Contents

External and community content

FAQ

Language and region support

Pricing

Regional availability

Stack Overflow

Subscription key

Try it!

UserVoice

What is Text Analytics?

10/9/2018 • 4 minutes to read • Edit Online

The Text Analytics API is a cloud-based service that provides advanced natural language processing over raw text, and includes four main functions: sentiment analysis, key phrase extraction, language detection, and entity linking.

The API is backed by resources in Microsoft Cognitive Services, a collection of machine learning and AI algorithms in the cloud, readily consumable in your development projects.

Capabilities in Text Analytics

Text analysis can mean different things, but in Cognitive Services, the Text Analytics API provides four types of analysis as described in the following table.

OPERATIONS	DESCRIPTION	APIS
Sentiment Analysis	Find out what customers think of your brand or topic by analyzing raw text for clues about positive or negative sentiment. This API returns a sentiment score between 0 and 1 for each document, where 1 is the most positive. The analysis models are pretrained using an extensive body of text and natural language technologies from Microsoft. For selected languages, the API can analyze and score any raw text that you provide, directly returning results to the calling application.	REST .NET
Key Phrase Extraction	Automatically extract key phrases to quickly identify the main points. For example, for the input text "The food was delicious and there were wonderful staff", the API returns the main talking points: "food" and "wonderful staff".	REST .NET
Language Detection	For up to 120 languages, detect which language the input text is written in and report a single language code for every document submitted on the request. The language code is paired with a score indicating the strength of the score.	REST .NET
Entity Recognition (Preview)	Identify and categorize entities in your text as people, places, organizations, date/time, quantities, percentages, currencies, and more. Well-known entities are also recognized and linked to more information on the web.	REST

Typical workflow

The workflow is simple: you submit data for analysis and handle outputs in your code. Analyzers are consumed as-is, with no additional configuration or customization.

1. Sign up for an access key. The key must be passed on each request.
2. Formulate a request containing your data as raw unstructured text, in JSON.
3. Post the request to the endpoint established during sign-up, appending the desired resource: sentiment analysis, key phrase extraction, language detection, or entity identification.
4. Stream or store the response locally. Depending on the request, results are either a sentiment score, a collection of extracted key phrases, or a language code.

Output is returned as a single JSON document, with results for each text document you posted, based on ID. You can subsequently analyze, visualize, or categorize the results into actionable insights.

Data is not stored in your account. Operations performed by the Text Analytics API are stateless, which means the text you provide is processed and results are returned immediately.

Supported languages

This section has been moved to a separate article for better discoverability. Refer to Supported languages in Text Analytics API for this content.

Data limits

All of the Text Analytics API endpoints accept raw text data. The current limit is 5,000 characters for each document; if you need to analyze larger documents, you can break them up into smaller chunks. If you still require a higher limit, contact us so that we can discuss your requirements.

LIMIT	VALUE
Maximum size of a single document	5,000 characters as measured by `String.Length`.
Maximum size of entire request	1 MB
Maximum number of documents in a request	1,000 documents

The rate limit is 100 calls per minute. Note that you can submit a large quantity of documents in a single call (up to 1000 documents).

Unicode encoding

The Text Analytics API uses Unicode encoding for text representation and character count calculations. Requests can be submitted in both UTF-8 and UTF-16 with no measurable differences in the character count. Unicode codepoints are used as the heuristic for character length and are considered equivalent for the purposes of text analytics data limits. If you use `String.Length` to get the character count, you are using the same method we use to measure data size.

Next steps

First, try the interactive demo. You can paste a text input (5,000 character maximum) to detect the language (up to 120), calculate a sentiment score, or extract key phrases. No sign-up necessary.

When you are ready to call the API directly:

- Sign up for an access key and review the steps for calling the API.
- Quickstart is a walkthrough of the REST API calls written in C#. Learn how to submit text, choose an analysis, and view results with minimal code.
- API reference documentation provides the technical documentation for the APIs. The documentation supports embedded calls so that you can call the API from each documentation page.
- External & Community Content provides a list of blog posts and videos demonstrating how to use Text Analytics with other tools and technologies.

See also

Cognitive Services Documentation page

Quickstart: Using C# to call the Text Analytics Cognitive Service

10/12/2018 • 3 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, and extract key phrases using the Text Analytics APIs with C#. The code was written to work on a .Net Core application, with minimal references to external libraries, so you could also run it on Linux or MacOS.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart.

You must also have the endpoint and access key that was generated for you during sign up.

Install the Nuget SDK Package

1. Create a new Console solution in Visual Studio.
2. Right click on the solution and click **Manage NuGet Packages for Solution**
3. Mark the **Include Prerelease** checkbox.
4. Select the **Browse** tab, and Search for **Microsoft.Azure.CognitiveServices.Language.TextAnalytics**
5. Select the Nuget package and install it.

TIP

While you could call the HTTP endpoints directly from C#, the Microsoft.Azure.CognitiveServices.Language SDK makes it much easier to call the service without having to worry about serializing and deserializing JSON.

A few useful links:

- SDK Nuget page
- SDK code

Call the Text Analytics API using the SDK

1. Replace Program.cs with the code provided below. This program demonstrates the capabilities of the Text Analytics API in 3 sections (language extraction, key-phrase extraction and sentiment analysis).
2. Replace the `Ocp-Apim-Subscription-Key` header value with an access key valid for your subscription.
3. Replace the location in `Endpoint` to the endpoint you signed up for. You can find the endpoint on Azure Portal resource. The endpoint typically starts with "https://[region].api.cognitive.microsoft.com", and in here please only include protocol and hostname.
4. Run the program.

```
using System;
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics;
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics.Models;
using System.Collections.Generic;
using Microsoft.Rest;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        /// <summary>
        /// Container for subscription credentials. Make sure to enter your valid key.
        string subscriptionKey = ""; //Insert your Text Anaytics subscription key
        /// </summary>
        class ApiKeyServiceClientCredentials : ServiceClientCredentials
        {
            public override Task ProcessHttpRequestAsync(HttpRequestMessage request, CancellationToken
cancellationToken)
            {
                request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
                return base.ProcessHttpRequestAsync(request, cancellationToken);
            }
        }

        static void Main(string[] args)
        {

            // Create a client.
            ITextAnalyticsClient client = new TextAnalyticsClient(new ApiKeyServiceClientCredentials())
            {
                Endpoint = "https://westus.api.cognitive.microsoft.com"
            }; //Replace 'westus' with the correct region for your Text Analytics subscription

            Console.OutputEncoding = System.Text.Encoding.UTF8;

            // Extracting language
            Console.WriteLine("===== LANGUAGE EXTRACTION ======");

            var result = client.DetectLanguageAsync(new BatchInput(
                    new List<Input>()
                        {
                          new Input("1", "This is a document written in English."),
                          new Input("2", "Este es un document escrito en Español."),
                          new Input("3", "这是一个用中文写的文件")
                    })).Result;

            // Printing language results.
            foreach (var document in result.Documents)
            {
                Console.WriteLine("Document ID: {0} , Language: {1}", document.Id,
document.DetectedLanguages[0].Name);
            }

            // Getting key-phrases
            Console.WriteLine("\n\n===== KEY-PHRASE EXTRACTION ======");

            KeyPhraseBatchResult result2 = client.KeyPhrasesAsync(new MultiLanguageBatchInput(
                        new List<MultiLanguageInput>()
                        {
                          new MultiLanguageInput("ja", "1", "猫は幸せ"),
                          new MultiLanguageInput("de", "2", "Fahrt nach Stuttgart und dann zum Hotel zu Fu."),
                          new MultiLanguageInput("en", "3", "My cat is stiff as a rock."),
                          new MultiLanguageInput("es", "4", "A mi me encanta el fútbol!")
                        })).Result;

            // Printing keyphrases
            foreach (var document in result2.Documents)
            {
                Console.WriteLine("Document ID: {0} ", document.Id);

                Console.WriteLine("\t Key phrases:");
                foreach (string keyphrase in document.KeyPhrases)
                {
                    Console.WriteLine("\t\t" + keyphrase);
                }
            }

            // Extracting sentiment
            Console.WriteLine("\n\n===== SENTIMENT ANALYSIS ======");

            SentimentBatchResult result3 = client.SentimentAsync(
                    new MultiLanguageBatchInput(
                        new List<MultiLanguageInput>()
                        {
                          new MultiLanguageInput("en", "0", "I had the best day of my life."),
                          new MultiLanguageInput("en", "1", "This was a waste of my time. The speaker put me
to sleep."),
                          new MultiLanguageInput("es", "2", "No tengo dinero ni nada que dar..."),
                          new MultiLanguageInput("it", "3", "L'hotel veneziano era meraviglioso. È un
bellissimo pezzo di architettura."),
                        })).Result;


            // Printing sentiment results
            foreach (var document in result3.Documents)
            {
                Console.WriteLine("Document ID: {0} , Sentiment Score: {1:0.00}", document.Id,
document.Score);
            }


            // Identify entities
            Console.WriteLine("\n\n===== ENTITIES ======");

            EntitiesBatchResult result4 = client.EntitiesAsync(
                    new MultiLanguageBatchInput(
                        new List<MultiLanguageInput>()
                        {
                          new MultiLanguageInput("en", "0", "The Great Depression began in 1929. By 1933, the
GDP in America fell by 25%.")
                        })).Result;

            // Printing entities results
            foreach (var document in result4.Documents)
            {
                Console.WriteLine("Document ID: {0} ", document.Id);

                Console.WriteLine("\t Entities:");

                foreach (EntityRecord entity in document.Entities)
                {
                    Console.WriteLine("\t\t" + entity.Name);
                }
            }

            Console.ReadLine();
        }
    }
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using Go to call the Text Analytics Cognitive Service

10/3/2018 • 10 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, extract key phrases, and identify linked entities using the Text Analytics APIs with Go.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart.

You must also have the endpoint and access key that was generated for you during sign-up.

Detect language request

The Language Detection API detects the language of a text document, using the Detect Language method.

1. Create a new Go project in your favorite code editor.
2. Add the code provided below.
3. Replace the `subscriptionKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westcentralus`) to the region you signed up for.
5. Save the file with a '.go' extension.
6. Open a command prompt on a computer with Go installed.
7. Build the file, for example: 'go build quickstart.go'.
8. Run the file, for example: 'quickstart'.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Replace the subscriptionKey string value with your valid subscription key
    const subscriptionKey = "<Subscription Key>"

    /*
    Replace or verify the region.

    You must use the same region in your REST API call as you used to obtain your access keys.
    For example, if you obtained your access keys from the westus region, replace
    "westcentralus" in the URI below with "westus".

    NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
    a free trial access key, you should not need to change this region.
    */
    const uriBase =    "https://westcentralus.api.cognitive.microsoft.com"
    const uriBase =     "https://westcentralus.api.cognitive.microsoft.com
    const uriPath = "/text/analytics/v2.0/languages"

    const uri = uriBase + uriPath

    data := []map[string]string{
        {"id": "1", "text": "This is a document written in English."},
        {"id": "2", "text": "Este es un document escrito en Español."},
        {"id": "3", "text": "这是一个用中文写的文件"},
    }

    documents, err := json.Marshal(&data)
    if err != nil {
        fmt.Printf("Error marshaling data: %v\n", err)
        return
    }

    r := strings.NewReader("{\"documents\": " + string(documents) + "}")

    client := &http.Client{
        Timeout: time.Second * 2,
    }

    req, err := http.NewRequest("POST", uri, r)
    if err != nil {
        fmt.Printf("Error creating request: %v\n", err)
        return
    }

    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

    resp, err := client.Do(req)
    if err != nil {
        fmt.Printf("Error on request: %v\n", err)
        return
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Printf("Error reading response body: %v\n", err)
        return
    }

    var f interface{}
    json.Unmarshal(body, &f)

    jsonFormatted, err := json.MarshalIndent(f, "", "  ")
    if err != nil {
        fmt.Printf("Error producing JSON: %v\n", err)
        return
    }
    fmt.Println(string(jsonFormatted))
}
```

Detect language response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "id": "1",
         "detectedLanguages": [
            {
               "name": "English",
               "iso6391Name": "en",
               "score": 1.0
            }
         ]
      },
      {
         "id": "2",
         "detectedLanguages": [
            {
               "name": "Spanish",
               "iso6391Name": "es",
               "score": 1.0
            }
         ]
      },
      {
         "id": "3",
         "detectedLanguages": [
            {
               "name": "Chinese_Simplified",
               "iso6391Name": "zh_chs",
               "score": 1.0
            }
         ]
      }
   ],
   "errors": [

   ]
}
```

Analyze sentiment request

The Sentiment Analysis API detects the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

1. Create a new Go project in your favorite code editor.
2. Add the code provided below.
3. Replace the `subscriptionKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westcentralus`) to the region you signed up for.
5. Save the file with a '.go' extension.
6. Open a command prompt on a computer with Go installed.
7. Build the file, for example: 'go build quickstart.go'.
8. Run the file, for example: 'quickstart'.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Replace the subscriptionKey string value with your valid subscription key
    const subscriptionKey = "<Subscription Key>"

    /*
    Replace or verify the region.

    You must use the same region in your REST API call as you used to obtain your access keys.
    For example, if you obtained your access keys from the westus region, replace
    "westcentralus" in the URI below with "westus".

    NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
    a free trial access key, you should not need to change this region.
    */
    const uriBase =    "https://westcentralus.api.cognitive.microsoft.com"
    const uriPath = "/text/analytics/v2.0/sentiment"

    const uri = uriBase + uriPath

    data := []map[string]string{
        {"id": "1", "language": "en", "text": "I really enjoy the new XBox One S. It has a clean look, it has
4K/HDR resolution and it is affordable."},
        {"id": "2", "language": "es", "text": "Este ha sido un dia terrible, llegué tarde al trabajo debido a
un accidente automobilistico."},
    }

    documents, err := json.Marshal(&data)
    if err != nil {
        fmt.Printf("Error marshaling data: %v\n", err)
        return
    }

    r := strings.NewReader("{\"documents\": " + string(documents) + "}")

    client := &http.Client{
        Timeout: time.Second * 2,
    }

    req, err := http.NewRequest("POST", uri, r)
    if err != nil {
        fmt.Printf("Error creating request: %v\n", err)
        return
    }

    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

    resp, err := client.Do(req)
    if err != nil {
        fmt.Printf("Error on request: %v\n", err)
        return
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Printf("Error reading response body: %v\n", err)
        return
    }

    var f interface{}
    json.Unmarshal(body, &f)

    jsonFormatted, err := json.MarshalIndent(f, "", "  ")
    if err != nil {
        fmt.Printf("Error producing JSON: %v\n", err)
        return
    }
    fmt.Println(string(jsonFormatted))
}
```

Analyze sentiment response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "score": 0.99984133243560791,
         "id": "1"
      },
      {
         "score": 0.024017512798309326,
         "id": "2"
      },
   ],
   "errors": [   ]
}
```

Extract key phrases request

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. The following example extracts key phrases for both English and Spanish documents.

1. Create a new Go project in your favorite code editor.
2. Add the code provided below.
3. Replace the `subscriptionKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westcentralus`) to the region you signed up for.
5. Save the file with a '.go' extension.
6. Open a command prompt on a computer with Go installed.
7. Build the file, for example: 'go build quickstart.go'.
8. Run the file, for example: 'quickstart'.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Replace the subscriptionKey string value with your valid subscription key
    const subscriptionKey = "<Subscription Key>"

    /*
    Replace or verify the region.

    You must use the same region in your REST API call as you used to obtain your access keys.
    For example, if you obtained your access keys from the westus region, replace
    "westcentralus" in the URI below with "westus".
    NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
    a free trial access key, you should not need to change this region.
    */
    const uriBase =    "https://westcentralus.api.cognitive.microsoft.com"
    const uriPath = "/text/analytics/v2.0/keyPhrases"

    const uri = uriBase + uriPath

    data := []map[string]string{
        {"id": "1", "language": "en", "text": "I really enjoy the new XBox One S. It has a clean look, it has
4K/HDR resolution and it is affordable."},
        {"id": "2", "language": "es", "text": "Si usted quiere comunicarse con Carlos, usted debe de llamarlo a
su telefono movil. Carlos es muy responsable, pero necesita recibir una notificacion si hay algun problema."},
        {"id": "3", "language": "en", "text": "The Grand Hotel is a new hotel in the center of Seattle. It
earned 5 stars in my review, and has the classiest decor I've ever seen."},
    }

    documents, err := json.Marshal(&data)
    if err != nil {
        fmt.Printf("Error marshaling data: %v\n", err)
        return
    }

    r := strings.NewReader("{\"documents\": " + string(documents) + "}")

    client := &http.Client{
        Timeout: time.Second * 2,
    }

    req, err := http.NewRequest("POST", uri, r)
    if err != nil {
        fmt.Printf("Error creating request: %v\n", err)
        return
    }

    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

    resp, err := client.Do(req)
    if err != nil {
        fmt.Printf("Error on request: %v\n", err)
        return
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Printf("Error reading response body: %v\n", err)
        return
    }

    var f interface{}
    json.Unmarshal(body, &f)

    jsonFormatted, err := json.MarshalIndent(f, "", "  ")
    if err != nil {
        fmt.Printf("Error producing JSON: %v\n", err)
        return
    }
    fmt.Println(string(jsonFormatted))
}
```

Extract key phrases response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "keyPhrases": [
            "HDR resolution",
            "new XBox",
            "clean look"
         ],
         "id": "1"
      },
      {
         "keyPhrases": [
            "Carlos",
            "notificacion",
            "algun problema",
            "telefono movil"
         ],
         "id": "2"
      },
      {
         "keyPhrases": [
            "new hotel",
            "Grand Hotel",
            "review",
            "center of Seattle",
            "classiest decor",
            "stars"
         ],
         "id": "3"
      }
   ],
   "errors": [  ]
}
```

Identify entities request

The Entities API identifies well-known entities in a text document, using the Entities method. The following example identifies entities for English documents.

1. Create a new Go project in your favorite code editor.
2. Add the code provided below.
3. Replace the `subscriptionKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westcentralus`) to the region you signed up for.
5. Save the file with a '.go' extension.
6. Open a command prompt on a computer with Go installed.
7. Build the file, for example: 'go build quickstart.go'.
8. Run the file, for example: 'quickstart'.

```
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
    "strings"
    "time"
)

func main() {
    // Replace the subscriptionKey string value with your valid subscription key
    const subscriptionKey = "<Subscription Key>"

    /*
    Replace or verify the region.

    You must use the same region in your REST API call as you used to obtain your access keys.
    For example, if you obtained your access keys from the westus region, replace
    "westus" in the URI below with "westcentralus".

    NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
    a free trial access key, you should not need to change this region.
    */
    const uriBase =    "https://westus.api.cognitive.microsoft.com"
    const uriPath = "/text/analytics/v2.1-preview/entities"

    const uri = uriBase + uriPath

    data := []map[string]string{
        {"id": "1", "language": "en", "text": "Jeff bought three dozen eggs because there was a 50%
discount."},
        {"id": "2", "language": "en", "text": "The Great Depression began in 1929. By 1933, the GDP in America
fell by 25%."},
    }

    documents, err := json.Marshal(&data)
    if err != nil {
        fmt.Printf("Error marshaling data: %v\n", err)
        return
    }

    r := strings.NewReader("{\"documents\": " + string(documents) + "}")

    client := &http.Client{
        Timeout: time.Second * 2,
    }

    req, err := http.NewRequest("POST", uri, r)
    if err != nil {
        fmt.Printf("Error creating request: %v\n", err)
        return
    }

    req.Header.Add("Content-Type", "application/json")
    req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)

    resp, err := client.Do(req)
    if err != nil {
        fmt.Printf("Error on request: %v\n", err)
        return
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Printf("Error reading response body: %v\n", err)
        return
    }

    var f interface{}
    json.Unmarshal(body, &f)

    jsonFormatted, err := json.MarshalIndent(f, "", "  ")
    if err != nil {
        fmt.Printf("Error producing JSON: %v\n", err)
        return
    }
    fmt.Println(string(jsonFormatted))
}
```

Entity extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                        "Offset": 0,
                        "Length": 20
                    }
                ],
                "WikipediaLanguage": "en",
                "WikipediaId": "Great Depression",
                "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
            },
            {
                "Name": "1929",
                "Matches": [
                    {
                        "Text": "1929",
                        "Offset": 30,
                        "Length": 4
                    }
                ],
                "Type": "DateTime",
                "SubType": "DateRange"
            },
            {
                "Name": "By 1933",
                "Matches": [
                    {
                        "Text": "By 1933",
                        "Offset": 36,
                        "Length": 7
                    }
                ],
                "Type": "DateTime",
                "SubType": "DateRange"
            },
            {
                "Name": "Gross domestic product",
                "Matches": [
                    {
                        "Text": "GDP",
                        "Offset": 49,
                        "Length": 3
                    }
                ],
                "WikipediaLanguage": "en",
                "WikipediaId": "Gross domestic product",
                "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
            },
            {
                "Name": "United States",
                "Matches": [
                    {
                        "Text": "America",
                        "Offset": 56,
                        "Length": 7
                    }
                ],
                "WikipediaLanguage": "en",
                "WikipediaId": "United States",
                "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                "Type": "Location"
            },
            {
                "Name": "25",
                "Matches": [
                    {
                        "Text": "25",
                        "Offset": 72,
                        "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using Java to call the Text Analytics Cognitive Service

10/3/2018 • 12 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, extract key phrases, and identify linked entities using the Text Analytics APIs with Java.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart. You must also have the endpoint and access key that was generated for you during sign up.

Detect language

The Language Detection API detects the language of a text document, using the Detect Language method.

1. Create a new Java project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `host` (currently `westus`) to the region you signed up for.
5. Run the program.

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.HttpsURLConnection;

/*
 * Gson: https://github.com/google/gson
 * Maven info:
 *     groupId: com.google.code.gson
 *     artifactId: gson
 *     version: 2.8.1
 *
 * Once you have compiled or downloaded gson-2.8.1.jar, assuming you have placed it in the
 * same folder as this file (DetectLanguage.java), you can compile and run this program at
 * the command line as follows.
 *
 * javac DetectLanguage.java -classpath .;gson-2.8.1.jar -encoding UTF-8
 * java -cp .;gson-2.8.1.jar DetectLanguage
 */
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

class Document {
    public String id, text;

    public Document(String id, String text){
        this.id = id;
        this.text = text;
    }
```

```java
}

class Documents {
    public List<Document> documents;

    public Documents() {
        this.documents = new ArrayList<Document>();
    }
    public void add(String id, String text) {
        this.documents.add (new Document (id, text));
    }
}

public class DetectLanguage {

// ***********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
    static String accessKey = "enter key here";

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
    static String host = "https://westus.api.cognitive.microsoft.com";

    static String path = "/text/analytics/v2.0/languages";

    public static String GetLanguage (Documents documents) throws Exception {
        String text = new Gson().toJson(documents);
        byte[] encoded_text = text.getBytes("UTF-8");

        URL url = new URL(host+path);
        HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "text/json");
        connection.setRequestProperty("Ocp-Apim-Subscription-Key", accessKey);
        connection.setDoOutput(true);

        DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
        wr.write(encoded_text, 0, encoded_text.length);
        wr.flush();
        wr.close();

        StringBuilder response = new StringBuilder ();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();

        return response.toString();
    }

    public static String prettify(String json_text) {
        JsonParser parser = new JsonParser();
        JsonObject json = parser.parse(json_text).getAsJsonObject();
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        return gson.toJson(json);
    }
    public static void main (String[] args) {
        try {
            Documents documents = new Documents ();
            documents.add ("1", "This is a document written in English.");
            documents.add ("2", "Este es un document escrito en Español.");
            documents.add ("3", "这是一个用中文写的文件");

            String response = GetLanguage (documents);
            System.out.println (prettify (response));
        }
        catch (Exception e) {
            System.out.println (e);
        }
    }
}
```

Language detection response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "id": "1",
         "detectedLanguages": [
            {
               "name": "English",
               "iso6391Name": "en",
               "score": 1.0
            }
         ]
      },
      {
         "id": "2",
         "detectedLanguages": [
            {
               "name": "Spanish",
               "iso6391Name": "es",
               "score": 1.0
            }
         ]
      },
      {
         "id": "3",
         "detectedLanguages": [
            {
               "name": "Chinese_Simplified",
               "iso6391Name": "zh_chs",
               "score": 1.0
            }
         ]
      }
   ],
   "errors": [

   ]
}
```

Analyze sentiment

The Sentiment Analysis API detexts the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

1. Create a new Java project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westus`) to the region you signed up for.
5. Run the program.

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.HttpsURLConnection;

/*
 * Gson: https://github.com/google/gson
 * Maven info:
 *     groupId: com.google.code.gson
 *     artifactId: gson
 *     version: 2.8.1
 *
 * Once you have compiled or downloaded gson-2.8.1.jar, assuming you have placed it in the
 * same folder as this file (GetSentiment.java), you can compile and run this program at
 * the command line as follows.
 *
 * javac GetSentiment.java -classpath .;gson-2.8.1.jar -encoding UTF-8
 * java -cp .;gson-2.8.1.jar GetSentiment
 */
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

class Document {
    public String id, language, text;

    public Document(String id, String language, String text){
        this.id = id;
        this.language = language;
        this.text = text;
    }
}

class Documents {
    public List<Document> documents;

    public Documents() {
        this.documents = new ArrayList<Document>();
    }
    public void add(String id, String language, String text) {
        this.documents.add (new Document (id, language, text));
    }
}

public class GetSentiment {

// ***********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
    static String accessKey = "enter key here";

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".
// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
    static String host = "https://westus.api.cognitive.microsoft.com";

    static String path = "/text/analytics/v2.0/sentiment";

    public static String GetSentiment (Documents documents) throws Exception {
        String text = new Gson().toJson(documents);
        byte[] encoded_text = text.getBytes("UTF-8");

        URL url = new URL(host+path);
        HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "text/json");
        connection.setRequestProperty("Ocp-Apim-Subscription-Key", accessKey);
        connection.setDoOutput(true);

        DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
        wr.write(encoded_text, 0, encoded_text.length);
        wr.flush();
        wr.close();

        StringBuilder response = new StringBuilder ();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();

        return response.toString();
    }

    public static String prettify(String json_text) {
        JsonParser parser = new JsonParser();
        JsonObject json = parser.parse(json_text).getAsJsonObject();
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        return gson.toJson(json);
    }

    public static void main (String[] args) {
        try {
            Documents documents = new Documents ();
            documents.add ("1", "en", "I really enjoy the new XBox One S. It has a clean look, it has 4K/HDR
resolution and it is affordable.");
            documents.add ("2", "es", "Este ha sido un dia terrible, llegué tarde al trabajo debido a un
accidente automobilistico.");

            String response = GetSentiment (documents);
            System.out.println (prettify (response));
        }
        catch (Exception e) {
            System.out.println (e);
        }
    }
}
```

Sentiment analysis response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "score": 0.999841332435060791,
         "id": "1"
      },
      {
         "score": 0.024017512798309326,
         "id": "2"
      },
   ],
   "errors": [   ]
}
```

Extract key phrases

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. The following example extracts Key phrases for both English and Spanish documents.

1. Create a new Java project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westus`) to the region you signed up for.
5. Run the program.

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.HttpsURLConnection;

/*
 * Gson: https://github.com/google/gson
 * Maven info:
 *     groupId: com.google.code.gson
 *     artifactId: gson
 *     version: 2.8.1
 *
 * Once you have compiled or downloaded gson-2.8.1.jar, assuming you have placed it in the
 * same folder as this file (GetKeyPhrases.java), you can compile and run this program at
 * the command line as follows.
 *
 * javac GetKeyPhrases.java -classpath .;gson-2.8.1.jar -encoding UTF-8
 * java -cp .;gson-2.8.1.jar GetKeyPhrases
 */
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

class Document {
    public String id, language, text;

    public Document(String id, String language, String text){
        this.id = id;
        this.language = language;
        this.text = text;
    }
}

class Documents {
    public List<Document> documents;

    public Documents() {
```

```
    public Documents() {
        this.documents = new ArrayList<Document>();
    }
    public void add(String id, String language, String text) {
        this.documents.add (new Document (id, language, text));
    }
}

public class GetKeyPhrases {

// ***********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
    static String accessKey = "enter key here";

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
    static String host = "https://westus.api.cognitive.microsoft.com";

    static String path = "/text/analytics/v2.0/keyPhrases";

    public static String GetKeyPhrases (Documents documents) throws Exception {
        String text = new Gson().toJson(documents);
        byte[] encoded_text = text.getBytes("UTF-8");

        URL url = new URL(host+path);
        HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "text/json");
        connection.setRequestProperty("Ocp-Apim-Subscription-Key", accessKey);
        connection.setDoOutput(true);

        DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
        wr.write(encoded_text, 0, encoded_text.length);
        wr.flush();
        wr.close();

        StringBuilder response = new StringBuilder ();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();

        return response.toString();
    }

    public static String prettify(String json_text) {
        JsonParser parser = new JsonParser();
        JsonObject json = parser.parse(json_text).getAsJsonObject();
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        return gson.toJson(json);
    }

    public static void main (String[] args) {
        try {
            Documents documents = new Documents ();
            documents.add ("1", "en", "I really enjoy the new XBox One S. It has a clean look, it has 4K/HDR
resolution and it is affordable.");
```

```
            documents.add ("2", "es", "Si usted quiere comunicarse con Carlos, usted debe de llamarlo a su
telefono movil. Carlos es muy responsable, pero necesita recibir una notificacion si hay algun problema.");
            documents.add ("3", "en", "The Grand Hotel is a new hotel in the center of Seattle. It earned 5
stars in my review, and has the classiest decor I've ever seen.");

            String response = GetKeyPhrases (documents);
            System.out.println (prettify (response));
        }
        catch (Exception e) {
            System.out.println (e);
        }
    }
}
```

Key phrase extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "keyPhrases": [
            "HDR resolution",
            "new XBox",
            "clean look"
         ],
         "id": "1"
      },
      {
         "keyPhrases": [
            "Carlos",
            "notificacion",
            "algun problema",
            "telefono movil"
         ],
         "id": "2"
      },
      {
         "keyPhrases": [
            "new hotel",
            "Grand Hotel",
            "review",
            "center of Seattle",
            "classiest decor",
            "stars"
         ],
         "id": "3"
      }
   ],
   "errors": [  ]
}
```

Identify entities

The Entities API identifies well-known entities in a text document, using the Entities method. The following example identifies entities for English documents.

1. Create a new Java project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uriBase` (currently `westus`) to the region you signed up for.
5. Run the program.

```java
import java.io.*;
import java.net.*;
import java.util.*;
import javax.net.ssl.HttpsURLConnection;

/*
 * Gson: https://github.com/google/gson
 * Maven info:
 *     groupId: com.google.code.gson
 *     artifactId: gson
 *     version: 2.8.1
 *
 * Once you have compiled or downloaded gson-2.8.1.jar, assuming you have placed it in the
 * same folder as this file (GetEntities.java), you can compile and run this program at
 * the command line as follows.
 *
 * javac GetEntities.java -classpath .;gson-2.8.1.jar -encoding UTF-8
 * java -cp .;gson-2.8.1.jar GetEntities
 */
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

class Document {
    public String id, language, text;

    public Document(String id, String language, String text){
        this.id = id;
        this.language = language;
        this.text = text;
    }
}

class Documents {
    public List<Document> documents;

    public Documents() {
        this.documents = new ArrayList<Document>();
    }
    public void add(String id, String language, String text) {
        this.documents.add (new Document (id, language, text));
    }
}

public class GetEntities {

// ***********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
    static String accessKey = "enter key here";

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
    static String host = "https://westus.api.cognitive.microsoft.com";

    static String path = "/text/analytics/v2.1-preview/entities";

    public static String GetEntities (Documents documents) throws Exception {
        String text = new Gson().toJson(documents);
        byte[] encoded_text = text.getBytes("UTF-8");

        URL url = new URL(host+path);
        HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setRequestProperty("Content-Type", "text/json");
        connection.setRequestProperty("Ocp-Apim-Subscription-Key", accessKey);
        connection.setDoOutput(true);

        DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
        wr.write(encoded_text, 0, encoded_text.length);
        wr.flush();
        wr.close();

        StringBuilder response = new StringBuilder ();
        BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));
        String line;
        while ((line = in.readLine()) != null) {
            response.append(line);
        }
        in.close();

        return response.toString();
    }

    public static String prettify(String json_text) {
        JsonParser parser = new JsonParser();
        JsonObject json = parser.parse(json_text).getAsJsonObject();
        Gson gson = new GsonBuilder().setPrettyPrinting().create();
        return gson.toJson(json);
    }

    public static void main (String[] args) {
        try {
            Documents documents = new Documents ();
            documents.add ("1", "en", "Jeff bought three dozen eggs because there was a 50% discount.");
            documents.add ("2", "en", "The Great Depression began in 1929. By 1933, the GDP in America fell by
25%.");

            String response = GetEntities (documents);
            System.out.println (prettify (response));
        }
        catch (Exception e) {
            System.out.println (e);
        }
    }
}
```

Entity extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                            "Offset": 0,
                            "Length": 20
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Great Depression",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                    "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
                },
                {
                    "Name": "1929",
                    "Matches": [
                        {
                            "Text": "1929",
                            "Offset": 30,
                            "Length": 4
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                },
                {
                    "Name": "By 1933",
                    "Matches": [
                        {
                            "Text": "By 1933",
                            "Offset": 36,
                            "Length": 7
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
    "errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using Node.js to call the Text Analytics Cognitive Service

10/3/2018 • 9 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, extract key phrases, and identify linked entities using the Text Analytics APIs with Node.JS.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart.

You must also have the endpoint and access key that was generated for you during sign up.

Detect language

The Language Detection API detects the language of a text document, using the Detect Language method.

1. Create a new Node.JS project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
'use strict';

let https = require ('https');

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
let accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
let uri = 'westus.api.cognitive.microsoft.com';
let path = '/text/analytics/v2.0/languages';

let response_handler = function (response) {
    let body = '';
    response.on ('data', function (d) {
        body += d;
    });
    response.on ('end', function () {
        let body_ = JSON.parse (body);
        let body__ = JSON.stringify (body_, null, '  ');
        console.log (body__);
    });
    response.on ('error', function (e) {
        console.log ('Error: ' + e.message);
    });
};

let get_language = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

let documents = { 'documents': [
    { 'id': '1', 'text': 'This is a document written in English.' },
    { 'id': '2', 'text': 'Este es un document escrito en Español.' },
    { 'id': '3', 'text': '这是一个用中文写的文件' }
]};

get_language (documents);
```

Language detection response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "id": "1",
         "detectedLanguages": [
            {
               "name": "English",
               "iso6391Name": "en",
               "score": 1.0
            }
         ]
      },
      {
         "id": "2",
         "detectedLanguages": [
            {
               "name": "Spanish",
               "iso6391Name": "es",
               "score": 1.0
            }
         ]
      },
      {
         "id": "3",
         "detectedLanguages": [
            {
               "name": "Chinese_Simplified",
               "iso6391Name": "zh_chs",
               "score": 1.0
            }
         ]
      }
   ],
   "errors": [

   ]
}
```

Analyze sentiment

The Sentiment Analysis API detexts the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

1. Create a new Node.JS project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
'use strict';

let https = require ('https');

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
let accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
let uri = 'westus.api.cognitive.microsoft.com';
let path = '/text/analytics/v2.0/sentiment';

let response_handler = function (response) {
    let body = '';
    response.on ('data', function (d) {
        body += d;
    });
    response.on ('end', function () {
        let body_ = JSON.parse (body);
        let body__ = JSON.stringify (body_, null, '  ');
        console.log (body__);
    });
    response.on ('error', function (e) {
        console.log ('Error: ' + e.message);
    });
};

let get_sentiments = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

let documents = { 'documents': [
    { 'id': '1', 'language': 'en', 'text': 'I really enjoy the new XBox One S. It has a clean look, it has 
4K/HDR resolution and it is affordable.' },
    { 'id': '2', 'language': 'es', 'text': 'Este ha sido un dia terrible, llegué tarde al trabajo debido a un 
accidente automobilistico.' },
]};

get_sentiments (documents);
```

Sentiment analysis response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "score": 0.999841332435607 91,
         "id": "1"
      },
      {
         "score": 0.024017512798309326,
         "id": "2"
      },
   ],
   "errors": [   ]
}
```

Extract key phrases

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. The following example extracts key phrases for both English and Spanish documents.

1. Create a new Node.JS project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
'use strict';

let https = require ('https');

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
let accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
let uri = 'westus.api.cognitive.microsoft.com';
let path = '/text/analytics/v2.0/keyPhrases';

let response_handler = function (response) {
    let body = '';
    response.on ('data', function (d) {
        body += d;
    });
    response.on ('end', function () {
        let body_ = JSON.parse (body);
        let body__ = JSON.stringify (body_, null, '  ');
        console.log (body__);
    });
    response.on ('error', function (e) {
        console.log ('Error: ' + e.message);
    });
};

let get_key_phrases = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

let documents = { 'documents': [
    { 'id': '1', 'language': 'en', 'text': 'I really enjoy the new XBox One S. It has a clean look, it has
4K/HDR resolution and it is affordable.' },
    { 'id': '2', 'language': 'es', 'text': 'Si usted quiere comunicarse con Carlos, usted debe de llamarlo a su
telefono movil. Carlos es muy responsable, pero necesita recibir una notificacion si hay algun problema.' },
    { 'id': '3', 'language': 'en', 'text': 'The Grand Hotel is a new hotel in the center of Seattle. It earned
5 stars in my review, and has the classiest decor I\'ve ever seen.' }
]};

get_key_phrases (documents);
```

Key phrase extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "keyPhrases": [
            "HDR resolution",
            "new XBox",
            "clean look"
         ],
         "id": "1"
      },
      {
         "keyPhrases": [
            "Carlos",
            "notificacion",
            "algun problema",
            "telefono movil"
         ],
         "id": "2"
      },
      {
         "keyPhrases": [
            "new hotel",
            "Grand Hotel",
            "review",
            "center of Seattle",
            "classiest decor",
            "stars"
         ],
         "id": "3"
      }
   ],
   "errors": [  ]
}
```

Identify linked entities

The Entities API identifies well-known entities in a text document, using the Entities method. The following example identifies entities for English documents.

1. Create a new Node.JS project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
'use strict';

let https = require ('https');

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
let accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
let uri = 'westus.api.cognitive.microsoft.com';
let path = '/text/analytics/v2.1-preview/entities';

let response_handler = function (response) {
    let body = '';
    response.on ('data', function (d) {
        body += d;
    });
    response.on ('end', function () {
        let body_ = JSON.parse (body);
        let body__ = JSON.stringify (body_, null, '  ');
        console.log (body__);
    });
    response.on ('error', function (e) {
        console.log ('Error: ' + e.message);
    });
};

let get_entities = function (documents) {
    let body = JSON.stringify (documents);

    let request_params = {
        method : 'POST',
        hostname : uri,
        path : path,
        headers : {
            'Ocp-Apim-Subscription-Key' : accessKey,
        }
    };

    let req = https.request (request_params, response_handler);
    req.write (body);
    req.end ();
}

let documents = { 'documents': [
    { 'id': '1', 'language': 'en', 'text': 'Jeff bought three dozen eggs because there was a 50% discount.' },
    { 'id': '2', 'language': 'en', 'text': 'The Great Depression began in 1929. By 1933, the GDP in America
fell by 25%.' }
]};

get_entities (documents);
```

Entity extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                            "Offset": 0,
                            "Length": 20
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Great Depression",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                    "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
                },
                {
                    "Name": "1929",
                    "Matches": [
                        {
                            "Text": "1929",
                            "Offset": 30,
                            "Length": 4
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "By 1933",
                    "Matches": [
                        {
                            "Text": "By 1933",
                            "Offset": 36,
                            "Length": 7
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using PHP to call the Text Analytics Cognitive Service

10/3/2018 • 9 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, extract key phrases, and identify linked entities using the Text Analytics APIs with PHP.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart.

You must also have the endpoint and access key that was generated for you during sign up.

Detect language

The Language Detection API detects the language of a text document, using the Detect Language method.

1. Create a new PHP project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `host` (currently `westus`) to the region you signed up for.
5. Run the program.

```php
<?php

// NOTE: Be sure to uncomment the following line in your php.ini file.
// ;extension=php_openssl.dll

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
$accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
$host = 'https://westus.api.cognitive.microsoft.com';
$path = '/text/analytics/v2.0/languages';

function DetectLanguage ($host, $path, $key, $data) {

    $headers = "Content-type: text/json\r\n" .
        "Ocp-Apim-Subscription-Key: $key\r\n";

    $data = json_encode ($data);

    // NOTE: Use the key 'http' even if you are making an HTTPS request. See:
    // http://php.net/manual/en/function.stream-context-create.php
    $options = array (
        'http' => array (
            'header' => $headers,
            'method' => 'POST',
            'content' => $data
        )
    );
    $context  = stream_context_create ($options);
    $result = file_get_contents ($host . $path, false, $context);
    return $result;
}

$data = array (
    'documents' => array (
        array ( 'id' => '1', 'text' => 'This is a document written in English.' ),
        array ( 'id' => '2', 'text' => 'Este es un document escrito en Español.' ),
        array ( 'id' => '3', 'text' => '这是一个用中文写的文件')
    )
);

print "Please wait a moment for the results to appear.";

$result = DetectLanguage ($host, $path, $accessKey, $data);

echo json_encode (json_decode ($result), JSON_PRETTY_PRINT);
?>
```

Language detection response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "id": "1",
         "detectedLanguages": [
            {
               "name": "English",
               "iso6391Name": "en",
               "score": 1.0
            }
         ]
      },
      {
         "id": "2",
         "detectedLanguages": [
            {
               "name": "Spanish",
               "iso6391Name": "es",
               "score": 1.0
            }
         ]
      },
      {
         "id": "3",
         "detectedLanguages": [
            {
               "name": "Chinese_Simplified",
               "iso6391Name": "zh_chs",
               "score": 1.0
            }
         ]
      }
   ],
   "errors": [

   ]
}
```

Analyze sentiment

The Sentiment Analysis API detexts the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

1. Create a new PHP project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `host` (currently `westus`) to the region you signed up for.
5. Run the program.

```php
<?php

// NOTE: Be sure to uncomment the following line in your php.ini file.
// ;extension=php_openssl.dll

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
$accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
$host = 'https://westus.api.cognitive.microsoft.com';
$path = '/text/analytics/v2.0/sentiment';

function GetSentiment ($host, $path, $key, $data) {

    $headers = "Content-type: text/json\r\n" .
        "Ocp-Apim-Subscription-Key: $key\r\n";

    $data = json_encode ($data);

    // NOTE: Use the key 'http' even if you are making an HTTPS request. See:
    // http://php.net/manual/en/function.stream-context-create.php
    $options = array (
        'http' => array (
            'header' => $headers,
            'method' => 'POST',
            'content' => $data
        )
    );
    $context  = stream_context_create ($options);
    $result = file_get_contents ($host . $path, false, $context);
    return $result;
}

$data = array (
    'documents' => array (
        array ( 'id' => '1', 'language' => 'en', 'text' => 'I really enjoy the new XBox One S. It has a clean
look, it has 4K/HDR resolution and it is affordable.' ),
        array ( 'id' => '2', 'language' => 'es', 'text' => 'Este ha sido un dia terrible, llegué tarde al
trabajo debido a un accidente automobilistico.' )
    )
);

print "Please wait a moment for the results to appear.";

$result = GetSentiment ($host, $path, $accessKey, $data);

echo json_encode (json_decode ($result), JSON_PRETTY_PRINT);
?>
```

Sentiment analysis response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "score": 0.99984133243560791,
         "id": "1"
      },
      {
         "score": 0.024017512798309326,
         "id": "2"
      },
   ],
   "errors": [   ]
}
```

Extract key phrases

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. The following example extracts key phrases for both English and Spanish documents.

1. Create a new PHP project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `host` (currently `westus`) to the region you signed up for.
5. Run the program.

```php
<?php

// NOTE: Be sure to uncomment the following line in your php.ini file.
// ;extension=php_openssl.dll

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
$accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
$host = 'https://westus.api.cognitive.microsoft.com';
$path = '/text/analytics/v2.0/keyPhrases';

function GetKeyPhrases ($host, $path, $key, $data) {

    $headers = "Content-type: text/json\r\n" .
        "Ocp-Apim-Subscription-Key: $key\r\n";

    $data = json_encode ($data);

    // NOTE: Use the key 'http' even if you are making an HTTPS request. See:
    // http://php.net/manual/en/function.stream-context-create.php
    $options = array (
        'http' => array (
            'header' => $headers,
            'method' => 'POST',
            'content' => $data
        )
    );
    $context  = stream_context_create ($options);
    $result = file_get_contents ($host . $path, false, $context);
    return $result;
}

$data = array (
    'documents' => array (
        array ( 'id' => '1', 'language' => 'en', 'text' => 'I really enjoy the new XBox One S. It has a clean
look, it has 4K/HDR resolution and it is affordable.' ),
        array ( 'id' => '2', 'language' => 'es', 'text' => 'Si usted quiere comunicarse con Carlos, usted debe
de llamarlo a su telefono movil. Carlos es muy responsable, pero necesita recibir una notificacion si hay algun
problema.' ),
        array ( 'id' => '3', 'language' => 'en', 'text' => 'The Grand Hotel is a new hotel in the center of
Seattle. It earned 5 stars in my review, and has the classiest decor I\'ve ever seen.' )
    )
);

print "Please wait a moment for the results to appear.";

$result = GetKeyPhrases ($host, $path, $accessKey, $data);

echo json_encode (json_decode ($result), JSON_PRETTY_PRINT);
?>
```

Key phrase extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "keyPhrases": [
            "HDR resolution",
            "new XBox",
            "clean look"
         ],
         "id": "1"
      },
      {
         "keyPhrases": [
            "Carlos",
            "notificacion",
            "algun problema",
            "telefono movil"
         ],
         "id": "2"
      },
      {
         "keyPhrases": [
            "new hotel",
            "Grand Hotel",
            "review",
            "center of Seattle",
            "classiest decor",
            "stars"
         ],
         "id": "3"
      }
   ],
   "errors": [  ]
}
```

Identify entities

The Entities API identifies well-known entities in a text document, using the Entities method. The following example identifies entities for English documents.

1. Create a new PHP project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `host` (currently `westus`) to the region you signed up for.
5. Run the program.

```php
<?php

// NOTE: Be sure to uncomment the following line in your php.ini file.
// ;extension=php_openssl.dll

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// Replace the accessKey string value with your valid access key.
$accessKey = 'enter key here';

// Replace or verify the region.

// You must use the same region in your REST API call as you used to obtain your access keys.
// For example, if you obtained your access keys from the westus region, replace
// "westcentralus" in the URI below with "westus".

// NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
// a free trial access key, you should not need to change this region.
$host = 'https://westus.api.cognitive.microsoft.com';
$path = '/text/analytics/v2.1-preview/entities';

function GetEntities ($host, $path, $key, $data) {

    $headers = "Content-type: text/json\r\n" .
        "Ocp-Apim-Subscription-Key: $key\r\n";

    $data = json_encode ($data);

    // NOTE: Use the key 'http' even if you are making an HTTPS request. See:
    // http://php.net/manual/en/function.stream-context-create.php
    $options = array (
        'http' => array (
            'header' => $headers,
            'method' => 'POST',
            'content' => $data
        )
    );
    $context  = stream_context_create ($options);
    $result = file_get_contents ($host . $path, false, $context);
    return $result;
}

$data = array (
    'documents' => array (
        array ( 'id' => '1', 'language' => 'en', 'text' => 'Jeff bought three dozen eggs because there was a
50% discount.' ),
        array ( 'id' => '2', 'language' => 'en', 'text' => 'The Great Depression began in 1929. By 1933, the
GDP in America fell by 25%.' )
    )
);

print "Please wait a moment for the results to appear.";

$result = GetEntities ($host, $path, $accessKey, $data);

echo json_encode (json_decode ($result), JSON_PRETTY_PRINT);
?>
```

Entity extraction response

A successful response is returned in JSON, as shown in the following example:

```json
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                            "Offset": 0,
                            "Length": 20
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Great Depression",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                    "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
                },
                },
                {
                    "Name": "1929",
                    "Matches": [
                        {
                            "Text": "1929",
                            "Offset": 30,
                            "Length": 4
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "By 1933",
                    "Matches": [
                        {
                            "Text": "By 1933",
                            "Offset": 36,
                            "Length": 7
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using Python to call the Text Analytics Cognitive Service

10/3/2018 • 6 minutes to read • Edit Online

This walkthrough shows you how to detect language, analyze sentiment, and extract key phrases using the Text Analytics APIs with Python.

You can run this example as a Jupyter notebook on MyBinder by clicking on the launch Binder badge:

launch binder

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this walkthrough.

You must also have the endpoint and access key that was generated for you during sign-up.

To continue with this walkthrough, replace `subscription_key` with a valid subscription key that you obtained earlier.

```
subscription_key = None
assert subscription_key
```

Next, verify that the region in `text_analytics_base_url` corresponds to the one you used when setting up the service. If you are using a free trial key, you do not need to change anything.

```
text_analytics_base_url = "https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/"
```

Detect languages

The Language Detection API detects the language of a text document, using the Detect Language method. The service endpoint of the language detection API for your region is available via the following URL:

```
language_api_url = text_analytics_base_url + "languages"
print(language_api_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/languages
```

The payload to the API consists of a list of `documents`, each of which in turn contains an `id` and a `text` attribute. The `text` attribute stores the text to be analyzed.

Replace the `documents` dictionary with any other text for language detection.

```
documents = { 'documents': [
    { 'id': '1', 'text': 'This is a document written in English.' },
    { 'id': '2', 'text': 'Este es un document escrito en Español.' },
    { 'id': '3', 'text': '这是一个用中文写的文件' }
]}
```

The next few lines of code call out to the language detection API using the `requests` library in Python to determine the language in the documents.

```
import requests
from pprint import pprint
headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
response  = requests.post(language_api_url, headers=headers, json=documents)
languages = response.json()
pprint(languages)
```

```
{'documents': [{'detectedLanguages': [{'iso6391Name': 'en',
                                       'name': 'English',
                                       'score': 1.0}],
                'id': '1'},
               {'detectedLanguages': [{'iso6391Name': 'es',
                                       'name': 'Spanish',
                                       'score': 1.0}],
                'id': '2'},
               {'detectedLanguages': [{'iso6391Name': 'zh_chs',
                                       'name': 'Chinese_Simplified',
                                       'score': 1.0}],
                'id': '3'}],
 'errors': []}
```

The following lines of code render the JSON data as an HTML table.

```
from IPython.display import HTML
table = []
for document in languages["documents"]:
    text  = next(filter(lambda d: d["id"] == document["id"], documents["documents"]))["text"]
    langs = ", ".join(["{0}({1})".format(lang["name"], lang["score"]) for lang in
document["detectedLanguages"]])
    table.append("<tr><td>{0}</td><td>{1}</td>".format(text, langs))
HTML("<table><tr><th>Text</th><th>Detected languages(scores)</th></tr>{0}</table>".format("\n".join(table)))
```

Analyze sentiment

The Sentiment Analysis API detexts the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

The service endpoint for sentiment analysis is available for your region via the following URL:

```
sentiment_api_url = text_analytics_base_url + "sentiment"
print(sentiment_api_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
```

As with the language detection example, the service is provided with a dictionary with a `documents` key that consists of a list of documents. Each document is a tuple consisting of the `id`, the `text` to be analyzed and the `language` of the text. You can use the language detection API from the previous section to populate this field.

```
documents = {'documents' : [
  {'id': '1', 'language': 'en', 'text': 'I had a wonderful experience! The rooms were wonderful and the staff
was helpful.'},
  {'id': '2', 'language': 'en', 'text': 'I had a terrible time at the hotel. The staff was rude and the food
was awful.'},
  {'id': '3', 'language': 'es', 'text': 'Los caminos que llevan hasta Monte Rainier son espectaculares y
hermosos.'},
  {'id': '4', 'language': 'es', 'text': 'La carretera estaba atascada. Había mucho tráfico el día de ayer.'}
]}
```

The sentiment API can now be used to analyze the documents for their sentiments.

```
headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
response  = requests.post(sentiment_api_url, headers=headers, json=documents)
sentiments = response.json()
pprint(sentiments)
```

```
{'documents': [{'id': '1', 'score': 0.7673527002334595},
               {'id': '2', 'score': 0.18574094772338867},
               {'id': '3', 'score': 0.5}],
 'errors': []}
```

The sentiment score for a document is between 0 and 1, with a higher score indicating a more positive sentiment.

Extract key phrases

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. This section of the walkthrough extracts key phrases for both English and Spanish documents.

The service endpoint for the key-phrase extraction service is accessed via the following URL:

```
key_phrase_api_url = text_analytics_base_url + "keyPhrases"
print(key_phrase_api_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases
```

The collection of documents is the same as what was used for sentiment analysis.

```
documents = {'documents' : [
  {'id': '1', 'language': 'en', 'text': 'I had a wonderful experience! The rooms were wonderful and the staff
was helpful.'},
  {'id': '2', 'language': 'en', 'text': 'I had a terrible time at the hotel. The staff was rude and the food
was awful.'},
  {'id': '3', 'language': 'es', 'text': 'Los caminos que llevan hasta Monte Rainier son espectaculares y
hermosos.'},
  {'id': '4', 'language': 'es', 'text': 'La carretera estaba atascada. Había mucho tráfico el día de ayer.'}
]}
headers   = {'Ocp-Apim-Subscription-Key': subscription_key}
response  = requests.post(key_phrase_api_url, headers=headers, json=documents)
key_phrases = response.json()
pprint(key_phrases)
```

```
{'documents': [
    {'keyPhrases': ['wonderful experience', 'staff', 'rooms'], 'id': '1'},
    {'keyPhrases': ['food', 'terrible time', 'hotel', 'staff'], 'id': '2'},
    {'keyPhrases': ['Monte Rainier', 'caminos'], 'id': '3'},
    {'keyPhrases': ['carretera', 'tráfico', 'día'], 'id': '4'}],
 'errors': []
}
```

The JSON object can once again be rendered as an HTML table using the following lines of code:

```
from IPython.display import HTML
table = []
for document in key_phrases["documents"]:
    text    = next(filter(lambda d: d["id"] == document["id"], documents["documents"]))["text"]
    phrases = ",".join(document["keyPhrases"])
    table.append("<tr><td>{0}</td><td>{1}</td>".format(text, phrases))
HTML("<table><tr><th>Text</th><th>Key phrases</th></tr>{0}</table>".format("\n".join(table)))
```

Identify entities

The Entities API identifies well-known entities in a text document, using the Entities method. The following example identifies entities for English documents.

The service endpoint for the entity linking service is accessed via the following URL:

```
entity_linking_api_url = text_analytics_base_url + "entities"
print(entity_linking_api_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.1-preview/entities
```

The collection of documents is below:

```
documents = {'documents' : [
  {'id': '1', 'text': 'Jeff bought three dozen eggs because there was a 50% discount.'},
  {'id': '2', 'text': 'The Great Depression began in 1929. By 1933, the GDP in America fell by 25%.'}
]}
```

Now, the documents can be sent to the Text Analytics API to receive the response.

```
headers   = {"Ocp-Apim-Subscription-Key": subscription_key}
response  = requests.post(entity_linking_api_url, headers=headers, json=documents)
entities = response.json()
```

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                            "Offset": 0,
                            "Length": 20
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Great Depression",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                    "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
                },
                {
                    "Name": "1929",
                    "Matches": [
                        {
                            "Text": "1929",
                            "Offset": 30,
                            "Length": 4
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                    "SubType": "DateRange"
                },
                {
                    "Name": "By 1933",
                    "Matches": [
                        {
                            "Text": "By 1933",
                            "Offset": 36,
                            "Length": 7
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ]
    ],
    "Errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview
Frequently asked questions (FAQ)

Quickstart: Using Ruby to call the Text Analytics Cognitive Service

10/4/2018 • 8 minutes to read • Edit Online

This article shows you how to detect language, analyze sentiment, extract key phrases, and identify linked entities using the Text Analytics APIs with Ruby.

Refer to the API definitions for technical documentation for the APIs.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**. You can use the **free tier for 5,000 transactions/month** to complete this quickstart.

You must also have the endpoint and access key that was generated for you during sign up.

Detect language

The Language Detection API detects the language of a text document, using the Detect Language method.

1. Create a new Ruby project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
require 'net/https'
require 'uri'
require 'json'

# **********************************************
# *** Update or verify the following values. ***
# **********************************************

# Replace the accessKey string value with your valid access key.
accessKey = 'enter key here'

# Replace or verify the region.
#
# You must use the same region in your REST API call as you used to obtain your access keys.
# For example, if you obtained your access keys from the westus region, replace
# "westcentralus" in the URI below with "westus".
#
# NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
# a free trial access key, you should not need to change this region.
uri = 'https://westus.api.cognitive.microsoft.com'
path = '/text/analytics/v2.0/languages'

uri = URI(uri + path)

documents = { 'documents': [
    { 'id' => '1', 'text' => 'This is a document written in English.' },
    { 'id' => '2', 'text' => 'Este es un document escrito en Español.' },
    { 'id' => '3', 'text' => '这是一个用中文写的文件' }
]}

puts 'Please wait a moment for the results to appear.'

request = Net::HTTP::Post.new(uri)
request['Content-Type'] = "application/json"
request['Ocp-Apim-Subscription-Key'] = accessKey
request.body = documents.to_json

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
    http.request (request)
end

puts JSON::pretty_generate (JSON (response.body))
```

Language detection response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "id": "1",
         "detectedLanguages": [
            {
               "name": "English",
               "iso6391Name": "en",
               "score": 1.0
            }
         ]
      },
      {
         "id": "2",
         "detectedLanguages": [
            {
               "name": "Spanish",
               "iso6391Name": "es",
               "score": 1.0
            }
         ]
      },
      {
         "id": "3",
         "detectedLanguages": [
            {
               "name": "Chinese_Simplified",
               "iso6391Name": "zh_chs",
               "score": 1.0
            }
         ]
      }
   ],
   "errors": [

   ]
}
```

Analyze sentiment

The Sentiment Analysis API detexts the sentiment of a set of text records, using the Sentiment method. The following example scores two documents, one in English and another in Spanish.

1. Create a new Ruby project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
require 'net/https'
require 'uri'
require 'json'

# **********************************************
# *** Update or verify the following values. ***
# **********************************************

# Replace the accessKey string value with your valid access key.
accessKey = 'enter key here'

# Replace or verify the region.
#
# You must use the same region in your REST API call as you used to obtain your access keys.
# For example, if you obtained your access keys from the westus region, replace
# "westcentralus" in the URI below with "westus".
#
# NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
# a free trial access key, you should not need to change this region.
uri = 'https://westus.api.cognitive.microsoft.com'
path = '/text/analytics/v2.0/sentiment'

uri = URI(uri + path)

documents = { 'documents': [
    { 'id' => '1', 'language' => 'en', 'text' => 'I really enjoy the new XBox One S. It has a clean look, it
has 4K/HDR resolution and it is affordable.' },
    { 'id' => '2', 'language' => 'es', 'text' => 'Este ha sido un dia terrible, llegué tarde al trabajo debido
a un accidente automobilistico.' }
]}

puts 'Please wait a moment for the results to appear.'

request = Net::HTTP::Post.new(uri)
request['Content-Type'] = "application/json"
request['Ocp-Apim-Subscription-Key'] = accessKey
request.body = documents.to_json

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
    http.request (request)
end

puts JSON::pretty_generate (JSON (response.body))
```

Sentiment analysis response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "score": 0.99984133243560791,
         "id": "1"
      },
      {
         "score": 0.024017512798309326,
         "id": "2"
      },
   ],
   "errors": [   ]
}
```

Extract key phrases

The Key Phrase Extraction API extracts key-phrases from a text document, using the Key Phrases method. The following example extracts key phrases for both English and Spanish documents.

1. Create a new Ruby project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
require 'net/https'
require 'uri'
require 'json'

# **********************************************
# *** Update or verify the following values. ***
# **********************************************

# Replace the accessKey string value with your valid access key.
accessKey = 'enter key here'

# Replace or verify the region.
#
# You must use the same region in your REST API call as you used to obtain your access keys.
# For example, if you obtained your access keys from the westus region, replace
# "westcentralus" in the URI below with "westus".
#
# NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
# a free trial access key, you should not need to change this region.
uri = 'https://westus.api.cognitive.microsoft.com'
path = '/text/analytics/v2.0/keyPhrases'

uri = URI(uri + path)

documents = { 'documents': [
    { 'id' => '1', 'language' => 'en', 'text' => 'I really enjoy the new XBox One S. It has a clean look, it
has 4K/HDR resolution and it is affordable.' },
    { 'id' => '2', 'language' => 'es', 'text' => 'Si usted quiere comunicarse con Carlos, usted debe de
llamarlo a su telefono movil. Carlos es muy responsable, pero necesita recibir una notificacion si hay algun
problema.' },
    { 'id' => '3', 'language' => 'en', 'text' => 'The Grand Hotel is a new hotel in the center of Seattle. It
earned 5 stars in my review, and has the classiest decor I\'ve ever seen.' },
]}

puts 'Please wait a moment for the results to appear.'

request = Net::HTTP::Post.new(uri)
request['Content-Type'] = "application/json"
request['Ocp-Apim-Subscription-Key'] = accessKey
request.body = documents.to_json

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
    http.request (request)
end

puts JSON::pretty_generate (JSON (response.body))
```

Key phrase extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
   "documents": [
      {
         "keyPhrases": [
            "HDR resolution",
            "new XBox",
            "clean look"
         ],
         "id": "1"
      },
      {
         "keyPhrases": [
            "Carlos",
            "notificacion",
            "algun problema",
            "telefono movil"
         ],
         "id": "2"
      },
      {
         "keyPhrases": [
            "new hotel",
            "Grand Hotel",
            "review",
            "center of Seattle",
            "classiest decor",
            "stars"
         ],
         "id": "3"
      }
   ],
   "errors": [  ]
}
```

Identify entities

The Entities API extracts entities in a text document, using the Entities method. The following example identifies entities for English documents.

1. Create a new Ruby project in your favorite IDE.
2. Add the code provided below.
3. Replace the `accessKey` value with an access key valid for your subscription.
4. Replace the location in `uri` (currently `westus`) to the region you signed up for.
5. Run the program.

```
require 'net/https'
require 'uri'
require 'json'

# **********************************************
# *** Update or verify the following values. ***
# **********************************************

# Replace the accessKey string value with your valid access key.
accessKey = 'enter key here'

# Replace or verify the region.
#
# You must use the same region in your REST API call as you used to obtain your access keys.
# For example, if you obtained your access keys from the westus region, replace
# "westcentralus" in the URI below with "westus".
#
# NOTE: Free trial access keys are generated in the westcentralus region, so if you are using
# a free trial access key, you should not need to change this region.
uri = 'https://westus.api.cognitive.microsoft.com'
path = '/text/analytics/v2.1-preview/entities'

uri = URI(uri + path)

documents = { 'documents': [
    { 'id' => '1', 'language' => 'en', 'text' => 'Jeff bought three dozen eggs because there was a 50%
discount.' },
    { 'id' => '2', 'language' => 'en', 'text' => 'The Great Depression began in 1929. By 1933, the GDP in
America fell by 25%.' },
]}

puts 'Please wait a moment for the results to appear.'

request = Net::HTTP::Post.new(uri)
request['Content-Type'] = "application/json"
request['Ocp-Apim-Subscription-Key'] = accessKey
request.body = documents.to_json

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
    http.request (request)
end

puts JSON::pretty_generate (JSON (response.body))
```

Entity extraction response

A successful response is returned in JSON, as shown in the following example:

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50",
                    "Matches": [
                        {
                            "Text": "50",
                            "Offset": 49,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "50%",
                    "Matches": [
                        {
                            "Text": "50%",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        },
        {
            "Id": "2",
            "Entities": [
                {
                    "Name": "Great Depression",
                    "Matches": [
                        {
                            "Text": "The Great Depression",
                            "Offset": 0,
                            "Length": 20
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Great Depression",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                    "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
                },
                {
                    "Name": "1929",
                    "Matches": [
                        {
                            "Text": "1929",
                            "Offset": 30,
                            "Length": 4
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "By 1933",
                    "Matches": [
                        {
```

```
                        {
                            "Text": "By 1933",
                            "Offset": 36,
                            "Length": 7
                        }
                    ],
                    "Type": "DateTime",
                    "SubType": "DateRange"
                },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
}
```

Next steps

Text Analytics With Power BI

See also

Text Analytics overview

Frequently asked questions (FAQ)

Tutorial: Integrate Power BI with the Text Analytics Cognitive Service

9/17/2018 • 12 minutes to read • Edit Online

Microsoft Power BI Desktop is a free application that lets you connect to, transform, and visualize your data. The Text Analytics service, part of Microsoft Azure Cognitive Services, provides natural language processing. Given raw unstructured text, it can extract the most important phrases, analyze sentiment, and identify well-known entities such as brands. Together, these tools can help you quickly see what your customers are talking about and how they feel about it.

In this tutorial, you'll learn how to:

- Use Power BI Desktop to import and transform data
- Create a custom function in Power BI Desktop
- Integrate Power BI Desktop with the Text Analytics Key Phrases API
- Use the Text Analytics Key Phrases API to extract the most important phrases from customer feedback
- Create a word cloud from customer feedback

Prerequisites

- Microsoft Power BI Desktop. Download at no charge.
- A Microsoft Azure account. Start a free trial or sign in.
- A Cognitive Services API account with the Text Analytics API. If you don't have one, you can sign up and use the free tier for 5,000 transactions/month (see pricing details to complete this tutorial.
- The Text Analytics access key that was generated for you during sign-up.
- Customer comments. You can use our example data or your own data. This tutorial assumes you're using our example data.

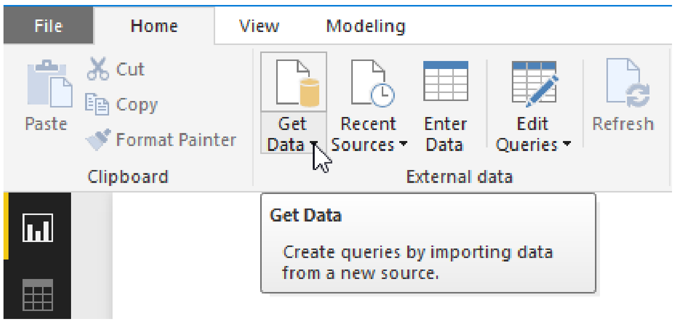
Load customer data

To get started, open Power BI Desktop and load the comma-separated value (CSV) file `FabrikamComments.csv` that you downloaded in Prerequisites. This file represents a day's worth of hypothetical activity in a fictional small company's support forum.

NOTE

Power BI can use data from a wide variety of sources, such as Facebook or a SQL database. Learn more at Facebook integration with Power BI and SQL Server integration with Power BI.

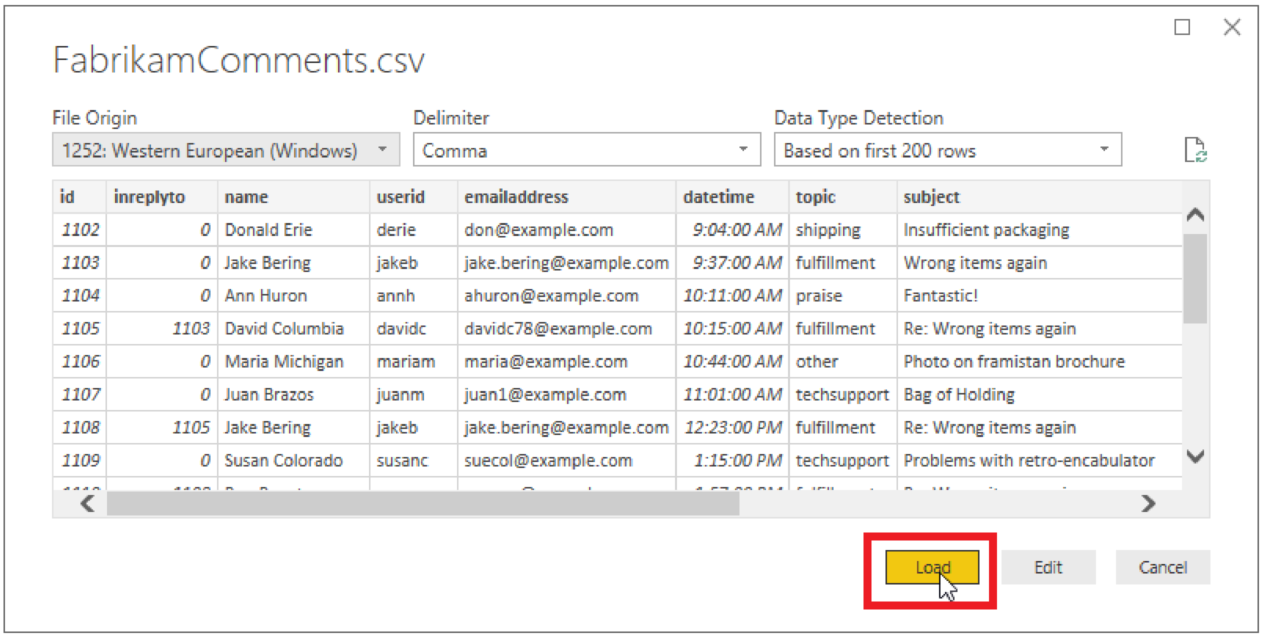
In the main Power BI Desktop window, select the **Home** ribbon. In the **External data** group of the ribbon, open the **Get Data** drop-down menu and select **Text/CSV**.



Get Data

Create queries by importing data from a new source.

The Open dialog appears. Navigate to your Downloads folder, or to the folder where you downloaded the `FabrikamComments.csv` file. Click `FabrikamComments.csv`, then the **Open** button. The CSV import dialog appears.



FabrikamComments.csv

File Origin: 1252: Western European (Windows)

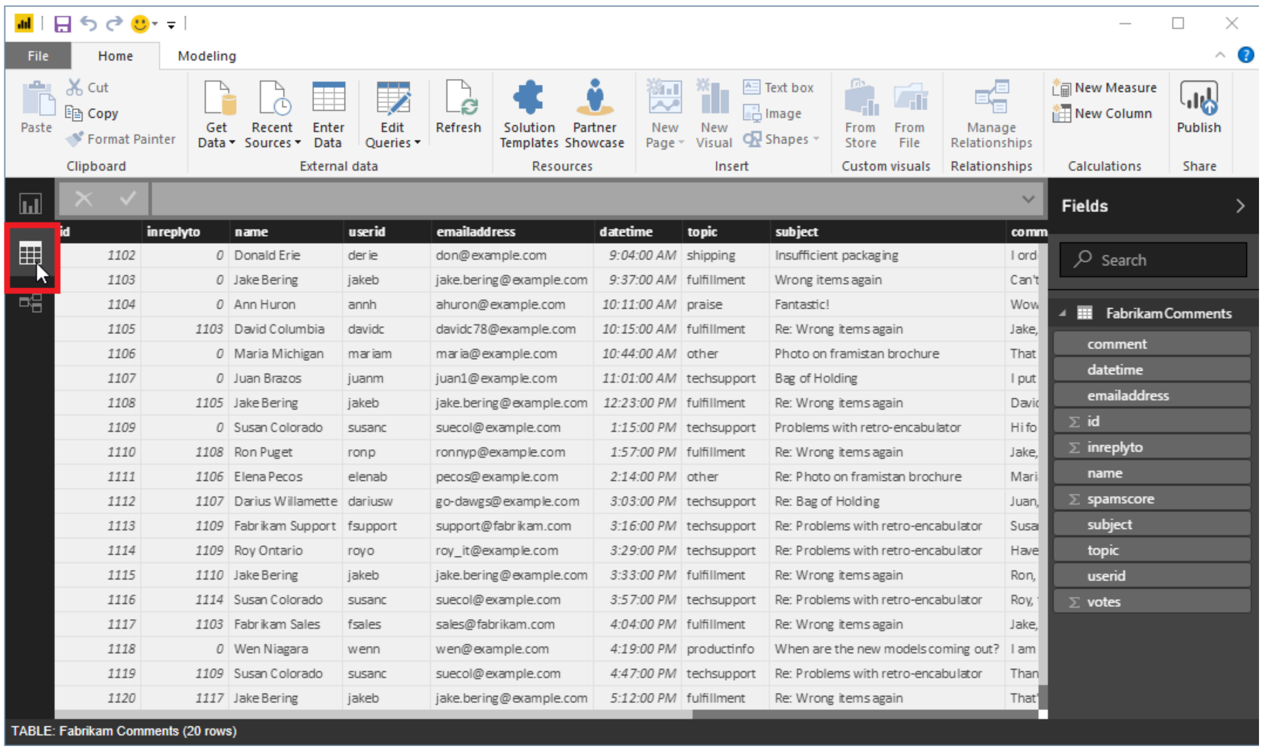
Delimiter: Comma

Data Type Detection: Based on first 200 rows

id	inreplyto	name	userid	emailaddress	datetime	topic	subject
1102	0	Donald Erie	derie	don@example.com	9:04:00 AM	shipping	Insufficient packaging
1103	0	Jake Bering	jakeb	jake.bering@example.com	9:37:00 AM	fulfillment	Wrong items again
1104	0	Ann Huron	annh	ahuron@example.com	10:11:00 AM	praise	Fantastic!
1105	1103	David Columbia	davidc	davidc78@example.com	10:15:00 AM	fulfillment	Re: Wrong items again
1106	0	Maria Michigan	mariam	maria@example.com	10:44:00 AM	other	Photo on framistan brochure
1107	0	Juan Brazos	juanm	juan1@example.com	11:01:00 AM	techsupport	Bag of Holding
1108	1105	Jake Bering	jakeb	jake.bering@example.com	12:23:00 PM	fulfillment	Re: Wrong items again
1109	0	Susan Colorado	susanc	suecol@example.com	1:15:00 PM	techsupport	Problems with retro-encabulator

The CSV import dialog lets you verify that Power BI Desktop has correctly detected the character set, delimiter, header rows, and column types. This information is all correct, so click **Load**.

To see the loaded data, click the **Data View** button on the left edge of the Power BI workspace. A table opens that contains the data, like in Microsoft Excel.



id	inreplyto	name	userid	emailaddress	datetime	topic	subject	comm
1102	0	Donald Erie	derie	don@example.com	9:04:00 AM	shipping	Insufficient packaging	I ord
1103	0	Jake Bering	jakeb	jake.bering@example.com	9:37:00 AM	fulfillment	Wrong items again	Can't
1104	0	Ann Huron	annh	ahuron@example.com	10:11:00 AM	praise	Fantastic!	Wow
1105	1103	David Columbia	davidc	davidc78@example.com	10:15:00 AM	fulfillment	Re: Wrong items again	Jake,
1106	0	Maria Michigan	mariam	maria@example.com	10:44:00 AM	other	Photo on framistan brochure	That
1107	0	Juan Brazos	juanm	juan1@example.com	11:01:00 AM	techsupport	Bag of Holding	I put
1108	1105	Jake Bering	jakeb	jake.bering@example.com	12:23:00 PM	fulfillment	Re: Wrong items again	Davic
1109	0	Susan Colorado	susanc	suecol@example.com	1:15:00 PM	techsupport	Problems with retro-encabulator	Hi fo
1110	1108	Ron Puget	ronp	ronnyp@example.com	1:57:00 PM	fulfillment	Re: Wrong items again	Jake,
1111	1106	Elena Pecos	elenab	pecos@example.com	2:14:00 PM	other	Re: Photo on framistan brochure	Mari
1112	1107	Darius Willamette	dariusw	go-dawgs@example.com	3:03:00 PM	techsupport	Re: Bag of Holding	Juan,
1113	1109	Fabrikam Support	fsupport	support@fabrikam.com	3:16:00 PM	techsupport	Re: Problems with retro-encabulator	Susa
1114	1109	Roy Ontario	royo	roy_it@example.com	3:29:00 PM	techsupport	Re: Problems with retro-encabulator	Have
1115	1110	Jake Bering	jakeb	jake.bering@example.com	3:33:00 PM	fulfillment	Re: Wrong items again	Ron,
1116	1114	Susan Colorado	susanc	suecol@example.com	3:57:00 PM	techsupport	Re: Problems with retro-encabulator	Roy,
1117	1103	Fabrikam Sales	fsales	sales@fabrikam.com	4:04:00 PM	fulfillment	Re: Wrong items again	Jake,
1118	0	Wen Niagara	wenn	wen@example.com	4:19:00 PM	productinfo	When are the new models coming out?	I am
1119	1109	Susan Colorado	susanc	suecol@example.com	4:47:00 PM	techsupport	Re: Problems with retro-encabulator	Than
1120	1117	Jake Bering	jakeb	jake.bering@example.com	5:12:00 PM	fulfillment	Re: Wrong items again	That'

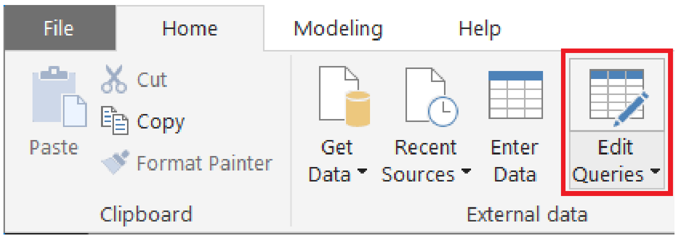
TABLE: Fabrikam Comments (20 rows)

Prepare the data

You may need to transform your data in Power BI Desktop before it's ready to be processed by the Key Phrases API of the Text Analytics service.

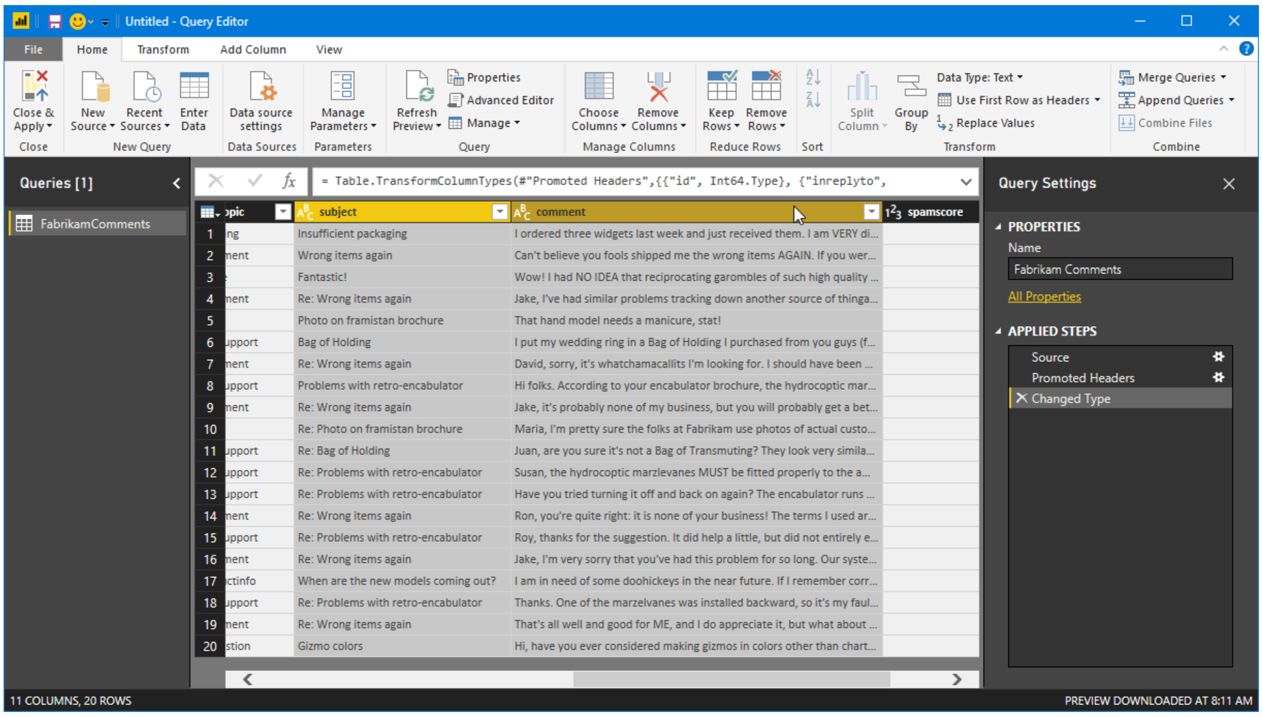
The sample data contains a `subject` column and a `comment` column. With the Merge Columns function in Power BI Desktop, you can extract key phrases from the data in both these columns, rather than just the `comment` column.

In Power BI Desktop, select the **Home** ribbon. In the **External data** group, click **Edit Queries**.



Select `FabrikamComments` in the **Queries** list at the left side of the window if it isn't already selected.

Now select both the `subject` and `comment` columns in the table. You may need to scroll horizontally to see these columns. First click the `subject` column header, then hold down the Control key and click the `comment` column header.

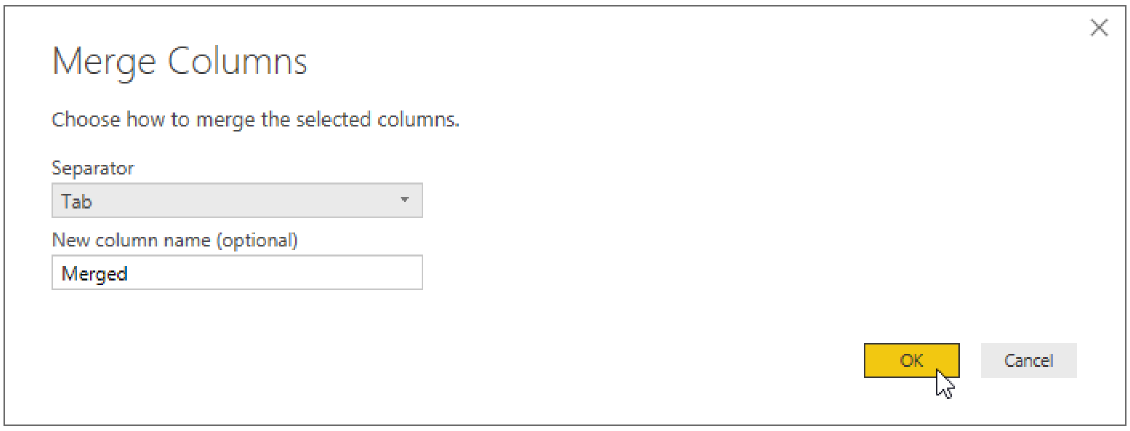


```
= Table.TransformColumnTypes(#"Promoted Headers",{{"id", Int64.Type}, {"inreplyto",
```

	opic	subject	comment	spamscore
1	ing	Insufficient packaging	I ordered three widgets last week and just received them. I am VERY di...	
2	ment	Wrong items again	Can't believe you fools shipped me the wrong items AGAIN. If you wer...	
3	e	Fantastic!	Wow! I had NO IDEA that reciprocating garombles of such high quality ...	
4	ment	Re: Wrong items again	Jake, I've had similar problems tracking down another source of thinga...	
5		Photo on framistan brochure	That hand model needs a manicure, stat!	
6	upport	Bag of Holding	I put my wedding ring in a Bag of Holding I purchased from you guys (f...	
7	ment	Re: Wrong items again	David, sorry, it's whatchamacallits I'm looking for. I should have been ...	
8	upport	Problems with retro-encabulator	Hi folks. According to your encabulator brochure, the hydrocoptic mar...	
9	ment	Re: Wrong items again	Jake, it's probably none of my business, but you will probably get a bet...	
10		Re: Photo on framistan brochure	Maria, I'm pretty sure the folks at Fabrikam use photos of actual custo...	
11	upport	Re: Bag of Holding	Juan, are you sure it's not a Bag of Transmuting? They look very simila...	
12	upport	Re: Problems with retro-encabulator	Susan, the hydrocoptic marzlevanes MUST be fitted properly to the a...	
13	upport	Re: Problems with retro-encabulator	Have you tried turning it off and back on again? The encabulator runs ...	
14	ment	Re: Wrong items again	Ron, you're quite right: it is none of your business! The terms I used ar...	
15	upport	Re: Problems with retro-encabulator	Roy, thanks for the suggestion. It did help a little, but did not entirely e...	
16	ment	Re: Wrong items again	Jake, I'm very sorry that you've had this problem for so long. Our syste...	
17	ctinfo	When are the new models coming out?	I am in need of some doohickeys in the near future. If I remember corr...	
18	upport	Re: Problems with retro-encabulator	Thanks. One of the marzelvanes was installed backward, so it's my faul...	
19	ment	Re: Wrong items again	That's all well and good for ME, and I do appreciate it, but what about ...	
20	stion	Gizmo colors	Hi, have you ever considered making gizmos in colors other than chart...	

11 COLUMNS, 20 ROWS
PREVIEW DOWNLOADED AT 8:11 AM

Select the **Transform** ribbon. In the **Text Columns** group of the ribbon, click **Merge Columns**. The Merge Columns dialog appears.


Merge Columns

Choose how to merge the selected columns.

Separator: Tab

New column name (optional): Merged

In the Merge Columns dialog, choose `Tab` as the separator, then click **OK.**

You might also consider filtering out blank messages using the Remove Empty filter, or removing unprintable characters using the Clean transformation. If your data contains a column like the `spamscore` column in the sample file, you can skip "spam" comments using a Number Filter.

Understand the API

The Key Phrases API of the Text Analytics service can process up to a thousand text documents per HTTP request. Power BI prefers to deal with records one at a time, so in this tutorial your calls to the API will include only a single document each. The Key Phrases API requires the following fields for each document being processed.

`id`	A unique identifier for this document within the request. The response also contains this field. That way, if you process more than one document, you can easily associate the extracted key phrases with the document they came from. In this tutorial, because you're processing only one document per request, you can hard-code the value of `id` to be the same for each request.
`text`	The text to be processed. The value of this field comes from the `Merged` column you created in the previous section, which contains the combined subject line and comment text. The Key Phrases API requires this data be no longer than about 5,000 characters.
`language`	The code for the natural language the document is written in. All the messages in the sample data are in English, so you can hard-code the value `en` for this field.

Create a custom function

Now you're ready to create the custom function that will integrate Power BI and Text Analytics. The function receives the text to be processed as a parameter. It converts data to and from the required JSON format and makes the HTTP request to the Key Phrases API. The function then parses the response from the API and returns a string that contains a comma-separated list of the extracted key phrases.

NOTE

Power BI Desktop custom functions are written in the Power Query M formula language, or just "M" for short. M is a functional programming language based on F#. You don't need to be a programmer to finish this tutorial, though; the required code is included below.

In Power BI Desktop, make sure you're still in the Query Editor window. If you aren't, select the**Home** ribbon, and in the **External data** group, click **Edit Queries**.

Now, in the **Home** ribbon, in the **New Query** group, open the **New Source** drop-down menu and select **Blank Query**.

A new query, initially named `Query1`, appears in the Queries list. Double-click this entry and name it `KeyPhrases`.

Now, in the **Home** ribbon, in the **Query** group, click **Advanced Editor** to open the Advanced Editor window. Delete the code that's already in that window and paste in the following code.

NOTE

The examples below assume the Text Analytics API endpoint begins with `https://westus.api.cognitive.microsoft.com`. Text Analytics allows you to create a subscription in 13 different regions. If you signed up for the service in a different region, please make sure to use the endpoint for the region you selected. You can find this endpoint by signing in to the Azure portal, selecting your Text Analytics subscription, and selecting the Overview page.

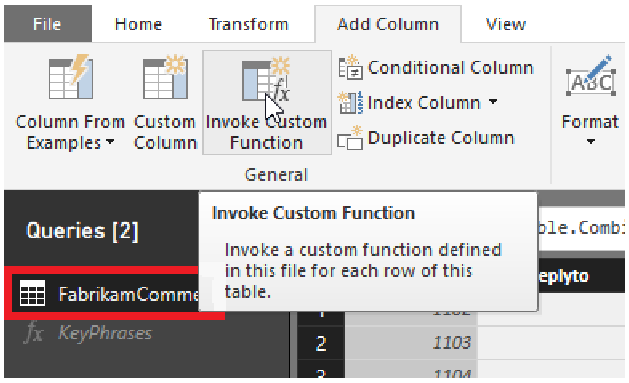
```
// Returns key phrases from the text in a comma-separated list
(text) => let
    apikey      = "YOUR_API_KEY_HERE",
    endpoint    = "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases",
    jsontext    = Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { language: ""en"", id: ""0"", text: " & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [#"Ocp-Apim-Subscription-Key" = apikey],
    bytesresp   = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    keyphrases  = Text.Lower(Text.Combine(jsonresp[documents]{0}[keyPhrases], ", "))
in  keyphrases
```

Replace `YOUR_API_KEY_HERE` with your Text Analytics access key. You can also find this key by signing in to the Azure portal, selecting your Text Analytics subscription, and selecting the Overview page. Be sure to leave the quotation marks before and after the key. Then click **Done.**

Use the custom function

Now you can use the custom function to extract the key phrases from each of the customer comments and store them in a new column in the table.

In Power BI Desktop, in the Query Editor window, switch back to the `FabrikamComments` query. Select the **Add Column** ribbon. In the **General** group, click **Invoke Custom Function**.

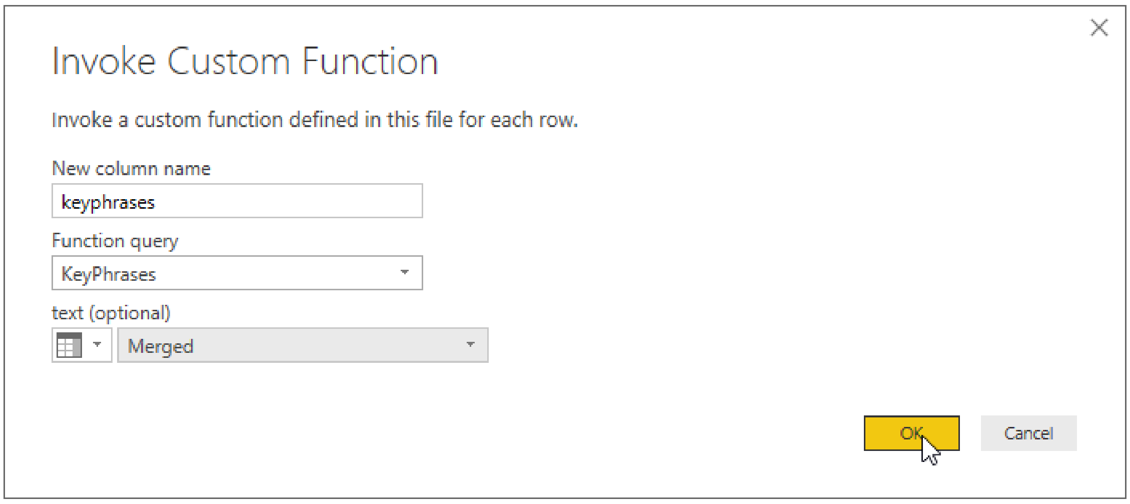


Invoke Custom Function

Invoke a custom function defined in this file for each row of this table.

The Invoke Custom Function dialog appears. In **New column name**, enter `keyphrases`. In **Function query**, select the custom function you created, `KeyPhrases`.

A new field appears in the dialog, **text (optional)**. This field is asking which column we want to use to provide values for the `text` parameter of the Key Phrases API. (Remember that you already hard-coded the values for the `language` and `id` parameters.) Select `Merged` (the column you created previously by merging the subject and message fields) from the drop-down menu.



Invoke Custom Function

Invoke a custom function defined in this file for each row.

New column name: keyphrases

Function query: KeyPhrases

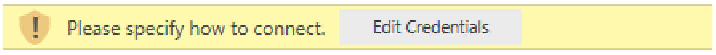
text (optional): Merged

Finally, click **OK.**

If everything is ready, Power BI calls your custom function once for each row in the table. It sends the queries to the Key Phrases API and adds a new column to the table to store the results. But before that happens, you may need to specify authentication and privacy settings.

Authentication and privacy

After you close the Invoke Custom Function dialog, a banner may appear asking you to specify how to connect to the Key Phrases API.

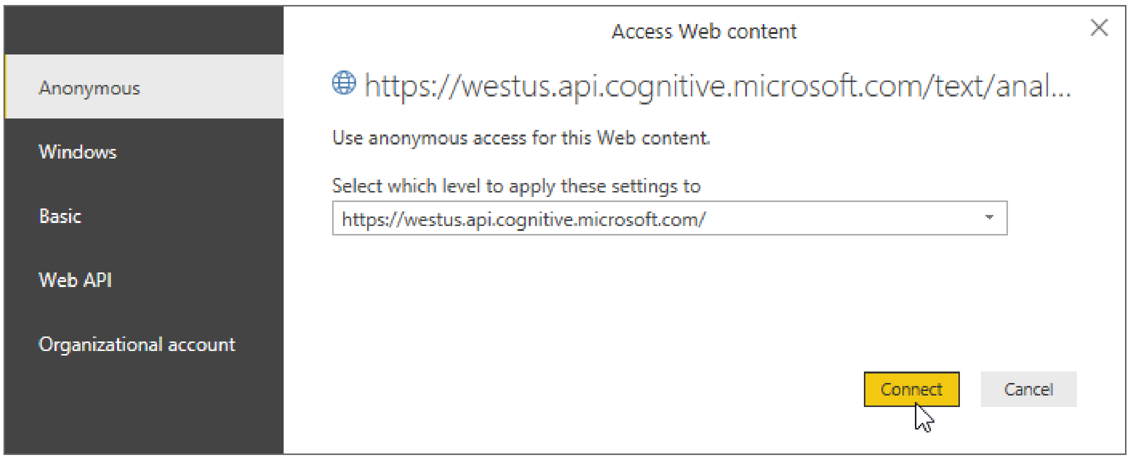


Please specify how to connect.

Click **Edit Credentials,** make sure `Anonymous` is selected in the dialog, then click **Connect.**

NOTE

You select `Anonymous` because the Text Analytics service authenticates you using your access key, so Power BI does not need to provide credentials for the HTTP request itself.



Access Web content

https://westus.api.cognitive.microsoft.com/text/anal...

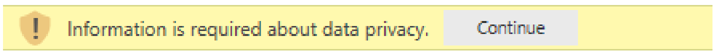
Use anonymous access for this Web content.

Select which level to apply these settings to

https://westus.api.cognitive.microsoft.com/

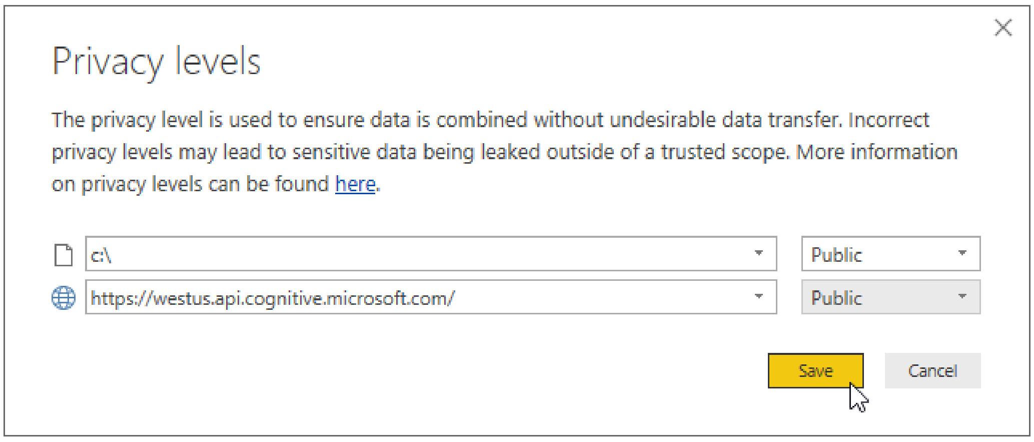
If you see the Edit Credentials banner even after choosing anonymous access, you may have forgotten to paste your Text Analytics access key into the code in the `KeyPhrases` custom function.

Next, a banner may appear asking you to provide information about your data sources' privacy.



Information is required about data privacy.

Click **Continue** and choose `Public` for each of the data sources in the dialog. Then click **Save.**



Privacy levels

The privacy level is used to ensure data is combined without undesirable data transfer. Incorrect privacy levels may lead to sensitive data being leaked outside of a trusted scope. More information on privacy levels can be found here.

c:\	Public
https://westus.api.cognitive.microsoft.com/	Public

Create the word cloud

Once you have dealt with any banners that appear, click **Close & Apply** in the Home ribbon to close the Query Editor.

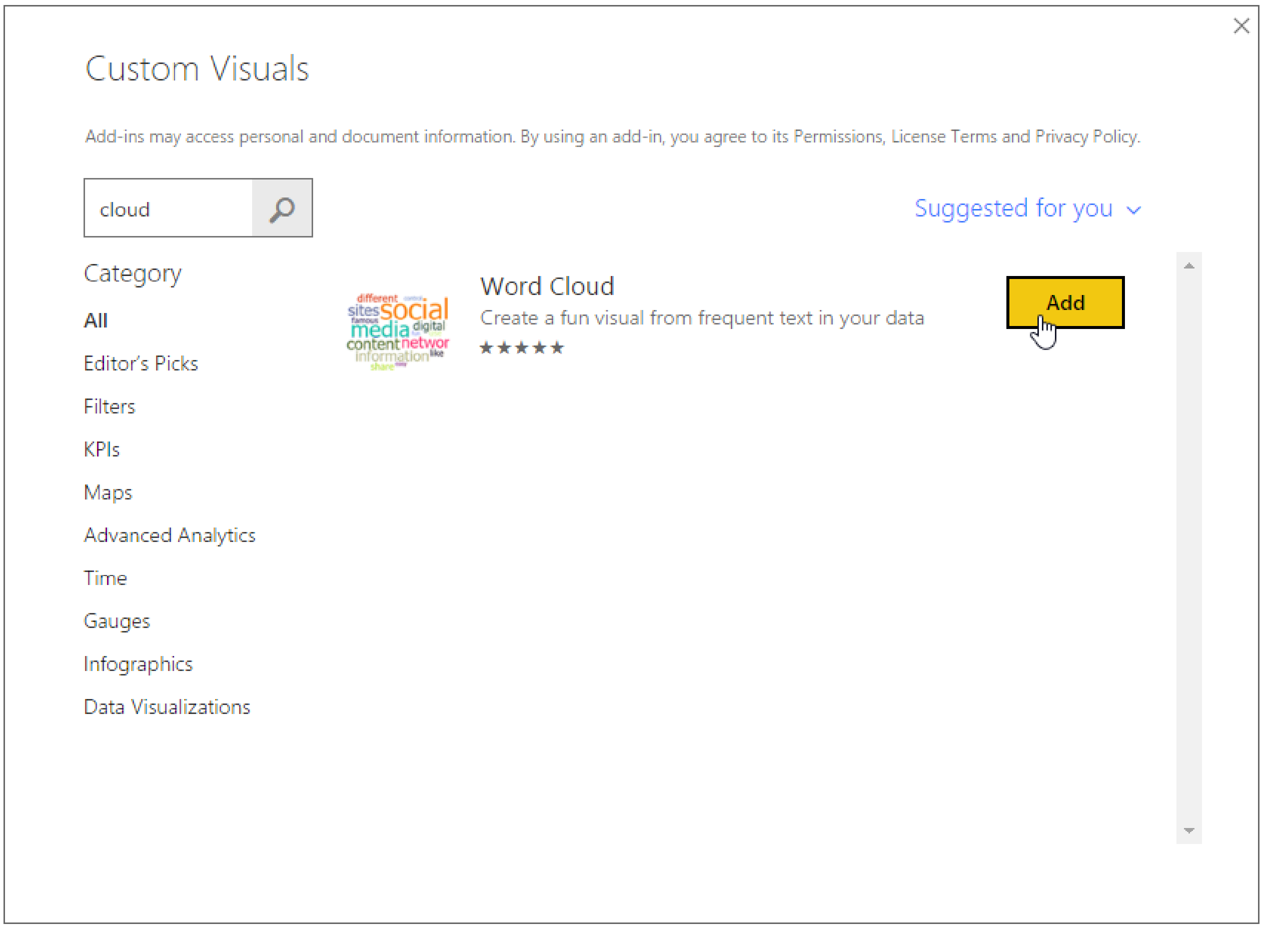
Power BI Desktop takes a moment to make the necessary HTTP requests. For each row in the table, the new `keyphrases` column contains the key phrases detected in the text by the Key Phrases API.

Now you'll use this column to generate a word cloud. To get started, click the **Report** button in the main Power BI Desktop window, to the left of the workspace.

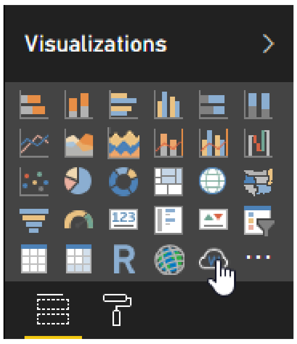
NOTE

Why use extracted key phrases to generate a word cloud, rather than the full text of every comment? The key phrases provide us with the *important* words from our customer comments, not just the *most common* words. Also, word sizing in the resulting cloud isn't skewed by the frequent use of a word in a relatively small number of comments.

If you don't already have the Word Cloud custom visual installed, install it. In the Visualizations panel to the right of the workspace, click the three dots (**...**) and choose **Import From Store**. Then search for "cloud" and click the **Add** button next the Word Cloud visual. Power BI installs the Word Cloud visual and lets you know that it installed successfully.

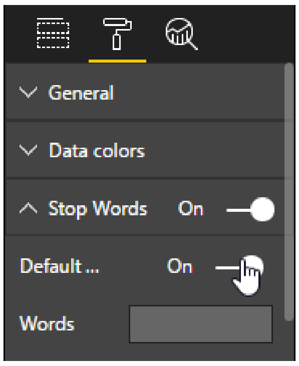

Custom Visuals
Add-ins may access personal and document information. By using an add-in, you agree to its Permissions, License Terms and Privacy Policy.
cloud
Suggested for you
Category
All
Editor's Picks
Filters
KPIs
Maps
Advanced Analytics
Time
Gauges
Infographics
Data Visualizations
Word Cloud
Create a fun visual from frequent text in your data
★★★★★
Add

First, click the Word Cloud icon in the Visualizations panel.

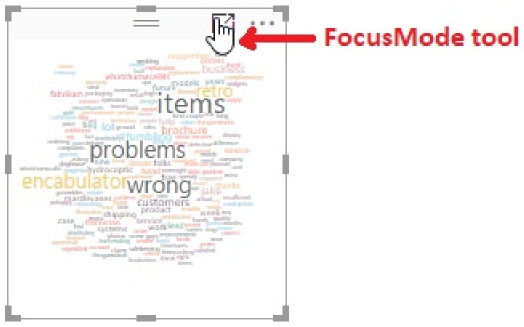


A new report appears in the workspace. Drag the `keyphrases` field from the Fields panel to the Category field in the Visualizations panel. The word cloud appears inside the report.

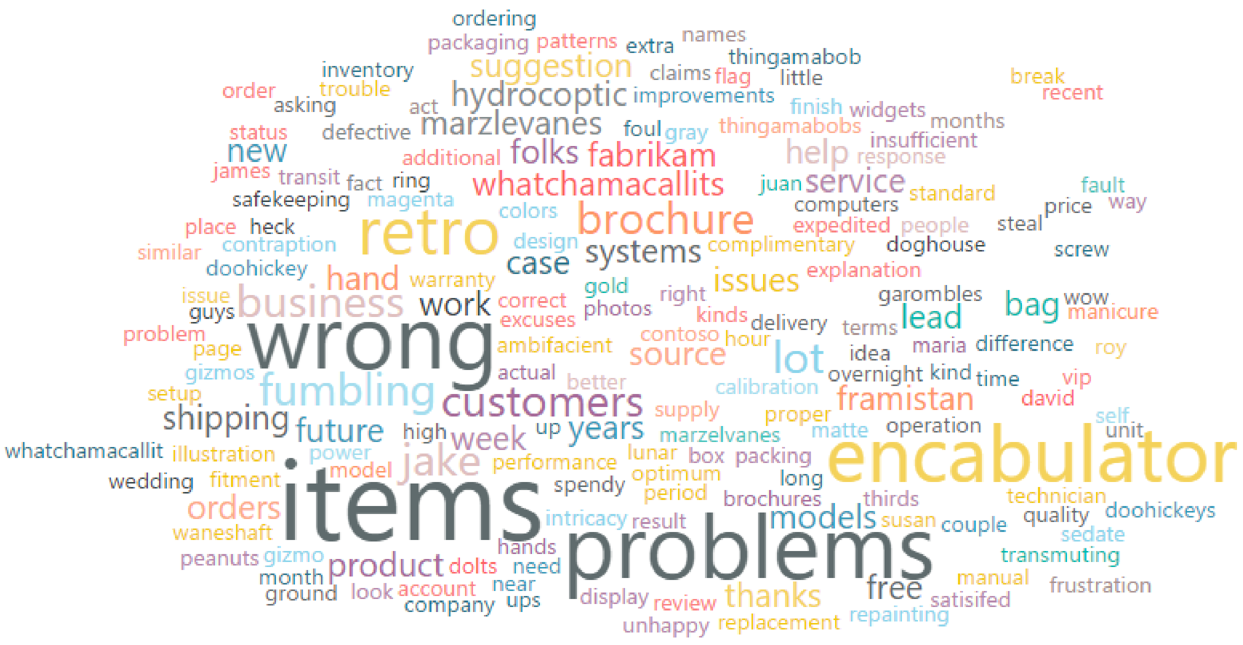
Now switch to the Format page of the Visualizations panel. In the Stop Words category, turn on **Default Stop Words** to eliminate short, common words like "of" from the cloud.



Down a little further in this panel, turn off **Rotate Text** and **Title**.



Click the Focus Mode tool in the report to get a better look at our word cloud. The tool expands the word cloud to fill the entire workspace, as shown below.



More Text Analytics services

The Text Analytics service, one of the Cognitive Services offered by Microsoft Azure, also provides sentiment analysis and language detection. The language detection in particular is useful if your customer feedback isn't all in English.

Both of these other APIs are similar to the Key Phrases API. That means you can integrate them with Power BI Desktop using custom functions that are nearly identical to the one you created in this tutorial. Just create a blank query and paste the appropriate code below into the Advanced Editor, as you did earlier. (Don't forget your access key!) Then, as before, use the function to add a new column to the table.

The Sentiment Analysis function below returns a score indicating how positive the sentiment expressed in the text is.

```
// Returns the sentiment score of the text, from 0.0 (least favorable) to 1.0 (most favorable)
(text) => let
    apikey      = "YOUR_API_KEY_HERE",
    endpoint    = "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment",
    jsontext    = Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { language: ""en"", id: ""0"", text: " & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [#"Ocp-Apim-Subscription-Key" = apikey],
    bytesresp   = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    sentiment   = jsonresp[documents]{0}[score]
in  sentiment
```

Here are two versions of a Language Detection function. The first returns the ISO language code (for example, `en` for English), while the second returns the "friendly" name (for example, `English`). You may notice that only the last line of the body differs between the two versions.

```
// Returns the two-letter language code (for example, 'en' for English) of the text
(text) => let
    apikey      = "YOUR_API_KEY_HERE",
    endpoint    = "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages",
    jsontext    = Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { id: ""0"", text: " & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [#"Ocp-Apim-Subscription-Key" = apikey],
    bytesresp   = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    language    = jsonresp[documents]{0}[detectedLanguages]{0}[iso6391Name]
in  language
```

```
// Returns the name (for example, 'English') of the language in which the text is written
(text) => let
    apikey      = "YOUR_API_KEY_HERE",
    endpoint    = "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages",
    jsontext    = Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { id: ""0"", text: " & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [#"Ocp-Apim-Subscription-Key" = apikey],
    bytesresp   = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    language    = jsonresp[documents]{0}[detectedLanguages]{0}[name]
in  language
```

Finally, here's a variant of the Key Phrases function already presented that returns the phrases as a list object, rather than as a single string of comma-separated phrases.

NOTE

Returning a single string simplified our word cloud example. A list, on the other hand, is a more flexible format for working with the returned phrases in Power BI. You can manipulate list objects in Power BI Desktop using the Structured Column group in the Query Editor's Transform ribbon.

```
// Returns key phrases from the text as a list object
(text) => let
    apikey      = "YOUR_API_KEY_HERE",
    endpoint    = "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases",
    jsontext    = Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { language: ""en"", id: ""0"", text: " & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [#"Ocp-Apim-Subscription-Key" = apikey],
    bytesresp   = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    keyphrases  = jsonresp[documents]{0}[keyPhrases]
in  keyphrases
```

Next steps

Learn more about the Text Analytics service, the Power Query M formula language, or Power BI.

Text Analytics API reference

Power Query M reference

Power BI documentation

How to sign up for the Text Analytics API

9/14/2018 • 2 minutes to read • Edit Online

Text Analytics resources are available 24-7 in the cloud. Before you can upload your content for analysis, you must sign up to get an access key. Each call to the API requires an access key on the request.

- Start with an Azure Subscription. You can create a free account to experiment at no charge.
- Create a Cognitive Services API account, choosing the **Text Analytics API**. Your key is generated when you sign up.

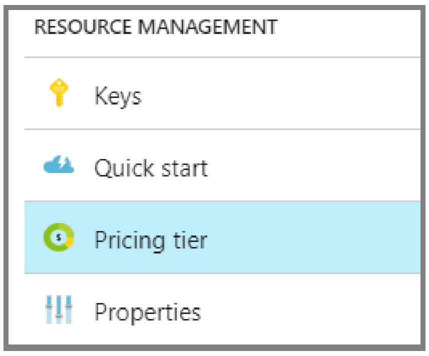
For Text Analytics, there is a Free tier for exploration and evaluation, and billable tiers for production workloads. You can have multiple sign-ups in each subscrption: one free, one paid, and so forth. You can switch to a tier offering more transactions if your request volume increases.

There is no service level agreement for services in Preview or the free tier. For more information, see SLA for Cognitive Services

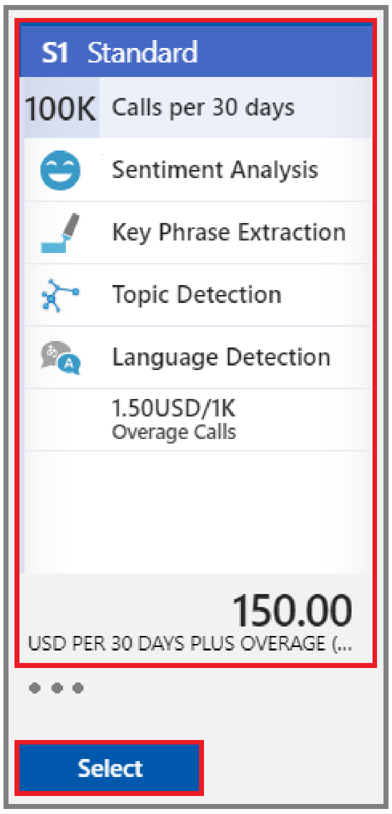
How to change tiers

Start with a Free tier and then transition to a billable tier for production workloads. Standard billing is offered at graduated levels. You can switch tiers and still keep the same endpoint and access keys.

1. Sign in to Azure portal and find your service.
2. Click **Price tier**.



3. Choose a tier and click **Select**. The new limits take effect as soon as the selection is processed.



How billing works

Billing is based on the number of transactions. You can purchase a block of transactions at a specific tier in a monthly billing cycle, and then if you go over, a small overage charge is applied per transaction. If you routinely go over the maximum limit, consider switching to a higher tier.

Please see the pricing page for more information.

What constitutes a transaction in the Text Analytics API?

Any annotation to a document counts as a transaction. Batch scoring calls will also take into consideration the number of documents that need to be scored in that transaction. So for instance, if 1,000 documents are sent for sentiment analysis in a single API call, that will count for 1,000 transactions.

See also

Text Analytics Overview
Frequently asked questions (FAQ)

Next steps

Get an access key

How to find endpoints and access keys for the Text Analytics Cognitive Service

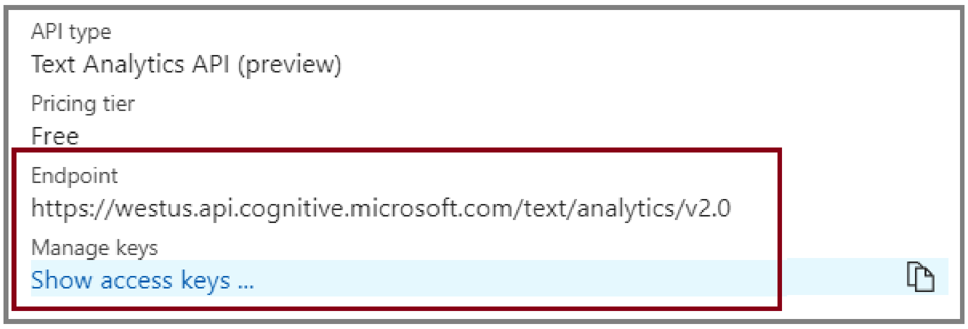
9/14/2018 • 2 minutes to read • Edit Online

When you sign up for Text Analytics, you get a personalized access key unique to your subscription. This key is required on each call to the Text Analytics API. If you haven't signed up, do so now to get your key.

If you need help finding the key, or to determine whether your subscription already has Text Analytics, use the following instructions to get the necessary information.

Find your service endpoint and access key

1. Sign in to Azure portal.
2. In the left navigation pane, select **All services**.
3. In Filter, type *Cognitive Services*. If your subscription has Text Analytics, it appears in the list, with an API Type of **Text Analytics API**.
4. Click the link to open the service blade. You can now get a key from **Resources > Keys** or by clicking **Show access keys** in the Essentials pane. The endpoint is also visible in the Essentials pane.



API type
Text Analytics API (preview)
Pricing tier
Free
Endpoint
https://westus.api.cognitive.microsoft.com/text/analytics/v2.0
Manage keys
Show access keys ...

See also

Text Analytics Overview
Frequently asked questions (FAQ)

Next steps

Call the Text Analytics API

How to call the Text Analytics REST API

10/9/2018 • 3 minutes to read • Edit Online

Calls to the **Text Analytics API** are HTTP POST/GET calls, which you can formulate in any language. In this article, we use REST and Postman to demonstrate key concepts.

Each request must include your access key and an HTTP endpoint. The endpoint specifies the region you chose during sign up, the service URL, and a resource used on the request: `sentiment`, `keyphrases`, `languages`, and `entities`.

Recall that Text Analytics is stateless so there are no data assets to manage. Your text is uploaded, analyzed upon receipt, and results are returned immediately to the calling application.

TIP

For one-off calls to see how the API works, you can send POST requests from the built-in **API testing console**, available on any API doc page. There is no setup, and the only requirements are to paste an access key and the JSON documents into the request.

Prerequisites

You must have a Cognitive Services API account with **Text Analytics API**.

You must have the endpoint and access key that is generated for you when you sign up for Cognitive Services.

JSON schema definition

Input must be JSON in raw unstructured text. XML is not supported. The schema is simple, consisting of the elements described in the following list.

You can currently submit the same documents for all Text Analytics operations: sentiment, key phrase, language detection, and entity identification. (The schema is likely to vary for each analysis in the future.)

ELEMENT	VALID VALUES	REQUIRED?	USAGE
`id`	The data type is string, but in practice document IDs tend to be integers.	Required	The system uses the IDs you provide to structure the output. Language codes, key phrases, and sentiment scores are generated for each ID in the request.
`text`	Unstructured raw text, up to 5,000 characters.	Required	For language detection, text can be expressed in any language. For sentiment analysis, key phrase extraction and entity identification, the text must be in a supported language.

ELEMENT	VALID VALUES	REQUIRED?	USAGE
`language`	2-character ISO 639-1 code for a supported language	Varies	Required for sentiment analysis, key phrase extraction, and entity linking; optional for language detection. There is no error if you exclude it, but the analysis is weakened without it. The language code should correspond to the `text` you provide.

For more information about limits, see Text Analytics Overview > Data limits.

Set up a request in Postman

The service accepts request up to 1 MB in size. If you are using Postman (or another Web API test tool), set up the endpoint to include the resource you want to use, and provide the access key in a request header. Each operation requires that you append the appropriate resource to the endpoint.

1. In Postman:

   - Choose **Post** as the request type.
   - Paste in the endpoint you copied from the portal page.
   - Append a resource.

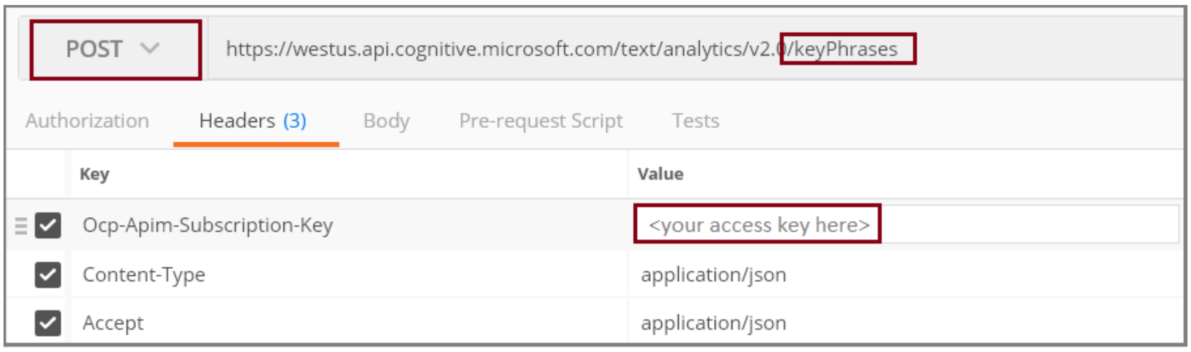
   Resource endpoints are as follows (your region may vary):

   - `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment`
   - `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases`
   - `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages`
   - `https://westus.api.cognitive.microsoft.com/text/analytics/v2.1-preview/entities`

2. Set the three request headers:

   - `Ocp-Apim-Subscription-Key`: your access key, obtained from Azure portal.
   - `Content-Type`: application/json.
   - `Accept`: application/json.

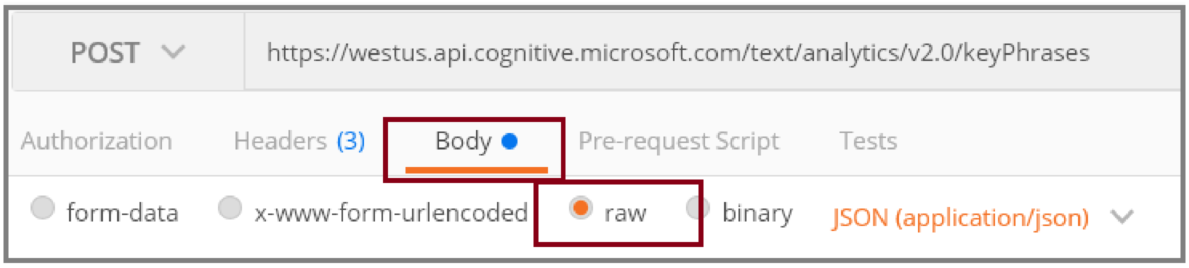
   Your request should look similar to the following screenshot, assuming a **/keyPhrases** resource.



https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases

Key	Value
Ocp-Apim-Subscription-Key	<your access key here>
Content-Type	application/json
Accept	application/json

3. Click **Body** and choose **raw** for the format.


POST https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases

Authorization | Headers (3) | Body ● | Pre-request Script | Tests

form-data | x-www-form-urlencoded | [✓ raw] | binary | JSON (application/json)

4. Paste in some JSON documents in a format that is valid for the intended analysis. For more information about a particular analysis, see the topics below:

   - Language detection
   - Key phrase extraction
   - Sentiment analysis
   - Entity recognition (Preview)

5. Click **Send** to submit the request. You can submit up to 100 requests per minute.

   In Postman, the response is displayed in the next window down, as a single JSON document, with an item for each document ID provided in the request.

See also

Text Analytics Overview
Frequently asked questions (FAQ)

Next steps

Detect language

Example: How to detect language in Text Analytics

9/14/2018 • 4 minutes to read • Edit Online

The Language Detection API evaluates text input and for each document and returns language identifiers with a score indicating the strength of the analysis. Text Analytics recognizes up to 120 languages.

This capability is useful for content stores that collect arbitrary text, where language is unknown. You can parse the results of this analysis to determine which language is used in the input document. The response also returns a score which reflects the confidence of the model (a value between 0 and 1).

Preparation

You must have JSON documents in this format: id, text

Document size must be under 5,000 characters per document, and you can have up to 1,000 items (IDs) per collection. The collection is submitted in the body of the request. The following is an example of content you might submit for language detection.

```
{
    "documents": [
        {
            "id": "1",
            "text": "This document is in English."
        },
        {
            "id": "2",
            "text": "Este documento está en inglés."
        },
        {
            "id": "3",
            "text": "Ce document est en anglais."
        },
        {
            "id": "4",
            "text": "本文件为英文"
        },
        {
            "id": "5",
            "text": "Этот документ находится на английском языке."
        }
    ]
}
```

Step 1: Structure the request

Details on request definition can be found in How to call the Text Analytics API. The following points are restated for convenience:

- Create a **POST** request. Review the API documentation for this request: Language Detection API
- Set the HTTP endpoint for language detection. It must include the `/languages` resource: `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages`
- Set a request header to include the access key for Text Analytics operations. For more information, see How to find endpoints and access keys.

- In the request body, provide the JSON documents collection you prepared for this analysis

> **TIP**
>
> Use Postman or open the **API testing console** in the documentation to structure a request and POST it to the service.

Step 2: Post the request

Analysis is performed upon receipt of the request. The service accepts up to 100 requests per minute. Each request can be a maximum of 1 MB.

Recall that the service is stateless. No data is stored in your account. Results are returned immediately in the response.

Step 3: View results

All POST requests return a JSON formatted response with the IDs and detected properties.

Output is returned immediately. You can stream the results to an application that accepts JSON or save the output to a file on the local system, and then import it into an application that allows you to sort, search, and manipulate the data.

Results for the example request should look like the following JSON. Notice that it is one document with multiple items. Output is in English. Language identifiers include a friendly name and a language code in ISO 639-1 format.

A positive score of 1.0 expresses the highest possible confidence level of the analysis.

```
{
    "documents": [
        {
            "id": "1",
            "detectedLanguages": [
                {
                    "name": "English",
                    "iso6391Name": "en",
                    "score": 1
                }
            ]
        },
        {
            "id": "2",
            "detectedLanguages": [
                {
                    "name": "Spanish",
                    "iso6391Name": "es",
                    "score": 1
                }
            ]
        },
        {
            "id": "3",
            "detectedLanguages": [
                {
                    "name": "French",
                    "iso6391Name": "fr",
                    "score": 1
                }
            ]
        },
        {
            "id": "4",
            "detectedLanguages": [
                {
                    "name": "Chinese_Simplified",
                    "iso6391Name": "zh_chs",
                    "score": 1
                }
            ]
        },
        {
            "id": "5",
            "detectedLanguages": [
                {
                    "name": "Russian",
                    "iso6391Name": "ru",
                    "score": 1
                }
            ]
        }
    ],
```

Ambiguous content

If the analyzer cannot parse the input (for example, assume you submitted a text block consisting solely of Arabic numerals), it returns `(Unknown)`.

```
    {
      "id": "5",
      "detectedLanguages": [
        {
          "name": "(Unknown)",
          "iso6391Name": "(Unknown)",
          "score": "NaN"
        }
      ]
```

Mixed language content

Mixed language content within the same document returns the language with the largest representation in the content, but with a lower positive rating, reflecting the marginal strength of that assessment. In the following example, input is a blend of English, Spanish, and French. The analyzer counts characters in each segment to determine the predominant language.

Input

```
{
  "documents": [
    {
      "id": "1",
      "text": "Hello, I would like to take a class at your University. ¿Se ofrecen clases en español? Es mi primera lengua y más fácil para escribir. Que diriez-vous des cours en français?"
    }
  ]
}
```

Output

Resulting output consists of the predominant language, with a score of less than 1.0, indicating a weaker level of confidence.

```
{
  "documents": [
    {
      "id": "1",
      "detectedLanguages": [
        {
          "name": "Spanish",
          "iso6391Name": "es",
          "score": 0.9375
        }
      ]
    }
  ],
  "errors": []
}
```

Summary

In this article, you learned concepts and workflow for language detection using Text Analytics in Cognitive Services. The following are a quick reminder of the main points previously explained and demonstrated:

- Language detection API is available for 120 languages.
- JSON documents in the request body include an id and text.
- POST request is to a `/languages` endpoint, using a personalized access key and an endpoint that is valid for your subscription.

- Response output, which consists of language identifiers for each document ID, can be streamed to any app that accepts JSON, including Excel and Power BI, to name a few.

See also

Text Analytics overview
Frequently asked questions (FAQ)
Text Analytics product page

Next steps

Analyze sentiment

Example: How to detect sentiment in Text Analytics

9/14/2018 • 4 minutes to read • Edit Online

The Sentiment Analysis API evaluates text input and returns a sentiment score for each document, ranging from 0 (negative) to 1 (positive).

This capability is useful for detecting positive and negative sentiment in social media, customer reviews, and discussion forums. Content is provided by you; models and training data are provided by the service.

Currently, Sentiment Analysis supports English, German, Spanish, and French. Other languages are in preview. For more information, see Supported languages.

Concepts

Text Analytics uses a machine learning classification algorithm to generate a sentiment score between 0 and 1. Scores closer to 1 indicate positive sentiment, while scores closer to 0 indicate negative sentiment. The model is pretrained with an extensive body of text with sentiment associations. Currently, it is not possible to provide your own training data. The model uses a combination of techniques during text analysis, including text processing, part-of-speech analysis, word placement, and word associations. For more information about the algorithm, see Introducing Text Analytics.

Sentiment analysis is performed on the entire document, as opposed to extracting sentiment for a particular entity in the text. In practice, there is a tendency for scoring accuracy to improve when documents contain one or two sentences rather than a large block of text. During an objectivity assessment phase, the model determines whether a document as a whole is objective or contains sentiment. A document that is mostly objective does not progress to the sentiment detection phrase, resulting in a .50 score, with no further processing. For documents continuing in the pipeline, the next phase generates a score above or below .50, depending on the degree of sentiment detected in the document.

Preparation

Sentiment analysis produces a higher quality result when you give it smaller chunks of text to work on. This is opposite from key phrase extraction, which performs better on larger blocks of text. To get the best results from both operations, consider restructuring the inputs accordingly.

You must have JSON documents in this format: id, text, language

Document size must be under 5,000 characters per document, and you can have up to 1,000 items (IDs) per collection. The collection is submitted in the body of the request. The following is an example of content you might submit for sentiment analysis.

```
    {
        "documents": [
            {
                "language": "en",
                "id": "1",
                "text": "We love this trail and make the trip every year. The views are breathtaking and well
worth the hike!"
            },
            {
                "language": "en",
                "id": "2",
                "text": "Poorly marked trails! I thought we were goners. Worst hike ever."
            },
            {
                "language": "en",
                "id": "3",
                "text": "Everyone in my family liked the trail but thought it was too challenging for the less
athletic among us. Not necessarily recommended for small children."
            },
            {
                "language": "en",
                "id": "4",
                "text": "It was foggy so we missed the spectacular views, but the trail was ok. Worth checking
out if you are in the area."
            },
            {
                "language": "en",
                "id": "5",
                "text": "This is my favorite trail. It has beautiful views and many places to stop and rest"
            }
        ]
    }
```

Step 1: Structure the request

Details on request definition can be found in How to call the Text Analytics API. The following points are restated for convenience:

- Create a **POST** request. Review the API documentation for this request: Sentiment Analysis API
- Set the HTTP endpoint for key phrase extraction. It must include the `/sentiment` resource: `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment`
- Set a request header to include the access key for Text Analytics operations. For more information, see How to find endpoints and access keys.
- In the request body, provide the JSON documents collection you prepared for this analysis.

TIP

Use Postman or open the **API testing console** in the documentation to structure the request and POST it to the service.

Step 2: Post the request

Analysis is performed upon receipt of the request. The service accepts up to 100 requests per minute. Each request can be a maximum of 1 MB.

Recall that the service is stateless. No data is stored in your account. Results are returned immediately in the response.

Step 3: View results

The sentiment analyzer classifies text as predominantly positive or negative, assigning a score in the range of 0 to 1. Values close to 0.5 are neutral or indeterminate. A score of 0.5 indicates neutrality. When a string cannot be analyzed for sentiment or has no sentiment, the score is always 0.5 exactly. For example, if you pass in a Spanish string with an English language code, the score is 0.5.

Output is returned immediately. You can stream the results to an application that accepts JSON or save the output to a file on the local system, and then import it into an application that allows you to sort, search, and manipulate the data.

The following example shows the response for the document collection in this article.

```
{
    "documents": [
        {
            "score": 0.9999237060546875,
            "id": "1"
        },
        {
            "score": 0.0000540316104888916,
            "id": "2"
        },
        {
            "score": 0.99990355968475342,
            "id": "3"
        },
        {
            "score": 0.980544924736023,
            "id": "4"
        },
        {
            "score": 0.99996328353881836,
            "id": "5"
        }
    ],
    "errors": []
}
```

Summary

In this article, you learned concepts and workflow for sentiment analysis using Text Analytics in Cognitive Services. In summary:

- Sentiment analysis API is available for selected languages.
- JSON documents in the request body include an id, text, and language code.
- POST request is to a `/sentiment` endpoint, using a personalized access key and an endpoint that is valid for your subscription.
- Response output, which consists of a sentiment score for each document ID, can be streamed to any app that accepts JSON, including Excel and Power BI, to name a few.

See also

Text Analytics overview
Frequently asked questions (FAQ)
Text Analytics product page

Next steps

Extract key phrases

Example: How to extract key phrases in Text Analytics

9/14/2018 • 3 minutes to read • Edit Online

The Key Phrase Extraction API evaluates unstructured text, and for each JSON document, returns a list of key phrases.

This capability is useful if you need to quickly identify the main points in a collection of documents. For example, given input text "The food was delicious and there were wonderful staff", the service returns the main talking points: "food" and "wonderful staff".

Currently, Key Phrase Extraction supports English, German, Spanish, and Japanese. Other languages are in preview. For more information, see Supported languages.

Preparation

Key phrase extraction works best when you give it bigger chunks of text to work on. This is opposite from sentiment analysis, which performs better on smaller blocks of text. To get the best results from both operations, consider restructuring the inputs accordingly.

You must have JSON documents in this format: id, text, language

Document size must be under 5,000 characters per document, and you can have up to 1,000 items (IDs) per collection. The collection is submitted in the body of the request. The following example is an illustration of content you might submit for key phrase extraction.

```
    {
        "documents": [
            {
                "language": "en",
                "id": "1",
                "text": "We love this trail and make the trip every year. The views are breathtaking and well
worth the hike!"
            },
            {
                "language": "en",
                "id": "2",
                "text": "Poorly marked trails! I thought we were goners. Worst hike ever."
            },
            {
                "language": "en",
                "id": "3",
                "text": "Everyone in my family liked the trail but thought it was too challenging for the less
athletic among us. Not necessarily recommended for small children."
            },
            {
                "language": "en",
                "id": "4",
                "text": "It was foggy so we missed the spectacular views, but the trail was ok. Worth checking
out if you are in the area."
            },
            {
                "language": "en",
                "id": "5",
                "text": "This is my favorite trail. It has beautiful views and many places to stop and rest"
            }
        ]
    }
```

Step 1: Structure the request

Details on request definition can be found in How to call the Text Analytics API. The following points are restated for convenience:

- Create a **POST** request. Review the API documentation for this request: Key Phrases API
- Set the HTTP endpoint for key phrase extraction. It must include the `/keyphrases` resource: `https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases`
- Set a request header to include the access key for Text Analytics operations. For more information, see How to find endpoints and access keys.
- In the request body, provide the JSON documents collection you prepared for this analysis

TIP

Use Postman or open the **API testing console** in the documentation to structure a request and POST it to the service.

Step 2: Post the request

Analysis is performed upon receipt of the request. The service accepts up to 100 requests per minute. Each request can be a maximum of 1 MB.

Recall that the service is stateless. No data is stored in your account. Results are returned immediately in the response.

Step 3: View results

All POST requests return a JSON formatted response with the IDs and detected properties.

Output is returned immediately. You can stream the results to an application that accepts JSON or save the output to a file on the local system, and then import it into an application that allows you to sort, search, and manipulate the data.

An example of the output for key phrase extraction is shown next:

```
    "documents": [
        {
            "keyPhrases": [
                "year",
                "trail",
                "trip",
                "views"
            ],
            "id": "1"
        },
        {
            "keyPhrases": [
                "marked trails",
                "Worst hike",
                "goners"
            ],
            "id": "2"
        },
        {
            "keyPhrases": [
                "trail",
                "small children",
                "family"
            ],
            "id": "3"
        },
        {
            "keyPhrases": [
                "spectacular views",
                "trail",
                "area"
            ],
            "id": "4"
        },
        {
            "keyPhrases": [
                "places",
                "beautiful views",
                "favorite trail"
            ],
            "id": "5"
        }
```

As noted, the analyzer finds and discards non-essential words, and keeps single terms or phrases that appear to be the subject or object of a sentence.

Summary

In this article, you learned concepts and workflow for key phrase extraction using Text Analytics in Cognitive Services. In summary:

- Key phrase extraction API is available for selected languages.
- JSON documents in the request body include an id, text, and language code.
- POST request is to a `/keyphrases` endpoint, using a personalized access key and an endpoint that is valid for your subscription.
- Response output, which consists of key words and phrases for each document ID, can be streamed to any app that accepts JSON, including Excel and Power BI, to name a few.

See also

Text Analytics overview

Frequently asked questions (FAQ)

Text Analytics product page

Next steps

Text Analytics API

How to use Named Entity Recognition in Text Analytics (Preview)

10/29/2018 • 5 minutes to read • Edit Online

The Entity Recognition API takes unstructured text, and for each JSON document, returns a list of disambiguated entities with links to more information on the web (Wikipedia and Bing).

Entity Linking and Named Entity Recognition

The Text Analytics' `entities` endpoint supprts both named entity recognition (NER) and entity linking.

Entity Linking

Entity linking is the ability to identify and disambiguate the identity of an entity found in text (for example, determining whether the "Mars" is being used as the planet or as the Roman god of war). This process requires the presence of a knowledge base to which recognized entities are linked - Wikipedia is used as the knowledge base for the `entities` endpoint Text Analytics.

In Text Analytics Version 2.0, only entity linking is available.

Named Entity Recognition (NER)

Named entity recognition (NER) is the ability to identify different entities in text and categorize them into pre-defined classes. The supported classes of entities are listed below.

In Text Analytics Version 2.1-Preview, both entity linking and named entity recognition (NER) are available.

Language support

Using entity linking in various languages requires using a corresponding knowledge base in each language. For entity linking in Text Analytics, this means each language that is supported by the `entities` endpoint will link to the corresponding Wikipedia corpus in that language. Since the size of corpora varies between languages, it is expected that the entity linking functionality's recall will also vary.

Supported Types for Named Entity Recognition

TYPE	SUBTYPE	EXAMPLE
Person	N/A*	"Jeff", "Bill Gates"
Location	N/A*	"Redmond, Washington", "Paris"
Organization	N/A*	"Microsoft"
Quantity	Number	"6", "six"
Quantity	Percentage	"50%", "fifty percent"
Quantity	Ordinal	"2nd", "second"
Quantity	NumberRange	"4 to 8"

TYPE	SUBTYPE	EXAMPLE
Quantity	Age	"90 day old", "30 years old"
Quantity	Currency	"$10.99"
Quantity	Dimension	"10 miles", "40 cm"
Quantity	Temperature	"32 degrees"
DateTime	N/A*	"6:30PM February 4, 2012"
DateTime	Date	"May 2nd, 2017", "05/02/2017"
Date Time	Time	"8am", "8:00"
DateTime	DateRange	"May 2nd to May 5th"
DateTime	TimeRange	"6pm to 7pm"
DateTime	Duration	"1 minute and 45 seconds"
DateTime	Set	"every Tuesday"
DateTime	TimeZone	
URL	N/A*	"http://www.bing.com"
Email	N/A*	"support@contoso.com"

* Depending on the input and extracted entities, certain entities may omit the `SubType`.

Preparation

You must have JSON documents in this format: id, text, language

For currently supported languages, see this list.

Document size must be under 5,000 characters per document, and you can have up to 1,000 items (IDs) per collection. The collection is submitted in the body of the request. The following example is an illustration of content you might submit to the entity linking end.

```
{"documents": [{"id": "1",
                "language": "en",
                "text": "Jeff bought three dozen eggs because there was a 50% discount."
                },
               {"id": "2",
                "language": "en",
                "text": "The Great Depression began in 1929. By 1933, the GDP in America fell by 25%."
                }
               ]
}
```

Step 1: Structure the request

Details on request definition can be found in How to call the Text Analytics API. The following points are restated for convenience:

- Create a **POST** request. Review the API documentation for this request: Entity Linking API
- Set the HTTP endpoint for entity extraction. It must include the `/entities` resource: `https://[your-region].api.cognitive.microsoft.com/text/analytics/v2.1-preview/entities`
- Set a request header to include the access key for Text Analytics operations. For more information, see How to find endpoints and access keys.
- In the request body, provide the JSON documents collection you prepared for this analysis

TIP

Use Postman or open the **API testing console** in the documentation to structure a request and POST it to the service.

Step 2: Post the request

Analysis is performed upon receipt of the request. The service accepts up to 100 requests per minute. Each request can be a maximum of 1 MB.

Recall that the service is stateless. No data is stored in your account. Results are returned immediately in the response.

Step 3: View results

All POST requests return a JSON formatted response with the IDs and detected properties.

Output is returned immediately. You can stream the results to an application that accepts JSON or save the output to a file on the local system, and then import it into an application that allows you to sort, search, and manipulate the data.

An example of the output for entity linking is shown next:

```
{
    "Documents": [
        {
            "Id": "1",
            "Entities": [
                {
                    "Name": "Jeff",
                    "Matches": [
                        {
                            "Text": "Jeff",
                            "Offset": 0,
                            "Length": 4
                        }
                    ],
                    "Type": "Person"
                },
                {
                    "Name": "three dozen",
                    "Matches": [
                        {
                            "Text": "three dozen",
                            "Offset": 12,
                            "Length": 11
                        }
                    ],
                    "Type": "Quantity",
                "SubType": "Number"
            },
            {
                "Name": "50",
                "Matches": [
                    {
                        "Text": "50",
                        "Offset": 49,
                        "Length": 2
                    }
                ],
                "Type": "Quantity",
                "SubType": "Number"
            },
            {
                "Name": "50%",
                "Matches": [
                    {
                        "Text": "50%",
                        "Offset": 49,
                        "Length": 3
                    }
                ],
                "Type": "Quantity",
                "SubType": "Percentage"
            }
        ]
    },
    {
        "Id": "2",
        "Entities": [
            {
                "Name": "Great Depression",
                "Matches": [
                    {
                        "Text": "The Great Depression",
                        "Offset": 0,
                        "Length": 20
                    }
                ],
                "WikipediaLanguage": "en",
                "WikipediaId": "Great Depression",
                "WikipediaUrl": "https://en.wikipedia.org/wiki/Great_Depression",
                "BingId": "d9364681-98ad-1a66-f869-a3f1c8ae8ef8"
            },
            {
                "Name": "1929",
                "Matches": [
                    {
                        "Text": "1929",
                        "Offset": 30,
                        "Length": 4
                    }
                ],
                "Type": "DateTime",
                "SubType": "DateRange"
            },
            {
                "Name": "By 1933",
                "Matches": [
                    {
                        "Text": "By 1933",
                        "Offset": 36,
                        "Length": 7
                    }
                ],
                "Type": "DateTime",
                "SubType": "DateRange"
            },
                {
                    "Name": "Gross domestic product",
                    "Matches": [
                        {
                            "Text": "GDP",
                            "Offset": 49,
                            "Length": 3
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "Gross domestic product",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/Gross_domestic_product",
                    "BingId": "c859ed84-c0dd-e18f-394a-530cae5468a2"
                },
                {
                    "Name": "United States",
                    "Matches": [
                        {
                            "Text": "America",
                            "Offset": 56,
                            "Length": 7
                        }
                    ],
                    "WikipediaLanguage": "en",
                    "WikipediaId": "United States",
                    "WikipediaUrl": "https://en.wikipedia.org/wiki/United_States",
                    "BingId": "5232ed96-85b1-2edb-12c6-63e6c597a1de",
                    "Type": "Location"
                },
                {
                    "Name": "25",
                    "Matches": [
                        {
                            "Text": "25",
                            "Offset": 72,
                            "Length": 2
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Number"
                },
                {
                    "Name": "25%",
                    "Matches": [
                        {
                            "Text": "25%",
                            "Offset": 72,
                            "Length": 3
                        }
                    ],
                    "Type": "Quantity",
                    "SubType": "Percentage"
                }
            ]
        }
    ],
    "Errors": []
}
```

Summary

In this article, you learned concepts and workflow for entity linking using Text Analytics in Cognitive Services. In summary:

- Entities API is available for selected languages.

- JSON documents in the request body include an id, text, and language code.
- POST request is to a `/entities` endpoint, using a personalized access key and an endpoint that is valid for your subscription.
- Response output, which consists of linked entities (including confidence scores, offsets, and web links, for each document ID) can be used in any application

See also

Text Analytics overview
Frequently asked questions (FAQ)
Text Analytics product page

Next steps

Text Analytics API

Tutorial: Connect to the Text Analytics Cognitive Service by using Connected Services in Visual Studio

9/14/2018 • 5 minutes to read • Edit Online

By using the Text Analytics Service, you can extract rich information to categorize and process visual data, and perform machine-assisted moderation of images to help curate your services.

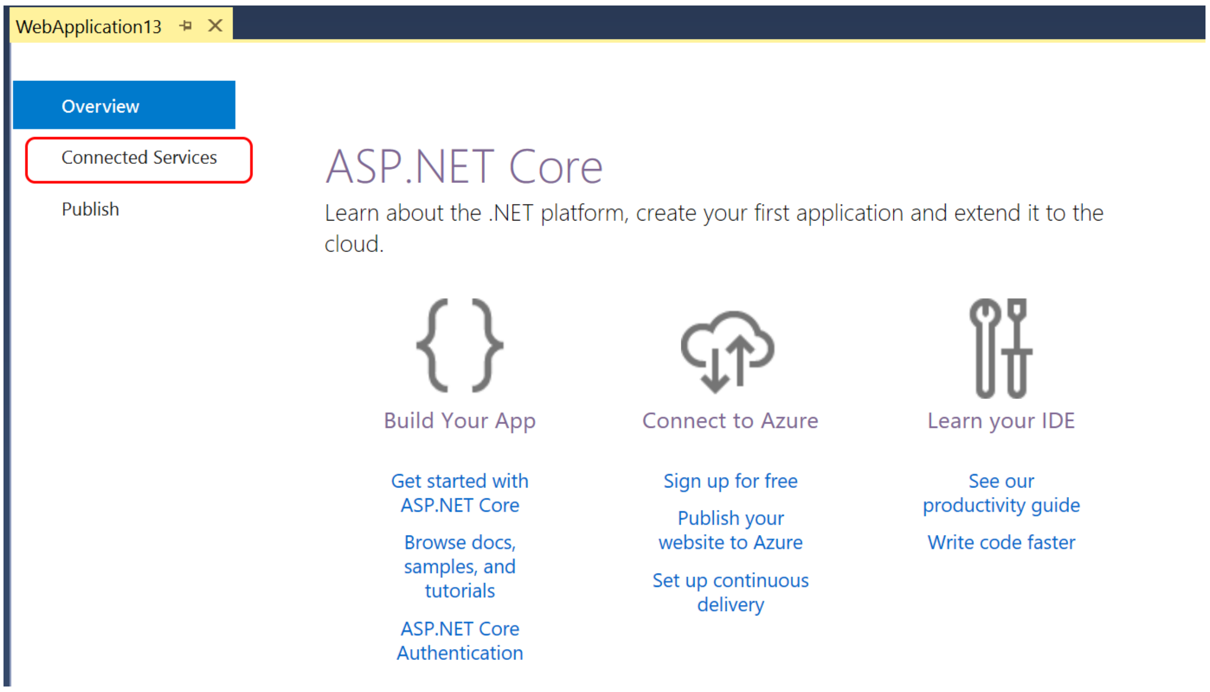
This article and its companion articles provide details for using the Visual Studio Connected Service feature for the Text Analytics Service. The capability is available in both Visual Studio 2017 15.7 or later, with the Cognitive Services extension installed.

Prerequisites

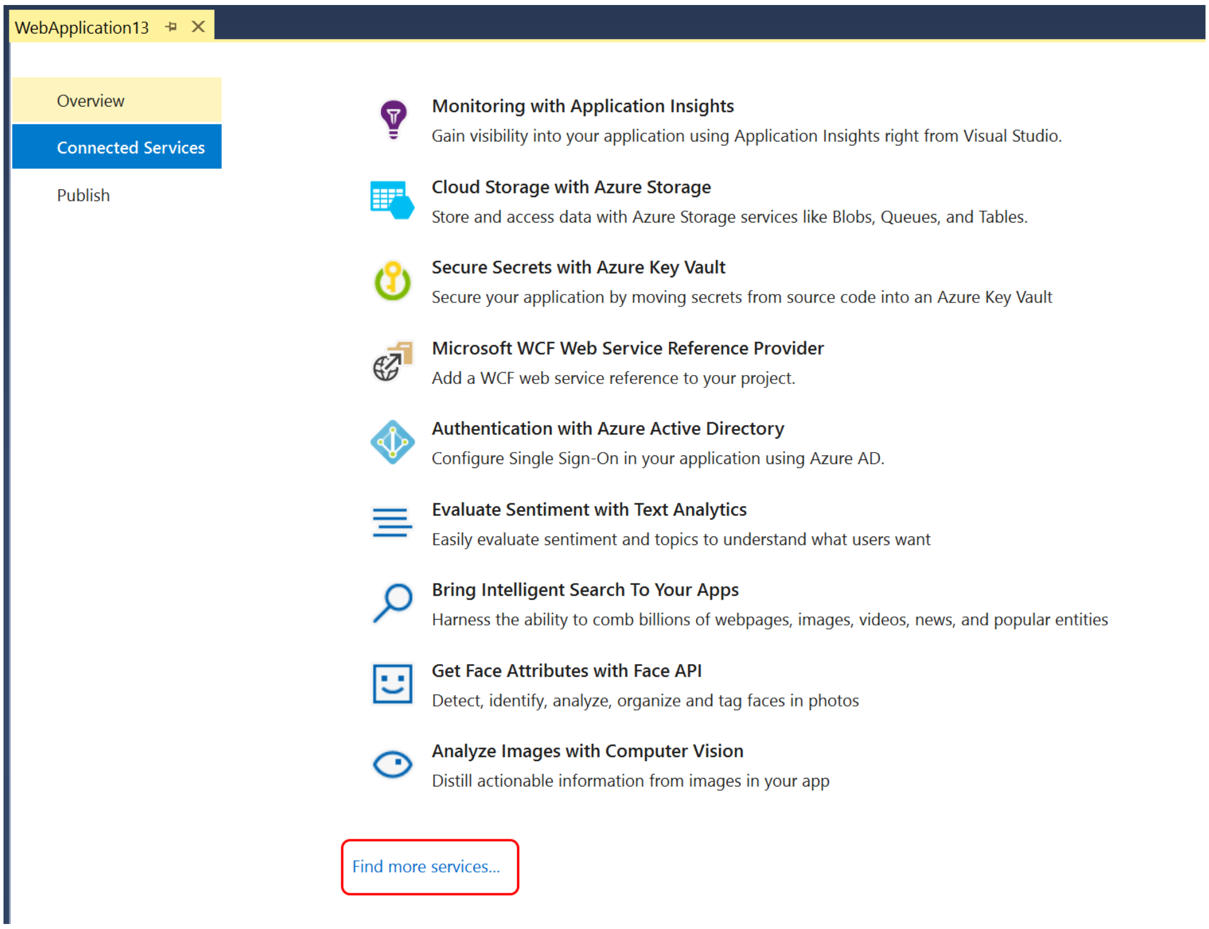
- An Azure subscription. If you do not have one, you can sign up for a free account.
- Visual Studio 2017 version 15.7, with the Web Development workload installed. Download it now.

Install the Cognitive Services VSIX Extension

1. With your web project open in Visual Studio, choose the **Connected Services** tab. The tab is available on the welcome page that appears when you open a new project. If you don't see the tab, select **Connected Services** in your project in Solution Explorer.


WebApplication13
Overview
Connected Services
Publish
ASP.NET Core
Learn about the .NET platform, create your first application and extend it to the cloud.
Build Your App
Get started with ASP.NET Core
Browse docs, samples, and tutorials
ASP.NET Core Authentication
Connect to Azure
Sign up for free
Publish your website to Azure
Set up continuous delivery
Learn your IDE
See our productivity guide
Write code faster

2. Scroll down to the bottom of the list of services, and select **Find more services**.


WebApplication13

Overview

Connected Services

Publish

Monitoring with Application Insights
Gain visibility into your application using Application Insights right from Visual Studio.

Cloud Storage with Azure Storage
Store and access data with Azure Storage services like Blobs, Queues, and Tables.

Secure Secrets with Azure Key Vault
Secure your application by moving secrets from source code into an Azure Key Vault

Microsoft WCF Web Service Reference Provider
Add a WCF web service reference to your project.

Authentication with Azure Active Directory
Configure Single Sign-On in your application using Azure AD.

Evaluate Sentiment with Text Analytics
Easily evaluate sentiment and topics to understand what users want

Bring Intelligent Search To Your Apps
Harness the ability to comb billions of webpages, images, videos, news, and popular entities

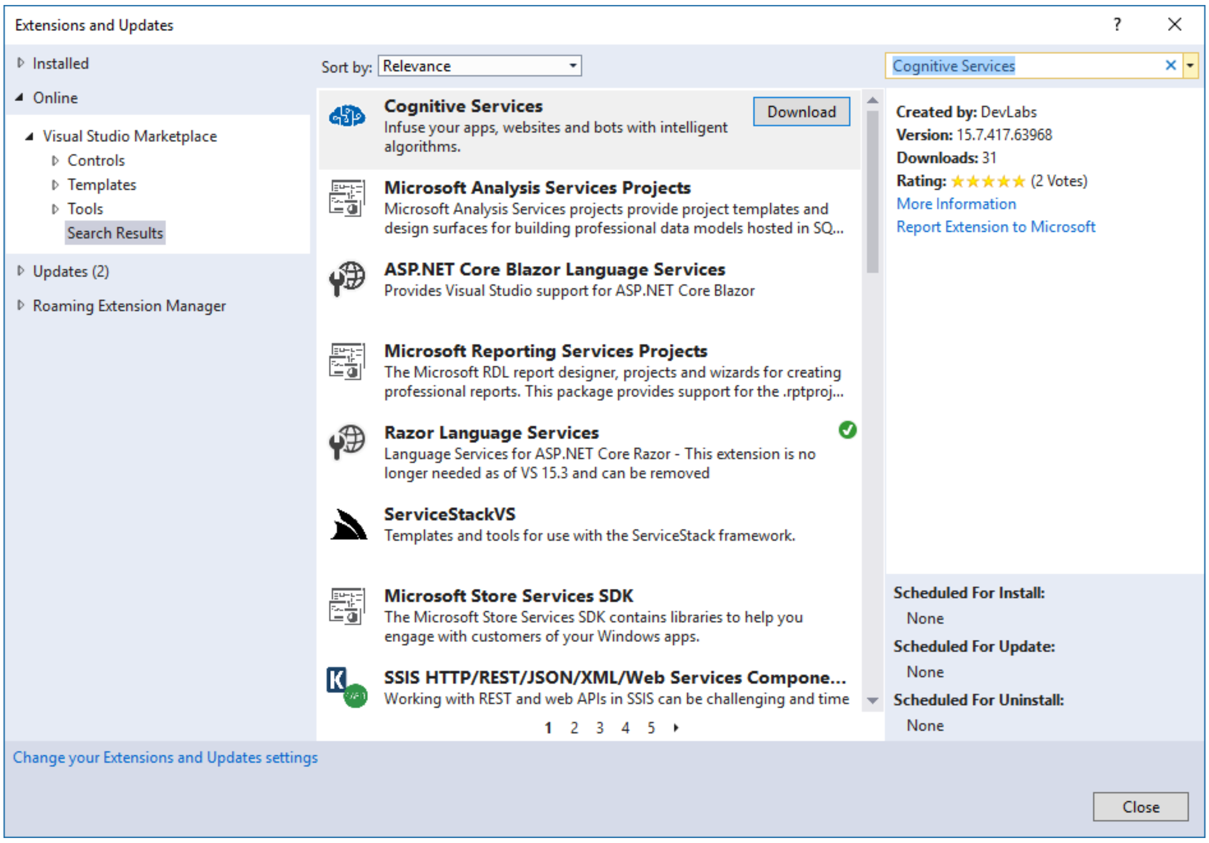
Get Face Attributes with Face API
Detect, identify, analyze, organize and tag faces in photos

Analyze Images with Computer Vision
Distill actionable information from images in your app

Find more services...

The **Extensions and Updates** dialog box appears.

3. In the **Extensions and Updates** dialog box, search for **Cognitive Services**, and then download and install the Cognitive Services VSIX package.


Extensions and Updates

Installed
Online
Visual Studio Marketplace
Controls
Templates
Tools
Search Results
Updates (2)
Roaming Extension Manager

Sort by: Relevance

Cognitive Services

Cognitive Services — Download
Infuse your apps, websites and bots with intelligent algorithms.

Microsoft Analysis Services Projects
Microsoft Analysis Services projects provide project templates and design surfaces for building professional data models hosted in SQ...

ASP.NET Core Blazor Language Services
Provides Visual Studio support for ASP.NET Core Blazor

Microsoft Reporting Services Projects
The Microsoft RDL report designer, projects and wizards for creating professional reports. This package provides support for the .rptproj...

Razor Language Services
Language Services for ASP.NET Core Razor - This extension is no longer needed as of VS 15.3 and can be removed

ServiceStackVS
Templates and tools for use with the ServiceStack framework.

Microsoft Store Services SDK
The Microsoft Store Services SDK contains libraries to help you engage with customers of your Windows apps.

SSIS HTTP/REST/JSON/XML/Web Services Compone...
Working with REST and web APIs in SSIS can be challenging and time

1 2 3 4 5

Created by: DevLabs
Version: 15.7.417.63968
Downloads: 31
Rating: ★★★★★ (2 Votes)
More Information
Report Extension to Microsoft

Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

Change your Extensions and Updates settings

Close

Installing an extension requires a restart of the integrated development environment (IDE).

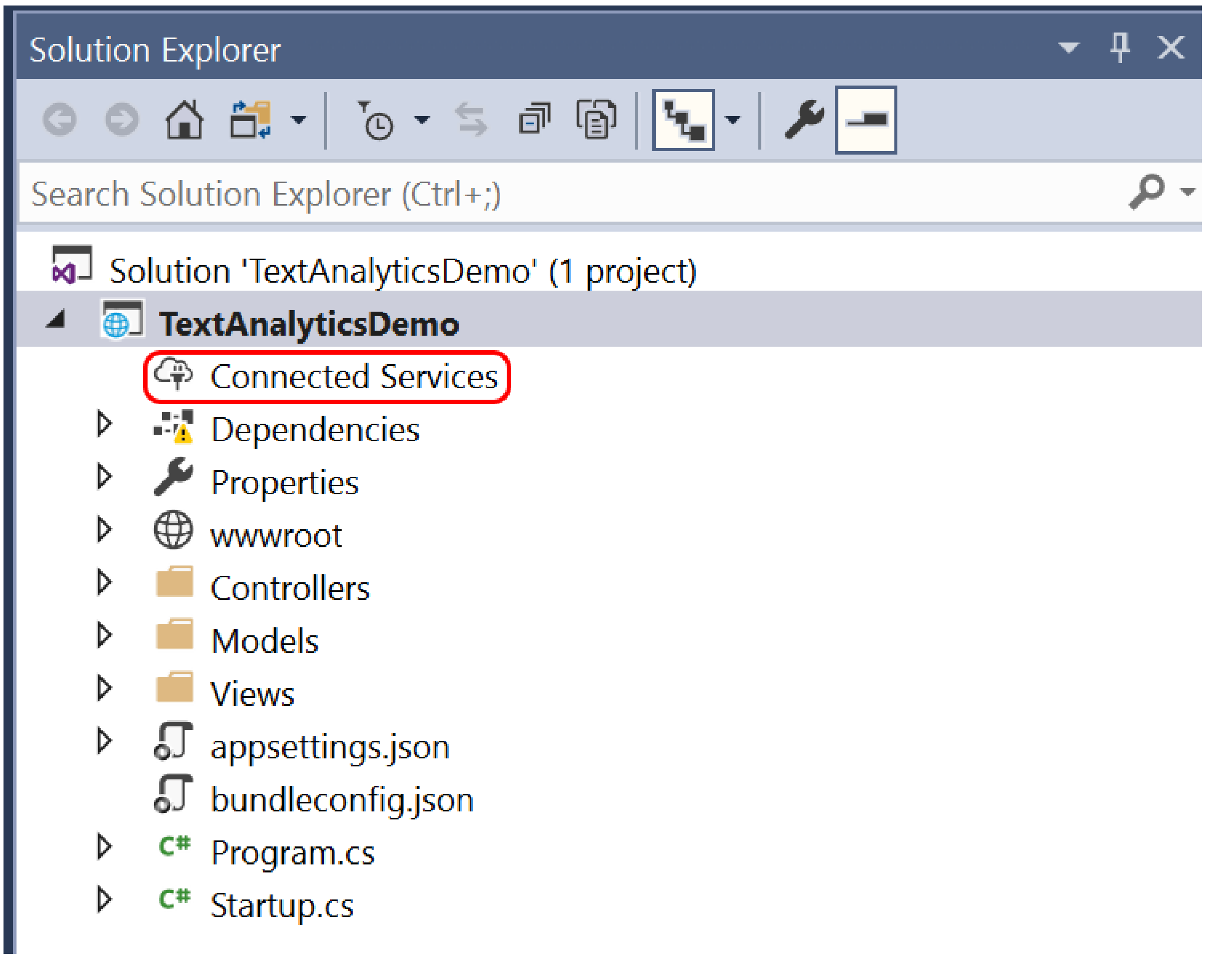
4. Restart Visual Studio. The extension installs when you close Visual Studio, and is available next time you

launch the IDE.

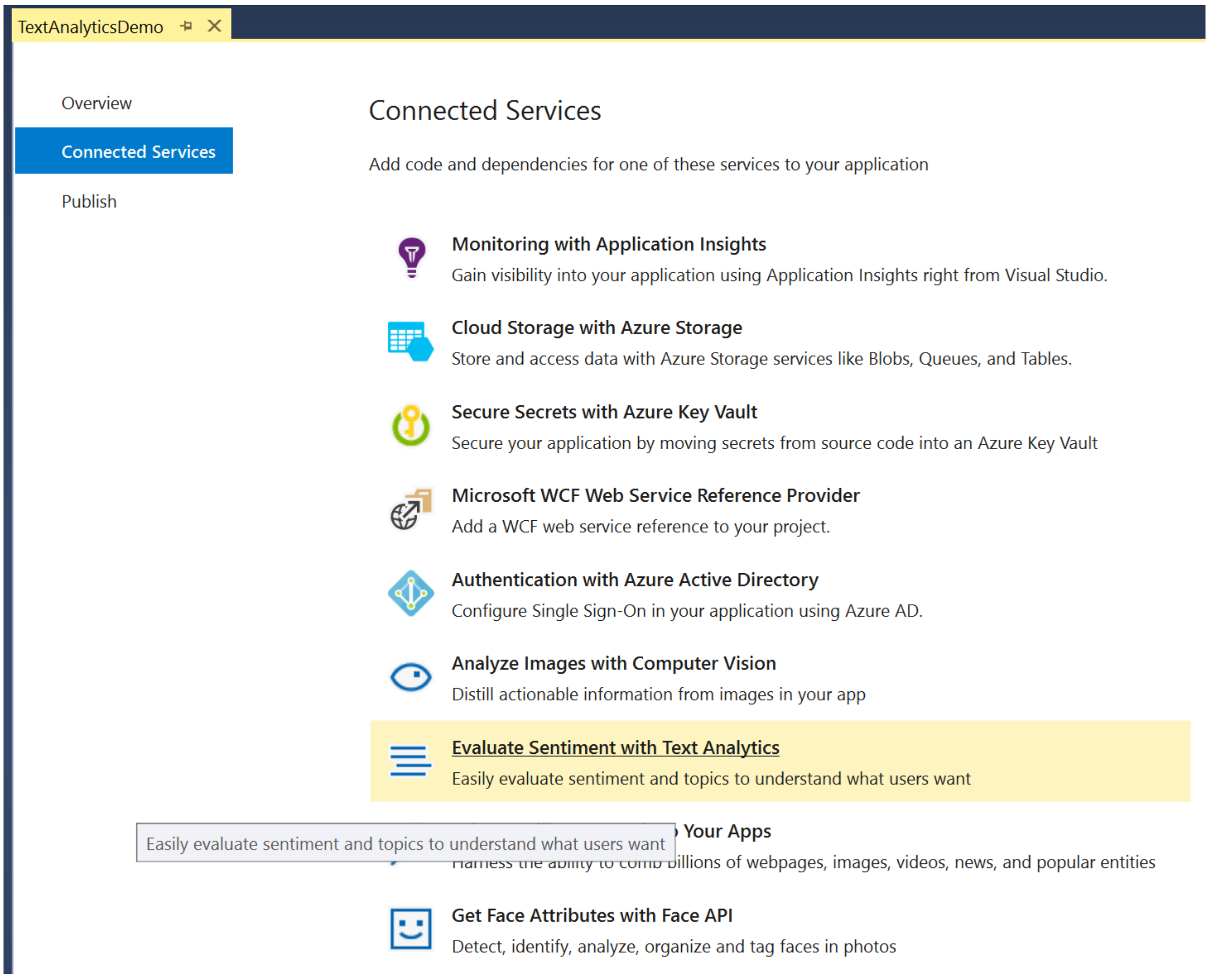
Add support to your project for the Text Analytics Service

1. Create a new ASP.NET Core web project called TextAnalyticsDemo. Use the Web Application (Model-View-Controller) project template with all the default settings. It's important to name the project MyWebApplication, so the namespace matches when you copy code into the project. The example in this articles uses MVC, but you can use the Text Analytics Connected Service with any ASP.NET project type.

2. In **Solution Explorer**, double-click on the **Connected Service** item. The Connected Service page appears, with services you can add to your project.



3. In the menu of available services, choose **Evaluate Sentiment with Text Analytics**.



TextAnalyticsDemo

Overview

Connected Services

Publish

Connected Services

Add code and dependencies for one of these services to your application

Monitoring with Application Insights
Gain visibility into your application using Application Insights right from Visual Studio.

Cloud Storage with Azure Storage
Store and access data with Azure Storage services like Blobs, Queues, and Tables.

Secure Secrets with Azure Key Vault
Secure your application by moving secrets from source code into an Azure Key Vault

Microsoft WCF Web Service Reference Provider
Add a WCF web service reference to your project.

Authentication with Azure Active Directory
Configure Single Sign-On in your application using Azure AD.

Analyze Images with Computer Vision
Distill actionable information from images in your app

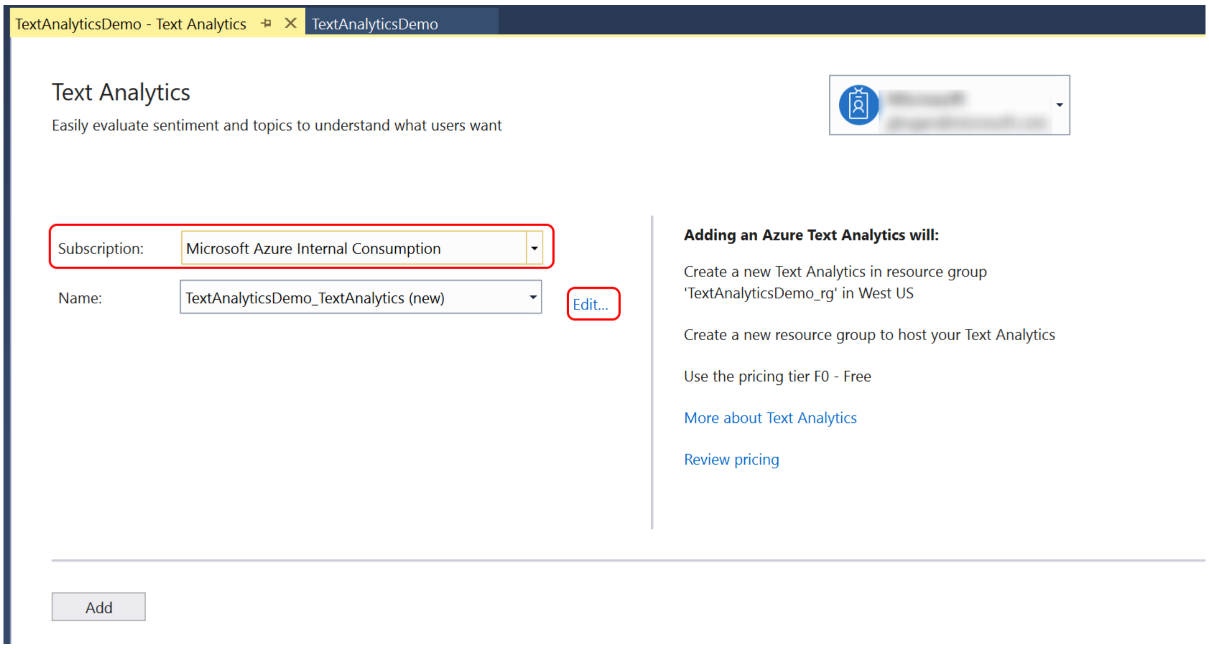
Evaluate Sentiment with Text Analytics
Easily evaluate sentiment and topics to understand what users want

Easily evaluate sentiment and topics to understand what users want

Your Apps
billions of webpages, images, videos, news, and popular entities

Get Face Attributes with Face API
Detect, identify, analyze, organize and tag faces in photos

If you've signed into Visual Studio, and have an Azure subscription associated with your account, a page appears with a dropdown list with your subscriptions.



TextAnalyticsDemo - Text Analytics TextAnalyticsDemo

Text Analytics

Easily evaluate sentiment and topics to understand what users want

Subscription: Microsoft Azure Internal Consumption

Name: TextAnalyticsDemo_TextAnalytics (new) Edit...

Adding an Azure Text Analytics will:

Create a new Text Analytics in resource group 'TextAnalyticsDemo_rg' in West US

Create a new resource group to host your Text Analytics

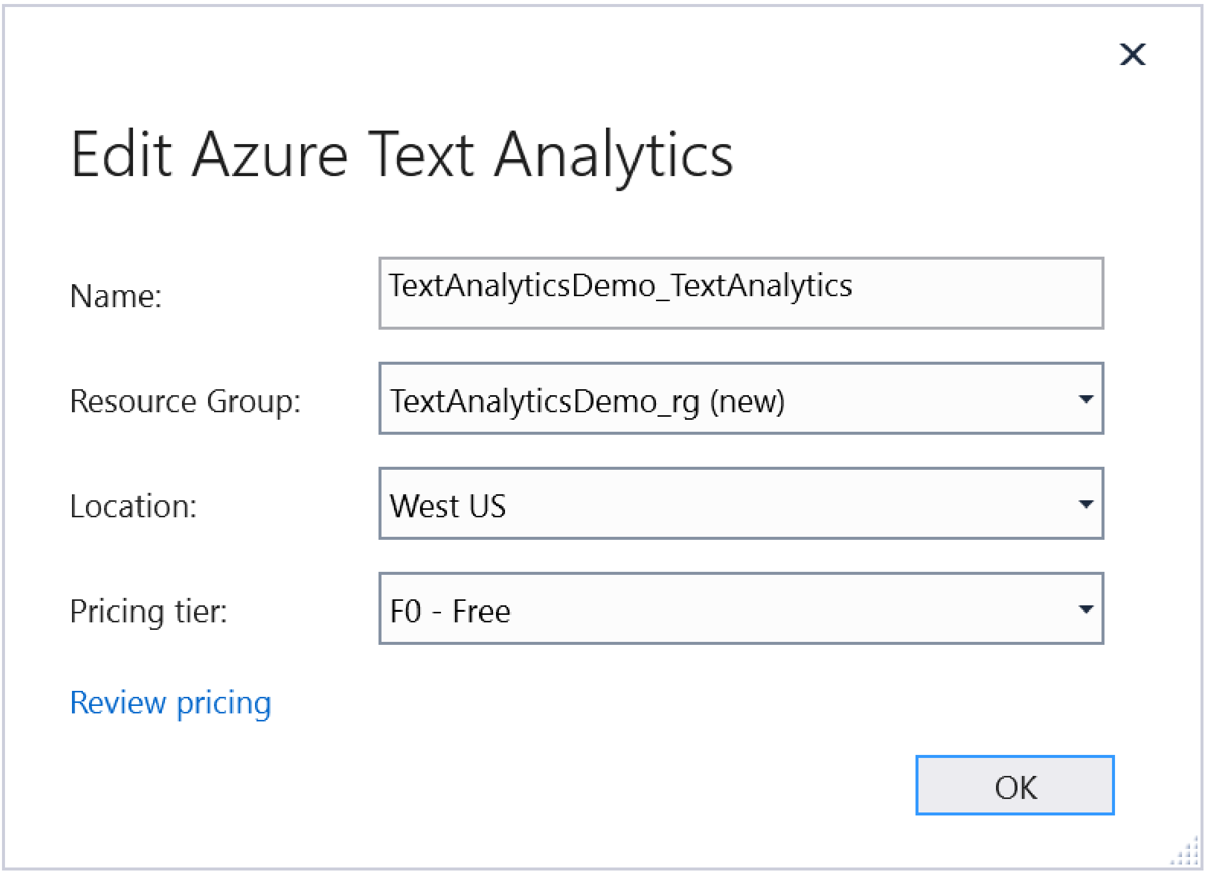
Use the pricing tier F0 - Free

More about Text Analytics

Review pricing

Add

4. Select the subscription you want to use, and then choose a name for the Text Analytics Service, or choose the **Edit** link to modify the automatically generated name, choose the resource group, and the Pricing Tier.


Edit Azure Text Analytics

Name: TextAnalyticsDemo_TextAnalytics

Resource Group: TextAnalyticsDemo_rg (new)

Location: West US

Pricing tier: F0 - Free

Follow the link for details on the pricing tiers.

5. Choose **Add** to add support for the Connected Service. Visual Studio modifies your project to add the NuGet packages, configuration file entries, and other changes to support a connection to the Text Analytics Service. The **Output Window** shows the log of what is happening to your project. You should see something like the following:

```
 [6/1/2018 3:04:02.347 PM] Adding Text Analytics to the project.
 [6/1/2018 3:04:02.906 PM] Creating new Text Analytics...
 [6/1/2018 3:04:06.314 PM] Installing NuGet package 'Microsoft.Azure.CognitiveServices.Language' version
1.0.0-preview...
 [6/1/2018 3:04:56.759 PM] Retrieving keys...
 [6/1/2018 3:04:57.822 PM] Updating appsettings.json setting: 'ServiceKey' = '<service key>'
 [6/1/2018 3:04:57.827 PM] Updating appsettings.json setting: 'ServiceEndPoint' =
'https://westus.api.cognitive.microsoft.com/text/analytics/v2.0'
 [6/1/2018 3:04:57.832 PM] Updating appsettings.json setting: 'Name' = 'TextAnalyticsDemo'
 [6/1/2018 3:05:01.840 PM] Successfully added Text Analytics to the project.
```

Use the Text Analytics Service to detect the language for a text sample.

1. Add the following using statements in Startup.cs.

```
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using Microsoft.Extensions.Configuration;
```

2. Add a configuration field, and add a constructor that initializes the configuration field in the Startup class to enable Configuration in your program.

```
    private IConfiguration configuration;

    public Startup(IConfiguration configuration)
    {
        this.configuration = configuration;
    }
```

3. Add a class file in the Controllers folder called DemoTextAnalyzeController and replace its contents with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics;
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics.Models;
using Microsoft.Extensions.Configuration;
using TextAnalyticsDemo.Models;

namespace TextAnalyticsDemo.Controllers
{
    public class DemoTextAnalyzeController : Controller
    {
        private IConfiguration configuration;

        public DemoTextAnalyzeController(IConfiguration configuration)
        {
            this.configuration = configuration;
        }

        public IActionResult TextAnalyzeResult(TextAnalyzeModel model)
        {

            if (!string.IsNullOrWhiteSpace(model.TextStr))
            {
                ITextAnalyticsAPI client = this.GetTextAnalyzeClient(new MyHandler());
                model.AnalyzeResult = client.DetectLanguage(
                    new BatchInput(
                        new List<Input>()
                        {
                            new Input("id",model.TextStr)
                        }));
            }
            return View(model);
        }

        [HttpPost("Analyze")]
        public IActionResult Analyze(TextAnalyzeModel model)
        {
            return RedirectToAction("TextAnalyzeResult", model);
        }

        // Using the ServiceKey from the configuration file,
        // get an instance of the Text Analytics client.
        private ITextAnalyticsAPI GetTextAnalyzeClient(DelegatingHandler handler)
        {
            string key = configuration.GetSection("CognitiveServices")["TextAnalytics:ServiceKey"];

            ITextAnalyticsAPI client = new TextAnalyticsAPI( handlers: handler);
            client.SubscriptionKey = key;
            client.AzureRegion = AzureRegions.Westus;

            return client;
        }
    }
}
```

The code includes GetTextAnalyzeClient to get the client object which you can use to call the Text Analytics API, and a request handler that calls DetectLanguage on a given text.

4. Add the MyHandler helper class which is used by the preceding code.

```
    class MyHandler : DelegatingHandler
    {
        protected async override Task<HttpResponseMessage> SendAsync(
        HttpRequestMessage request, CancellationToken cancellationToken)
        {
            // Call the inner handler.
            var response = await base.SendAsync(request, cancellationToken);

            return response;
        }
    }
```

5. In the Models folder, add a class for the model.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics.Models;

namespace Demo.Models
{
    public class TextAnalyzeModel
    {
        public string TextStr { get; set; }

        public LanguageBatchResult AnalyzeResult { get; set; }

        public SentimentBatchResult AnalyzeResult2 { get; set; }
    }
}
```

6. Add a View to show the analyzed text, the language determined, and the score that represents the confidence level in the analysis. To do this, right-click on the **Views** folder, choose **Add**, then **View**. In the dialog box that appears, provide a name *TextAnalyzeResult*, accept the defaults to add a new file called *TextAnalyzeResult.cshtml* in the **Views** folder and copy the following contents into it:

```
@using System
@model Demo.Models.TextAnalyzeModel

@{
    ViewData["Title"] = "TextAnalyzeResult";
}

<h2>Text Language</h2>

<div class="row">
    <section>
        <form asp-controller="DemoTextAnalyze" asp-action="Analyze" method="POST"
              class="form-horizontal" enctype="multipart/form-data">
            <table width="90%">
                <tr>
                    <td>
                        <input type="text" name="TextStr" class="form-control" />
                    </td>
                    <td>
                        <button type="submit" class="btn btn-default">Analyze</button>
                    </td>
                </tr>
            </table>
        </form>
    </section>
</div>

<h2>Result</h2>
<div>
    <dl class="dl-horizontal">
        <dt>
            Text :
        </dt>
        <dd>
            @Html.DisplayFor(model => model.TextStr)
        </dd>
        <dt>
            Language Name :
        </dt>
        <dd>
            @Html.DisplayFor(model => model.AnalyzeResult.Documents[0].DetectedLanguages[0].Name)
        </dd>
        <dt>
            Score :
        </dt>
        <dd>
            @Html.DisplayFor(model => model.AnalyzeResult.Documents[0].DetectedLanguages[0].Score)
        </dd>
    </dl>
</div>
<div>
    <hr />
    <p>
        <a asp-controller="Home" asp-action="Index">Return to Index</a>
    </p>
</div>
```

7. Build and run the example locally. Enter some text and see what language Text Analytics detects.

Clean up resources

When no longer needed, delete the resource group. This deletes the cognitive service and related resources. To delete the resource group through the portal:

1. Enter the name of your resource group in the Search box at the top of the portal. When you see the resource

group used in this tutorial in the search results, select it.

2. Select **Delete resource group**.
3. In the **TYPE THE RESOURCE GROUP NAME:** box type in the name of the resource group and select **Delete**.

Next steps

Learn more about the Text Analytics Service by reading the Text Analytics Service Documentation.

External & community content for the Text Analytics Cognitive Service

9/14/2018 • 2 minutes to read • Edit Online

Links in this article lead you to helpful web content developed and produced by partners and professionals with experience in using the Text Analytics API.

Blogs

- Text Analytics API original announcement (Azure blog)
- Using Text Analytics Key Phrase Cognitive Services API from PowerShell (MSDN blog)
- R Quick tip: Microsoft Cognitive Services' Text Analytics API (R Bloggers)
- Sentiment analysis in Logic App using SQL Server data (TechNet blog)
- Sentiment analysis with Dynamics 365 CRM Online (MSDN blog)
- PowerBI blog: Extraction of key phrases from Facebook messages: Part 1 and Part 2

Videos

- Logic App to detect sentiment and extract key phrases from your text
- Sentiment Analysis using Power BI and Microsoft Cognitive Services
- Text analytics extract key phrases using Power BI and Microsoft Cognitive Services

Next steps

Are you looking for information about a feature or use-case that we don't cover? Consider requesting or voting for it on UserVoice.

See also

StackOverflow: Azure Text Analytics API
StackOverflow: Azure Cognitive Services

Frequently Asked Questions (FAQ) about the Text Analytics Cognitive Service

9/14/2018 • 2 minutes to read • Edit Online

Find answers to commonly asked questions about concepts, code, and scenarios related to the Text Analytics API for Microsoft Cognitive Services on Azure.

Can Text Analytics identify sarcasm?

Analysis is for positive-negative sentiment rather than mood detection.

There is always some degree of imprecision in sentiment analysis, but the model is most useful when there is no hidden meaning or subtext to the content. Irony, sarcasm, humor, and similarly nuanced content rely on cultural context and norms to convey intent. This type of content is among the most challenging to analyze. Typically, the greatest discrepancy between a given score produced by the analyzer and a subjective assessment by a human is for content with nuanced meaning.

Can I add my own training data or models?

No, the models are pretrained. The only operations available on uploaded data are scoring, key phrase extraction, and language detection. We do not host custom models. If you want to create and host custom machine learning models, consider the machine learning capabilities in Microsoft R Server.

Can I request additional languages?

Sentiment analysis and key phrase extraction are available for a select number of languages. Natural language processing is complex and requires substantial testing before new functionality can be released. For this reason, we avoid pre-announcing support so that no one takes a dependency on functionality that needs more time to mature.

To help us prioritize which languages to work on next, vote for specific languages on User Voice.

Why does key phrase extraction return some words but not others?

Key phrase extraction eliminates non-essential words and standalone adjectives. Adjective-noun combinations, such as "spectacular views" or "foggy weather" are returned together.

Generally, output consists of nouns and objects of the sentence. Output is listed in order of importance, with the first phrase being the most important. Importance is measured by the number of times a particular concept is mentioned, or the relation of that element to other elements in the text.

Why does output vary, given identical inputs?

Improvements to models and algorithms are announced if the change is major, or quietly slipstreamed into the service if the update is minor. Over time, you might find that the same text input results in a different sentiment score or key phrase output. This is a normal and intentional consequence of using managed machine learning resources in the cloud.

Next steps

Is your question about a missing feature or functionality? Consider requesting or voting for it on our UserVoice web site.

See also

StackOverflow: Text Analytics API
StackOverflow: Cognitive Services

Language and region support for the Text Analytics API

10/10/2018 • 2 minutes to read • Edit Online

This article explains which languages are supported for each operation: sentiment analysis, key phrase extraction, and language detection.

Language Detection

The Text Analytics API can detect up to 120 different languages. Language Detection returns the "script" of a language. For instance, for the phrase "I have a dog" it will return `en` instead of `en-US`. The only special case is Chinese, where the language detection capability will return `zh_CHS` or `zh_CHT` if it can determine the script given the text provided. In situations where a specific script cannot be identified for a Chinese document, it will return simply `zh`.

Sentiment Analysis, Key Phrase Extraction, and Entity Recognition

For sentiment analysis, key phrase extraction, and entity recognition, the list of supported languages is more selective as the analyzers are refined to accommodate the linguistic rules of additional languages.

Language list and status

Language support is initially rolled out in preview, graduating to generally available (GA) status, independently of each other and of the Text Analytics service overall. It's possible for languages to remain in preview, even while Text Analytics API transitions to generally available.

LANGUAGE	LANGUAGE CODE	SENTIMENT	KEY PHRASES	ENTITY RECOGNITION	NOTES
Danish	`da`	✔ *	✔		
Dutch	`nl`	✔ *	✔		
English	`en`	✔	✔	✔ *	
Finnish	`fi`	✔ *	✔		
French	`fr`	✔	✔		
German	`de`	✔ *	✔		
Greek	`el`	✔ *			
Italian	`it`	✔ *	✔		
Japanese	`ja`		✔		
Korean	`ko`		✔		

LANGUAGE	LANGUAGE CODE	SENTIMENT	KEY PHRASES	ENTITY RECOGNITION	NOTES
Norwegian (Bokmål)	`no`	✔ *	✔		
Polish	`pl`	✔ *	✔		
Portuguese (Portugal)	`pt-PT`	✔	✔		`pt` also accepted
Portuguese (Brazil)	`pt-BR`		✔		
Russian	`ru`	✔ *	✔		
Spanish	`es`	✔	✔	✔ **	
Swedish	`sv`	✔ *	✔		
Turkish	`tr`	✔ *			

* indicates language support in preview

** Entity extraction for Spanish is only available in (version 2.1-preview)

See also

Cognitive Services Documentation page
Cognitive Services Product page