UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Ingegneria Elettrica Elettronica Informatica

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Progetto Advanced Programming Languages

Analysis of text categories

Anno accademico 2022-2023

Studente:

Andrea Paolo Ventimiglia 1000039024

Sommario

INTRODUZIONE

L'applicazione **TextAnalyses** permette agli utenti di analizzare testi per comprendere le parole più frequenti e il genere di appartenenza. Grazie all'interfaccia sviluppata, gli utenti possono anche confrontare testi diversi ed ottenere statistiche.

L'architettura dell'applicazione prevede:

- Due interfacce grafiche sviluppate in **C#** con Windows Forms, per interagire con l'applicazione in modo intuitivo.
- Un Back-End **Python** con Flask, che espone API REST per ricevere testi ed eseguire operazioni di analisi testuale.
- Un modulo **Go** che gestisce la connessione al database MongoDB Atlas, per salvare e recuperare i testi analizzati. Permette anche operazioni aggiuntive sui dati.

L'integrazione di queste diverse componenti permette di fornire un'applicazione completa, che dall'interfaccia utente fino alla persistenza dei dati sfrutta le potenzialità di diversi linguaggi. L'utente può interagire con semplicità tramite il client grafico e ottenere risultati avanzati di text mining attraverso Python e Go.

1. Front-End (C#)

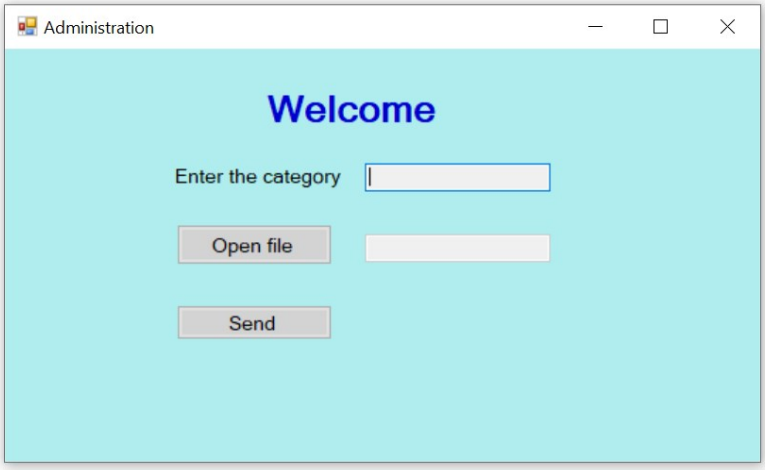
1.1. Introduzione

Nell'applicazione **TextAnalyses** sono stati sviluppati due Client, "Administrator" e "User". Lo sviluppo di entrambi i client si basa sulla gestione di eventi; infatti, ad ogni azione dell'utente corrisponderà un evento che grazie agli handler, presenti nei vari forms, sarà gestito e permetterà di eseguire un'azione/operazione specifica.

1.2. Client "Administrator"

Questo Client è finalizzato all'inserimento di una "categoria" tramite l'inserimento del relativo nome e di un file ".txt" dove effettuare il text mining delle parole.

1.2.1. Interfaccia



1.3. Client "User"

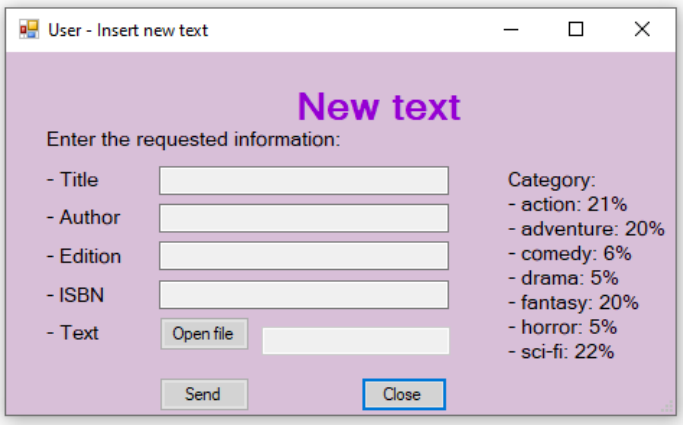
Questo client permette agli utenti di effettuare diverse operazioni di text mining in modo intuitivo tramite un'interfaccia grafica sviluppata con Windows Forms.

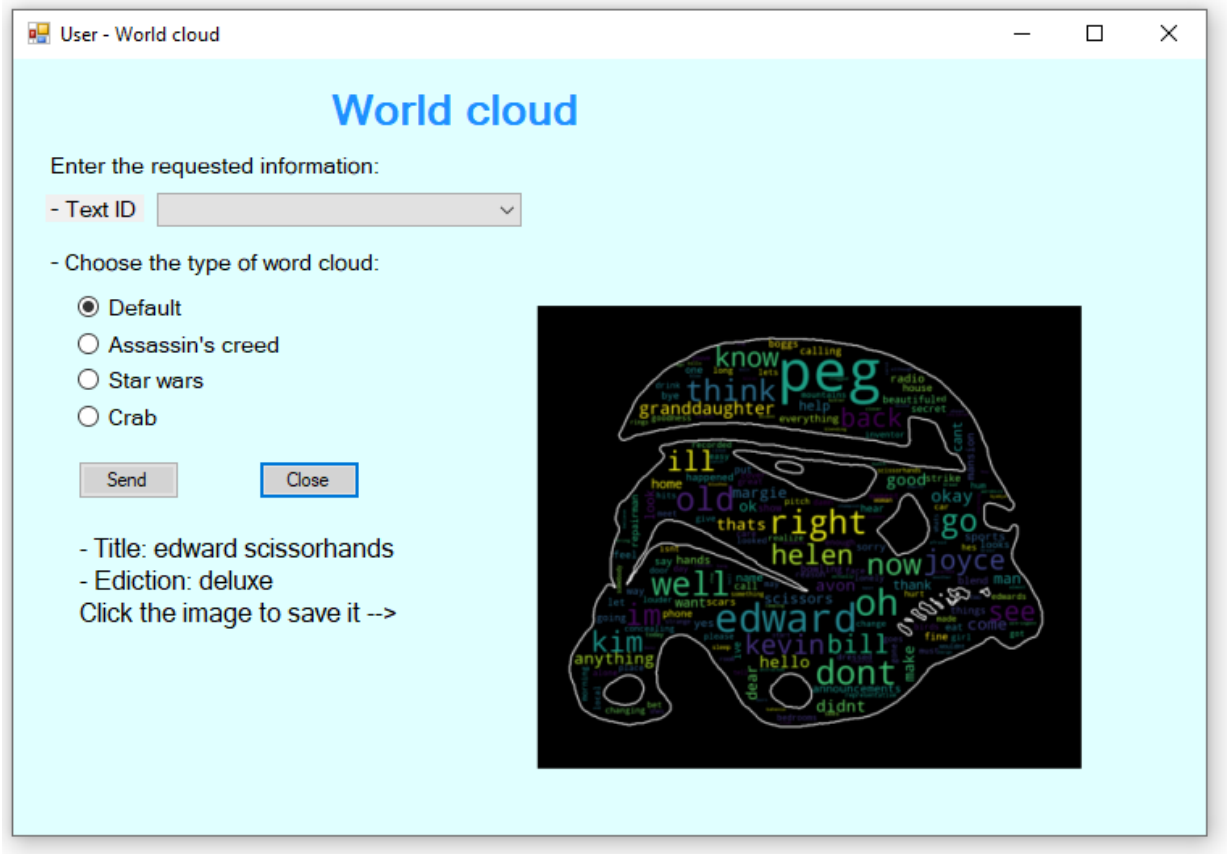
Sono presenti diversi form per le varie funzionalità:

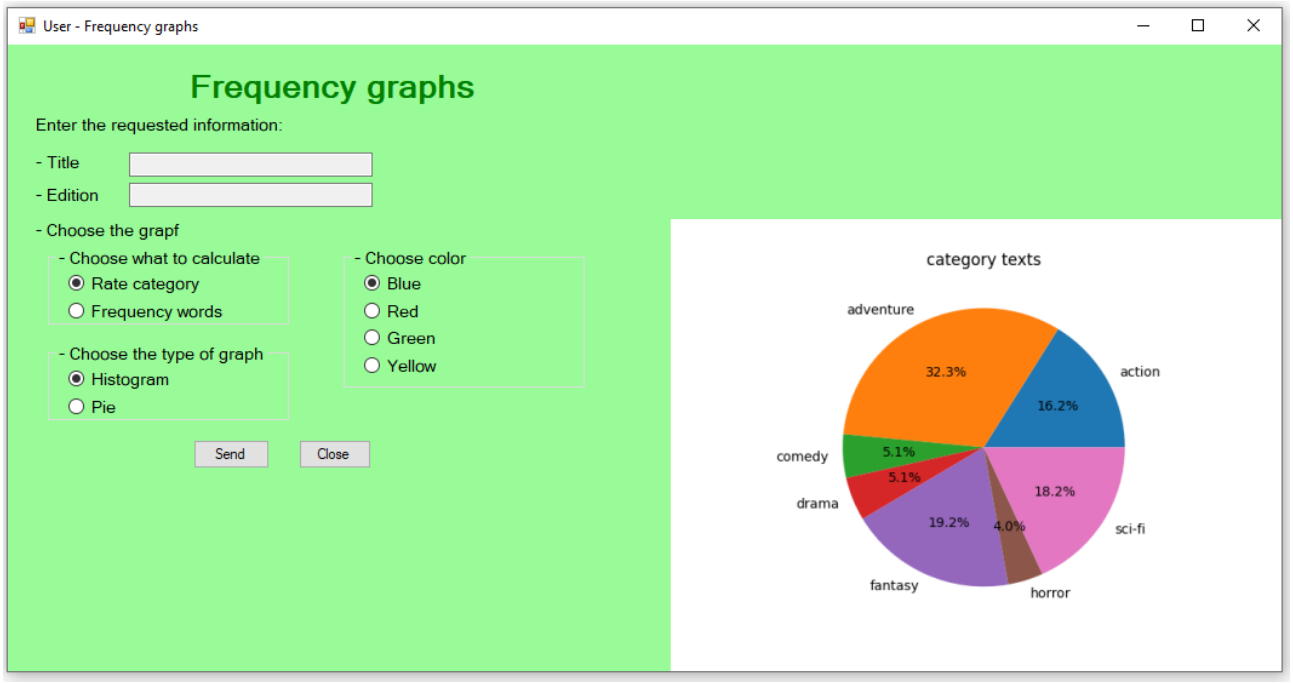
- **HomePage**: permette di visualizzare e selezionare le operazioni disponibili nell'applicazione tramite pulsanti.
- **Insert New Text**: consente di inserire i metadati di un nuovo testo (titolo, autore, etc.) e caricare il file .txt contenente il testo.
- **Word Cloud**: genera un'immagine word cloud salvabile da un testo precedentemente caricato.
- **Frequency Graphs**: mostra statistiche sulle parole più frequenti in un testo e sui generi di appartenenza.

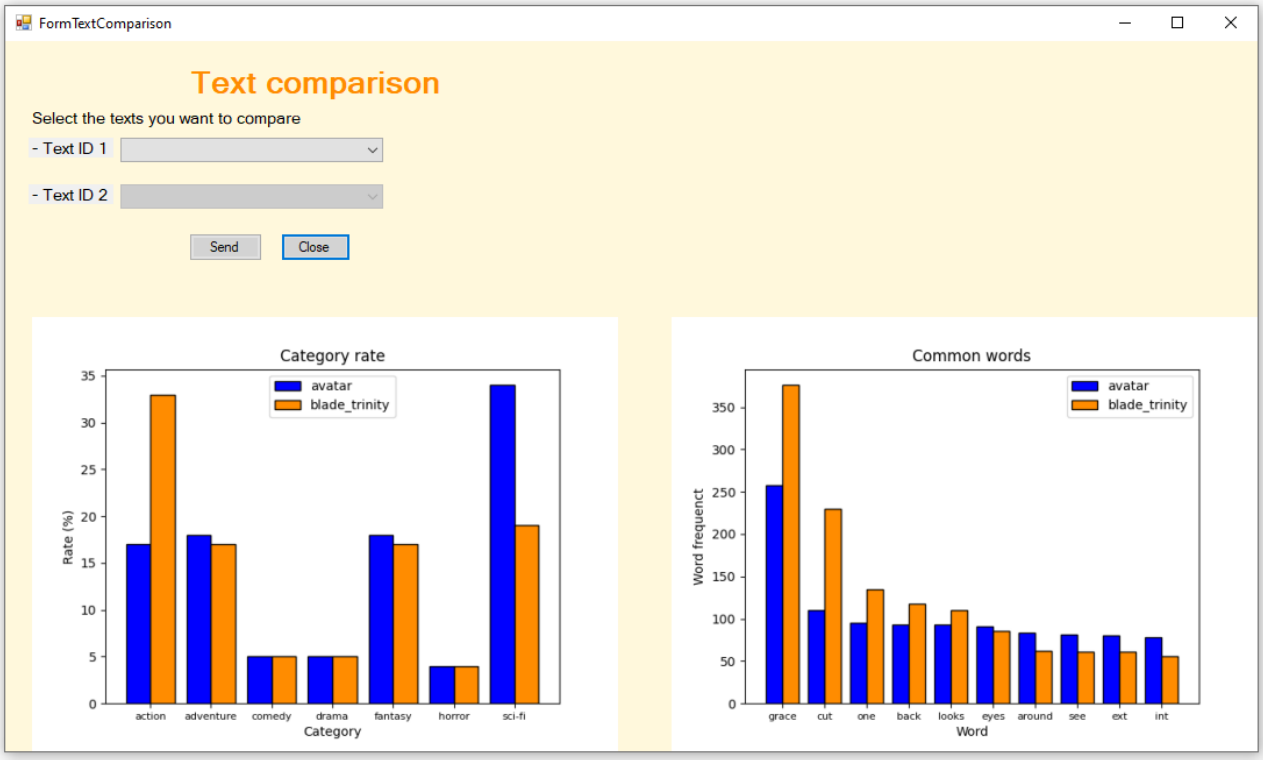
- **Text Search**: visualizza la lista dei testi di un autore e il grafico delle categorie più usate.
- **Text Comparison**: permette di confrontare due testi mostrando le statistiche comparative.

1.3.1. Interfaccia






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. Scelte implementative

Di seguito vengono riportate e descritte alcune implementazioni di codice più significative.

Nel form “Home Page”, attraverso un button “Go” si può passare da un form all’altro.

```
/* ---------------------- 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
}
```

Figura 1.1 Home Page

Nella HomePage viene creata una lista in cui vengono caricati i metadati di tutti i testi presenti nel database. In questo modo la lista rimane sempre aggiornata con i testi dell'utente, evitando di dover effettuare ripetute query al database.

Quando si inserisce un nuovo testo tramite l'operazione InsertNewText, esso viene salvato sia nel database che nella lista. Ciò consente di lavorare sempre con i dati più recenti senza dover ricaricare la lista di testi ad ogni operazione.

```
/* ---------------------- 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);
    }
}
```

Figura 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);
    }
}
```

Figura 1.2 Class Text

Nel form “Text Comparison” i due comboBox che permettono di selezionare i testi presenti nel DB e caricati precedentemente vengono riempiti come mostrato nel codice sotto. Inoltre, viene anche implementato il meccanismo per impedire che si possa selezionare lo stesso testo.

```
/* --------------------- 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);
}
```

Figura 1.4 Text Comparison

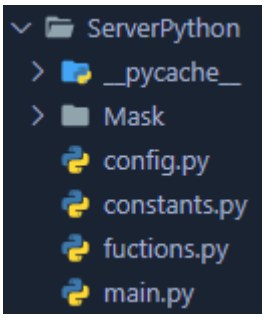
2. Back-End (Python)

2.1. Introduzione

Il server python si occupa dell'implementazione dell'algoritmo di text mining, generezione dei dizionari utili per effettuare le operazioni statistiche e generazione di immagini. Inoltre, permette la comuncazione tra il server Go e i client in C#.

2.2. Architettura

Il server Python è così strutturato:



Il server permette di implementare diverse funzioni per eseguire le operazioni descritte nel capitolo "C#".

Tali operazioni sono:

- **frequencyCategory:** tale funzione riceve in ingresso una determinata categoria e il relativo file".txt". Da esso si effettua un'operazione di text mining generando un dizionario contenente le parole e le relative frequenze per poi inviarle, tramite Rest Api, al server Go.

- **insertNewText:** tale funzione riceve in ingresso le informazioni relative ad un nuovo testo e il relativo file ".txt" . Da esso effettua un'operazione di text mining per calcolare la frequenza delle parole e invia i dati, tramite Rest Api, al server Go per calcolare le categorie di appartenenza.

- **wordCloud:** tale funzione riceve ingresso un dizionario contenente le frequenze delle parole di un testo e genera un'immagine word cloud. Per poter utilizzare la libreria word cloud bisogna installarla mediante i comandi che troviamo nel readme.

- **frequencyGraphs:** tale funzione riceve in ingresso due dizionari contenente le frequenze delle parole di due diversi testi e ne calcola le statistiche generando i relativi grafici.

- **textSearch:** tale funzione riceve in ingresso dei dizionari dei vari testi scritti da un determinato autore per poi inviarli inoltrarli al server Go mediante Rest Api. In fine, con i tati ricevuti dal server Go, creerà un grafico statistico.

- **textComparison:** tale funzione riceve in ingresso due dizionari contenti le frequenze di ogni parola e li invia al server Go. I risultati ottenuti dal server vengono ri elaborati e poi riportati i dei grafici.

2.3. Codice significativo

Di seguito vengono riportate e descritte alcune implementazioni di codice più significative.

Nella seguente funzione viene creato un dizionario contenente la coppia word/frequency andando ad analizzare parola per parola il contenuto del file ".txt" ricevuto in ingresso.

```
# ---- Frequency words
def frequencyWords(wordlist, punctuations, uninteresting_words):
    """ Parses a list of words and returns a dictionary with the frequency of each word.
        Parameters:
        - wordlist: list of words to analyze
        - punctuations: punctuation string to remove
        - uninteresting_words: uninteresting words to remove

        Come back:
        - word_dictionary: dictionary {word: frequency}"""

    # Algorithm for counting word frequency
    word_dictionary = {}
    for word in wordlist:
        new_string=""
        for character in word:
            #if character is not in punctuations and a numbers:
            if character.isalpha() and character not in punctuations:
                new_string = new_string+character.lower()
        if new_string not in uninteresting_words and len(new_string)>0:
            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
```

Di seguito un frammento di codice che permette di generare un grafico:

```
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
```

Infine, viene riportato il codice per generare un'immagine word cloud:

```
# 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. Introduzione

Il server Go si occupa del collegamento con un database MongoDB Atlas ed effettua delle operaizioni per elaborare i dati ricevuti dal server Python prima di inserirli nel database.

La comunicazione con Python avviene tramite chiamate Rest Api.

3.2. Architettura

Il server permette di implementare diverse funzioni per eseguire le operazioni descritte nel capitolo “C#”.

Tali operazioni sono:

- **ReceiveCategory**: tale funzione riceve in ingresso una categoria con relativa mappa. La funzione controlla se la categoria esiste già oppure no. Se la categoria è già presente aggiorna il corrispondente dizionario confrontandolo con quello ricevuto, altrimenti crea una nuova categoria. In entrambi i casi verrà aggiornata la relativa collection nel database.

- **ReceiveText**: tale funzione riceve in ingresso le informazioni relative ad un testo. La funzione controlla se il testo esiste già oppure no. Se il testo non è presente lo aggiunge alla collection nel database. Sia se il testo esiste oppure no calcolerà l’appartenenza del testo alle categorie e restituirò le relative percentuali.

- **ReceiveTextList**: interroga il database e restituisce la lista con le informazioni di tutti i testi presenti nella collection.

- **TextComparison**: riceve in ingresso due dizionari relativi a due diversi testi, calcola le relative percentuali delle categorie di appartenenza e crea due nuove mappe contenente le parole comuni e le relative frequenze.

3.3. Codice significativo

Di seguito vengono riportate e descritte alcune implementazioni di codice più significative.

Nel seguente frammento di codice riportiamo la connessione al database e una chiamata Handle Function:

```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)
    })
```

Nella seguente funzione viene passata una mappa (con coppia words/frequency) da cui viene calcolata e ritornata una nuova mappa contenente le percentuali di appartenenza ad una categoria.

```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
}
```

In questa funzione viene effettuata una query per interrogare il database. Il risultato ottenuto è la lista dei testi con i relativi attributi.

```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)
}
```