("text")))
df_sample = df_sample.withColumn("text", regexp_replace(col("text"),␣
 ↪r"http\S+|www\S+", ""))
df_sample = df_sample.withColumn("text", regexp_replace(col("text"), r"@\w+",␣
 ↪""))
df_sample = df_sample.withColumn("text", regexp_replace(col("text"), r"#\w+",␣
 ↪""))
df_sample = df_sample.withColumn("text", regexp_replace(col("text"),␣
 ↪r"[^a-zA-Z\s]", ""))

# Feature Engineering: Tokenization, Stopwords Removal, TF-IDF
tokenizer = Tokenizer(inputCol="text", outputCol="words")
stopwords_remover = StopWordsRemover(inputCol="words",␣
 ↪outputCol="filtered_words")
hashing_tf = HashingTF(inputCol="filtered_words", outputCol="raw_features",␣
 ↪numFeatures=10000)
idf = IDF(inputCol="raw_features", outputCol="features")
```

[8]:
```
# Train Model: Logistic Regression
lr = LogisticRegression(featuresCol="features", labelCol="label")

# Pipeline
pipeline = Pipeline(stages=[tokenizer, stopwords_remover, hashing_tf, idf, lr])

# Train Model
train_data, test_data = df_sample.randomSplit([0.8, 0.2], seed=42)
model = pipeline.fit(train_data)
```

[9]:
```
predictions = model.transform(test_data)
predictions.select("text", "label", "prediction").show(10)
```

```
+--------------------+-----+----------+
|                text|label|prediction|
+--------------------+-----+----------+
|  great pic  im j…|    0|       1.0|
|  nah do it under…|    0|       1.0|
| all the old show…|    0|       1.0|
| damn im doing so…|    0|       1.0|
| haha well i wish…|    0|       0.0|
| i cant dude i ha…|    0|       0.0|
|      im right here |    0|       1.0|
| mmm that looks g…|    0|       0.0|
| thats the worst …|    0|       1.0|
| the studio for t…|    0|       1.0|
+--------------------+-----+----------+
```

```
only showing top 10 rows
```

[10]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Accuracy Calculation
accuracy_evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="accuracy")
accuracy = accuracy_evaluator.evaluate(predictions)

precision_evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="weightedPrecision")
precision = precision_evaluator.evaluate(predictions)

recall_evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="weightedRecall")
recall = recall_evaluator.evaluate(predictions)

f1_evaluator = MulticlassClassificationEvaluator(labelCol="label",␣
 ↪predictionCol="prediction", metricName="f1")
f1_score = f1_evaluator.evaluate(predictions)

# Print Evaluation Metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1_score:.4f}")
```

```
Accuracy: 0.5951
Precision: 0.5967
Recall: 0.5951
F1 Score: 0.5935
```

[ ]:
```

```