**Deployment of News Article Categorization Application Using AWS Services**

**Objective:**  
The objective of this project is to deploy a fine-tuned text classification model using AWS services to categorize news articles into predefined categories such as Sports, Politics, Technology, and Entertainment. The deployment involves leveraging AWS resources, including S3 for storing the model, EC2 for hosting the web application, and RDS for logging user interactions. The goal is to build a scalable, secure, and accessible web-based solution using Streamlit or Gradio, enabling users to classify news articles efficiently while integrating MLOps best practices.

**>>>import pandas as pd**

**Explanation**: Imports the pandas library, a powerful data manipulation and analysis tool in Python. It is used here to load and process CSV datasets. The alias pd is a common convention for brevity.

**>>>from sklearn.feature\_extraction.text import TfidfVectorizer**

**Explanation**: Imports the TfidfVectorizer class from the scikit-learn library. This class converts text data into a numerical format (TF-IDF scores) that can be used by machine learning models. TF-IDF stands for Term Frequency-Inverse Document Frequency, a method to evaluate the importance of words in documents.

**>>>from sklearn.linear\_model import LogisticRegression**

**Explanation**: Imports the LogisticRegression class from scikit-learn. This is a machine learning algorithm used for classification tasks. Here, it will be trained to predict class labels based on the TF-IDF features.

**>>>import joblib**

**Explanation**: Imports the joblib library, which provides tools for saving and loading Python objects (like trained models and vectorizers) to/from disk efficiently. This is useful for persisting the trained model and vectorizer for later use.

**>>>train\_data = pd.read\_csv("C:/Users/laksh/Downloads/archive/train.csv")**

**>>>test\_data = pd.read\_csv("C:/Users/laksh/Downloads/archive/test.csv")**

**Explanation**: Loads the training and testing datasets from CSV files located at the specified file paths using pandas.read\_csv().

* train\_data: The training dataset, which will be used to train the model.
* test\_data: The testing dataset, which will be used to evaluate the model (though evaluation isn’t shown in this code snippet).
* File paths are hardcoded and specific to the user’s local environment (C:/Users/laksh/Downloads/archive/).

**>>>print("Train Data Columns:", train\_data.columns)**

**>>>print("Test Data Columns:", test\_data.columns)**

**Explanation**: Prints the column names of the train\_data and test\_data DataFrames to the console.

* train\_data.columns and test\_data.columns return the list of column names in each dataset.
* This step is useful for verifying the structure of the datasets (e.g., ensuring expected columns like Title, Description, and Class Index exist).

***# Combining the title and description columns into a single column named text***

**>>>train\_data['text'] = train\_data['Title'] + " " + train\_data['Description']**

**>>>test\_data['text'] = test\_data['Title'] + " " + test\_data['Description']**

* **Explanation**: Creates a new column text in both train\_data and test\_data by concatenating the Title and Description columns with a space (" ") between them.
  + This combines the two text fields into a single feature for text analysis, assuming both fields are relevant for classification.
  + The result is a single column (text) that contains the full text content for each row.

**# Defining features (X) and labels (y)**

**>>>X\_train = train\_data['text']**

**>>>y\_train = train\_data['Class Index']**

**>>>X\_test = test\_data['text']**

**>>>y\_test = test\_data['Class Index']**

**Explanation**: Defines the input features (X) and target labels (y) for training and testing.

* X\_train: The text data from the text column of train\_data, used as input features for training.
* y\_train: The target labels from the Class Index column of train\_data. The comment indicates this dataset uses Class Index instead of a column named Label.
* X\_test: The text data from the text column of test\_data, used as input features for testing.
* y\_test: The target labels from the Class Index column of test\_data.

**# Initialize and fit TF-IDF vectorizer**

**>>>vectorizer = TfidfVectorizer(max\_features=5000, stop\_words='english', ngram\_range=(1, 2))**

**Explanation**: Initializes a TfidfVectorizer object with specific parameters:

* max\_features=5000: Limits the vocabulary to the top 5,000 most frequent terms, reducing dimensionality and computationalcost.
* stop\_words='english': Removes common English words (e.g., "the", "is") that are typically uninformative for classification.
* ngram\_range=(1, 2): Includes both unigrams (single words) and bigrams (two-word phrases) in the vocabulary, capturing some contextual information.

**>>>X\_train\_tfidf = vectorizer.fit\_transform(X\_train)**

**Explanation**: Fits the TfidfVectorizer to the training data (X\_train) and transforms it into a TF-IDF matrix.

* fit\_transform() learns the vocabulary and IDF values from X\_train and converts the text into a sparse matrix of TF-IDF scores.
* X\_train\_tfidf: The resulting matrix where rows are documents (samples) and columns are features (terms), with numerical values representing TF-IDF scores.

**>>>X\_test\_tfidf = vectorizer.transform(X\_test)**

**Explanation**: Transforms the test data (X\_test) into a TF-IDF matrix using the already-fitted vectorizer.

* transform() applies the same vocabulary and IDF values learned from the training data, ensuring consistency between training and testing features.
* X\_test\_tfidf: The resulting TF-IDF matrix for the test data, aligned with the training data’s feature space.

**>>>classifier = LogisticRegression(max\_iter=1000, random\_state=42)**  
  
**Explanation**: Initializes a LogisticRegression classifier with specific parameters:

* max\_iter=1000: Sets the maximum number of iterations for the solver to converge (increased from default to handle larger datasets).
* random\_state=42: Fixes the random seed for reproducibility of results.

**>>>classifier.fit(X\_train\_tfidf, y\_train)**

**Explanation**: Trains the Logistic Regression model using the TF-IDF-transformed training data (X\_train\_tfidf) and corresponding labels (y\_train).

* The model learns coefficients for each feature to predict the Class Index.

**>>>joblib.dump(vectorizer, 'tfidf\_vectorizer.joblib')**

**>>>joblib.dump(classifier, 'logistic\_regression\_model.joblib')**

**Explanation**: Saves the trained vectorizer and classifier to disk using joblib.dump().

* 'tfidf\_vectorizer.joblib': File where the fitted TF-IDF vectorizer is saved.
* 'logistic\_regression\_model.joblib': File where the trained Logistic Regression model is saved.
* These files can be loaded later (e.g., with joblib.load()) to make predictions without retraining.

**>>>print("Model and vectorizer saved successfully!")**

**Explanation**: Prints a confirmation message to the console indicating that the vectorizer and model were successfully saved to disk.

**NEXT STEP IN AWS**

***1. Create S3 Bucket***

* Login to AWS Console:
  + Go to the [S3 Dashboard](https://console.aws.amazon.com/s3).
* Create a New Bucket:
  + Click on "Create bucket".
  + Provide a unique name for your bucket (e.g., news-article-classifier-model).
  + Choose a region close to your EC2 instance (e.g., US East (N. Virginia)).
  + Leave the rest of the settings as default unless you have specific requirements (like versioning or encryption).
  + Click "Create bucket".
* Upload Files:
  + After creating the bucket, click on the bucket name.
  + Click "Upload", and add your model files (tfidf\_vectorizer.joblib,logistic\_regression\_model.joblib) and the app.py file.

***2. Create IAM User***

* Login to AWS Console:
  + Go to the [IAM Dashboard](https://console.aws.amazon.com/iam).
* Create a New User:
  + In the left panel, click on "Users" and then "Add user".
  + Enter a username (e.g., news-article-user).
  + Select "Programmatic access" (to generate access keys).
  + Click "Next: Permissions".
* Set Permissions:
  + You can either choose "Attach policies directly" or use a predefined policy like AmazonS3FullAccess and AmazonEC2FullAccess for simplicity.
  + Click "Next: Tags" (optional), then "Review", and "Create user".
* Download Credentials:
  + Save the Access Key ID and Secret Access Key for future use.

***3. Create EC2 Instance***

* Login to AWS Console:
  + Navigate to the [EC2 Dashboard](https://console.aws.amazon.com/ec2).
* Launch Instance:
  + Click on "Launch Instance".
  + Choose an Amazon Machine Image (AMI) – select Amazon Linux 2 or Ubuntu (or any other OS of your choice).
  + Select an instance type (e.g., t2.micro for the Free Tier).
  + Configure instance details:
    - Network: Default VPC.
    - Subnet: Default.
    - Enable auto-assign Public IP: Yes.
  + Add storage: Leave the default storage (8 GB).
  + Add tags: (optional, e.g., Name: "news-article-classifier").
  + Configure security group:
    - Add a rule to allow inbound HTTP (port 80) and Streamlit (port 8501) traffic.
    - Ensure SSH access is allowed (port 22) from your IP for instance management.
  + Review and launch the instance.
  + Create a new key pair or select an existing one, and download the .pem file.
* Connect to EC2 Instance:

In the EC2 console, click on your instance**.**

>Click on connect to EC2 instances. **RUN THE FOLLOWING IN EC2 TERMINAL**

1. AWS CLI Setup & File Downloads

* Command: **aws --version**  
  Explanation: Verifies that AWS CLI is correctly installed on your EC2 instance.
* Command: **aws configure**  
  Explanation: Configures the AWS CLI with your credentials (Access Key ID, Secret Access Key), default region, and output format. This step ensures that AWS CLI can access your AWS resources.
* Command**: aws s3 cp s3://your-s3-bucket-name/tfidf\_vectorizer.joblib .**  
  Explanation: Downloads the TF-IDF vectorizer file from your S3 bucket to your EC2 instance.
* Command: **aws s3 cp s3://your-s3-bucket-name/logistic\_regression\_model.joblib .**Explanation: Downloads the Logistic Regression model from your S3 bucket to your EC2 instance.

2. Python Package Installations

* Command: **pip3 install joblib**  
  Explanation: Installs the joblib package to load serialized models (such as your Logistic Regression model and TF-IDF vectorizer).
* Command: **pip3 install scikit-learn**  
  Explanation: Installs the scikit-learn package, which is required for the Logistic Regression model that you have trained.

**Model Loading and Testing**

Command:python3

**>>>** **import joblib**

**>>>tfidf\_vectorizer = joblib.load('tfidf\_vectorizer.joblib')**

**>>>logistic\_regression\_model = joblib.load('logistic\_regression\_model.joblib')**

**Explanation:** Loads the TF-IDF vectorizer and Logistic Regression model into memory for use in making predictions.

**Command:** **pip3 uninstall scikit-learn**  
**Explanation:** Uninstalls the current version of scikit-learn due to version conflicts between the saved model and the installed version.

**Command:** **pip3 install scikit-learn==1.6.0**  
**Explanation:** Installs the version of scikit-learn (1.6.0) that matches the version used to train your saved model.

**Model Testing**

IN python3

**>>> text\_input = "Reuters - Short-sellers, Wall Street's dwindling..."**

**>>tfidf\_input = tfidf\_vectorizer.transform([text\_input])**

**>>>prediction = logistic\_regression\_model.predict(tfidf\_input)**

Explanation: Transforms the input text into the format expected by the model (TF-IDF) and makes a prediction using the Logistic Regression model.

**Code Update:**

**>>> category\_mapping = {1: 'World', 2: 'Sports', 3: 'Business', 4: 'Sci/Tech'}**

**>>>predicted\_category\_name = category\_mapping[prediction[0]]**

**>>>print("Predicted category:", predicted\_category\_name)**  
Explanation: Adds a category mapping for human-readable category names, so the model's numeric prediction is converted to a string (e.g., 3 becomes "Business").

**MySQL Installation Issues  
  
IN EC2 TERMINAL  
sudo yum install -y https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm**

**sudo yum install mysql-community-client**

**Explanation:** Attempts to install MySQL client on the EC2 instance. This is necessary for connecting to an RDS MySQL database for logging user activities.

**Command:** sudo yum clean all and sudo yum makecache  
**Explanation:** Clears and regenerates the yum cache to ensure that the MySQL client installation can be completed without issues.

**Streamlit Application Setup**

**Command:** ***nano app.py***  
**Explanation:** Opens the file editor to create your Streamlit application (app.py), which will allow users to interact with the model and predict news categories.

**Explanation of Code in app.py:**

* **Login & Database Integration:** Manages user login and stores login data in MySQL.
* **Model Prediction:** Allows users to input news text and receive predicted categories.
* **Admin Dashboard (for admins):** Displays login history from the MySQL database.

**Commands to Install Dependencies:**pip3 install streamlit

pip3 install joblib

pip3 install pandas

pip3 install scikit-learn

**Explanation:** Installs the necessary Python libraries for the Streamlit app (Streamlit, joblib, pandas, scikit-learn).  
  
**Running the Streamlit Application  
  
Command:** **streamlit run app.py**  
**Explanation:** Launches the Streamlit application, which serves the model and provides a user interface for prediction and viewing login history.

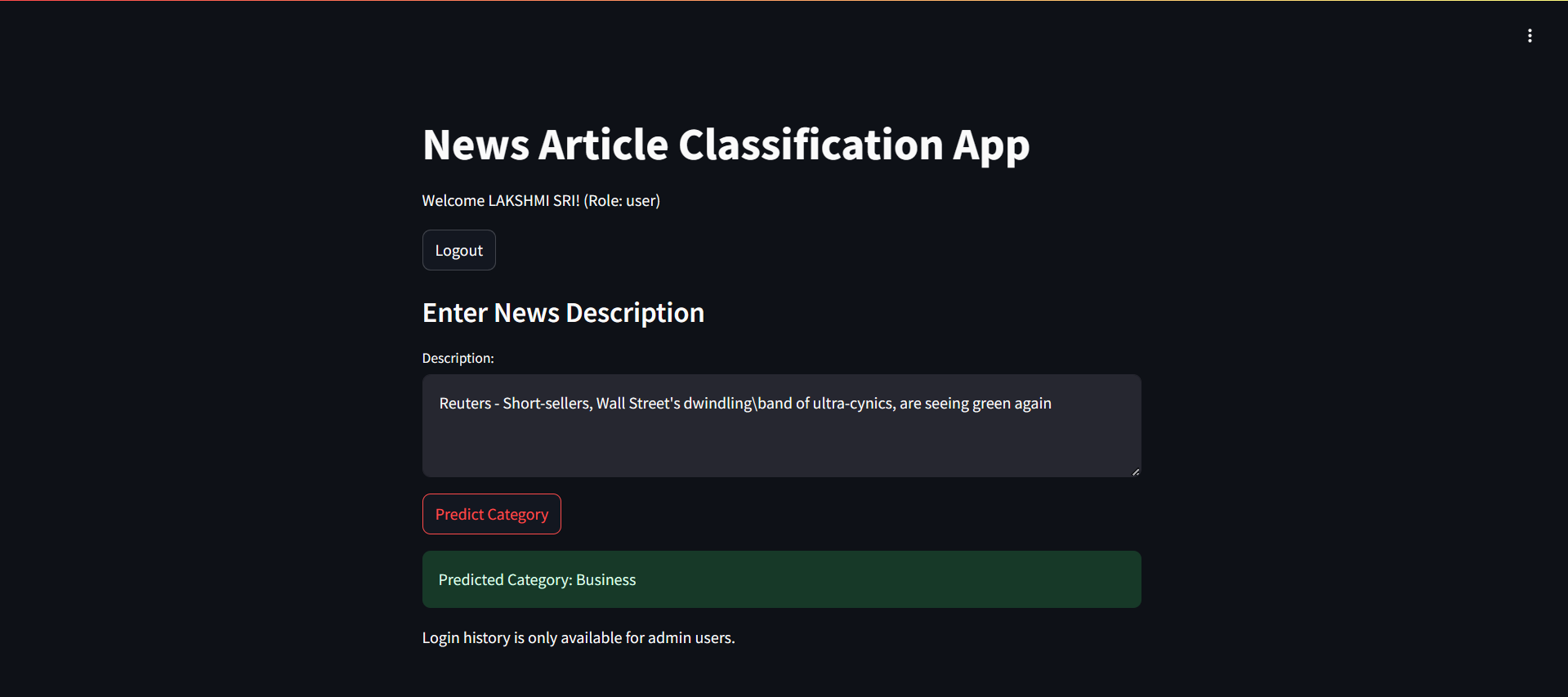
**Summary**

* **AWS CLI Configuration**: Set up AWS CLI to access S3 for downloading model files.
* **Python Dependencies**: Installed required libraries to work with the model (joblib, scikit-learn).
* **MySQL Setup**: Installed and configured MySQL to log user activities.
* **Streamlit App Creation**: Built and tested a web application for predicting news categories and viewing login history, with MySQL integration for storing logs.

**Conclusion**

You successfully deployed a **News Article Categorization Application** on AWS EC2, integrated with an RDS MySQL database for user login tracking. The Streamlit app enables users to log in, track their history, and classify news articles into predefined categories using a pre-trained Logistic Regression model and TF-IDF vectorizer. The app is securely hosted, with proper IAM roles for AWS resource access. Your project integrates machine learning, cloud infrastructure, and web development, providing a scalable and functional solution for categorizing news articles.

**STREAMLIT OUTPUT**

****