Deploying a Multi-label Text Classification BERT Model on AWS SageMaker with Netlify and AWS Lambda

Objective:

Deploy a multi-label text classification BERT model on AWS SageMaker for inference, with a web application hosted on Netlify that sends POST requests to AWS Lambda for predictions.

Steps:

- Train and Save the BERT Model:
  - Train a multi-label text classification model using BERT architecture.
  - Save the trained model artifacts (model weights, tokenizer, etc.) in a format compatible with SageMaker, such as a TensorFlow SavedModel.
- Set Up AWS SageMaker:
  - Create a SageMaker endpoint for hosting the BERT model.
  - Upload the saved model artifacts to an S3 bucket.
  - Configure the SageMaker endpoint with the necessary security settings.
- Create AWS Lambda Function:
  - Develop an AWS Lambda function that will handle incoming POST requests from the Netlify web app.
  - Use a runtime that supports the BERT model's inference requirements, such as Python 3.8.
  - Include the AWS SDK for Python (Boto3) to interact with SageMaker.
- Grant Lambda Execution Role Permissions:
  - Assign an IAM role to the Lambda function with the necessary permissions to invoke the SageMaker endpoint and access the S3 bucket containing the model artifacts.
- Deploy the Lambda Function:
  - Deploy the Lambda function and make note of the endpoint URL.
- Set Up Netlify Web App:
  - Develop a web application using the framework of your choice (React, Vue, Angular, etc.).
  - Implement a form that takes user input for text classification.

    - Configure the form to send POST requests to the AWS Lambda function's endpoint.
- Connect Netlify to Lambda:
    - In the Netlify project settings, configure environment variables containing AWS Lambda endpoint details.
    - Set up the Netlify deployment to automatically trigger when changes are pushed to the repository.
- Testing:
    - Test the end-to-end flow by entering text input in the web app, submitting the form, and verifying the predictions returned by the Lambda function.

Monitoring:

After deploying a machine learning model, it is crucial to monitor various metrics to ensure its performance, reliability, and effectiveness over time. Here are different types of metrics that someone should monitor after deployment:

- Accuracy:
    - Definition: The proportion of correctly classified instances among the total instances.
    - Importance: Provides an overall measure of the model's correctness.
    - Considerations: May not be sufficient for imbalanced datasets, and additional metrics are needed.
- Precision, Recall, and F1 Score:
    - Precision:
        - Definition: The proportion of true positives among all predicted positives.
        - Importance: Indicates how many of the predicted positive instances are actually positive.
    - Recall:
        - Definition: The proportion of true positives among all actual positives.
        - Importance: Measures the model's ability to capture all positive instances.
    - F1 Score:
        - Definition: The harmonic mean of precision and recall.

        - Importance: Balances precision and recall, especially in imbalanced datasets.
- Confusion Matrix:
    - Definition: A table showing true positives, true negatives, false positives, and false negatives.
    - Importance: Provides a detailed breakdown of the model's performance across different classes.
- Area Under the Receiver Operating Characteristic (ROC AUC):
    - Definition: Measures the area under the ROC curve, which represents the trade-off between true positive rate and false positive rate.
    - Importance: Useful for binary classification models, indicating the model's ability to distinguish between classes.
- Latency and Throughput:
    - Latency Definition: The time it takes for a model to generate predictions.
    - Throughput Definition: The number of predictions a model can make in a given time.
    - Importance: Important for real-time applications; high latency may impact user experience.
- Resource Utilization:
    - CPU and Memory Usage Definition: Monitor the resources consumed during inference.
    - Importance: Helps in optimizing the model for efficient resource utilization.