## What is your experiment about

a) Describe here what your experiment is doing. Provide a reference to your SE project documentation (PDF)

Brief description of SDR classifier: The SDR

is an essential element in HTM (Hierarchical temporal memory) framework as it is

responsible to detect and learn the relationship between the Temporal Memoryâ€™s present

state at time t and the future value at t+n, where n indicates no. of stages in future to

be inferred [1]. SDR works by giving outliers in prediction probability by reinforcing

the imperative weight matrix at each turn. The incorrect predictions are ensured to be

penalized by successive iterations. The weight matrix, which is the key component in

the classifier, is essentially

b) What is the input?

Input consists of three things that is: alpha, bucket index, the actual value and the

input from temporal memory. Where alpha is a random number and is given by the user.

Bucket index is basically the index of the memory where the actual value goes.

Actual value is the actual input value from the user.

Input from temporal memory are basically the activated cells.The input in json looks like this

~~~json

{

"Alpha": 1.0,

"BucketIndex": 4,

"ActualValueInBucket": 44.0,

"InputFromTemporalMemory": [1,3,6,7,8,9,0]

}

~~~

c) What is the output?

Ouput is the basically the array of probabilities of the buckets depicting which bucket

has the highest probability and can be checked for future prediction.

d) What your algoritham does? How ?

We established that the SDR Classifier functions by receiving inputs from the encoder

and the temporal memory, in the code it is specified accordingly:

1. Inputs by the encoder at instant t: This is implemented by a list of

object classification with two values. classification[0] includes the bucket

2. Record number of the current iteration: This is implemented using integer variable recordNum.

3. Activated bit pattern of temporal memory for the input at instant t+n:

This is implemented by initiating a data structure patternNz(1d array of integer) which stores activated cells of temporal memory.[1]

## Cloud Project

The execution of this project according to the workflow is implemented by the class Experiment.cs.

It is situated in MyExperiment directory in the project folder and called by the main Program.cs

for implementation. The purpose of experiment class is to establish the folder path locally

where files will be downloaded from blob storage and the data present in the downloaded file is

executed and uploaded back on azure (Blob & Table). The program will then be executed until

signaled to cancel via a cancellation token. The results of the program will then be uploaded in

the storage blob and table. This stream of actions is illustrated in several methods discussed below.

### Code Description

The first three objects as shown below are established to be used throughout the code.

a) storageProvider: Concerns associations with the blob storage i.e. uploading files in blob and

downloading file from it.

b) logger: Concerns appending structured messages regarding instant at hand throughout the code

called logs.

c) config: Represents the configuration selected for the project, associates with container names,

queue, and group ID etc.

After the object declarations, the constructor for the class is declared as shown.

It is to be noted that configSection is the variable established to bind with config above.

It is to be noted that the argument configSection make use of the descendent configuration (IConfiguarationSection) which internally uses InitHelper class,

in order to read configuration or fetch details from the appconfig.json file of the project.

In the next step a directory is created, Path.combine analogy is used to concatenate the

address using the two arguments that are Environment.SpecialFolder and locally available

path as given in the configuration file, appconfig.json . This ensures a suitable file path.

~~~c#

/// <inheritdoc/>

public async Task RunQueueListener(CancellationToken cancelToken)

{

CloudQueue queue = await AzureQueueOperations.CreateQueueAsync(config, logger);

while (cancelToken.IsCancellationRequested == false)

{

var message = await queue.GetMessageAsync(cancelToken);

try

{

if (message != null)

{

// STEP 1. Reading message from Queue and deserializing

var experimentRequestMessage =

JsonConvert.DeserializeObject<ExerimentRequestMessage>(message.AsString);

logger?.LogInformation(

$"Received message from the queue with experimentID: " +

$"{experimentRequestMessage.ExperimentId}, " +

$"description: {experimentRequestMessage.Description}, " +

$"name: {experimentRequestMessage.Name}");

~~~

Preceding the constructor is the method RunQueueListener. It aims at listening the queue message for

the program to be executed as illustrated in Fig 3.3. Following describes how the method works:

The arguments of the method receive a token that should be cancelled in further time of execution

with type CancellationToken. A queue is created at first using the storage account information of

the user which contains the configuration for the program to be executed.

Following that, the state of the token is checked. At a cancel state (subject to the user

pressing the key), the process is stopped, and an appropriate log is registered. At a non- cancel

state, the method follows 6 steps.

Step I: The message in queue is read as illustrated in the above code. For a not null message

(which signals for the execution of the program), the message is deserialized or converted

from byte stream to object in memory. A log is then logged on the console showcasing details in

the queue message such as groupID, name, description, etc.

~~~C#

// STEP 2. Downloading the input file from the blob storage

var fileToDownload = experimentRequestMessage.InputFile;

var localStorageFilePath = await storageProvider.DownloadInputFile(fileToDownload);

logger?.LogInformation(

$"File download successful. Downloaded file link: {localStorageFilePath}");

// STEP 3. Running SE experiment with inputs from the input file

var result = await Run(localStorageFilePath);

result.Description = experimentRequestMessage.Description;

var resultAsByte = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(result));

~~~

Step II: The name of the training data file present in the queue message is downloaded from the

blob and the local path of the file returned is stored in the variable localStorageFilePath.

As shown in below code snippet the method DownloadInputFile in class AzureBlobOperation.cs is used to

download the input file. Its approach is straightforward as illustrated in below code.

The file name to be downloaded is passed in the arguments. First the local repository path is

created, and the file name is appended. Next, using the storage account connection string an

instance blob is initialized by using the credentials of the account and the URI of the file name

in the storage blob. The contents in the blob are then downloaded in the file using the instance

with DownloadToFileAsync.

~~~C#

public async Task<string> DownloadInputFile(string fileName)

{

if (StringUtilities.isBlankOrNull(fileName))

{

throw new EmptyStringException("File name cannot be empty or null");

}

string localStorageFilePath = Path.Combine(Experiment.DataFolder, new FileInfo(fileName).Name);

Microsoft.Azure.Storage.CloudStorageAccount cloudStorageAccount = Microsoft.Azure.Storage.CloudStorageAccount.Parse(config.StorageConnectionString);

var blob = new CloudBlockBlob(new Uri(fileName), cloudStorageAccount.Credentials);

await blob.DownloadToFileAsync(localStorageFilePath, FileMode.Create);

return localStorageFilePath;

}

~~~

Step III: With the training data file in the local repository the program is run by providing

the path of the file stored. The result of the program returned by Run is serialized or converted

into byte stream for further processing. During the execution of the Run method as illustrated

in the below Fig. 3.6, firstly it desterilizes the data from the downloaded file after that it

runs the software engineering experiment using method RunSoftwareEngineeringExperiment and records

the results with the time instants and duration of execution.

~~~C#

public async Task<ExperimentResult> Run(string localFileName)

{

var seProjectInputDataList =

JsonConvert.DeserializeObject<List<SeProjectInputDataModel>>(FileUtilities.ReadFile(localFileName));

var startTime = DateTime.UtcNow;

// running until the input ends

string uploadedDataURI = await RunSoftwareEngineeringExperiment(seProjectInputDataList);

// Added delay

Thread.Sleep(5000);

var endTime = DateTime.UtcNow;

logger?.LogInformation(

$"Ran the experiment SDRClassifier as per input from the blob storage");

long duration = endTime.Subtract(startTime).Seconds;

// Creating a result file over here.

var res = new ExperimentResult(this.config.GroupId, Guid.NewGuid().ToString());

UpdateExperimentResult(res, startTime, endTime, duration, localFileName, uploadedDataURI);

return res;

}

~~~

In above code, RunSoftwareEngineeringExperiment showcases the loop at which input data is processed

by calling the different methods of the SDR classifier. Then, a file is created to record the

respective results. Below code snippet shows how the SDRClassifer output file is uploaded in blob using method

UploadResultFile.

~~~c#

// STEP 4. Uploading result file to blob storage

var uploadedUri =

await storageProvider.UploadResultFile("ResultFile-" + Guid.NewGuid() + ".txt",

resultAsByte);

logger?.LogInformation($"Uploaded result file on blob");

result.SeExperimentOutputBlobUrl = Encoding.ASCII.GetString(uploadedUri);

// STEP 5. Uploading result file to table storage

await storageProvider.UploadExperimentResult(result);

// STEP 6. Deleting the message from the queue

await queue.DeleteMessageAsync(message, cancelToken);

~~~

Step IV: The result of the program is uploaded in blob storage with the method UploadResultFile in

class AzureBlobOperations.cs. An appropriate log is appended, and the blob URI is stored in result.

The method UploadResultFile uses a straightforward approach as illustrated in the below code snippet,

a blob service client object is created to create an object of container client. The file name of

result is appended in an appropriate local path.

~~~C#

public async Task<byte[]> UploadResultFile(string fileName, byte[] data)

{

if (StringUtilities.isBlankOrNull(fileName))

{

throw new EmptyStringException("File name cannot be empty or null");

}

// Creates a BlobServiceClient object which will be used to create a container client

BlobServiceClient blobServiceClient = new BlobServiceClient(config.StorageConnectionString);

// Create the container and return a container client object

BlobContainerClient containerClient = blobServiceClient.GetBlobContainerClient(config.ResultContainer);

// Create a local file in the ./data/ directory for uploading and downloading

string localFilePath = Path.Combine(Experiment.DataFolder, fileName);

~~~

The file is then uploaded to the blob storage and the URI of the file in blob is returned as

shown.

~~~C#

// Write text to the file

// Adding a check to write a data in a file only if data is not equal to null

// This is important as we need to re-use this method to upload a file in which data has already been written

if (data != null)

{

File.WriteAllBytes(localFilePath, data);

}

// Get a reference to a blob

var blobClient = containerClient.GetBlobClient(fileName);

// Open the file and upload its data

FileStream uploadFileStream = File.OpenRead(localFilePath);

await blobClient.UploadAsync(uploadFileStream, true);

uploadFileStream.Close();

return Encoding.ASCII.GetBytes(blobClient.Uri.ToString());

~~~

Step V: The result is uploaded in storage table using UploadExperimentResult in the

AzureBlobOperation.cs illustrated below.

class

~~~C#

public async Task UploadExperimentResult(ExperimentResult result)

{

if (result == null)

{

throw new ObjectShouldNotBeNUllException("Result object cannot be null");

}

CloudTable table =

await AzureTableOperations.CreateTableAsync(config.ResultTable,

config.StorageConnectionStringCosmosTable);

await AzureTableOperations.InsertOrMergeEntityAsync(table, result);

}

~~~

A table is created using the storage connection string with method CreateTableAsync

in the class AzureTableOperations and the results are either inserted if similar entity is not

present in the table or merged if it is. That is done by InsertOrMergeEntityAsync method of

class AzureTableOperations.cs.

Step VI: The message of the queue is deleted. The cancellation token and the message is passed in the method

DeleteMessageAsync. This is to observe the cancellation token which the delete is taking place.

If the cancel token signals to stop the process the delete will not occur.

## How to run experiment

Step1: Create two blob containers one is input-files and second one is result-files. Upload the below json file

in the input-files contaniner

~~~json

[

{

"Alpha": 1.0,

"BucketIndex": 4,

"ActualValueInBucket": 44.0,

"InputFromTemporalMemory": [1,3,6,7,8,9,0]

},

{

"Alpha": 1.0,

"BucketIndex": 5,

"ActualValueInBucket": 55.0,

"InputFromTemporalMemory": [1,2,3,4,8,9,0]

},

{

"Alpha": 1.0,

"BucketIndex": 6,

"ActualValueInBucket": 66.0,

"InputFromTemporalMemory": [2,5,6,7,8,9,0]

}

]

~~~

Step2: Update the connection strings in the file appsettings.json.

~~~json

"MyConfig": {

"GroupId": "groupa2020",

HERE --> "StorageConnectionString": "DefaultEndpointsProtocol=https;AccountName=cloudcomputingstorage20;AccountKey=GwCc6mZi05JDNwsNfE7+sbgNc1q2egwvoqgQj7DIa4OLsa4bQ8AKRxft0f6x9Re1pRm5iJfnJ+U+T1BeUV3hmg==;EndpointSuffix=core.windows.net",

"TrainingContainer": "training-files",

"ResultContainer": "result-files",

"ResultTable": "results",

"Queue": "trigger-queue",

HERE --> "StorageConnectionStringCosmosTable": "DefaultEndpointsProtocol=https;AccountName=group2020azurecosmosdb;AccountKey=uGrrCIJ5yqYKSjGr8bs4QPChRFKp23XzRZi7z7iUf0dNwLVDtygBStsfLTuGxTlZcEH7V6BEpryifNZbjWQIEw==;TableEndpoint=https://group2020azurecosmosdb.table.cosmos.azure.com:443/;",

"LocalPath": "mycloudproject-data"

}

~~~~

Step3: Start the experiment, you will see a queue will be created in the azure with the name as triggered

queue. Copy and paste the below data in the queue message.

NOTE: Please update the url of the InputFile according to your container location

~~~json

{

"ExperimentId" : "123",

"InputFile" : "https://cloudcomputingstorage20.blob.core.windows.net/input-files/InputData.json",

"Name" : "Testing input1.json",

"Description" : "project review"

}

~~~

Step4: You will observe a file will be downloaded and experiment will run and the respective results

are uploaded on azure blob container and table

## Results Validation

Its very easy to validate the results, just check on azure blob container names result-files ypu should see

the result file. Similarly you can see the result file in the azure table storage. latest output file

can be found by using the timestamp. That is latest file will be having the most recent timestamp.

## References

[1] â€œImplementation of SDR Classifierâ€, [Online]. Available:

https://github.com/UniversityOfAppliedSciencesFrankfurt/se- cloud-2019- 2020/blob/GroupA2020/Source/HTM/GroupA2020Documentation AndVideo/SDR\_Classifier\_Research\_Paper.pdf