This tutorial is designed to let you quickly start exploring and developing applications with the Cloud Natural Language API. It is designed for people familiar with basic programming, though even without much programming knowledge, you should be able to follow along. Having walked through this tutorial, you should be able to use the Reference documentation (/natural-language/docs/reference/rest/) to create your own basic applications.

This tutorial steps through a Natural Language API application using Python code. The purpose here is not to explain the Python client libraries, but to explain how to make calls to the Natural Language API. Applications in Java and Node.js are essentially similar. Consult the Natural Language API Samples (/natural-language/docs/samples) for samples in other languages (including the sample in this tutorial).

This tutorial has several prerequisites:

- You've set up a Cloud Natural Language API project (/natural-language/docs/getting-started#set_up_a_project) in the Google Cloud Console.
- You've set up your environment using Application Default Credentials (/natural-language/docs/common/auth#adc) in the Google Cloud Console.
- You are familiar with Python (https://www.python.org/) in the Google Cloud Console programming.
- You have set up your Python development environment. It is recommended that you have the latest version of Python, `pip`, and `virtualenv` installed on your system. For instructions, see the Python Development Environment Setup Guide (https://cloud.google.com/python/setup) for Google Cloud Platform.
- You've installed the Google Cloud Client Library for Python (/natural-language/docs/reference/libraries)

This tutorial walks you through a basic Natural Language API application, using `classifyText` requests, which classifies content into categories along with a confidence score, such as:

To see the list of all available category labels, see Categories (/natural-language/docs/categories).

In this tutorial, you will create an application to perform the following tasks:

- Classify multiple text files and write the result to an index file.
- Process input query text to find similar text files.
- Process input query category labels to find similar text files.

The tutorial uses content from Wikipedia. You could create a similar application to process news articles, online comments, and so on.

You can find the tutorial source code in the Python Client Library Samples (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/language/classify_text) on GitHub.

This tutorial uses sample source text from Wikipedia. You can find the sample text files in the resources/texts (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/language/classify_text/resources/texts) folder of the GitHub project.

To use the Cloud Natural Language API, you must to import the `language` module from the `google-cloud-language` library. The `language.types` module contains classes that are required for creating requests. The `language.enums` module is used to specify the type of the input text. This tutorial classifies plain text content (`language.enums.Document.Type.PLAIN_TEXT`).

To calculate the similarity between text based on their resulting content classification, this tutorial uses `numpy` for vector calculations.

You can use the Python client library to make a request to the Natural Language API to classify content. The Python client library encapsulates the details for requests to and responses from the Natural Language API.

The `classify` function in the tutorial calls the Natural Language API `classifyText` method, by first creating an instance of the `LanguageServiceClient` class, and then calling the `classify_text` method of the `LanguageServiceClient` instance.

The tutorial `classify` function only classifies text content for this example. You can also classify the content of a web page by passing in the source HTML of the web page as the `text` and by setting the `type` parameter to `language.enums.Document.Type.HTML`.

For more information, see Classifying Content (/natural-language/docs/classifying-text). For details about the structure of requests to the Natural Language API, see the Natural Language API Reference (/natural-language/docs/reference/rest).

The returned result is a dictionary with the category labels as keys, and confidence scores as values, such as:

The tutorial Python script is organized so that it can be run from the command line for quick experiments. For example you can run:

he content to be classified must have at least 20 tokens (words) in order for the Natural Language API to return a
ise.

The `index` function in the tutorial script takes, as input, a directory containing multiple text files, and the path to a file where it stores the indexed output (the default file name is `index.json`). The `index` function reads the content of each text file in the input directory, and then passes the text files to the Cloud Natural Language API to be classified into content categories.

The results from the Cloud Natural Language API for each file are organized into a single dictionary, serialized as a JSON string, and then written to a file. For example:

To index text files from the command line with the default output filename `index.json`, run the following command:

Once the index file (default file name = `index.json`) has been created, we can make queries to the index to retrieve some of the filenames and their confidence scores.

One way to do this is to use a category label as the query, which the tutorial accomplishes with the `query_category` function. The implementation of the helper functions, such as `similarity`, can be found in the `classify_text_tutorial.py` file. In your applications the similarity scoring and ranking should be carefully designed around specific use cases.

For a list of all of the available categories, see Categories (/natural-language/docs/categories).

As before, you can call the `query_category` function from the command line:

You should see output similar to the following:

Alternatively, you can query with text that may not be part of the indexed text. The tutorial `query` function is similar to the `query_category` function, with the added step of making a `classifyText` request for the text input, and using the results to query the index file.

To do this from the command line, run:

This prints something similar to the following:

With the content classification API you can create other applications. For example:

- Classify every paragraph in an article to see the transition between topics.
- Classify timestamped content and analyze the trend of topics over time.
- Compare content categories with content sentiment using the `analyzeSentiment` method.
- Compare content categories with entities mentioned in the text.

Additionally, other GCP products can be used to streamline your workflow:

- In the sample application for this tutorial, we processed local text files, but you can modify the code to process text files stored in a Google Cloud Storage bucket by passing a Google Cloud Storage URI to the `classify_text` method.
- In the sample application for this tutorial, we stored the index file locally, and each query is processed by reading through the whole index file. This means high latency if you have a large amount of indexed data or if you need to process numerous queries. Datastore (/datastore/docs) is a natural and convenient choice for storing the index data.