Chapter 5: Natural Language Classifiers

Learning Bluemix & Cognitive

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales

IBM

Chapter 4: Classifying short phrases

- What is a classifier?
- Creating classifiers
  - Enabling the service
  - Creating the csv file
  - Creating the classifier
  - Telling your application how to access the classifier
- Looking at the classification results

What is the Watson NLC Service?

- The IBM Watson Natural Language Classifier service applies deep learning techniques to make predictions about the best predefined classes for short sentences or phrases, **generally 1000 characters or fewer**. The classes can trigger a corresponding action in an application, such as directing a request to a location or person, or answering a question. After training, the service returns information for texts that it hasn't seen before. The response includes the name of the top classes and confidence values.

- The service is trained by supplying a text (comma-separated-values) file with a phrase and then the target classification. A partial example from an industry classification file follows:

797 char

A&D businesses, AerospacDefense

A&D industry, AerospacDefense

A&D M&A, AerospacDefense

A&D products, AerospacDefense

aerospace prime contractors/, AerospacDefense

aftermarket service offerings, AerospacDefense

air travel, AerospacDefense

aircraft industry, AerospacDefense

airline travel, AerospacDefense

asset availability, AerospacDefense

asset related data, AerospacDefense

budget deficits, AerospacDefense

commercial acquisition practices, AerospacDefense
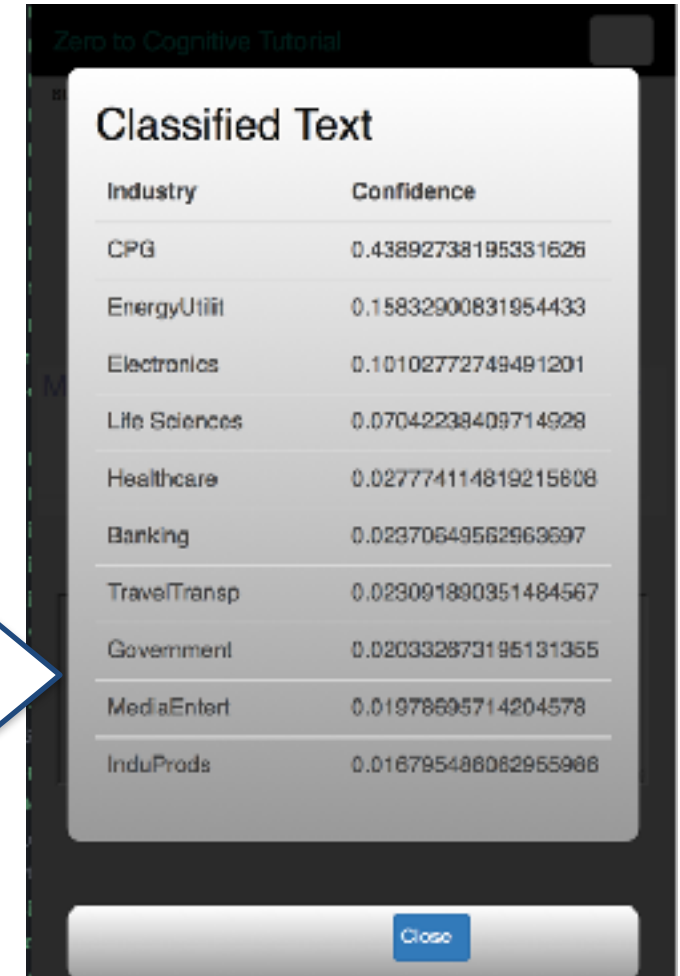
commercial aircraft industry, AerospacDefense

complete products, AerospacDefense

complex value chains, AerospacDefense

contractors/ system integrators, AerospacDefense

corporate support functions, AerospacDefense

defense industry, AerospacDefense

defense prime contractors/, AerospacDefense

development costs, AerospacDefense

domestic air travel, AerospacDefense

first-tier subcontractors, AerospacDefense

fourth-tier subcontractors, AerospacDefense

fuel costs, AerospacDefense

fuel-efficient aircraft, AerospacDefense

fundamental A&D research, AerospacDefense

global supply chains, AerospacDefense

We will enhance our original app to display the results from classifying your spoken words.
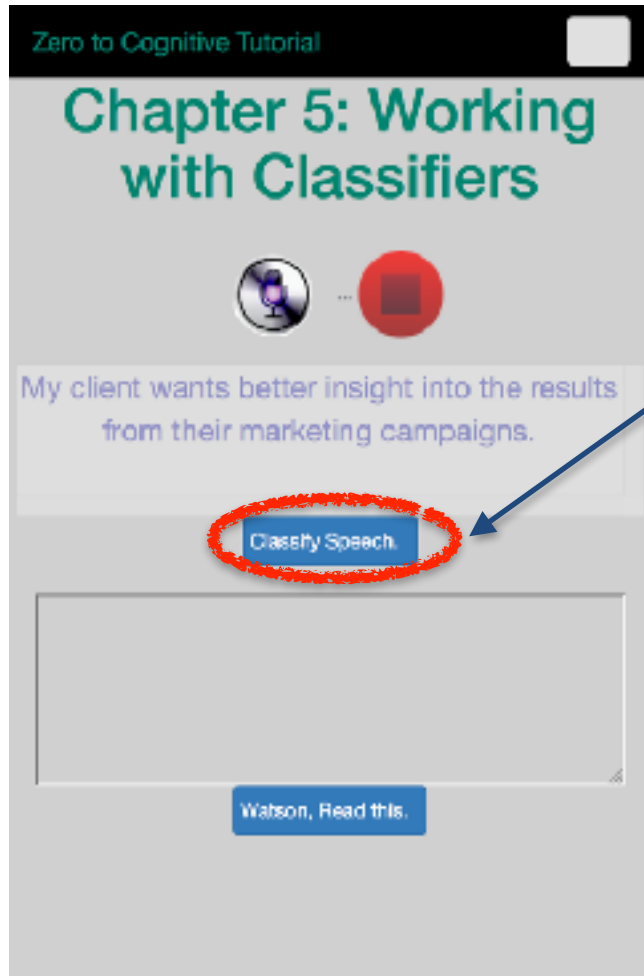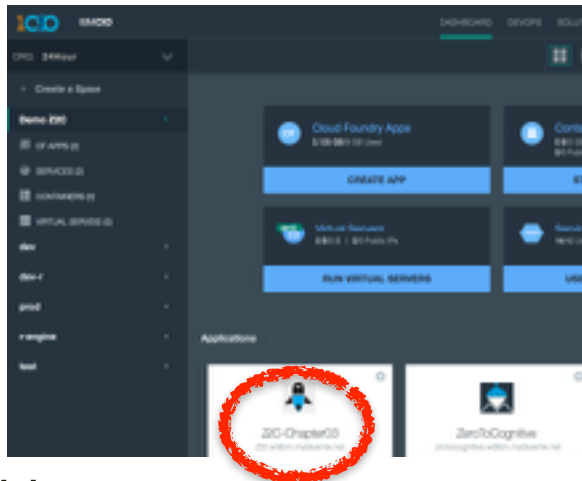

- We will add a "classify" button to the index.html file.

- Which will create a pop-up page to display the results of the Natural Language Classification
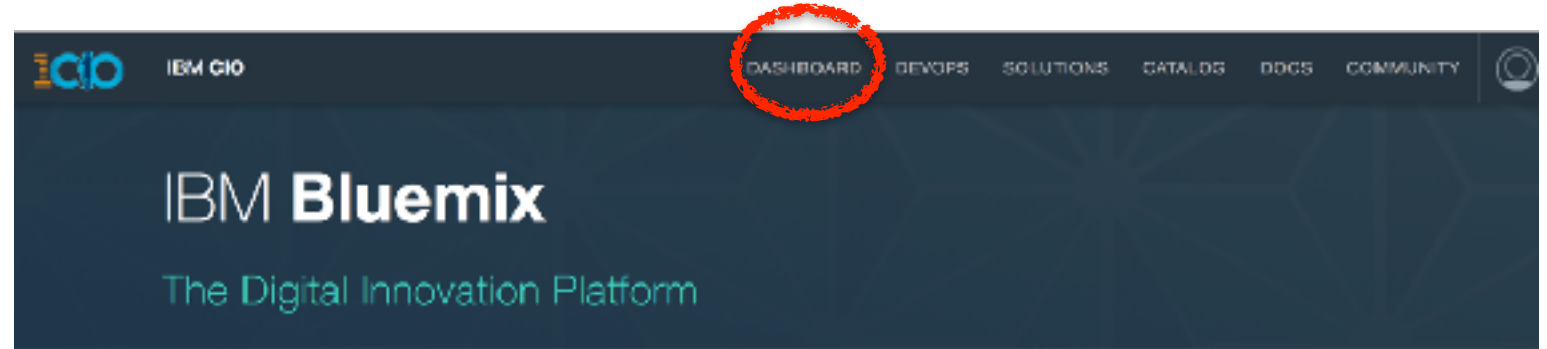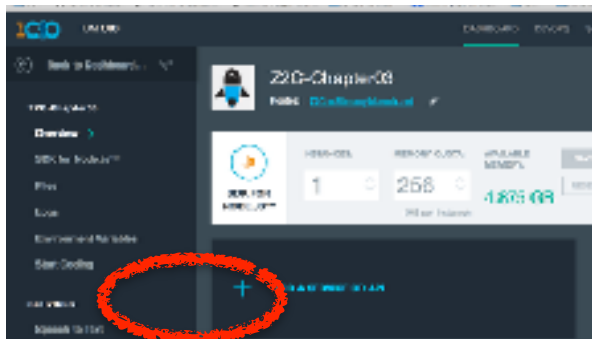
Watson NLC

- Add the service:

  - Go to your Bluemix dashboard and to your app.

    - Select Watson Natural Language Classifiers

  - Add a service or API

    - Get your credentials

orporation
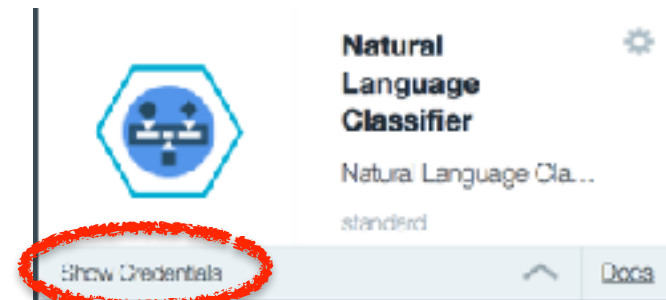
Update your credentials file

- By now, this should be looking very familiar.
  - open your env.json file and add your credentials for Natural Language Classification

```
16        "watson_nlc": {
17            "version": "v1",
18            "url": "https://gateway.watsonplatform.net/natural-language-classifier/api",
19            "password": "your NLC pasword",
20            "username": "your NLC user name",
21            "localJSON": true
22        }
```

Create a new restful service to do the classification

- Continuing in our theme of keeping unique parts of the program separate, we will create a new file for our classification. We have to add two lines to our 'router.js' file so that it can find the new services.

```
5   var classifier = require('./features/classifier');
```

```
12  // classify using NLC
13  router.post('/api/understand/classifyInd*', classifier.classifyInd);
```

- The classifier.js file will have 2 functions in it and the same kind of require statements as we have previously used.
- lines 1-5 define required resources
- line 7 gets the execution environment
- lines 9-10 set up NC
- line 11 defines the classifier we will use.

```
1   var extend = require('extend');
2   var cfenv = require('cfenv');
3   var watson = require('watson-developer-cloud');
4   var WATSON_NLC_SERVICE_NAME = "Watson-NLC-Service";
5   var config = require("../../env.json");
6
7   var appEnv = cfenv.getAppEnv();
8
9   var serviceCreds = appEnv.getServiceCreds(WATSON_NLC_SERVICE_NAME) || process.env.NLC_CREDS ||
10  var natural_language_classifier = watson.natural_language_classifier(serviceCreds);
11  var classifier_id_industry = process.env.NLC_CLASSIFIER_ID || 'your-classifier-id';
```

Creating the restful service (II)

- Line 12 names the exported service, This is what the router sees
- Line 13 gets the text from the post
- Line 15 defines an anonymous function, which on
- Line 16 calls the classifier and
- Line 20 returns the results which are
- Line 22 sent to a formatting function

- Line 29 check if the call was successful
- Lines 30-31 format an error message
- Lines 32-34 else format the classification
- Line 37 format and send the results back (note that JSON.stringify was called twice)

```
12  exports.classifyInd = function(req, res) {
13      _text = req.body.cquery;
14      i_output = {};
15      (function(_res) {
16          natural_language_classifier.classify({
17              text: _text,
18              classifier_id: classifier_id_industry
19          },
20          function(err, response) {
21              _res.writeHead(200, { "Content-Type": "text/plain" });
22              _res.end(nlc_res("Industry", err, response));
23          });
24      })(res)
25  }
```

```
27  function nlc_res(classifier, err, response) {
28      var _output = [];
29      if (err) {
30          console.log(classifier + ' error:', err);
31          _output = { 'error': JSON.stringify(err, null, 2) };
32      } else {
33          _output = { results: JSON.stringify(response, null, 2) };
34      }
35      console.log("Printing from nlc_res");
36      console.log(_output);
37      return (JSON.stringify(_output, null, 2));
```

curl on your local machine

- "curl", or Command-line URL, is a utility on your laptops.
- **Windows**:
    - Open the git bash shell (this was installed when you installed git on your laptop).
    - The git bash shell looks like a windows command prompt with unusual colors.
    - Execute the curl commands from within the git bash shell
- **OSX**
    - If you don't already have curl installed, I recommend that you use the "homebrew" service.
    - If you don't already have home-brew installed, go here: http://brew.sh/ and follow the directions. Once you have home-brew installed, type this command in a terminal window:
    brew install curl and press enter.
    - you can now execute curl commands from a terminal window
- **Linux**
    - If you don't already have curl installed, type the following command in a terminal window:
    sudo apt-get install curl and press enter.
    provide your laptop log in password when requested.
    - you can now execute curl commands from a terminal.

Creating your own Natural Language Classifier

To create a classifier, you need a .csv file with text strings and classification terms. This Chapter has an industry classifier which was used as part of another project. It is called "industry.csv".

you can create a classifier either from within an application, or directly from your terminal/command prompt window. We will do this from a terminal window.

Enter the following command, using your user-id and password credentials from Watson NLC:

- Change the to Chapter05/Documentation folder (cd ~/Documents/GitHub/ZeroToCognitive/Chapter05/Documentation)
- curl -u "**your user name**":"**your password**" -F training_data=@industry.csv -F training_metadata="{\"language\":\"en\", \"name\":\"Industry2016\"}" "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers"
- Write down the classifier_id value returned by Watson when you enter this command and put this into line 11 in your classifier.js file

Check the status of your classifier by entering the following command:

- curl -u "**your user name**":"**your password**" "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/**your-NLC-Classifier-id**"
- You can use your classifier when the returned status shows that the process is complete. This typically takes 10-15 minutes.

List all of your classifiers:

- curl -u "**your user name**":"**your password**" "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers"

Delete a specific classifier:

- curl -X DELETE -u "**your user name**":"**your password**" "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/**your-NLC-Classifier-id**"

Using the results from the classifier

- Watson NLC returns a JSON object with the results of the classification. The part of the object about which we care is the "classes" array. Everything else is interesting, but we don't need it for what we're doing.

- The classes array has up to ten 2-value pairs. The first value is the name of the classifier (in our case, an industry name) and the second has the probability that this is the correct match.

- Our browser-side javascript has to do 3 things:

  1. Activate the "classify" button

  2. Get the text to be classified and send it to the restful service

  3. Get the results from the restful service and display them.

Update index.html

- Keeping our principle of 'separation of concerns' in mind, we're going to make a very small modification to the index.html file and then create a new html file to display the results of the classification.

- We need to add a button to the index.html file and we will add a second javascript file as well:

- we will add 4 lines of code.

  - Row 32 creates a placeholder for the modal dialog

  -
    ```
    31              <div class="container", id="top" >
    32              <div id="modal"></div>
    ```

  - Rows 49 and 51 just specify that this is a "row" kind of an addition. Line 50 is the actual button.

```
49              <div class="row">
50                  <div class="col-md-6"><center><a id="classifySpeech" class="btn btn-primary"
                    style="padding-left: 0.3em">Classify Speech.</a></center></div>
51              </div>
```

  - And we will put all of the javascript code associated with NLC into a new file.

```
75          <script src="js/z2c-NLC.js"></script>
```

Activate the classify button

- We're only going to add a couple of lines to the initPage function. The body of the logic associated with the new button will go into the new browser javascript file.

```
74    displayNLC.on("click",  function()
75    {
76        var nlcPage = "displayNLC.html";
77        checkNLC(nlcPage, stt_out);
78    });
79  }
```

- We will create the 'checkNLC' function in the z2c-NLC.js file.

Browser: Display the NLC results (2)

- The NLC results are displayed on a pop-up page using a combination of an HTML page fragment and some CSS coding.

- The two key elements on this page are lines 7-8 and line 14.

- Lines 7-8 define the table for displaying the results

- Line 14 has the button we'll use to close the window.

```
2    <div id="modal_NLC" class="modalDialog">
3      <div>
         <h2>Classified Text</h2>
5        <div class="row">
6          <div class="col-lg-12">
7            <table class="table" id="industryResult">
8              <thead> <tr> <th>Industry</th> <th>Confidence</th></tr> </thead>
9              <tbody><tr></tr> </tbody> </table>
10         </div>
11       </div>
12     </div>
13     <div class="row">
14       <div class="col-md-6"><center><a id="close_NLC" class="btn btn-primary"
15     </div>
16   </div>
```

```
3    .modalDialog {
4      position: fixed;
5      font-family: Arial, Helvetica, sans-serif;
6      top: 0; right: 0; bottom: 0; left: 0;
7      background: rgba(0,0,0,.8); color: #0F0F0F;
8      z-index: 99999;
9      opacity:1;
10     -webkit-transition: opacity 400ms ease-in;
11     -moz-transition: opacity 400ms ease-in;
12     transition: opacity 400ms ease-in;
13   }
14
15   .modalDialog > div {
16     width: 400px; position: relative; margin: 10% auto;
17     padding: 5px 20px 13px 20px;
18     border-radius: 10px;
19     background: #fff;
20     background: -moz-linear-gradient(#fff, #999);
21     background: -webkit-linear-gradient(#fff, #999);
22     background: -o-linear-gradient(#fff, #999);
23   }
```

- The CSS to the left has two sections. The upper controls the black background for the pop-up window, the lower controls how the table with the NLC results will be displayed.

Browser: Activate the button, get text, send it to the server

- Activation is handled in the 'displayNLC.on' code written earlier

- Line 2 defines the function

- Line 5 gets the text to be analyzed
  and places it in an object

```
1   // displayNLC_results
2   function checkNLC(_display, _source)
3   {
4       var options = {};
5       options.cquery = $(_source)[0].innerText;
6       $.when($.get(_display), $.post('/api/understand/classifyInd', options)).done(function(_page,
        _nlc_results){
```

- Line 6 which is used in a post to the server.

- Line 6 also retrieves an html page fragment (html file) which will be used to display the results from the classification process. The jQuery $.when().done() function allows us to send a series of requests in parallel and then use the combined results when all server calls have completed. The results are returned in the variables defined at the end of line 6 (_page and _nlc_results)

- We're creating a modal (can't do anything else until this window is closed) window to display the NLC results.

- lines 9-11 take the retrieved html template and display it

```
8       var _target= $("#modal");
9       _target.append(_page);
10      _target.height($(window).height());
11      _target.show();
```

- Lines 12-13 extract the classes from the NLC JSON object

- Line 14 calls a formatting routine

- Lines 15-18 activate the close button for this window

```
12      _data = _nlc_results[0];
13      _classes = JSON.parse(JSON.parse(_data).results).classes;
14      displayNLC($("#industryResult"), _classes);
15      closeNLC=$("#close_NLC");
16      closeNLC.on("click", function(){
17          console.log("closeNLC clicked.")
18          $("#modal").empty();
```

Browser: Display the NLC results (1)

- Line 23 defines the function
- Line 25 empties the table
- Line 28 initializes a variable for the while loop
- Line 29 loops through all of the returned classifiers
- Lines 32-35 process the data for the loop
  - we're using an anonymous function
  - this is highly recommended whenever loops are processed in Javascript.
  - The loop is passed a counter and the classes table on line 35
  - Line 34 uses those values in a 'safe' space to create the table

```javascript
23  function displayNLC(_target, _results)
24  {
25      var target = _target;
26      target.find("tr:not(:first)").remove();
27      indName = _results[0]["class_name"];
28      var len = _results.length;
29      _idx = 0;
30      while (_idx < len)
31      {
32          (function(_idx, data)
33              {  _cStr = data[_idx]["class_name"];
34                  target.append("<tr><td style='width: 60%'>"+_cStr
                    +"</td><td>"+data[_idx]["confidence"]+"</td></tr>");})
35              (_idx, _results)
36          _idx++;
37      }
38      target.append("</table>");
39  }
```

The Plan: 30 minute Chapters with an hour or two of practice

1. The Story, Architecture for this app
2. Setting up Bluemix
3. Building your first Watson App   (Watson Speech to Text)
4. Getting Watson to talk back   (Watson Text to Speech)
5. Understanding Classifiers   (Watson NLC)
6. Creating a custom dialog with Watson   (custom Q&A, session management)
7. Authentication   (puts C2 thru 6 together)
8. Alchemy News   (Watson Alchemy)
9. Visual Recognition and Images   (Watson Visual Recognition)
10. Watson Conversations   (Watson Conversations)
11. Rank & Retrieve   (Watson Alchemy + Rank & Retrieve)
12. Getting started on my first client prototype   Design Thinking, Stories, Architecture, Keeping it simple

Chapter 6: Understanding Classifiers

Learning Bluemix & Cognitive

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales

IBM