Assignment No. 4

Title: Use Twitter data for sentiment analysis. The dataset is 3MB in size and has 31,962 tweets. Identify the tweets which are hate tweets and which are not

Theory:

1. **Introduction:** Sentiment analysis is a natural language processing (NLP) technique that determines whether a given text conveys a positive, negative, or neutral emotion. This project focuses on detecting hate speech in tweets — an important task for creating safer online environments. By leveraging machine learning and big data techniques, we can efficiently analyze large datasets of social media content.
2. **Key Components:**
    - **PySpark:** Enables distributed data processing, making it scalable for large datasets.
    - **Text Preprocessing:** Cleans raw text by removing noise (punctuation, numbers, etc.).
    - **Tokenizer:** Splits tweets into individual words for analysis.
    - **StopWordsRemover:** Filters out common but unimportant words (e.g., 'the', 'is').
    - **HashingTF and IDF:** Converts text data into numerical vectors, balancing frequent and rare words.
    - **Naive Bayes Classifier:** A probabilistic model effective for text classification.
3. **Input and Output:**
    - **Input:** A CSV file containing 31,962 tweets with their respective sentiment labels (e.g., positive, negative, neutral).
    - **Output:** A prediction for each tweet, classifying it as hate speech or not.
4. **Program Flow:**
    - **Load Data:** Reads the CSV file into a PySpark DataFrame.
    - **Preprocess Data:** Cleans tweets and prepares them for analysis.
    - **Feature Engineering:** Tokenizes, removes stop words, and applies TF-IDF.
    - **Train-Test Split:** Divides the dataset into training (80%) and testing (20%) data.
    - **Model Training:** Trains a Naive Bayes classifier on the training set.
    - **Predictions:** Generates predictions on the test data.
    - **Evaluation:** Measures accuracy to assess model performance.
5. **Explanation of Code:**
    - **Data Loading:** The dataset is read into a DataFrame with appropriate column names.
    - **Text Cleaning:** A user-defined function (UDF) removes punctuation, converts text to lowercase, and eliminates numbers.
    - **Label Encoding:** StringIndexer converts sentiment labels into numerical format for the classifier.
    - **Pipeline Setup:** A PySpark pipeline chains multiple stages — tokenization, stop word removal, feature extraction, and model training — into one streamlined workflow.

- **Model Training:** The Naive Bayes classifier learns patterns from the training data.
- **Predictions and Evaluation:** The model predicts sentiments on test data, and accuracy is calculated using MulticlassClassificationEvaluator.

Observations:

Key observations during analysis:

- **Data Cleaning is Critical:** Raw tweets often contain noise (punctuation, numbers, mixed cases). Preprocessing ensures that the model focuses on the content rather than irrelevant symbols.
- **Feature Engineering Impacts Performance:** Tokenization, stop word removal, and TF-IDF conversion help transform text into a numerical format suitable for machine learning.
- **Balanced Dataset Matters:** If hate tweets are underrepresented, the classifier may fail to learn effectively. Data exploration helps uncover such imbalances.
- **Naive Bayes for Text:** Naive Bayes works well for text classification due to its assumption of feature independence, which matches the bag-of-words model.
- **Accuracy Evaluation:** Measuring accuracy and inspecting predictions help validate the model's performance.

Design Decisions:

The project involves several key design choices to ensure performance, scalability, and ease of use:

- **Using PySpark for Scalability:** PySpark efficiently handles large datasets by distributing the data processing workload across multiple nodes.
- **Custom Text Cleaning Function:** A user-defined function ensures consistent preprocessing, converting text to lowercase, removing punctuation, and filtering numbers.
- **StringIndexer for Label Conversion:** Converts sentiment labels into numerical format for compatibility with machine learning algorithms.
- **Pipeline Setup:** The end-to-end pipeline automates data preparation, feature engineering, and model training, ensuring a consistent workflow.
- **Naive Bayes Classifier:** Selected for its effectiveness in text classification tasks, especially when combined with TF-IDF features.
- **Train-Test Split:** An 80-20 split ensures the model has sufficient data to learn patterns while leaving enough unseen data for evaluation.
- **Evaluation Metric:** Accuracy is used to measure performance, though precision, recall, and F1-score could provide additional insights in future iterations.

Spark, large-scale multivariate analysis can be performed efficiently, making it a valuable approach in modern data science applications.

Code:

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF, StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml import Pipeline
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType
import re
import string

# Initialize Spark session
spark = SparkSession.builder.appName("TwitterSentimentAnalysis").getOrCreate()

# Load dataset
df_train = spark.read.csv("/content/twitter_training.csv", header=False, inferSchema=True)
df_train = df_train.toDF("ID", "Topic", "Sentiment", "Tweet")

# Text cleaning function
def clean_text(text):
    if not isinstance(text, str):  # Ensure text is a string
        return ""
    text = text.lower()
    text = re.sub(f"[{string.punctuation}]", "", text)
    text = re.sub(r"\d+", "", text)
    return text

clean_text_udf = udf(clean_text, StringType())
df_train = df_train.withColumn("Cleaned_Tweet", clean_text_udf(col("Tweet")))

# Convert labels to numerical format
indexer = StringIndexer(inputCol="Sentiment", outputCol="label")

# Tokenization and TF-IDF feature extraction
tokenizer = Tokenizer(inputCol="Cleaned_Tweet", outputCol="words")
remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
hashingTF = HashingTF(inputCol="filtered_words", outputCol="rawFeatures",
numFeatures=10000)
idf = IDF(inputCol="rawFeatures", outputCol="features")

# Naive Bayes classifier
nb = NaiveBayes()
```

```
# Build pipeline
pipeline = Pipeline(stages=[indexer, tokenizer, remover, hashingTF, idf, nb])

# Split data
train_data, test_data = df_train.randomSplit([0.8, 0.2], seed=42)

# Train the model
model = pipeline.fit(train_data)

# Predictions
predictions = model.transform(test_data)

# Evaluation
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
metricName="accuracy")
accuracy = evaluator.evaluate(predictions)

print(f"Accuracy: {accuracy:.2f}")
predictions.select("Sentiment", "Cleaned_Tweet", "prediction").show(10)
```

Output:

```
Accuracy: 0.66
+----------+--------------------+----------+
| Sentiment|       Cleaned_Tweet|prediction|
+----------+--------------------+----------+
|  Negative|          amazon wtf|       0.0|
|  Negative|i am really disap...|       0.0|
|  Negative|im really disappo...|       0.0|
|   Neutral|admit it subs cra...|       2.0|
|  Negative| amazon probably ...|       0.0|
|  Negative|amazon probably s...|       0.0|
|Irrelevant|youve purchased  ...|       0.0|
|   Neutral|love speculative ...|       3.0|
|  Negative|amazon be having ...|       0.0|
|  Negative|amazon be having ...|       2.0|
+----------+--------------------+----------+
only showing top 10 rows
```

```
Label Encoding Mapping:
Negative -> 0
Positive -> 1
Neutral -> 2
Irrelevant -> 3
```

Conclusion:

Sentiment analysis, particularly hate speech detection, is an essential application of NLP. This project demonstrates how PySpark enables scalable analysis of large datasets. The pipeline approach streamlines data preparation, feature extraction, and classification.

The Naive Bayes classifier, combined with effective text preprocessing and TF-IDF features, produces an accurate sentiment prediction model. Further improvements, such as handling imbalanced datasets, using more advanced models like Logistic Regression or BERT, or incorporating context-aware embeddings, could enhance performance.

This approach has practical applications in content moderation, social media monitoring, and online community management.