Text classification model with SageMaker BlazingText

Abstract

The problem at hand revolves around integrating an Amazon SageMaker model into a web-based application. The motivation behind this lies in the desire to make an NLP text classifier model, particularly employing the BlazingText algorithm (Amazon SageMaker, 2024) as in the past I have created a classifier using Word2Vec (Mikolov, 2013).

Within the context of the AWS AI Ecosystem, leveraging SageMaker offers a comprehensive solution for both training and deploying machine learning models. By utilizing built-in algorithms like BlazingText, development time is streamlined, allowing a focus on deployment and integration efforts. Crucially, AWS provides robust tools such as SageMaker endpoints and AWS Lambda, facilitating seamless communication between the model endpoint and the front-end web application.

This project not only demonstrates proficiency in machine learning model development but also showcases the ability to deliver end-to-end solutions within the AWS ecosystem. Given AWS's widespread adoption in industry for solving problems at scale, showcasing this project in a portfolio underscores one's competence in deploying AI solutions in real-world settings.

Contents

Abstract 2

Introduction 3

AI Ecosystem Architecture Used 3

Model Description 4

Scalability Considerations 10

References 11

Figure 1: Architecture used to create AI/ML ecosystem. 4
Figure 2: AWS SageMaker Notebook Instances. 4
Figure 3: AWS SageMaker Training Jobs. 5
Figure 4: AWS SageMaker Endpoints. 5
Figure 5: AWS Lambda Function. 6
Figure 6: AWS Lambda Code Source. 6
Figure 7: AWS API Gateway. 7
Figure 8: AWS API Gateway Method Execution. 7
Figure 9: AWS API Gateway Response. 7
Figure 10: AWS IAM Roles. 8
Figure 11: AWS S3 Buckets. 8
Figure 12: AWS S3 Bucket 1. 9
Figure 13: AWS S3 Bucket 2. 9
Figure 14: AWS CloudWatch Billing Alarm. 9

Introduction

In today's rapidly growing era, companies are continuously searching for innovative methods to increase customer engagement and improve support services. Natural Language Understanding (NLU) is revolutionizing customer interactions by automating responses and delivering more context-sensitive information. Utilizing the AWS AI framework, particularly Amazon SageMaker and AWS Lambda, allows for the deployment of sophisticated machine learning models. These models are capable of understanding and sorting customer queries, thereby enhancing both the speed and customization of customer service experiences.

The main objective is to establish an AI-driven system that can automatically categorize customer requests based on their content, be it billing, technical support, or general queries, while also gauging the urgency and emotional tone to prioritize and customize responses effectively. This strategy enhances the productivity of human agents by allowing them to concentrate on more intricate and urgent issues, ultimately boosting customer satisfaction.

Amazon SageMaker, along with BlazingText, provides a robust solution for text classification due to their advanced functionalities and user-friendly design. SageMaker offers a holistic platform for comprehensive machine learning processes, facilitating efficient model training, deployment, and management at scale. BlazingText excels in processing text data by creating efficient Word2Vec embeddings, essential for understanding textual semantics. Its ability to perform unsupervised learning enables it to generate high-quality embeddings without requiring labeled (Lasseter, n.d.) Data, making it adaptable for diverse textual applications. Moreover, the integration of SageMaker with AWS infrastructure delivers strong performance, scalability, and cost efficiency, positioning it as a superior choice for businesses aiming to implement text classification models in real-world settings. Using AWS SageMaker and BlazingText simplifies the development process, enhances training efficiency, and ensures smooth deployment of text classification systems.

This comprehensive approach not only outlines the practical applications of our solution but also details the technical procedures involved, effectively linking the business challenge with its technological resolution. This dual perspective clarifies both the purpose and functionality of our solution.

AI Ecosystem Architecture Used

The AWS architecture in Figure 1 shows the process for text classification utilizing Amazon SageMaker's BlazingText model. At the start of the workflow, users submit text data for classification. This could be done through a web interface or API that accepts user inputs. Initially, the user's data interacts with an S3 bucket that is set up for static web hosting. This suggests that a front-end application is hosted here, possibly providing an interface for users to input their data. Once the data is inputted, it is routed through the API Gateway. This AWS service manages the traffic between the users and back-end services, acting as a gatekeeper to process requests and route them appropriately. After passing through the API Gateway, the data is processed by a Lambda function. This serverless compute service executes code in response to events—in this case, the receipt of text data. The Lambda function likely handles tasks like pre-processing the text data before sending it to the

SageMaker endpoint. The processed data is then sent to a SageMaker endpoint that hosts the BlazingText model. This model performs the actual text classification based on the input data. Finally, the results from the SageMaker model are sent back to the user. This output could be in the form of classified text data or predictions that the user application displays or uses in further processes.

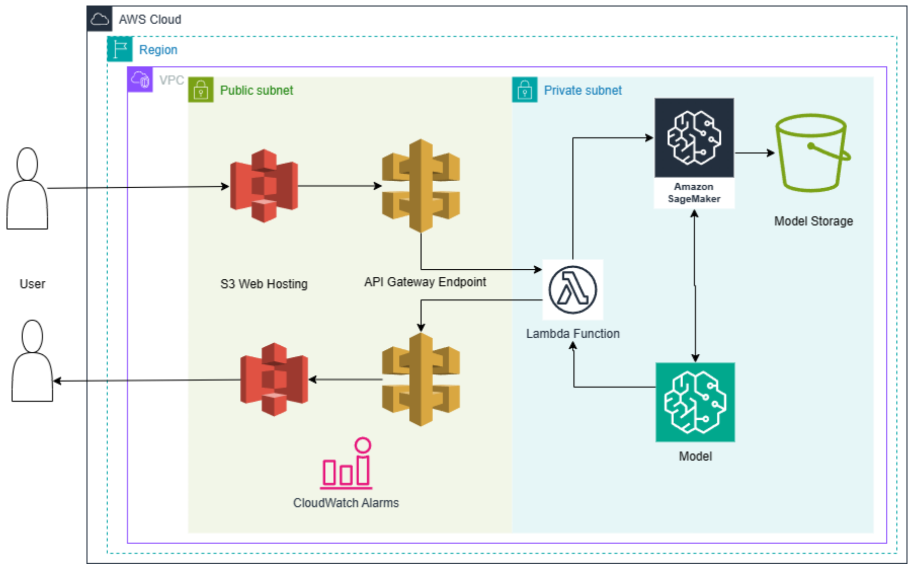


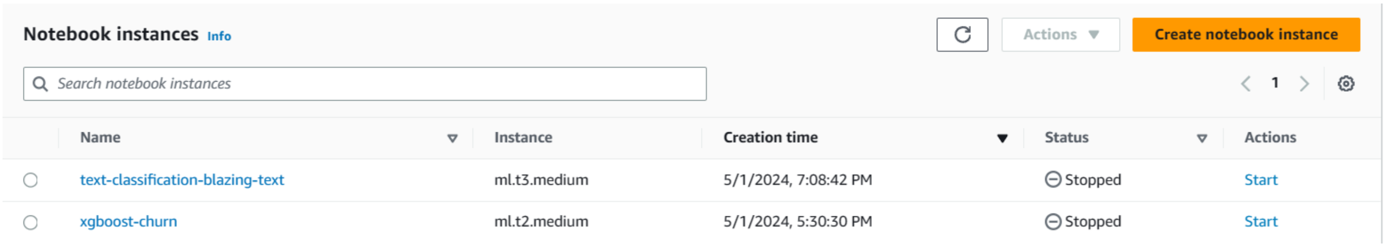
Figure 1: Architecture used to create AI/ML ecosystem.

Amazon Web Services Used

1. Amazon SageMaker

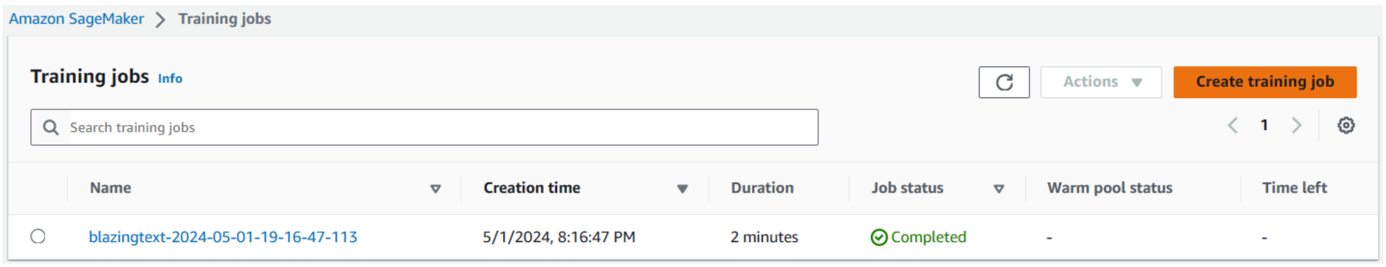
I worked with the Sagemaker Python SDK that handles some of the complexity involved in coding while at the same time limiting the user-defined customizations. The key factor here was our use of the aws container BlazingText which helped with all the necessary dependencies and configurations. To instantiate the model, I configured the BlazingText estimator with several parameters. I named my notebook instance 'text-classification-blazing-text' as shown in Figure 2.

I was having some issues selecting EC2 instances as I am using the free-tire account. The model training took about 2 minutes as shown in Figure 3, with ml.m4.4xlarge instance as it was the minimum compute required for the task. (sagemaker, 2024)



	Name	Instance	Creation time	Status	Actions
○	text-classification-blazing-text	ml.t3.medium	5/1/2024, 7:08:42 PM	Stopped	Start
○	xgboost-churn	ml.t2.medium	5/1/2024, 5:30:30 PM	Stopped	Start

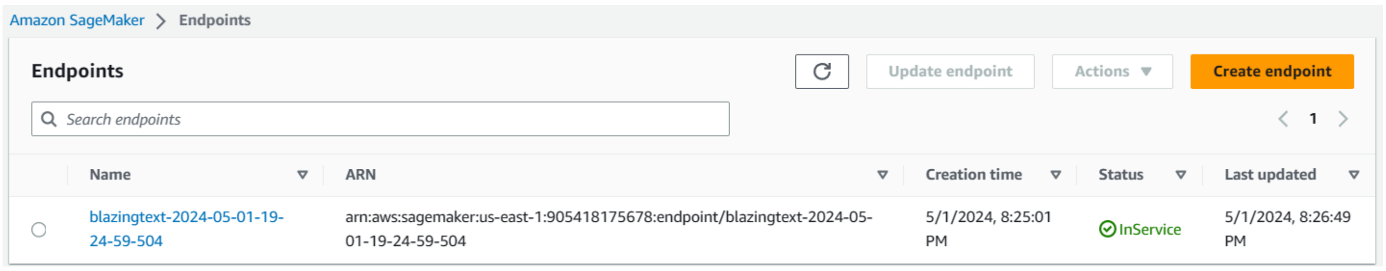
Figure 2: AWS SageMaker Notebook Instances.



Name	Creation time	Duration	Job status	Warm pool status	Time left
blazingtext-2024-05-01-19-16-47-113	5/1/2024, 8:16:47 PM	2 minutes	Completed	-	-

Figure 3: AWS SageMaker Training Jobs.

For deployment, we transitioned the trained model to a SageMaker endpoint named 'blazingtext-2024-05-01-19-24-59-504' as shown in the Figure 4, which was configured to handle prediction requests. This step involved setting up another **EC2 instance (instance type = 'ml.t2.medium')**, though this time the focus was on optimizing for inference and minimize operational costs.



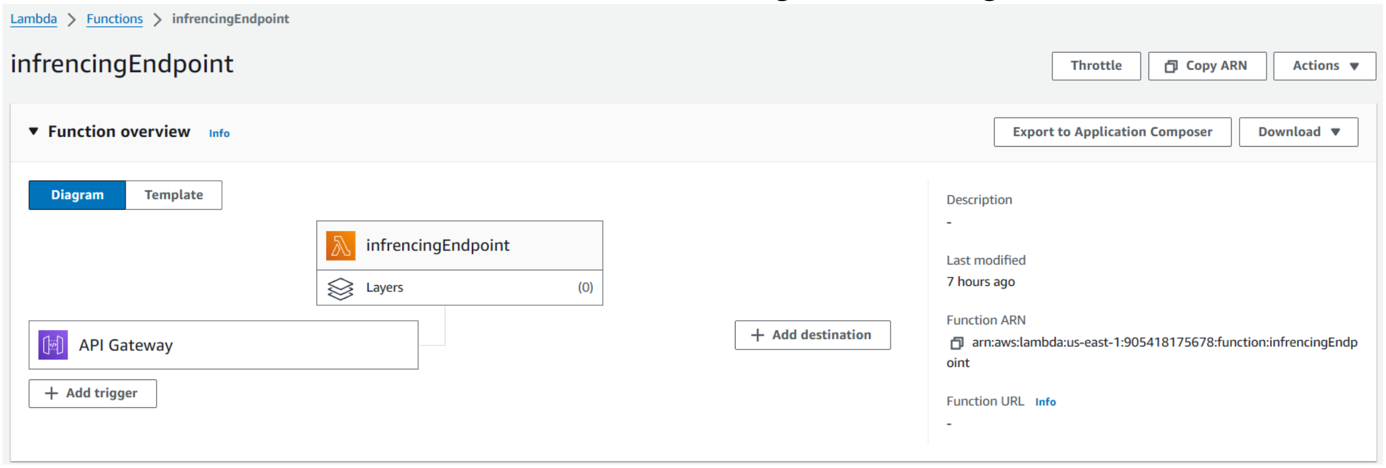
Name	ARN	Creation time	Status	Last updated
blazingtext-2024-05-01-19-24-59-504	arn:aws:sagemaker:us-east-1:905418175678:endpoint/blazingtext-2024-05-01-19-24-59-504	5/1/2024, 8:25:01 PM	InService	5/1/2024, 8:26:49 PM

Figure 4: AWS SageMaker Endpoints.

The deployed model expects input data in a JSON format as whereas during the training the data format is not important. The endpoint processes the incoming requests and return predictions which needed to parsed and then shown to the end-user.

2. AWS Lambda

To make the model accessible to user as a Web interface, An AWS Lambda Function named 'infrencingEndpoint' as shown in Figure 5, is connected with SageMaker endpoint. Lambda function used an intermediary by taking the data through pre-processing and interactions with SageMaker, given in Figure 6. After fixing the issues with the permission to sagemaker and increasing the runtime, this setup was facilitated by an **AWS IAM** role specifically for Lambda to grant it necessary permissions and better runtime. (Patrick Denny, 2024)



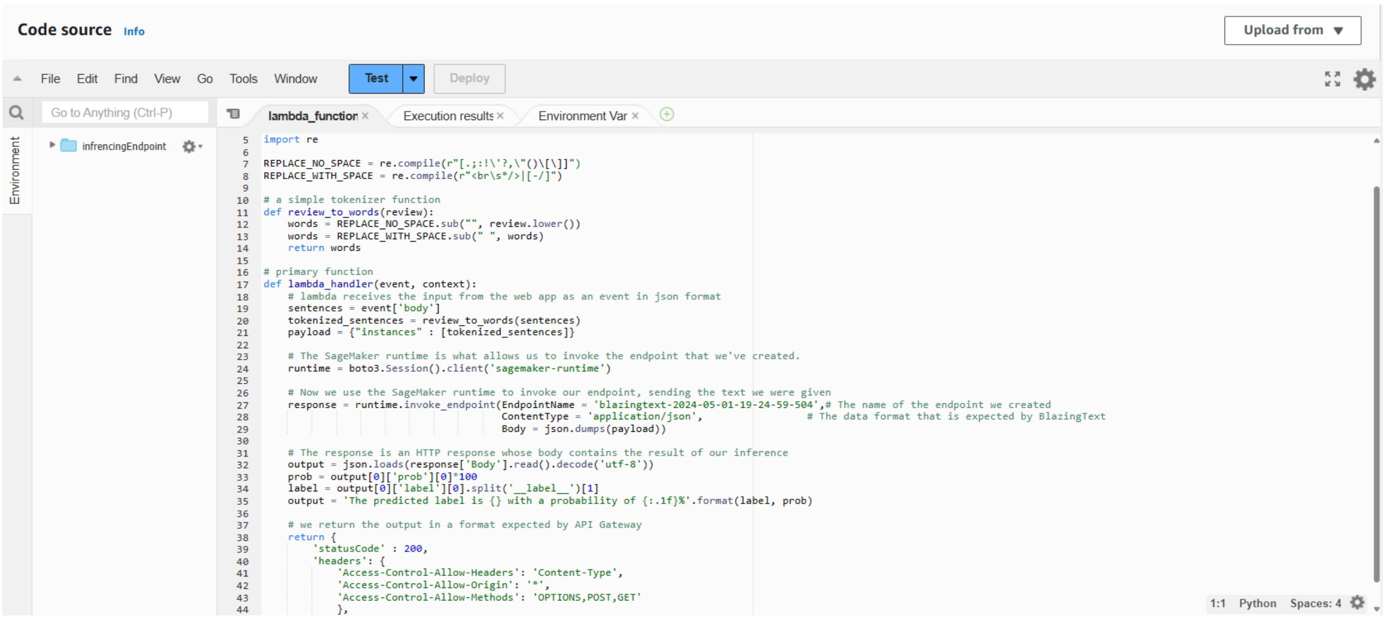
Description
-

Last modified
7 hours ago

Function ARN
arn:aws:lambda:us-east-1:905418175678:function:infrencingEndpoint

Function URL
-

Figure 5: AWS Lambda Function.

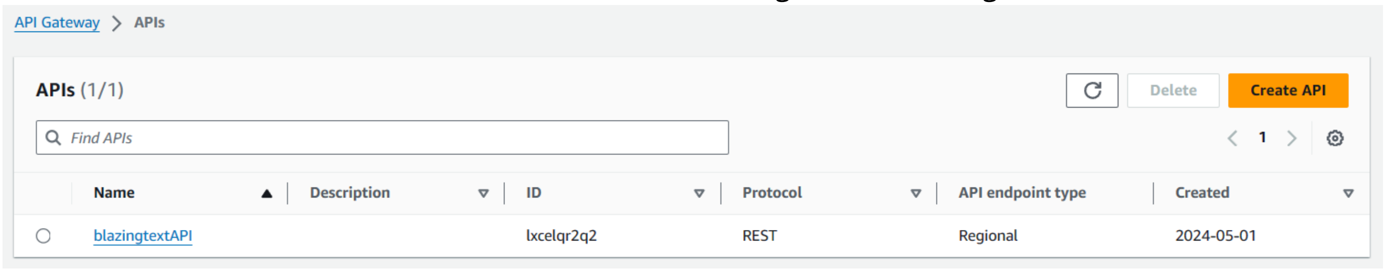


```python
import re

REPLACE_NO_SPACE = re.compile(r"[.;:!\'?,\"()\[\]]")
REPLACE_WITH_SPACE = re.compile(r"<br\s*/>|[-/]")

# a simple tokenizer function
def review_to_words(review):
    words = REPLACE_NO_SPACE.sub("", review.lower())
    words = REPLACE_WITH_SPACE.sub(" ", words)
    return words

# primary function
def lambda_handler(event, context):
    # lambda receives the input from the web app as an event in json format
    sentences = event['body']
    tokenized_sentences = review_to_words(sentences)
    payload = {"instances" : [tokenized_sentences]}

    # The SageMaker runtime is what allows us to invoke the endpoint that we've created.
    runtime = boto3.Session().client('sagemaker-runtime')

    # Now we use the SageMaker runtime to invoke our endpoint, sending the text we were given
    response = runtime.invoke_endpoint(EndpointName = 'blazingtext-2024-05-01-19-24-59-504',# The name of the endpoint we created
                                       ContentType = 'application/json',                 # The data format that is expected by BlazingText
                                       Body = json.dumps(payload))

    # The response is an HTTP response whose body contains the result of our inference
    output = json.loads(response['Body'].read().decode('utf-8'))
    prob = output[0]['prob'][0]*100
    label = output[0]['label'][0].split('__label__')[1]
    output = 'The predicted label is {} with a probability of {:.1f}%'.format(label, prob)

    # we return the output in a format expected by API Gateway
    return {
        'statusCode' : 200,
        'headers': {
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
```

Figure 6: AWS Lambda Code Source.

3. Amazon API Gateway

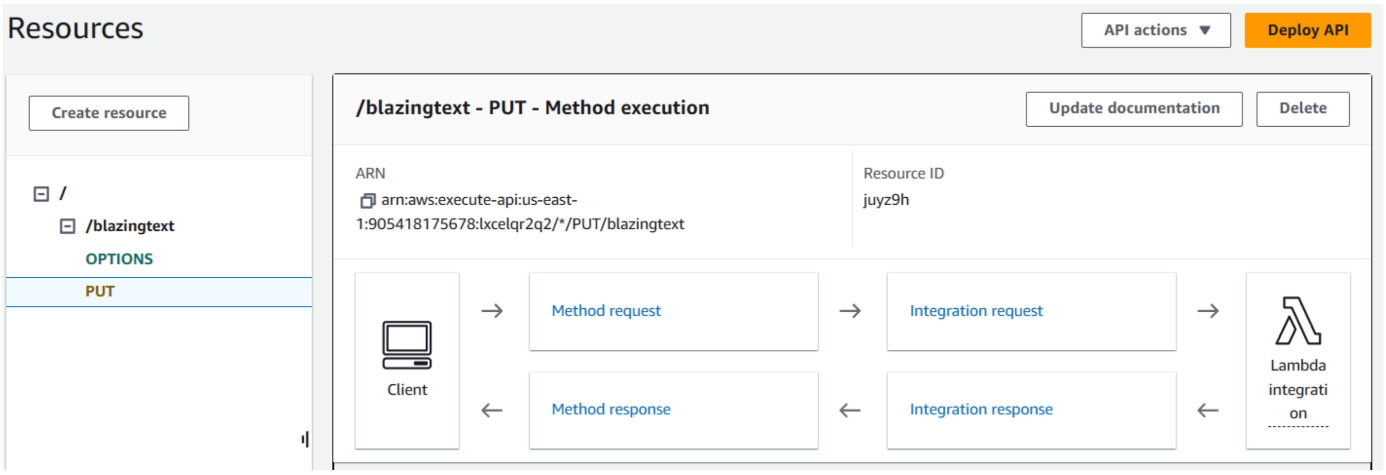
Following Lambda, AWS API Gateway (Olsen, 19 JUL 2018) named 'blazingtextAPI, as given in Figure 7, is used to create a public HTTP endpoint. This gateway sends user requests received over the frontend into API calls that trigger the Lambda function as shown in Figure 8, thereby integrating our entire backend workflow. I tried to test the API Gateway using the request body {"body": "Michael Jackson was the greatest singer and dancer in the world"} and received the result as /blazingtext - PUT method test results shown in Figure 9. (Amazon Web Services, 2024)


API Gateway > APIs

APIs (1/1)

Name	Description	ID	Protocol	API endpoint type	Created
blazingtextAPI		lxcelqr2q2	REST	Regional	2024-05-01

Figure 7: AWS API Gateway.

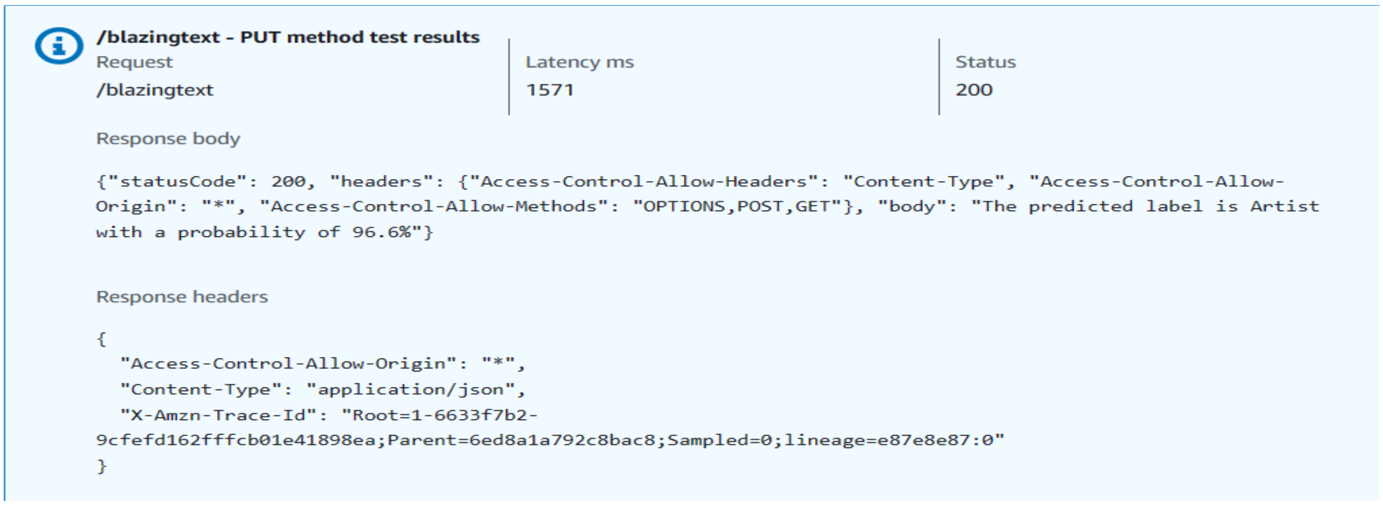

Resources

/blazingtext - PUT - Method execution

ARN
arn:aws:execute-api:us-east-1:905418175678:lxcelqr2q2/*/PUT/blazingtext

Resource ID
juyz9h

Figure 8: AWS API Gateway Method Execution.


/blazingtext - PUT method test results

Request
/blazingtext

Latency ms
1571

Status
200

Response body

```
{"statusCode": 200, "headers": {"Access-Control-Allow-Headers": "Content-Type", "Access-Control-Allow-Origin": "*", "Access-Control-Allow-Methods": "OPTIONS,POST,GET"}, "body": "The predicted label is Artist with a probability of 96.6%"}
```

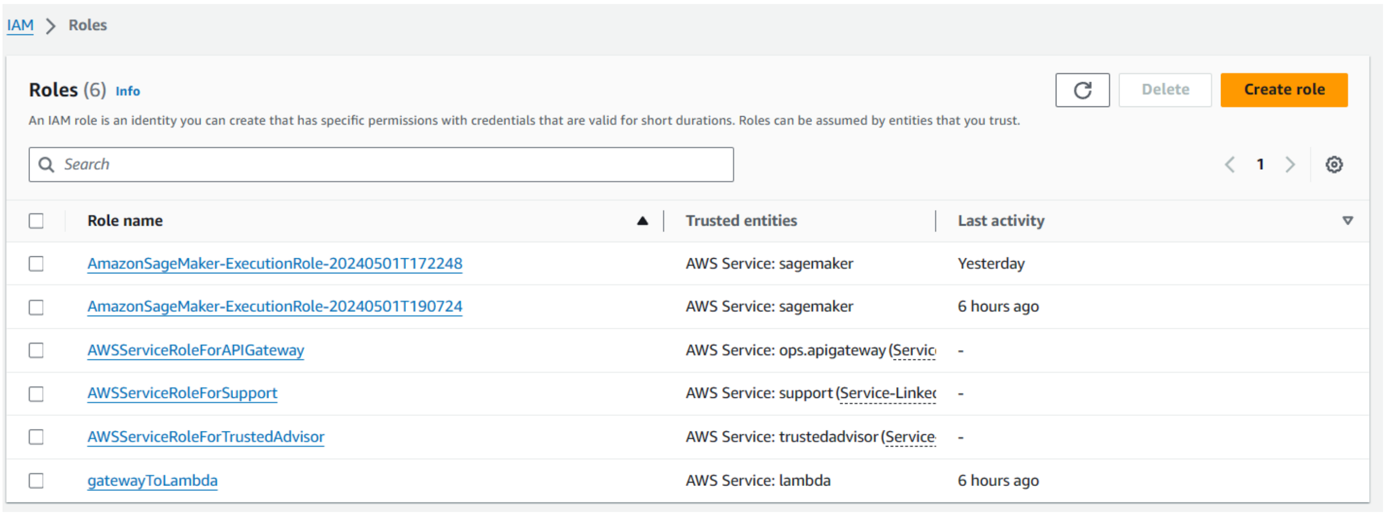
Response headers

```
{
  "Access-Control-Allow-Origin": "*",
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-6633f7b2-9cfefd162fffcb01e41898ea;Parent=6ed8a1a792c8bac8;Sampled=0;lineage=e87e8e87:0"
}
```

Figure 9: AWS API Gateway Response.

4. AWS IAM

AWS Identity and Access Management (IAM) plays a crucial role in maintaining the security of our operations by providing detailed control over access to AWS resources. Through IAM, we have implemented strict policies that define permissions and roles, ensuring the protection of data. This includes restricting access to the S3 bucket where data is stored. In order to access aws services through *boto3 sdk*, I have given permissions to other services through roles as given in Figure 10. (Denny, 2024)



IAM > Roles

Roles (6) Info

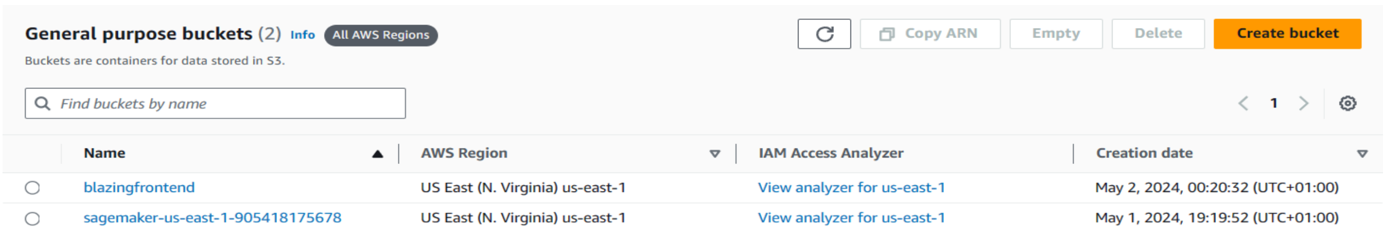
An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AmazonSageMaker-ExecutionRole-20240501T172248	AWS Service: sagemaker	Yesterday
AmazonSageMaker-ExecutionRole-20240501T190724	AWS Service: sagemaker	6 hours ago
AWSServiceRoleForAPIGateway	AWS Service: ops.apigateway (Servic	-
AWSServiceRoleForSupport	AWS Service: support (Service-Linke	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service	-
gatewayToLambda	AWS Service: lambda	6 hours ago

Figure 10: AWS IAM Roles.

5. AWS S3

Amazon Simple Storage Service (S3) offers scalable and secure object storage that is essential for our machine learning application. It handles the extensive datasets needed for model training and securely maintains the model's binary files. Due to its durability and high availability, S3 is perfectly suited for storing our vital data backups. In this environment, I have used two S3 buckets (Figure 11). First one is used to store the data and the model as shown in the Figure 12 and bucket involved in making our solution publicly accessible by hosting a simple web application on Amazon S3, which interacted with the backend through the API Gateway as shown in Figure 13. This allowed users to interact with our model seamlessly, providing text input and receiving predictions directly through a browser-based interface.

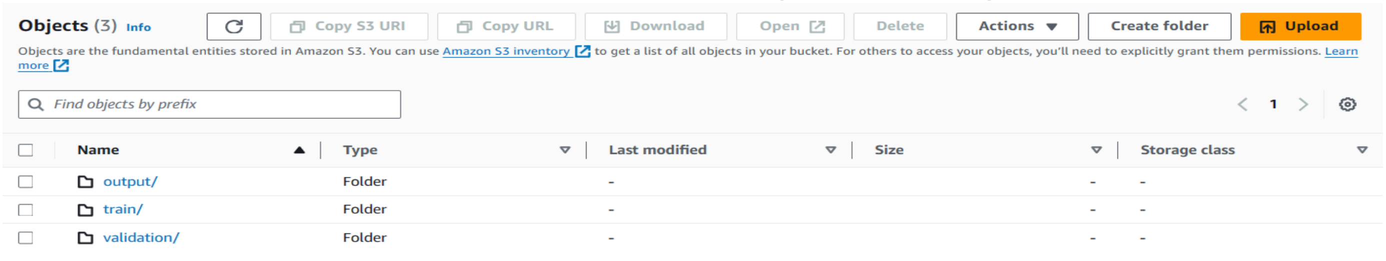


General purpose buckets (2) Info All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
blazingfrontend	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 2, 2024, 00:20:32 (UTC+01:00)
sagemaker-us-east-1-905418175678	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 1, 2024, 19:19:52 (UTC+01:00)

Figure 11: AWS S3 Buckets.

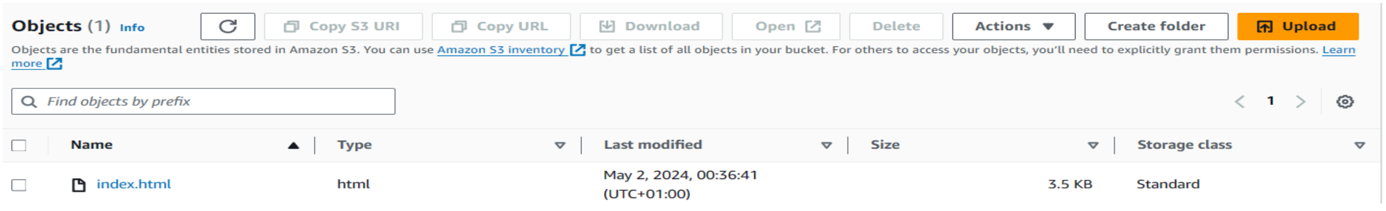


Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
output/	Folder	-	-	-
train/	Folder	-	-	-
validation/	Folder	-	-	-

Figure 12: AWS S3 Bucket 1.



Objects (1) Info

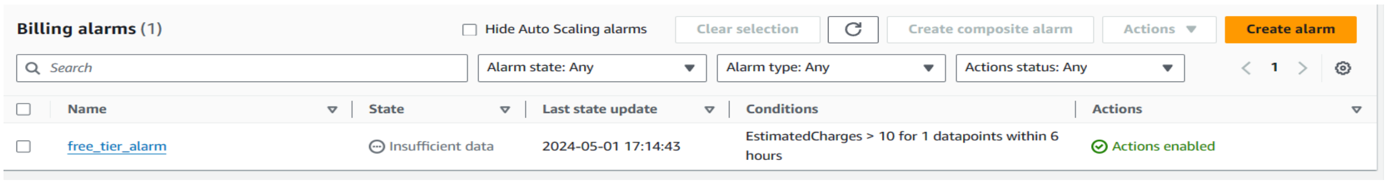
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	May 2, 2024, 00:36:41 (UTC+01:00)	3.5 KB	Standard

Figure 13: AWS S3 Bucket 2.

6. AWS CloudWatch

Monitoring the performance and health of our application is crucial, and AWS CloudWatch is central to this task. It shows real-time monitoring and logging capabilities for compute, training, APIs hits which allow us to observe the system's operation, set alarms, and automatically respond to changes in the environment. I used cloudwatch to setup a threshold EstimatedCharges > $10 for 1 datapoints within 6 hours as I am using a free-tier account as shown in Figure 14.



Billing alarms (1)

Alarm state: Any · Alarm type: Any · Actions status: Any

Name	State	Last state update	Conditions	Actions
free_tier_alarm	Insufficient data	2024-05-01 17:14:43	EstimatedCharges > 10 for 1 datapoints within 6 hours	Actions enabled

Figure 14: AWS CloudWatch Billing Alarm.

AWS's robust cloud infrastructure to train and deploy a machine learning model efficiently. By using services like SageMaker, S3, CloudWatch, EC2, Lambda, and API Gateway, have developed a scalable, cost-effective solution capable of processing and predicting large volumes of text data, significantly enhancing our application's accessibility and usability.

Model Description

Firstly, steps of gathering and sorting out the appropriate training data (Gupta, 2018) for the fitted BlazingText model. The DBpedia ontology is developed from a 2014 version of DBpedia by selecting 14 nonintersecting classes. It has been trained with 560,000 examples and tested with 70,000 examples.

In the context of our problem, we leverage Amazon SageMaker and the BlazingText algorithm for text classification. BlazingText, akin to the popular word2vec model but optimized for parallel processing, is particularly effective for categorizing large volumes of text data swiftly. Our primary datasets are hosted in an AWS S3 bucket, segmented into training, validation, and testing subsets to facilitate effective model training and evaluation.

The use Amazon SageMaker and BlazingText algorithm (Amazon SageMaker, 2024) for text classification in the context of the problem. BlazingText, a popular word2vec model but optimized for parallel processing, is efficient with large text categorization. The S3 bucket is used to store the datasets segmented into training, validation, and evaluation as the model would require for the training and evaluation purpose.

A containerized BlazingText instance is created using the SageMaker SDK in the code workflow. Further steps consist of a container installation where it takes BlazingText image from AWS's repository ensuring the updated version of algorithm. Estimator object is then instantiated including EC2 configuration details that will resemble the type, volume size, and instance role used for access control, as it is important that it determines the volume of computational resources which is quite pivotal in maintaining the balance between the cost and performance.

In the training process, using a text/plain format with file loading `sagemaker.session.s3_input`, specifies how the data will be split during training. The `fit` which is method of our estimator object executes training including logs (used when tracking accuracy metrics) to monitor programing. Our model achieved good accuracy scores post the training phase—98.69% on training data and 97% on validation data—showing strong learning capabilities and generalization. Amazon SageMaker Automatic Model Tuning can be used in case of the low accuracies and over fitting. (Amazon Web Services, Sep 11, 2019)

For the model deployment, which is trained and stored on S3. The EC2 instance type ('ml.t2.medium') for optimizing cost-efficiency is selected. The deployed model used as an endpoint that expects inference data in `application/json` format. This configuration helps in efficient data exchange by other aws services, such as a Lambda function, to process real-time text classification requests.

Scalability Considerations

When planning for scalability in an application using Amazon SageMaker and the BlazingText model, several key aspects need to be considered, including data handling, computational resources, and system architecture.

The data scalability is important. As the volume of data increases, the storage and data processing methods must adapt. In our case, use AWS S3 for data storage provides robust scalability and data redundancy. However, as the dataset grows, considerations around data partitioning and possibly using S3's lifecycle policies to archive older data become important. Amazon DynamoDB can be used in case if fast and predictable performance with seamless scalability is needed. Amazon DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance. (Amazon Web Services, 2024)

Computational scalability involves selecting and adjusting the EC2 instances used for training and deployment. While training a model like BlazingText, which requires significant computational power, choosing the right instance type (like 'ml.m4.4xlarge' for training) that balances cost and performance is essential. For deployment, a smaller instance (such as 'ml.t2.medium') might be sufficient, but this depends on the expected load. **Auto-scaling** policies should be implemented to dynamically adjust the number of instances in response to real-time demand, ensuring efficient resource use without manual intervention. (Denny, 2024)

The overall system architecture must support scalability. This involves setting up a load balancer in front of the SageMaker endpoint to distribute incoming inference requests evenly across multiple instances. Implementing a serverless architecture, such as AWS Lambda, to handle request pre-processing and post processing can scale automatically with the request load, thereby managing sudden spikes in traffic effectively. (Denny, 2024) Additionally, monitoring and logging mechanisms must be robust, using services like AWS CloudWatch to track usage patterns and system health, which aids in proactive scaling and system optimization.

In summary, a holistic approach to scalability, addressing data management, computational resources, and architectural design, ensures that the application remains responsive and cost-efficient as demands increase. (Clarke Rodgers, 2019) For instance, an application handling real-time text classification for social media posts would need to rapidly adjust computational resources during high-traffic events like public holidays or major news breaks, requiring efficient auto-scaling and data streaming capabilities.

References

Amazon SageMaker, 2024. *BlazingText algorithm*. [Online]
Available at: https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext.html

Amazon Web Services, 2024. *Amazon DynamoDB Documentation*. [Online]
Available at: https://docs.aws.amazon.com/dynamodb/?icmpid=docs_homepage_featuredsvcs

Amazon Web Services, 2024. *Build an API Gateway REST API with Lambda integration*. [Online]
Available at: https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started-with-lambda-integration.html

Amazon Web Services, Sep 11, 2019. *Amazon SageMaker Technical Deep Dive Series*. [Online]
Available at:
https://www.youtube.com/watch?v=xpZFNIOaQns&list=PLhr1KZpdzukcOr_6j_zmSrvYnLUtgqsZz&index=8

Clarke Rodgers, 2019. *A Holistic Approach to Securing AWS at Enterprise Scale*. [Online]
Available at: https://aws.amazon.com/blogs/enterprise-strategy/use-the-cloud-to-protect-the-cloud-a-holistic-approach-to-securing-aws-at-enterprise-scale/

Denny, P., 2024. *AWS Cloud Security - Tuesday Week 4 Material (2024)*. [Online]
Available at: https://learn.ul.ie/d2l/le/lessons/17937/topics/634587

Denny, P., 2024. *Week 8 - Cloud Architecture, Automatic Scaling*. [Online]
Available at: https://learn.ul.ie/d2l/le/lessons/17937/units/248987

Denny, P., 2024. *Week 8 - Cloud Architecture, Automatic Scaling*. [Online]
Available at: https://learn.ul.ie/d2l/le/lessons/17937/units/248987

Gupta, S., 2018. *Github*. [Online]
Available at: https://github.com/saurabh3949/Text-Classification-Datasets/

Lasseter, A., n.d. [Online]
Available at: https://github.com/austinlasseter/blazing-text-dbpedia

Mikolov, T., 2013. *Distributed Representations of Words and Phrases*. [Online]
Available at:
https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

Olsen, R., 19 JUL 2018. *AWS Machine Learning Blog*. [Online]
Available at: https://aws.amazon.com/blogs/machine-learning/call-an-amazon-sagemaker-model-endpoint-using-amazon-api-gateway-and-aws-lambda/

Patrick Denny, 2024. *AWS Cloud Security - Tuesday Week 4 Material*. [Online]
Available at: https://learn.ul.ie/d2l/le/lessons/17937/topics/634587

sagemaker, 2024. *Amazon SageMaker Python SDK*. [Online]
Available at: https://sagemaker.readthedocs.io/en/stable/overview.html#sagemaker-serverless-inference