**Simple Guide to the Cyberbullying Detection Project**

This project detects cyberbullying in text data using a machine learning model built with PySpark. I used a Docker container to create a consistent environment, prepared datasets stored in HDFS, and wrote code to train and test the model. Below, I’ll walk you through how I managed the setup and arrived at the code.

**Step 1: Setting Up the Environment with Docker**

To make sure everything works the same way for everyone, I used **Docker** to create a container with all the tools I needed: Spark, Python, HDFS (for storing data), and YARN (for managing resources). Here’s how we did it:

**Dockerfile Creation**

I started with a basic Dockerfile to set up the environment:

dockerfile

# Starting with an official Spark image

# Installing Python and tools

# Telling Spark to use Python 3

# Installing PySpark

# Opening ports for Spark UI, HDFS, and YARN

# Starting Spark master

This gave me a running Spark cluster with YARN and HDFS.

**Step 2: Preparing the Data**

I used several CSV datasets containing text and labels (0 for no cyberbullying, 1 for cyberbullying). These were stored in HDFS at paths like:

* hdfs://localhost:9000/input/youtube\_parsed\_dataset.csv
* hdfs://localhost:9000/input/twitter\_sexism\_parsed\_dataset.csv

Each dataset had a Text column (the message) and an oh\_label column (0 or 1). I uploaded these files to HDFS before running the code.

**Step 3: Writing and Running the Code**

I wrote the code in a Jupyter notebook connected to our Spark cluster. Here’s how I built it up to what you see now:

**Starting the Spark Session**

I began by connecting to the Spark cluster:

python

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("CyberbullyingDetection").master("yarn").getOrCreate()

This links omy code to the YARN-managed Spark cluster in the Docker container.

**Loading and Combining Datasets**

I loaded all the datasets from HDFS, cleaned them up, and combined them:

python

from functools import reduce

from pyspark.sql import DataFrame

dataset\_paths = [

"hdfs://localhost:9000/input/youtube\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/twitter\_sexism\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/twitter\_racism\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/twitter\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/toxicity\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/kaggle\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/attack\_parsed\_dataset.csv",

"hdfs://localhost:9000/input/aggression\_parsed\_dataset.csv"

]

df\_list = []

for path in dataset\_paths:

df\_temp = spark.read.csv(path, header=True, inferSchema=True)

df\_temp = df\_temp.select("Text", "oh\_label")

df\_temp = df\_temp.filter(df\_temp.oh\_label.isNotNull() & df\_temp.oh\_label.isin(0, 1))

df\_temp = df\_temp.withColumn("label", df\_temp.oh\_label.cast("double"))

df\_temp = df\_temp.select("Text", "label")

df\_list.append(df\_temp)

combined\_df = reduce(DataFrame.union, df\_list)

* **What this does**: Loads each CSV, keeps only Text and oh\_label, removes nulls, ensures labels are 0 or 1, and merges everything into one big dataset.

**Feature Extraction and Model Setup**

I turned the text into numbers the model can use and set up a logistic regression model:

python

from pyspark.ml.feature import HashingTF, IDF, Tokenizer

from pyspark.ml.classification import LogisticRegression

from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="Text", outputCol="words")

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures")

idf = IDF(inputCol="rawFeatures", outputCol="features")

lr = LogisticRegression(maxIter=10, regParam=0.01)

pipeline = Pipeline(stages=[tokenizer, hashingTF, idf, lr])

* **Tokenizer**: Breaks text into words.
* **HashingTF**: Turns words into a number vector.
* **IDF**: Adjusts the vector based on word rarity.
* **LogisticRegression**: The model that learns to predict 0 or 1.
* **Pipeline**: Combines all steps into one process.

**Training and Testing the Model**

I split the data and trained the model:

python

(train\_data, test\_data) = combined\_df.randomSplit([0.8, 0.2], seed=42)

model = pipeline.fit(train\_data)

* **80% training, 20% testing**: Splits the data to train and check the model.

**Evaluating the Model**

I checked how well the model works:

python

from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

predictions = model.transform(test\_data)

evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

auc = evaluator.evaluate(predictions)

print(f"Area Under ROC: {auc}")

precision\_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")

precision = precision\_evaluator.evaluate(predictions)

print(f"Precision: {precision}")

recall\_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedRecall")

recall = recall\_evaluator.evaluate(predictions)

print(f"Recall: {recall}")

f1\_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")

f1 = f1\_evaluator.evaluate(predictions)

print(f"F1-Score: {f1}")

predictions.select("Text", "label", "prediction").show(10, truncate=False)

* **Metrics**: AUC (overall performance), precision (accuracy of positive predictions), recall (catching all positives), and F1-score (balance of precision and recall).
* **Show predictions**: Displays some test results.

**Manual Testing**

I tested the model with example texts:

python

manual\_data = [

("You’re such a loser, go away!", 1.0),

("I hate you, you’re worthless.", 1.0),

("This is a beautiful day!", 0.0),

("Great job, keep it up!", 0.0),

("You’re so dumb, why are you here?", 1.0),

("Thanks for sharing, I appreciate it.", 0.0),

("Nobody cares about you, idiot.", 1.0),

("Let’s agree to disagree, no worries.", 0.0)

]

manual\_df = spark.createDataFrame(manual\_data, ["Text", "label"])

manual\_predictions = model.transform(manual\_df)

manual\_predictions.select("Text", "label", "prediction").show(truncate=False)

* **What this does**: Tests the model on hand-picked examples to see if it predicts correctly.

**Real-Time Detection**

I added a function to check new text:

python

def detect\_cyberbullying(text):

single\_df = spark.createDataFrame([(text,)], ["Text"])

prediction = model.transform(single\_df).select("prediction").collect()[0][0]

if prediction == 1.0:

return "Block or Blur: Inappropriate content detected."

else:

return "Allow: Content is safe."

print(detect\_cyberbullying("You’re an idiot!"))

print(detect\_cyberbullying("Nice work, well done!"))

* **Purpose**: Simulates real-time use, like in a chat app.

**Stopping the Session**

Finally, I closed Spark:

python

spark.stop()

**What I Achieved**

With this setup and code, I created a system that:

* Loads multiple datasets from HDFS.
* Trains a model to detect cyberbullying.
* Tests it on real data and manual examples.
* Can classify new text in real-time.