Developing Intelligent Apps

Lab 2 – Creating a Text Analytics Application

By Gerry O'Brien

Overview

In this lab you will construct a client application that takes a hashtag as input, searches Twitter for that hashtag getting results, then passing these into the Text Analytics API for sentiment analysis.

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A Text Analytics API account key
- A Twitter account
- A Twitter API key (instructions in lab for this)
- A web browser and Internet connection
- A computer running Windows 10 and at least build 10586
- Microsoft Visual Studio 2015 Community Edition

Get a Text Analytics API Key

In this exercise you will sign up for the Text Analytics API key that will be used to consume the sentiment analysis portion of the lab. If you already have an API key for this service, you may skip this exercise.

1. Navigate to **Cognitive Services** in the Azure Portal and ensure **Text Analytics** is selected as the 'API type'.
2. Select a plan. You may select the **free tier for 5,000 transactions/month**. As is a free plan, you will not be charged for using the service. You will need to login to your Azure subscription.
3. Complete the other fields and create your account.
4. After you sign up for Text Analytics, find your **API Key**. Copy the primary key, or keep the page open, as you will paste it in your code in a later exercise.

Get a Twitter Consumer (API) Key

In this exercise you will generate a consumer key for your application to access the Twitter APIs. Twitter now uses OAuth for secure access to the Twitter functions from your applications. You must have a valid

Twitter account with your mobile number entered in your profile before you can create a consumer key.

1. Open your browser and go to https://dev.twitter.com/overview/documentation.
2. Locate the Manage My Apps option in the left nav area and click it.
3. Click the Create New App button in the upper right of the page.
4. Complete the required information and accept the developer agreement.
5. Click Create Your Twitter Application
6. Once the application is created you are taken the Details page. You will use the Consumer Key (API Key) on this page in your application for accessing the Twitter Search API. But first you will also need Access Tokens as well.
7. Click the Keys and Access Tokens tab.
8. Click on the Create my access token button towards the bottom of the page. Twitter generates the Access Token and Access Token Secret. These will be needed later when out application is ready to authenticate for Twitter searches.

Prepare the Application

In this exercise you will open and prepare the C# Universal application for searching Twitter. Ensure that you have downloaded the DAT211xLab2Starter.zip file from GitHub.

Configure for Twitter and Text Analytic APIs

You should have already completed the previous exercise to ensure that you have a Twitter account and consumer key.

1. Download and uncompress the starter application from GitHub.
2. Open Visual Studio.
3. Open the DAT211xLab2Starter solution that you just uncompressed.
4. If MainPage.xaml isn't already open, open it by double clicking to get an idea of the UI for this simple application. You will be working with a Universal Windows Application for this lab.
5. Click the arrow next to MainPage.xaml in the Solution Explorer
6. Double-click MainPage.xaml.cs to open the code file for this application.
7. Immediately following the opening brace for the MainPage class, and before the constructor for MainPage(), copy and paste the following code:

```
/// <summary>
        /// Azure portal URL.
        /// </summary>
        private const string BaseUrl =
"https://westus.api.cognitive.microsoft.com/";

        /// <summary>
        /// Your account key goes here.
        /// </summary>
        private const string AccountKey = "";

        /// <summary>
        /// Twitter Consumer Key
        /// </summary>
```

```
private const string consumerKey = "";
private const string consumerSecretKey = "";
private const string accessToken = "";
private const string accessTokenSecretKey = "";
```

8. The baseUrl is used to connect to the Cognitive Services for the Text Analytics APIs
9. You will need to replace your AccountKey with the one you generated when signing up for TextAnalytics and the other keys are replaced with those you created for the Twitter setup.

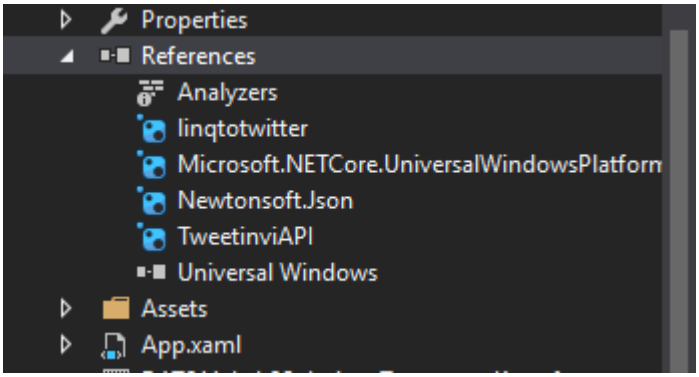
   NOTE: hardcoding API and account keys is not a best-practice but is only used to keep the lab simple. Normally you want to keep these in a configuration file that you control access to.

10. Your application is now set to authenticate with Twitter and Microsoft's Cognitive services.

Import the Necessary Libraries

In this exercise, you will import two libraries that will make coding the application much easier. Twitter requires OAuth for authentication with your applications that access Twitter and the Tweetinvi library makes this authentication easy. Also, you will be working with JSON as the data format in this application and the Newtonsoft.Json library makes serializing and deserializing JSON relatively easy.

1. With Visual Studio open, right click on References in the Dat211Lab2Starter project, in Solution Explorer.
2. Select Manage Nuget Packages.
3. Ensure that All is selected in the Package source: drop down in the top right corner of the Package Manager window.
4. Enter Tweetinvi in the search box
5. Install TweetinviAPI. (This lab uses TweetinviAPI version 0.9.14.)
6. Visual Studio will take a little time to download, install, and configure the project with the library. The output window at the bottom will indicate when installation has finished.
7. Now enter Newtonsoft in the search box.
8. Select and install the Newtonsoft.Json library by James Newton-King.
9. Again, wait until the installation has indicated it is finished.
10. Verify that you have these libraries listed in the References folder as shown here:



11. You now have the key libraries that will help us build our application. You can close the Nuget Package Manager window.

Create Additional Classes to Support the JSON Results

The Text Analytics API that we will use is the sentiment analysis portion. The response is in JSON format but we need to gain access to the score values that are returned and the easiest way is to deserialize the JSON and create a list of objects from which we can gain access to the score values. We will use these to calculate the average of the sentiment values for the tweets passed in.

1. Right-click the project and choose Add, then select Class.
2. Name the class Sentiment and click OK
3. Once the class file opens in Visual Studio, add two properties to the class using this code:

```
public string score { get; set; }
public string id { get; set; }
```

4. The preceding code sets up an object that will represent the unique entries in the JSON file that provide the ID and the score values returned from the Text Analytics API
5. Right-click the project and choose Add, then select Class again.
6. Name this class RootObject and click OK.
7. Paste this code in the RootObject class file.

```
public List<Sentiment> documents {get; set; }
```

8. This class will be used to contain a list of Sentiment objects representing all of the returned scores and IDs from the JSON file.
9. Our two additional class files are now complete.
10. Leave Visual Studio open for the next exercise

Create the Twitter Authorization Method

The first thing we need to do is add the necessary using directives in our code file and setup the authorization for Twitter. Our first action is to query twitter using a hashtag but we need to have our credentials for Twitter first.

1. At the top of the MainPage.xaml.cs file, add the following using directives

```
using Tweetinvi;
using Tweetinvi.Core.Parameters;
```

2. Create a method called AuthorizeTwitter() that takes no parameters and returns void
3. Within the AuthorizeTwitter() method, add the following line of code to setup our authorization.

```
Auth.SetUserCredentials(consumerKey, consumerSecretKey, accessToken,
accessTokenSecretKey);
```

4. The Auth object is part of the Tweetinvi library and it takes your Twitter keys and tokens and authorizes your application to work with Twitter. Without Tweetinvi, you would need to write code to request a bearer token and then use that bearer token with each request you made to Twitter. This method handles the necessary authorization for you.

Write the Remaining Methods

Now that we have setup the application to authenticate with Twitter, we are now ready to begin writing code that will take a hashtag parameter and call the Twitter Search API, returning tweets related to that hashtag. Note that Twitter only allows for search results from the previous 7 days maximum but that is fine for our application scenario.

Create the Twitter Search Method

1. Create a private method named SearchTwitter() that returns a Generic List<String> object.
2. In this method, paste the following code:

```
string input = txtInput.Text;
string searchParam = input.Replace("#", "%23");
var searchParameter = new TweetSearchParameters(searchParam)
{
    SearchType = Tweetinvi.Core.Enum.SearchResultType.Popular,
    MaximumNumberOfResults = 100
};
var results = Search.SearchTweets(searchParameter);
List<String> tweets = new List<string>();
foreach (var item in results)
{
    tweets.Add(item.ToString());
}
return tweets;
```

3. In this code, we first get the parameter from the text box and assign it to the variable input. Feel free to add validation code if you want to make your application more robust.
4. Twitter requires our queries to be properly encoded so we replace the hashtag with %23 in the search term. You can also just choose to prepend the %23 in front of any entry and not require the user to enter the hashtag in the first place.
5. Tweetinvi allows us to setup a set of search parameters that can be applied to our Twitter searches and the searchParameter variable does just that for us. It uses the value entered by our user, then adds parameters for the search type and the max results to return. Here we use Popular as the search type as we are interested in knowing about the sentiment for the most popular tweets for our search term.
6. Next, we call Tweetinvi's SearchTweets method to go and get the results for us.
7. Using a foreach loop, we fill the Generic list object and return it back to our calling method.

Create the FormatRequestJSON Method

1. Create a private method named FormatRequestJSON that takes a Generic List<String> object as input and returns a string.
2. In this method, paste the following code:

```
int idCounter = 1;

StringBuilder reqJSON = new StringBuilder();
reqJSON.Append("{\"documents\":[");
foreach (string item in values)
{
    reqJSON.Append("{\"id\":\"" + idCounter + "\",\"text\":\"" + item +
"\"},");
```

```
            idCounter++;
        }

        reqJSON.Append("]}");
        return reqJSON.ToString();
```

3. You have another option for this code whereby you could create an object from the results returned, rather than a List<>, and then use the library for serializing the object list into a JSON but for this example, the JSON format is very specific for the Text Analytics API request so we have opted to build the JSON the long way.
4. Once we have added all of the Tweet texts into our JSON string, we return back to the calling method.

Create the Methods to Call the Text Analytics APIs

1. We will create two asynchronous methods for calling the API.
2. Paste the following code:

```
    private async void MakeRequests(string request)
    {
        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri(BaseUrl);

            // Request headers.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", AccountKey);
            client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));

            // Request body. Insert your text data here in JSON format.
            byte[] byteData = Encoding.UTF8.GetBytes(request);

            // Detect sentiment:
            var uri = "text/analytics/v2.0/sentiment";
            var response = await CallEndpoint(client, uri, byteData);
            tbResults.Text = response;

        }
    }

    private async Task<String> CallEndpoint(HttpClient client, string uri, byte[]
byteData)
    {
        using (var content = new ByteArrayContent(byteData))
        {
            content.Headers.ContentType = new
MediaTypeHeaderValue("application/json");
            var response = await client.PostAsync(uri, content);
            return await response.Content.ReadAsStringAsync();
        }
    }
```

3. These two methods work together to setup an Http client that is used to make the request to the APIs. Note that your Text Analytics API account key is used here.
4. The JSON we created in the previous method is passed in to the MakeRequests() method where it is encoded as a byte array and then passed to the CallEndpoint() method. CallEndpoint makes the actual call to the API and gets the response asynchronously.

5. Back in the MakeRequests() method, we display the sentiment API results in the text box on the app's UI so that you can see the values returned.

Create the AnalyzeSentiment Method

1. Create a method called AnalyzeSentiment() that returns void but accepts a string value as input
2. Paste the following code in the method:

```
double total = 0;

RootObject obj = JsonConvert.DeserializeObject<RootObject>(value);

foreach (var item in obj.documents)
{
    total += (Double.Parse(item.score) * 100);
}
double average = total / obj.documents.Count;

Windows.UI.Xaml.Media.SolidColorBrush myBrush = new
Windows.UI.Xaml.Media.SolidColorBrush();

if (average > 0 && average <= 40)
{
    myBrush.Color = Colors.Red;
    rectSentiment.Fill = myBrush;
}
else if (average > 40 && average <= 65)
{
    myBrush.Color = Colors.Yellow;
    rectSentiment.Fill = myBrush;
}
else
{
    myBrush.Color = Colors.Green;
    rectSentiment.Fill = myBrush;
}
```

3. In this method, we use the Newtonsoft.JSON library to deserialize the JSON data that was returned from the Text Analytics API. To deserialize, we need the RootObject that we created earlier, which will contain a list of Sentiment objects that have the id and score values from the JSON file.
4. The foreach loop is used to calculate the average sentiment. Recall that the Text Analytics API returns sentiment values in the range of 0 to 1. Values closer to 0 represent negative sentiment while positive sentiment is indicated closer to 1.
5. The if structure sets up colors to use for the rectangle on the UI and indicates the average sentiment by using a range of values to display either red for negative sentiment, yellow for mediocre, and green for positive. Feel free to adjust these to suit your own ranges of sentiment.

Create the Event Handlers for the Buttons

1. Switch back to the designer for MainPage.xaml and double-click the Get Sentiment button. This should create an event handler called btnGetSentiment_Click().
2. Paste the following code in the event handler:

```
AuthorizeTwitter();
```

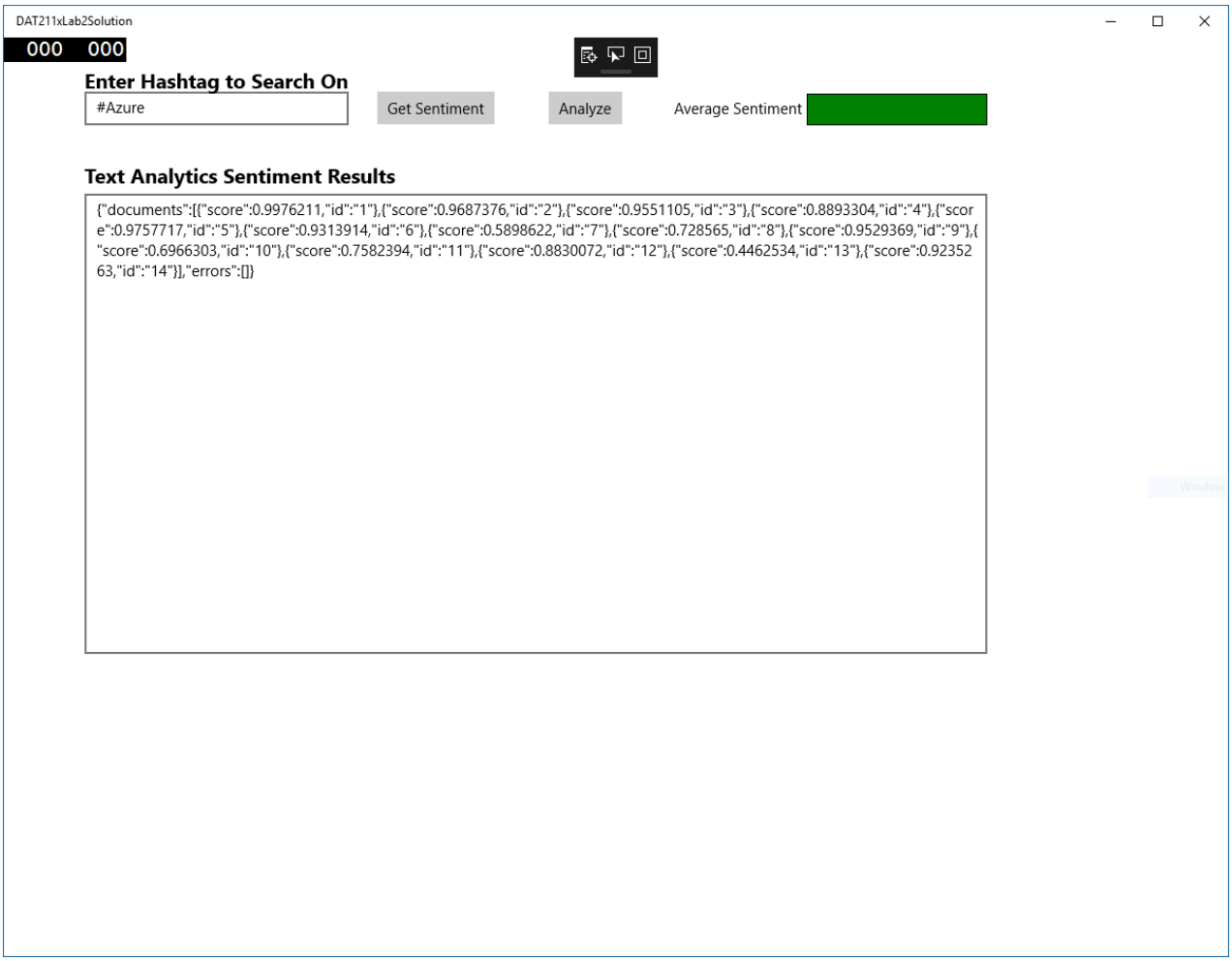
```
List<String> response = SearchTwitter();
string request = FormatRequestJSON(response);
MakeRequests(request);
```

3. This code simply makes the necessary calls to the methods we use to search for the Tweets and to format the JSON request for the Text Analytics API.
4. Switch back to the designer and double-click the Analyze button to create the btnAnalyze_Click event handler and type in the following single line of code:

   AnalyzeSentiment(tbResults.Text);

5. This completes the coding necessary for this application.
6. Run the application, enter a hashtag to search on, click Get Sentiment and see if the JSON result is returned and displays in the results text box.

   NOTE: There are instances where you may encounter a result that indicates a bad request. The application has not been debugged to identify the reasons but it is suspected that some characters being returned in the Tweets is causing an issue in the JSON request to the Text Analytics API. Feel free to troubleshoot on your own to solve this.
7. Once you have valid results returned, click the Analyze button to determine the sentiment. An example is shown here when searching on #Azure.



Summary

In this lab you have:

- Created a Twitter application and set up authorization for your app
- Setup an account for the Text Analytics API
- Created a simple client application using C# and Visual Studio to search Twitter and get a sentiment from the Microsoft Cognitive Services on the text in those Tweets.