

Building Cognitive Applications with IBM Watson Services: Volume 4 Natural Language Classifier

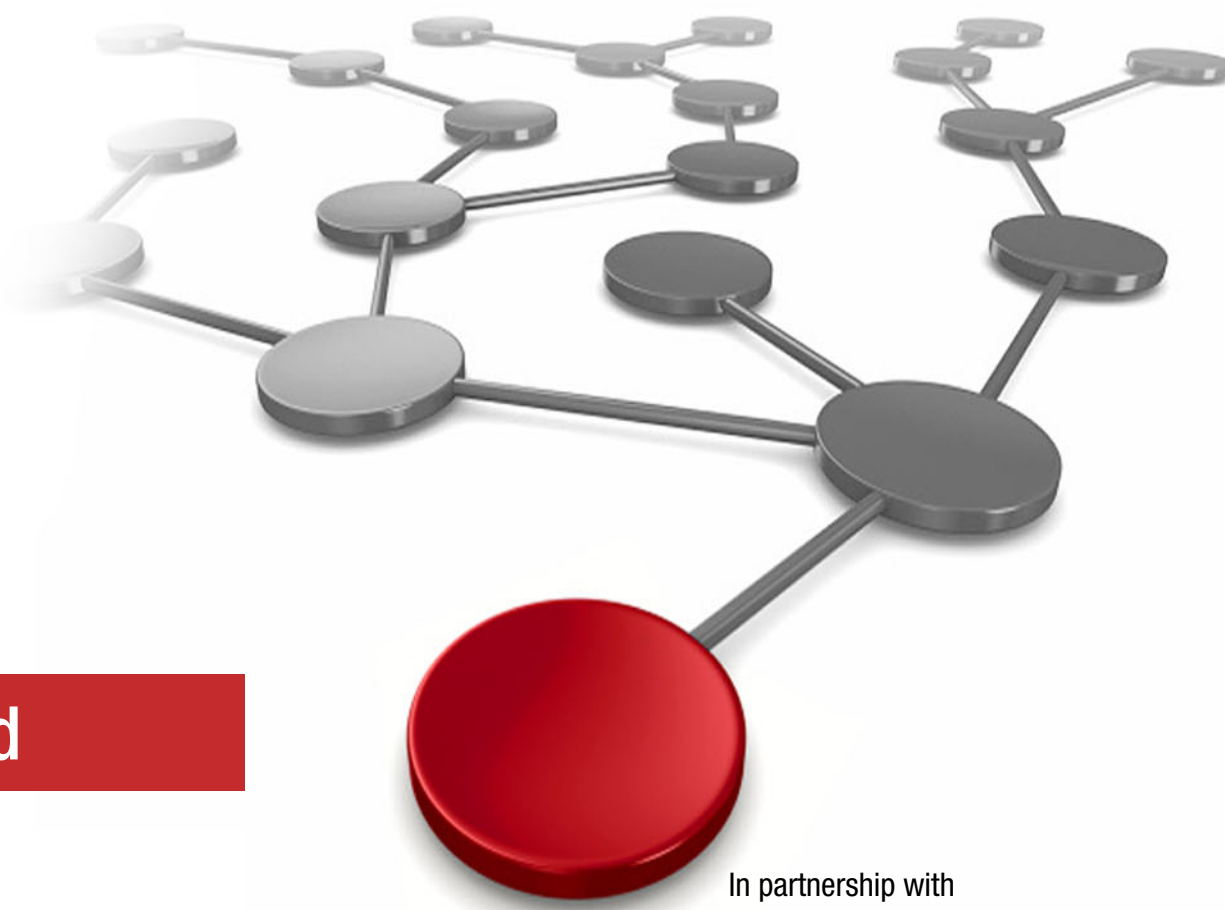
Marcelo Mota Manhaes

Taemin Ko

Abeer Selim

Omar Amer

Lak Sri



In partnership with
IBM Skills Academy Program



International Technical Support Organization

**Building Cognitive Applications with IBM Watson
Services: Volume 4 Natural Language Classifier**

May 2017

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (May 2017)

This edition applies to IBM Watson services in IBM Bluemix.

© Copyright International Business Machines Corporation 2017. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	ix
Comments welcome	ix
Stay connected to IBM Redbooks	ix
Chapter 1. Basics of Natural Language Classifier service	1
1.1 Using the Natural Language Classifier service	2
1.1.1 Prepare training data	2
1.1.2 Create and train the classifier	4
1.1.3 Query the trained classifier	7
1.1.4 Evaluate results and update the data	9
1.2 References	10
Chapter 2. Creating a Natural Language Classifier service in Bluemix	11
2.1 Requirements	12
2.2 Creating the Natural Language Classifier service instance	12
2.2.1 Creating the Natural Language Classifier service instance from the Bluemix website	12
2.2.2 Creating the Natural Language Classifier service instance using Cloud Foundry commands	14
2.3 What to do next	17
Chapter 3. Healthcare questions and answers	19
3.1 Getting started	20
3.1.1 Objectives	20
3.1.2 Prerequisites	20
3.1.3 Expected results	20
3.2 Architecture	22
3.3 Two ways to deploy the application: Step-by-step and quick deploy	23
3.4 Step-by-step implementation	23
3.4.1 Downloading the project from Git	23
3.4.2 Preparing training data	24
3.4.3 Creating and training the classifier	24
3.4.4 Creating the Node.js Express Healthcare Q and A application	28
3.4.5 Deploying the Healthcare Q and A application on Bluemix	35
3.4.6 Testing the application	36
3.5 Quick deployment of application	42
3.6 References	42
Chapter 4. News Classification	45
4.1 Getting started	46
4.1.1 Objectives	46
4.1.2 Prerequisites	46
4.1.3 Expected results	47
4.2 Architecture	49

4.3	Two ways to deploy the application: Step-by-step and quick deploy	50
4.4	Step-by-step implementation	50
4.4.1	Downloading the project from Git	51
4.4.2	Reviewing the project structure	56
4.4.3	Creating a Cloudant noSQL DB service instance	56
4.4.4	Preparing training data	59
4.4.5	Creating and training the classifier	63
4.4.6	Querying the trained classifier	68
4.4.7	Evaluating results and updating training data	73
4.4.8	Deploying the application	85
4.4.9	Testing the application	94
4.5	Quick deployment of application	97
4.6	References	99
Chapter 5. SPAM Classifier		101
5.1	Getting started	102
5.1.1	Objectives	102
5.1.2	Prerequisites	102
5.1.3	Expected results	102
5.2	Architecture	104
5.2.1	Component perspective	104
5.2.2	Role and activity perspective	105
5.3	Two ways to deploy the application: Step-by-step and quick deploy	106
5.4	Step-by-step implementation	106
5.4.1	Creating a Node-RED application	106
5.4.2	Cloning the Git project	109
5.4.3	Preparing training data	109
5.4.4	Creating and training the classifier	110
5.4.5	Querying the trained classifier	111
5.4.6	Evaluating results and updating training data	116
5.5	Quick deployment of application	120
5.6	References	122
Appendix A. Additional material		123
Locating the web material		123
Related publications		125
IBM Redbooks		125
Online resources		125
Help from IBM		126

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Bluemix®	IBM Watson®	Tivoli®
Cloudant®	IBM Watson IoT™	Watson™
developerWorks®	Rational®	Watson IoT™
Global Business Services®	Redbooks®	WebSphere®
Global Technology Services®	Redpapers™	
IBM®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The *Building Cognitive Applications with IBM Watson Services* series is a seven-volume collection that introduces IBM® Watson™ cognitive computing services. The series includes an overview of specific IBM Watson® services with their associated architectures and simple code examples. Each volume describes how you can use and implement these services in your applications through practical use cases.

The series includes the following volumes:

- ▶ *Volume 1 Getting Started*, SG24-8387
- ▶ *Volume 2 Conversation*, SG24-8394
- ▶ *Volume 3 Visual Recognition*, SG24-8393
- ▶ *Volume 4 Natural Language Classifier*, SG24-8391
- ▶ *Volume 5 Language Translator*, SG24-8392
- ▶ *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- ▶ *Volume 7 Natural Language Understanding*, SG24-8398

Whether you are a beginner or an experienced developer, this collection provides the information you need to start your research on Watson services. If your goal is to become more familiar with Watson in relation to your current environment, or if you are evaluating cognitive computing, this collection can serve as a powerful learning tool.

This IBM Redbooks® publication, Volume 4, introduces the Watson Natural Language Classifier service. This service applies cognitive computing techniques to return best matching predefined classes for short text inputs such as a sentence or phrase. The book describes concepts that you need to understand to create, use, and train the classifier. It describes how to prepare training data and create and train the classifier to connect the classes to example texts so that the service can apply the classes to new inputs. It also provides examples of applications that demonstrate how to use the Watson Natural Language Classifier service in practical use cases. You can develop and deploy the sample applications by following a step-by-step approach and by using the provided code snippets. Alternatively, you can download an existing Git project to more quickly deploy the application.

Authors

This book was produced by a team of specialists from around the world, working in collaboration with the IBM International Technical Support Organization.

Marcelo Mota Manhaes is a Certified IT Specialist in IBM Global Technology Services®, IBM Brazil. Marcelo is an IT Delivery Architect; his areas of expertise include cloud computing, software automation tools, business analytics, and cognitive computing. Marcelo has over 20 years of experience in the IT industry. He led several projects to design and build cognitive solutions such as an application that helps managers to evaluate the performance of their employees and a question answering system that uses Watson Natural Language Classifier (NLC), Retrieve and Rank, and Language Translator to enable IBM Knowledge Center users to find technical information by asking questions in their native language. Marcelo teaches cloud computing and cognitive systems at the Universidade Positivo. He is the author of several IBM Redbooks publications. Marcelo holds a B.S. in Computer Science from Universidade Federal do Paraná—UFPR and an M.S. in Computer Science from Universidade Tecnológica Federal do Paraná—UTFPR.

Taemin Ko is an IT Architect in Software Lab Services, IBM Korea. His primary responsibility is to help clients to accelerate software delivery. This includes Software Process Innovation, mentoring and coaching of software engineering practices, such as Agile Transformation, and IBM Rational® based Tool Chain Innovation. Taemin is also responsible for architecture definition and implementation of Internet of Things (IoT) solutions using IBM Watson®. He was in charge of Watson internalization and enablement for Lotte Group of Korea. He is currently working on the design and implementation of a chatbot enabled by Watson Conversation API for one of the major card companies in Korea.

Abeer Selim is a Certified IT Specialist Level 2 in IBM Global Business Services® and the Integration Practice Lead at the Client Innovation Center (CIC), IBM Egypt. She has over 11 years of experience in the IT industry. Abeer co-authored several scientific papers such as *Machine Learning Methodologies in Brain-Computer Interface Systems*, *Machine learning methodologies in P300 speller Brain-Computer Interface systems*, and *Electrode Reduction Using ICA and PCA in P300 Visual Speller Brain-Computer Interface System*. Abeer holds a B.S. and M.S. in Biomedical and Systems Engineering from Cairo University in Egypt.

Omar Amer is a Package Solution Consultant in cognitive computing at IBM Egypt. He is a subject matter expert (SME) for IBM Watson and IBM cloud technologies. Omar participated in several projects implementing cognitive computing solutions with Watson APIs, Watson Explorer, and Watson Knowledge Studio.

Lak Sri currently serves as a Program Director in the IBM developerWorks® part of the IBM Digital Business Group organization. Lak leads innovation in the developer activation space. He was the technical leader for the *Building Cognitive Applications with IBM Watson Services* Redbooks series. Lak led the development of the IBM Cloud Application Developer Certification program and the associated course. Earlier he worked as a Solution Architect for Enterprise Solutions in Fortune 500 companies using IBM Tivoli® products. He also built strategic partnerships in education and IBM Watson IoT™. Lak is an advocate and a mentor in several technology areas, and he volunteers to plan and support local community programs.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader, ITSO.

Thanks to the following people for their contributions to this project:

Swin Voon Cheok

Ecosystem Development (EcoD) Strategic Initiative, IBM Systems

Juan Pablo Napoli

Skills Academy Worldwide Leader, Global University Programs

Teja Tummalapalli

IBM Digital Business Group

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Basics of Natural Language Classifier service

This chapter introduces the *IBM Watson Natural Language Classifier* service. The Natural Language Classifier service applies cognitive computing techniques to return best matching predefined classes for short text inputs, such as a sentence or phrase.

Unlike traditional APIs, many cognitive services require being trained first before they can be used; the Watson Natural Language Classifier (NLC) service is one of those services that must be trained before using.

This chapter provides an overview of the process for creating and using the classifier. It includes snippets with code examples to perform some of the steps in the process.

The following topics are covered in this chapter:

- ▶ Using the Natural Language Classifier service
- ▶ References

1.1 Using the Natural Language Classifier service

Figure 1-1 provides an overview of the four steps that are included in the process of creating and using the classifier.

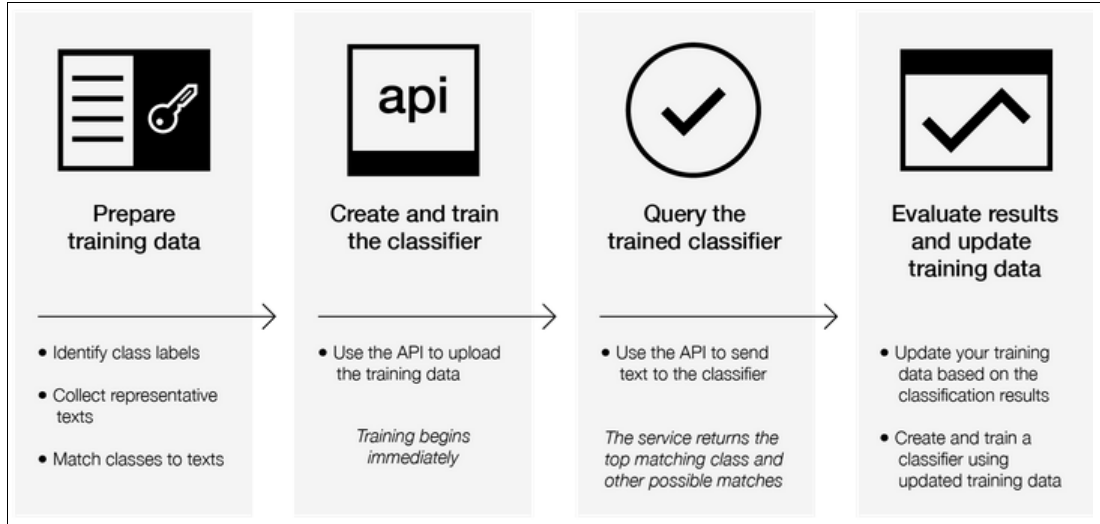


Figure 1-1 Using the Natural Language Classifier service: Process steps

To use the Natural Language Classifier service in your application, you must train the classifier following these steps:

1. Prepare training data
2. Create and train the classifier
3. Query the trained classifier
4. Evaluate results and update the data

The following sections take you through a simple example following these steps to train the classifier.

1.1.1 Prepare training data

To prepare the training data, follow these steps:

1. Identify class labels. These are the classes that the classifier will output.
2. Collect representative text.
3. Match classes to text. That is, create the training data by matching text with their respective classes.

Identify class labels

Class labels represent the result labels that describe the intent of the input text. Class labels are the output of a trained classifier.

To train the classifier, you prepare a training CSV file that is used when the classifier is created.

For the simple example described in this chapter, two class labels are identified: Health and VeterinaryHealth. In a real production scenario, usually a larger number of class labels are identified.

Collect representative texts

Gather representative texts for each class label for training purposes, These texts show the classifier examples for each class and serve as training data. These examples should be similar to the actual text input that will be provided to the classifier in production.

Representative text for Health class labels

The following text examples can be associated with the Health class labels:

- ▶ How much does it cost to get an occupational health card?
- ▶ What are steps required to get a health card?
- ▶ I want to be immune from Hepatitis B.

Representative text for VeterinaryHealth class labels

The following text examples can be associated with the VeterinaryHealth class labels:

- ▶ I need to know regulations for importing animals/veterinary products into the markets.
- ▶ Where can I adopt a pet from a shelter?
- ▶ Where can someone obtain health cards for veterinary?
- ▶ How to get a post mortem report for my pet?

Match classes to text

Now you create a file in CSV format with two columns:

- ▶ Column one is the *input* text
- ▶ Column two is the *class label* for that text

Table 1-1 shows the input text and corresponding class label for the example in this chapter.

Table 1-1 Training data to create a CSV file

Input text	Class label
How much does it cost to get an occupational health card	Health
What are steps required to get a health card	Health
I want to be immune from Hepatitis B	Health
I need to know regulations for importing animals/veterinary products into the Markets	VeterinaryHealth
Where can I adopt a pet from a shelter	VeterinaryHealth
Where can someone obtain health cards for veterinary	VeterinaryHealth
How to get a post mortem report for my pet	VeterinaryHealth

Example 1-1 shows the CSV file created from Table 1-1.

Example 1-1 Training data in CSV format

```
How much does it cost to get an Occupational health card,Health
What are steps required to get a health Card,Health
I want to be immune from Hepatitis B,Health
I need to know regulations for importing animals/veterinary products into the
Markets,VeterinaryHealth
Where Can I adopt a pet from a shelter,VeterinaryHealth
Where can someone obtain Health cards for veterinary,VeterinaryHealth
How to get a post mortem report for my pet,VeterinaryHealth
```

You can access the training CSV file at the GitHub web page:

<https://gist.github.com/snippet-java/044c616801cea023930bed41efed6488>

Note: This simple example shows only two class labels and three and four text samples for each. In a production scenario, many more class labels and text samples of training data should be provided.

1.1.2 Create and train the classifier

Before you can create a classifier, the Natural Language Classifier service instance must be created as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11.

After creating the Natural Language Classifier service instance, create a classifier that is associated with the service instance. Specify the classifier name and training CSV file, and then upload the training CSV file that you created in 1.1.1, “Prepare training data” on page 2 to train the classifier. The classifier ID will be returned.

Figure 1-2 shows a simplified diagram representing the creation of the classifier.

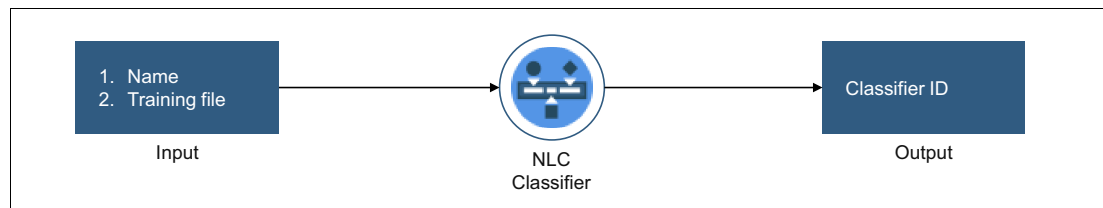


Figure 1-2 Create the Natural Language Classifier service classifier

You can create the classifier and upload training data using one of the following methods:

- ▶ Using the toolkit in IBM Bluemix®
- ▶ Programmatically, with simple programs written in languages such as Java and Node.js
- ▶ Using command-line tools, such as cURL

The following examples show code snippets in different technologies to create the classifier and upload the training data passing the following parameters:

- ▶ Credentials of the associated service instance
- ▶ Classifier name
- ▶ The CSV file with the training data to upload

Example 1-2 shows a code snippet in Node.js to create the classifier and upload the training data.

Example 1-2 Code snippet: NodeJS

```
var watson = require('watson-developer-cloud');
var fs = require('fs');
var natural_language_classifier = watson.natural_language_classifier({
  username: '{username}',
  password: '{password}',
  version: 'v1'
});
var params = {
  language: 'en',
  name: 'My Classifier',
  training_data: fs.createReadStream('./train.csv')
};
natural_language_classifier.create(params, function(err, response) {
  if (err) console.log(err);
  else
    console.log(JSON.stringify(response, null, 2));
});
```

Example 1-3 shows a code snippet in Java to create the classifier and upload the training data.

Example 1-3 Code snippet: Java

```
import java.io.File;

import
com.ibm.watson.developer_cloud.natural_language_classifier.v1.NaturalLanguageClassifier;
import com.ibm.watson.developer_cloud.natural_language_classifier.v1.model.*;

public class SimpleServlet {
    public static void main(String[] arg) {
        NaturalLanguageClassifier service = new NaturalLanguageClassifier();
        service.setUsernameAndPassword("{username}", "{password}");
        Classifier classifier = service.createClassifer("My Classifier", "en",
new File("./train.csv")).execute();
        System.out.println(classifier);
    }
}
```

Example 1-4 shows a code snippet in cURL to upload the training data.

Example 1-4 Code snippet: cURL

```
curl -u "{username}":"{password}" -F training_data=@train.csv -F
training_metadata="{\"language\": \"en\", \"name\": \"HealthClassifier\"}"
https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers
```

Response

Example 1-5 shows the response returned when running the code to upload the training data.

Example 1-5 Code snippet: Response

```
{
  "classifier_id": "10D41B-nlc-1",
  "name": "My Classifier",
  "language": "en"
  "created": "2015-05-28T18:01:57.393Z",
  "url":
  "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers
  /10D41B-nlc-1",
  "status": "Training",
  "status_description": "The classifier instance is in its training phase, not yet
  ready to accept classify requests"
}
```

The `classifier_id` value shows a unique identifier for each classifier. Multiple classifiers can be associated with the same Natural Language Classifier service instance.

The status shows the classifier status. When the classifier is ready to accept requests, the status changes from Training to Available.

Check the classifier status

Before you can use the classifier, you must check the status. The following code snippets provide examples of how to check the status.

Note: In the following code snippets, replace "{classifier}" with the "classifier_id" value obtained in the response (see Example 1-5).

Example 1-6 shows a code snippet in Node.js to check status of the classifier.

Example 1-6 Code snippet: NodeJS

```
var watson = require('watson-developer-cloud');
var fs = require('fs');
var natural_language_classifier = watson.natural_language_classifier({ username:
'"{username}"', password: ' "{password}"', version: 'v1' });

natural_language_classifier.status({
  classifier_id: ' "{classifier}"'
}, function(err, response) {
  if (err) console.log('error: ', err);
  else console.log(JSON.stringify(response, null, 2));
});
```

Example 1-7 shows a code snippet in Java to check status of the classifier.

Example 1-7 Code snippet: Java

```
import java.io.File;

import
com.ibm.watson.developer_cloud.natural_language_classifier.v1.NaturalLanguageClass
ifier;
import com.ibm.watson.developer_cloud.natural_language_classifier.v1.model.*;

public class SimpleServlet {
    public static void main(String[] arg) {
        NaturalLanguageClassifier service = new NaturalLanguageClassifier();
        service.setUsernameAndPassword("{username}", "{password}");
        Classifier classifier = service.getClassifier("{classifier}").execute();
        System.out.println(classifier);
    }
}
```

Example 1-8 shows a code snippet in cURL to check status of the classifier.

Example 1-8 Code snippet: cURL

```
curl -u "{username}":"{password}"
https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/
{classifier}
```

Response

When the classifier is trained, the status changes to Available (see Example 1-9). You can now use the classifier.

Example 1-9 Status response for a trained classifier

```
{ "classifier_id": "{classifier}",
  "name": "My Classifier",
  "language": "en",
  "created": "2015-05-28T18:01:57.393Z",
  "url":
  "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers
/10D41B-nlc-1",
  "status": "Available",
  "status_description": "The classifier instance is now available and is ready to
take classifier requests.",
}
```

1.1.3 Query the trained classifier

After the classifier is trained, you can query it. Figure 1-3 on page 8 represents querying the classifier by providing the classifier ID and input text.

The API returns a response that includes the name of the class for which the classifier has the highest confidence. Other class-confidence pairs are listed in descending order of

confidence. The confidence value represents a percentage, and higher values represent higher confidences.

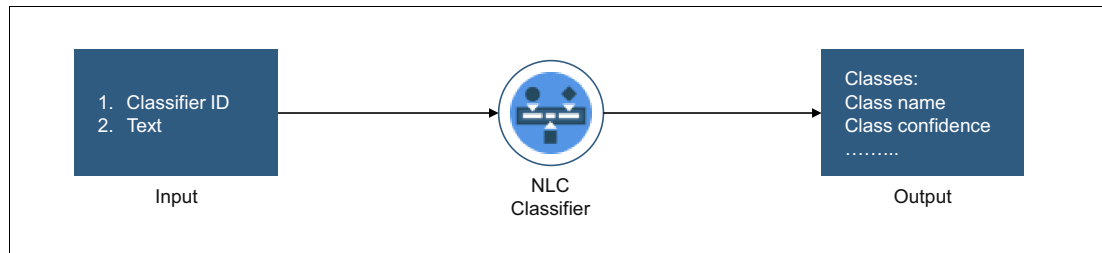


Figure 1-3 Querying the classifier

The classification process divides the value of 1 (100%) among all defined class labels and outputs a value for each class label (percentage) that can be thought of as the confidence level for each class label as shown Figure 1-3.

The following examples show code snippets to query the classifier.

Example 1-10 shows a snippet in Node.js to run a query on a classifier by specifying the classifier ID.

Example 1-10 Code snippet: Node.js, querying the classifier

```
var watson = require('watson-developer-cloud');
var fs = require('fs');
var natural_language_classifier = watson.natural_language_classifier({ username:
'username', password: 'password', version: 'v1' });

natural_language_classifier.classify({
  text: 'I want a health card',
  classifier_id: 'classifier'
}, function(err, response) {
  if (err) console.log('error: ', err);
  else console.log(JSON.stringify(response, null, 2));
});
```

Example 1-11 shows a snippet in Java to run a query on the Natural Language Classifier classifier.

Example 1-11 Code snippet: Java, querying the classifier

```
import java.io.File;

import
com.ibm.watson.developer_cloud.natural_language_classifier.v1.NaturalLanguageClassifier;
import com.ibm.watson.developer_cloud.natural_language_classifier.v1.model.*;

public class SimpleServlet {
  public static void main(String[] arg) {
    NaturalLanguageClassifier service = new NaturalLanguageClassifier();
    service.setUsernameAndPassword("username", "password");
    Classification classifier = service.classify("classifier"
, "I want a health card").execute();
    System.out.println(classifier);
  }
}
```

```
}  
}
```

Example 1-12 shows a snippet in cURL to run a query on the Natural Language Classifier classifier.

Example 1-12 Code snippet: Querying the classifier, cURL

```
curl -u "{username}":"{password}" -H "Content-Type:application/json" -d  
"{\"text\":\"I want a health card\"}"  
https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/  
{classifier}/classify
```

Query response

Example 1-13 shows the response returned when querying the classifier.

Example 1-13 Query response

```
{  
  "classes": [  
    {  
      "confidence": 0.9858005113688728,  
      "class_name": "Health"  
    },  
    {  
      "confidence": 0.014199488631127315,  
      "class_name": "VeterinaryHealth"  
    }  
  ],  
  "classifier_id": "f5b42ex171-nlc-2121",  
  "text": "I want a health card",  
  "top_class": "Health",  
  "url":  
  "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/  
  /f5b42ex171-nlc-2121"  
}
```

In this code snippet:

- ▶ The "text" value shows the input text in the query request.
- ▶ The "classes" value is an array that contains the list of defined class labels and the confidence for each. *Confidence* is a value between 0 (0%) and 1 (100%), indicating the confidence for each class label for the query input text.

The sum of confidence for all classes is 1. The classes in the array are ordered in a descending order of confidence. That is, the class label with the highest confidence is always the first element in the `classes` array.

1.1.4 Evaluate results and update the data

The objective of this step in the process is to improve the results returned by the classifier:

1. Detect wrong or weak confidence cases for user input text.
2. Change or restructure user's phrases into generic representative text.

3. Match text to their corresponding class label.
4. Add new text to the original training data and create a new classifier.
5. Repeat this cycle when quality of classification drops to a certain lower limit.

1.2 References

See the following resources:

- ▶ Overview of the IBM Watson Natural Language Classifier service:

<https://www.ibm.com/watson/developercloud/doc/natural-language-classifier/index.html>

- ▶ Getting started with the Natural Language Classifier service:

<https://www.ibm.com/watson/developercloud/doc/natural-language-classifier/getting-started.html>



Creating a Natural Language Classifier service in Bluemix

IBM Watson Developer Cloud offers a variety of services for developing cognitive applications. One of these services, which is the focus of this book, is the Watson Natural Language Classifier (NLC) service.

This chapter explains how to create an instance of the Natural Language Classifier service in Bluemix that is required for the use cases described in this book.

The following topics are covered in this chapter:

- ▶ Requirements
- ▶ Creating the Natural Language Classifier service instance
- ▶ What to do next

2.1 Requirements

To create a service and perform the use cases in this book, you must have a Bluemix account. You can register to create an account and log in at [IBM Bluemix](#). When you log in, you are prompted to authenticate with your email or IBM ID and password.

2.2 Creating the Natural Language Classifier service instance

The two ways to create the Natural Language Classifier service instance are as follows:

- ▶ Creating the Natural Language Classifier service instance from the Bluemix website
- ▶ Creating the Natural Language Classifier service instance using Cloud Foundry commands

2.2.1 Creating the Natural Language Classifier service instance from the Bluemix website

To create the service, follow these steps:

1. Log in to the [IBM Bluemix](#) website.
2. When the home page opens, click **Catalog**.
3. On the IBM Bluemix Catalog page (Figure 2-1) scroll to the Services section, select **Watson**, and then click **Natural Language Classifier**.

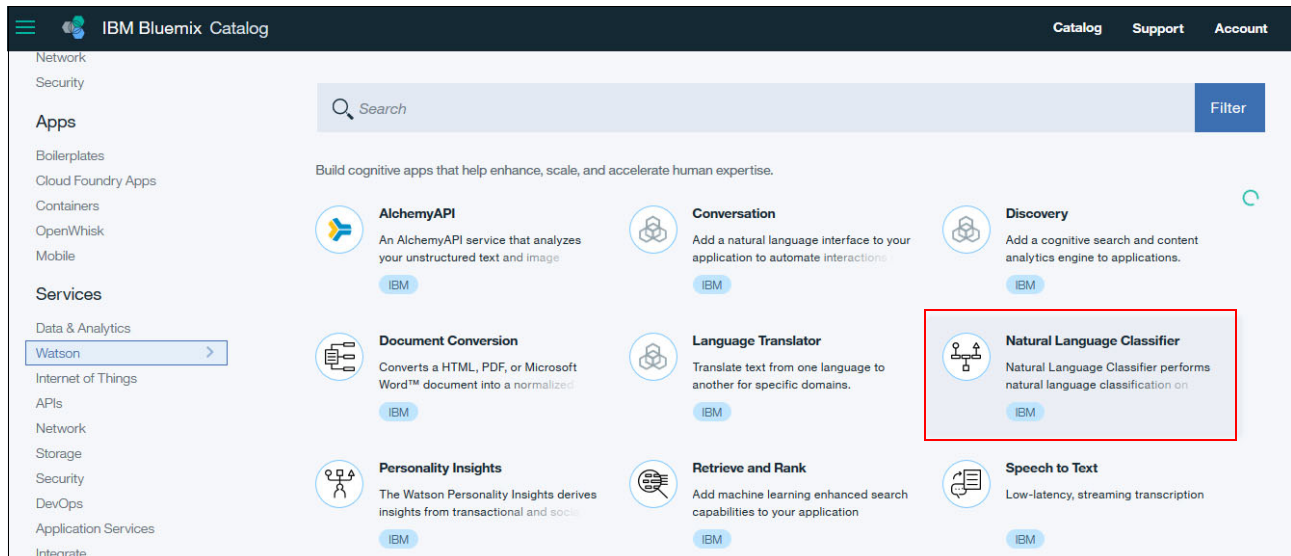


Figure 2-1 Natural Language Classifier in the Bluemix Catalog

4. On the Natural Language Classifier page (Figure 2-2 on page 13), create the service. You can change the Service name and Credentials name fields by using your personal choices or keep the default values. The important point is that for the instance of the service being created, the credential name will have the username and password. Click **Create**.

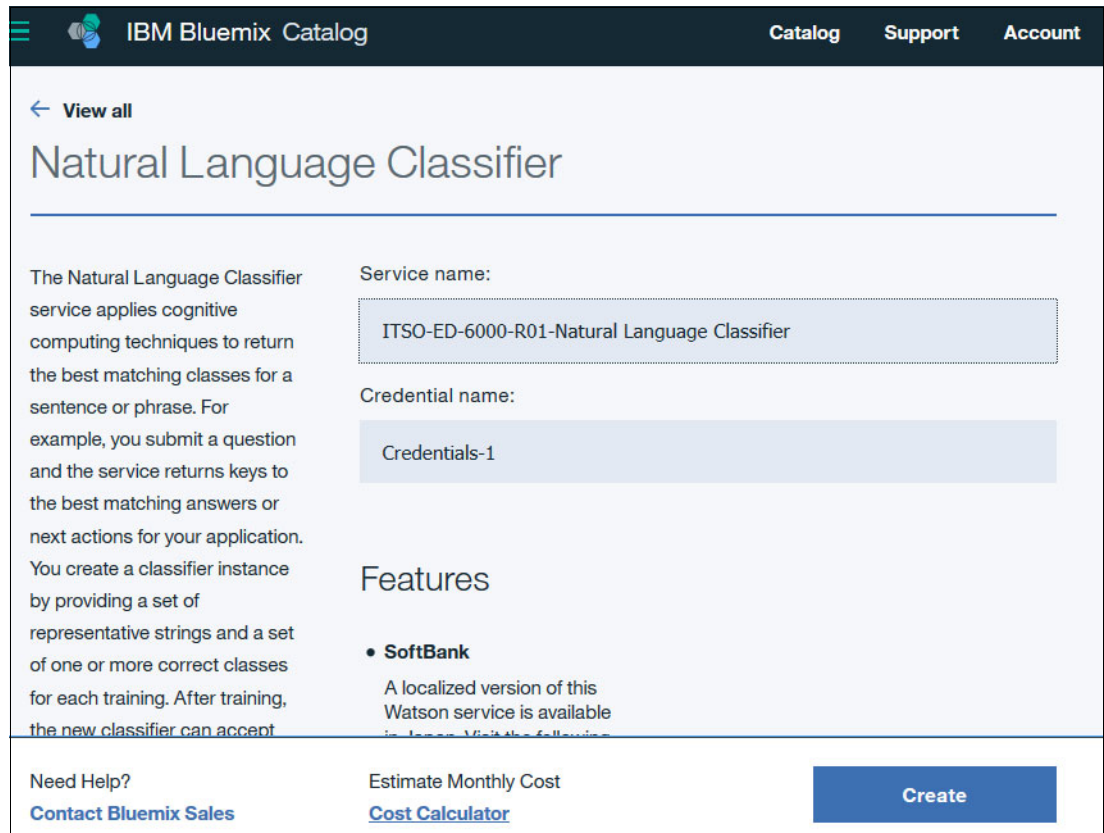


Figure 2-2 Creating Natural Language Classifier service instance

5. Get the credentials (username and password) from the service instance for later use. Click the service instance and select the **Service Credentials** tab.
6. Click **View Credentials** and get the username and password values (Figure 2-3 on page 14).

Important: The service instance credentials, username and password, are used in the next chapters.

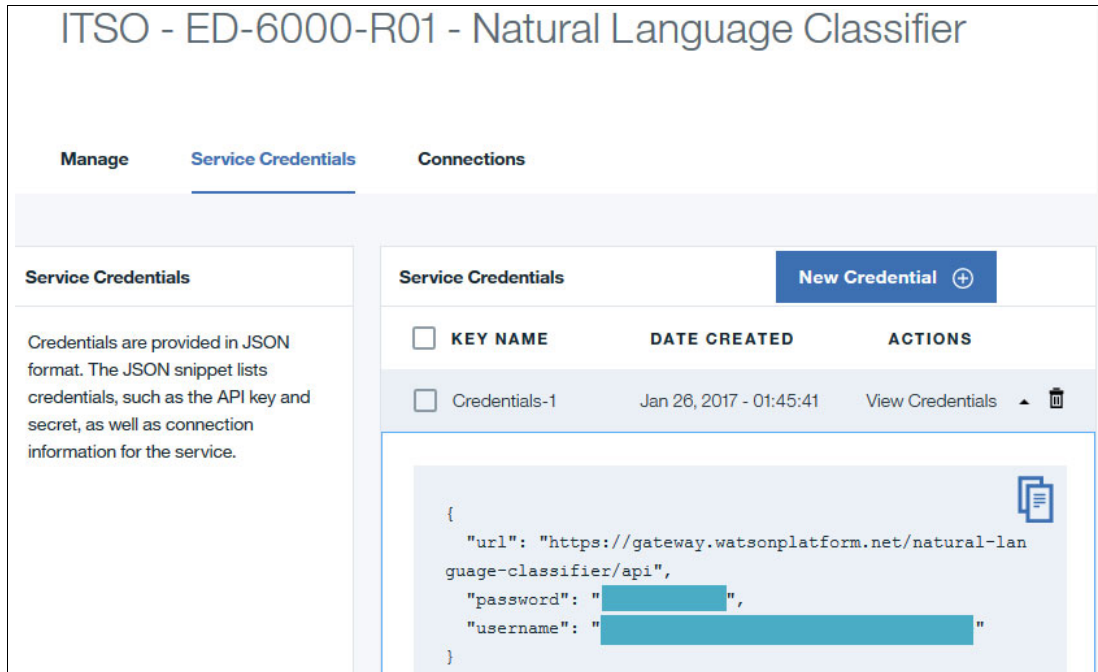


Figure 2-3 Get user name and password from Natural Language Classifier service instance

2.2.2 Creating the Natural Language Classifier service instance using Cloud Foundry commands

To create the service, follow these steps:

1. Download the Cloud Foundry software [Cloud Foundry software](#) and install it on your computer.
2. Open a command prompt.
3. Run the **cf login** command and insert the email and password for your Bluemix account in the sequence shown in Example 2-1.

Example 2-1 Run login and provide email and password for the Bluemix account

```
cf login
  API endpoint: https://api.ng.bluemix.net
  ¢
  Email> <PUT_YOUR_BUEMIX_EMAIL_ACCOUNT>
  ¢
  Password> <PUT_YOUR_PASSWORD_ACCOUNT>

  Authenticating...
  OK
  Targeted org <YOUR_ORGANIZATION>
```

4. Select a Bluemix space to host the service (Example 2-2 on page 14).

Example 2-2 Select a Bluemix space

```
Select a space (or press enter to skip):
1. dev
2. qa
3. Prod
```

```
Space> 1
Targeted space dev
API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: <YOUR_BLUEMIX_EMAIL_ACCOUNT>
Org: <YOUR_ORGANIZATION>
Space: dev
```

5. Run the following command to create an instance of the service:

```
cf create-service <service> <service_plan> <service_instance>
```

About the command:

cf create-service The Cloud Foundry command to create the service instance.

<service> The name of the service you want to create an instance of. For Natural Language Classifier, use `Natural_Language_Classifier`.

<service_plan> The pricing plan.

<service_instance> The service instance name you want to use.

Example 2-3 shows the command.

Example 2-3 The cf create-service command

```
cf create-service Natural_Language_Classifier standard "ITS0-
ED-6000-R01 - Natural Language Classifier"
Creating service instance ITS0-ED-6000-R01 - Natural Language Classifier in org
<YOUR_ORGANIZATION>/ space dev as <YOUR_BLUEMIX_EMAIL_ACCOUNT>...
OK
```

6. List the service information by using the **cf service <service_name>** command to confirm that it was successfully created (Example 2-4).

Example 2-4 The cf service command

```
cf service "ITS0-ED-6000-R01 - Natural Language Classifier"

Service instance: ITS0-ED-6000-R01 - Natural Language Classifier
Service: natural_language_classifier
Bound apps:
Tags:
Plan: standard
Description: Natural Language Classifier performs natural language classification
on question texts. A user would be able to train their data and the predict the
appropriate class for an input question.
Documentation url: https://www.ibm.com/watson/developercloud/nl-classifier.html
Dashboard:
https://www.ibm.com/watson/developercloud/dashboard/en/nl-classifier-dashboard.html

Last Operation
Status: create succeeded
Message:
Started: 2017-02-16T17:16:49Z
Updated:
```

7. Create user and password credentials to access the service by using this command:

```
cf create-service-key <service_instance> <service_key>.
```

About the command:

cf create-service-key The Cloud Foundry command to create the service key with user and password.

<service_instance> The name of the Natural Language Classifier service instance.

<service_key> The name of the service key you want to create.

Example 2-5 shows this command.

Example 2-5 The cf create-service-key command

```
cf create-service-key "ITSO-ED-6000-R01 - Natural Language Classifier" myKeys
Creating service key myKeys for service instance ITSO-ED-6000-R01 - Natural
Language Classifier as <YOUR_BLUEMIX_EMAIL_ACCOUNT>...
OK
```

8. Get the username and password in order to access the service later by running the following command:

```
cf service-key <service_instance> <service_key>
```

About the command:

cf service-key The Cloud Foundry command to view the username and password in the service key.

<service_instance> The name of the service instance.

<service_key> The name of the service key.

Example 2-6 shows this command.

Example 2-6 Use cf service-key to get username and password

```
cf service-key "ITSO-ED-6000-R01 - Natural Language Classifier" myKeys
Getting key myKeys for service instance ITSO-ED-6000-R01 - Natural Language
Classifier as <YOUR_BLUEMIX_EMAIL_ACCOUNT>...
```

```
{
  "password": "egDxZXXEoXJR",
  "url": "https://gateway.watsonplatform.net/natural-language-classifier/api",
  "username": "189db2d8-95e2-XXXX-9a3c-1fba7f991c41"
}
```

Important: The service instance credentials, username and password, are used in the next chapters.

2.3 What to do next

Creating the service instance is a required step for the remaining chapters in this book. With username and password collected from Natural Language Classifier service instance, you can go through the next chapters:

- ▶ Chapter 3, “Healthcare questions and answers” on page 19.
- ▶ Chapter 4, “News Classification” on page 45.
- ▶ Chapter 5, “SPAM Classifier” on page 101.



Healthcare questions and answers

This chapter introduces the use of the Watson Natural Language Classifier (NLC) service in an application. The Watson Natural Language Classifier service applies deep learning techniques to predict the best predefined classes for short sentences or phrases. The classes prediction can be used for triggering a corresponding action in an application, such as answering a question.

This chapter describes steps to create a simple Healthcare question and answer (Q and A) application. The application is an example of a use case for a cognitive application using the Watson Natural Language Classifier service. The main objective of the Healthcare Q and A application is to provide answers to questions that are related to a healthcare community or organization.

In this chapter, you work with code that is partially developed and therefore the chapter provides code snippets for you to use.

The following topics are covered in this chapter:

- ▶ Getting started
- ▶ Architecture
- ▶ Two ways to deploy the application: Step-by-step and quick deploy
- ▶ Step-by-step implementation
- ▶ Quick deployment of application
- ▶ References

3.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

3.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- ▶ Create a Healthcare Q and A application using Node.js and running in IBM Bluemix.
- ▶ Prepare training data in a CSV file.
- ▶ Implement the Watson Natural Language Classifier service using Node.js.
- ▶ Train the classifier using the prepared CSV file.
- ▶ Use the Bluemix web user interface to create and manage services.

3.1.2 Prerequisites

To complete the steps in this chapter, ensure that you meet the following prerequisites

- ▶ Bluemix account
- ▶ The instructions to create an Natural Language Classifier service, as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11
- ▶ An Internet browser such as Chrome, Firefox, Internet Explorer, or Safari
- ▶ Install Cloud Foundry
- ▶ Basic JavaScript skills
- ▶ Review the available Bluemix regions and select the most appropriate based on your location
- ▶ Git basics

3.1.3 Expected results

By following the steps in this book, you should be able to use a browser to run the application. The application works as follows:

1. In a web browser open the [Healthcare Q and A Application](#) to see a running version on Bluemix (Figure 3-1 on page 21).

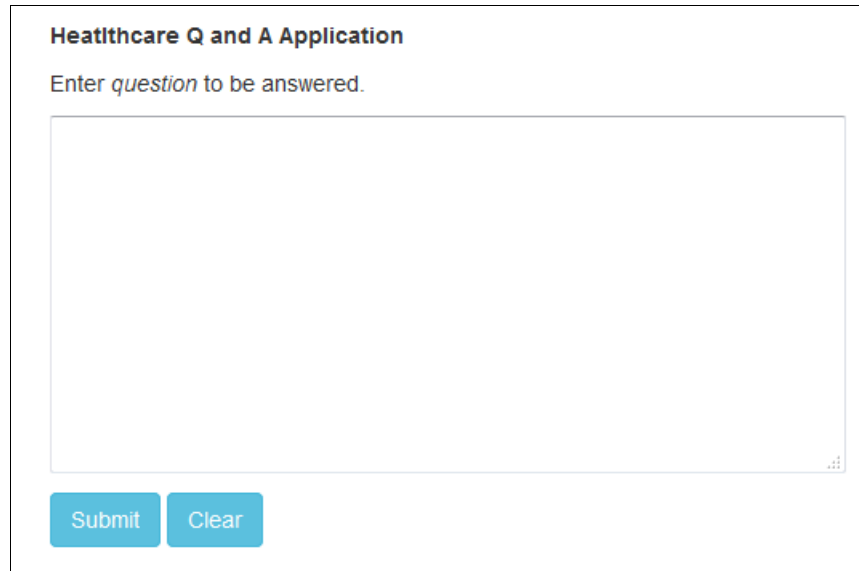


Figure 3-1 Healthcare Q and A application

2. Enter a question (such as “Where are clinics?”) and click **Submit**. A page opens to display an answer (Figure 3-2).

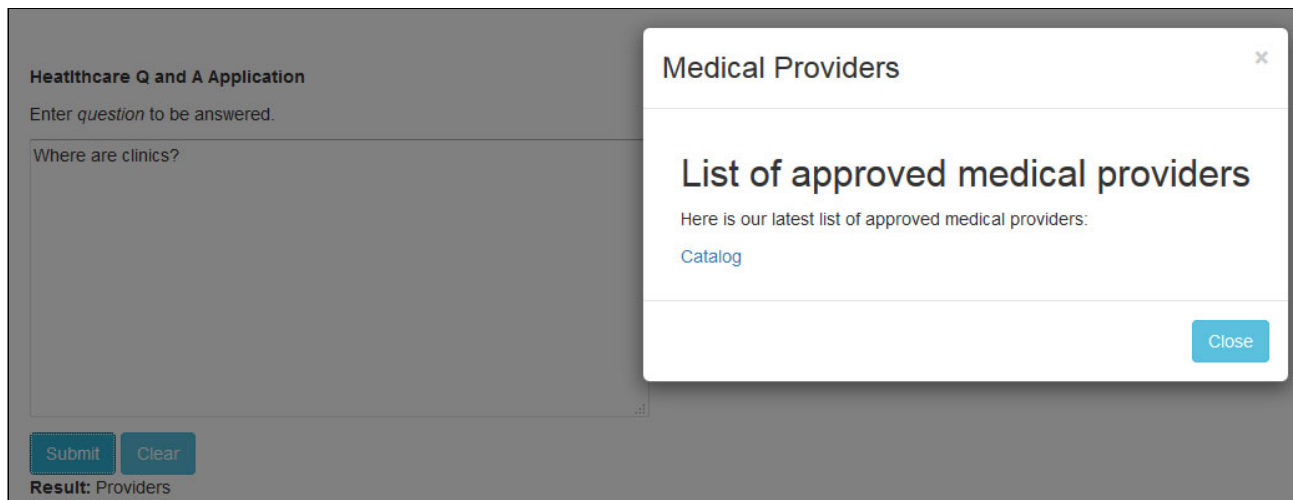


Figure 3-2 Healthcare Q and A application results displayed

3.2 Architecture

The Healthcare Q and A application is composed of a web interface, application logic, Watson Natural Language Classifier service, and Node.js run time. The application logic orchestrates a classification service. The Watson Natural Language Classifier service classifies to which category the specified question belongs. Node.js run time uses the Express framework as the integration platform between the web interface and the Watson Natural Language Classifier service.

Figure 3-3 shows the components of the application and flow:

1. User submits a healthcare related question through the web interface.
2. Web interface posts the received question to the application.
3. Application logic routes the question to Node.js Express as the application integration platform.
4. Node.js in turn receives the question and sends it to the Watson Natural Language Classifier service to be classified.
5. Watson Natural Language Classifier service returns the response, which includes the top class representing the category of the question.
6. Node.js returns the Natural Language Classifier response to the application logic.
7. Application logic identifies the web interface page to be displayed as per the question category returned from the Watson Natural Language Classifier service.
8. Web interface displays the page that includes the answer to the question.

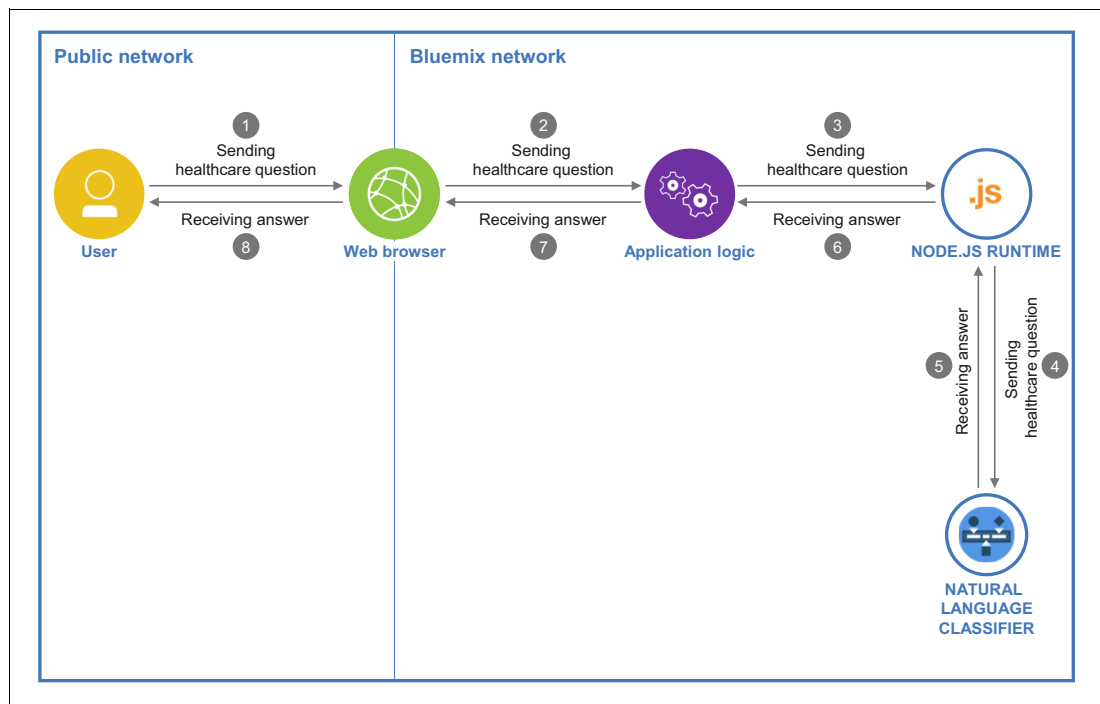


Figure 3-3 Architecture overview diagram

3.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- ▶ Step-by-step deployment (*incomplete*) version of the application

This repository contains an incomplete version of the application and is used in all sections of 3.4, “Step-by-step implementation” on page 23. This version takes you through the key steps to integrate the IBM Watson services with the application logic.

- ▶ Quick deployment (*complete*) version of the application

This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 3.5, “Quick deployment of application” on page 42.

3.4 Step-by-step implementation

Deploying this application involves the following steps:

1. Downloading the project from the Git repository.
2. Preparing the training data.
3. Creating and training the Natural Language Classifier service.
4. Creating the Node.js Express Healthcare Q and A application.
5. Deploying the application.
6. Testing the application.

3.4.1 Downloading the project from Git

Start by downloading the code. The code is basically the same as the quick-deployment version, however it is missing some important parts to be developed by you.

- ▶ Download the *incomplete* code (step-by-step deployment version):

<https://github.com/snippet-java/redbooks-nlc-201-healthcare-nodejs-student.git>

After downloading, explore the downloaded folder to become familiar with its structure so you can more easily follow the step-by-step deployment.

- ▶ You can also download the *complete* code (quick deployment version) that you can use for verification or as a code reference:

<https://github.com/snippet-java/redbooks-nlc-201-healthcare-nodejs.git>

3.4.2 Preparing training data

This section focuses on preparing the training data CSV file for the Healthcare Q and A application. The file should include a list of questions that are categorized into different classes. After randomly listing some of the expected questions, most of the questions seemed to fit into five main categories of questions representing five classes (Figure 3-4):

- Policy:** Questions about healthcare policy, contracts, and plans
- Providers:** Questions about service providers
- Products:** Questions about products and offers
- About:** Questions about the healthcare organization and contact information
- Claim:** Questions about claims and reimbursements

Can you nominate good Pediatrics?	Providers
What are the HealthCare insurance products proposed?	Products
I would like to contact the support team?	About
Is there any time limitation to submitting my claim?	Claim
How do I manage my policy?	Policy

Figure 3-4 Sample questions for different categories

A bulk of related questions for each category are in the training file.

This training file is used as an input to the next step, 3.4.3, “Creating and training the classifier” on page 24.

You can find the training data CSV file `hcqa_training_data.csv` in the `hcqaNaturalLanguageClassifier_Student` folder that you downloaded as described in 3.4.1, “Downloading the project from Git” on page 23.

3.4.3 Creating and training the classifier

Note: You must create a Natural Language Classifier service instance in Bluemix as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11 before performing the steps in this section.

After creating the Natural Language Classifier service instance, the next step is to create and train a classifier associated with the service instance, by using the prepared training CSV file from 3.4.2, “Preparing training data” on page 24:

1. Log in to your Bluemix account, open the **Manage** tab, and click **Access the beta toolkit**, (Figure 3-5 on page 25).

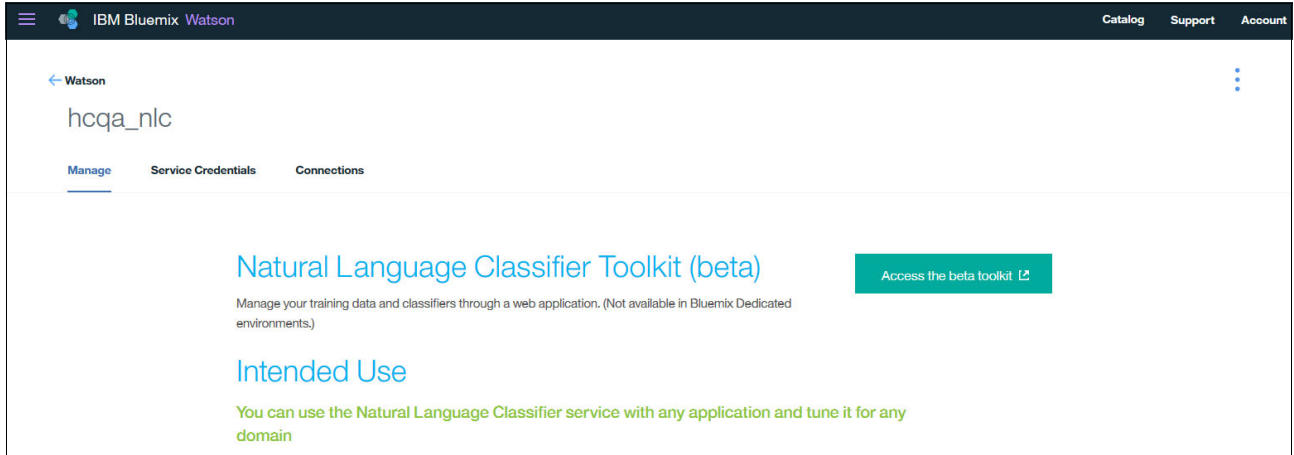


Figure 3-5 Training process step 1: Access the toolkit

2. Click **Sign in with Bluemix** (Figure 3-6).



Figure 3-6 Training process step 2: Sign in with Bluemix

3. Click **Confirm** to grant the toolkit access to your previously created Natural Language Classifier service instance (Figure 3-7).

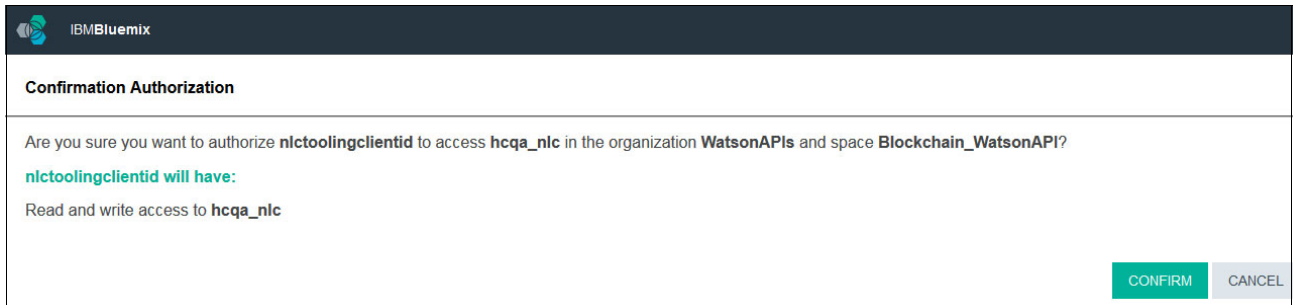


Figure 3-7 Training process step 3: Confirm

4. Click **Add training data** (Figure 3-8).

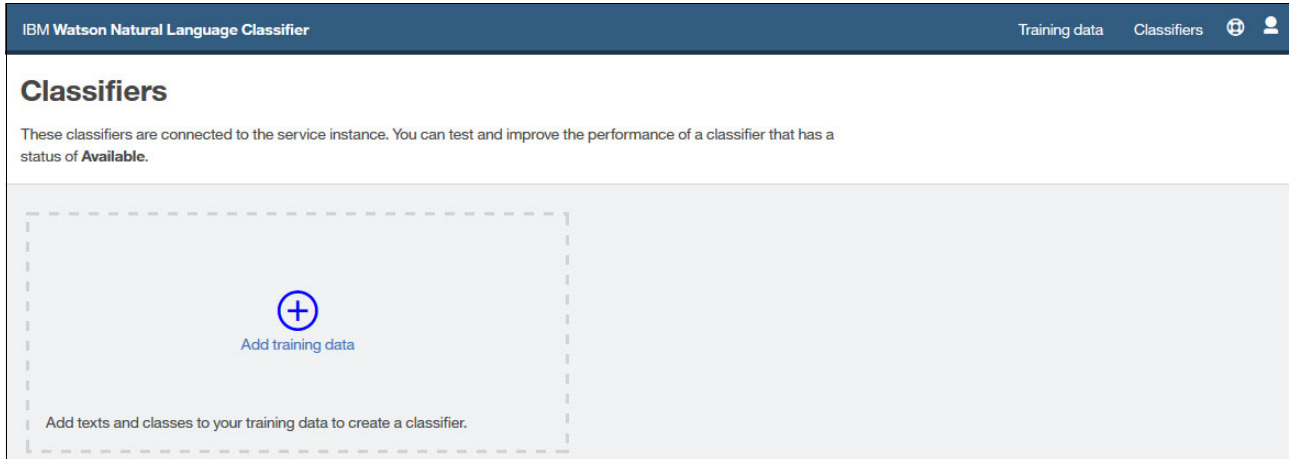


Figure 3-8 Training process step 4: Add training data

5. Upload the prepared training CSV file by clicking the upload icon (Figure 3-9). Then click **Create classifier**.

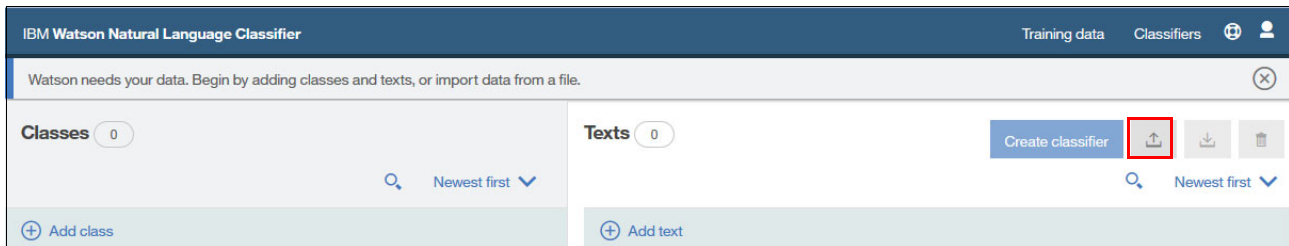


Figure 3-9 Training process step 5: Create classifier and upload training data

6. Specify the classifier name in the Name field, and then click **Create** (Figure 3-10).

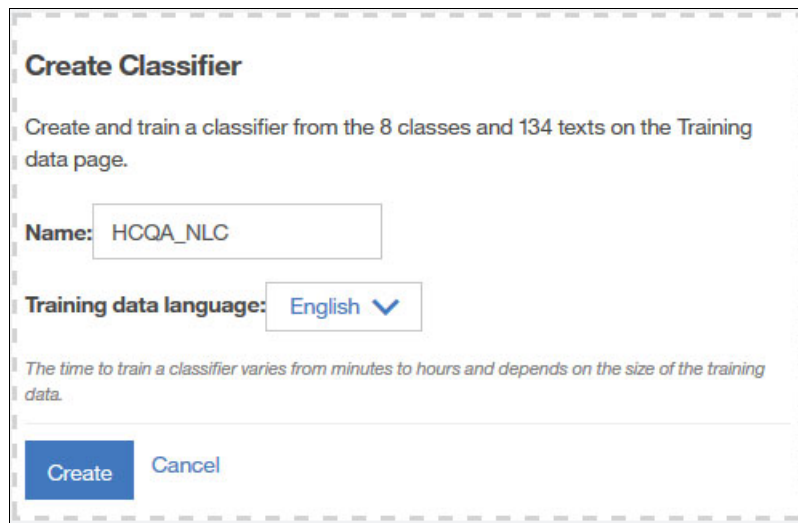


Figure 3-10 Training process step 6: Provide a name for classifier

7. Reload the page to be sure that the classifier creation process is in progress (Figure 3-11).

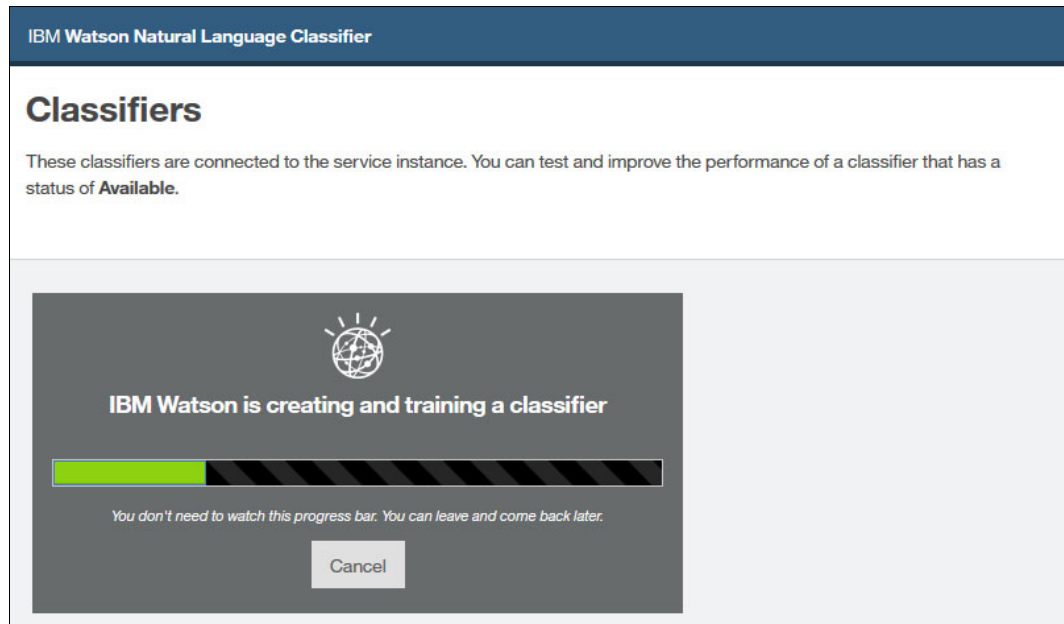


Figure 3-11 Training process step 7: Progress bar

8. After the training process completes, the classifier is listed with a Classifier ID (Figure 3-12). This value will be used later in the JSON configuration file, for the Node.js Healthcare Q and A application.

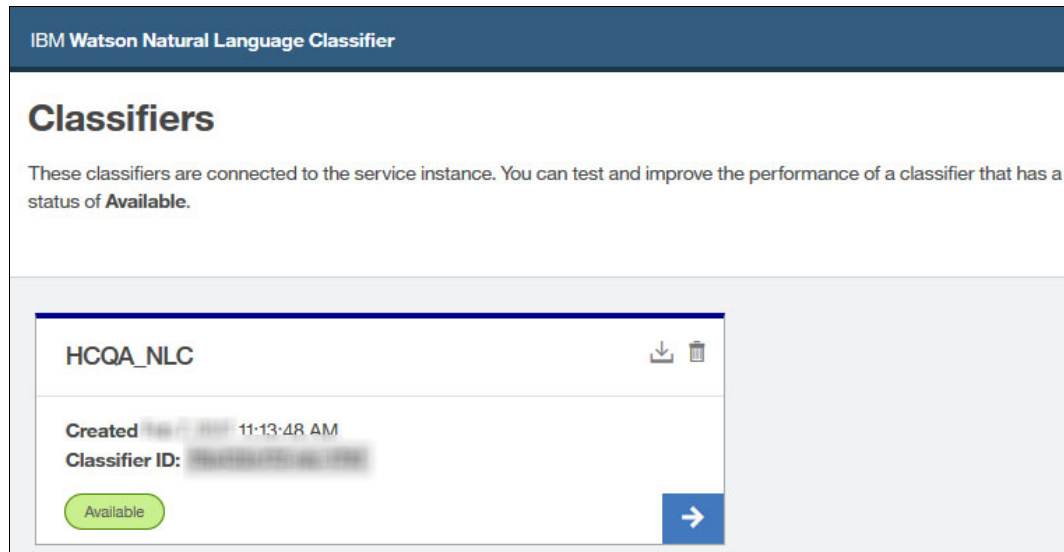


Figure 3-12 Training process step 8: Training process completed

3.4.4 Creating the Node.js Express Healthcare Q and A application

To create the Healthcare Q and A application, complete these steps.

1. Download the project from the Git repository:

<https://github.com/snippet-java/redbooks-nlc-201-healthcare-nodejs-student.git>

2. Set the dependencies:

- a. Navigate to the downloaded application folder, and open the package.json file:

```
../hcqaNaturalLanguageClassifier/package.json
```

The file has a list of required dependencies (Figure 3-13).

```
1 {
2   "name": "hcqanaturallanguageclassifier",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./bin/www"
7   },
8   "dependencies": {
9     "body-parser": "~1.15.2",
10    "cookie-parser": "~1.4.3",
11    "debug": "~2.2.0",
12    "ejs": "~2.5.2",
13    "express": "~4.14.0",
14    "morgan": "~1.7.0",
15    "serve-favicon": "~2.3.0"
16  }
17 }
18
```

Figure 3-13 Snapshot from the package.json file: Dependencies

- b. In the package.json file, add the following dependency to line number 16 (Figure 3-14 on page 29):

```
"watson-developer-cloud": "2.14.8"
```

The Watson module will provide access to the high-level wrappers for each of the Watson cognitive services running on IBM Bluemix.


```

1  {
2    "name": "hcqanaturallanguageclassifier",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "body-parser": "~1.15.2",
10     "cookie-parser": "~1.4.3",
11     "debug": "~2.2.0",
12     "ejs": "~2.5.2",
13     "express": "~4.14.0",
14     "morgan": "~1.7.0",
15     "serve-favicon": "~2.3.0",
16     "watson-developer-cloud": "~2.14.8"
17   }
18 }
19

```

Figure 3-14 The package.json file after adding Watson-developer-cloud to dependency list

3. Set the service credentials and classifier ID.
 - a. Open the config.js configuration file /hcqaNaturalLanguageClassifier/config.js to add the Natural Language Classifier service credentials and classifier ID.
 - b. Add the values for the username and password that you obtained when you created the service as described in Chapter 2, "Creating a Natural Language Classifier service in Bluemix" on page 11.
 - c. Add the classifier ID that you obtained when you created the classifier in step 8 on page 27. See Figure 3-15.

```

1  var config = {
2    watson: {
3    natural_language_classifier: {
4      username: "",
5      password: "",
6      version: "v1",
7      id: ""
8    }
9  }
10 };
11 module.exports = config;

```

Figure 3-15 Snapshot from the config.js file: Setting service credentials and classifier ID

- d. Save the config.js file.
4. Configure the application route. This route specifies the URI that points to the file that includes the required application actions.
 - a. Open the following file:

```
../hcqaNaturalLanguageClassifier/app.js
```
 - b. Add the following text to line number 10 (Figure 3-16 on page 30), and add an extra new line after that:

```
var nlc = require('./routes/nlc');
```

This line of code specifies the route file (n1c in this example).

```
1  var express = require('express');
2  var path = require('path');
3  var favicon = require('serve-favicon');
4  var logger = require('morgan');
5  var cookieParser = require('cookie-parser');
6  var bodyParser = require('body-parser');
7
8  var index = require('./routes/index');
9  var users = require('./routes/users');
10 var n1c = require('./routes/n1c');
11
12 var app = express();
13
14 // view engine setup
15 app.set('views', path.join(__dirname, 'views'));
16 app.set('view engine', 'ejs');
17
18 // uncomment after placing your favicon in /public
19 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
20 app.use(logger('dev'));
21 app.use(bodyParser.json());
22 app.use(bodyParser.urlencoded({ extended: false }));
23 app.use(cookieParser());
24 app.use(express.static(path.join(__dirname, 'public')));
25
26 app.use('/', index);
27 app.use('/users', users);
28
29
30 // catch 404 and forward to error handler
31 app.use(function(req, res, next) {
32   var err = new Error('Not Found');
33   err.status = 404;
34   next(err);
35 });
36
37 // error handler
38 app.use(function(err, req, res, next) {
39   // set locals, only providing error in development
40   res.locals.message = err.message;
41   res.locals.error = req.app.get('env') === 'development' ? err : {};
42
43   // render the error page
44   res.status(err.status || 500);
```

Figure 3-16 Snapshot from the app.js file highlighting the location of newly added code

- c. Specify the URI that the application will use to load and access the file.
Add the following text to line number 28 in the app.js file (Figure 3-17 on page 31):
`app.use('/n1c', n1c);`

```
25
26 app.use('/', index);
27 app.use('/users', users);
28 app.use('/nlc', nlc);
29
30 // catch 404 and forward to error handler
31 app.use(function(req, res, next) {
32   var err = new Error('Not Found');
33   err.status = 404;
34   next(err);
35 });
36
```

Figure 3-17 Snapshot from the app.js file highlighting the route file and URI

- d. Review the added lines as shown in Figure 3-18, and then save and close the file.

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var index = require('./routes/index');
9 var users = require('./routes/users');
10 var nlc = require('./routes/nlc');
11
12 var app = express();
13
14 // view engine setup
15 app.set('views', path.join(__dirname, 'views'));
16 app.set('view engine', 'ejs');
17
18 // uncomment after placing your favicon in /public
19 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
20 app.use(logger('dev'));
21 app.use(bodyParser.json());
22 app.use(bodyParser.urlencoded({ extended: false }));
23 app.use(cookieParser());
24 app.use(express.static(path.join(__dirname, 'public')));
25
26 app.use('/', index);
27 app.use('/users', users);
28 app.use('/nlc', nlc);
29
30 // catch 404 and forward to error handler
31 app.use(function(req, res, next) {
32   var err = new Error('Not Found');
33   err.status = 404;
34   next(err);
35 });
36
37 // error handler
38 app.use(function(err, req, res, next) {
39   // set locals, only providing error in development
40   res.locals.message = err.message;
41   res.locals.error = req.app.get('env') === 'development' ? err : {};
42
43   // render the error page
44   res.status(err.status || 500);

```

Figure 3-18 Snapshot for the app.js file highlighting the updates

By adding these lines, the application will load the new route file any time the user accesses the application using this URI:

```
http://server:port/nlc
```

5. Integrate the Healthcare application with the Watson Natural Language Classifier service by creating the server-side code. In this step, create the file that includes the implemented code to be run when the route is accessed:

- a. Create the `nlc.js` file in the following path:

```
../hcqaNaturalLanguageClassifier/routes/nlc.js
```

- b. Open the `nlc.js` file and add the code snippet shown in Example 3-1, which loads the required node modules and the previously created `config.js` file.

Example 3-1 An nlc.js snippet: Loads the required node modules

```
var config = require('../config');
var express = require('express');
var bodyParser = require('body-parser');
var watson = require('watson-developer-cloud');
```

- c. Set up the Express router:

```
var router = express.Router();
```

- d. Create a `body-parser` instance for working with the URL that is encoded from data. The `body-parser` middleware will be used to gain access to HTML form parameters.

```
var urlEncodedParser = bodyParser.urlencoded({extended:false});
```

- e. Declare an instance of the Natural Language Classifier service and use the credentials and classifier ID in JSON format in the configuration file, as shown in Example 3-2.

Example 3-2 Creating the Natural Language Classifier service instance: The nlc.js snippet

```
var urlEncodedParser = bodyParser.urlencoded({extended:false});
//Declare an instance of the NLC service; use the credentials and classifier
//id in JSON format in the configuration file
var natural_language_classifier =
watson.natural_language_classifier(config.watson.natural_language_classifier
);
```

- f. Add a function to handle HTTP POST requests to URL `/nlc`. This function will perform classification on the input text entered by the user. The data will be submitted through an HTML form in a parameter named `source`. Example 3-3 shows these details:

- i. Perform classification on the `source` form parameter.
- ii. A function to be run when classification analysis is completed.
- iii. Print the returned errors to the server console.
- iv. Print the returned results data to the server console.
- v. Send the returned JSON data to the client application.

Example 3-3 Classifying input text: nlc.js snippet

```
router.post('/', urlEncodedParser, function(req, res, next){
//Perform classification on the source form parameter.
  natural_language_classifier.classify({
    'text': req.body.source,
    'classifier_id': config.watson.natural_language_classifier.id
  },
//A function to be run when classification analysis is completed
```

```

    function(err, response){
        if(err){
//Print the returned errors to the server console.
            console.log('error:', err);
        }else{
//Print the returned results data to the server console
            console.log(JSON.stringify(response, null, 2));
//Send the returned JSON data to the client application
            res.json(response);
        }
    }
});
});

```

- g. To the end of the file, add the following line, which will make the routes available to the remainder of the application:

```
module.exports = router;
```

The complete file is shown in Figure 3-19.

```

1  var config = require('../config');
2  var express = require('express');
3  var bodyParser = require('body-parser');
4  var watson = require('watson-developer-cloud');
5
6  var router = express.Router();
7  var urlEncodedParser = bodyParser.urlencoded({extended:false});
8
9
10 //Declare an instance of the NLC service by using the credentials and classifier id in JSON format in the configuration file
11 var natural_language_classifier = watson.natural_language_classifier(config.watson.natural_language_classifier);
12
13 router.post('/', urlEncodedParser, function(req, res, next){
14     natural_language_classifier.classify({
15         'text': req.body.source,
16         'classifier_id': config.watson.natural_language_classifier.id
17     },
18     //A function to be run when classification analysis is completed.
19     function(err, response){
20         if(err){
21             //Print the returned errors into the server console.
22             console.log('error:', err);
23         }else{
24             //Print the returned results data into the server console.
25             console.log(JSON.stringify(response, null, 2));
26             //Send the returned JSON data to the client application.
27             res.json(response);
28         }
29     });
30 });
31
32 module.exports = router;

```

Figure 3-19 Snapshot for the nlc.js file

The Node.js route, which is the nlc.js file that will run the specified cognitive services on data submitted through an HTML form has been created.

6. Review the index.js file. This file is the application main page that includes the input field and renders the results in a new page by using EJS templates.
 - a. Open the index.ejs file, which is in the following path:

```
../hcqaNaturalLanguageClassifier/views/index.ejs
```
 - b. Review the code, which has the following details:
 - A function for sending data to the server through an AJAX request (Figure 3-20 on page 34).

```

function classify() {
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200) {
        var json = JSON.parse(xhr.responseText);
        document.getElementById("result").innerHTML = json.top_class;
        document.getElementById("submitButton2").setAttribute("data-target", "#" + json.top_class);
        document.getElementById("submitButton2").click();
      }
    }
  }
  xhr.open("POST", "./nlc", true);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send("source=" + document.getElementById("source").value);
}

```

Figure 3-20 *index.ejs: JavaScript classify function*

- A simple function for resetting or clearing values in the page (Figure 3-21).

```

function clearData() {
  document.getElementById("source").value = "";
  document.getElementById("result").innerHTML = "";
}

```

Figure 3-21 *index.ejs: JavaScript clearData function*

- A simple text area to accept input from the user (Figure 3-22).

```

<p/>
<div><textarea id="source" name="source" rows="10" cols="75"></textarea></div>
<p/>

```

Figure 3-22 *index.ejs: textarea*

- Buttons for invoking the JavaScript functions `classify()` and `clearData()` (Figure 3-23).

```

<div>
  <button id="submitButton" onclick="classify()" type="button" class="btn btn-info">Submit</button>
  <button id="submitButton2" type="button" data-toggle="modal" style="display:none"></button>
  <button onclick="clearData()" type="button" class="btn btn-info" data-toggle="modal" >Clear</button>
</div>

```

Figure 3-23 *index.ejs: Buttons for invoking JavaScript functions*

- Modals for each class that displays the answers for different categories and classes (Figure 3-24 on page 35).

```

<!-- Modal -->
<div id="Policy" class="modal fade" role="dialog" style="z-index:999999999;">
  <div class="modal-dialog">
    <!-- Modal content-->
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal">*</button>
        <h3 class="modal-title">Policies and Plans</h3>
      </div>
      <div class="modal-body">
        <div class="container-fluid">
          <h4>1. Group Life & Medical Plans</h4>
          <p>Like all well run companies, yours will rely on your people to succeed. It's therefore crucial to offer the best benefits package you can, to attract also worth remembering that many employees today see these benefits as a key consideration when assessing the employment market</p>
          <p class="calltoaction"><a href="/en/Business/Find-our-products/Group-Life--Medical-Plans/index.html" class="calltoactionlink">Learn more about our Group
          <h4>2. Have your circumstances changed? <strong>&nbsp;</strong></h4>
          <p>We want to make sure that you always have the right protection for your circumstances. If things change &ndash; for example, you may start a family, amount or type of protection you need.&nbsp;</p>
          <p>So if anything changes, just call us on <strong>19798</strong> (from outside Egypt +202 2 461 9020) so we can make sure you're still receiving t
          <h4>5. List of required documents for any change in your policy? <strong>&nbsp;</strong></h4>
          <p>Here is our required document for any change in your policy.</p>
          <p>&bull; <a href="/en/home/assets/pdf/required-documents-for-policy-changes1.pdf">Required documents for policy changes.</a></p>
        </div>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-info" data-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
<!-- Modal -->

```

Figure 3-24 index.ejs: Examples of Modals

- c. Close the index.ejs file.

3.4.5 Deploying the Healthcare Q and A application on Bluemix

The steps in this section guide in pushing the final application to the Bluemix environment and making it publicly accessible to consumers:

1. Open the manifest.yml file (Figure 3-25) and review its contents:

```
../hcqaNaturalLanguageClassifier/manifest.yml
```

```

1  ---
2  applications:
3  - path: .
4    memory: 256M
5    instances: 1
6    domain: mybluemix.net
7    name: hcqaNaturalLanguageClassifier
8    host: hcqaNaturalLanguageClassifier
9    buildpack: sdk-for-nodejs
10   disk_quota: 1024M

```

Figure 3-25 The manifest.yml file

This is a helper file for identifying and pushing the application to the IBM Bluemix environment. The file is used to declare resources and metadata required for your application to run in Bluemix and also used to bind existing services to the application.

2. Each application hosted in IBM Bluemix must have a unique sub-domain. To distinguish your application from others, make the host name unique by appending the initials of your name to the host value.

```
host: hcqaNaturalLanguageClassifier-AS
```

Note: Replace AS by your initials

3. Save and close the file after updates.

- Review the Bluemix regions (Table 3-1). A Bluemix region is a defined geographical territory that you can deploy your applications to. Select the appropriate Bluemix region closer to your location to reduce the application latency.

Table 3-1 Regions

Region	Location	Prefix	cf API endpoint	UI console
US South	Dallas, US	ng	api.ng.bluemix.net	console.ng.bluemix.net
United Kingdom	London, UK	eu-gb	api.eu-gb.bluemix.net	console.eu-gb.bluemix.net
Sydney	Sydney, Australia	au-syd	api.au-syd.bluemix.net	console.au-syd.bluemix.net

- Set the API endpoint for the Bluemix region you selected (from Table 3-1) and log in to Bluemix using your Bluemix credentials.

Return to the command window and enter the commands shown in Example 3-4.

Example 3-4 Enter these commands in the command window

```
cf api https://api.ng.bluemix.net
cf login -u <<Your Bluemix username>>
```

- When prompted enter your Bluemix password.
- Push the health care application to be deployed in Bluemix.

In the command prompt window enter the commands shown in Example 3-5.

Example 3-5 Deploying the application to Bluemix

```
cd <<The project path username>>\hcqaNaturalLanguageClassifier
cf push 'hcqaNaturalLanguageClassifier'
```

The Cloud Foundry command-line tool will examine the contents of the `manifest.yml` and `package.json` files and push the application to Bluemix.

- Exit the CF tool:

```
cf logout
```

3.4.6 Testing the application

To test the Healthcare Q and A application, complete these steps:

- In the web browser, enter the following URL to open the application. Replace `_AS` by your initials as you entered them in step 2 on page 35 (Figure 3-26 on page 37):

```
http://hcqanaturallanguageclassifier-AS.mybluemix.net/
```

Note: To assess the quality of the training, the test input questions should not be *exactly* the same as the questions included in the training set.

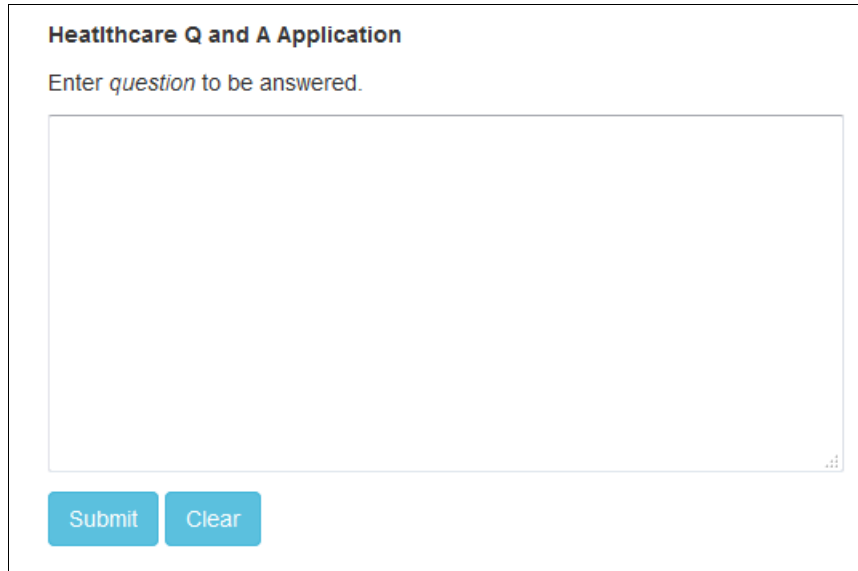


Figure 3-26 Healthcare Q and A application

2. Enter a test question, such as “Where are clinics?” (which represents a Provider class question) and click **Submit** (Figure 3-27). The figure shows that the question is classified as *Providers*, as expected.

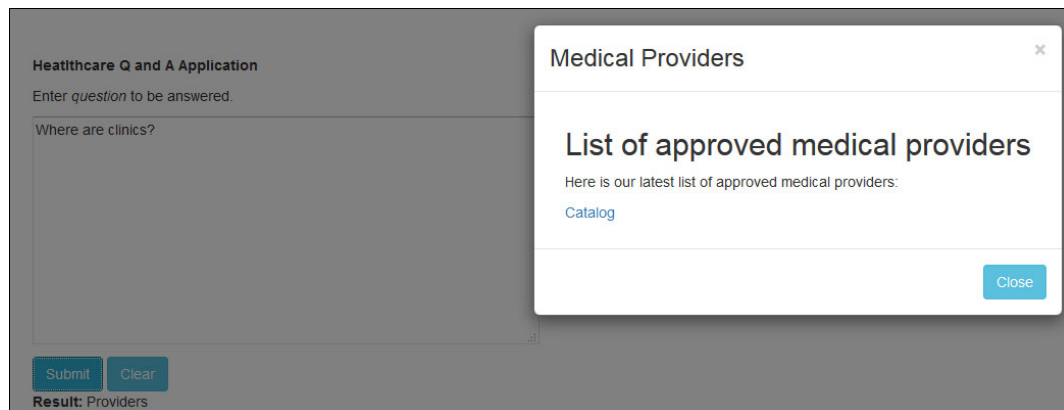


Figure 3-27 Healthcare Q and A application results displaying Providers class results

3. Check the logs by first opening the application from the dashboard (Figure 3-28). Click the application.

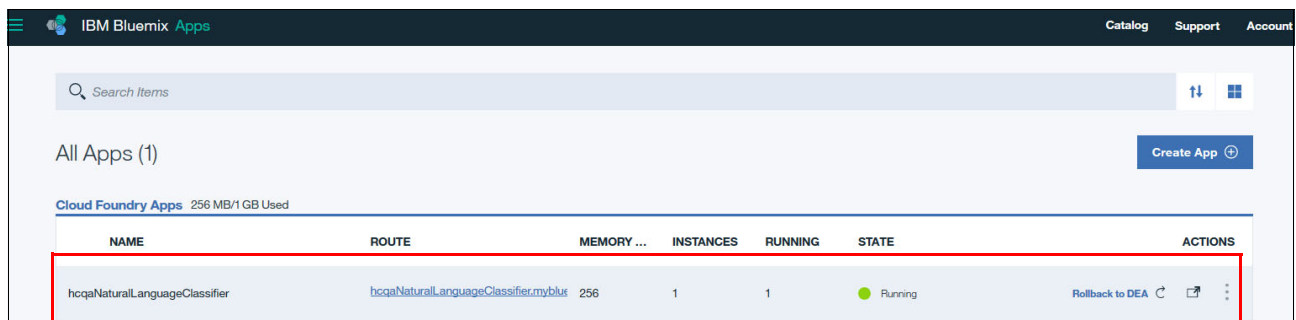


Figure 3-28 Healthcare Q and A application

Then, on the left panel, click **Logs** to view the application logs (Figure 3-29).

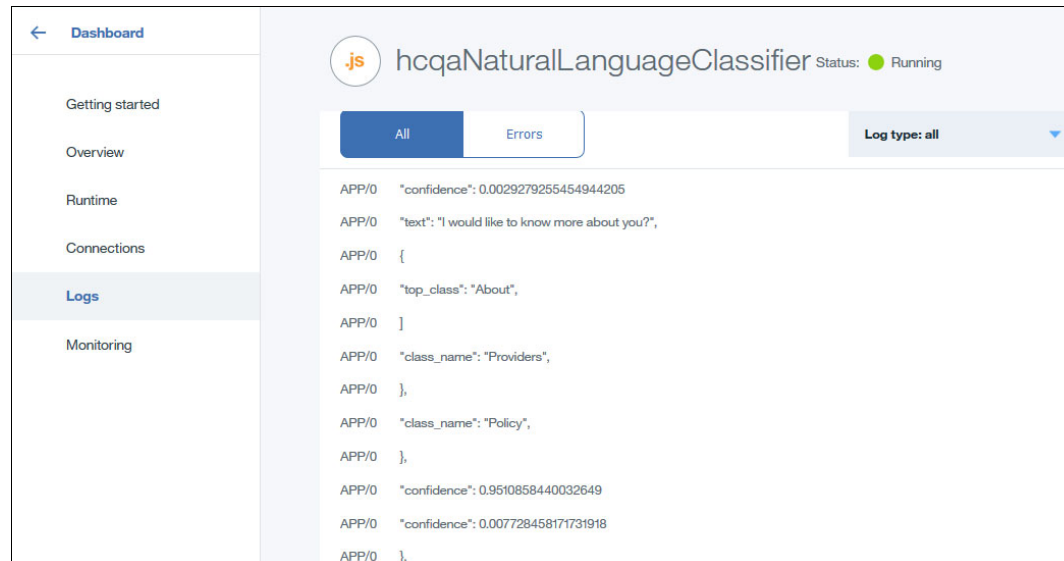


Figure 3-29 Returned JSON results from the classifier

4. Back to the web application, enter a test question which represents an About class question such as “I would like to know more about you?” and click **Submit** (Figure 3-30). The figure shows that the question is classified as *About*, as expected.

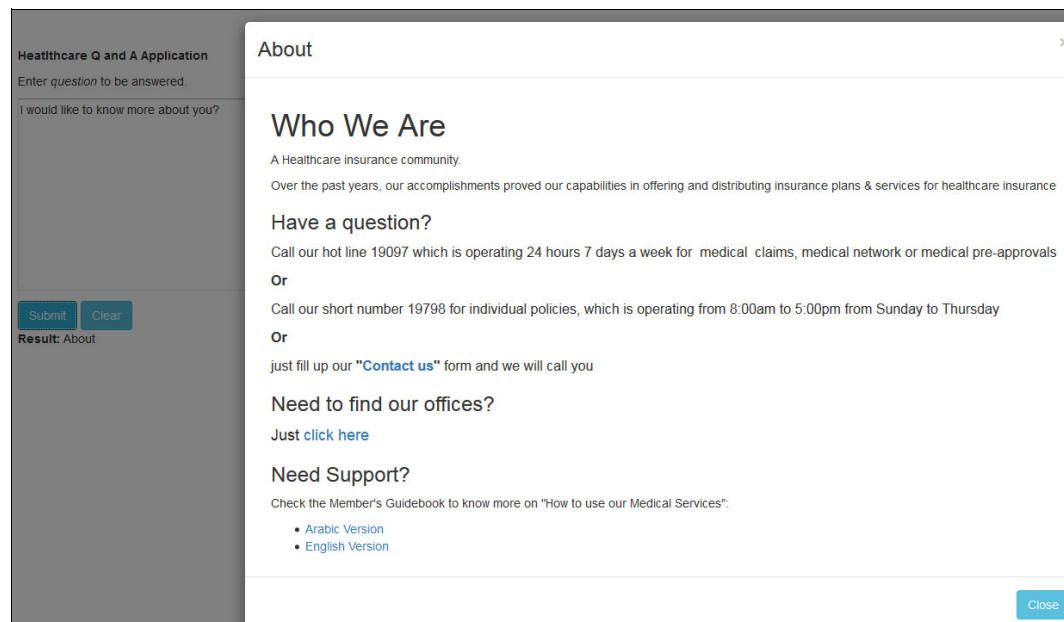


Figure 3-30 Healthcare Q and A application results displaying About class results

5. Review the logs (Figure 3-31).

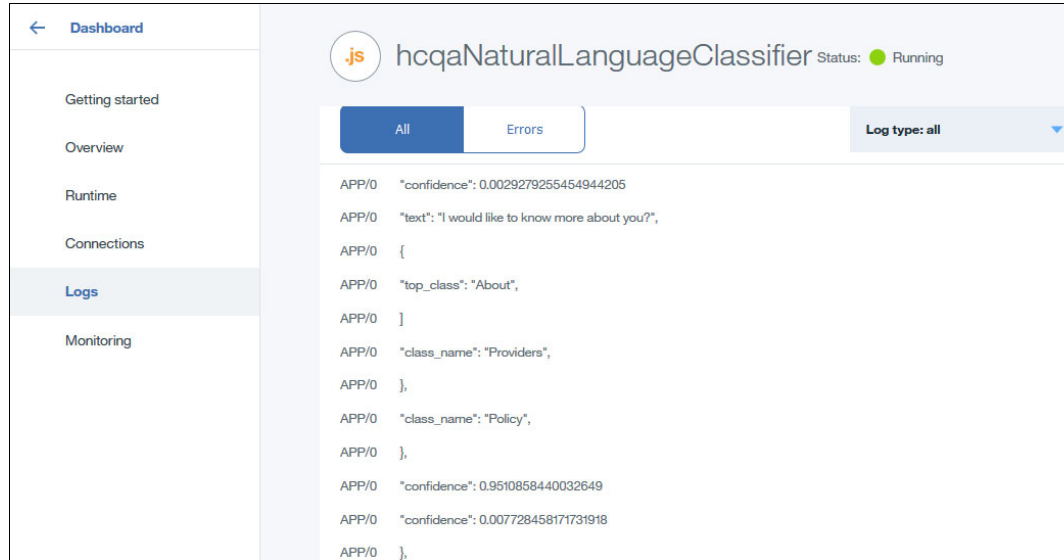


Figure 3-31 Returned JSON results from the classifier

6. Enter a question, such as “What are the policies?” (which represents a Policy class question) and click **Submit**. (Figure 3-32). The figure shows that the question is classified as *Policy*, as expected.

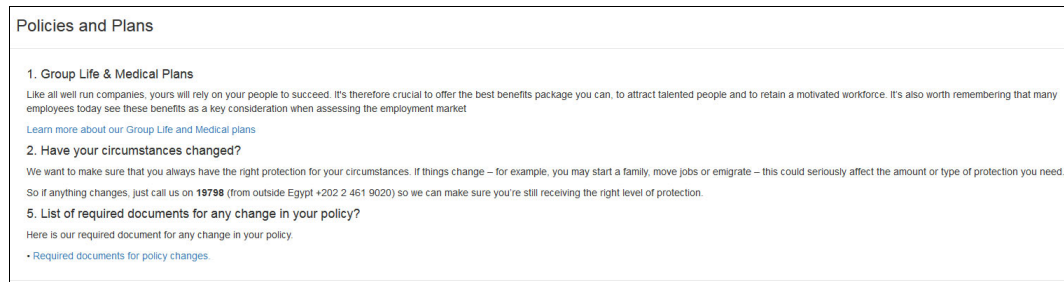


Figure 3-32 Healthcare Q and A application results displaying Policy class results

7. Review the logs (Figure 3-33).

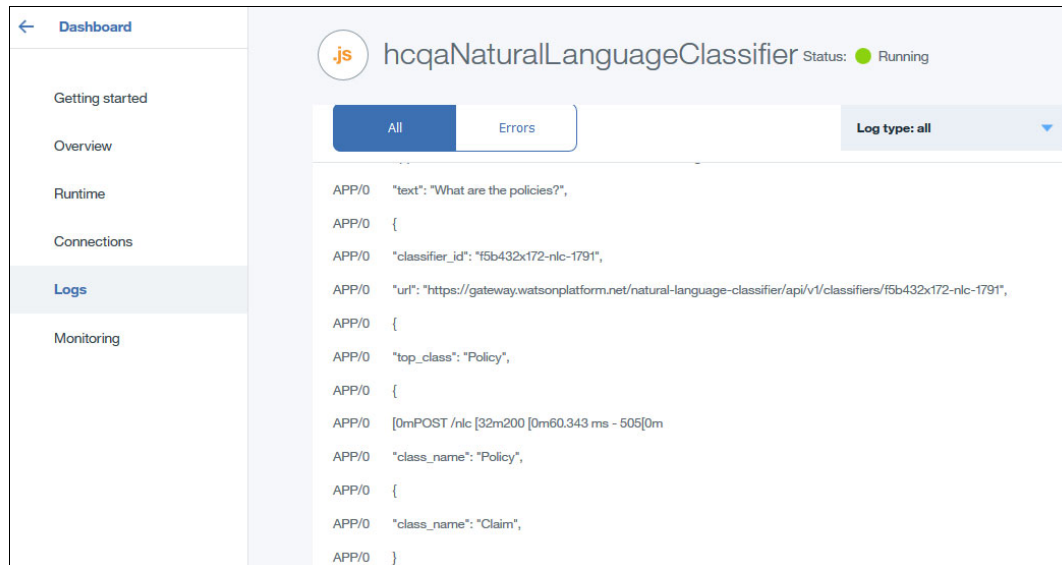


Figure 3-33 Returned JSON results from the classifier

8. Enter a test question, such as “How to claim?” (which represents a Claim class question) and click **Submit** (Figure 3-34). The figure shows that the question is classified as *Claim*, as expected.

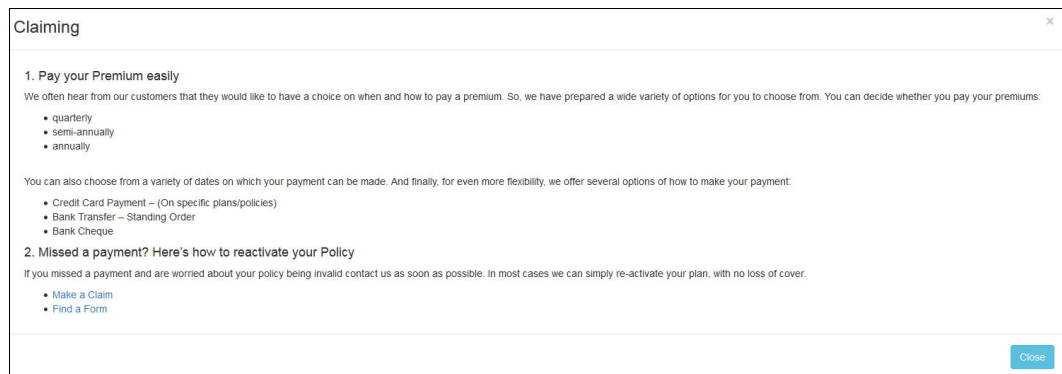


Figure 3-34 Healthcare Q and A application results displaying Claim class results

9. Review the logs (Figure 3-35).

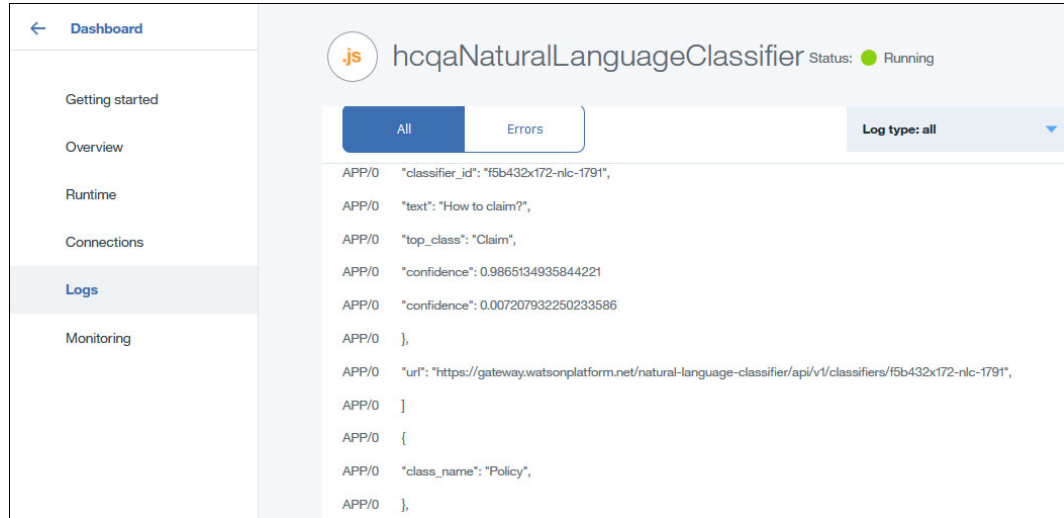


Figure 3-35 Returned JSON results from the classifier

10. Enter a test question, such as “Can you provide background about the products and offers?” (which represents a Products class question) and click **Submit** (Figure 3-36). The figure shows that the question is classified as *Products*, as expected.

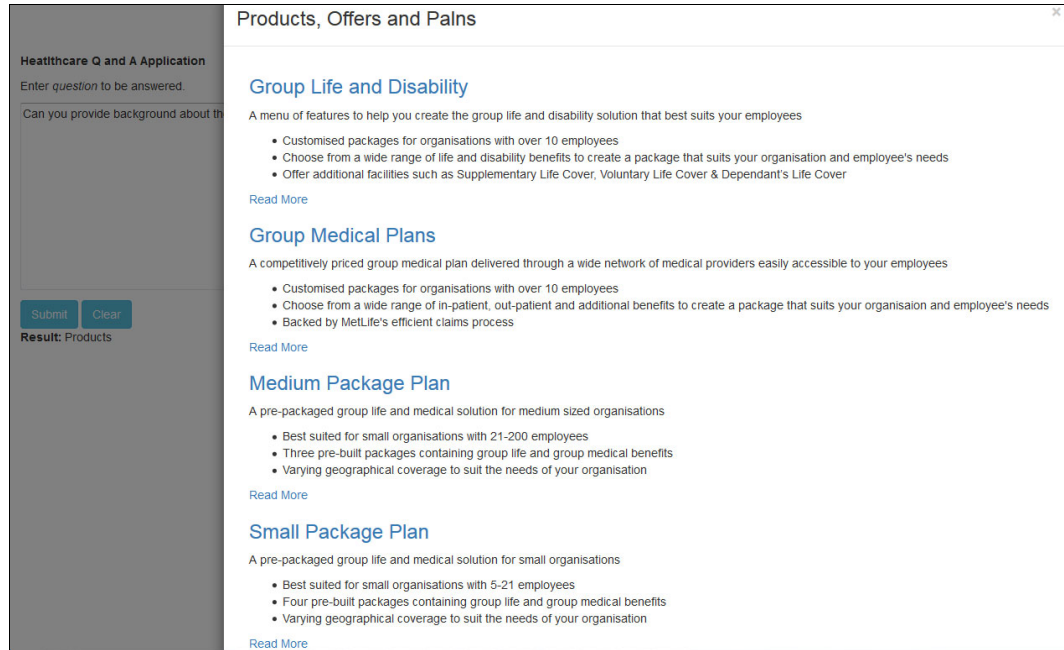


Figure 3-36 Healthcare Q and A application results displaying Products class results

- ▶ Node.js:
<https://nodejs.org/en/>
- ▶ Express and Node.js tutorial:
<https://codeforgeek.com/2014/06/express-nodejs-tutorial/>
- ▶ Natural Language Classifier; Authentication:
<http://www.ibm.com/watson/developercloud/natural-language-classifier/api/v1/?node#authentication>
- ▶ Getting Started with IBM Watson Node.js SDK:
<http://www.slideshare.net/pgodby/getting-started-with-ibm-watson-apis-sdks>



News Classification

Watson Natural Language Classifier (NLC) provides a machine-learning classifier that combines complex convolutional neural networks with a sophisticated language model to learn and understand language. Behind this complexity, the Watson Natural Language Classifier service is easy to use.

This use case shows a web application, named *News Classification*, that calls a classifier that is already trained in using public news data, and the classifier responds with the type of news the text is related to.

This chapter describes how to use the Natural Language Classifier service to develop a sample use case in Java that classifies news text into five types:

- ▶ Business
- ▶ Entertainment
- ▶ Politics
- ▶ Technology
- ▶ Sports

The following topics are covered in this chapter:

- ▶ Getting started
- ▶ Architecture
- ▶ Two ways to deploy the application: Step-by-step and quick deploy
- ▶ Step-by-step implementation
- ▶ Quick deployment of application
- ▶ References

4.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

4.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- ▶ Prepare training and test data
- ▶ Create and train a Natural Language Classifier instance
- ▶ Run test queries
- ▶ Evaluate the classifier with test data and check its accuracy
- ▶ Deploy a Java application that uses the classifier with Eclipse or with Git.

4.1.2 Prerequisites

Be sure the following prerequisites are met:

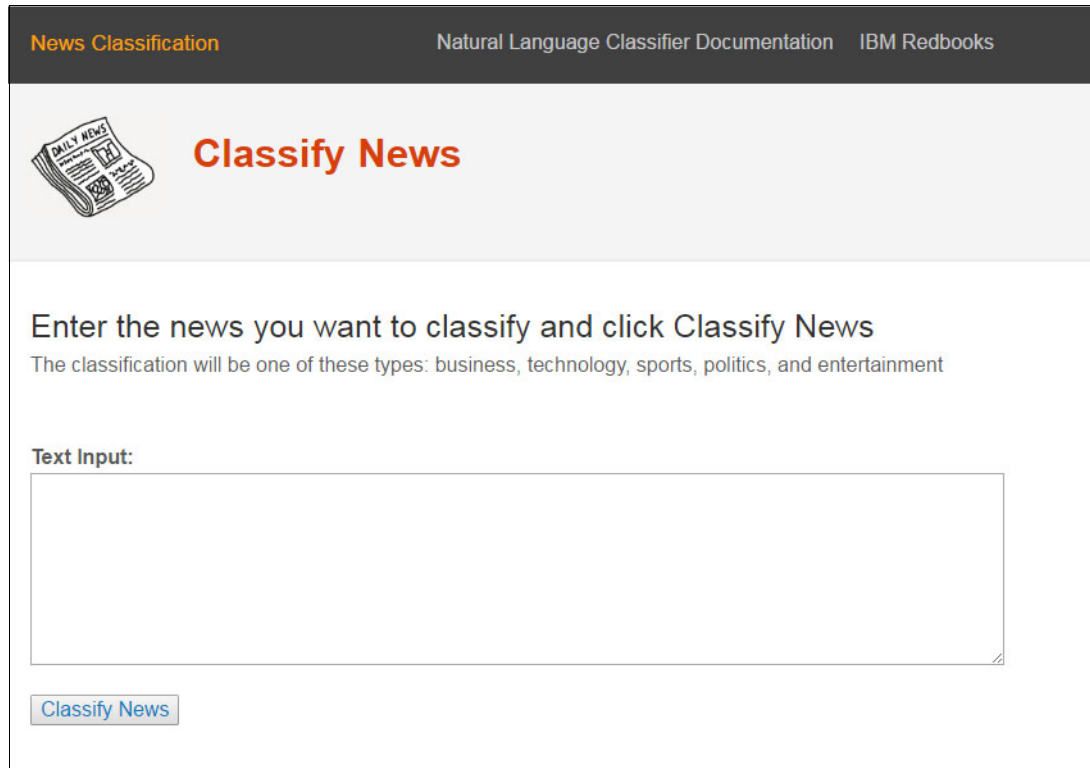
- ▶ Review Chapter 1, “Basics of Natural Language Classifier service” on page 1.
This chapter is important to help you understand the basics of Natural Language Classifier.
- ▶ Create a Natural Language Classifier service instance as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11.
- ▶ Some Java programming language background
A basic Java programming background is important to understand the code.
- ▶ IBM Bluemix account
Bluemix is an open standard, cloud platform for building, running, and managing applications and services. A Bluemix account is essential because the sample Java web application is deployed into it.
For more information, see the [Bluemix](#) website.

These software requirements are also necessary:

- ▶ Access to a Windows desktop or Linux.
- ▶ IBM SDK, Java Technology Edition, Version 8, which you can [download](#). This prerequisite applies if you are using Eclipse.
- ▶ If you use Eclipse, then install and set up [Eclipse Neon](#) with Bluemix tools.
- ▶ If you are using Git client:
 - [Git download and installation](#)
 - [Cloud Foundry download and installation](#)

4.1.3 Expected results

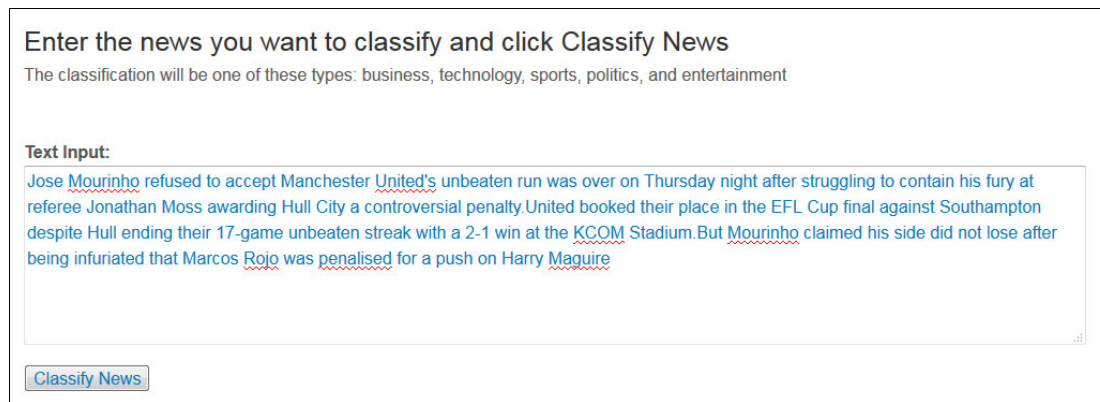
Figure 4-1 shows the home page of the [News Classification web application](#) that you can test. Here the user inputs text.



The screenshot shows the home page of the News Classification web application. At the top, there is a dark header with the text "News Classification" in orange, "Natural Language Classifier Documentation" in white, and "IBM Redbooks" in white. Below the header, there is a light gray banner with a newspaper icon on the left and the text "Classify News" in orange. The main content area has a white background and contains the following text: "Enter the news you want to classify and click Classify News" followed by "The classification will be one of these types: business, technology, sports, politics, and entertainment". Below this is a "Text Input:" label and a large empty text input field. At the bottom of the input field is a "Classify News" button.

Figure 4-1 News Classification home page

A user enters news information in the Text input field, and clicks **Classify News** (Figure 4-2).



The screenshot shows the same web application interface as Figure 4-1, but with text entered into the input field. The text is: "Jose Mourinho refused to accept Manchester United's unbeaten run was over on Thursday night after struggling to contain his fury at referee Jonathan Moss awarding Hull City a controversial penalty. United booked their place in the EFL Cup final against Southampton despite Hull ending their 17-game unbeaten streak with a 2-1 win at the KCOM Stadium. But Mourinho claimed his side did not lose after being infuriated that Marcos Rojo was penalised for a push on Harry Maguire". The text is displayed in blue with red wavy underlines indicating spelling corrections. The "Classify News" button is still visible at the bottom.

Figure 4-2 Input text on news classification application

Then, the application lists the classification type in the Top Class field (Figure 4-3). In this case, Top Class indicates news input is sports.

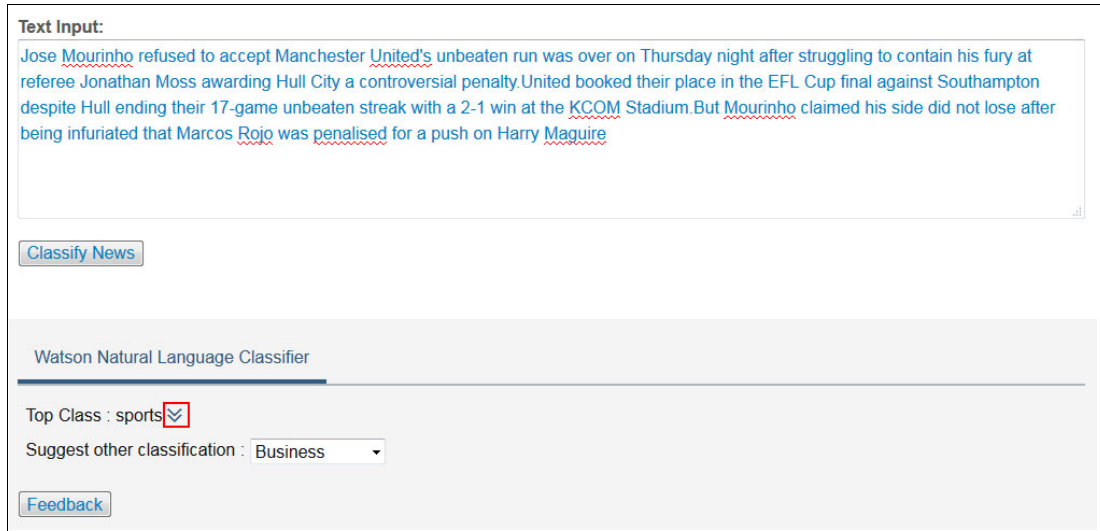


Figure 4-3 News classification results

To see the classification details, click the double down-arrow icon to the right of the top class to expand classification information.

Figure 4-4 shows the expanded details panel of the Top Class. It shows the confidence of the response and the other classes and confidences from the most confidence to least confidence.



Figure 4-4 Classification details for the sample input text

The last function is the **Feedback** button for users who do not agree with the top class result. It saves the feedback suggestion for analysis by subject matter experts (Figure 4-5).

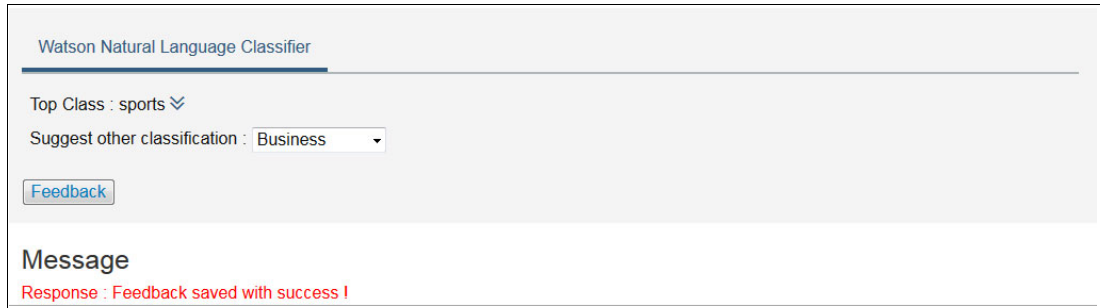


Figure 4-5 Feedback function

4.2 Architecture

An overview of the application architecture is shown in Figure 4-6 and is described next.

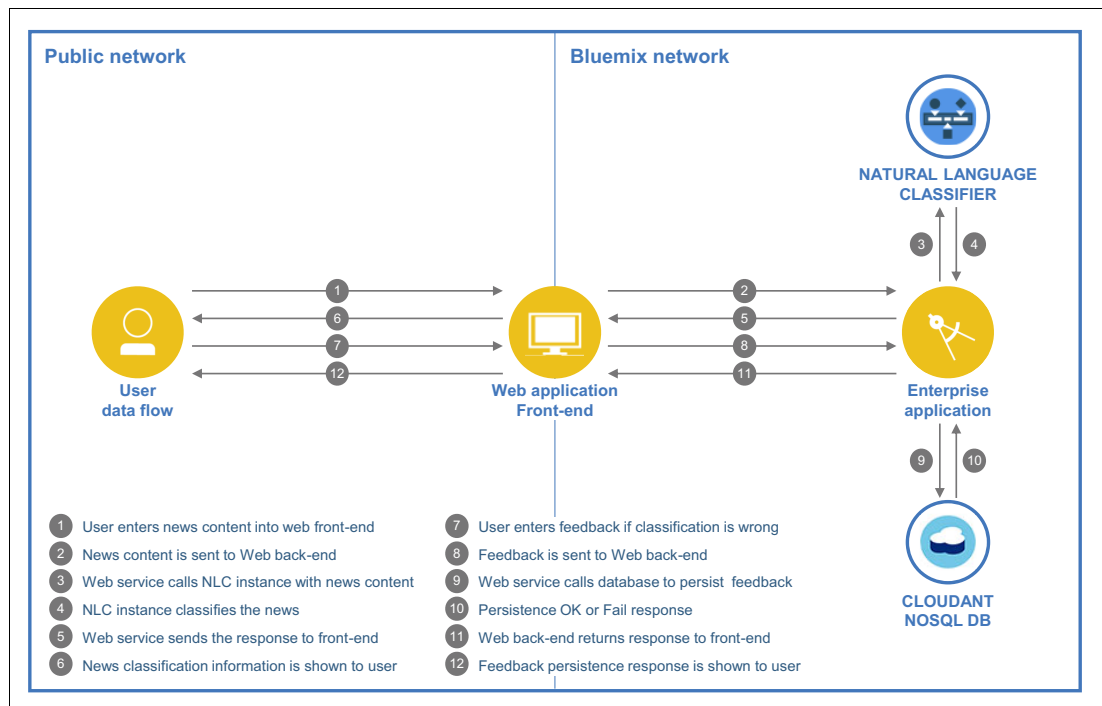


Figure 4-6 News Classification architecture diagram

The steps in the diagram are as follows:

1. The user inserts news content, as text, into the web interface and requests that it be classified (clicks the **Classify Text** button).
2. The news text is sent to the application (enterprise back end) for processing.
3. The enterprise application (web service) calls the Watson Natural Language Classifier service to evaluate what type of news is the best match for input text.
4. The Natural Language Classifier service returns the response to the enterprise application.

5. The enterprise application forwards the response to the web application front-end.
6. The web interface manages the data information, performs some front-end processing, and makes the response available for the user to view.
7. The other operation is user feedback provided if the news classification is incorrect. The user clicks **Feedback** and submits the correct classification.
8. The web application front-end passes the user feedback request to the web application back end.
9. The web application back-end calls the IBM Cloudant® noSQL DB service to persist the feedback.
10. The database responds with the insert operation result to the enterprise application.
11. The enterprise application passes the response to the web front-end.
12. The user interface displays the results to the user.

Although in this use case, the web application is deployed to the IBM WebSphere® Liberty profile, it could also be deployed to Tomcat.

For more information about application server hosting for enterprise applications, see [About WebSphere Liberty](#) and [Apache Tomcat](#).

4.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- ▶ Step-by-step deployment (*incomplete*) version of the application

This repository contains an incomplete version of the application and is used in all sections of 4.4, “Step-by-step implementation” on page 50. This version takes you through the key steps to integrate the IBM Watson services with the application logic.

- ▶ Quick deployment (*complete*) version of the application

This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 4.5, “Quick deployment of application” on page 97.

4.4 Step-by-step implementation

Implementing this use case involves the following steps:

1. Downloading the project from Git.
2. Creating a Cloudant noSQL DB service instance.
3. Reviewing the project structure.
4. Preparing training data.
5. Creating and training the classifier.
6. Querying the trained classifier.
7. Evaluating the results and updating the training data.
8. Deploying the application.
9. Testing the application.

4.4.1 Downloading the project from Git

This section explains how to download the sample *News Classification* project (incomplete version of the application), which is available at:

<https://github.com/snippet-java/redbooks-nlc-201-news-java-student.git>

You can use either of the following options:

- ▶ Import the sample Git project to Eclipse
- ▶ Clone the sample Git project by using the Git command line

Import the sample Git project to Eclipse

Install and configure Eclipse Neon with Bluemix Tools and Java SDK 8. The information to download and install the software is listed in 4.1.2, “Prerequisites” on page 46.

After setting up Eclipse in your workstation, complete these steps:

1. Import the Git project into Eclipse. Select **File** → **Import**. When the Import window opens (Figure 4-7) select **Git** → **Projects from Git** and click **Next**.

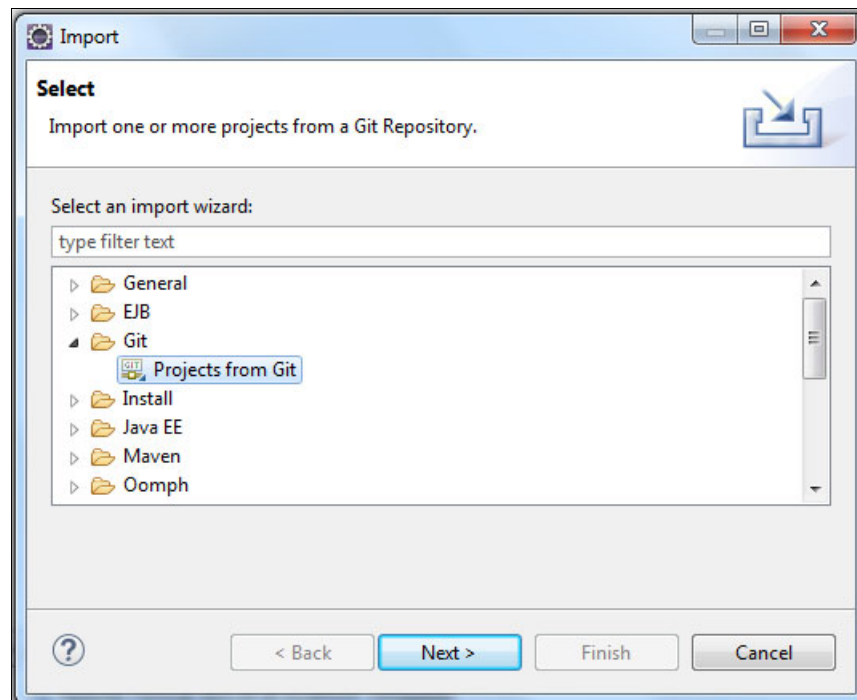


Figure 4-7 Import a Git project to Eclipse

2. In the next window (Figure 4-8), select **Clone URI**.

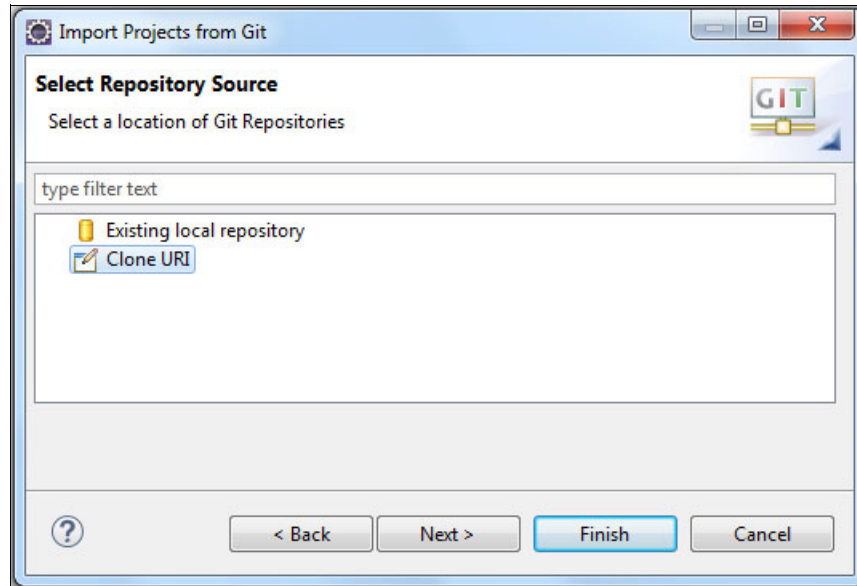


Figure 4-8 Import Projects from Git: Clone URI

3. In the next window (Figure 4-9), add the following URI, and click **Next**:
`https://github.com/snippet-java/redbooks-nlc-201-news-java-student.git`

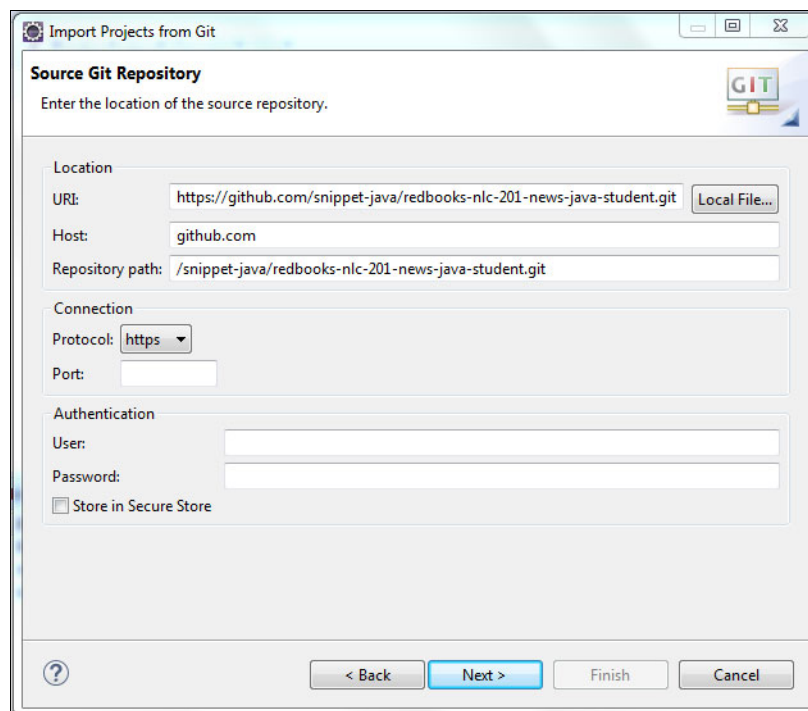


Figure 4-9 Select URI for Git repository in Eclipse

4. The Git branches are listed (Figure 4-10). Select the **master** branch and click **Next**.

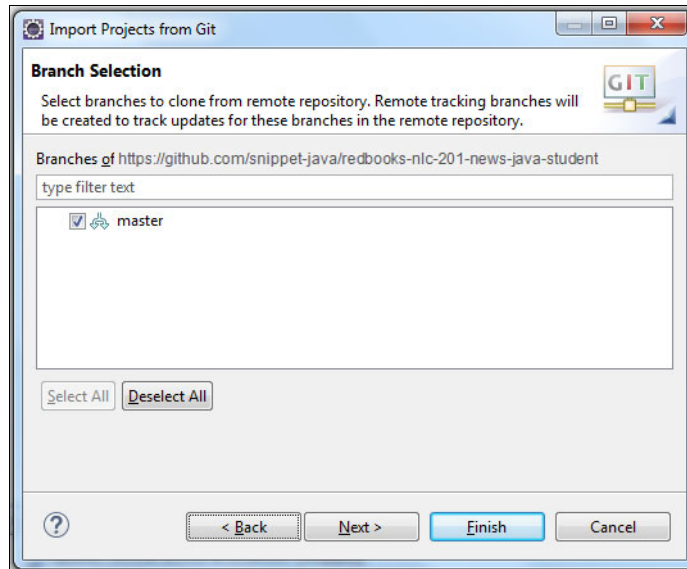


Figure 4-10 Master branch selected to import Git project

5. A local version of the Git project will be created. Specify the destination directory in your local workstation and click **Next** (Figure 4-11).

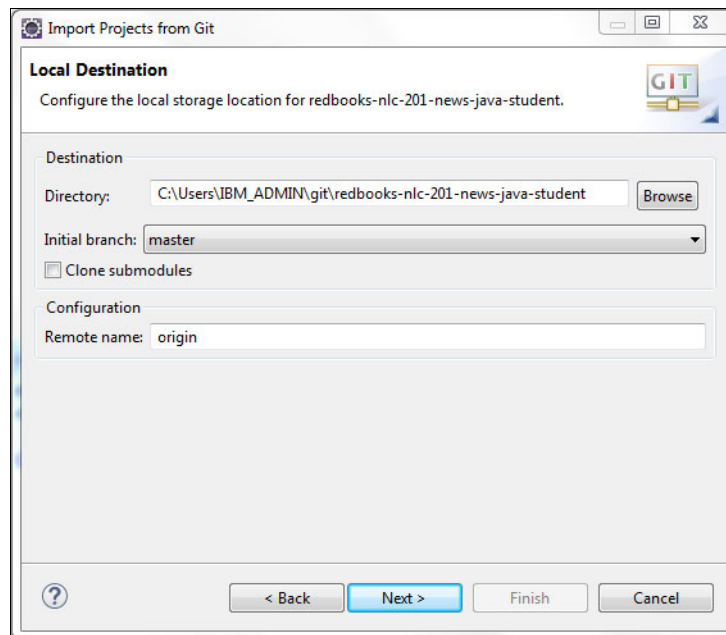


Figure 4-11 Local storage location for Git project

6. The last steps are to configure the Eclipse project. In the next window (Figure 4-12) select **Import Existing Eclipse Project** and click **Next**.

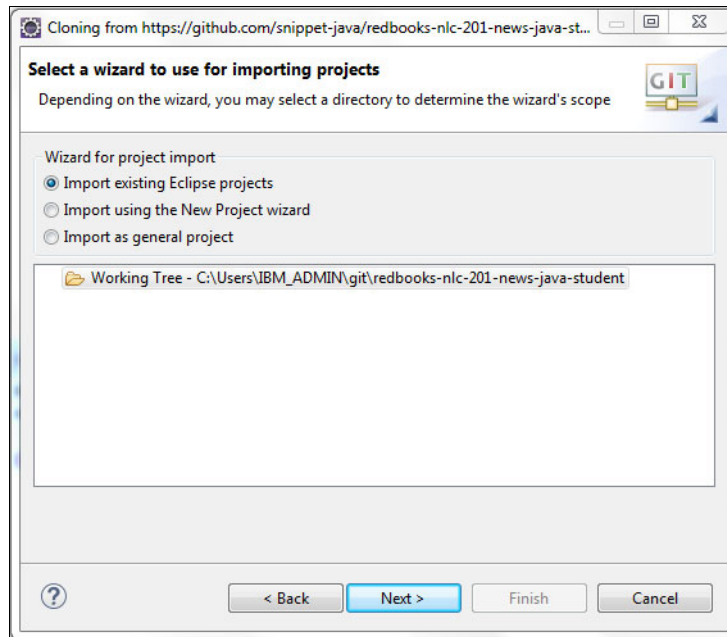


Figure 4-12 Import existing Eclipse project option

7. Confirm your settings (Figure 4-13) and click **Finish**.

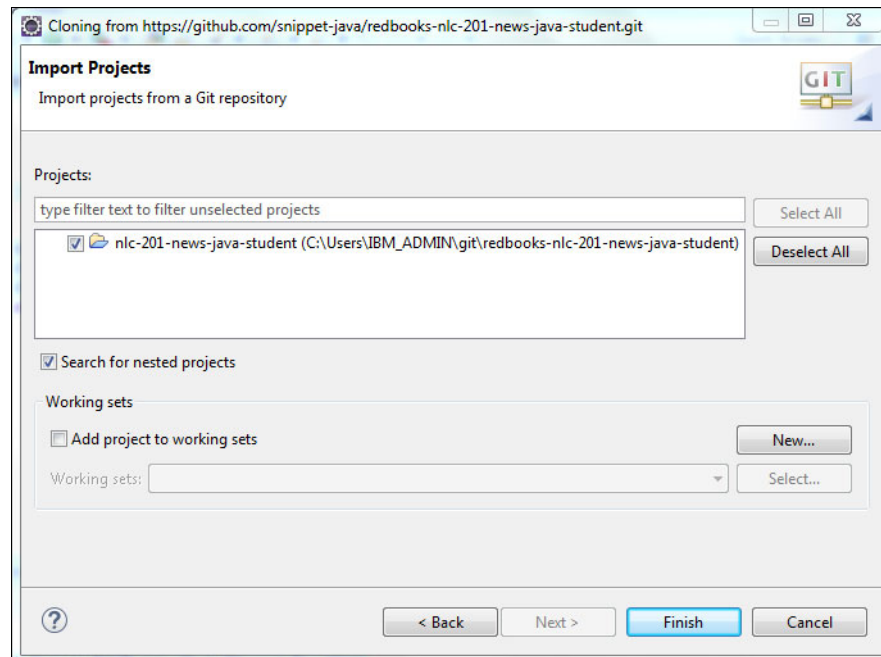


Figure 4-13 Project selected confirmation

The result shows that the Eclipse project is imported into the workspace (Figure 4-14). This project will be the platform for other steps in this chapter.

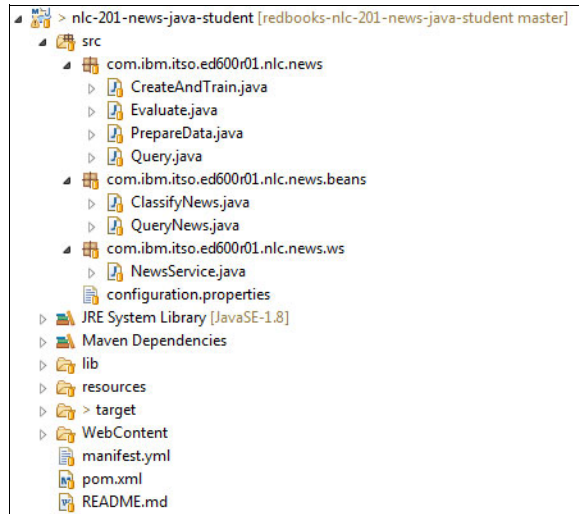


Figure 4-14 Project imported with success from Git

Clone the sample Git project by using the Git command line

If you do not want to use Eclipse for this use case, you can use Git. The requirement for this section is to install Git before you start. See 4.1.2, “Prerequisites” on page 46.

Complete the following steps:

1. Open a command prompt and set up Git by using the following command:
`git config --global http.sslVerify false`
2. Choose an empty directory to download the project code.
3. Run the command in the selected directory:
`git clone https://github.com/snippet-java/redbooks-nlc-201-news-java-student.git`
4. Change to the `redbooks-nlc-201-news-java-student` directory:
`cd redbooks-nlc-201-news-java-student`
5. Check the project content (Example 4-1):

Example 4-1 Project content

```
Directory of C:\Users\IBM_ADMIN\student\redbooks-nlc-201-news-java-student
1.031 .classpath
1.101 .project
<DIR>      .settings
<DIR>      lib
274 manifest.yml
1.755 pom.xml
2.493 README.md
<DIR>      resources
<DIR>      src
<DIR>      target
<DIR>      WebContent
```

4.4.2 Reviewing the project structure

Several project components are important to highlight:

- ▶ The `com.ibm.itso.ed600r01.nlc.news` package in the `src` folder contains the `PrepareData`, `CreateAndTrain`, `Query`, and `Evaluate` Java classes required by the process to use the Natural Language Classifier service that is described in 1.1, “Using the Natural Language Classifier service” on page 2.
- ▶ The `com.ibm.itso.ed600r01.nlc.news.beans` package in the `src` folder contains beans that are used to communicate the web front-end with web back-end by using Java to JSON format.
- ▶ The `com.ibm.itso.ed600r01.nlc.news.ws` package includes the Java web service to access the back-end application. This code receives news text to classify and feedback classification from users.
- ▶ The `resource` folder contains the training and test data set samples that are used in this use case.
- ▶ The `pom.xml` file has all Java package dependencies for this project.
- ▶ The `manifest.yml` files contains the template to deploy the application in Bluemix.

4.4.3 Creating a Cloudbant noSQL DB service instance

Another requirement for the web application is to be prepared for client feedback about the quality of the classification. A database repository will be created to save client feedback that can be used by the SMEs to improve the quality of the classification.

You can choose one of the following ways to create the Cloudbant noSQL service:

- ▶ From Bluemix
- ▶ From the command line

Create a Cloudant noSQL DB service instance from Bluemix

Complete the following steps:

1. Open the IBM Bluemix Catalog page (top menu on the right) and select **Services** → **Data & Analytics** from the left menu and click **Cloudant noSQL DB** (Figure 4-15).

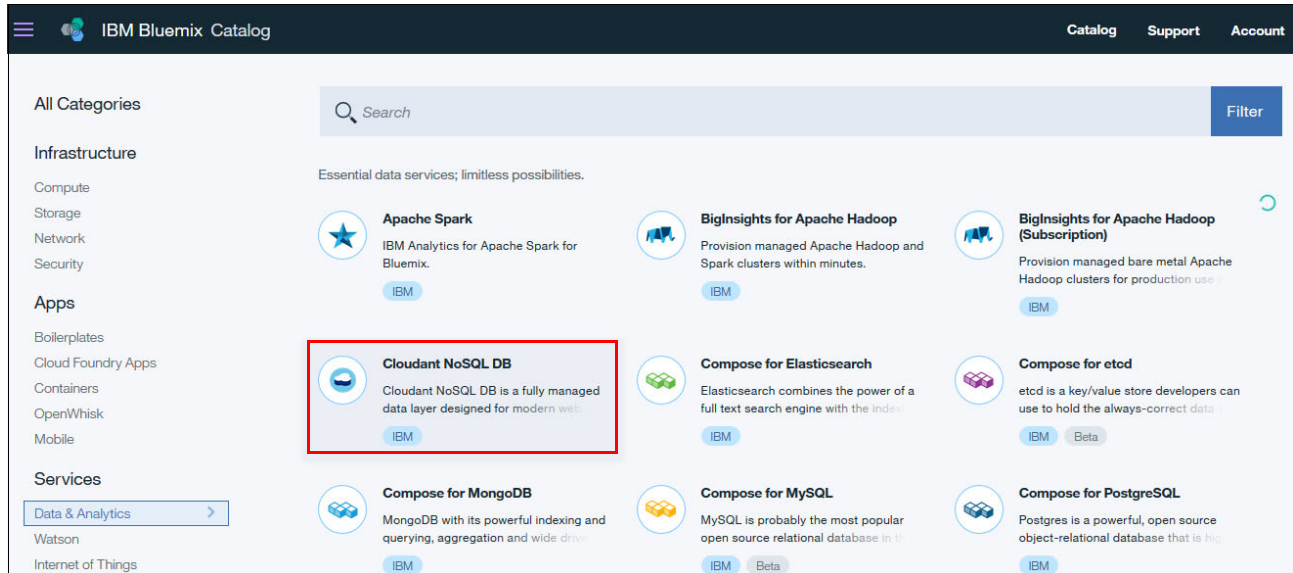


Figure 4-15 Cloudant noSQL service on Bluemix

2. Click **Create** to create the service instance (Figure 4-16).

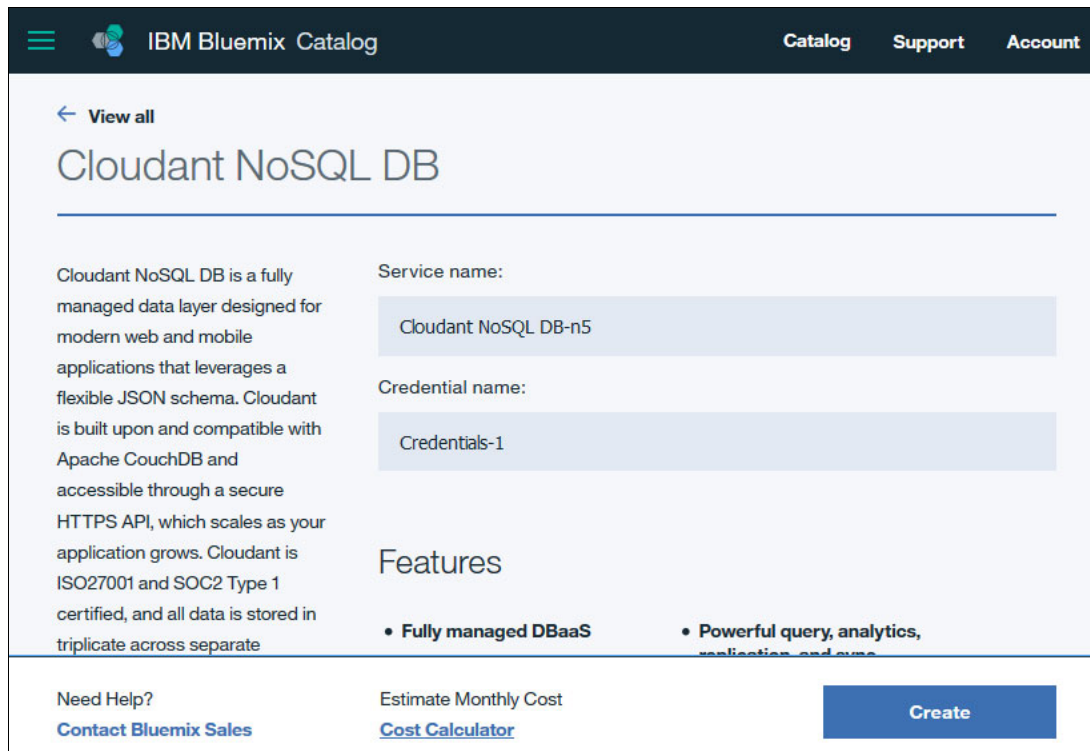


Figure 4-16 Create Cloudant noSQL DB service

The service name will be used to configure the application when it is deployed.

Important: Take note of the space in which you are creating the service. The application and the Cloudbant noSQL DB service must be deployed in the same space.

Create a Cloudbant noSQL DB service from command line

To create the service, follow these steps:

1. Download and install the [Cloud Foundry software](#) on your workstation.
2. Open a command prompt.
3. Run **cf login** and supply the email and password for your Bluemix account as shown in Example 4-2.

Example 4-2 Run cf login

```
cf login
  API endpoint: https://api.ng.bluemix.net

Email> <PUT_YOUR_BLUEMIX_EMAIL_ACCOUNT>

Password> <PUT_YOUR_PASSWORD_ACCOUNT>

Authenticating...
OK
Targeted org <YOUR_ORGANIZATION>
```

4. Select a Bluemix space on which to host the service as shown in Example 4-3.

Example 4-3 Select a space

```
Select a space (or press enter to skip):
1. dev
2. qa
3. Prod
Space> 1
Targeted space dev
API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: <YOUR_BLUEMIX_EMAIL_ACCOUNT>
Org: <YOUR_ORGANIZATION>
Space: dev
```

5. Run the following command to create a Cloudbant noSQL DB service instance (Example 4-4 on page 59):

```
cf create-service <service> <service_plan> <service_instance>
```

The command has these values:

cf create-service	The Cloud Foundry command to create a service instance
<service>	The name of the service you want to create an instance of; Cloudbant noSQL DB in this case.
<service_plan>	The name of the plan, in this example the plan name is Lite.
<service_instance>	The name you provide for your service instance. You use this name to refer to your service instance in other commands when you configure and deploy the News Classifier application. If your service instance includes spaces, surround the service instance name with double or single quotation marks depending on the operating system where you run the command prompt. In this example the service instance name is News Classifier Feedback.

Example 4-4 The cf create-service command

```
cf create-service CloudantNoSQLDB Lite "News Classifier Feedback"
Creating service instance News Classifier Feedback in org
<YOUR_ORGANIZATION>/ space dev as <YOUR_BUEMIX_EMAIL_ACCOUNT>...
OK
```

6. List the service information by running the `cf service <service_name>` command to confirm that it was created successfully (Example 4-5).

Example 4-5 Confirm successful creation

```
cf service " News Classifier Feedback"

Service instance: News Classifier Feedback
Service: cloudantNoSQLDB
Bound apps:
Tags:
Plan: Lite
Description: Cloudant NoSQL DB is a fully managed data layer designed for
modern web and mobile applications that leverages a flexible JSON schema.
Cloudant is built upon and compatible with Apache CouchDB and accessible
through a secure HTTPS API, which scales as your application grows. Cloudant is
ISO27001 and SOC2 Type 1 certified, and all data is stored in triplicate across
separate physical nodes in a cluster for HA/DR within a data center.
Documentation url:
https://console.ng.bluemix.net/docs/#services/Cloudant/index.html#Cloudant
Dashboard:
https://cloudantbroker.ng.bluemix.net/dashboard/9e763bb2-c702-4bb6-8547-f30b34c
25b87

Last Operation
Status: create succeeded
Message:
Started: 2017-02-16T20:20:04Z
Updated:
```

4.4.4 Preparing training data

When preparing training data, an important place to start is by choosing a good data set. The features of a good data set are explained in this section.

The raw data in some situations is already in the required comma-separated value (CSV) file format, but in other situations you can find data in other formats. In this case, converting the source format into CSV format needs some data preparation. Even if the file is a CSV file, it might need some data preparation to be ready to use as input to the classifier. Those steps are explained in this section.

The strategy here is to create two data sets:

- ▶ *Training data* to train the classifier.
- ▶ *Test data* to test the classifier. The test data set will be used in 4.4.7, “Evaluating results and updating training data” on page 73.

Figure 4-17 on page 60 shows the general activities to prepare the training and test data sets.

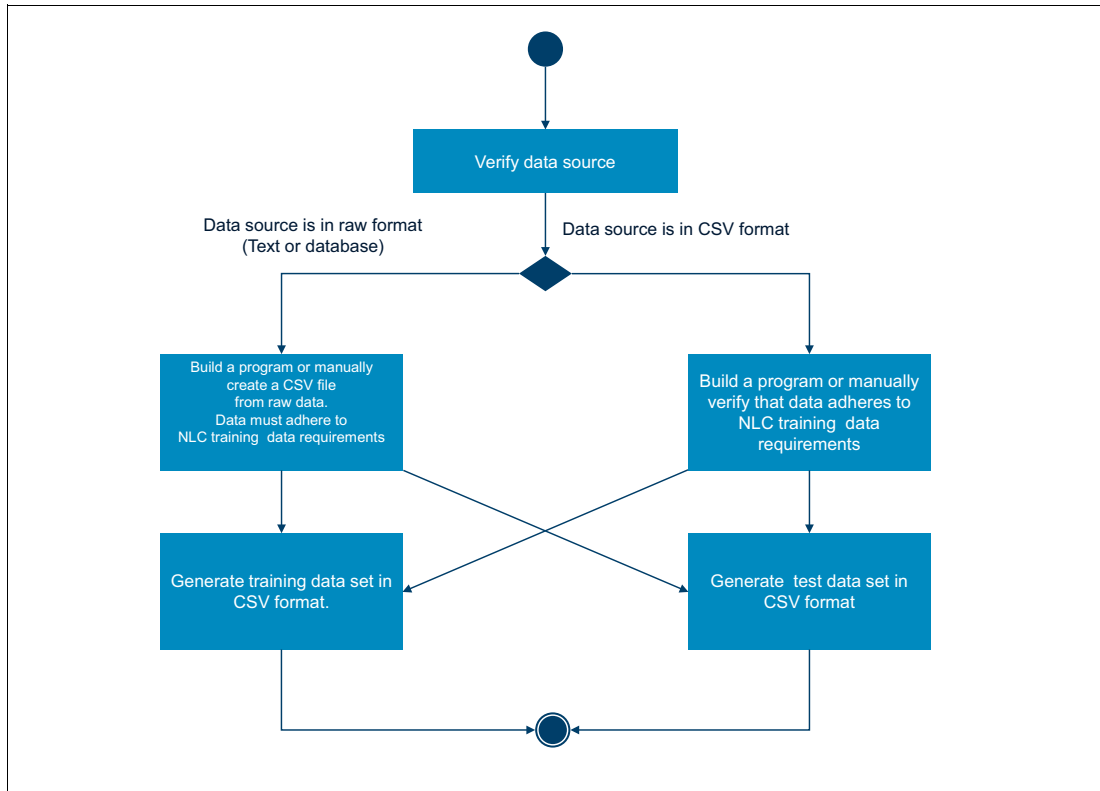


Figure 4-17 Activities for preparing training and test data

Figure 4-18 shows a snapshot of the training data used in the News Classification use case. Column “A” includes a list of news text. Column “B” has one of five classifications: business, entertainment, politics, technology, and sports.

A	B
The market for artificial intelligence (AI) technologies is flourishing. Beyond the hype and the heightened media attention the numerous startups and the internet giants racing to acquire them there is a significant increase in investment and adoption by enterprises. A Narrative Science survey found last year that 38% of enterprises are already using AI growing to 62% by 2018. Forrester Research predicted a greater than 300% increase in investment in artificial intelligence in 2017 compared with 2016. IDC estimated that the AI market will grow from \$8 billion in 2016 to more than \$47 billion in 2020.	technology
economic growth slowed sharply in the fourth quarter as a plunge in shipments of soybeans weighed on exports but steady consumer spending and rising business investment pointed to sustained strength in domestic demand.	business
Half of Brazil’s population cannot prove full legal ownership of their homes depriving authorities in the recession-hit country of a major source of taxes and deterring local investment a senior government official said. An estimated 100 million people lack property rights a senior Ministry of Cities official told the Thomson Reuters Foundation highlighting the need for Brazil to bring its housing sector into the formal economy.	Politics
Fluminense Football Club known simply as Fluminense is a Brazilian club best known for its football team that plays in the Brazilian Championship A series	sports
He supposedly died at the end of the Fox drama’s fourth season. But now Wentworth Miller is back as the gritty Michael Scofield in the action-packed Prison Break season five trailer.	entertainment

Figure 4-18 Training data for news classification

For this use case two CSV files were built manually, one for training the classifier and one for testing the classifier. To obtain the CSV files for this use case, go to:

<https://github.com/snippet-java/redbooks-nlc-201-news-java-student.git>

Find the following files in directory `/redbooks-n1c-201-news-java-student/resources/`:

- ▶ `news-train.csv`

Training set to be used for bootstrap classification when creating the classifier in 4.4.5, “Creating and training the classifier” on page 63.

- ▶ `news-test.csv`

Test set to be used to evaluate the quality of the classification results in 4.4.7, “Evaluating results and updating training data” on page 73.

To build a training data set with an acceptable syntax and good quality, consider these guidelines:

- ▶ The training and test sets are prepared for UTF-8 format.
- ▶ If you have a comma (,) in the text, insert quotes around the text.
- ▶ The maximum length of a text value is 1024 characters.
- ▶ The training and test data have at least five records (rows) and no more than 15,000 records.
- ▶ Limit the length of input text to fewer than 60 words.
- ▶ Limit the number of classes to several hundred classes.

For more information about preparing the training data, see [Using your own data to train the Natural Language Classifier](#).

If you want to prepare your own data in CSV format, it must have two columns, the first one with the text to classify and the second column with classification types (business, entertainment, politics, technology, sports). You can validate the data by using the **PrepareData** program, which is described later.

Data can be prepared in two ways:

- ▶ Prepare training data on Eclipse
- ▶ Prepare training data on the command line

Prepare training data on Eclipse

Complete these steps:

1. On Eclipse (see project in Figure 4-14 on page 55), right-click the **PrepareData.class** and select **Run As** → **Run Configurations** (Figure 4-19).

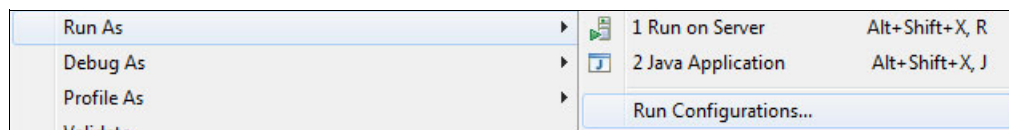


Figure 4-19 Run as Java main program in Eclipse

2. On the Main tab, confirm that `com.ibm.itso.ed600r01.n1c.news.PrepareData` is selected (Figure 4-20 on page 62).

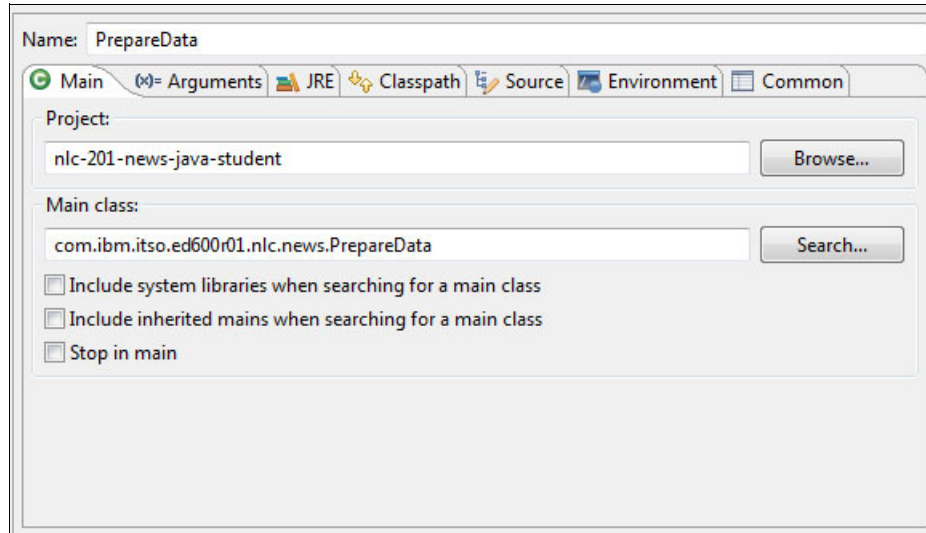


Figure 4-20 PrepareData class selected class to run

3. In the Arguments tab, for Program arguments, enter the name of the CSV file to be prepared and click **Run** (Figure 4-21).

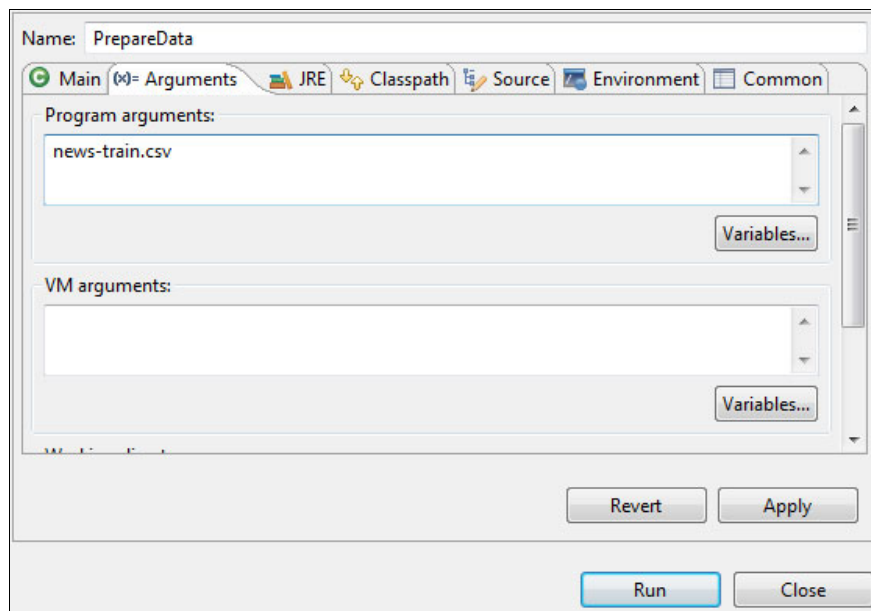


Figure 4-21 Arguments to call PrepareData class

This program checks text constraints such as column size, special characters such as `\n` `\r` and others. The output is shown in Figure 4-22.

```
<terminated> PrepareData [Java Application] C:\Program Files (x86)\eclipseDevelopmentPackage-6.3.20\ibm_sdk80\bin\javaw.exe (
Preparing File news-train.csv to be ready for Natural Language Classifier input
Fixing 1024 chars for text length, handling special chars like line feed and format to UTF-8 format
Data prepared !
```

Figure 4-22 PrepareData output on Eclipse

Prepare training data on the command line

Complete these steps:

1. Open a command prompt on your computer.
2. Set up `java.exe` in your path.
3. Change to the resources directory of the project that was prepared in “Clone the sample Git project by using the Git command line” on page 55:

```
cd redbooks-nlc-201-news-java-student/resources
```

4. Run the following scripts, which, in turn, run Java commands:

- For Windows: `PrepareData.bat <csv file path>`
- For Linux: `./PrepareData.sh <csv file path>`

The output is shown in Example 4-6.

Example 4-6 Output

```
./PrepareData.sh news-train.csv  
Preparing File news-train.csv to be ready for Natural Language Classifier input  
Fixing 1024 chars for text length, handling special chars like line feed and form  
at to UTF-8 format  
Data prepared!
```

Note: The CSV file path can be the file name only if it is in the resources project folder.

4.4.5 Creating and training the classifier

Note: You must create an Natural Language Classifier service instance in Bluemix as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11 before performing the steps in this section.

The `news-train.csv` (shown in the examples in 4.4.4, “Preparing training data” on page 59) will be used to create and train the classifier. This step is called *bootstrap classification*. The bootstrap classification (Figure 4-23), can be validated by subject matter experts (SMEs) for accuracy using other data, called test data, and if necessary correct classification problems.

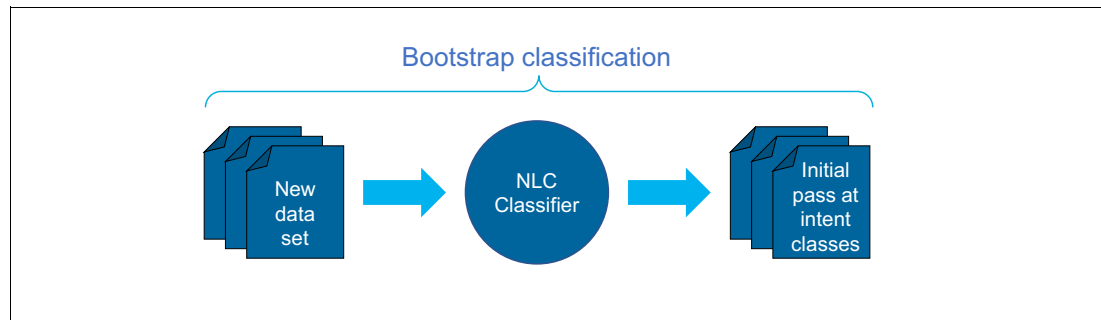


Figure 4-23 Bootstrap classification

This step is highly sensitive to good training data provided from the prepare data step and can be continuously improved depending on the target accuracy level, using other data sets.

The program to create and train the classifier is simple. Figure 4-24 shows the activities to create and train the classifier.

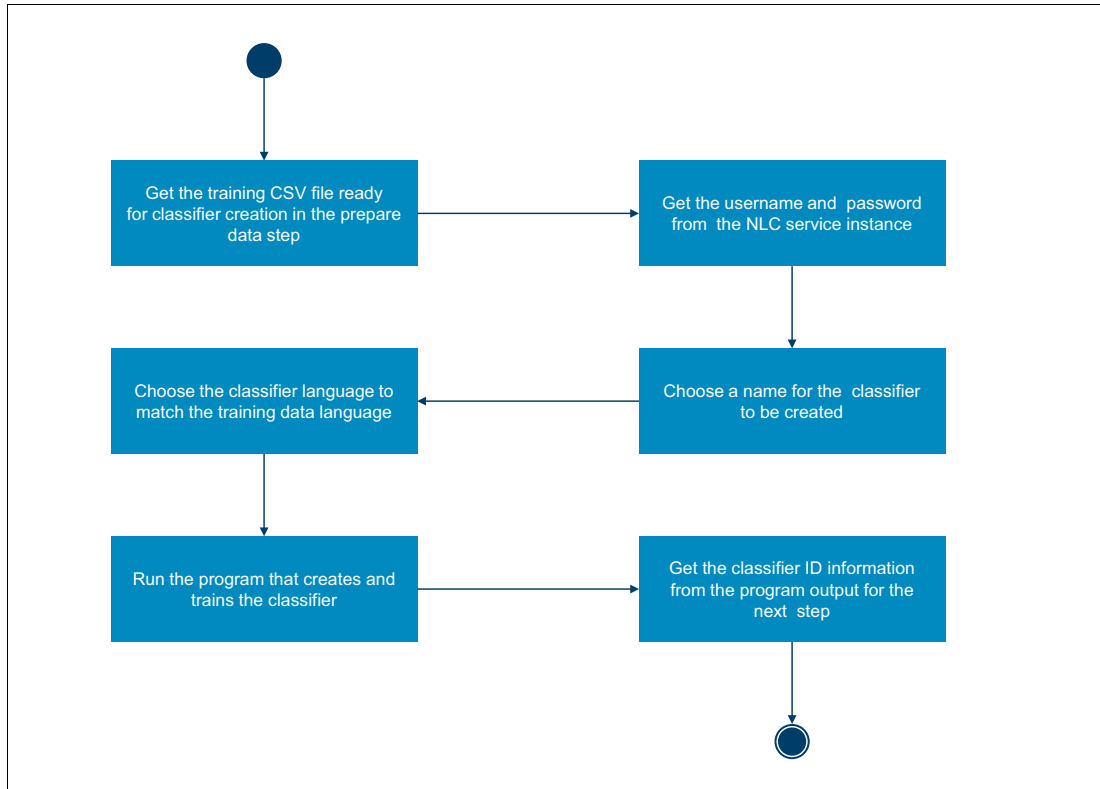


Figure 4-24 Create and train activities

The figure shows the following activities:

1. The CSV file created from the training data set in 4.4.4, “Preparing training data” on page 59 is used as input to create and train the classifier.
2. The service credentials (username and password) that you obtained when you created the Natural Language Classifier service instance are needed when you create the classifier.
3. Choose a name for the classifier.
4. The classifier language must match the language that is used to train the classifier.
5. Run the program to create the classifier specifying the information listed in the previous steps.
6. After the program runs successfully, get the classifier ID which will be used later.

For more information about [creating a classifier](#), see the Watson Developer Cloud website.

For the use case in this chapter, a Java program is provided to create and train the classifier. The Java class is `CreateAndTrain.class`. The next steps describe running it.

The program can be used in one of the following two ways to create and train the classifier:

- ▶ Create and train the classifier on Eclipse
- ▶ Create and train the classifier on the command line

Create and train the classifier on Eclipse

Complete the following steps:

1. On Eclipse, right-click the `CreateAndTrain.class` class and select **Run As** → **Run Configurations** (Figure 4-19 on page 61).
2. On the Main tab, confirm that `com.ibm.itso.ed600r01.nlc.news.CreateAndTrain` is selected (Figure 4-25).

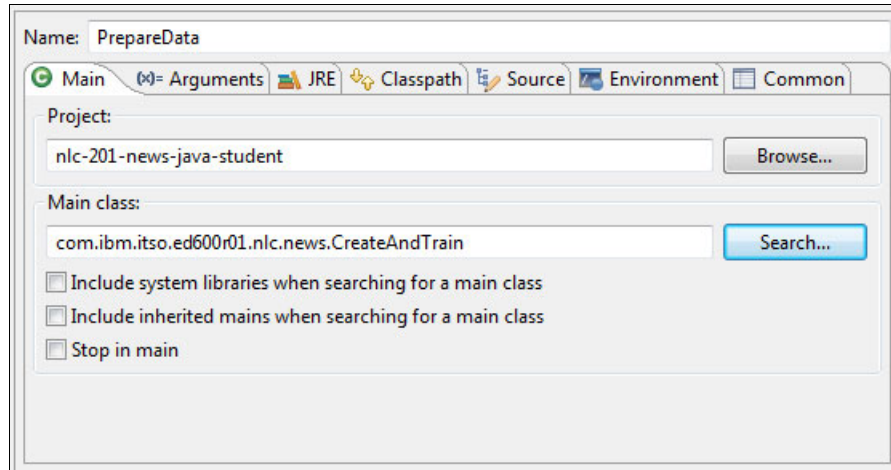


Figure 4-25 `CreateAndTrain` class is selected

3. In the Arguments tab, enter the following program arguments to create a classifier (see the example in Figure 4-26 on page 66):

```
<csv file path> <user> <password> <classifier_name> <language>
```

The arguments have the following meanings:

csv file path	The location in the local computer of the CSV file that will be uploaded as a training set, for example <code>news-train.csv</code> . The CSV file path can be just the file name if the file is currently in the resources project folder.
user	The username from the Natural Language Classifier service instance.
password	The password from the Natural Language Classifier service instance.
classifier_name	The name for the classifier.
language	The language used to train the classifier.

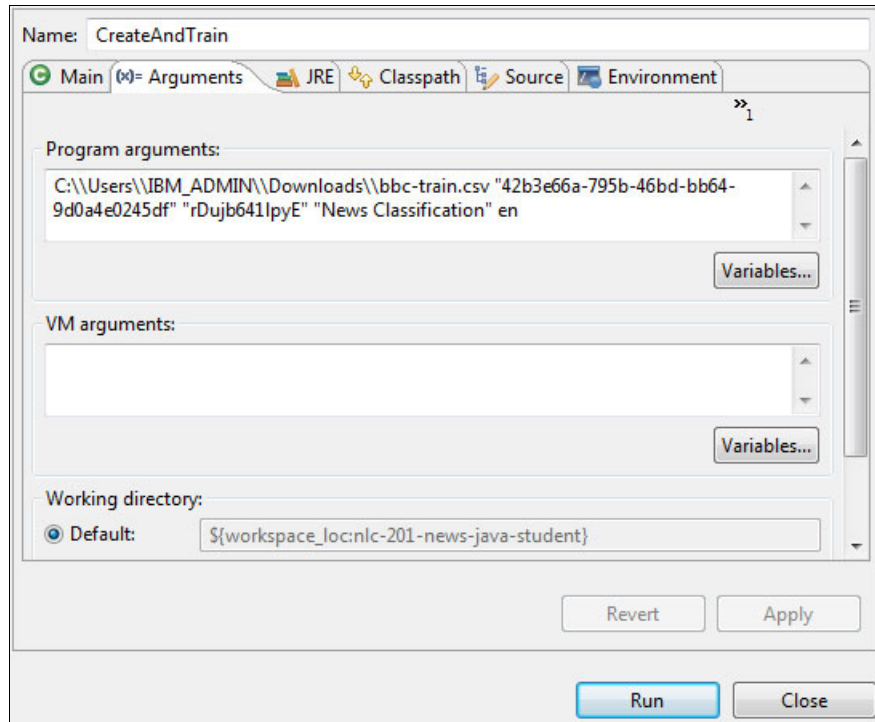


Figure 4-26 Define arguments to call create and train program in Eclipse

4. After specifying the arguments, click **Run**.

The important point is that user and password arguments match the ones that were obtained when the Natural Language service instance was created.

The program is executed as a Java application in Eclipse (Figure 4-27).

```
<terminated> News Classification - CreateAndTrain [Java Application] C:\Program Files (x86)\eclipseDevelopmentPackage\ibm_sdk80\bin\javaw.exe (23 de jan de 2017 17:24:26)
{
  "classifier_id": "ff18c7x157-nlc-5650",
  "language": "en",
  "name": "News Classification",
  "status": "Training",
  "created": "2017-01-23T19:24:47.221",
  "status_description": "The classifier instance is in its training phase, not yet ready to accept classify requests",
  "url": "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/ff18c7x157-nlc-5650"
}
```

Figure 4-27 Create and training program output

The last activity (shown in Figure 4-24 on page 64) is to get the classifier ID to be used in the next step described in 4.4.6, “Querying the trained classifier” on page 68.

The program output shows information about the classifier creation returned by the Watson API. It includes `classifier_id`, the language for which the classifier was created, the name of the classifier, the status of the classifier, and more.

Important values shown in Figure 4-27 on page 66 are:

- ▶ The `classifier_id` parameter: This is the ID of the new trained classifier. It will be used in 4.4.6, “Querying the trained classifier” on page 68 and in 4.4.8, “Deploying the application” on page 85).
- ▶ The `status` parameter: Shows that the classifier is not ready for queries. It will be available for the next step (query) only when status changes to `Available`.

Create and train the classifier on the command line

Complete the following steps:

1. Open a command prompt in your computer.
2. Set up `java.exe` in your path
3. Change to the `resources` directory of the project prepared in “Clone the sample Git project by using the Git command line” on page 55. Example:

```
cd redbooks-nlc-201-news-java-student/resources
```

4. Run the following scripts:

– For Windows:

```
CreateAndTrain.bat <csv file path> <user> <password> <classifier_name>  
<language>
```

– For Linux:

```
./CreateAndTrain.sh <csv file path> <user> <password> <classifier_name>  
<language>
```

The command has the following information:

csv file path	The location in the local computer of the CSV file that will be uploaded as a training set, for example <code>news-train.csv</code> . The CSV file path can be just the file name if it is currently in the <code>resources</code> project folder.
user	The user name obtained from Natural Language Classifier service instance.
password	The password obtained from the Natural Language Classifier service instance.
classifier_name	The name for the classifier.
language	The language used to train the classifier.

The output is similar to Example 4-7.

Example 4-7 Output

```
CreateAndTrain.bat news-train.csv 53bf6841-xx4c-4812-9fe5-fc25af43876f  
hJy62XXY7fot "Class News Simulator" en  
  
java -cp  
../target/redbooks-nlc-201-news-java-student.jar;../lib/opencsv-3.3.jar;../lib/  
java-sdk-3.  
5.3-jar-with-dependencies.jar com.ibm.itso.ed600r01.nlc.news.CreateAndTrain  
news  
-train.csv 53bf6841-b04c-4812-9fe5-fc25af43876f hJy62p0Y7fot "Class News  
Simula-tor" en  
{
```

```
"classifier_id": "f5bbbx174-nlc-3736",
"language": "en",
"name": "Class News Simulator",
"status": "Training",
"created": "2017-02-14T18:37:39.887",
"status_description": "The classifier instance is in its training phase, not yet ready to accept classify requests",
"url": "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/f5bbbx174-nlc-3736"
}
```

The program output shows information about the classifier creation returned by the Watson API. It includes `classifier_id`, the language for which the classifier was created, the name of the classifier, the status of the classifier, and more.

Important values shown in Example 4-7 on page 67 are:

- ▶ The `classifier_id` parameter: This is the ID of the new trained classifier. It is used in 4.4.6, “Querying the trained classifier” on page 68 and in 4.4.8, “Deploying the application” on page 85).
- ▶ The `status` parameter: Shows that the classifier is not ready for queries. It will be available for the next step (query) only when status changes to `Available`.

4.4.6 Querying the trained classifier

After the classifier is trained, you can query it. In this step you use the Watson API to send text to the trained classifier. The service returns the top matching class and other possible matches with the associated confidence.

The flow of steps to query the classifier are shown Figure 4-28 on page 69.

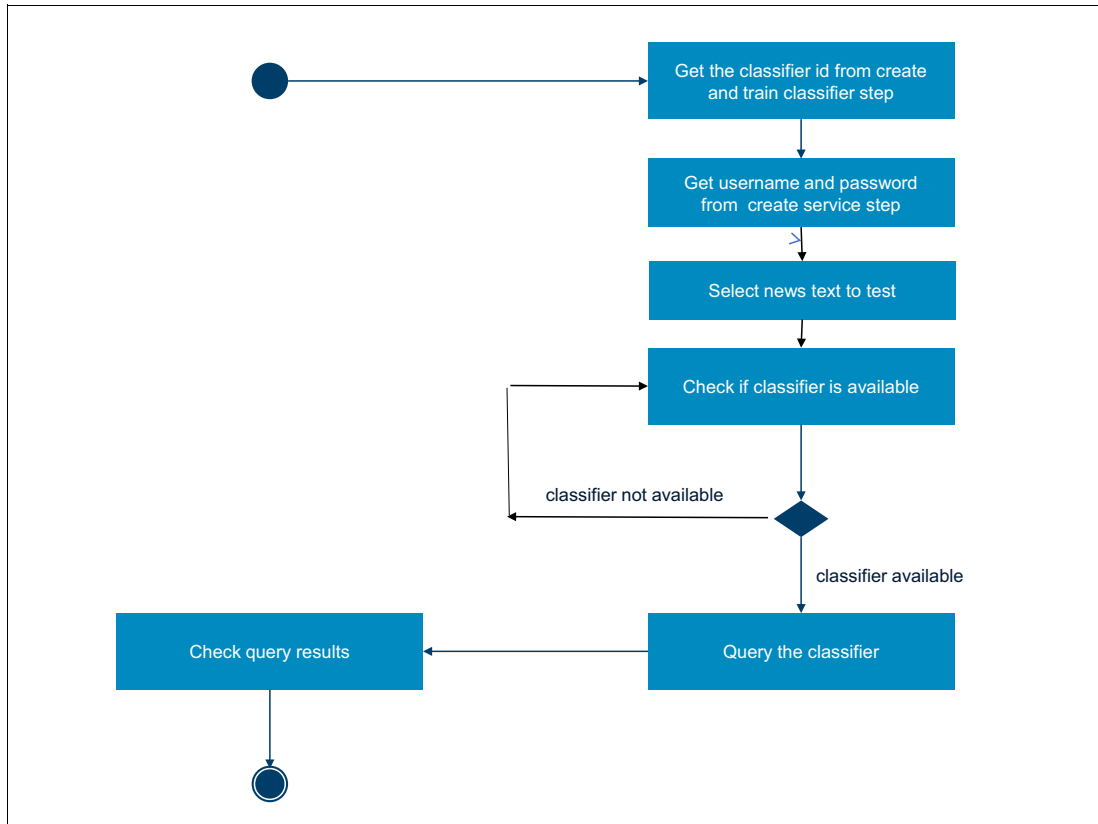


Figure 4-28 Query the classifier flow

The figure shows the following steps implemented in the Java `Query.class` created for this use case to query the classifier:

1. Get the `classifier` ID from the create and train classifier step described in 4.4.5, “Creating and training the classifier” on page 63.
2. Get the username and password from the Natural Language Classifier service instance created as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11.
3. Select the news text message that will be used as input to query the classifier for classification.
4. Check the status of the classifier until it is `Available`.
5. Run the query. Input the parameters collected in the previous steps to a Java program to query the classifier.

The Java class created for this step in the News Classification use case is `Query.class`.

The program can be used in one of the following two ways to query the trained classifier:

- ▶ Query the trained classifier with Eclipse
- ▶ Query the trained classifier with the command line

Query the trained classifier with Eclipse

Complete the following steps to run the Java Main program on Eclipse:

1. Right-click the `Query.class` class and select **Run As** → **Run Configurations**.
2. On the Main tab confirm that `com.ibm.itso.ed600r01.news.Query` is selected (Figure 4-29).

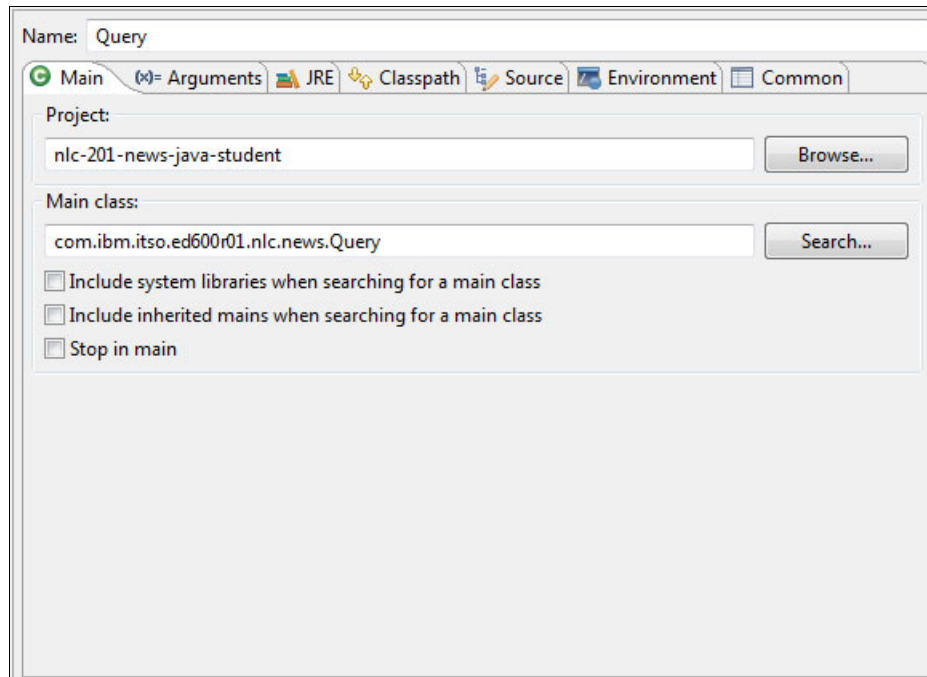


Figure 4-29 Query class execution on Eclipse

3. In the Arguments tab, enter the following parameters (see the example in Figure 4-30 on page 71):

```
<classifier_id> <user> <password> <query_text>
```

The arguments have the following meanings:

- | | |
|----------------------|---|
| classifier_id | The classifier ID obtained in 4.4.5, “Creating and training the classifier” on page 63. |
| user | The username from the Natural Language Classifier service instance credentials obtained when you created the service instance as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11. |
| password | The password from the Natural Language Classifier service instance credentials obtained when you created the service instance. |
| query_text | The news text to classify. If the text has more than one word, enclose the text in double quotation marks. For example, use the following query text as a parameter: “He supposedly died at the end of the drama’s fourth season. But now Wentworth Miller is back as the gritty Michael Scofield in the action-packed Prison Break season five trailer.” |

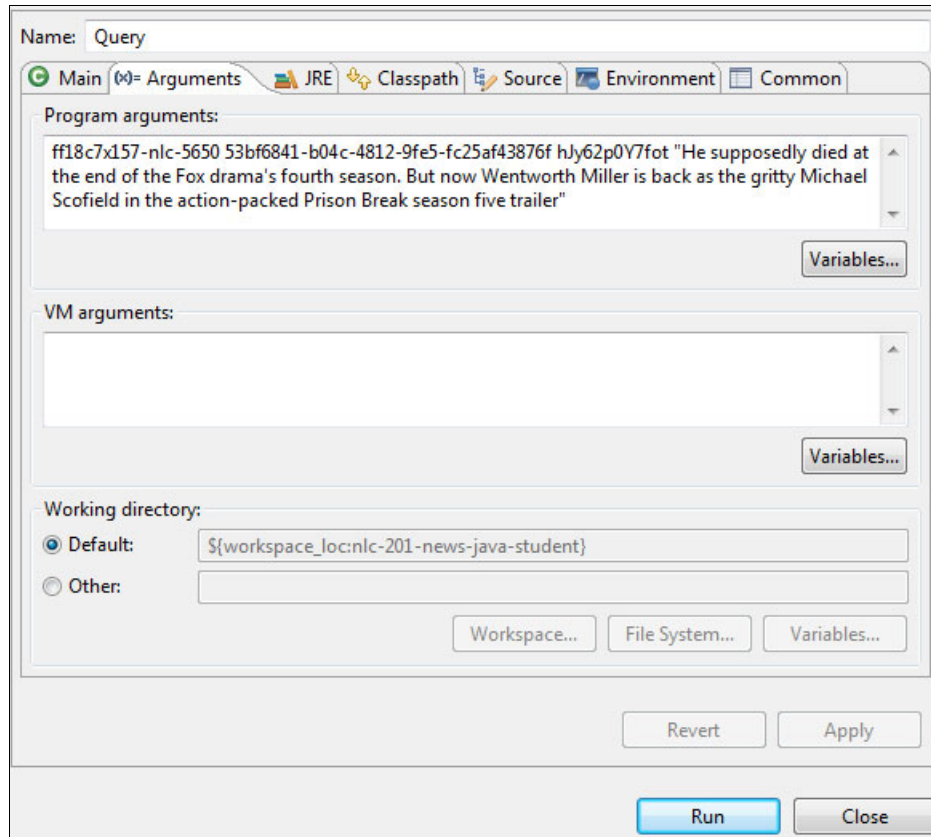


Figure 4-30 Query program execution on Eclipse

4. After specifying the arguments, click **Run**.
5. Check the results. Figure 4-31 shows an example of the API response when running a query in this use case.

```
<terminated> Query [Java Application] C:\Program Files (x86)\eclipseDevelopmentPackage-6.3.20\ibm_sdk80\bin\javaw.exe (9 de fev de 2017 16:19:23)
Status of Classifier ff18c7x157-nlc-5650 - AVAILABLE
Results for query He supposedly died at the end of the Fox drama's fourth season. But now Wentworth Miller is back
{
  "classes": [
    {
      "confidence": 0.8575665436245274,
      "class_name": "entertainment"
    },
    {
      "confidence": 0.11310460026913388,
      "class_name": "sport"
    },
    {
      "confidence": 0.015873119271676537,
      "class_name": "politics"
    },
    {
      "confidence": 0.007139259284434174,
      "class_name": "business"
    },
    {
      "confidence": 0.006316477550227919,
      "class_name": "technology"
    }
  ],
  "classifier_id": "ff18c7x157-nlc-5650",
  "text": "He supposedly died at the end of the Fox drama's fourth season. But now Wentworth Miller is back as the gritty Michael Scofield in the action-packed Prison Break season five trailer",
  "top_class": "entertainment",
  "url": "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/ff18c7x157-nlc-5650"
}
```

Figure 4-31 Query results output

Query the trained classifier with the command line

Complete these steps:

1. Open a command prompt in your computer.
2. Set up `java.exe` in your path.
3. Change to the resources directory of the project prepared in 4.4.1, "Downloading the project from Git" on page 51:

```
cd redbooks-nlc-201-news-java-student/resources
```

4. Run the following scripts:

– For Windows:

```
Query.bat <classifier_id> <user> <password> <query_text>
```

– For Linux:

```
./Query.sh <classifier_id> <user> <password> <query_text>
```

The command has the following values:

classifier_id	The classifier ID obtained in 4.4.5, "Creating and training the classifier" on page 63.
user	The username from the Natural Language Classifier service instance credentials obtained when you created the service instance as described in Chapter 2, "Creating a Natural Language Classifier service in Bluemix" on page 11.
password	The password from the Natural Language Classifier service instance credentials obtained when you created the service instance.
query_text	The news text to classify. If the text has more than one word, enclose the text in double quotation marks.

The output is similar to Example 4-8.

Example 4-8 Output

```
$ ./Query.sh f5b432x172-nlc-3699 53bf6841-b04c-4812-9fe5-fc25af43876f hJy62p0Y7fot
"economic growth slowed sharply in the fourth quarter as a plunge in shipments of
soybeans weighed on exports, but steady consumer spending and rising business
investment pointed to sustained strength in domestic demand"
Status of Classifier f5b432x172-nlc-3699 - AVAILABLE
Results for query economic growth slowed sharply in the fourth quarter as a plunge
in shipments of soybeans weighed on exports, but steady consumer spending and
rising business investment pointed to sustained strength in domestic demand
{
  "classes": [
    {
      "confidence": 0.9906648553297829,
      "class_name": "business"
    },
    {
      "confidence": 0.004622361487155504,
      "class_name": "politics"
    },
    {
      "confidence": 0.0018452044719471966,
      "class_name": "technology"
    }
  ],
}
```

```

    {
      "confidence": 0.0016647389251637564,
      "class_name": "sports"
    },
    {
      "confidence": 0.0012028397859506108,
      "class_name": "entertainment"
    }
  ],
  "classifier_id": "f5b432x172-nlc-3699",
  "text": "economic growth slowed sharply in the fourth quarter as a plunge in
ship-ments of soybeans weighed on exports, but steady consumer spending and rising
business investment pointed to sustained strength in domestic demand",
  "top_class": "business",
  "url":
  "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers
/f5b432x172-nlc-3699"
}

```

Query classifier response

The following information in the query response is important:

- ▶ `status` shows the classifier status. When the classifier is ready to accept requests, the status is changes from `Training` to `Available`. Before running the query you should check the status of the classifier to confirm that it is available to accept queries. If the status is *not* `Available`, the program ends.
- ▶ `classes` is an array that contains the list of defined class labels and the confidence for each. This array represents the query results in JSON format. The classes in the array are ordered in a descending order of confidence, that is, the class label with the highest confidence is always the first element in the `classes` array.
- ▶ Other parameters in the query response are:
 - `classifier_id`: Classifier ID
 - `text`: Shows the input text in the query request
 - `top_class`: Class with the highest confidence
 - `url` to reach the classifier.

4.4.7 Evaluating results and updating training data

The objective of this step in the process is to improve the results returned by the classifier. It is a critical step to ensure that the classifier will perform successfully in a production environment.

These are the main approaches you can follow for evaluating results:

- ▶ Evaluating results with the Natural Language Classifier toolkit interactive wizard
- ▶ Evaluating results programmatically
 - Running the program with Eclipse
 - Running the program with the command line

Evaluate results with the Natural Language Classifier toolkit interactive wizard

The first approach to evaluation is validation by SMEs and adjusting the classifier if accuracy is not aligned with the desired outcome. You can also include customer feedback providing a way for users to input their feedback about the classification results.

Figure 4-32 provides an overview of the process.

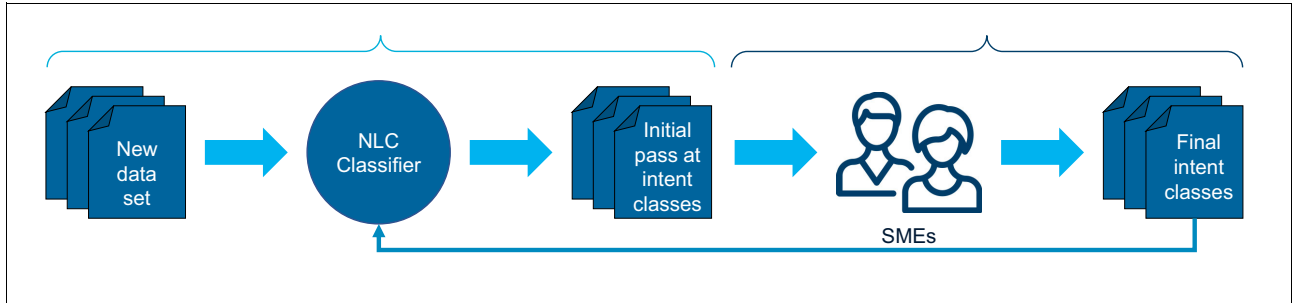


Figure 4-32 Manual validation of classifier

Manual validation activities are shown in more detail in Figure 4-33.

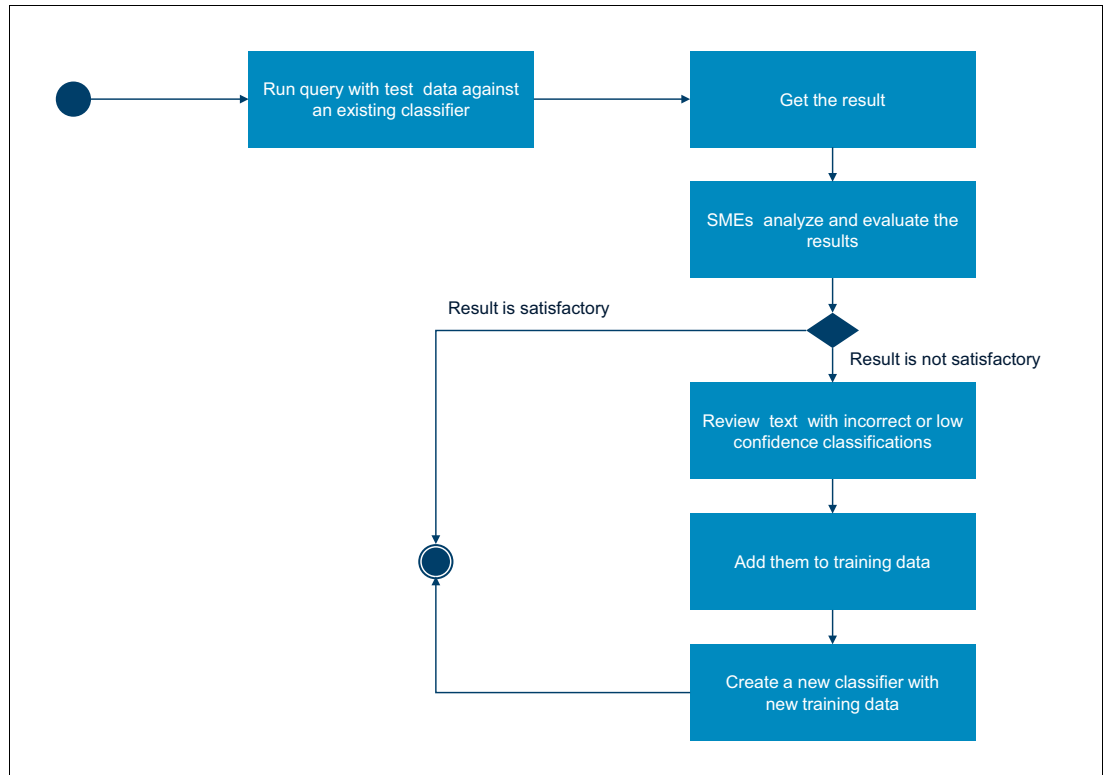


Figure 4-33 Manual results evaluation activities

The figure shows the following activities:

1. Run query with a test data set to test an existing classifier. This step can be performed by a system administrator or SME.
2. Get the query results. The results can be collected by a test program, for example by running a single query in the classifier or a set of queries from a test data source and

collecting responses. Another approach is to collect feedback from customers in a production web application and save the feedback to a database.

3. SMEs analyze and evaluate the query results. For example, they determine if the top class is correct or if the level of accuracy is satisfactory based on the required threshold, for example 95% of precision.
4. If the results are satisfactory, the evaluation process ends.
5. If the results are not satisfactory, the SMEs make changes to adjust the training data for the text that was incorrectly classify.
6. Create a new training data set with the new data and create a new classifier with the new training data.

These steps can be executed using the Natural Language Classifier toolkit in Bluemix:

1. Log in to Bluemix and scroll down to the Services section.
2. Click the Natural Language Classifier service.
3. Click the **Manage** tab.
4. Click **Access the beta toolkit** (see Figure 4-34).

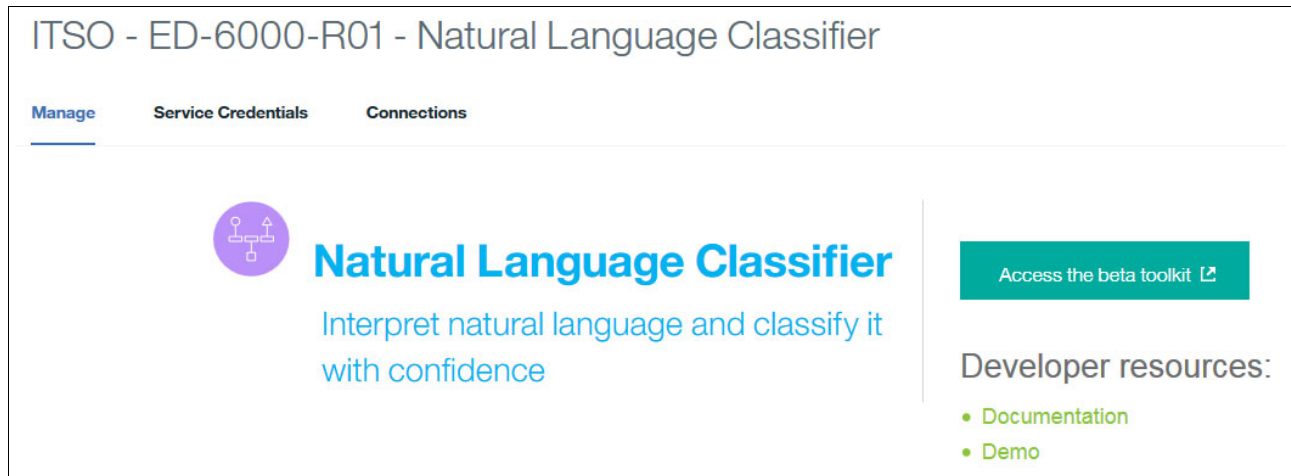


Figure 4-34 Accessing the Natural language Classifier toolkit

5. Select the **News Classification** classifier that was created in 4.4.5, “Creating and training the classifier” on page 63 and click the (next) arrow (Figure 4-35).

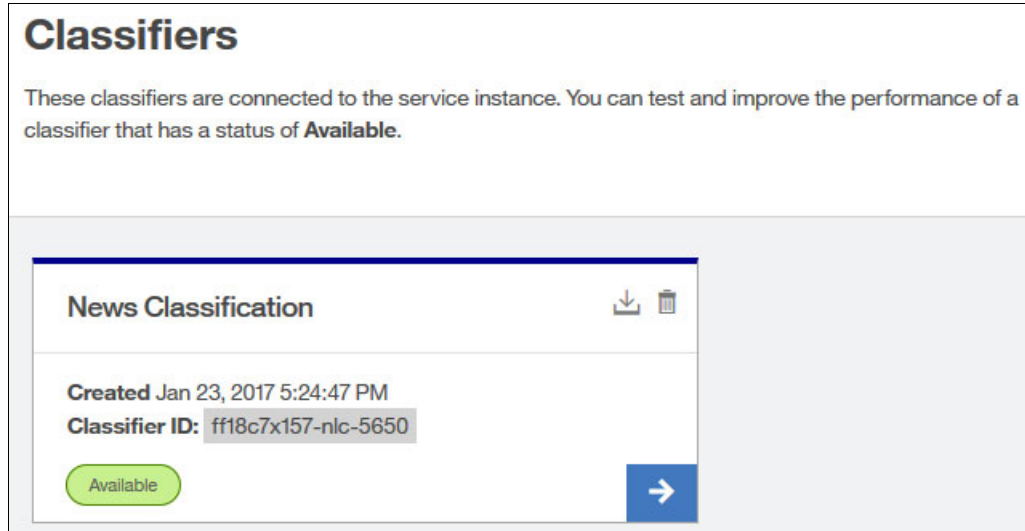


Figure 4-35 News Classifier selection for evaluation

6. On the next page, click **Use test data** to load a test CSV file, or enter text in the input field and click **Classify** (Figure 4-36).

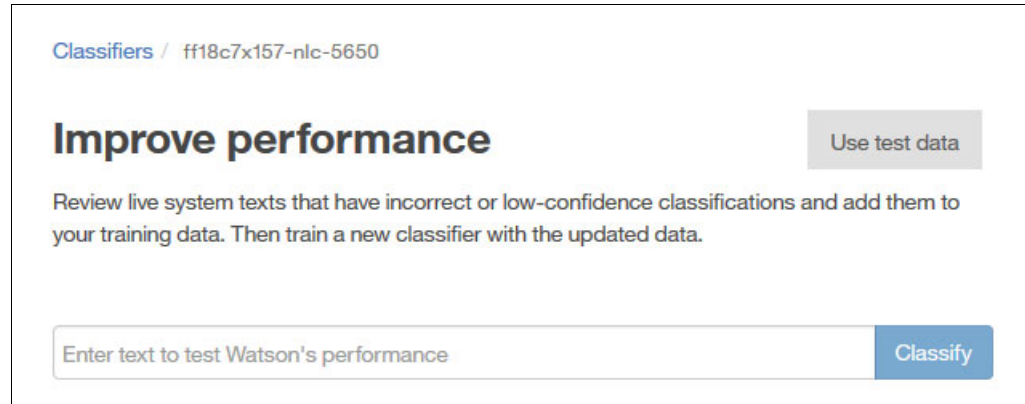


Figure 4-36 Options for input test data

In this example, click **Use test data** and select the `news-test-nlc-toolkit.csv` file in the resources folder of the sample project (Figure 4-37 on page 77).

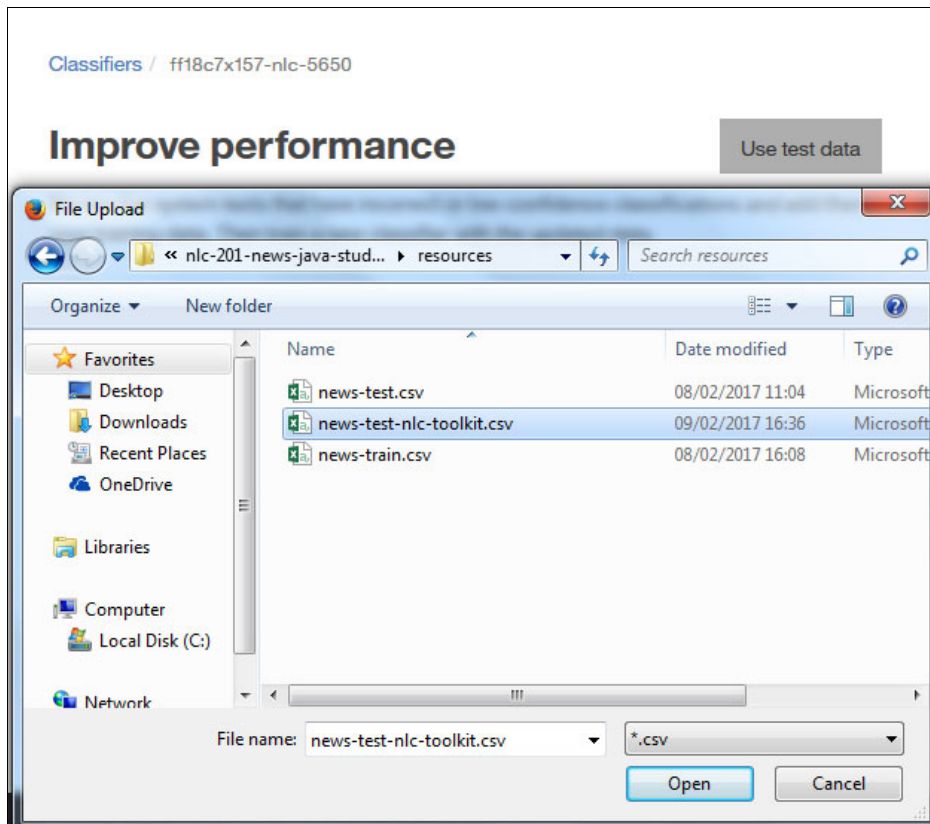


Figure 4-37 Test data selection

You receive two messages indicating that the test CSV file was loaded successfully (Figure 4-38).

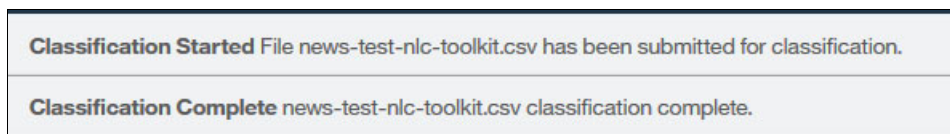


Figure 4-38 Test data loading completion

Also, you receive information about how the classifier segmented the test data into news categories. Figure 4-39 shows text that should have been classified as technology but was incorrectly classified business as the top class. In this case, mark this classification as incorrect.

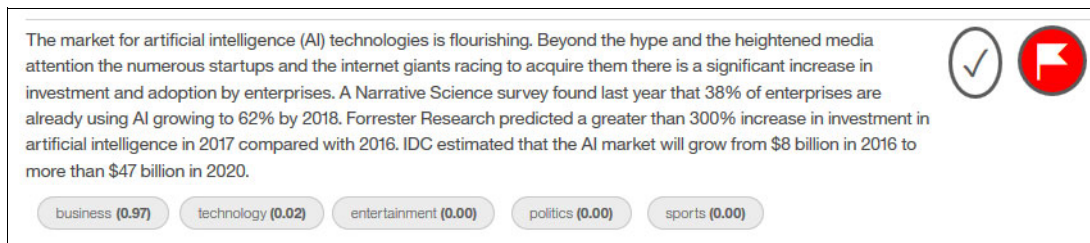


Figure 4-39 Handling incorrect classification

Figure 4-40 shows the correct classification defined by SMEs. The classifier classified the text correctly as sports type.

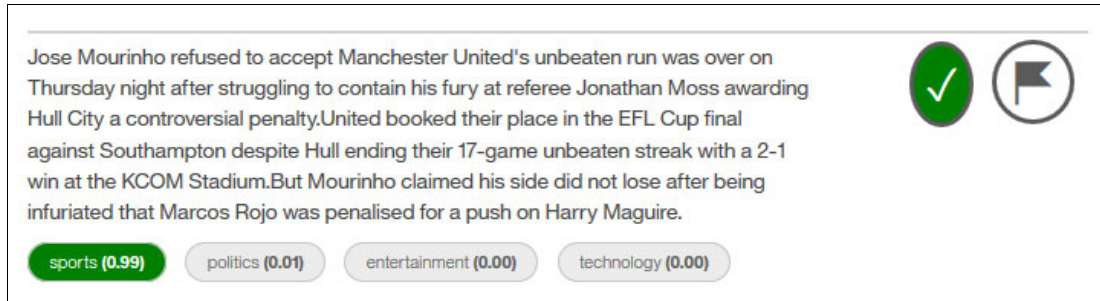


Figure 4-40 Handling correct classifications

7. SMEs evaluate each text classification from the test data set. Figure 4-41 shows the results.
8. After all text classification results are evaluated, click **Add to training data** to create a new training data set to improve the classifier performance. Note that a new classifier must be created with the new training data.

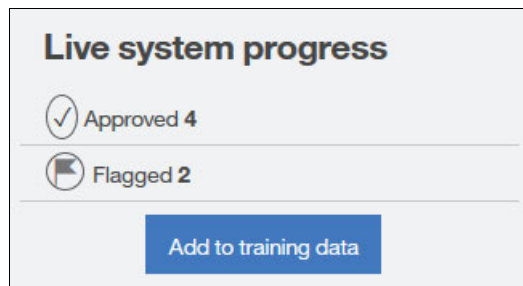


Figure 4-41 Evaluation results

- The training data window opens (Figure 4-42). Review the information. If text is marked by the SME as wrongly classified, the toolkit provides a suggestion for a class.

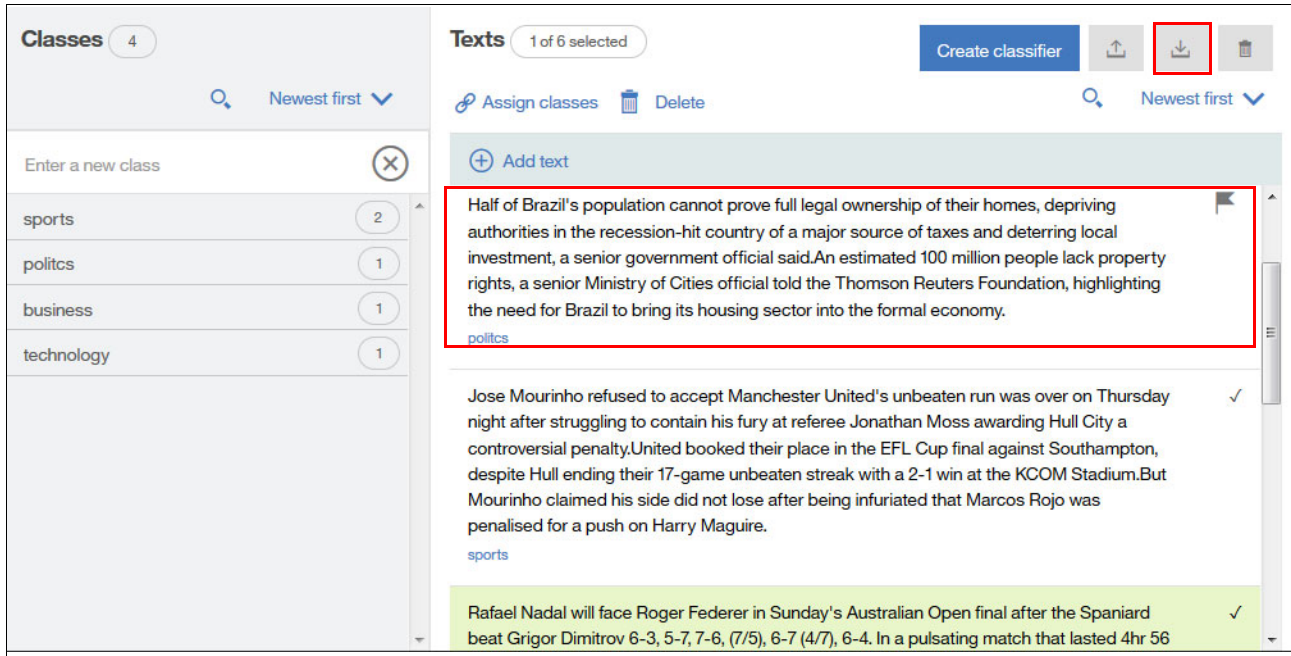


Figure 4-42 Training page

- Select incorrectly classified text and drag the correct class from the Classes section on the left to the Texts section on the right.

Figure 4-42 shows that the `politics` class was dragged to the text that was classified as `business` before.

- Download the corrected CSV file by clicking the download icon and add the data to the training data set.
- Create a new classifier with the new and improved training data set. Click **Create classifier** (Figure 4-42) to create the new classifier.

Note: The `classifier` ID of the new classifier is not the same as the classifier ID of the original classifier. The classifier ID must be updated in the programs that use it to access the new classifier.

This process is continuous until the classifier reaches a good value for accuracy, aligned with business needs, for example 80% correct classification.

Evaluate results programmatically

You can automate the process described in “Evaluate results with the Natural Language Classifier toolkit interactive wizard” on page 74 by creating a program that queries an existing classifier using a test data set. The test data set has the correct classes defined by the previous work of the SMEs. If accuracy results are not satisfactory, the test data is added to the training data set and used to train a new classifier. This approach is described in Figure 4-43 on page 80.

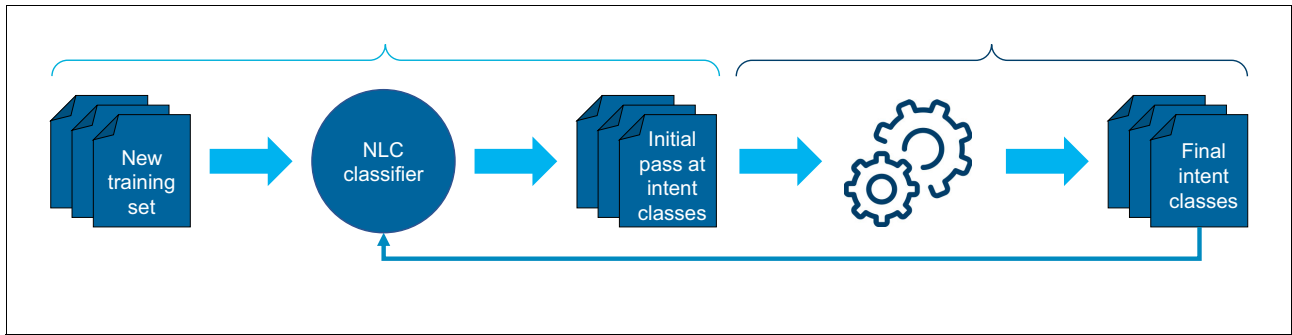


Figure 4-43 Automated validation of classifier

The activities are shown in Figure 4-44.

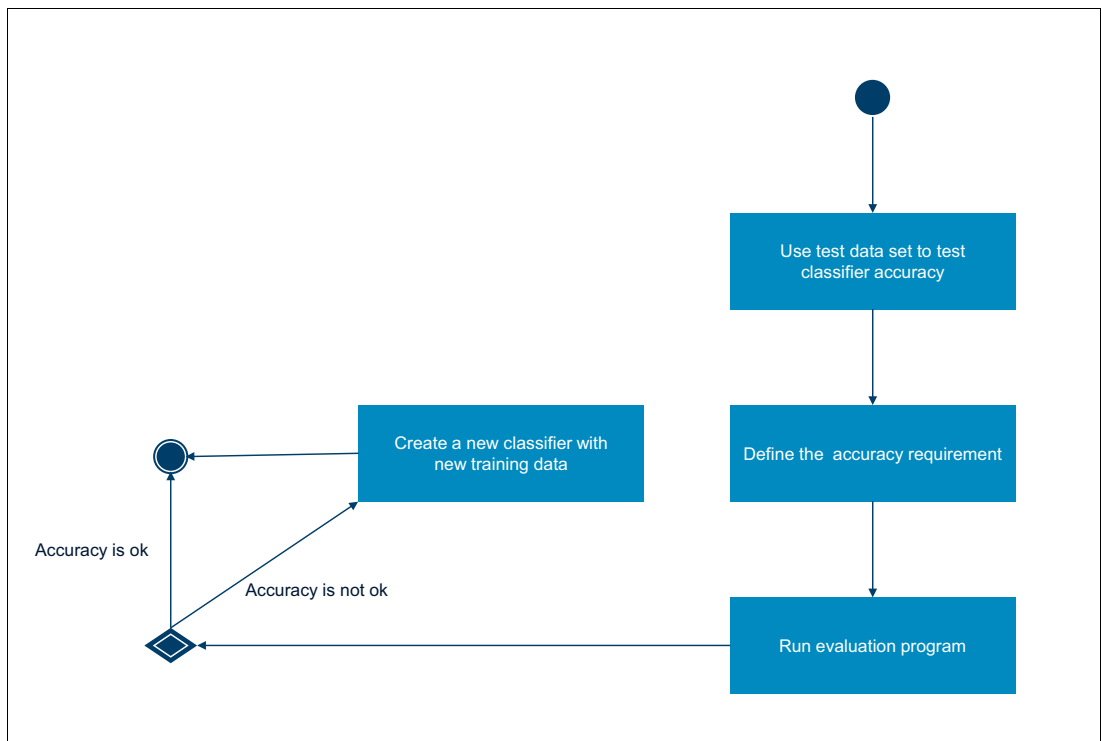


Figure 4-44 Evaluation automated

These are the activities:

1. A test data set was prepared and is available.
2. The SMEs defined the accuracy threshold, for example 70%, according to business requirements.
3. Test data set and accuracy requirement are input to the evaluation program.
4. The evaluation program queries the classifier for each line in the test data set.
5. The results are evaluated and the accuracy calculated.
6. If the accuracy is below the threshold, create a new training data set and create and train a new classifier.

Figure 4-45 shows a high-level flow of the Evaluate program.

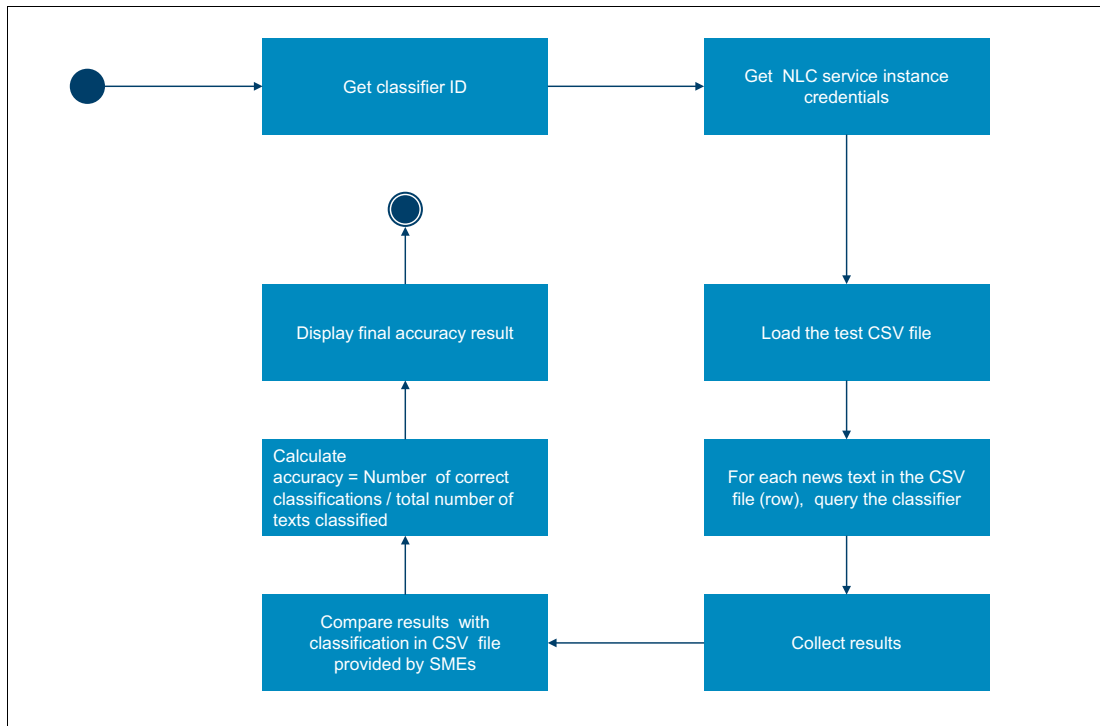


Figure 4-45 Evaluate program flow

The figure shows the following steps in the Evaluate program:

1. Get classifier ID.
2. Get Natural Language Classifier service credentials (username and password).
3. Load the test CSV file.
4. For each text (row) in the test CSV file query the classifier.
5. Collect the classification results, that is, save the results in an ArrayList structure.
6. Compare results. The classifier results for each line of text in the test CSV file is compared with the classification provided by SMEs for the same text. The number of correct classifications is computed.
7. Calculate the accuracy with the following formula:
$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{number of texts classified}}$$

Two main methods are defined in the Evaluate.class program:

► **batchClassify**

Loads the test CSV file and classifies each line of text. It returns an ArrayList; each element contains the classification by the classifier and by the SME for the same line of text. Figure 4-46 provides a flow diagram for this class.

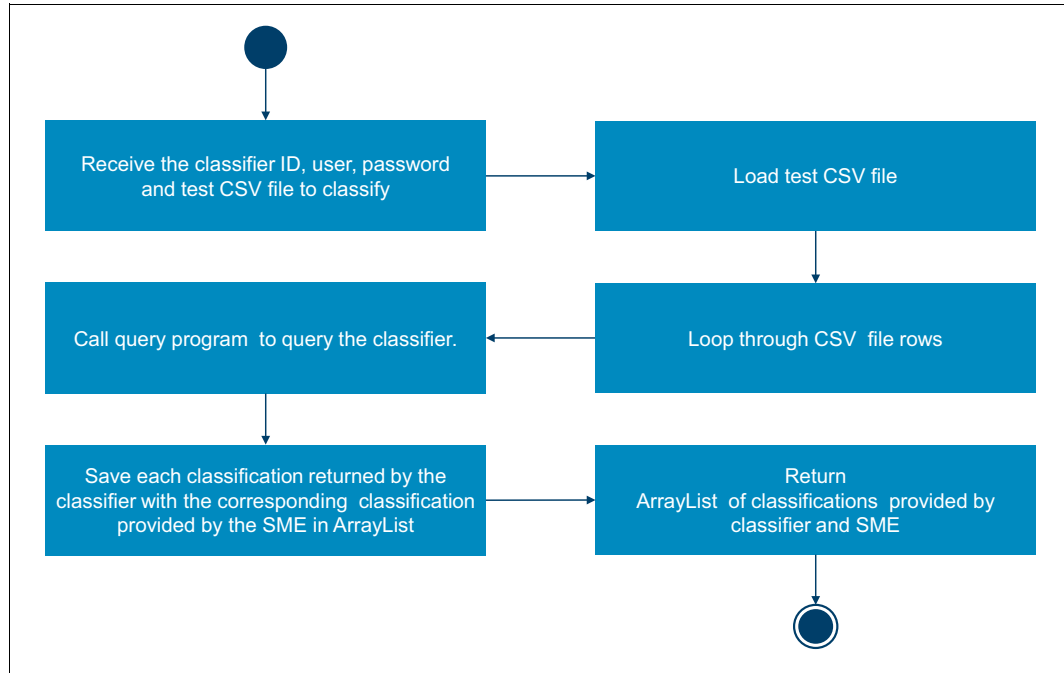


Figure 4-46 The batchClassify method of Evaluate class program

► **generateAccuracy**

This method receives the ArrayList from batchClassify, determines the correct and incorrect classifications and calculates the accuracy. Figure 4-47 provides a flow diagram for this class.

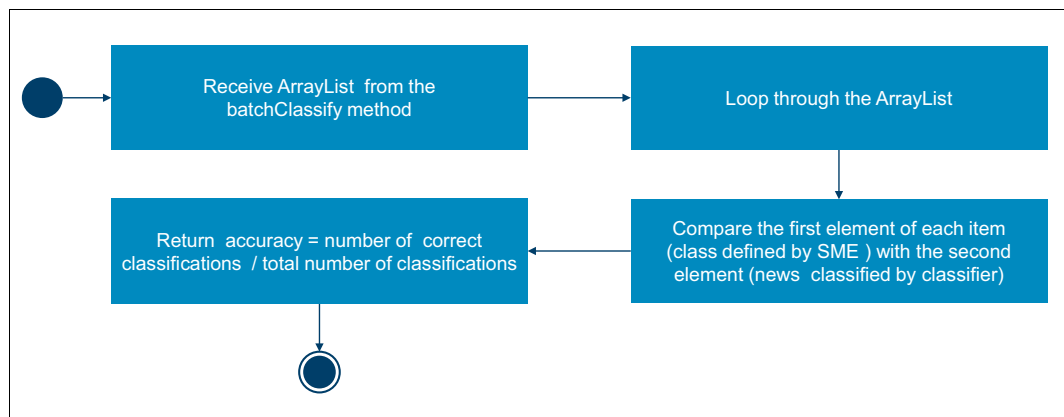


Figure 4-47 The generateAccuracy method of Evaluate class

Run the Evaluate program with Eclipse

Complete the following steps to run the Evaluate program with Eclipse:

1. Right-click `Evaluate.class` and select **Run As** → **Run Configurations**.
2. On the Main tab confirm that `com.ibm.itso.ed600r01.news.Evaluate` is selected (Figure 4-48).

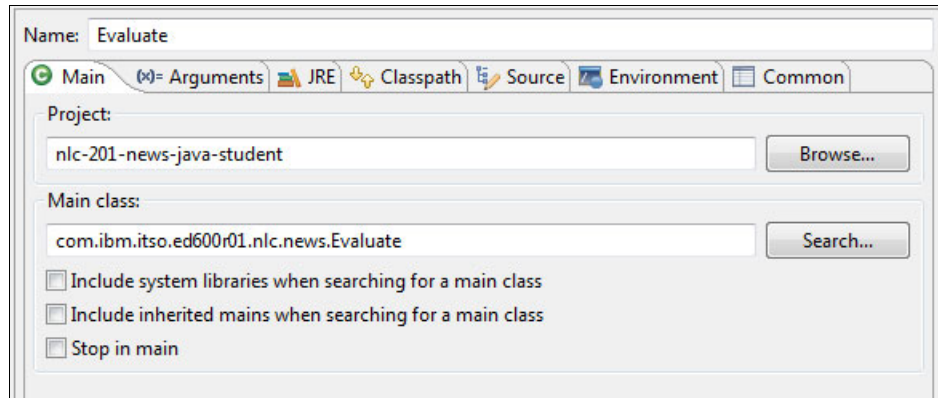


Figure 4-48 Evaluate program configuration when running on Eclipse

3. In the Arguments tab, enter the following parameters and click **Run** (Figure 4-49 on page 84):

```
<classifier_id> <user> <password> <csv_file_test>
```

The parameters have the following meanings:

- | | |
|----------------------|--|
| classifier_id | The classifier ID created in the create and train phase. |
| user | The username from the Natural Language Classifier service instance. |
| password | The password from the Natural Language Classifier service instance. |
| csv_file_test | The location in the local computer of the test CSV file. For this example, use the <code>news-test-nlc-toolkit.csv</code> file in the resource folder. |

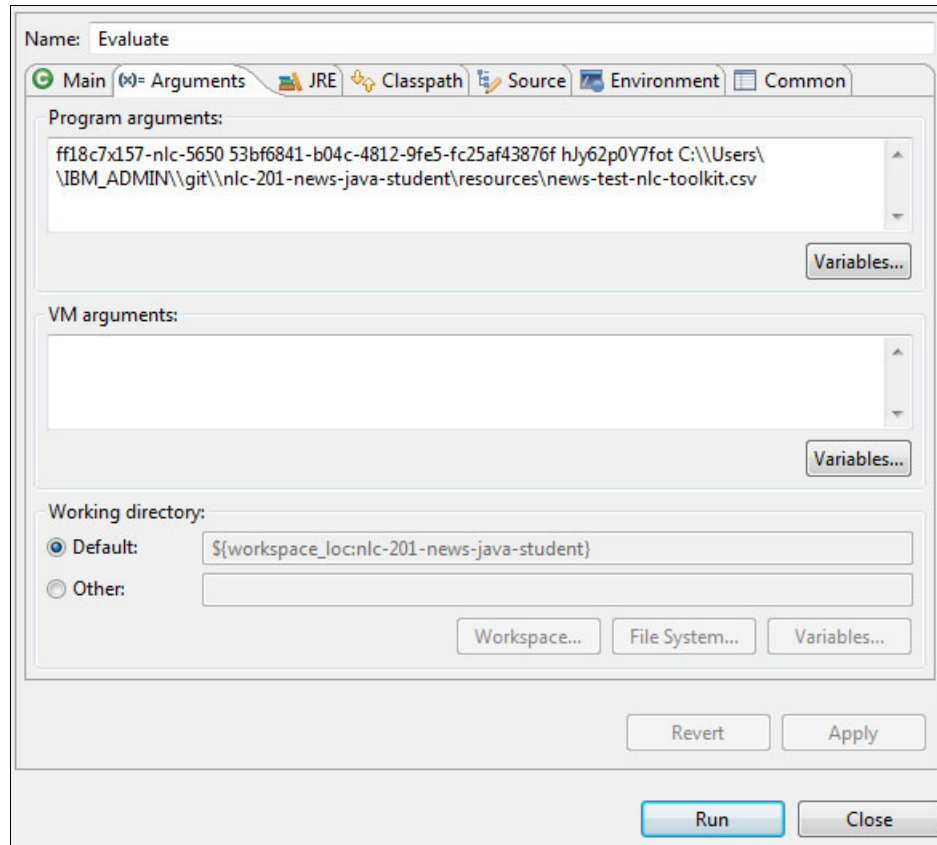


Figure 4-49 Evaluate arguments for execution on Eclipse

The final accuracy is output, as Figure 4-50 shows. In this case, the figure shows 71,43% of correct classification from the test set.

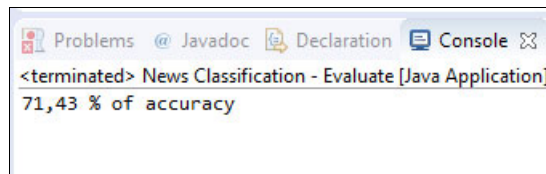


Figure 4-50 Evaluate program output on Eclipse

Run the Evaluate program from the command line

Follow these steps:

1. Open a command prompt in your computer.
2. Set up java.exe in your path.
3. Change to the resources directory of the project.


```
cd redbooks-nlc-201-news-java-student/resources
```


4. Run the Java command

– For Windows:

```
Evaluate.bat <classifier_id> <user> <password> <csv_file_test>
```

– For Linux:

```
./Evaluate.sh <classifier_id> <user> <password> <csv_file_test>
```

The command has these values:

- classifier_id** The classifier ID created in the create and train phase.
- user** The username from the Natural Language Classifier service instance.
- password** The password from the Natural Language Classifier service instance.
- csv_file_test** The location in the local computer of the test CSV file. For this example, use the news-test-nlc-toolkit.csv file in the resource folder.

The output is as follows:

```
$ ./Evaluate.sh ff18c7x157-nlc-5650 53bf6841-XXXc-4812-9fe5-fc25af43876f  
hJy62XXX7fot news-test-nlc-toolkit.csv  
71,43 % of accuracy
```

This output shows 71,43% of correct classification from the test data set.

Manage evaluated results and update the training data

Using the output from the Evaluate program, the SMEs can decide, based on business needs, whether they have to create a new classifier using the test data set to improve performance.

The Evaluation of the classifier performance can be continuous depending on business needs and user feedback.

Figure 4-51 shows an approach to evaluate the classifier performance by saving user feedback with the information needed for analysis in a database. SMEs analyze the feedback and decide whether the classifier must be improved.

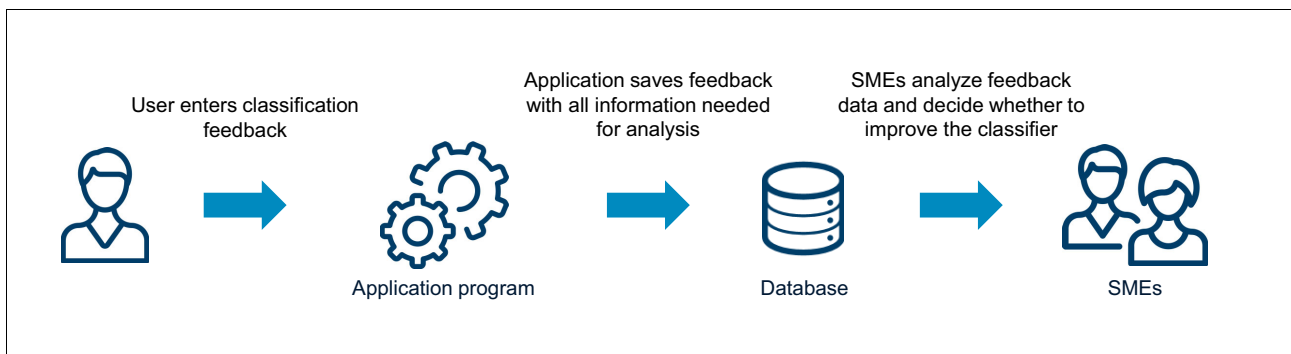


Figure 4-51 Classifier Feedback process in the application program

4.4.8 Deploying the application

This section shows two options for deploying the application to Bluemix:

- ▶ Deploy the application from Eclipse
- ▶ Deploy the application from command line

Deploy the application from Eclipse

To deploy the application from Eclipse, the Bluemix tools must be installed as listed in 4.1.2, “Prerequisites” on page 46. You will use the Bluemix Eclipse plug-in to deploy the application to Bluemix.

Complete the following steps:

1. Right-click project n1c-201-new-java-student in Eclipse and select **Run As** → **Run on Server**.
2. Under Select the server type, click **IBM** → **IBM Bluemix** and click **Next** (Figure 4-52).

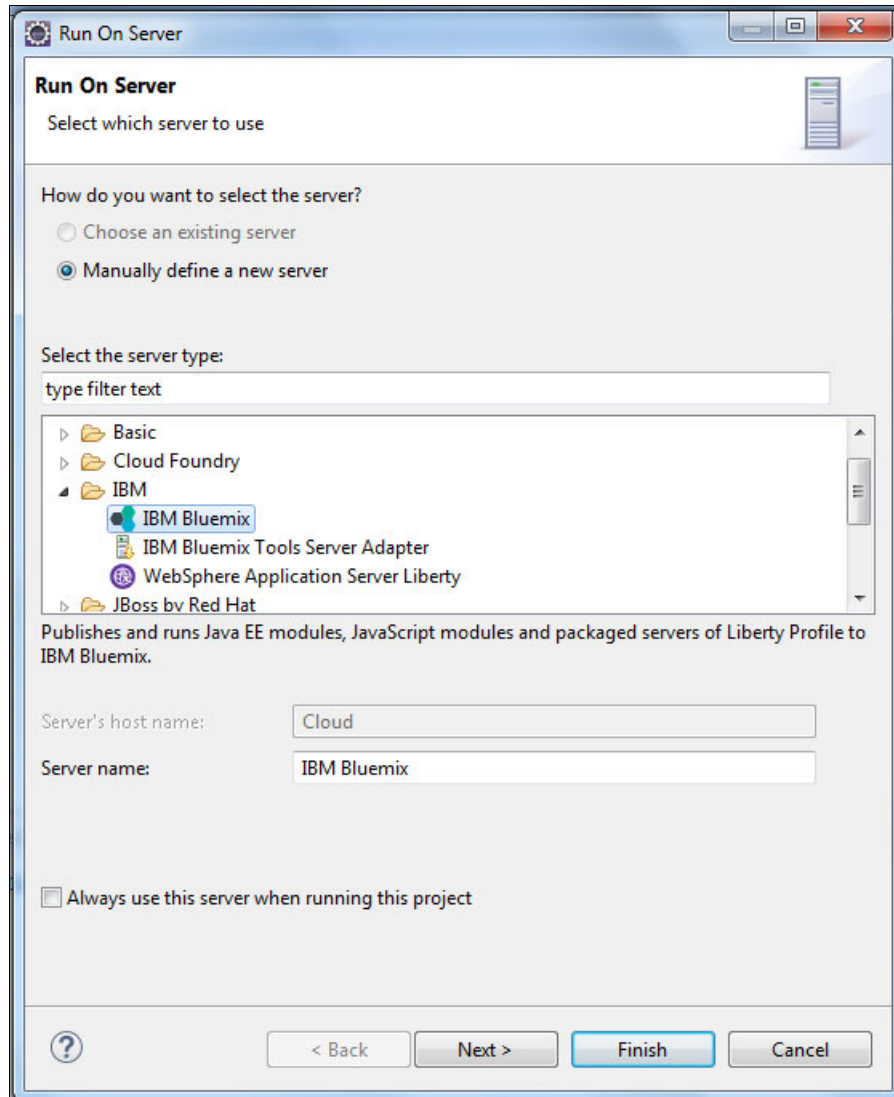


Figure 4-52 Select Bluemix server to host application

3. Enter the Bluemix account information (email and password) and click **Next** (Figure 4-53 on page 87).

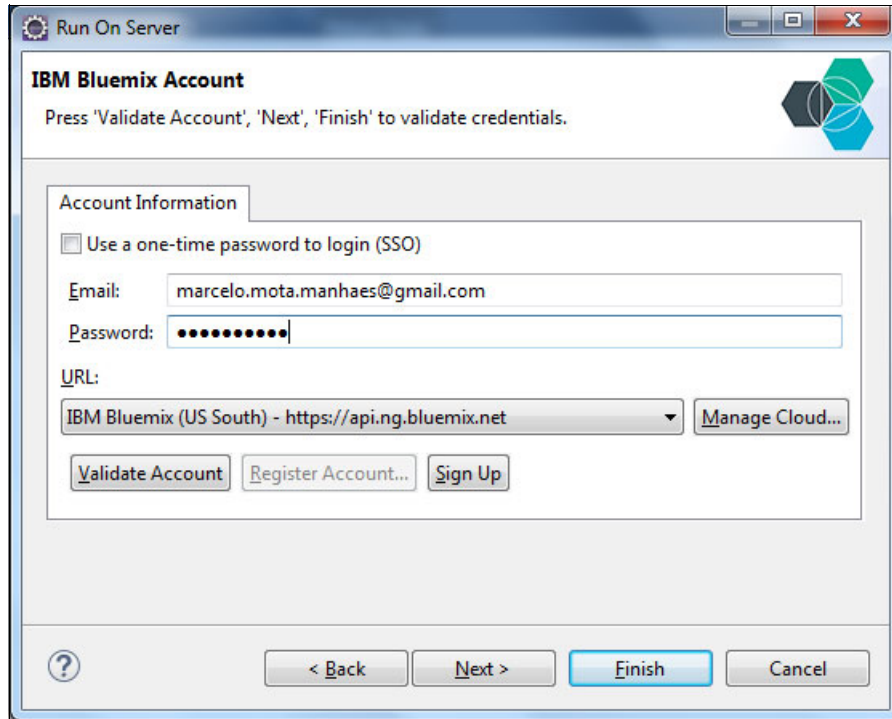


Figure 4-53 Username and password for Bluemix account

4. Select the Bluemix space that will host the application (Figure 4-54), By default, dev space is available if no other is created by the user. Click **Next**.

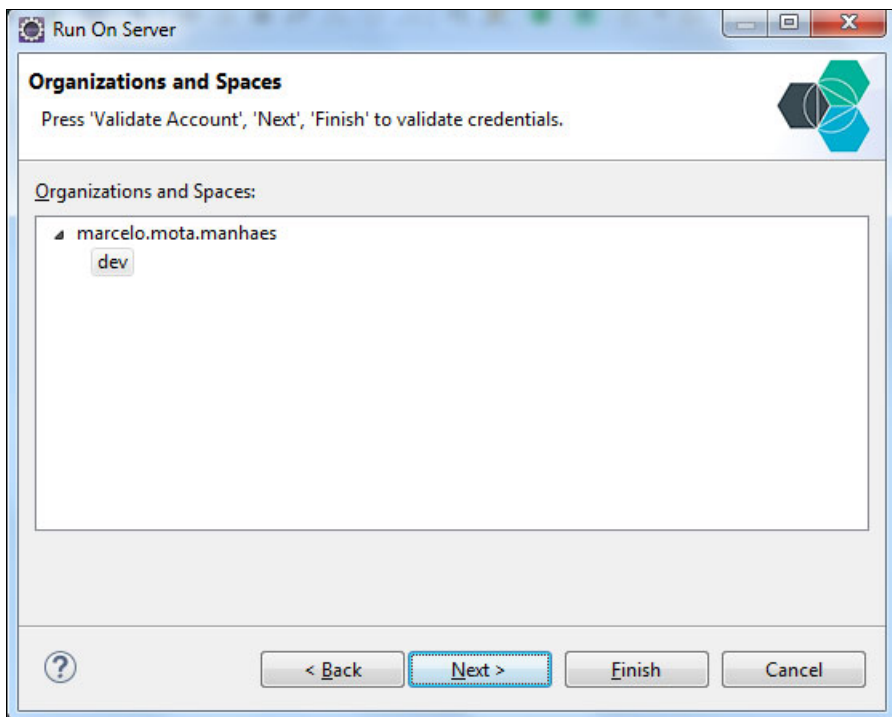


Figure 4-54 Select space to deploy application

5. The application selected to be deployed to Bluemix is recognized. Click **Finish** (Figure 4-55).

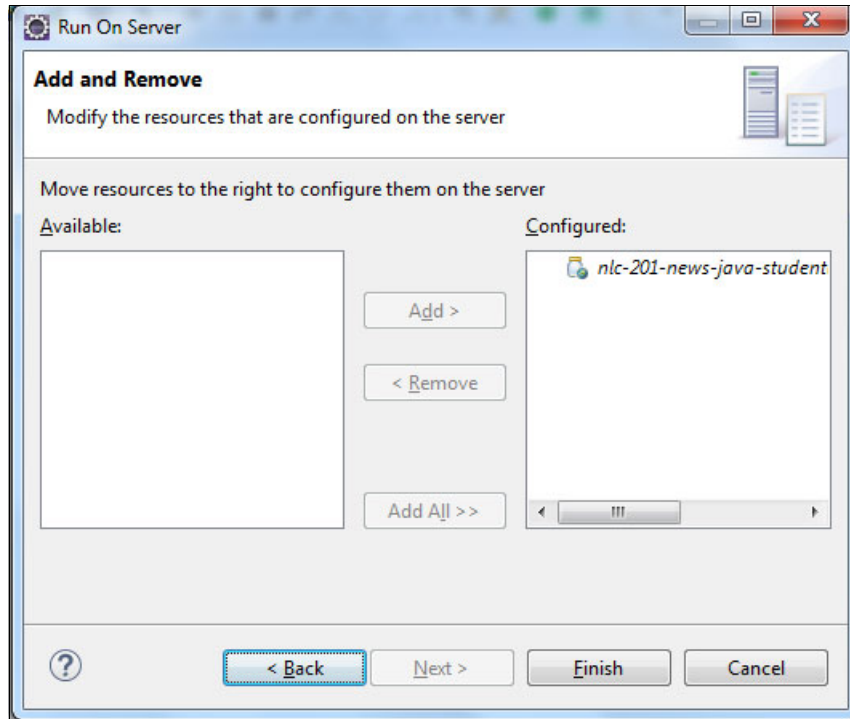


Figure 4-55 Application selected for deployment to Bluemix

6. The application details (Figure 4-56 on page 89) show the required configuration before the plug-in starts the deployment on the Bluemix.

Buildpack URL defines the application server to which the application is deployed. If this field is empty, the application will be hosted by IBM WebSphere Liberty profile.

If you want to deploy into Apache Tomcat you will need to use the Java Buildpack URL. See the [list of community build packs](#).

Select **Save to manifest file** and click **Next**.

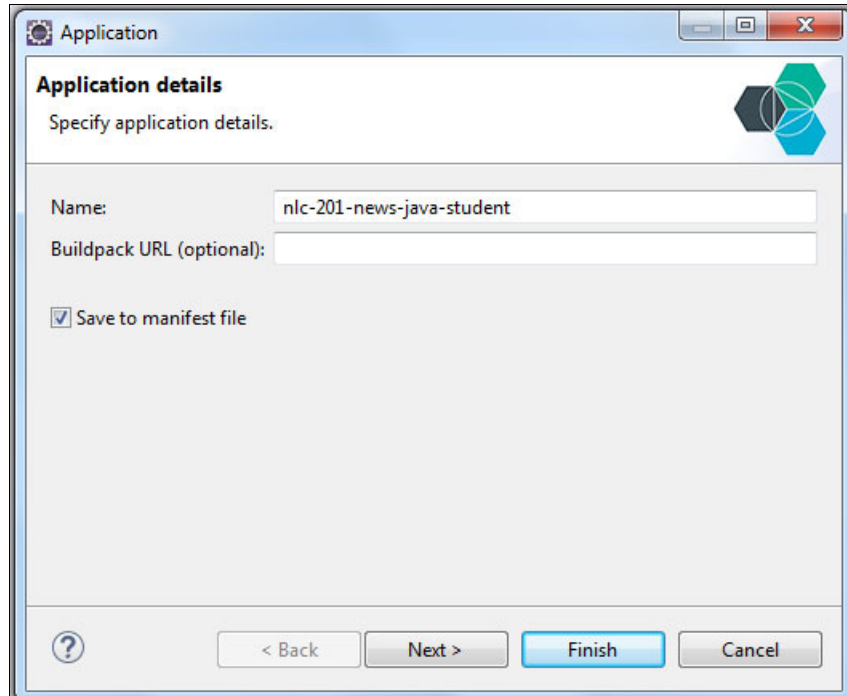


Figure 4-56 Buildpack selection to host application

7. The Launch deployment window (Figure 4-57) shows the deployment details. These include memory to be used, URL to access the application which is built by default using the application name plus the Bluemix domain. You can change this URL and verify that no one is using this URL by clicking **Validate**. Click **Next**.

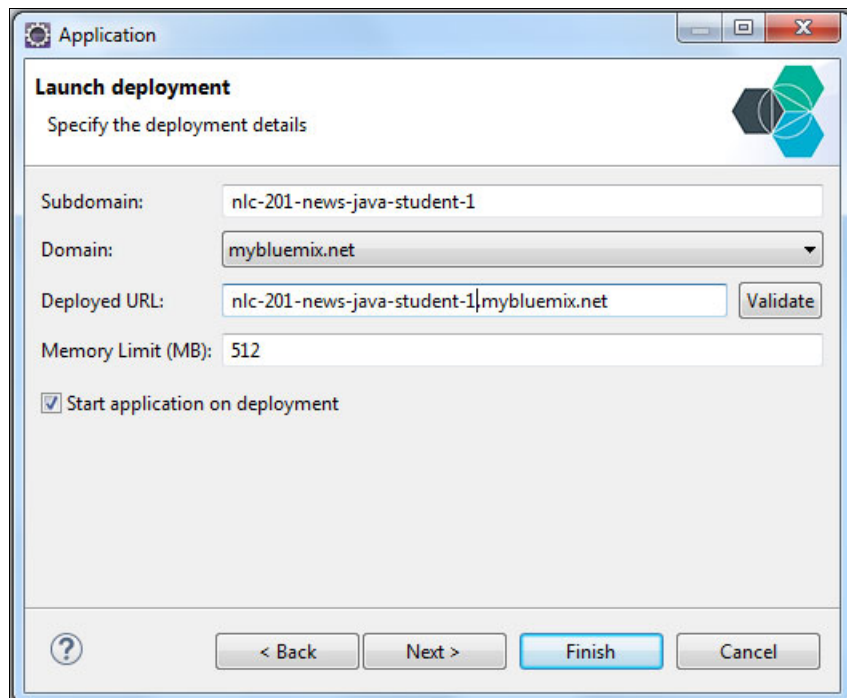


Figure 4-57 Launch deployment configuration on Bluemix

- This step binds the news classification service instance created in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11.

The Cloudant noSQL DB service instance must also be bound to the application; it is used by the application to save customer feedback. This service was created in 4.4.3, “Creating a Cloudant noSQL DB service instance” on page 56.

Select the services as shown in Figure 4-58 and click **Next**.

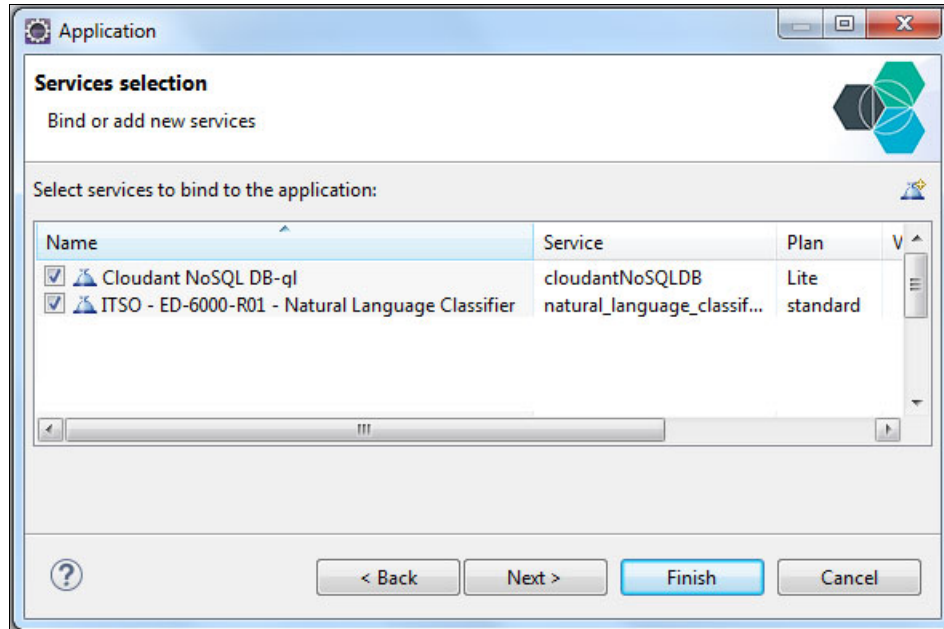


Figure 4-58 Select services to bind to the application

- Create a variable to store the `classifier_id` so it can be received by the application code. In the Environment Variables window click **New**.

Name the variable CLASSIFIER ID and in the Value field enter the classifier ID of the classifier created in 4.4.5, “Creating and training the classifier” on page 63.

Click **OK** and then click **Finish** (Figure 4-59 on page 91).

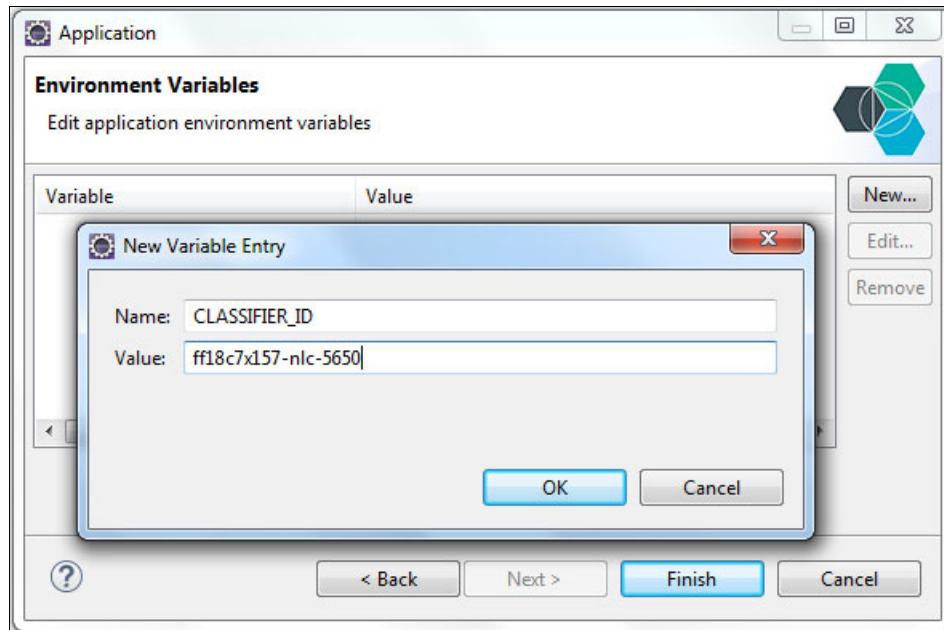


Figure 4-59 Classifier ID variable set up for News Application

If you lose the classifier ID, complete these steps:

- a. Log in to Bluemix.
- b. In the left menu, select **Services** → **Dashboard**.
- c. Go to the Natural Language Classifier service.
- d. Click **Manage** → **Access Beta Toolkit**.
- e. Click the classifiers link (right top menu).
- f. On the classifier page, the classifier ID is displayed (Figure 4-60).

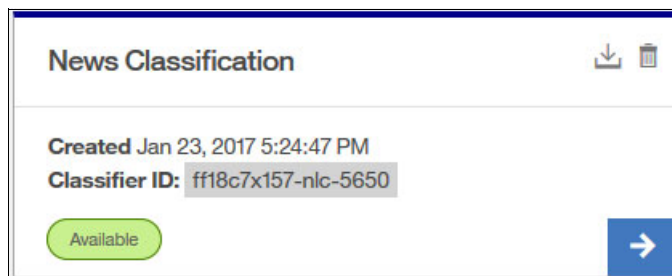


Figure 4-60 Displaying the classifier ID

The web application is deployed on Bluemix and several deployment messages are displayed on the Eclipse console, such as Liberty binaries download to host application, WebSphere Liberty profile server start logs, and others. Look for the message indicating that the application is running (Figure 4-61).

```
[AUDIT ] CWMKF0011I: The server defaultServer is ready to run a smarter planet.
[INFO ] CWMKF0008I: Feature update completed in 12.824 seconds.
[INFO ] CWWKO0219I: TCP Channel defaultHttpEndpoint has been started and is now listening for
Container became healthy
[Application Running Check] - Application appears to be running - nlc-201-news-java-student.
```

Figure 4-61 Application running on Bluemix

Deploy the application from the command line

This section assumes that you cloned the sample Git project as described in “Clone the sample Git project by using the Git command line” on page 55.

Follow these steps:

1. Change to the `redbooks-nlc-201-news-java-student` directory:
`cd redbooks-nlc-201-news-java-student`
2. In the root application directory, run `cf login` and put the email and password account for Bluemix in sequence (Example 4-9):

Example 4-9 The `cf login` command

```
cf login
  API endpoint: https://api.ng.bluemix.net
$
Email> <PUT_YOUR_BUEMIX_EMAIL_ACCOUNT>
$
Password> <PUT_YOUR_PASSWORD_ACCOUNT>

Authenticating...
OK
Targeted org <YOUR_ORGANIZATION>
```

3. Select the Bluemix space to host the application (Example 4-10):

Example 4-10 Select a space

```
Select a space (or press enter to skip):
1. dev
2. qa
3. Prod
Space> 1
Targeted space dev
API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: <YOUR_BUEMIX_EMAIL_ACCOUNT>
Org: <YOUR_ORGANIZATION>
Space: dev
```

4. Get the services names that will be bound to the application:
 - The first service is the Natural Language Classifier service instance that was created in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11.
 - The second service is the Cloudant NoSQL DB service instance to save client feedback that was created in 4.4.3, “Creating a Cloudant noSQL DB service instance” on page 56.

To get the name of services, run the `cf services` command and copy the name of the services from the name column (Example 4-11):

Example 4-11 The `cf services` command

```
cf services
Getting services in org <YOUR_ORGANIZATION> / space dev as
<YOUR_BUEMIX_EMAILACCOUNT>...
OK

name                service                plan    bound    last
```



```

Cloudant NoSQL DB-g1  cloudantNoSQLDB  Lite  apps  operation
create
succeeded

ITS0 - ED-6000-R01 - natural_language_standard  create
Natural Language classifier  succeeded
Classifier

```

5. In another command prompt, edit the `manifest.yml` file (Example 4-12) in the root of directory and change these items:
 - a. The host line: Insert your host name.
 - b. In the services section: Insert the names collected in step 4 on page 92.
 - c. The CLASSIFIER_ID line: Insert the value obtained in 4.4.5, “Creating and training the classifier” on page 63.

Example 4-12 The manifest.yml file

```

---
applications:
- name: nlc-201-news-java-student
  memory: 512M
  host: <YOUR_HOST_NAME>
  domain: mybluemix.net
  services:
  - <YOUR_CLOUDANT_SERVICE_NAME>
  - <YOUR_NATURAL_CLASSIFIER_SERVICE_NAME>
  env:
  CLASSIFIER_ID: <YOUR_CLASSIFIER_ID>

```

If you lose the classifier ID, complete these steps:

- i. Log in to Bluemix.
- ii. In the left menu, select **Services** → **Dashboard**.
- iii. Go to the Natural Language Classifier service.
- iv. Click **Manage** → **Access Beta Toolkit**.
- v. Click the classifiers link (right top menu).
- vi. On the classifiers page, the classifier ID is displayed (Figure 4-62).



Figure 4-62 Classifier ID collected on Bluemix service

- d. Save all the changes to the `manifest.yml` file.
6. At the prompt and from the root directory (`redbooks-nlc-201-news-java-student`) push the application to Bluemix. The information to deploy is in the `manifest.yml` file (step 5):

```
cf push nlc-201-news-java-student -p target\nlc-201-news-java-student.war
```

7. See the results (Example 4-13). The results show the application state, number of instances, memory usage, URL to access the application and other technical information.

Example 4-13 Results

```
requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: nlc-201-news-java-student.mybluemix.net
last uploaded: Wed Feb 15 14:37:18 UTC 2017
stack: cflinuxfs2
buildpack: Liberty for Java(TM) (WAR, liberty-16.0.0_4,
buildpack-v3.7-20170118-
2046, ibmjdk-1.8.0_20161213, env)
```

Note: The `urls` value will match the host and domain you entered in step 5 on page 93.

4.4.9 Testing the application

To test the application enter the application URL in a browser to display the home page (Figure 4-63).

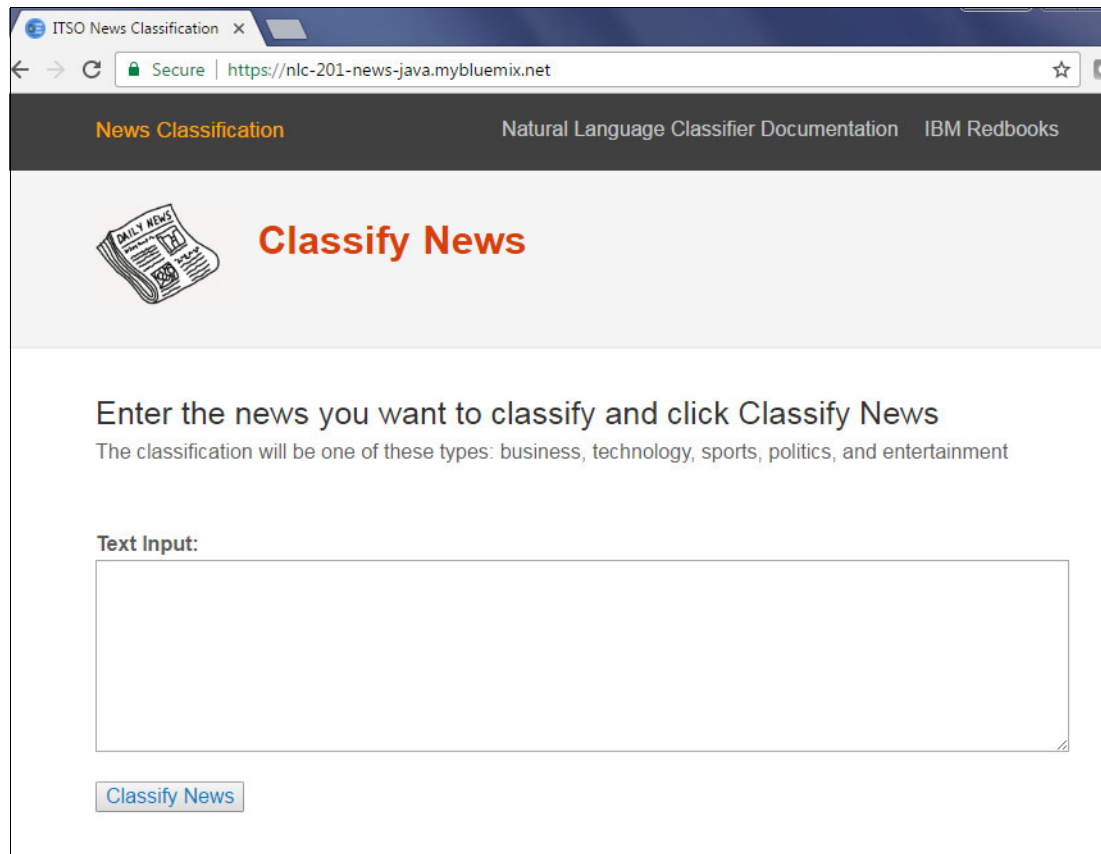


Figure 4-63 News application home page

Enter news text in the Text input field and click **Classify News**.

The result shows the Top Class suggested classification type (Figure 4-64). If the type is not correct, select another classification from the Suggest other classification pull-down and click **Feedback**.

Enter the news you want to classify and click Classify News


The classification will be one of these types: business, technology, sports, politics, and entertainment

Text Input:

Jose Mourinho refused to accept Manchester United's unbeaten run was over on Thursday night after struggling to contain his fury at referee Jonathan Moss awarding Hull City a controversial penalty. United booked their place in the EFL Cup final against Southampton despite Hull ending their 17-game unbeaten streak with a 2-1 win at the KCOM Stadium. But Mourinho claimed his side did not lose after being infuriated that Marcos Rojo was penalised for a push on Harry Maguire.

[Classify News](#)

Watson Natural Language Classifier

Top Class : sports 

Suggest other classification :

[Feedback](#)

Figure 4-64 News application results

Click the icon to the right of Top Class to see the classifier output details (Figure 4-65).


```
Top Class : sports   
{ -  
  "classes": [ -  
    { -  
      "confidence": 0.9864532252599781,  
      "class_name": "sports"  
    },  
    { -  
      "confidence": 0.00801902711571135,  
      "class_name": "politics"  
    },  
    { -  
      "confidence": 0.0028254919329835614,  
      "class_name": "entertainment"  
    },  
    { -  
      "confidence": 0.0014045313912377656,  
      "class_name": "technology"  
    },  
    { -  
      "confidence": 0.0012977243000891836,  
      "class_name": "business"  
    }  
  ],  
}
```

Figure 4-65 New application results in detail

If you do not agree with the classification and make a suggestion by using the feedback feature, your suggestion is sent to the database. With this data, the SMEs can verify and improve classifier accuracy (as explained in 4.4.7, “Evaluating results and updating training data” on page 73). See Figure 4-66.

```
{  
  "classifier_id": "90e7b7x198-nlc-36405",  
  "text": "Jose Mourinho refused to accept Manchester United's unbeaten run was over on Thursday night after struggling to contain his fury at referee Jonathan Moss awarding Hull City a controversial penalty. United booked their place in the EFL Cup final against Southampton despite Hull ending their 17-game unbeaten streak with a 2-1 win at the KCOM Stadium. But Mourinho claimed his side did not lose after being infuriated that Marcos Rojo was penalised for a push on Harry Maguire.",  
  "top_class": "sports",  
  "url": "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/90e7b7x198-nlc-36405"  
}
```

Suggest other classification :

[Feedback](#)

Figure 4-66 Feedback to change classification option

4.5 Quick deployment of application

A second Git repository is provided so that you can run the News Classification application even if you did not perform the steps described in 4.4, “Step-by-step implementation” on page 50.

1. You need a [Bluemix](#) account. Experimental Watson Services can be used at no cost.
2. Follow the requirements in 4.1.2, “Prerequisites” on page 46 to install Git and Cloud Foundry software on your local computer.
3. Open a command prompt and set up Git using the following command:

```
git config --global http.sslVerify false
```

4. Choose an empty directory to download the code.
5. Run the command in the selected directory:

```
git clone https://github.com/snippet-java/redbooks-nlc-201-news-java.git
```

6. Change to the redbooks-nlc-201-news-java directory:

```
cd redbooks-nlc-201-news-java
```

7. In the root application directory, run the **cf login** and provide the email and password account for Bluemix in the sequence shown in Example 4-14.

Example 4-14 The cf login command

```
cf login
  API endpoint: https://api.ng.bluemix.net
ϕ
Email> <PUT_YOUR_BLUEMIX_EMAIL_ACCOUNT>
ϕ
Password> <PUT_YOUR_PASSWORD_ACCOUNT>

Authenticating...
OK
Targeted org <YOUR_ORGANIZATION>
```

8. Select the Bluemix space to host the application (Example 4-15).

Example 4-15 Select a space

```
Select a space (or press enter to skip):
1. dev
2. qa
3. Prod
Space> 1
Targeted space dev
API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: <YOUR_BLUEMIX_EMAIL_ACCOUNT>
Org: <YOUR_ORGANIZATION>
Space: dev
```

9. Create the Natural Language Classifier service:

```
cf create-service natural_language_classifier standard my-nlc-service
```

10. Create service keys (service credentials) to access the Natural Language Classifier service:

```
cf create-service-key my-nlc-service myKey
```

11. Retrieve the service keys from the Natural Language Classifier service to use after:

```
cf service-key my-nlc-service myKey
```

12. Create a service for the database feedback function of this application:

```
cf create-service CloudantNoSQLDB Lite "News Classifier Feedback"
```

13. The Natural Language Classifier service must be trained before you can successfully use this application. The training data is provided in the `resources/news-train.csv` file from the `redbooks-nlc-201-news-java` root directory. Open the resources directory:

```
cd redbooks-nlc-201-news-java/resources
```

14. Execute the Java command:

– For Windows:

```
CreateAndTrain.bat news-train.csv <user> <password> "News Classifier" en
```

– For Linux:

```
./CreateAndTrain.sh news-train.csv <user> <password> "News Classifier" en
```

The command has the following parameters:

user The user name from step 11.

password The password from step 11.

The information output from this command will show the classifier ID. Keep this information.

15. At the command prompt, edit the `manifest.yml` in the root directory (`redbooks-nlc-201-news-java`) and change the following information (Example 4-16):

a. `host`: Use a host name that is unique.

b. `CLASSIFIER_ID`: Insert the value you created in step 14 on page 98.

Example 4-16 The manifest.yml file

```
---
applications:
- name: nlc-201-news-java
  memory: 512M
  host: <YOUR_HOST_NAME>
  domain: mybluemix.net
  services:
  - News Classifier Feedback
  - News Classifier
  env:
    CLASSIFIER_ID: <YOUR_CLASSIFIER_ID>
```

16. Save all the changes to the `manifest.yml` file.

17. At the prompt and from the root directory (`redbooks-nlc-201-news-java`), push the application to Bluemix:

```
cf push nlc-201-news-java -p target\nlc-201-news-java-student.war
```

18. After completing these steps, you are ready to test your application. Start a browser and enter the URL of your application:

```
<YOUR_HOST_NAME>.mybluemix.net
```

4.6 References

See the following resources:

- ▶ Create classifier:

https://www.ibm.com/watson/developercloud/natural-language-classifier/api/v1/#create_classifier

- ▶ Using your own data to train the Natural Language Classifier:

<https://www.ibm.com/watson/developercloud/doc/natural-language-classifier/using-your-data.html>



SPAM Classifier

The SPAM Classifier application in this use case reads mail subject or contents that the user provides and classifies whether the mail is spam or not. The user provides feedback to the classification results about whether it is correctly or incorrectly classified. User feedback is saved for additional training of the Natural Language Classifier classifier.

SPAM Classifier uses Natural Language Classifier (NLC) service, one of the cognitive capabilities that IBM Watson provides. It understands natural language and classifies text into one of several predefined classes. The classifier is *trained* with training data, which is prepared for each purpose but can be improved with additional training from new training data to make the classifier smarter.

The following topics are covered in this chapter:

- ▶ Getting started
- ▶ Architecture
- ▶ Two ways to deploy the application: Step-by-step and quick deploy
- ▶ Step-by-step implementation
- ▶ Quick deployment of application
- ▶ References

5.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

5.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- ▶ Understand practical applications of the Watson Natural Language Classifier service, such as spam classification.
- ▶ Follow the procedure to use the Natural Language Classifier service.
- ▶ Implement and deploy the use case application in a Node-RED environment on Bluemix.

5.1.2 Prerequisites

To build Watson Natural Language Classifier service based on Watson Natural Language Classifier on Bluemix and implement a controller in Node-RED, you must have the following accounts, resources, knowledge, and experiences:

- ▶ Bluemix account
- ▶ Node-RED application on Bluemix
- ▶ cURL, a command-line tool for transferring data by URL syntax
- ▶ Internet browser such as Chrome, Firefox, Internet Explorer, Safari
- ▶ Basic implementation skill with JavaScript

5.1.3 Expected results

By following the steps in this book, you should be able to run the application in a browser by interacting with the classifier through three web pages:

1. Request classification (Figure 5-1).

On the first page, the user enters mail subject or content, as one line, to be classified and then submits the request.

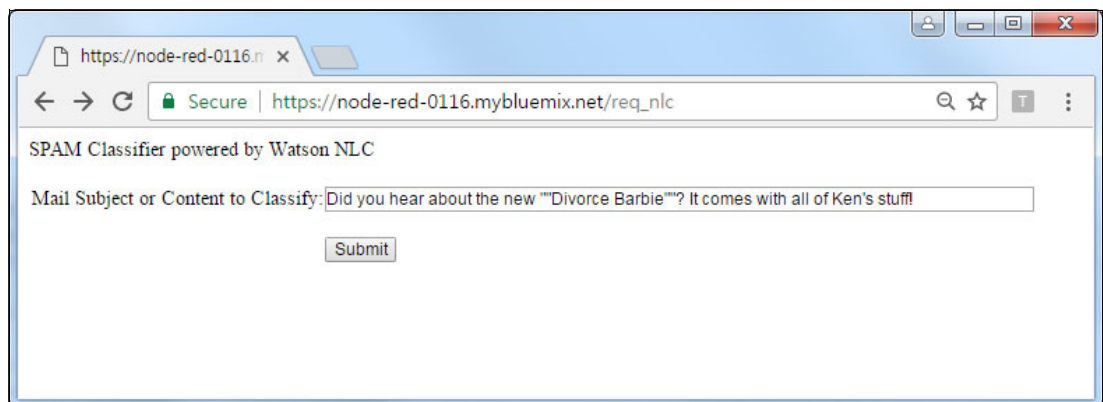


Figure 5-1 Input page of mail subject or content

2. Display classification result (Figure 5-2).

User receives the classification results and is prompted to agree or disagree with the results. Every time the user provides feedback about the classification results, the feedback is saved to a Cloudfant database in Bluemix for additional training.

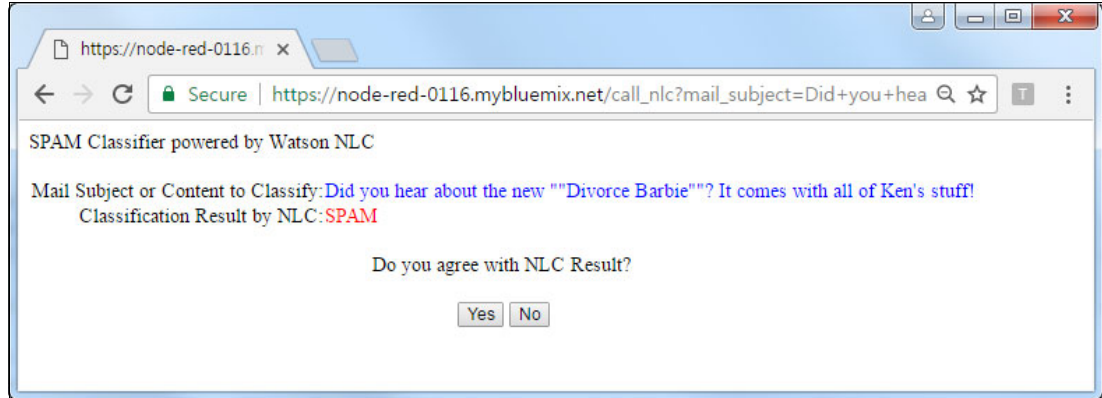


Figure 5-2 Classification result is displayed

3. Review user feedback (Figure 5-3).

For the user's request of feedback review, the SPAM Classifier displays the user feedback from the Cloudfant database.

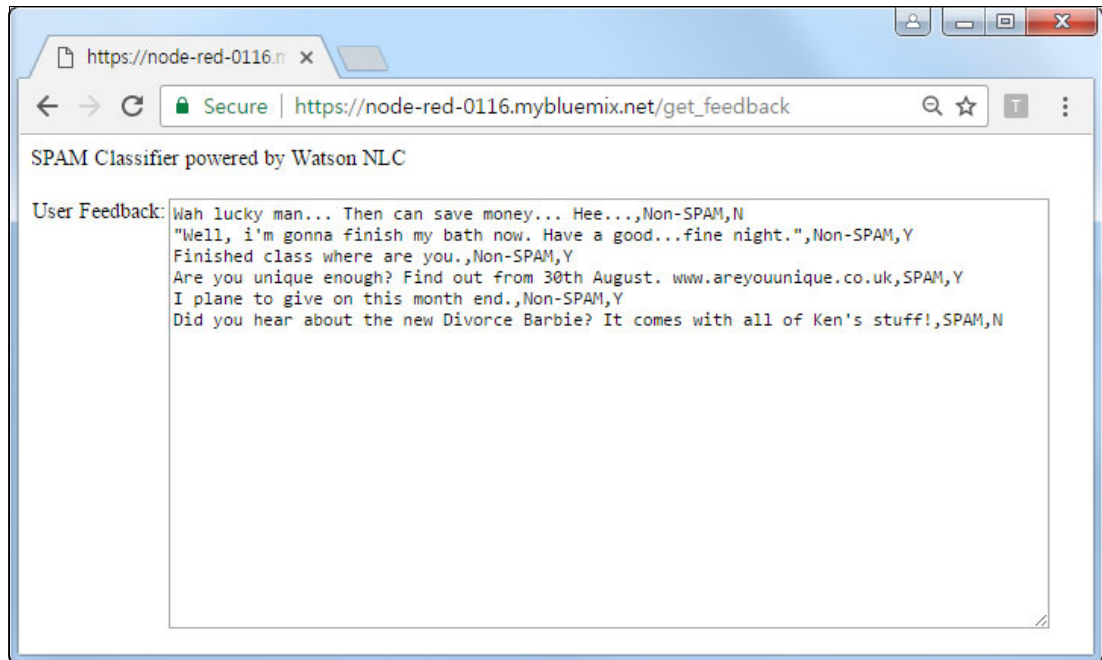


Figure 5-3 User feedback is displayed

5.2 Architecture

The SPAM Classifier architecture is described from the following perspectives:

- ▶ Static perspective is described in a component perspective.
- ▶ Dynamic perspective is described in a role and activity perspective.

SPAM Classifier application is composed of an application controller, Watson Natural Language Classifier Service, and data store. The application controller orchestrates the classification service. Watson Natural Language Classifier service classifies whether the subject or content of mail is spam or non-spam. The data store saves the user feedback about the classification result.

5.2.1 Component perspective

Figure 5-4 shows the components and data flow.

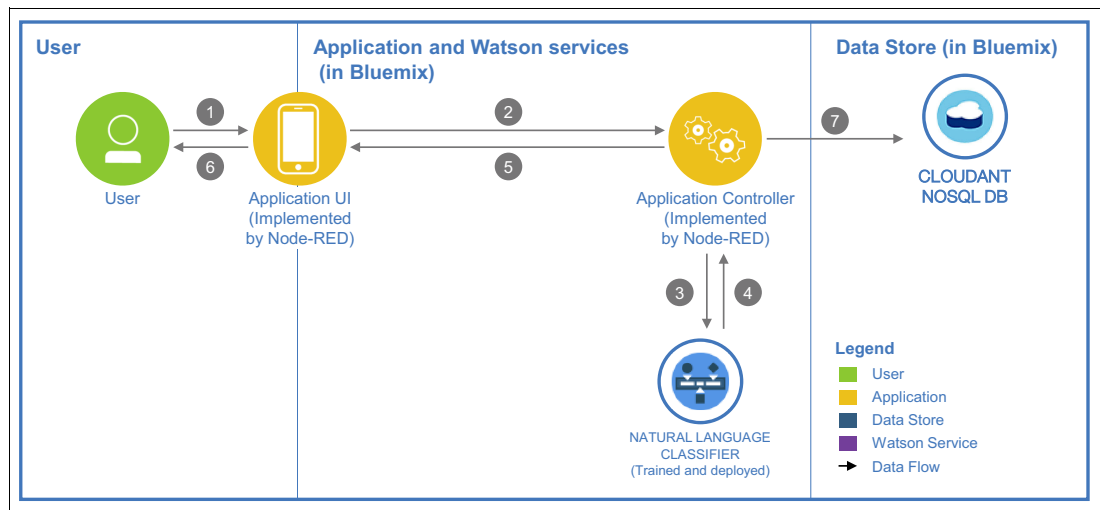


Figure 5-4 Component diagram

Data flows as follows:

1. Mail subject or content, approve or reject.
2. Mail subject or content, approve or reject.
3. Mail subject or content.
4. Classification result: spam or non-spam.
5. Classification result: spam or non-spam.
6. Classification result: spam or non-spam.
7. User feedback regarding classification result.

5.2.2 Role and activity perspective

Figure 5-5 shows the role and activity service flow.

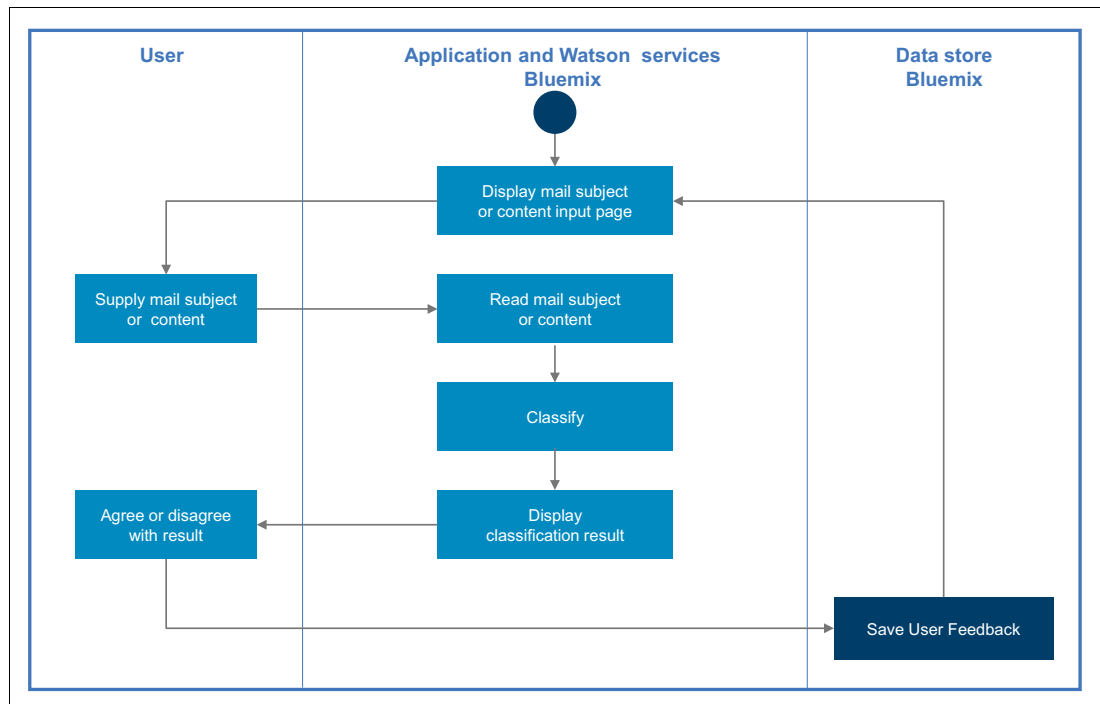


Figure 5-5 Role and activity diagram

The flow from the role and activity perspective is as follows:

1. User accesses the SPAM Classifier application URL with a web browser.
2. Application controller displays the input page.
3. User enters mail subject or content on the input form and submits it.
4. Application controller reads mail subject or content and queries the Natural Language Classifier classifier to classify whether it is spam or non-spam.
5. Application controller displays the classification result, from the Watson Natural Language Classifier service, in a web response to the user.
6. User provides feedback by agreeing or disagreeing with the classification result.
7. Application controller saves user feedback into data store to update the training data.

5.3 Two ways to deploy the application: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- ▶ Step-by-step deployment (*incomplete*) version of the application

This repository contains an incomplete version of the application and is used in all sections of 5.4, “Step-by-step implementation” on page 106. This version takes you through the key steps to integrate the IBM Watson services with the application logic.

- ▶ Quick deployment (*complete*) version of the application

This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 5.5, “Quick deployment of application” on page 120.

5.4 Step-by-step implementation

Deploying this application involves the following steps:

1. Creating a Node-RED application
2. Cloning the Git project
3. Preparing training data
4. Creating and training the classifier
5. Querying the trained classifier
6. Evaluating results and updating training data

5.4.1 Creating a Node-RED application

For the deployment of this use case, the application UI and application controller to query the classifier are developed in a Node-RED app and user feedback is saved to Cloudant noSQL DB. You should create both, the Node-RED application and the Cloudant noSQL DB service on Bluemix.

The web pages, controller, and Watson Natural Language Classifier service are implemented in Node-RED. After you log in to Bluemix, create an app of Node-RED:

1. In IBM Bluemix, open the full catalog (Figure 5-6 on page 107). Under Apps, click **Boilerplates** → **Node-RED Starter**.

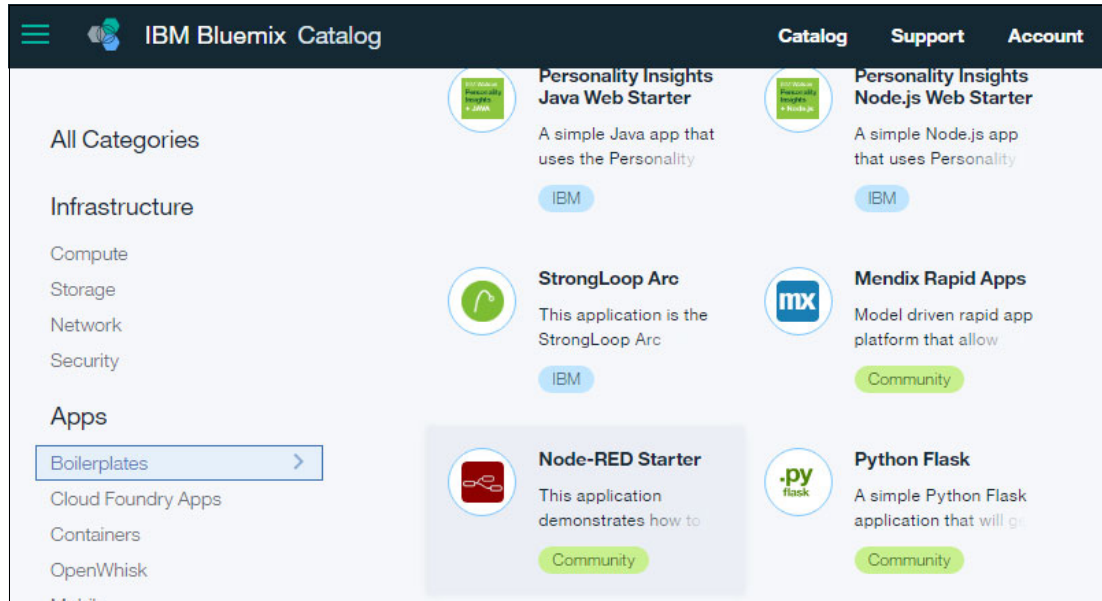


Figure 5-6 IBM Bluemix Catalog: Node-RED Started

2. Provide an App name (Figure 5-7) and click **Create**.

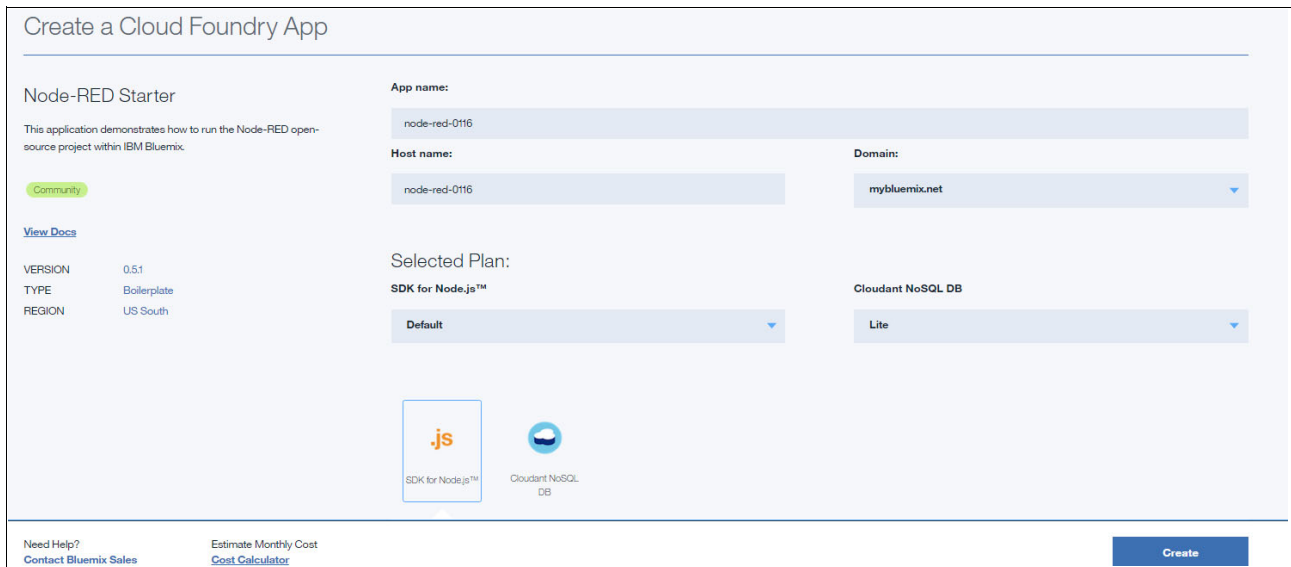


Figure 5-7 Create Node-RED App

3. The App is now created. Click the URL link in the ROUTE column (Figure 5-8 on page 108).

All Apps (1) Create App +

Cloud Foundry Apps 1 GB/8 GB Used

NAME	ROUTE	STATE	ACTIONS
node-red-0116	node-red-0116.mybluemix.net	● Running	↻ ↗ ⋮

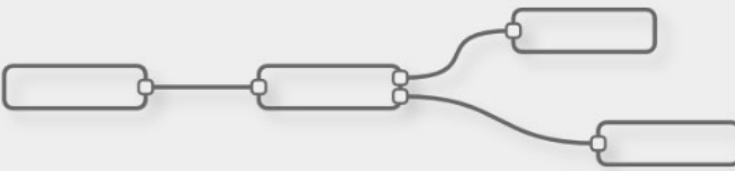
Figure 5-8 Node-RED App created

4. Node-RED in Bluemix opens (Figure 5-9). Click **Go to your Node-RED flow editor**.

Node-RED in Bluemix

Node-RED in Bluemix

A visual tool for wiring the Internet of Things



Node-RED provides a browser-based editor that makes it easy to wire together flows that can be deployed to the runtime in a single-click.

Go to your Node-RED flow editor

Figure 5-9 Node-RED editor

5. Now you have a Node-RED development environment available (Figure 5-10).

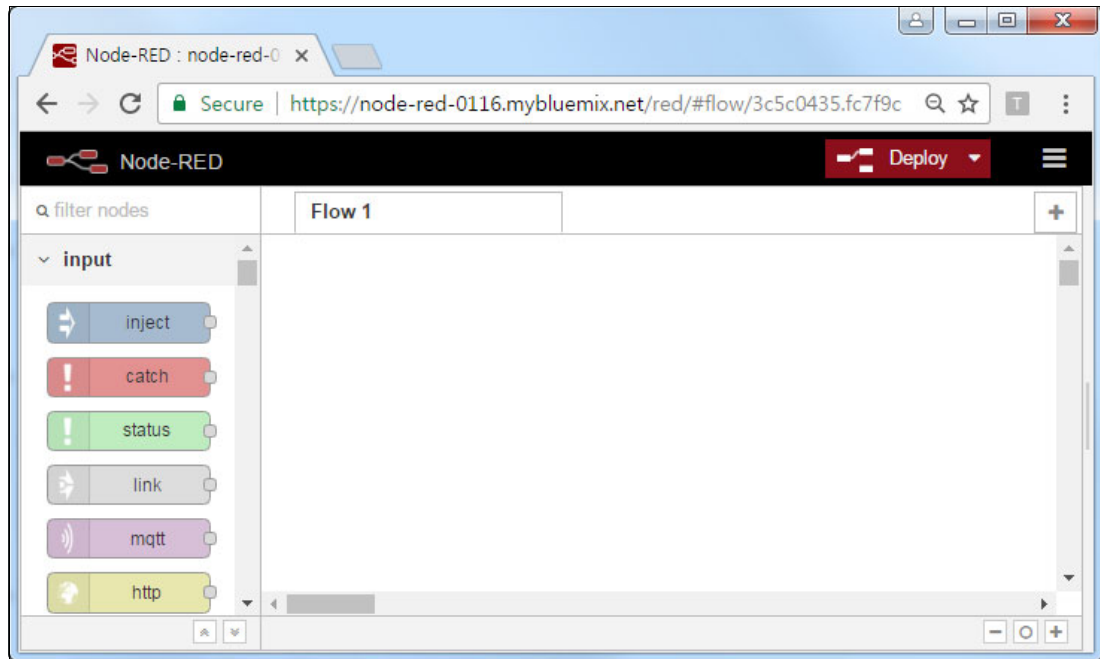


Figure 5-10 Node-RED editor

5.4.2 Cloning the Git project

A Git project was created for this use case. Clone the Git project:

1. Open a command prompt and set up Git by using the following command:

```
git config --global http.sslVerify false
```
2. Choose an empty directory to download the code.
3. Run the command in the selected directory:

```
git clone  
https://github.com/snippet-java/redbooks-nlc-201-spam-nodered-student.git
```
4. Change to the nlc-201-spam-nodered-student directory:

```
cd nlc-201-spam-nodered-student
```

After cloning the project, you can find the exported Node-RED flow and training data for the project:

- ▶ Node-RED flow: nlc-201-spam-nodered-student/defaults/flow.json
- ▶ Training data: nlc-201-spam-nodered-student /resources/spam_training_1.csv

5.4.3 Preparing training data

Training data should be prepared in advance because when you create the classifier, training data should be provided at that point. The training data is in the form of a comma separated value (CSV) file, which is composed of text and a label. A convenient approach is to create data in Microsoft Excel and save it in CSV format, for example a spam_training_1.csv file.

Figure 5-11 shows example training data.

	A	B
1	Go until jurong point, crazy.. Available only in bugis n great world la	Non-SPAM
2	Ok lar... Joking wif u oni...	Non-SPAM
3	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.	SPAM
4	U dun say so early hor... U c already then say...	Non-SPAM
5	Nah I don't think he goes to usf, he lives around here though	Non-SPAM
6	FreeMsg Hey there darling it's been 3 week's now and no word back	SPAM
7	Even my brother is not like to speak with me. They treat me like aids	Non-SPAM

Figure 5-11 Training data

Watson Natural Language Classifier supports multiple classifications. In the SPAM Classifier application, training data has only two classifications: SPAM or Non-SPAM. Each line of data should be labelled with only one of them.

About the data

Training data used in this use case was compiled by Tiago Agostinho de Almeida and José María Gómez Hidalgo. More information is in 5.6, “References” on page 122.

5.4.4 Creating and training the classifier

This section describes the steps to create and train the classifier.

Create a service of the Natural Language Classifier

You must create a Natural Language Classifier service instance in Bluemix as described in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11 before performing the steps in this section.

When you develop an application in Node-RED, the classifier should run in the same space of Bluemix where Node-RED runs. For this use case, the Natural Language Classifier service was created in the same space with the following credential information:

```
url          "https://gateway.watsonplatform.net/natural-language-classifier/api"
password     "y1wcQL63akRX"
username     "1b2749fe-7581-42e2-ad3e-115c022ef8cd"
```

Create a classifier with initial training data

Now you are ready to create a classifier with the `curl` command (Example 5-1).

Example 5-1 The curl command

```
curl -i -u "<username>":"<password>" -F training_data=@ <traing_data_file_path> -F
training_metadata="{\"language\": \"en\", \"name\": \"TutorialClassifier\"}"
"https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers"
```

In this example, replace the following information:

- Replace `<username>` and `<password>` with service credentials obtained when you created the service as explained in Chapter 2, “Creating a Natural Language Classifier service in Bluemix” on page 11, for example, `"1b2749fe-7581-42e2-ad3e-115c022ef8cd"` and `"y1wcQL63akRX"` respectively.

- ▶ Replace <training_data_file_path> with the full path of the training data file, which includes the folder and file name. If you execute a **cURL** command in the folder where the training data file is located, you can specify the file name without the full path (for example, "spam_training_1.csv").

Figure 5-12 shows an example of **curl** command.

```
C:\Watson>curl -i -u "1b2749fe-7581-42e2-ad3e-115c022ef8cd":"y1wcQL63akRX" -F training_data=@spam_training_1.csv -F training_metadata="{<W"languageW":W"enW",W"nameW":W"TutorialClassifierW"}" "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers"
```

Figure 5-12 Sample **curl** command

The cURL response

After the **cURL** command runs successfully, it returns a response with classifier ID, for example "f5b42fx173-nlc-3980", which you need to retrieve for later use. Training begins immediately with the initial training data.

Figure 5-13 on page 111 shows an example of a **cURL** response.

```
{
  "classifier_id" : "f5b42fx173-nlc-3980",
  "name" : "TutorialClassifier",
  "language" : "en",
  "created" : "2017-02-14T07:50:21.598Z",
  "url" : "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/f5b42fx173-nlc-3980",
  "status" : "Training",
  "status_description" : "The classifier instance is in its training phase, not yet ready to accept classify requests"
}
```

Figure 5-13 Sample **cURL** response

5.4.5 Querying the trained classifier

As the component diagram shows (Figure 5-4 on page 104), the user interacts with the SPAM Classifier application through a web user interface. The Watson Natural Language Classifier service performs classification of the user input. The application controller orchestrates the overall process.

Request classification

You create the request classification page in Node-RED for the user to access through a web address. For this use case, pages were previously created by the authors. Follow these steps to import the pages into the Node-RED environment:

1. In the Node-RED editor, click the top right menu and select **Import** → **Clipboard** (Figure 5-14).

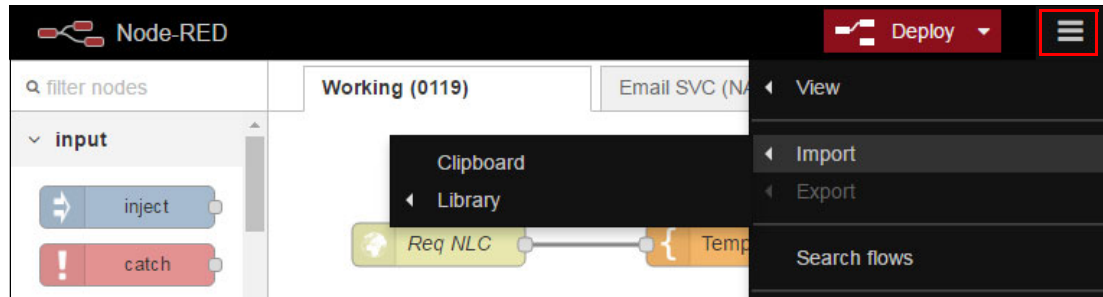


Figure 5-14 Import menu

2. The clipboard window opens. Copy the code (Example 5-2 is the code snippet to import). Paste the code and click **Import** (Figure 5-15).

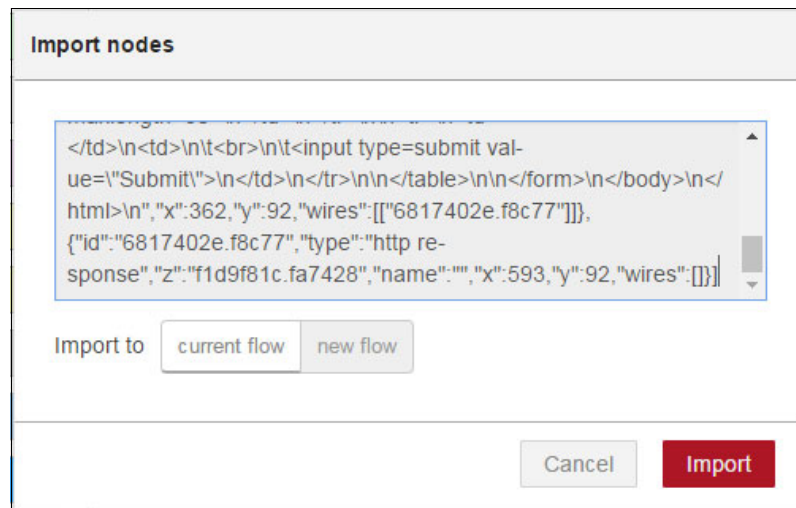


Figure 5-15 Paste sample code

Example 5-2 shows the sample code to import. This snippet is part of an exported Node-RED flow file (`flow.json`), which is included in the project folder cloned by Git. After you import the entire content of `flow.json`, you get all nodes and the links between them.

Example 5-2 Sample code to import (`flow.json`)

```
[{"id": "3a346689.6c13ca", "type": "http in", "z": "f1d9f81c.fa7428", "name": "/req_nlc", "url": "/req_nlc", "method": "get", "swaggerDoc": "", "x": 95.89584350585938, "y": 102, "wires": [{"e225af86.0689e"}]}, {"id": "e225af86.0689e", "type": "template", "z": "f1d9f81c.fa7428", "name": "Template: Req NLC", "field": "payload", "fieldType": "msg", "format": "handlebars", "syntax": "plain", "template": "<html>\n<body>\nSPAM Classifier powered by Watson NLC\n<br><br>\n<form action=\"\n/call_nlc\">\n\n<table>\n\n<tr>\n\n<td align=right>\n\n<tMail Subject or Content to Classify:\n</td>\n\n<td>\n\n<input type=text name=\"mail_subject\" size=80
```


To import these modules into the Node-RED environment, complete these steps:

1. In the Node-RED editor, click the top right menu and select **Import** → **Clipboard**.
2. The clipboard window opens. Copy the code (Example 5-3 is the code snippet to import). Paste the code and click **Import**.

Example 5-3 is sample code to import. This snippet is part of the exported Node-RED flow file (`flow.json`), which is included in the project folder cloned by Git. After you import the entire content of `flow.json`, you get all nodes and the links between them.

Example 5-3 Sample code to import (flow.json)

```
[{"id":"3b41f3fb.1f941c","type":"http
in","z":"f1d9f81c.fa7428","name":"/call_nlc","url":"/call_nlc","method":"get","swaggerDo
c":"","x":92.89582824707031,"y":273,"wires":[["5f7cbce2.f278b4"]]},{"id":"28d7ff89.2d239
","type":"watson-natural-language-classifier","z":"f1d9f81c.fa7428","name":"NLC","mode":
"classify","language":"en","classifier":"f5b42fx173-nlc-3980","x":502.8958282470703,"y":
273,"wires":[["544d5bac.0ac034"]]},{"id":"5f7cbce2.f278b4","type":"function","z":"f1d9f8
1c.fa7428","name":"Parse mail_subject","func":"/**\n * Set msg.payload with mail_subject
user provided\n * Save mail_subject into global context for later use\n */\n\nvar
mail_subject = msg.req.query.mail_subject;\n\ncontext.global.mail_subject =
mail_subject;\nmsg.payload = mail_subject;\n\nreturn
msg;\n","outputs":1,"noerr":0,"x":293.8958282470703,"y":273,"wires":[["28d7ff89.2d239"]
]},{"id":"a3f5e89a.e214d8","type":"http response","z":"f1d9f81c.fa7428","name":"Display
NLC
Result","x":879.8958129882812,"y":384,"wires":[]},{"id":"2bd7fc4a.094154","type":"templa
te","z":"f1d9f81c.fa7428","name":"Template: NLC
Result","field":"payload","fieldType":"msg","format":"handlebars","syntax":"mustache","t
emplate":"<!--\npayload.mail_subject: mail_subject user provided\npayload.top_class:
classification result against mail_subject, either SPAM or
Non-SPAM\n-->\n\n<html>\n<head>\n<script>\n    function clickYes() {\n
document.form1.user_feedback.value = \"Y\";\n        document.form1.submit();\n    }\n
function clickNo() {\n        document.form1.user_feedback.value = \"N\";\n
document.form1.submit();\n    }\n</script>\n</head>\n<body>\nSPAM Classifier
powered by Watson NLC\n<br><br>\n\n<form name=form1
action=\"/update_feedback\">\n\n<input type=hidden name=\"user_feedback\">\n\n<table
border=0>\n\n<tr>\n<td align=right>\n\nMail Subject or Content to
Classify:\n\n</td>\n\n<td>\n\n<font color=\"blue\">{{payload.mail_subject}}</font>\n\n<input
type=hidden name=\"mail_subject\"
value=\"{{payload.mail_subject}}\"\n\n</td>\n\n</tr>\n\n<tr>\n\n<td
align=right>\n\nClassification Result by NLC:\n\n</td>\n\n<td>\n\n<font
color=\"red\">{{payload.top_class}}</font>\n\n<input type=hidden
name=\"classification_result\"
value=\"{{payload.top_class}}\"\n\n</td>\n\n</tr>\n\n<tr>\n\n<td colspan=2
align=center>\n\n<br>\nDo you agree with NLC Result?\n\n<br><br>\n\n<input type=button
value=\"Yes\" onClick=\"javascript:clickYes();\"\n\n<input type=button value=\"No\"
onClick=\"javascript:clickNo();\"\n\n</td>\n\n</tr>\n\n</table>\n\n</form>\n\n</body>\n\n</html
>\n","x":640.8958129882812,"y":384,"wires":[["a3f5e89a.e214d8"]]},{"id":"2124dc51.d636c4
","type":"comment","z":"f1d9f81c.fa7428","name":"User Submit -> Call NLC to classify
SPAM / Non-SPAM","info":"When user submit \"mail subject or contents\", \n\nsystem ask NLC
to classify if it is SPAM or
Non-SPAM","x":231.8958282470703,"y":221,"wires":[]},{"id":"544d5bac.0ac034","type":"func
tion","z":"f1d9f81c.fa7428","name":"Deliver mail_subject","func":"/**\n * Get
mail_subject from global context and deliver to template generator\n
*/\n\nmsg.payload.mail_subject = context.global.mail_subject;\n\nreturn
msg;\n","outputs":1,"noerr":0,"x":376.72222900390625,"y":384.8055725097656,"wires":[["2bd7
fc4a.094154"]]}]
```

3. After the import, you have nodes connected to one another (Figure 5-19).

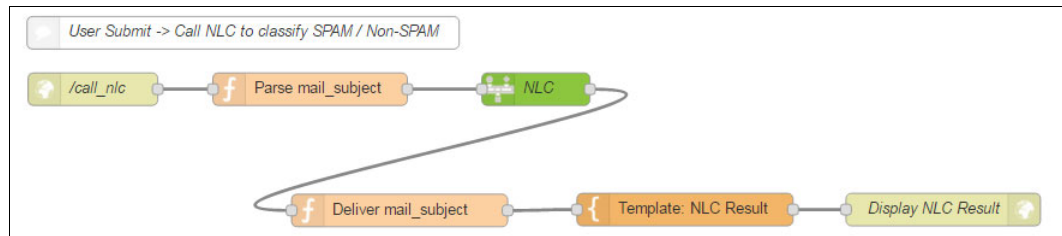


Figure 5-19 Nodes imported

4. Update the configuration information of the Natural Language Classifier node. Double-click the **NLC** node. When the editor opens (Figure 5-20), update the following fields (with the service you created) as in the example and click **Done**.

Username 1b2749fe-7581-42e2-ad3e-115c022ef8cd
 Password y1wcQL63akRX
 Classifier ID f5b42fx173-nlc-3980

Figure 5-20 Configuration of Natural Language Classifier

5. Click **Deploy** to apply changes.
6. For the user's request of classification, SPAM Classifier will classify the mail subject or content into SPAM or Non-SPAM, display the result, and ask if the user agrees with the results (Figure 5-21).

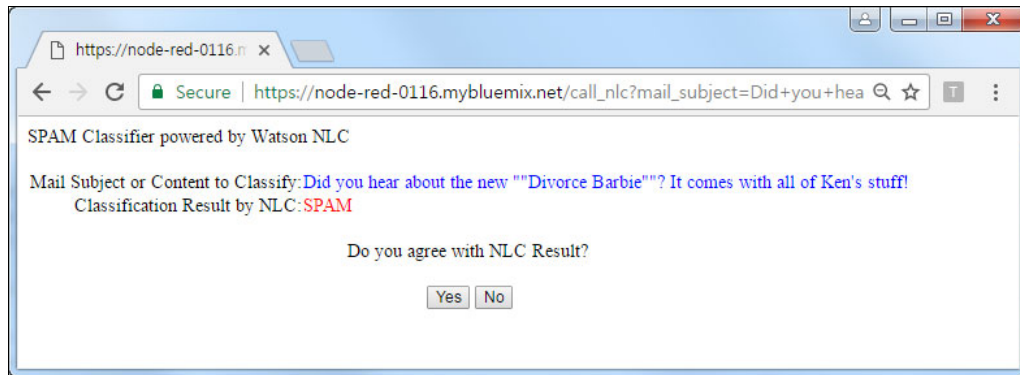


Figure 5-21 Classification result displayed

7. After the user clicks **Yes** or **No**, the user feedback is saved to the Cloudant noSQL DB.

5.4.6 Evaluating results and updating training data

Every time a user provides feedback to the classification result, the feedback is saved to the Cloudant noSQL DB in Bluemix for additional training.

Save user feedback into Cloudant noSQL DB

To import feedback into the Node-RED environment, complete these steps:

1. In the Node-RED editor, click the top right menu and select **Import** → **Clipboard**.
2. The clipboard window opens. Copy the code (Example 5-4 is the code snippet to import). Paste the code and click **Import**.

Example 5-4 is the sample code to import. This snippet is part of the exported Node-RED flow file (`flow.json`), which is included in the project folder cloned by Git. After you import the entire content of `flow.json`, you get all nodes and the links between them.

Example 5-4 Sample code to import (flow.json)

```
[{"id":"bc4b6594.b53508","type":"http
in","z":"f1d9f81c.fa7428","name":"/update_feedback","url":"/update_feedback","method":"g
et","swaggerDoc":"","x":116.79165649414062,"y":586.8889465332031,"wires":[["b6021075.bf4
ef"]]}, {"id":"b6021075.bf4ef","type":"function","z":"f1d9f81c.fa7428","name":"Parse
user_feedback","func":"/**\n * Compose a string which is saved into cloudant\n * String
Format: mail_subject,classification_result,user_feedback\n * - mail_subject:
mail_subject user provided\n * - classification_result: either SPAM or Non-SPAM\n * -
user_feedback: either Y or N\n */\n\n// replace quotation with space to avoid error
while saving into cloudant\nvar mail_subject = msg.req.query.mail_subject.trim();\nif
(mail_subject.indexOf("\\\"") > -1) {\n    var arr = mail_subject.split("\\\"");\n
mail_subject = \"\";\n    for (var i=0; i<arr.length; i++) {\n        if (mail_subject
!= \"\") {\n            mail_subject + \" \";\n        }\n        mail_subject =
mail_subject + arr[i];\n    }\n}\nmsg.payload = \"    mail_subject\n    + \"\n    +
msg.req.query.classification_result\n    + \"\n    +
msg.req.query.user_feedback\n;\n\nreturn
msg;\n","outputs":1,"noerr":0,"x":385.7916564941406,"y":585.888916015625,"wires":[["d127
1eb8.a2cce","1092b881.601317"]]}, {"id":"20b581e7.42c8fe","type":"comment","z":"f1d9f81c.
fa7428","name":"User click Yes or No -> Update User Feedback into
Cloudant","info":"Format: Mail Subject, Classification Result, User Feedback\nExample:
\\\"Hi world !\\\", SPAM,
Y","x":241.79165649414062,"y":534.888916015625,"wires":[]}, {"id":"d1271eb8.a2cce","type"
:"template","z":"f1d9f81c.fa7428","name":"Alert and
Redirect","field":"payload","fieldType":"msg","format":"handlebars","syntax":"plain","te
mplate":"<html>\n<body>\n    <script>\n        alert(\"Your feedback was saved for later
training. Thank you.\")\n        location.href = \"/req_nlc\"\n
</script>\n</body>\n</html>","x":643.7326812744141,"y":646.2326965332031,"wires":[["a6bd
45dd.c0e7f8"]]}, {"id":"a6bd45dd.c0e7f8","type":"http
response","z":"f1d9f81c.fa7428","name":"Redirect to
/req_nlc","x":871.7916412353516,"y":646.0000915527344,"wires":[]}, {"id":"1092b881.601317
","type":"cloudant out","z":"f1d9f81c.fa7428","name":"User
Feedback","cloudant":"","database":"my_database","service":"node-red-0116-cloudantNoSQLD
B","payonly":true,"operation":"insert","x":633.7916412353516,"y":585.8889770507812,"wire
s":[]}]
```

3. After the import, you have nodes connected with one another (Figure 5-22).

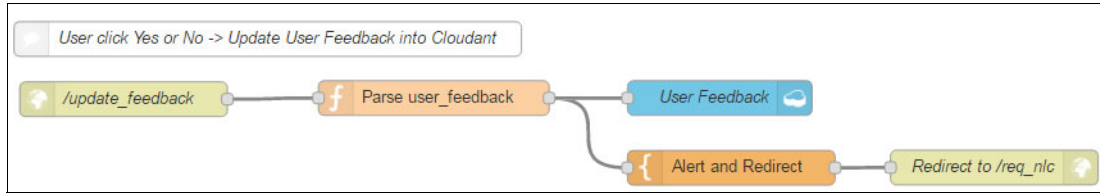


Figure 5-22 Nodes imported for feedback

4. Update the configuration information of the Cloudant *out* node. To do this, double-click the **User Feedback** node. When the editor opens (Figure 5-23), update the following fields as in the example and click **Done**.

Service node-red-0116-cloudantNoSQLDB
Database my_database

The screenshot shows the configuration dialog for a Cloudant out node. The fields are as follows:

- Service: node-red-0116-cloudantNoSQLDB
- Database: my_database
- Operation: insert
- Only store msg.payload object?:
- Name: User Feedback

Figure 5-23 Configuration of Cloudant out

5. Click **Deploy** to apply changes.

Feedback format

Each feedback is a one-line string composed of the following three items separated by a comma.

- ▶ Mail subject or content that user provided
- ▶ Classification result, either SPAM or Non-SPAM
- ▶ User feedback, either Y or N (for Yes or No)

For example, if user provides the text `Hi , Worl d` as mail subject or content, the classifier will classify it as Non-SPAM, and if the user agrees with the classification result, the feedback string would be:

`Hi , Worl d, Non-SPAM, Y`

Review user feedback

As users keep providing feedback, The feedback data is store in the Cloudant DB. You should review the Cloudant DB data periodically and create new training data to improve the classifier performance.

Complete the following steps to import the code snippet for reviewing user feedback into the Node-RED environment:

1. In the Node-RED editor, click the top right menu and select **Import** → **Clipboard**.
2. The clipboard window opens. Copy the code (Example 5-5 is the code snippet to import). Paste the code and click **Import**.

Example 5-5 is the sample code to import. This snippet is part of the exported Node-RED flow file (`flow.json`), which is included in the project folder cloned by Git. After you import the entire content of `flow.json`, you get all nodes and the links between them.

Example 5-5 Sample code to import (flow.json)

```
[{"id":"4c877c19.f98864","type":"http
in","z":"f1d9f81c.fa7428","name":"/get_feedback","url":"/get_feedback","method":"get","s
waggerDoc":"","x":95.89579772949219,"y":838.000244140625,"wires":[["a49abc66.27aa4"]]},{"
id":"9fa9b6f4.e7e938","type":"template","z":"f1d9f81c.fa7428","name":"Template: User
Feedback","field":"payload","fieldType":"msg","format":"handlebars","syntax":"mustache",
"template":"<html>\n<body>\nSPAM Classifier powered by Watson NLC\n<br><br>\n\n<table
border=0>\n\n<tr>\n<td valign=top>\n\tUser Feedback:\n</td>\n<td>\n<textarea cols=80
rows=20>\n{payload}\n</textarea>\n</td>\n</tr>\n\n</table>\n\n</body>\n</html>\n",
"x":513.8957977294922,"y":939.000244140625,"wires":[["747d6044.05772"]]},{"id":"747d6044.057
72","type":"http
response","z":"f1d9f81c.fa7428","name":"","x":746.8957977294922,"y":938.000244140625,"wi
res":[]},{"id":"a49abc66.27aa4","type":"cloudant in","z":"f1d9f81c.fa7428","name":"User
Feedback","cloudant":"","database":"my_database","service":"node-red-0116-cloudantNoSQLD
B","search":"_all_","design":"","index":"","x":355.8957977294922,"y":837.000244140625,"w
ires":[["e33f4f6e.4da6f"]]},{"id":"e33f4f6e.4da6f","type":"function","z":"f1d9f81c.fa742
8","name":" Collect records","func":"/**\n * Collect user feedback records from
cloudant\n * Record format: mail_subject,classification_result,user_feedback\n * -
mail_subject: mail_subject user provided\n * - classification_result: either SPAM or
Non-SPAM\n * - user_feedback: either Y or N\n */\n\nvar len = msg.payload.length;\nvar
newPayload = \"\";\nfor (var i=0; i<len; i++) {\n  if (newPayload != \"\") {\n
newPayload = newPayload + \"\\n\";\n  }\n  \n  var str =
msg.payload[i].payload.toString();\n  \n  // Enclose mail_subject with quotation if
it contains comma (,)\n  var arr = str.split(\"\\,\\\");\n  if (arr.length > 3) {\n
str = \n      parse_mail_subject(arr)\n      + \"\\,\\n\"
+
arr[arr.length-2]\n      + \"\\,\\n\"
+ arr[arr.length-1];\n  }\n  \n
newPayload = newPayload + str;\n}\n\nmsg.payload = newPayload;\nreturn
msg;\n\n\nfunction parse_mail_subject(arr) {\n  \n  var str = \"\";\n  for (var
i=0; i<arr.length-2; i++) {\n    if (str != \"\") {\n      str = str +
\"\\,\\n\";\n    }\n    str = str + arr[i];\n  }\n  str = \"\\\"\\\"\" + str +
\"\\\"\\\"\";\n  \n  return
str;\n}"},"outputs":1,"noerr":0,"x":573.8957977294922,"y":837.000244140625,"wires":[["9
fa9b6f4.e7e938"]]},{"id":"4834bed2.bf2be","type":"comment","z":"f1d9f81c.fa7428","name":
"Get User Feedback from Cloudant","info":"For later
training","x":159.8957977294922,"y":789.000244140625,"wires":[]}]
```

3. After the import, now you have nodes connected with one another (Figure 5-24).

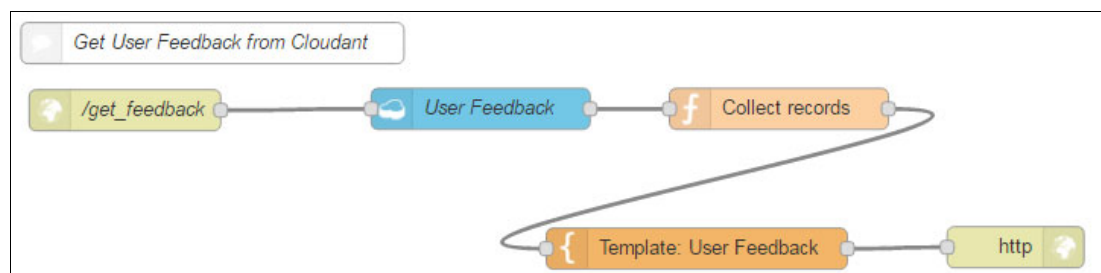


Figure 5-24 Nodes imported

- Update the configuration information of the Cloudant *in* node. To do this, double-click the **User Feedback** node. When the editor opens (Figure 5-25), update the following fields as in the example and click **Done**.

Service node-red-0116-c1oudantNoSQLDB
Database my_database

Figure 5-25 Configuration of Cloudant *in*

- Click **Deploy** to apply changes.
- When the user requests to review the feedback, SPAM Classifier displays the user feedback from the Cloudant DB (Figure 5-26).

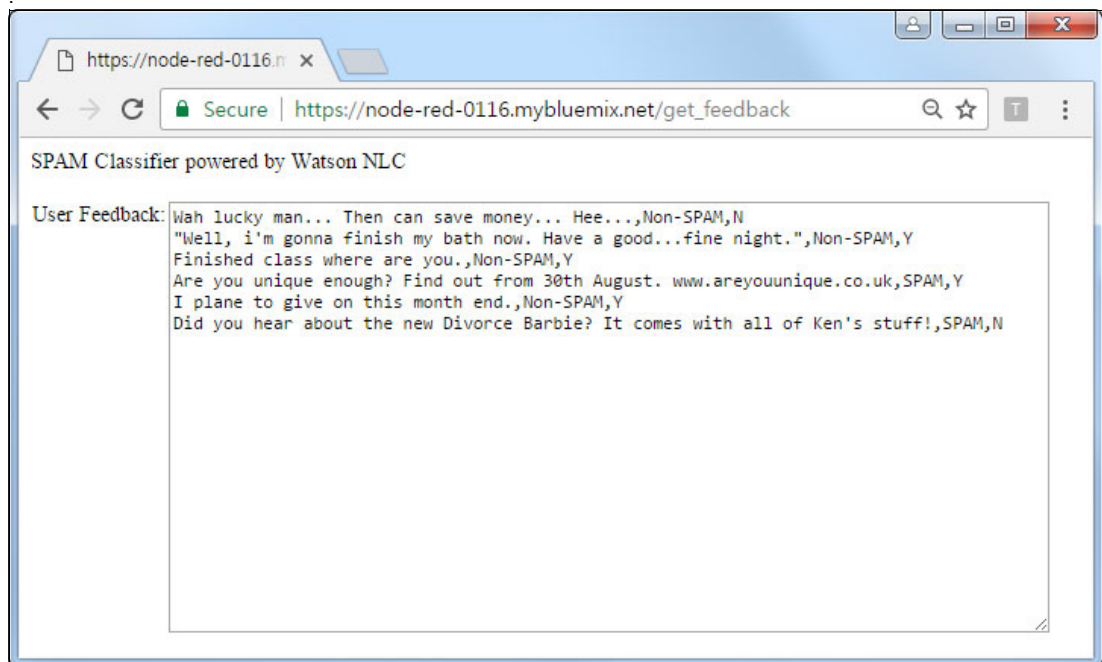


Figure 5-26 User feedback displayed

5.5 Quick deployment of application

As described in 5.3, “Two ways to deploy the application: Step-by-step and quick deploy” on page 106, a Git repository containing the full application code is provided so that you can run the application with minimal steps and more quickly. Here you can create the necessary Natural Language Classifier service, Cloudant noSQL DB service, and Node-RED development environment instead of following the detailed described in 5.4, “Step-by-step implementation” on page 106.

Complete these steps to deploy the application more quickly:

1. You need a [Bluemix](#) account. If you do not have one, create one.
2. Install Git and Cloud Foundry in your local computer.
3. Open a command prompt and set up Git using the following command:

```
git config --global http.sslVerify false
```

4. Choose an empty directory to download the code.
5. Run the command in the selected directory:

```
git clone https://github.com/snippet-java/redbooks-nlc-201-spam-nodered.git
```

6. Change to the nlc-201-spam-nodered directory:

```
cd nlc-201-spam-nodered
```

7. In the root application directory, run the **cf login** command and replace <BLUEMIX_EMAIL> and <BLUEMIX_PASSWORD> with your Bluemix account information, and select an organization (<ORG_NO>) to use (Example 5-6).

Example 5-6 Execute login and set email and password

```
cf login
API endpoint>: https://api.ng.bluemix.net
Email> <BLUEMIX_EMAIL>
Password> <BLUEMIX_PASSWORD>
Authenticating...
OK
Select an org (or press enter to skip)
1. sample_org1
2. sample_org2
org> <ORG_NO>
```

8. Select a Bluemix space to host the application (Example 5-7).

Example 5-7 Select Bluemix space

```
Select a space (or press enter to skip):
1. dev
2. qa
3. Prod
Space> 1
Targeted space dev
API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: < BLUEMIX_EMAIL>
Org: < BLUEMIX_PASSWORD>
Space: dev
```

9. Create the Natural Language Classifier service:

```
cf create-service natural_language_classifier standard my-nlc-service
```

10. Create service keys to access the Natural Language Classifier service:
`cf create-service-key my-nlc-service myKey`
11. Retrieve the service keys from the Natural Language Classifier service:
`cf service-key my-nlc-service myKey`
12. Create a Cloudant noSQL DB service for the feedback function of this application:
`cf create-service CloudantNoSQLDB Lite "SPAM Feedback"`
13. Create a classifier instance with initial training data. Follow the steps in 5.4.4, "Creating and training the classifier" on page 110. Training data is available in the resources folder.
14. In the `manifest.yml` file, update `<My_Name>` and `<My_Host>` with your unique values (Example 5-8).

Example 5-8 Update manifest.yml file

```

applications:
- path: .
  memory: 512M
  instances: 1
  domain: mybluemix.net
  name: <My_Name>
  host: <My_Host>
  disk_quota: 1024M
  services:
  - <My_Name>-cloudantNoSQLDB
  env:
    NODE_RED_STORAGE_NAME: <My_Name>-cloudantNoSQLDB
declared-services:
<My_Name>-cloudantNoSQLDB:
  label: cloudantNoSQLDB
  plan: Lite

```

15. Node-RED needs a Service of Cloudant NoSQL DB for storage. Create a service before you push the application to Bluemix:
`cf create-service CloudantNoSQLDB Lite "<My_Name>-cloudantNoSQLDB"`
16. Now, you can push the application to Bluemix:
`cf push`
17. In Bluemix, after you enter the Node-RED environment you created, you can review the default flows developed in Node-RED. However, you should update the configuration information of Natural Language Classifier node and Cloudant node with those you created.

5.6 References

See the following resources:

- ▶ Carmine, DiMascio. *Create a natural language classifier that identifies spam*. IBM developerWorks, 2016
<https://www.ibm.com/developerworks/library/cc-spam-classification-service-watson-nlc-bluemix-trs/index.html>
- ▶ Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. Contributions to the study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (ACM DOCENG'11), Mountain View, CA, USA, 2011.
<http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>



Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The following Git repositories are available to help you with the examples in these chapters:

- ▶ Chapter 3, “Healthcare questions and answers” on page 19:
 - For the *incomplete* code (step-by-step implementation version):
<https://github.com/snippet-java/redbooks-nlc-201-healthcare-nodejs-student.git>
 - For the *complete* code (quick deployment version) that you can use for verification or as a code reference:
<https://github.com/snippet-java/redbooks-nlc-201-healthcare-nodejs.git>
- ▶ Chapter 4, “News Classification” on page 45:
 - For the *incomplete* code (step-by-step implementation version):
<https://github.com/snippet-java/redbooks-nlc-201-news-java-student.git>
 - For the *complete* code (quick deployment version):
<https://github.com/snippet-java/redbooks-nlc-201-news-java.git>
- ▶ Chapter 5, “SPAM Classifier” on page 101:
 - For the *incomplete* code (step-by-step implementation version):
<https://github.com/snippet-java/redbooks-nlc-201-spam-nodered-student.git>
 - For the *complete* code (quick deployment version):
<https://github.com/snippet-java/redbooks-nlc-201-spam-nodered>

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

The volumes in the *Building Cognitive Applications with IBM Watson APIs* series:

- ▶ *Volume 1 Getting Started*, SG24-8387
- ▶ *Volume 2 Conversation*, SG24-8394
- ▶ *Volume 3 Visual Recognition*, SG24-8393
- ▶ *Volume 4 Natural Language Classifier*, SG24-8391
- ▶ *Volume 5 Language Translator*, SG24-8392
- ▶ *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- ▶ *Volume 7 Natural Language Understanding*, SG24-8398

You can search for, view, download or order these documents and other Redbooks, Redpapers™, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ▶ IBM Bluemix; create an account or log in:
<https://console.ng.bluemix.net>
- ▶ Cloud Foundry software download and install:
<https://github.com/cloudfoundry/cli/releases>
- ▶ Healthcare Q and A Application to see a running version:
<http://hcqanaturalanguageclassifier.mybluemix.net/>
- ▶ IBM SDK, Java Technology Edition, Version 8 download:
<https://developer.ibm.com/javasdk/downloads/>
- ▶ Git client downloads and installation:
<https://git-scm.com/downloads>
- ▶ News Classification web application to see a running version:
<https://nlc-201-news-java.mybluemix.net/>
- ▶ Eclipse Neon with Bluemix tools; install and set up:
<https://www.ibm.com/cloud-computing/bluemix/eclipse>

- ▶ Application server hosting for enterprise apps:
 - WebSphere Liberty:
<https://developer.ibm.com/wasdev/websphere-liberty/>
 - Apache Tomcat:
<http://tomcat.apache.org/>
- ▶ Using your own data to train the Natural Language Classifier (Watson Developer Cloud):
<https://www.ibm.com/watson/developercloud/doc/natural-language-classifier/using-your-data.html>
- ▶ Creating a classifier (Watson Developer Cloud):
https://www.ibm.com/watson/developercloud/natural-language-classifier/api/v1/#create_classifier
- ▶ Community buildpacks list:
<https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks#community-created>
- ▶ SPAM Classifier application to see a running version:
https://node-red-0116.mybluemix.net/req_nlc

Also see the list of online resources for the following chapters in this book:

- ▶ Basics of Natural Language Classifier API: 1.2, “References” on page 10
- ▶ Healthcare Questions and Answers: 3.6, “References” on page 42
- ▶ News Classification: 4.6, “References” on page 99
- ▶ SPAM Classifier: 5.6, “References” on page 122

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

(0.2"spine)

0.17"->0.473"

90->249 pages



SG24-8391-00

ISBN 0738442593

Printed in U.S.A.

Get connected

