ASHESI UNIVERSITY COLLEGE

USING NEURAL NETWORK IN CONTEXT AWARENESS TO IMPROVE EMAIL MANAGEMENT

UNDERGRADUATE APPLIED PROJECT

B. Sc. Computer Science

David Ngugi Wainaina

April 2016

ASHESI UNIVERSITY COLLEGE

Using Neural Network in Context Awareness to Improve Email Management

APPLIED PROJECT

Applied project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

David Ngugi Wainaina

April 2016

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of thesis laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

ACKNOWLEDGEMENTS

Thanks to God Almighty for his mercies and grace and giving me strength all throughout the execution of this project.

Special thanks to my supervisor Mr. Aelaf Dafla for encouraging me and guiding me through this project.

I also extend my gratitude to my friends at Ashesi who kept me awake while doing this project. Special thanks to Claire Chemutai for encouraging me throughout the project.

Finally, I thank the whole of Ashesi fraternity who made me a family member throughout my stay at Ashesi University College.

Abstract

This paper details a context awareness application which can be used in personal computers to enhance user experience. Users are distracted frequently and lose focus because there is so much information accessible through the computer. The users have to sift through for the information they really needed at that moment in time.

Contexto application address this problem by analysing the information which comes to the system and decide whether it falls in the current user's current context or not. It applies a neural network which uses sum of the weights assigned to each input to make the decision if a given threshold.

The application addresses the problem by filtering emails received in outlook mail client. Each email which comes to the inbox will be analysed to determine which context it falls in and it is flagged for that context. The user can now read emails intended to the context he/she is working in efficiently without unnecessary disturbance. This is done by analysing the email's sender and subject. The sender and words contained in the subject will have their own weights in a database. Each input's weight is multiplied by the weight of input category; whether sender or subject keywords. The sum is then compared to a threshold and if it is more than the threshold, the algorithm will determine which context the email best falls in.

The experiment demonstrated that it is possible to determine information context using neural network analysis and rearrange it to improve user experience. It is possible to extend the implementation for the email client to file system and browser to make a holistic user experience.

Table of Contents

Table of Figures

Chapter 1: Introduction

Since the world became a global village, connected computers receives a lot of information which might be disruptive to the users. This paper will focus on a solution to reduce the distraction caused by constant flow of information. Email communication is used as an example that the problem can be addressed using context awareness. The solution was implemented and tested using Microsoft outlook software as the email client.

Over the years, scientists have been making efforts to make smart software and devices which will take into consideration user's context. Context is any information characterizing a situation related to the interaction between users, applications and surrounding environment (Musumba & Nyongesa, 2013). In this proposed application, context will refer to what a user will be working on when using their personal computers. Frequently, users get emails which do not correspond to the context they are working on. They resort to ignoring the emails but even this will involve getting to know who sent the email or what the subject entails. Others go to the extent of reading only flagged emails due to bulkiness of emails sent to their inbox. If they are senior people in an organization, they may have personal assistants who will read the emails for them and decides how to respond to each one of them. However, this is more expensive.

This application address this issue by making sure users will only have to be alerted or read emails which corresponds to the contexts they are working on at that moment. The inbox is also analyzed and emails are clustered into groups according to their similarity to identify the contexts (states) the user will be.

1.1 Background

Context awareness has been around since it was used by scientist to help users get faster access to information or other useful object in the Olivetti Research Lab in 1994. An experiment done in the Olivetti Research Lab involving development of the Active Badge system which helped to adapt users' applications to people's whereabouts considered location, identities of neighbouring users and objects and changes to those objects as context (Schilit, Adams, & Want, 1994). Another team of scientists defined context as location, identities of neighbouring users, time and environmental characteristics such as season and temperature (Musumba & Nyongesa, 2013). Dey defined context as user's emotional state, focus of attention, location and orientation (Musumba & Nyongesa, 2013). It is evident all definitions aims at making sure our software are aware of what surrounds us and adapts accordingly.

Computer users gets a lot of information which are unstructured and not relevant to their context. From a research done on a company it was established that 43% of emails dealt with were non-business related (Jackson, Dawson, & Wilson, 2001). It took the average employee an average of 1 minute 44 seconds to react to a new email notification by opening the email application (Jackson, Dawson, & Wilson, 2001).After reacting to the emails, users spent an average of 64 seconds to recover before they returned to their work (Jackson, Dawson, & Wilson, 2001). When users are interrupted by emails when working on a task, the task will take one third longer of the time it could have taken to be completed without interruptions (Jackson, Dawson, & Wilson, 2001). Given employees receives a new email every five minutes, they could end up receiving 96 email interruptions in an 8 hours working day (Marulanda-Carter & Jackson, 2012). Taking into account it takes an average of 1 minute 44 seconds to react to a new email and 43% of the emails are actually not work related, there will be a lot of time used for non-productive matters.

To handle this problem of interruptions, many email users in a workplace tends to only read emails which are coming from their bosses or are personally directed to them. Some even responds to emails which are only flagged as highly important while some will opt to have a personal assistant read the emails for them and decide the responses. This is an ineffective way of dealing with emails and this paper proposes Contexto application which will group emails according to user's context corresponding to the tasks the user gets engaged with. Emails which falls in the current context will be allowed to show notifications and prioritised over other emails. This will reduce the amount of email interruptions which occurs when users are working on a specific important task thus improving effectiveness and productivity.

To determine the context of an email, an algorithm which implements neural network is used. The email sender and subject are used as inputs and are given weights according to their importance to each context in the system. When an email arrives, the context of that email is derived by analysing the sender weight and subject keywords weights. If the context of that email corresponds to the user's current context, then the user will receive the notification and it will show in the inbox. Emails which don't fall in the context will not produce alerts and will wait for user attention when the user decides to read other emails apart from the one in their context at their selected convenient time.

1.2 Objectives

- To create a system which will help users in managing emails and other information according to their context.
- To enable user create new contexts.
- To classify emails according to user contexts.
- To reduce time used in reading irrelevant emails depending on the task at hand.
- To cluster emails into similar email groups which can be used to define context.

Chapter 2: Design

2.1 Scenarios

To understand what users need when making this application, consider the following scenario.

2.1.1 Faculty Email User

A Computer Science faculty at Ashesi University is preparing a proposal which will be presented to the executive board. The presentation will be in 20 minutes time and he is doing finishing touches and rehearsing on the presentation. He sets his context to proposal presentation on his outlook mail client. He is also waiting for an email from a fellow faculty which will have images to be used in the presentation. He remembers an exchange of emails he had with a consultant on the proposal and would want to reference it. He opens the outlook email client and the outlook explorer is showing emails from the current context folder which pertains this proposal. He easily retrieves the email and references it on his presentation. While still working on the presentation, he receives the email he was waiting for and it displays a notification window. He opens it and quickly adds the images on his presentation. Another email arrives and displays a notification window. He opens it and realizes it does not belong to the current context. He moves it to another context folder and continues working. Though he usually gets many emails, in this period, he only receives two emails which were perceived to belong in the context. He is able to complete the preparations and presents to the executive board.

2.2 User Cases

The above scenario provides user requirements which are implemented in the application. The user may decide to change the current context or create a new one. Once the context is set, outlook client will show emails from the set context folder. New emails will only show notification if they belong to that context. If an email which arrives is marked

to be in the current context but the user decides otherwise, moving the email will also change its context. User may also select to view the inbox in the automatic clusters created by the application.

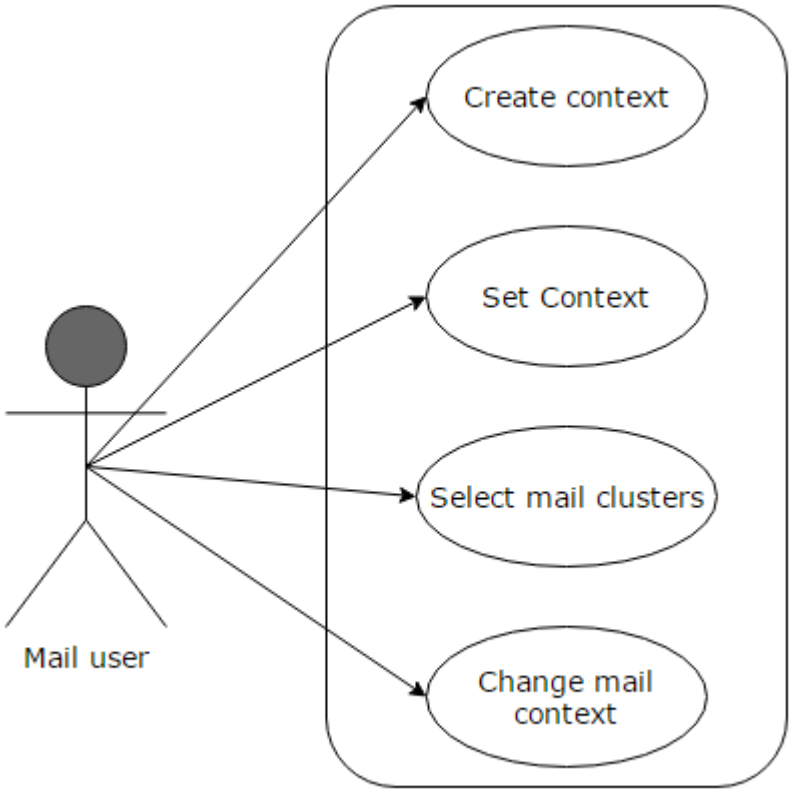

Figure 2.1 Use case diagram of a mail user

2.3 Functional and Non-Functional Requirements

The application's functional and non-functional requirements are developed from the above use case. Figure 2.2 shows an activity diagram for this application. The activity diagram starts when opening the outlook email client application and the three different flow in the activity diagram happens independent of one another. This activity diagram also gives more insight when formulating the functional and non-functional requirements.

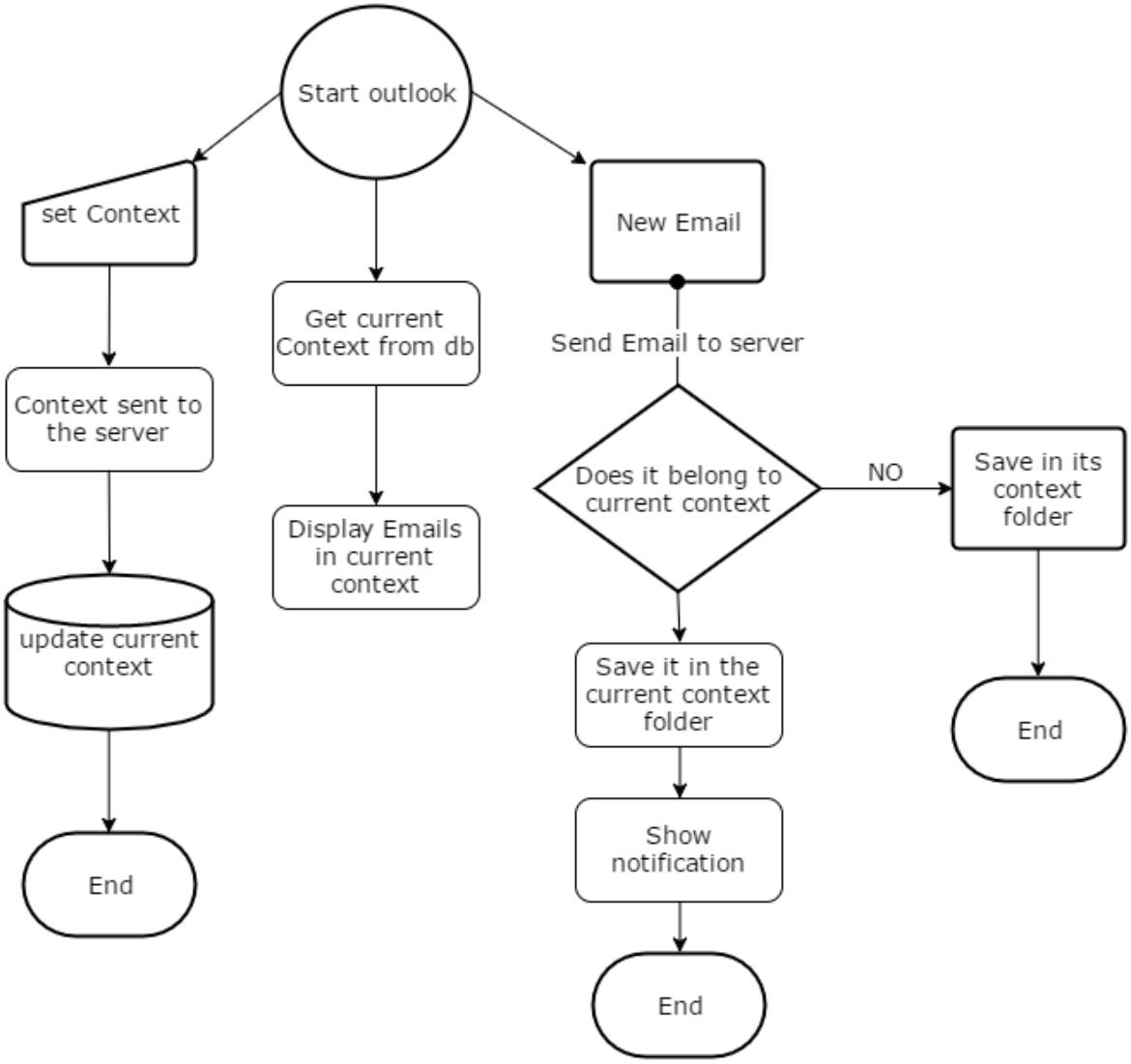

Figure 2.2 Contexto Application Flowchart

2.3.1 Non Functional Requirements

The application was developed with the following non-functional requirement:

- The system should put an email object in its context with an accuracy of over 70%.
- The outlook email client add-in should not be bulky to affect its start-up time.
- It should allow updates without losing data.
- The add-in should be installed by downloading from online portal or an application setup.

2.3.2 Functional Requirements

The implementation was based on these requirements:

- Users should be able to create new contexts.
- Users should be able to set their contexts.
- User's email client current explorer should only display emails belonging to set context.
- Users should be able to use the created automatic email clusters.
- Users should be able to move emails from one context to another.
- Users should only get notified when an email belonging to set context arrives.

2.4 System Architecture

Figure 2.4 below is the system architecture of the Contexto application system. The diagram shows the major components; outlook email client add-in, server side application and the database component. The outlook email client add-in handles the user interface for setting context and also communicates with the server side application. It sends data to the server for processing and gets back the results. The results are used to decide where email objects falls in the contexts or clusters on the email client.

The server side application interacts with the add-in. It adds new contexts to the database and sets the current context in the database. This component is also responsible for computing the context of an email object depending on the stored email keywords weights in the database. It also calculates email clusters for the user and presents them to the email client add-in.

The database stores all the context information and maps the important keywords with all keywords. Decision regarding which context an email object falls in depends on the weights of keywords and sender with respect to contexts stored in the database.

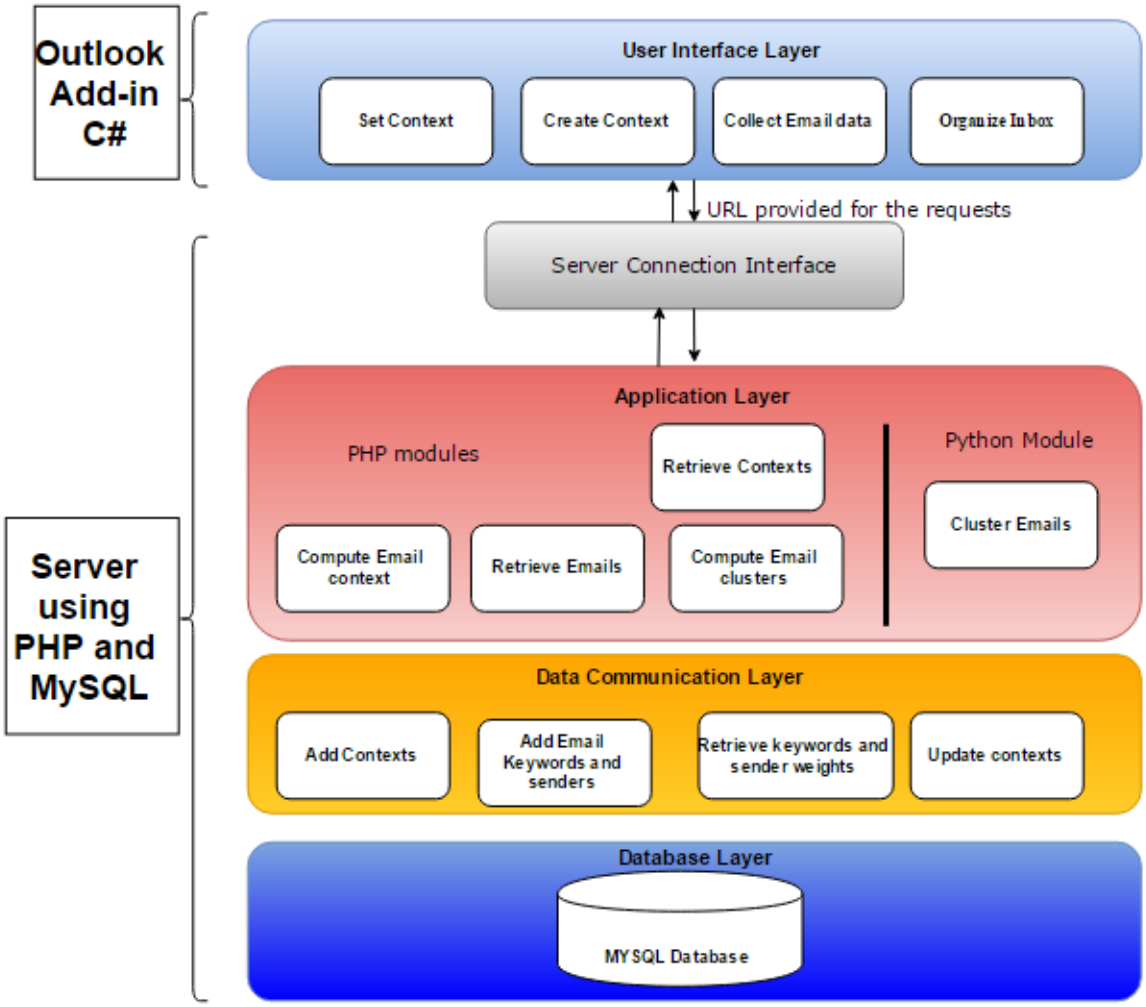

Figure 2.3 System Architecture

2.5 Database Architecture

The database plays an important role in classifying emails in their context. The tables involved are; mail_keywords which stores all the email subject keywords, context_table which stores all the user contexts, mail_sender which holds all the email senders, keyword_context which maps the keywords with contexts and their weights and sender_context which maps senders with contexts and their weights.

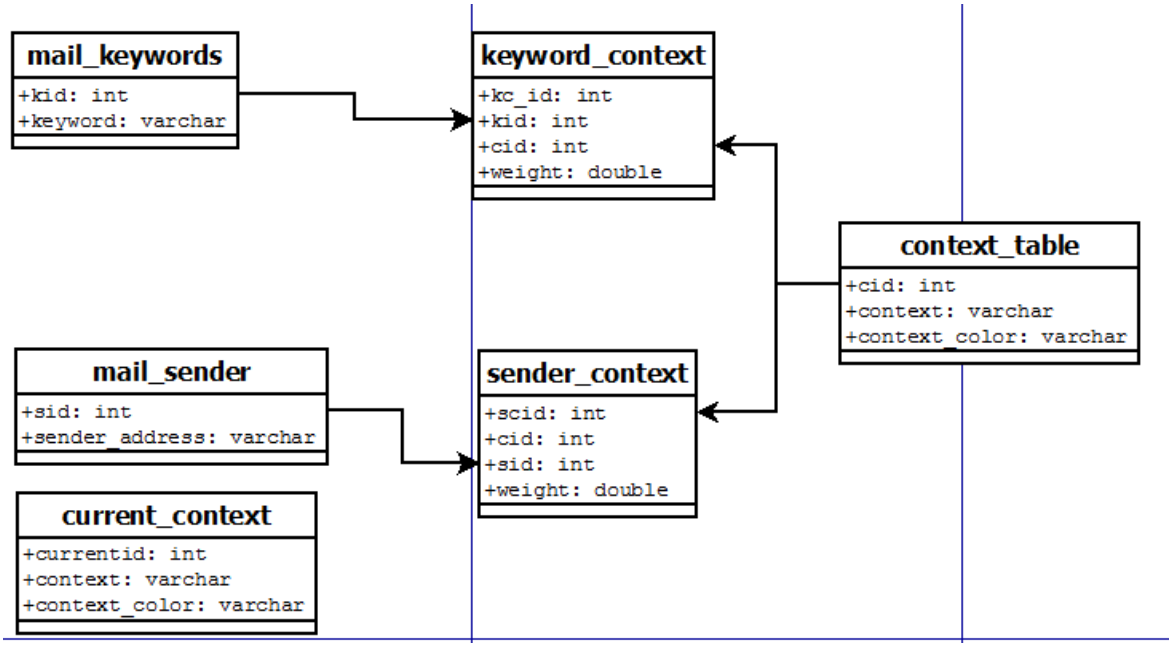

Figure 2.4 Database Architecture

Chapter 3: Implementation

3.1 Implementation Overview

This system comprises of three components; outlook email client add-in, server side of the system and the application database. The outlook email client add-in component help users create new context and set working context. When a new context is created, the information is sent to the server and inserted into the application database. Users also use the interface to set working context which is immediately set in the database. It also helps in collecting data from the email client. Meta data of new emails and existing emails is sent to the server for processing by this component. It gets back the output decisions from the server and sets the email inbox accordingly. Since one of the main aim is to avoid distractions, it automatically disables notifications and only allows them when incoming emails belongs to the set context. It is responsible for assigning email objects to different categories which represents user contexts. In addition to the email client add-in, the major component of this application is the server side application system. It is responsible for computing the context of each incoming email object and returning the context of the email back to the email client add-in. It also varies the weights of context keywords in the database when users decide an email belongs to a certain context. It communicates with the database for data and computes the results. The final component is the database itself which stores all the context details and maps them with keywords. It does not allow duplicates but allows the weights of keywords to be updated when users decides to put emails in a certain context.

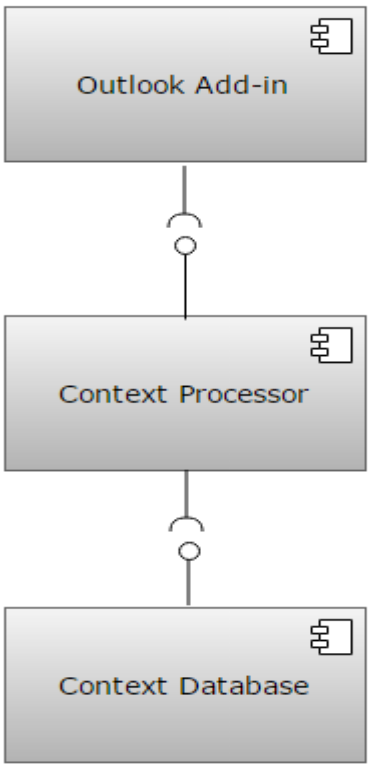

Figure 3.1: Contexto application Component diagram

This application has been developed using different tools. The outlook email client add-in was developed using Microsoft Visual Studio IDE and C# as the language of choice. C# provides a rich library of programming add-ins for MS office applications. JetBrains PhpStorm was used to program the server side of the application using PHP. Python was used to implement clustering using machine learning on the server side. Python provides a rich library of machine learning which made the implementation easier. The server is centralized and can be accessed from anywhere. Users will have their own accounts to ease the computation and provide personalized results.

Contexto application implementation was done in two steps. The first step uses supervised machine learning algorithm and the second step involved non-supervised machine learning algorithm. Supervised learning is used to detect email contexts wile non-supervised learning is used to arrange emails objects into clusters.

3.1.1 Supervised Learning Implementation

For supervised learning, neural networks algorithm is used and relies on a database to get the correct analysis on an email object. When an email context is calculated by the neural network algorithm, it is sent back to the outlook email client and added to that context. In the outlook email client, each context has its own folder and category. Thus if an email is given a certain context, then it will be categorized and moved to that particular context category and folder respectively.

3.1.2 Unsupervised Learning Implementation

Non-supervised learning uses TF-IDF followed by k-means algorithm to cluster the emails into similar clusters. For email clustering, TF-IDF algorithm is used to index words in an email object and develop their vectors representing measure of similarity. These vectors are in turn used in the k-means algorithm to cluster the emails. The clusters are then sent back to outlook mail client as clusters suggestions which a user can activate. This chapter will explain all these implementations in details.

3.2 Outlook Add-in

An outlook add-in is installed in the mail client to act as the gateway between outlook and the application (server) and also provides a user interface. When creating a new context, a user will have to enter the name of context and choose the desired colour for it. The name is used to create a context folder and category which bears the context name as label and chosen colour as the category colour. The context name and colour are also sent to the server which stores it in the database. When choosing the context, a drop down is provided which enumerates the context which are stored in the database. When a user selects one of the context, the mail outlook displays the emails context folder containing context emails. A button for context settings is also provided among the drop down. When clicked, it will provide a form which gives access to all contexts and buttons to create new context,

rename context, change the colour of context and delete context. New clusters are selected from the same drop down menu and they will be created automatically.

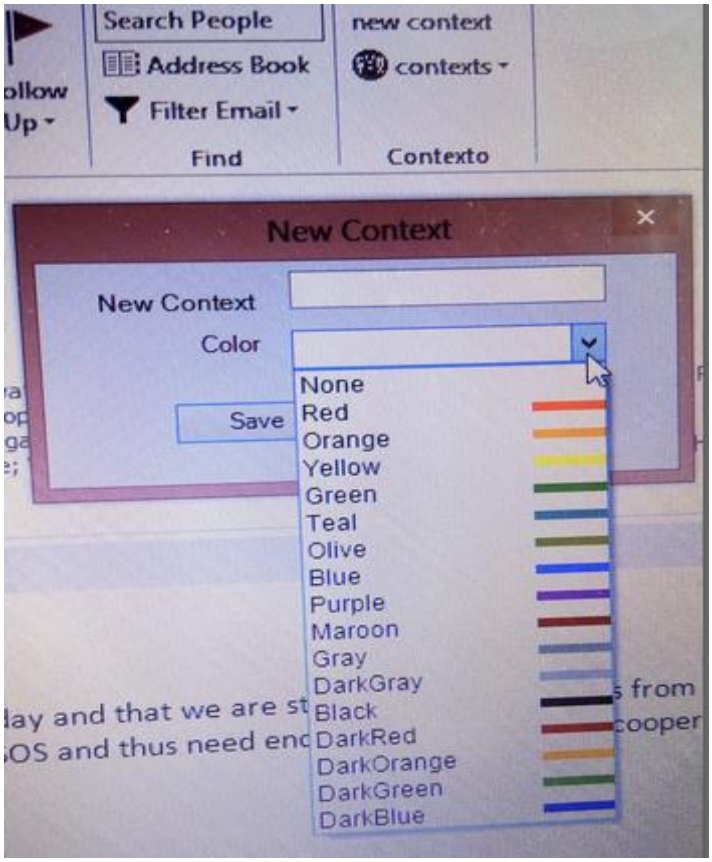

Figure 3.2 Create new context

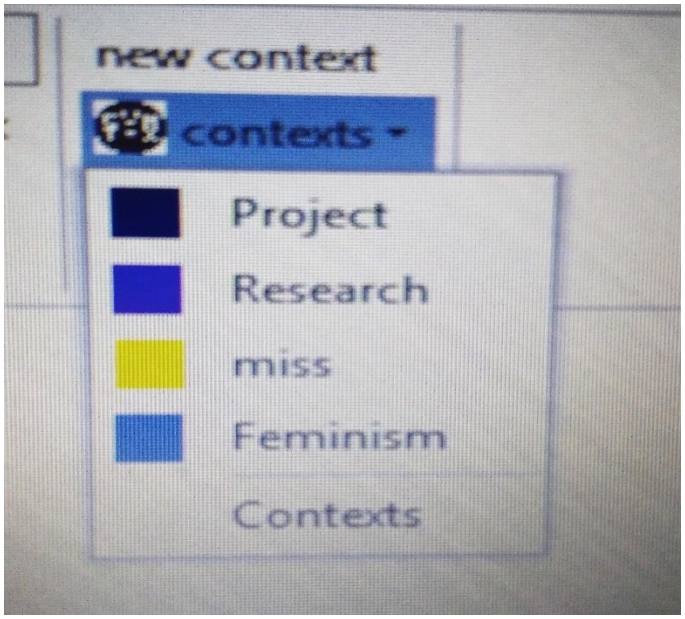

Figure 3.3 Select current context

3.2.1 Creating the Add-in

The add-in was created using C# programming language in the .NET framework. C# provides a rich library of programming Microsoft office products. Using Visual Studio, a new add-in is created by choosing the office product of interest and the programming language of choice; between C# and VB.

The add-in is implemented as a home ribbon group item in the outlook email client. To position it in the appropriate ribbon and location, the ribbon unique identification is needed. This is used to identify an outlook item from all other items and is communicated to the outlook client during installation. A unique identification for the group neighbouring the application add-in also required to specify this location. These identities are provided by Microsoft for office developers. For this application, TabMail was the identifier need to position the add-in.

In the add-in interface provided, the ‘new context’ input is a button which when clicked opens a new windows form shown in figure 3.1. The form provides a textbox for new context name and a dropdown containing possible outlook category colours. When a user inputs a name and choses the context colour and clicks the save button, the add-in fetches the name from the textbox and colour from the dropdown.

The add-in interface also provides a dropdown which enumerates user contexts fetched from the database. To choose the current context, a user will have to click on one of the contexts in the dropdown which in turn opens the context folder. Newly created contexts can also be viewed from this dropdown.

3.3 Email Processing

The outlook add-in provides an event handler which is activated when a new email arrives. When an email arrives in the outlook mail client its subject, sender and body are collected by the new email event handler.

```
  IncomingMail = (Outlook.MailItem)Item;
  var client = new WebClient();
  var subject = IncomingMail.Subject;
  var mail_sender = GetSenderSMTPAddress(IncomingMail);
  var url =
"http://localhost:8000/server/Contexto/ContextRequests.php?cmd=4&m
ailSubject=" + subject + "&mailSender=" + mail_sender;
 string requestResult = client.DownloadString(url);
 if (!requestResult.Equals("None"))
  {
   IncomingMail.Categories = requestResult;

Move_NewMail_To_ContextFolder(requestResult, IncomingMail);

  }
```

A snippet of algorithm used to fetch email metadata and decide next step

A connection is set up with the server and email parameters are sent for context determination using neural networks. When the context is determined, it is returned to the Contexto add-in which determines the next best step. Since context is implemented as an outlook category model, the mail will be assigned that context category and also moved to the context folder. If that is the set current context, a desktop notification will be shown. However, if the email does not fall in the current set context, the notification does not show up since it is not in the display folder.

Outlook displays email notification if it is delivered in the current folder display. Thus, when a context is set, all emails which belongs to that context will be delivered in the folder which is in display and a notification will appear. Other emails will be moved to their context folder and those who don't belong to any context will be temporarily moved to a temporary folder and immediately returned to the inbox folder. Since emails immediately moved from inbox do not show notification, the user will not be disrupted by emails which do not belong to set current context.

For email clustering, the subject, sender and mail body are sent to the server periodically and clustered according to similarity. In this case, the email body provides more information about email object similarity which are not stored somewhere as in the case of context analysis. When the clusters are returned, they are added in the dropdown for selection. They only get activated when the user explicitly clicks on them. If a selected cluster did not exist prior to the selection, a new mail category is created and assigned the name of the cluster and all the emails belonging to that cluster shows up in the inbox panel. If the cluster already exists as a category, the selection results to filtering of inbox to that category.

3.4 Mail Analysis Using Neural Networks

Neural networks algorithm is used to determine the context of an email object. Since it depends on learning, a database is implemented to store the learnt knowledge. PHP script is used for this implementation and processing is done in the server. There are tables which are used to store all the email senders and subject keywords. These data have unique constraint thus do not duplicate in the database tables.

Each of the sender addresses and keywords are initially matched with all the available contexts and assigned a weight of 0.5. The sender_context from figure 2.5 indicates that, each context id is matched with each sender id and a weight associated to this match. Also, from the keyword_context table, each keyword id is matched with each context id and a weight is set for this association. The mail keywords and mail senders are not repeated in their tables.

If a user moves an email to a certain context folder, its weights for sender and subject keywords are increased in the database to match that shift. If the email was in another context folder, the subject keywords and email sender weights for that context will be reduced.

The formula below describes how a context is determined for a particular email object:

$$\text{C(m)} = f(maxW) = \begin{cases} 1, & maxW > \alpha \\ 0, & maxW < \alpha \end{cases}$$

$$\text{maxW} = \max(W_{sci} * W_{hs}) + \left(\left(\frac{1}{n} = \sum_{j=1}^{j=n} W_{kjci} \right) * W_{hk} \right)$$

C(m) represents the context of which the maximum weight **maxW** belongs to. This is only activated if maxW is greater than threshold α.

$\boldsymbol{W_{sci}}$ Represents the weight of sender in context **ci** where **i** is the context number.

$\boldsymbol{W_{hs}}$ Represents the weight of hidden layer for email senders.

$\frac{1}{n} = \sum_{j=1}^{j=n} W_{kjci}$ This formula gets the average weight of j subject keywords for context **ci**.

W_{hk} This represents the weight of hidden layer for email keywords.

The first part calculates the sender weights. This involves getting the sender id and looking for it in the sender_context table. Since each sender is matched with all the available contexts and is assigned weight for each of them, all weights for each context are taken. The second part involves getting the weights for each keywords in the email subject. Since each keyword is matched with every context and is assigned weights, the algorithm gets the average weight of all keywords for each context. The third stage involves weighing the data at a higher decision level, the sender weights for each context are multiplied by email sender general weight while the keywords weights for each context are multiplied by email subject general weights. The two resulting weights are added for each context and the maximum of this addition is taken. If the maximum weight is more than a set threshold, then the email will be assigned to that context. If the maximum weight is less than the threshold, then the email context for that email object cannot be determined. Upon determining the maximum weight, the context is sent back to the outlook email client and email is assigned to that context category.

The following steps will happen when an email comes:

i. Get email metadata
    a. Email: Sender -> dwainaina3@gmail.com
    b. Subject: Project submission

ii. Lookup for sender in the sender_context and get the weight to each context

iii. Lookup for keywords project and submission from the keyword_context and get the weights for each context

iv. Calculate the email context as shown below:

Table 3.1 Email weights

Input	Context	Weight
dwainaina3@gmail.com	updates	0.5
dwainaina3@gmail.com	project	0.7
Project	updates	0.3
Project	project	0.8
Submission	updates	0.6
Submission	project	0.6

C = sender weight * email sender weight + (kw1+kw2)/2 * email keyword weight

The Email belongs to context with greatest weighted value

Updates = 0.5 * 0.6 + (0.3+0.6)/2 * 0.4 = 0.336

Project = 0.7 * 0.6 + (0.8+0.6)/2 * 0.4 = 0.7

Therefore, the email belongs to project context.

3.5 Mail Clustering Using K-Means

For clustering, all emails senders, subjects and bodies are sent to the server. This part is implemented in python since it provides rich library for this un-supervised learning

algorithm. Upon receiving the mail objects metadata, they are processed using TF-IDF algorithm to get their similarity in vectors. The vectors are then used when processing the number of clusters for the email inbox. When the clusters are developed, the data is sent back to outlook client where they are temporarily added to a dropdown of contexts. They only get activated when a user clicks on them.

Chapter 4: Testing

Testing is an essential part in software engineering. This application underwent a number of testing which will be discussed in this chapter.

4.1 Unit Testing

For unit testing, database connection and data manipulation in the database are done. To test application's connection to the database, PHPUnit test is used. The code below was used and the result is given.

```
include_once(dirname(__FILE__)."\..\base.php");
class baseTest extends PHPUnit_Framework_TestCase
{
    /*
     * Function to test database connection
     */
    public function testDatabaseConnection()
    {
        $connection = new base();

        $this->assertTrue($connection->connect());
    }

}


The result of database connection PHPUnit test is given below:

C:\xampp\htdocs\server\Contexto>phpunit Tests/baseTest.php
PHPUnit 3.7.21 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 1.75Mb

OK (1 test, 1 assertion)
```

Figure 4.1 Database Connection Test

Figure 4.2 shows unit testing on inserting contexts into the database. The result is also given. If insertion is successful, the insert function is expected to return true.

PHPUnit test to check whether data is retrieved from the database was also done. The snippet below shows how it was implemented and the results. The function which

retrieves data from the database is expected to return true when it is successful. Figure 4.3 gives the details.

```
include_once(dirname(__FILE__)."\..\Contexts.php");
class ContextsTest extends PHPUnit_Framework_TestCase
{

    /*
   * Function checks whether an insertion in the database is successful
   */

    public function testDatabaseInsertion()
    {
        $contexts = new Contexts();

        $this->assertTrue($contexts->insertContext("UnitTest","Red"));
    }
}
```

The result for this PHPUnit Test is given below:

C:\xampp\htdocs\server\Contexto>phpunit Tests/ContextsTest.php

PHPUnit 3.7.21 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 1.75Mb

OK (1 test, 1 assertion)

Figure 4.2 Database Insertion Test

The code to be tested:

```
/*
 * This function gets all the contexts from context table
 * @returns boolean Returns true if successful and false otherwise
 */

public function getAllContexts()
{
    $sql = "select * from context_table";

    return $this->query($sql);
}
```

Unit Testing Code:

```
include_once(dirname(__FILE__)."\..\Contexts.php");
class ContextsTest extends PHPUnit_Framework_TestCase
{

    public function testGetAllContexts()
    {
        $allContexts = new Contexts();

        $this->assertTrue($allContexts->getAllContexts());
    }
}
```

Figure 4.3 Data retrieval from Database (a)

```
C:\xampp\htdocs\server\Contexto>phpunit Tests/ContextsTest.php

PHPUnit 3.7.21 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 1.75Mb

OK (1 test, 1 assertion)
```

Figure 4.3 Data retrieval from the Database (b)

4.2 Component Testing

This testing was done on two components, outlook add-in component and context processor component. The context processor component which analyses emails and decide

their context is implemented in PHP. The outlook add-in component is responsible for the user interactions and is implemented in C#.

4.2.1 Testing Context Analysis component

Context analysis component is tested by sending a request to the server and providing parameters. To test if a request to create a new context is executed, the Url below was run on and the result are shown below in figure 4.4. When the same request to create a new context is done again, it is rejected and result differs as shown in figure 4.5.

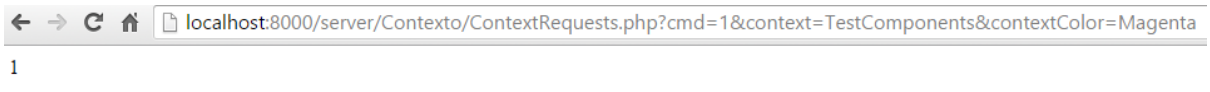

Figure 4.4 New context Test a) Success

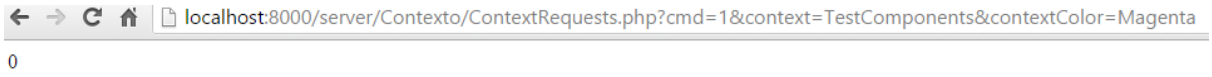

Figure 4.5 New context Test b) Failure

This test verifies that, the request to add works and context ‘TestComponents’ is added only once in the database. The color of a context is also unique and thus an entry of a color which is already used will not be allowed as shown from the below test.

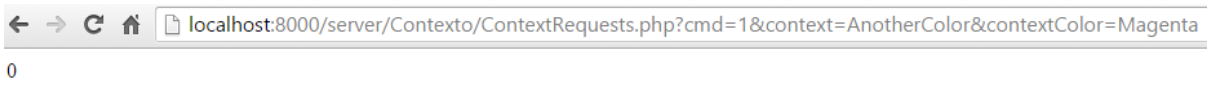

Figure 4.6 Adding an existing context

The resulting contexts table in the database will have the values shown below.

cid	context	context_color
2	Project	DarkBlue
3	Research	BlueViolet
5	miss	Yellow
7	Feminism	CornflowerBlue
8	UnitTest	Red
24	TestComponents	Magenta

Figure 4.7 Database content After Test

The resulting contexts can also be tested when retrieved from the database. A request to get all the context in the database is expected to return a json object containing all the contexts. JSONLint is used to test whether this the result is in correct json format.

Figure 4.8 All Contexts Request

After getting all the context from the above request, the output is entered into a json validator engine and the result is shown below. This service is provided online from JSONLint website; http://pro.jsonlint.com/

```
{
    "result": 1,
    "contexts": [
        {
            "cid": "2",
            "context": "Project",
            "context_color": "DarkBlue"
        },
        {
            "cid": "3",
            "context": "Research",
            "context_color": "BlueViolet"
        },
        {
            "cid": "5",
            "context": "miss",
            "context_color": "Yellow"
        },
        {
            "cid": "7",
            "context": "Feminism",
            "context_color": "CornflowerBlue"
        },
        {
            "cid": "8",
            "context": "UnitTest",
            "context_color": "Red"
        },
        {
            "cid": "24",
            "context": "TestComponents",
            "context_color": "Magenta"
        }
    ]
}
```


Figure 4.9 json validation

The major functionality in this application is also tested. The application's ability to get the context of an email object when provided with the email sender and email subject.

Figure 4.10 Email context Analyzed

As show above, an email with subject: *Project is presentation* and sender: *dwainaina3@gmail.com* is analyzed to belong in Project context.

4.2.2 Email Clustering Testing

Email clustering using K-Means algorithm did not give expected results. When clustered, similarity was mainly based on email salutation and signature. Therefore, content did not have a great influence on clustering. Moreover, email senders address are split wherever a full stop and '@' appears. This implies that, an address, for example, david.wainaina@gmail.com, will result to 'david', 'Wainaina', 'gmail and 'com'. Since many email sender's address will have institutions or email service providers in their addresses, it the clustering algorithm splits them and recognizes them as similar.

{"emailArray":{"emailClusters":[2 , 0 , 0 , 0 , 0 , 0 , 4 , 3 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 2 , 0 , 0 , 0 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 4 , 4 , 2 , 2 , 0 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 4 , 2 , 4 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 0 , 0 , 0 , 0 , 2 , 0 , 2 , 4 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 0 , 4 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 0 , 0 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 4 , 2 , 2 , 2 , 0 , 4 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 0 , 2 , 2 , 0 , 2 , 0 , 2 , 2 , 0 , 2 , 0 , 2 , 0 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 2 , 3 , 2 , 2 , 2 , 4 , 0 , 2 , 2 , 0 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 3 , 4 , 2 , 2 , 4 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 0 , 0 , 2 , 2 , 4 , 2 , 3 , 2 , 2 , 2 , 2 , 4 , 3 , 2 , 2 , 4 , 0 , 4 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 1 , 1 , 2 , 2 , 0 , 2 , 0 , 0 , 0 , 2 , 4 , 4 , 0 , 0 , 2 , 4 , 2 , 2 , 2 , 2 , 2 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 4 , 2 , 0 , 4 , 2 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 4 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 0 , 3 , 2 , 2 , 2 , 2 , 4 , 4 , 2 , 2 , 2 , 3 , 4 , 4 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 3 , 2 , 3 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 3 , 2 , 0 , 2 , 0 , 2 , 2 , 0 , 2 , 2 , 2 , 0 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 4 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 1 , 1 , 1 , 1 , 1 , 1 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 1 , 2 , 0 , 0 , 0 , 0 , 1 , 1 , 1 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 3 , 2 , 3 , 2 , 0 , 2 , 0 , 2 , 2 , 2 , 3 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 4 , 0 , 0 , 2 , 0 , 0 , 0 , 0 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 2 , 2 , 2 , 2 , 2 , 0 , 2 , 0 , 2 , 2 , 2 , 2] }, "clusters":[{" Cluster 0 ":[" com " , " gmail " , " dwainaina3 " , " onmicrosoft " , " aucampus "]} , {" Cluster 1 ":[" adafla " , " project " , " meeting " , " edu " , " gh "]} , {" Cluster 2 ":[" ashesi " , " edu " , " gh " , " election " , " alartey "]} , {" Cluster 3 ":[" akesse " , " shedrach " , " mcf " , " general " , " elections "]} , {" Cluster 4 ":[" nana " , " genfi " , " fw " , " edu " , " gh "]}]}

Figure 4.11 Email Clustering results

Figure 4.11 shows result gotten from email clustering attempt. The email clusters of analyzed emails are shown in the emailClusters array. Each email is directly mapped with its clusters. The derived clusters main keywords are shown in the clusters array. At a closer look, words such as 'com', 'dwainaina3','onmicrosoft' and 'aucampus' are recognized as main keywords for cluster 0. However, these words are email addresses which have broken down into individual words. Therefore, with such basis of clustering,

the algorithm does not give an efficient email cluster. With this establishment, this email clustering feature was not added into the outlook add-in for users.

4.2.3 Outlook Add-In Testing'

The add-in was tested to check whether it can add contexts into the database and change the current outlook explorer view to a selected context.

To test whether a user can create a new context from the outlook mail client using the outlook add-in, the add-in was installed and new context button which is mounted on the ribbon clicked. This resulted to a new form which has entries for new context name and color. On clicking the save button after entering the name and selecting color from the dropdown, the add-in is expected to send this information to the server which will add it into the database.

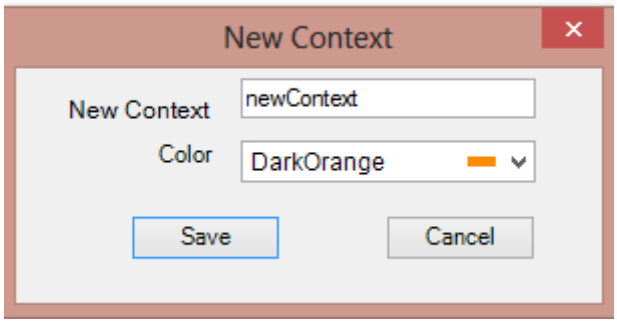

Figure 4.11 New context in outlook

After clicking the save button, the context was verified to have been added into the database as show in the figure below.

cid	context	context_color
2	Project	DarkBlue
3	Research	BlueViolet
5	miss	Yellow
7	Feminism	CornflowerBlue
8	UnitTest	Red
24	TestComponents	Magenta
25	newContext	DarkOrange

Figure 4.12 New context added from add-in

Thus, the test checking whether the outlook add-in can add contexts into the database was successful and concluded that it is possible to create new contexts from the outlook mail client.

All contexts can be viewed from the contexto add-in by clicking on contexts dropdown. Users use this dropdown to select the current context. The figure below show all the contexts created by the user. The context which was newly created is shown as the last item in the list.

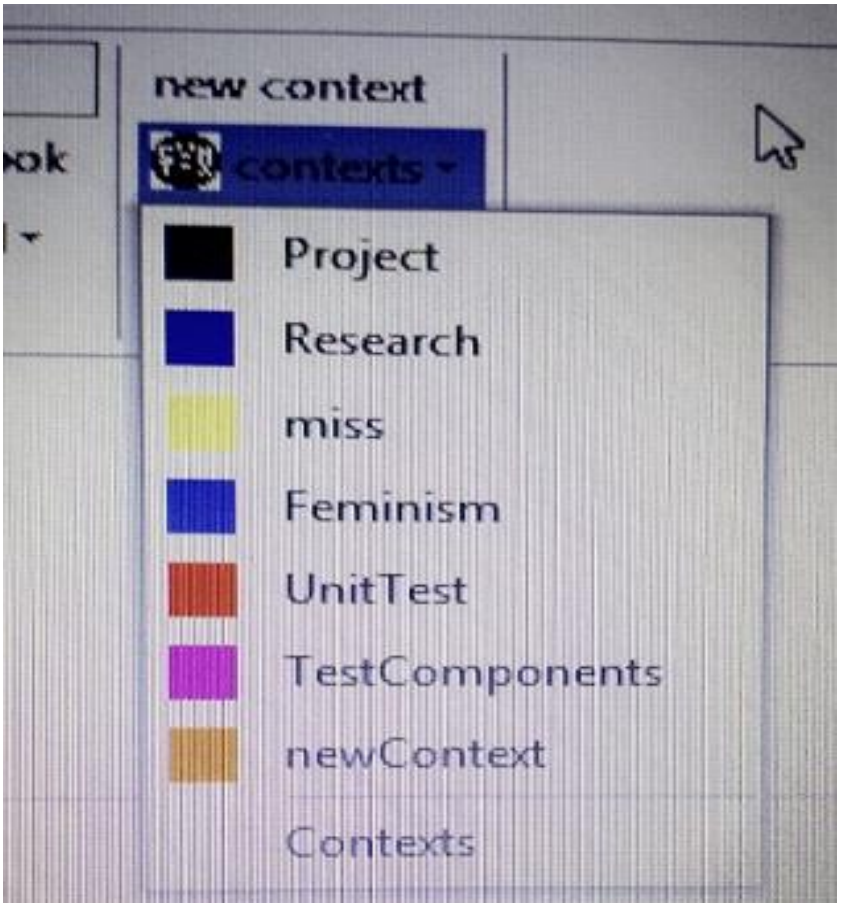

Figure 4.13 All contexts Dropdown from outlook

A user sets the current context by clicking on any of the context in the dropdown. When clicked, it displays the context folder and also updates the current context in the database.

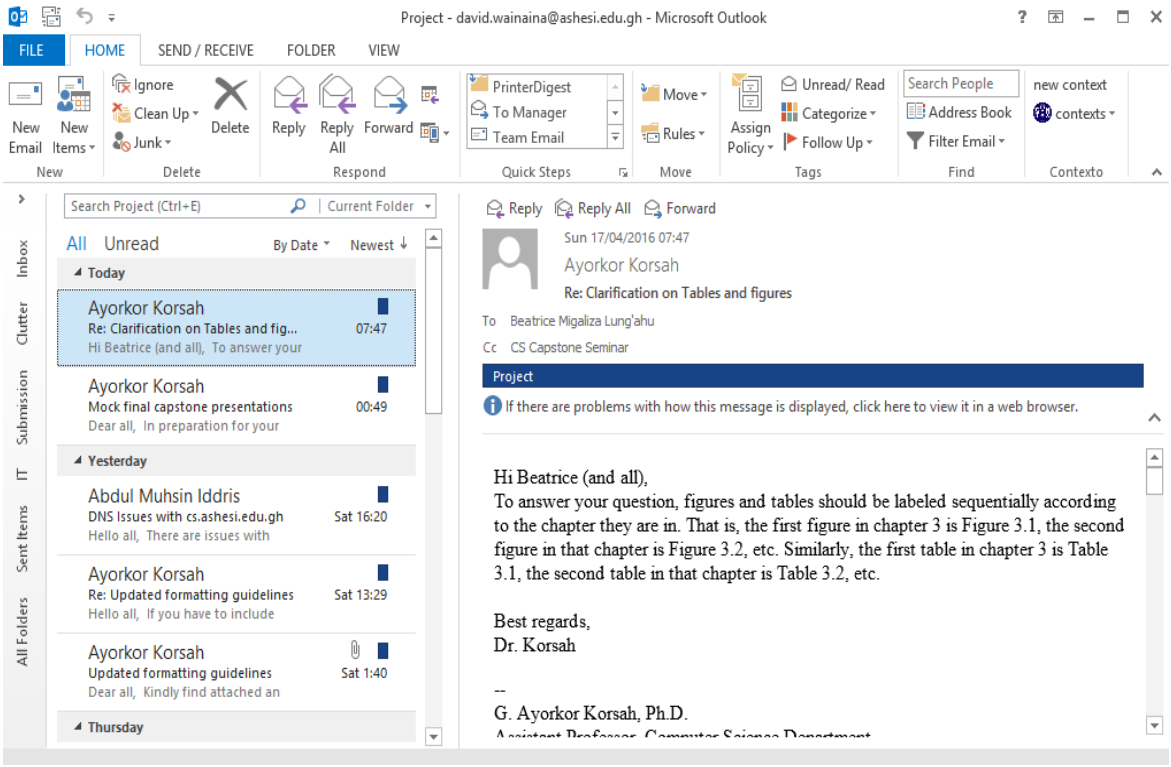

Figure 4.14 Current context emails.

4.3 User Testing

The application was installed in an outlook application for use. When using the application, new emails were categorized in the first context stored in the database. It turns out, when new data comes in, all the context will have the same value of weights for the email. Therefore, when the algorithm is searching for the context with largest weight value, the first context in the database is returned. In this case, a user will have to move emails back to inbox folder if they don't belong to that context. This will result to subsequent similar emails not being categorized in that context.

However, this is a nuisance to users since most emails cannot fall in a particular context. It will be reasonable to train the algorithm about emails which belong to a context by moving them into the context folder rather than training it about emails which do not belong in the context by moving them outside the context folder. To correct this, the algorithm threshold was increased to 0.5. Thus all emails which have the maximum weight

of 0.5 and below for a particular context, will not be categorized in that context. This solved this issue and users will have to train the algorithm about which emails which fall into particular context.

Chapter 5: Challenges, Future Works and Conclusion

5.1 Conclusions

To conclude, this application was developed to aid email users in arranging emails according to the context they are working on. Email users should be able to concentrate their time on emails which are relevant to any task they are working on at any moment in time without much hustle. The application met all but one of its objectives.

Using the application, users can manage their emails into context with ease. After creating contexts, users can train the application to classify the emails into contexts and also helps in minimizing distraction by muting notification for non-current context emails. However, the application could meet the objective of creating email clusters due to choice of algorithm used and other unavoidable factors as explained in the Chapter 4 section 4.2.2.

With good use of this application, users will not spend a lot of time checking emails which do not concern their tasks when working on them. This will help a lot in saving time and accessing only relevant information when working on their current tasks.

5.2 Future Works

Using context awareness can make personal computers more aware of our working dynamics and change according to offer more direct assistance. Another aspect of this system can be used to help users have faster access to files in the file explorer. Depending on the context, the system will predict which files the user will need and upon opening the home file explorer, these files will be presented in the view. This will make it easier for users to access files which are directly linked to the context they are working on.

Another aspect of this system is browsing experience. By customizing user searches according to context, the browser will be able to present the best result depending on the current user context.

These work can be implemented as there are available APIs for customizing file explorer and browsers. Context awareness is the next big thing as we enter the world of Internet of Things.

5.3 Challenges

There was a design decision on how to view emails from a particular folder. I realized that, when showing the emails, I search from the inbox for the emails with category as the current context, the search input box might be closed resulting to loss of the search result view. To fix this, I had to create folders which will hold context emails and show the view of this folder in the outlook explorer when needed. The big question is, should the user be allowed to change the current folder in view without switching context and still enjoy the applications' benefits of putting off email notifications not belonging to current context? If the user is denied this freedom, it will be like a coup d'état on the outlook email client where the user does not have control over his software anymore.

Email clustering is another aspect which could be fully implemented. The clusters which were formed did not reflect the desired effect. Clusters were based on common words such as salutation and email signature. To omit these two aspects of any email is very challenging thus could be implemented.

References

Jackson, T., Dawson, R., & Wilson, D. (2001). The cost of email interruption. *Journal of Systems & Information Technology, 5*, 81 - 92. Retrieved from http://dx.doi.org/10.1108/13287260180000760

Marulanda-Carter, L., & Jackson, T. (2012). Effects of e-mail addiction and interruptions on employees. *Journal of Systems and Information Technology, 14*, 82 - 94. Retrieved from http://dx.doi.org/10.1108/13287261211221146

Musumba, G. W., & Nyongesa, H. O. (2013). Context awareness in mobile computing: A review. *International Journal of Machine Learning and Applications, 2*. Retrieved from http://dx.doi.org/10.4102/ijmla.v2i1.5

Schilit, B. N., Adams, N., & Want, R. (1994). Context-Aware Computing Applications. *IEEE Computer Society, 16*, 85 - 90. Retrieved from http://www.cs.cmu.edu/~anind/courses/ubicomp-f2009/Context-Schilit.pdf