UNIVERSITÀ
degli STUDI
di CATANIA

Department of Electrical, Electronic and Computer Engineering

MASTER'S DEGREE COURSE IN COMPUTER ENGINEERING

Advanced Programming Languages project

Analysis of text categories

Academic Year 2022-2023

Student:

Andrea Paolo Ventimiglia 1000039024

Summary

INTRODUCTION

The **TextAnalyses** application allows users to analyze texts to understand the most frequent words and the gender they belong to. Thanks to the developed interface, users can also compare different texts and obtain statistics.

The architecture of the application includes:

- Two graphical interfaces developed in **C#** with Windows Forms, to interact with the application in an intuitive way.
- A Python **Back-End** with Flask, which exposes REST APIs to receive texts and perform text analysis operations.
- A **Go module** that manages the connection to the MongoDB Atlas database, to save and retrieve the analyzed texts. It also allows additional operations on the data.

The integration of these different components makes it possible to provide a complete application, which from the user interface to the persistence of the data exploits the potential of different languages. The user can easily interact via the graphical client and get advanced text mining results through Python and Go.

1. Front-End (C#)

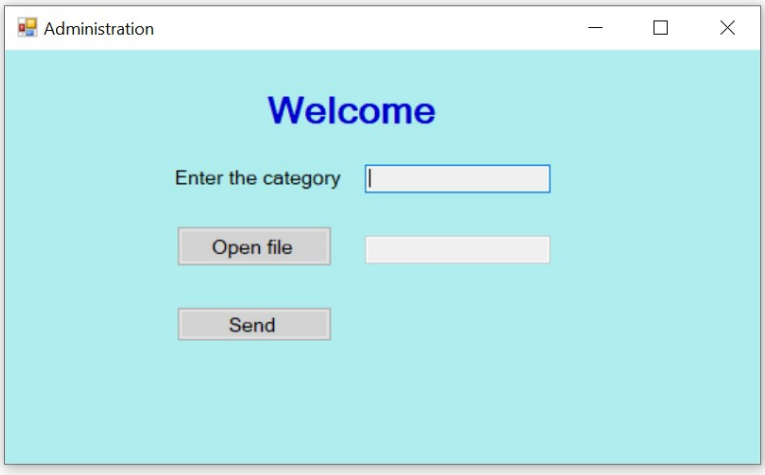
1.1. Introduction

In the **TextAnalyses application**, two Clients have been developed, "Administrator" and "User". The development of both clients is based on event handling; In fact, each user action will correspond to an event that, thanks to the handlers, present in the various forms, will be managed and will allow you to perform a specific action/operation.

1.2. Client "Administrator"

This Client is aimed at inserting a "category" by entering its name and a ".txt" file where the words can be mined.

1.2.1. Interface



1.3. Client "User"

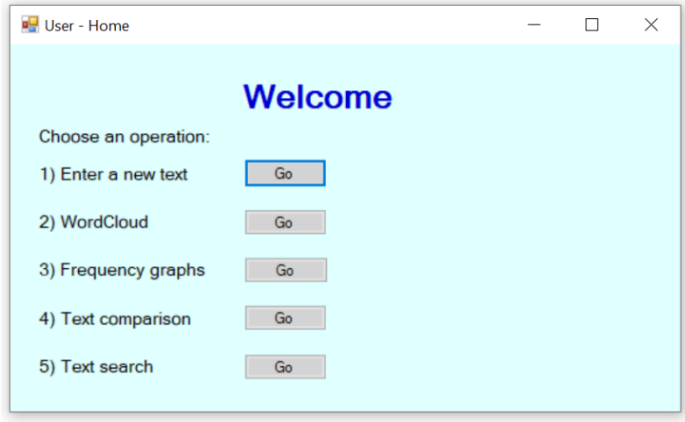
This client allows users to perform various text mining operations in an intuitive way through a graphical interface developed with Windows Forms.

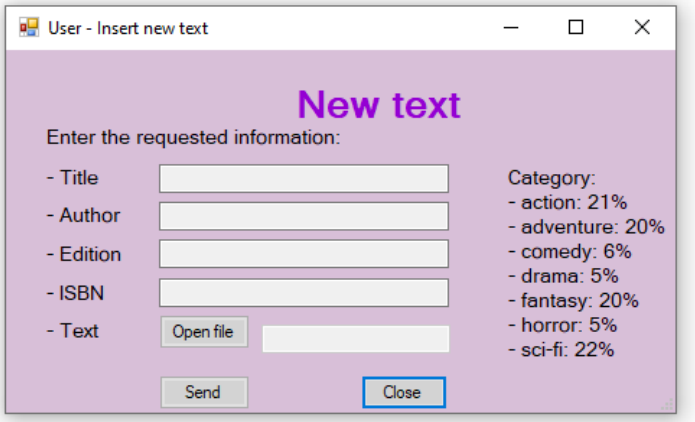
There are several forms for the various functions:

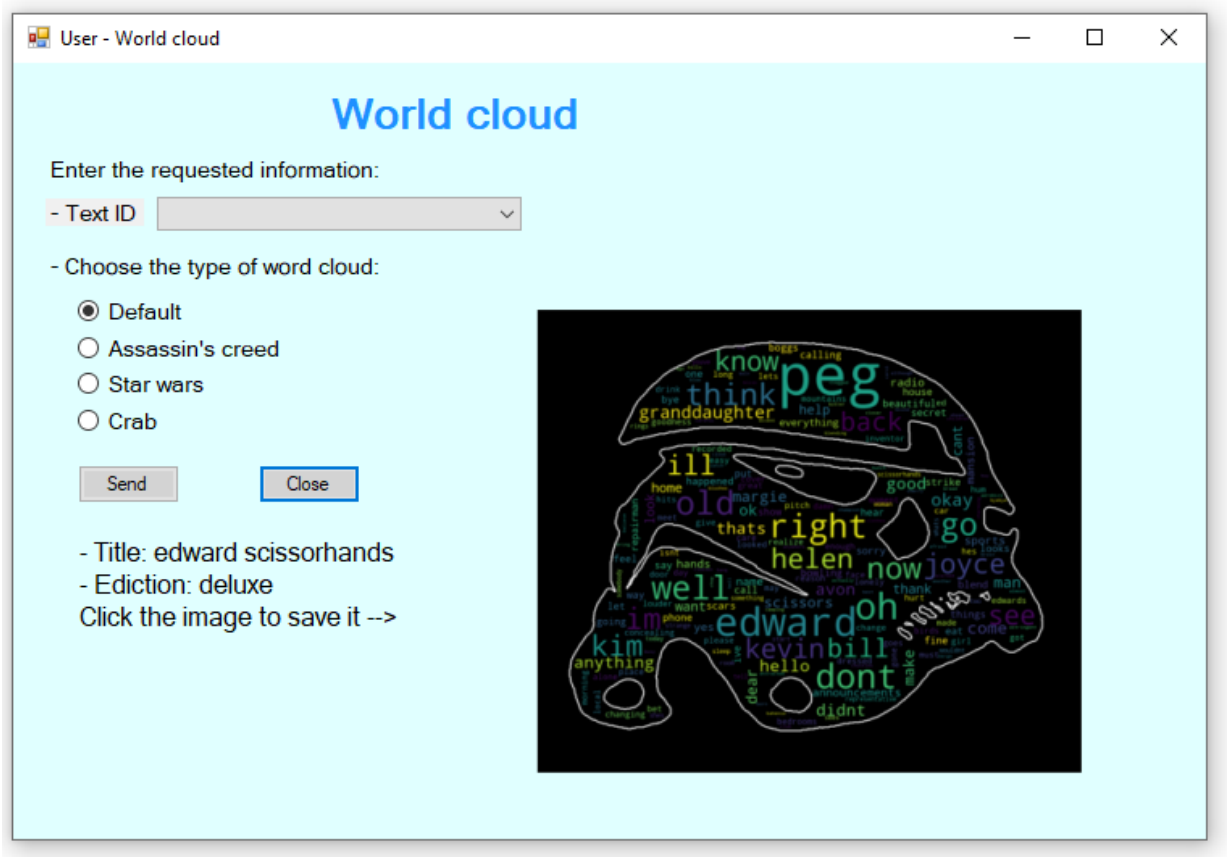
- **HomePage**: allows you to view and select the operations available in the application by means of buttons.
- **Insert New Text**: allows you to insert the metadata of a new text (title, author, etc.) and upload the .txt file containing the text.
- **Word Cloud**: Generates a saveable word cloud image from previously uploaded text.

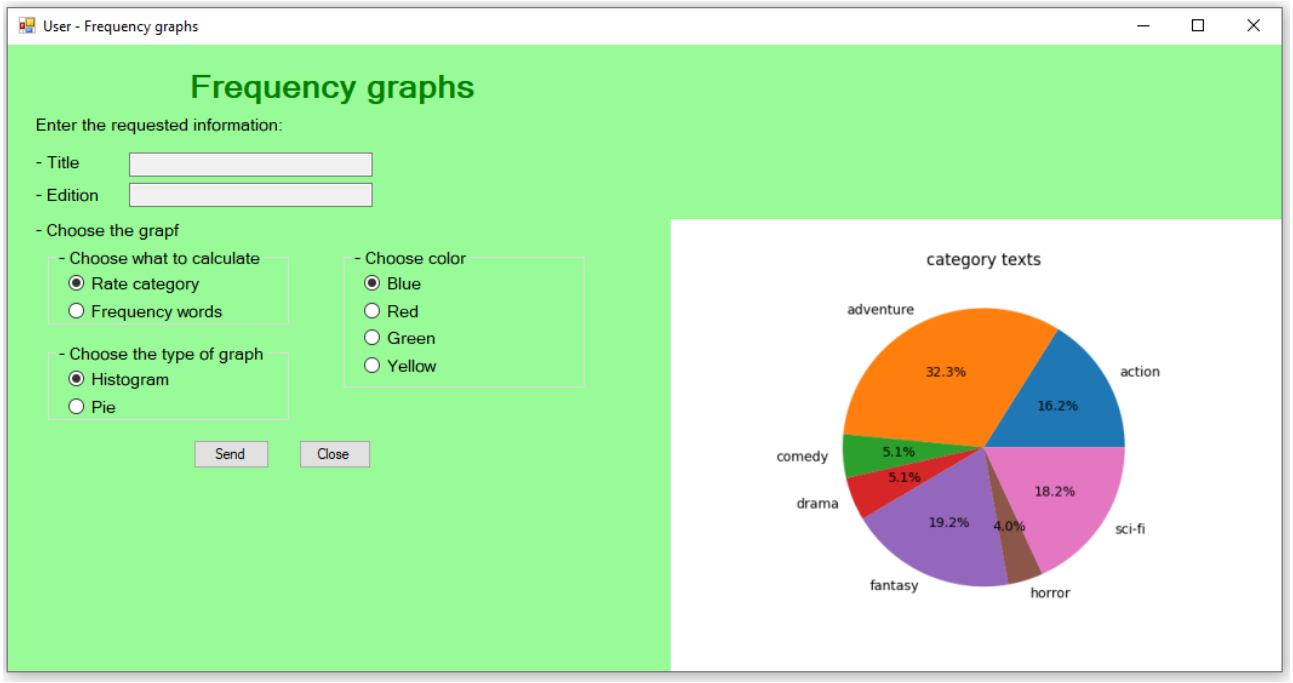
- **Frequency Graphs**: shows statistics on the most frequent words in a text and the genders to which they belong.
- **Text Search**: displays the list of texts by an author and the graph of the most used categories.
- **Text Comparison**: allows you to compare two texts by showing comparative statistics.

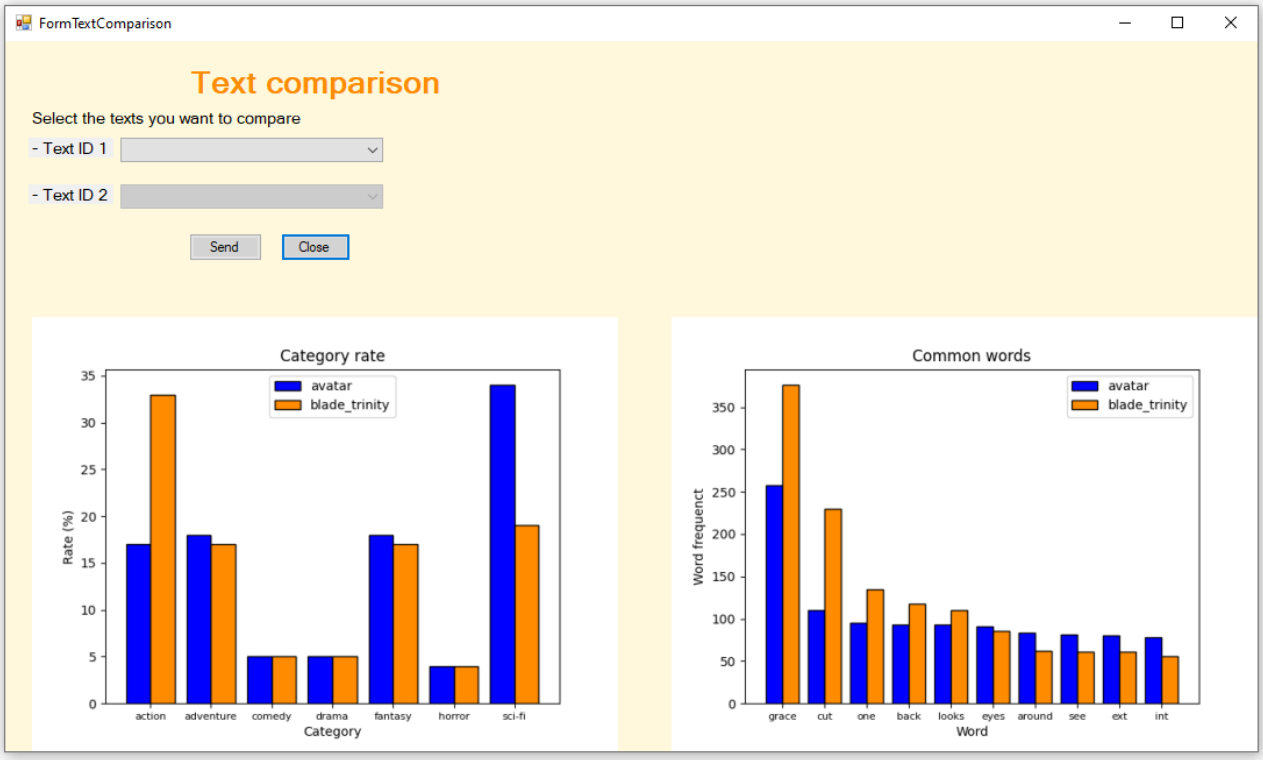
1.3.1. Interface








User - Frequency graphs
Frequency graphs
Enter the requested information:
- Title
- Edition
- Choose the grapf
- Choose what to calculate
Rate category
Frequency words
- Choose color
Blue
Red
Green
Yellow
- Choose the type of graph
Histogram
Pie
Send
Close
category texts
adventure
action
comedy
drama
fantasy
horror
sci-fi
32.3%
16.2%
5.1%
5.1%
18.2%
19.2%
4.0%


FormTextComparison
Text comparison
Select the texts you want to compare
- Text ID 1
- Text ID 2
Send
Close
Category rate
avatar
blade_trinity
Rate (%)
Category
action
adventure
comedy
drama
fantasy
horror
sci-fi
Common words
Word frequenct
Word
grace
cut
one
back
looks
eyes
around
see
ext
int

1.3.2. Implementation choices

Here are some of the most significant code implementations.

In the "Home Page" form, you can switch from one form to another through a "Go" button.

```
/* ---------------------- EVENT: Click ----------------------*/
1 riferimento
private void ButtonInsertNewText_Click(object sender, EventArgs e)
{
    FormInsertNewText insertNewText = new FormInsertNewText();
    insertNewText.ShowDialog(); // This way I can go back to the home screen only when the new screen is closed
}

1 riferimento
private void ButtonGoWordClound_Click(object sender, EventArgs e)
{
    FormWorldCloud worldCloud = new FormWorldCloud();
    worldCloud.ShowDialog(); // This way I can go back to the home screen only when the new screen is closed
}

1 riferimento
private void ButtonGoFrequencyGraphs_Click(object sender, EventArgs e)
{
    FormFrequencyGraphs frequencyGraphs = new FormFrequencyGraphs();
    frequencyGraphs.ShowDialog(); // This way I can go back to the home screen only when the new screen is closed
}
```

Figure 1.1 Home Page

A list is created on the Homepage in which the metadata of all texts in the database is loaded. In this way, the list always remains up-to-date with the user's texts, avoiding the need to make repeated queries to the database.

When you insert a new text using the InsertNewText operation, it is saved in both the database and the list. This allows you to always work with the latest data without having to reload the text list every time.

```
/* ---------------------- FUNCTIONS ----------------------*/
1 riferimento
private async void FillList()
{
    try
    {
        using (var client = new HttpClient())
        {
            // Set base URL of the REST API
            var endPoint = new Uri(ConfigRest.UrlRetrieveTextList);

            // Send POST request to the REST API
            var response = await client.GetAsync(endPoint);
            if (response.IsSuccessStatusCode)
            {
                string content = await response.Content.ReadAsStringAsync();
                textList = JsonConvert.DeserializeObject<List<Text>>(content);
            }
            else
            {
                MessageBox.Show("Failed to send JSON file. Status code: " + response.StatusCode);
            }
        }
    }
    catch (Exception ex)
    {
        ShowErrorMessage("An error occurred: " + ex.Message);
    }
}
```

Figure 1.3 Home Page

```
class Text
{
    3 riferimenti
    public string Id { get; set; }
    10 riferimenti
    public string Title { get; set; }
    3 riferimenti
    public string Author { get; set; }
    8 riferimenti
    public string Edition { get; set; }
    1 riferimento
    public string Isbn { get; set; }
    6 riferimenti
    public string Words { get; set; }

    1 riferimento
    public Text(string title, string author, string edition, string isbn, string words)
    {
        this.Id = title + " - edition " + edition;
        this.Title = title;
        this.Author = author;
        this.Edition = edition;
        this.Isbn = isbn;
        this.Words = words;

        GlobalList.Add(this);
    }
}
```

Figure 1.2 Class Text

In the "Text Comparison" form, the two comboBoxes that allow you to select the texts in the DB and previously loaded are filled in as shown in the code below. In addition, the mechanism to prevent you from being able to select the same text is also implemented.

```
/* ---------------------- FUNCTIONS ----------------------*/
1 riferimento
void FillComboBoxX2()
{
    comboBoxListText1.SelectedIndex = -1;
    comboBoxListText2.SelectedIndex = -1;

    //add items to both combos
    comboBoxListText1.Items.AddRange(textList.Select(i => i.Title + " - edition " + i.Edition).ToArray());
    comboBoxListText2.Items.AddRange(textList.Select(i => i.Title + " - edition " + i.Edition).ToArray());

    // attach event handlers
    comboBoxListText1.SelectedIndexChanged += ComboBoxListText1_SelectedIndexChanged;
    comboBoxListText1.SelectionChangeCommitted += ComboBoxListText1_SelectionChangeCommitted;

    comboBoxListText2.SelectedIndexChanged += ComboBoxListText2_SelectedIndexChanged;
    comboBoxListText2.SelectionChangeCommitted += ComboBoxListText2_SelectionChangeCommitted;
}
/* ---------------------- EVENT: SelectedIndexChanged and SelectionChangeCommitted ---
2 riferimenti
private void ComboBoxListText1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBoxListText1.SelectedIndex != -1) {
        //remove selected item from comboBoxListText2
        string selected = comboBoxListText1.SelectedItem.ToString();
        comboBoxListText2.Enabled = true;
        comboBoxListText2.Items.Remove(selected);
    }
}
2 riferimenti
private void ComboBoxListText1_SelectionChangeCommitted(object sender, EventArgs e)
{
    // add item to comboBoxListText2
    string deselected = comboBoxListText1.SelectedItem.ToString();
    comboBoxListText2.Items.Add(deselected);
}

2 riferimenti
private void ComboBoxListText2_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBoxListText2.SelectedIndex != -1)
    {
        //remove selected item from comboBoxListText1
        string selected = comboBoxListText2.SelectedItem.ToString();
        comboBoxListText1.Items.Remove(selected);
    }
}
2 riferimenti
private void ComboBoxListText2_SelectionChangeCommitted(object sender, EventArgs e)
{
    // add item to comboBoxListText1
    string deselected = comboBoxListText2.SelectedItem.ToString();
    comboBoxListText1.Items.Add(deselected);
}
```

Figure 1.4 Text Comparison

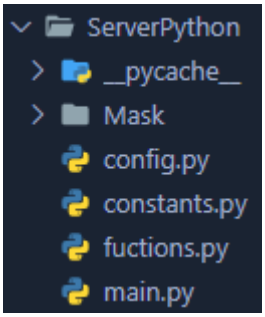
2. Back-End (Python)

2.1. Introduction

The python server takes care of the implementation of the text mining algorithm, the generation of dictionaries useful for carrying out statistical operations and the generation of images. It also allows communication between the Go server and clients in C#.

2.2. Architecture

The Python server is structured as follows:



The server allows you to implement various functions to perform the operations described in the "C#" chapter.

These operations are:

- **frequencyCategory:** This function receives a certain category and its ".txt" file. From it, a text mining operation is carried out by generating a dictionary containing the words and their frequencies and then sending them, via Rest API, to the Go server.

- **insertNewText:** this function receives the information about a new text and its ".txt" file. From it, it performs a text mining operation to calculate the frequency of words and sends the data, via Rest API, to the Go server to calculate the categories to which it belongs.

- **wordCloud:** This function receives input from a dictionary containing the frequencies of the words of a text and generates a word cloud image. To use the word cloud library, you need to install it using the commands in the readme.

- **frequencyGraphs:** this function receives two input dictionaries containing the frequencies of the words of two different texts and calculates the statistics by generating the relative graphs.

- **textSearch:** this function receives dictionaries of the various texts written by a specific author and then forwards them to the Go server via Rest API. Finally, with the data received from the Go server, it will create a statistical graph.

- **textComparison:** This function receives two input dictionaries containing the frequencies of each word and sends them to the Go server. The results obtained by the server are processed and then graphed.

2.3. Meaningful Code

Here are some of the most significant code implementations.

In the following function, a dictionary containing the word/frequency pair is created by analyzing word by word the contents of the ".txt" file received as input.

```
# ---- Frequency words
def frequencyWords(wordlist, punctuations, uninteresting_words):
    """ Parses a list of words and returns a dictionary with the frequency of each word.
        Parameters:
match typeGraph2:
    case "Histogram":
        ax.bar(labels, values, color = color, ec="black")
        ax.set_ylabel("Rate %")
        ax.set_title(title)

        plt.xticks(rotation=30, ha='right')
        plt.tight_layout()

    case "pie":
        #ax.pie(values, labels=labels)
        ax.pie(values, labels=labels, autopct='%1.1f%%')
        ax.set_title(title)
return plt                                                    :
            if new_string in word_dictionary:
                value = word_dictionary[new_string]
                word_dictionary[new_string] = value +1
            else:
                word_dictionary[new_string] = 1
    return word_dictionary
```

Below is a snippet of code that allows you to generate a graph:

Finally, here's the code to generate a word cloud image:

```
# Create the worldcloud
if(wcMaskType =="default"):
    wc = WordCloud(background_color="black", contour_width=3, contour_color='white')
else:
    maskPath = "mask/" + wcMaskType + "_mask.png"
    print(maskPath)
    # Let's create the mask
    mask = np.array(Image.open(maskPath))
    wc = WordCloud(background_color="black", mask=mask, contour_width=3, contour_color='white')

wc.generate_from_frequencies(wordDictionary)

# Create the image
plt.figure(frameon=False)                # Create the figure for the image
plt.imshow(wc, interpolation="bilinear") # Show the wc image
plt.axis("off")

return plt
```

3. Front-End (Go)

3.1. Introduction

The Go server takes care of the connection with a MongoDB Atlas database and performs operations to process the data received from the Python server before inserting it into the database.

Communication with Python is done through Rest API calls.

3.2. Architecture

The server allows you to implement various functions to perform the operations described in the "C#" chapter.

These operations are:

- **ReceiveCategory**: This function receives a category and its map. The function checks whether the category already exists or not. If the category is already present, update the corresponding dictionary by comparing it with the one received, otherwise create a new category. In both cases, the relevant collection in the database will be updated.

- **ReceiveText**: This function receives information about a text as input. The function checks whether the text already exists or not. If the text is not present, it adds it to the collection in the database. Whether the text exists or not, it will calculate the text's membership in the categories and return the relative percentages.

- **ReceiveTextList**: queries the database and returns the list with the information of all the texts in the collection.

- **TextComparison**: Receives two input dictionaries for two different texts, calculates the relative percentages of the categories to which they belong, and creates two new maps containing common words and their frequencies.

3.3. Meaningful Code

Here are some of the most significant code implementations.

In the following code snippet we report the connection to the database and a Handle Function call:

```go
/********** MAIN **********/
func main() {
    fmt.Println("Welcome")

    //Connection to DB
    client, err := mongo.NewClient(options.Client().ApplyURI("XXXXXXXXXX"))

    if err != nil {
        log.Fatal(err)
    }

    // Create a context with a 10 second timeout
    ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
    // Make the client connection passing the context
    err = client.Connect(ctx)
    if err != nil {
        log.Fatal(err)
    }

    database := client.Database("TextAnalysis")

    /* Handle Functuons */
    http.HandleFunc("/receive-category", func(w http.ResponseWriter, r *http.Request) {

        services.ReceiveCategory(w, r, database)
    })
```

In the following function, a map (with a words/frequency pair) is passed from which a new map containing the percentages of belonging to a category is calculated and returned.

```go
/* ------ Categories Rate ------ */…
func categoriesRate(newTest WordDictionary, database *mongo.Database) map[string]float64 {
    cRate := make(map[string]float64)
    totalRate := 0.0

    // Let's get the collection from the DB
    categoryCollection := database.Collection("Categorys")

    // We find the information we need excluding the id
    findOptions := options.Find()
    findOptions.SetProjection(bson.D{{"_id", 0}})
    cur, err := categoryCollection.Find(context.TODO(), bson.D{}, findOptions)
    if err != nil {
        panic(err)
    }

    //Retrieving search results
    var results []CategoryDict
    if err = cur.All(context.TODO(), &results); err != nil {
        panic(err)
    }

    // Loop over each category dictionary
    for _, category := range results {
        // Loop over the words of the new text
        for textWord, _ := range newTest {
            // Check if the word is present in the current category
            if _, found := category.Words[textWord]; found {
                cRate[category.Category] += 1
                totalRate++
            }
        }
    }

    // Calculate the rate as a percentage and round it off
    for key, val := range cRate {
        cRate[key] = math.Round((val / totalRate) * 100)
    }
    return cRate
}
```

This function queries the database. The result is a list of texts with their attributes.

```go
/* ------ Receive Text List ------ */…
func ReceiveTextList(w http.ResponseWriter, r *http.Request, database *mongo.Database) {
    // closing the body of request
    defer r.Body.Close()

    //Recovery of all texts
    textCollection := database.Collection("Texts")
    findOptions := options.Find()
    findOptions.SetProjection(bson.D{{"_id", 0}})
    cur, err := textCollection.Find(context.TODO(), bson.D{}, findOptions)
    if err != nil {
        panic(err)
    }

    var results []TextDictionaryList
    if err = cur.All(context.TODO(), &results); err != nil {
        panic(err)
    }

    //Map to json conversion
    jsonResponse, _ := json.Marshal(results)

    w.WriteHeader(http.StatusOK)
    w.Header().Set("Content-Type", "application/json")
    w.Write(jsonResponse)
}
```