## ☑ 1. Import Required Libraries

python

CopyEdit

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import IDF, StopWordsRemover, Tokenizer, HashingTF
from pyspark.sql.functions import when, col
from pyspark.ml.classification import LogisticRegression
```

These are key ML and SQL functions used for:

- **Text preprocessing:** Tokenizer, StopWordsRemover
- **Feature extraction:** HashingTF, IDF
- **Classification:** Logistic Regression
- **Data manipulation:** when, col

## ☑ 2. Create Spark Session

python

CopyEdit

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

Initializes a Spark application named **"Twitter"**.

## ☑ 3. Load and Clean Data

python

CopyEdit

```
df = spark.read.csv("/content/twitter.csv", inferSchema=True, header=True)
df = df.filter(col("tweet").isNotNull()).filter(col("label").isNotNull())
```

- Reads a CSV file with tweets and labels.
- Infers schema (detects data types).
- Filters out rows where **tweet** or **label** is null.

☑ Your DataFrame likely has these two columns:

- **tweet** → The text content of a tweet.
- **label** → Classification label, e.g., 0, 1, or -1.

## ☑ 4. Tokenization (Splitting into Words)

python

CopyEdit

```
tokenizer = Tokenizer(inputCol="tweet", outputCol="words")
tokenized_df = tokenizer.transform(df)
```

- Splits each tweet into individual words.
- Adds a new column "words" to hold the list of words.

### ✅ 5. Remove Stopwords (common words like "is", "the")

python

CopyEdit

```
filtered_words = StopWordsRemover(inputCol="words", outputCol="filtered_words")

filtered = filtered_words.transform(tokenized_df)
```

- Removes unimportant or **stopwords** from the list of words.
- Stores cleaned words in "filtered_words" column.

### ✅ 6. Calculate Term Frequency using HashingTF

python

CopyEdit

```
term_freq = HashingTF(inputCol="filtered_words", outputCol="term_freq", numFeatures=5000)

term_freqency = term_freq.transform(filtered)
```

- Converts filtered words into numeric **term frequency (TF)** vectors.
- Each tweet becomes a sparse feature vector of size 5000.
- "term_freq" column holds these vectors.

### ✅ 7. Calculate Inverse Document Frequency (IDF)

python

CopyEdit

```
idf = IDF(inputCol="term_freq", outputCol="idf")

idf_model = idf.fit(term_freqency)

idf = idf_model.transform(term_freqency)
```

- Applies **IDF** to weigh down common terms and highlight rare but important words.
- "idf" column now holds **TF-IDF vectors**, the final features.

### ✅ 8. Fix Labels

python

CopyEdit

```
idf = idf.withColumn("label", when(col("label") == -1, 3).otherwise(col("label")))
```

- Replaces -1 with 3 in the label column.
- Logistic Regression doesn't work well with negative labels.

### ✅ 9. Train-Test Split

python

CopyEdit

```
train, test = idf.randomSplit([0.8, 0.2], seed=42)
```

- Randomly splits data:
    - **80% for training**
    - **20% for testing**
- seed=42 ensures reproducibility.

---

### ✅ 10. Train Logistic Regression Model

python

CopyEdit

```
LR = LogisticRegression(featuresCol="idf", labelCol="label")

model = LR.fit(train)
```

- Trains a **Logistic Regression** model using:
    - idf as feature column (TF-IDF vectors).
    - label as the target class.

---

### ✅ 11. Make Predictions

python

CopyEdit

```
predictions = model.transform(test)

predictions.show()
```

- Applies the model to the test set.
- Shows the predicted labels along with actual values and feature vectors.

---

### ✅ OUTPUT EXPLAINED

The output of predictions.show() will be a table with columns like:

| tweet | label | words | filtered_words | term_freq | idf | rawPrediction | probability | prediction |
|---|---|---|---|---|---|---|---|---|

**Important Output Columns:**

- **label** → Actual label from dataset.
- **idf** → TF-IDF feature vector used for prediction.
- **rawPrediction** → Raw model scores.
- **probability** → Predicted probability for each class.
- **prediction** → Final predicted label (most likely class).

---

### ✅ Summary of Flow

1. **Read & Clean Data**
2. **Tokenize → Remove Stopwords → TF → IDF**
3. **Convert -1 label to 3**
4. **Split data**
5. **Train Logistic Regression model**
6. **Predict on test data**

Would you like to **evaluate the model** using accuracy or F1-score next? I can help you add that too.

4o

**You said:**

explain 6th and 7th steps

**ChatGPT said:**

Absolutely! Let's dive deeper into **Step 6** and **Step 7** — these are crucial steps where your text data gets converted into meaningful **numerical features** using **TF** and **IDF**.

### ◆ STEP 6: HashingTF — Term Frequency Vectorization

python

CopyEdit

```
term_freq = HashingTF(inputCol="filtered_words", outputCol="term_freq", numFeatures=5000)

term_freqency = term_freq.transform(filtered)
```

**◆ What's Happening?**

- You're converting the cleaned list of words (**filtered_words**) into **numerical feature vectors** using the **Hashing Term Frequency (HashingTF)** technique.
- This gives you a sparse vector that shows **how often each term appears** in a document (in this case, each tweet).

**◆ Why Hashing?**

- Instead of keeping a big dictionary of all words (which can be memory-heavy), it uses a **hash function** to map words to a fixed number of features (5000 in your case).
- This is efficient and scales well with large datasets.

**✅ Output:**

- A new column called **term_freq** is added to the DataFrame.
- Each row now contains a **sparse vector** of 5000 dimensions representing **term frequencies** of the tweet.

**🔍 Example:**

Suppose the cleaned words are: ["spark", "is", "awesome", "spark"]

Then HashingTF might give:

css

CopyEdit

```
(5000,[193,3321,4821],[2.0,1.0,1.0])
```

- 5000 → size of the vector
- [193, 3321, 4821] → indices where non-zero values exist (based on hash function)
- [2.0, 1.0, 1.0] → frequencies of those hashed words

---

◈ **STEP 7: IDF — Inverse Document Frequency**

python

CopyEdit

```
idf = IDF(inputCol="term_freq", outputCol="idf")

idf_model = idf.fit(term_freqency)

idf = idf_model.transform(term_freqency)
```

◆ **What's Happening?**

- Now you're applying **Inverse Document Frequency (IDF)** to the **term frequency vectors**.
- TF alone treats all terms equally, but IDF reduces the importance of **common words** and boosts the importance of **rare words**.

◆ **Why TF-IDF?**

- It’s a combination of:
  - **TF (Term Frequency)** — how often a word appears in a tweet.
  - **IDF (Inverse Document Frequency)** — how *rare* the word is across all tweets.
- This results in a better representation of meaningful words.

✅ **Output:**

- A new column **idf** is added to the DataFrame.
- This is your final **feature vector** for each tweet, used by the machine learning model.

---

🔍 **Example:**

If "the" appears in almost every tweet, its IDF will be low, and hence its weight in the vector will be reduced.

Whereas a unique word like "terrorism" (if it appears only in few tweets) gets a higher weight and is treated as more **informative**.