Carnegie Mellon University

04652 EAI Systems, Spring 2024

Mini 2, Assignment 4: Dashboards in Dash, part 2 w/AWS

Expected individual study hours: 16

Points: 100

Purpose

In lab 3 of this Mini, you implemented the Iris Dashboard, including support for examining data, creating instances of Tensorflow models, training those model instances, and batch testing.

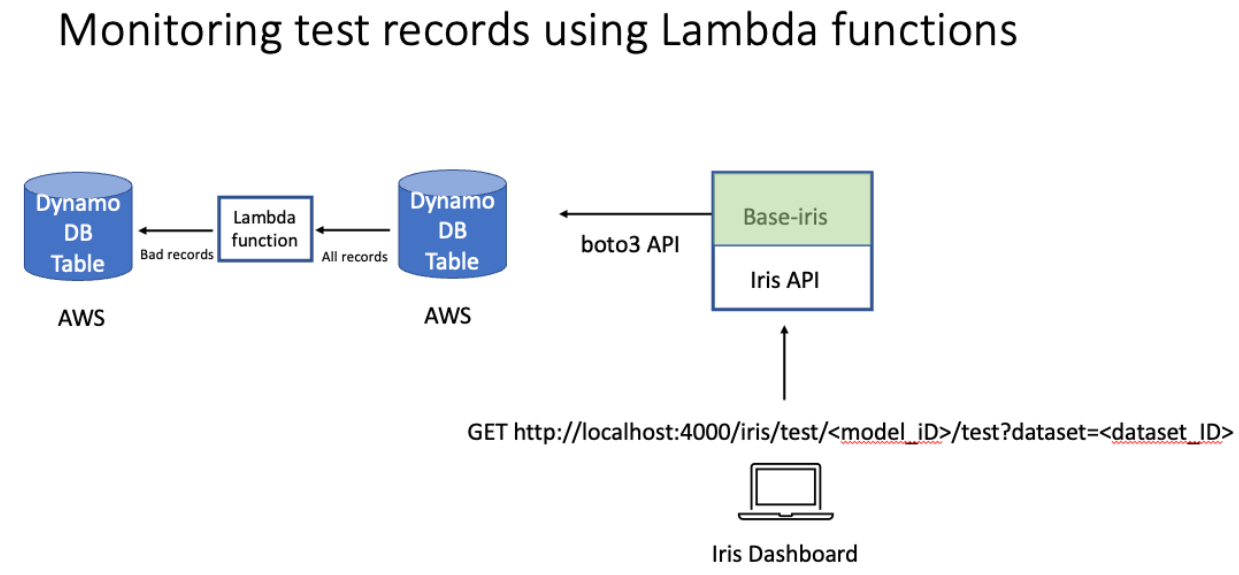
The purpose of this lab 4 assignment is to extend that dashboard in two ways: (1) to write those test records to the Cloud using Amazon AWS DynamoDB tables, and (2) to flag potential problem records in a second DynamoDB table for possible use in retraining the Iris model.

Target Learning Objectives

- The importance of tracking model quality using periodic test sets and recording those test results for later analytics and for use in model retraining.
- To build a working knowledge of core features of a popular Cloud platform such as AWS – including how to instantiate AWS service instances and access them from client programs such as the Iris model components.

Preparation and setup in AWS

The end-to-end architecture of this assignment is as shown in the figure below. The front-end Dash UI issues HTTP requests to your API module. The API module calls in to imported model implementation (base-iris) functions in the same component (no network hop). Finally, the base-iris implementation of the test() function streams records out to an AWS DynamoDB table using their client SDK called boto3. An AWS "Lambda" function receives DB updates and examines them for potential problems which are routed to a second DynamoDB table:



Create your AWS Cloud account and user

1. Navigate to aws.amazon.com and click on the “Sign in to the console” button at the top right of the screen.

2. On the console sign-in page, select “Create a new AWS account”

3. Enter your email address and desired account name in the next page and select “Verify email address”. Respond with the verification code as requested.

4. All of the AWS services we’ll use in this course are part of the Free Tier so will incur no charges on your credit card – even though a card is required to set up the account.

5. Once you have an account established, log in to the console at console.aws.amazon.com using the “Root user” option.

6. Select the “Services” link at the top left of the console page and then “IAM” in the list of available services.

7. The next step is to create a new “User” in the IAM (Identity and Access Management) service and then an API key pair under that user which will allow you to create service instances such as DynamoDB and issue API calls to them. Select “Users” to get started.

8. Select “Add users” at the top right of the User list page.

9. Enter your desired User name and select “Next”

10. Select the “Attach policies directly” option and enter DynamoDB in the “Permissions policies” filter box. Select “AmazonDynamoDBFullAccess” by clicking on its checkbox to the left.

11. Select “Next” to proceed to the Review and create page. Select “Create user” there to complete this step.

Create your API key and secret

12. Once you have set up your user, then click on that user name in the Users page and select the “Security credentials” tab.

13. Under the “Access keys” section, select “Create access key”

14. Select the “Local code” access type and click on the box to understand the recommendation.

15. Select “Next” to create the keys

16. Download and store the credentials file and then edit it to declare the default AWS region to be used such as `af-south-1` for Cape Town. Store the credentials file off your home directory in ~/.aws/credentials

17. The completed credentials file will have a format such as

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
region=af-south-1
```

18. You will use these credentials in setting up access from the "boto3" SDK in your client code. If you have questions on this step, check out: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html

Create your DynamoDB table

19. Return and be sure you're logged in to the aws console (`console.aws.amazon.com`) and have selected the desired region at the top right of the page, e.g. `eu-central-1`

20. Create an instance of a DynamoDB table by selecting DynamoDB from the "Services" link at the top left of your console page. In the DynamoDB "home" page, select "Create table" and provide your desired table name, e.g. IrisExtended. Enter "partition_key" and "sort_key" for the corresponding fields and leave the Default settings selected. Finally, select "Create table" at the bottom right of the page.

Install and test the "boto3" AWS SDK

21. Install the boto3 SDK itself in your Python environment with "pip3 install boto3"

22. You can check quickly that your boto3 SDK is configured correctly and credentials are valid with:

```
python3
# Run this code:
>>import boto3
>>ddb = boto3.client('dynamodb')
>>ddb.describe_limits()
```

You should get an output like the following:

```
{'AccountMaxReadCapacityUnits': 80000, 'AccountMaxWriteCapacityUnits': 80000,
'TableMaxReadCapacityUnits': 40000, 'TableMaxWriteCapacityUnits': 40000,
'ResponseMetadata': {'RequestId': 'CNDPO7FIOSNI7IUMLL15GBTGDFVV4KQNSO5AEMVJF66Q9ASUAAJG',
'HTTPStatusCode': 200, 'HTTPHeaders': {'server': 'Server', 'date': 'Mon, 27 Feb 2023
17:55:35 GMT', 'content-type': 'application/x-amz-json-1.0', 'content-length': '143',
'connection': 'keep-alive', 'x-amzn-requestid':
'CNDPO7FIOSNI7IUMLL15GBTGDFVV4KQNSO5AEMVJF66Q9ASUAAJG', 'x-amz-crc32': '3062975651'},
'RetryAttempts': 0}}
```

End of preparation and setup

Lab task – Extend model implementation with logging to AWS

1. Provided in post_score.py is the "post_score()" function which will write test record results individually to AWS. Your task is to extend your back-end test() function itself to prepare inputs as needed by the post_score() function and to call that function for each record in the test dataframe.

2. The text format of the "Features" field to be written to DynamoDB should be the string value of a dictionary of that record's fields, like:

```
"{'species': 'setosa', 'elevation': 161.8, …, 'area_ratios': 41.26}"
```

You will need to construct this string from the features of the test record being stored.

The full set of fields written to DynamoDB is:

```
'partition_key': current_time,      # set by provided code
'sort_key': "abc",                  # set by provided code
'Features': feature_string,         # computed as above
'Class' : class_string,             # predicted class
'Actual' : actual_string,           # actual class input
'Probability' : prob_string         # prob of prediction
```

3. Included in the lab4_header.py file in Canvas is a number of statements that provide the API key to AWS for your account as well as bind to the DynamoDB table you create to store your results. Fill in those statements with your corresponding credentials and table name.

Preparation and setup – Detect retraining candidates using Lambda functions

The Lambda function filters the incoming records and selects those considered candidates for retraining either due to incorrect predictions or low confidence. Those records are copied to a 2nd DynamoDB table in AWS which can then be examined for possible retraining.

Create your second DynamoDB table for storing retraining records

1. Return and be sure you're logged in to the aws console (`console.aws.amazon.com`) and have selected the desired region at the top right of the page, e.g. `eu-central-1`

2. Create a 2nd instance of a DynamoDB table by selecting DynamoDB from the "Services" link at the top left of your console page. In the DynamoDB "home" page, select "Create table" and provide your desired table name, e.g. IrisExtendedRetrain. Enter "partition_key" and "sort_key" for the corresponding fields and leave the Default settings selected. Finally, select "Create table" at the bottom right of the page.

Create the required AWS Role and Policies

1. In this step, we'll create an AWS "Role" to operate the Lambda function and provide your credentials to it. Roles have attached "Policies" which control what they are able to do. We'll attach two policies to our role: (1) to allow the Lambda function to monitor the DynamoDB stream, and (2) to allow the Lambda function in turn to write to DynamoDB.
2. Return to the AWS console home page and select the IAM service.
3. Select Roles from the left hand menu.
4. Select Create role at the top right
5. Leave the default "AWS Service" option selected, and choose Lambda as the use case and select "Next" at the bottom
6. In the filter policies box enter dynamodb and hit enter
7. In the filtered list, select both `AWSLambdaDynamoDBExecutionRole` and `Amazon-DynamoDBFullAccess`. The first policy allows your lambda function to monitor the DynamoDB stream, and the second policy allows it to write to DynamoDB as well.
8. Proceed to the next page, Enter a name for the Role we're creating (with its attached policies just selected), e.g. IrisLambdaRole. For Select tags, none are required so just select "Create Role".

End of preparation and setup

Lab task – Detect retraining candidates using Lambda functions

1. Download the lambda.py function from Canvas in the Files/Mini2 folder

2. Edit the function and provide your specific DynamoDB retraining table name in the TableName field of the `client.put_item` call, e.g. IrisExtendedRetrain.

3. Return to the DynamoDB console page, select your first table, e.g. IrisExtended (not the retraining table), and then select the "Exports and streams" tab of that table.

4. Select "Turn On" under "DynamoDB stream details" and select "New image" is selected for View type. Click on "Turn on stream"

5. In the Trigger section of the tab, select "Create trigger"

6. Select "Create new" in the function details form

7. Leave "Author from scratch" selected, provide a function name, e.g. IrisLambdaFn, and select Python 3.9 selected as Runtime option. Leave x86_64 as the Architecture.

8. Expand the "Change default execution role" dropdown and select "Use an existing Role". Enter the name of the role created above, e.g. IrisLambdaRole.

9. Select "Create function"

10. In the source view, replace the default code with the downloaded python file you prepared above and select "Deploy"

11. Return to the "Create a trigger" page and select your newly deployed function from the pull-down list and select "Create trigger". Verify the trigger appears as "Enabled" in your DynamoDB table's Trigger section at the bottom of the Exports and Streams tab.

12. Test your complete back-end deployment by creating a new item in the first, scoring, table and verifying that the update is reflected all the way to the second, retraining, table. Do this by opening the DynamoDB page, then select Create Item under "Actions".

13. Hint: Any partition_key and sort_key strings are ok, but then select "Add new attribute" of String type, enter a placeholder for all the scoring fields uploaded with the post_score() function above. These test items can be deleted before running the actual client code so you don't need to enter the actual fields used in your application…just one or more with random content to test the lambda function.

14. After testing the deployed lambda.py function to be sure of your end-to-end configuration, update the function implementation to test for two types of potentially problematic records that should be copied to the new DynamoDB table for consideration in retraining:

    a. Records where the predicted class does not match the ground-truth label
    b. Records where the prediction confidence is below some threshold, e.g. 0.9

Only records matching one of these conditions should be copied to the 2nd table! Hint: in our model using the extended iris features the confidence is almost always 1 and the predicted class almost always matches the expected ground truth. I suggest you "hack" a small test dataset by changing the class names in col 0 to some other incorrect class, e.g. change setosa to versicolor for a few records just to be sure your lambda function has something to detect ☺

15. Use the "Monitor" tab on your Lambda function page to read the function logs if you don't see results appear in your retraining table :-) Click on "View Cloudwatch logs" and select the most recent log entry. Typical issues are missing permissions (check that Role assignment) and mismatched field names in the incoming event from what the function may be looking for once you start modifying the starter template python file.

Upload your updated backend code into GitHub in a "lab4" folder under your already-shared repo. Include screen shots of the record content of both DynamoDB tables with working content shown.

Due

Tuesday, April 30th at 12:59 PM CAT