Extract custom entities

In addition to other natural language processing capabilities, Azure AI Language Service enables you to define custom entities, and extract instances of them from text.

To test the custom entity extraction, we'll create a model and train it through Azure AI Language Studio, then use a command line application to test it.

Provision an *Azure AI Language* resource

If you don't already have one in your subscription, you'll need to provision an **Azure AI Language service** resource. Additionally, use custom text classification, you need to enable the **Custom text classification & extraction** feature.

1. In a browser, open the Azure portal at `https://portal.azure.com`, and sign in with your Microsoft account.
2. Select the **Create a resource** button, search for *Language*, and create an **Azure AI Language Service** resource. When asked about *Additional features*, select **Custom text classification & extraction**. Create the resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Select or create a resource group*
    - **Region**: *Choose any available region*
    - **Name**: *Enter a unique name*
    - **Pricing tier**: Select **F0** (*free*), or **S** (*standard*) if F is not available.
    - **Storage account**: New storage account:
        - **Storage account name**: *Enter a unique name*.
        - **Storage account type**: Standard LRS
    - **Responsible AI notice**: Selected.
3. Select **Review + create,** then select **Create** to provision the resource.
4. Wait for deployment to complete, and then go to the deployed resource.
5. View the **Keys and Endpoint** page. You will need the information on this page later in the exercise.

Upload sample ads

After you've created the Azure AI Language Service and storage account, you'll need to upload example ads to train your model later.

1. In a new browser tab, download sample classified ads from `https://aka.ms/entity-extraction-ads` and extract the files to a folder of your choice.
2. In the Azure portal, navigate to the storage account you created, and select it.
3. In your storage account select **Configuration**, located below **Settings**, and screen enable the option to **Allow Blob anonymous access** then select **Save**.
4. Select **Containers** from the left menu, located below **Data storage**. On the screen that appears, select **+ Container**. Give the container the name `classifieds`, and set **Anonymous access level** to **Container (anonymous read access for containers and blobs)**.

   > **NOTE**: When you configure a storage account for a real solution, be careful to assign the appropriate access level. To learn more about each access level, see the Azure Storage documentation.

5. After creating the container, select it and click the **Upload** button and upload the sample ads you downloaded.

Create a custom named entity recognition project

Now you're ready to reate a custom named entity recognition project. This project provides a working place to build, train, and deploy your model.

> **NOTE**: You can also create, build, train, and deploy your model through the REST API.

1. In a new browser tab, open the Azure AI Language Studio portal at `https://language.cognitive.azure.com/` and sign in using the Microsoft account associated with your Azure subscription.
2. If prompted to choose a Language resource, select the following settings:
   - **Azure Directory**: The Azure directory containing your subscription.
   - **Azure subscription**: Your Azure subscription.
   - **Resource type**: Language.
   - **Language resource**: The Azure AI Language resource you created previously.

   If you are not prompted to choose a language resource, it may be because you have multiple Language resources in your subscription; in which case:

e. On the bar at the top if the page, select the **Settings (⚙)** button.

   f. On the **Settings** page, view the **Resources** tab.

   g. Select the language resource you just created, and click **Switch resource**.

   h. At the top of the page, click **Language Studio** to return to the Language Studio home page

3. At the top of the portal, in the **Create new** menu, select *Custom named entity recognition**.
4. Create a mew project with the following settings:
    - **Connect storage**: *This value is likely already filled. Change it to your storage account if it isn't already*
    - **Basic information**:
    - **Name**: `CustomEntityLab`
        - **Text primary language**: English (US)
        - **Does your dataset include documents that are not in the same language?** : *No*
        - **Description**: `Custom entities in classified ads`
    - **Container**:
        - **Blob store container**: classifieds
        - **Are your files labeled with classes?**: No, I need to label my files as part of this project

Label your data

Now that your project is created, you need to label your data to train your model how to identity entities.

1. If the **Data labeling** page is not already open, in the pane on the left, select **Data labeling**. You'll see a list of the files you uploaded to your storage account.
2. On the right side, in the **Activity** pane, select **Add entity** and add a new entity named `ItemForSale`.
3. Repeat the previous stept o create the following entities:
    - `Price`
    - `Location`
4. After you've created your three entities, select **Ad 1.txt** so you can read it.
5. In *Ad 1.txt*:

a. Highlight the text *face cord of firewood* and select the **ItemForSale** entity.
   b. Highlight the text *Denver, CO* and select the **Location** entity.
   c. Highlight the text *$90* and select the **Price** entity. 1.In the **Activity** pane, note that this document will be added to the dataset for training the model.

6. Us the **Next document** button to move to the next document, and continue assigning text to appropriate entities for the entire set of documents, adding them all to the training dataset.
7. When you have labeled the last document (*Ad 9.txt*), save the labels.

Train your model

After you’ve labeled your data, you need to train your model.

1. Select **Training jobs** in the pane on the left.
2. Select **Start a training job**
3. Train a new model named `ExtractAds`
4. Choose **Automatically split the testing set from training data**

    TIP: In your own extraction projects, use the testing split that best suits your data. For more consistent data and larger datasets, the Azure AI Language Service will automatically split the testing set by percentage. With smaller datasets, it’s important to train with the right variety of possible input documents.

5. Click **Train**

    IMPORTANT: Training your model can sometimes take several minutes. You’ll get a notification when it’s complete.

Evaluate your model

In real world applications, it’s important to evaluate and improve your model to verify it’s performing as you expect. Two pages on the left show you the details of your trained model, and any testing that failed.

Select **Model performance** on the left side menu, and select your `ExtractAds` model. There you can see the scoring of your model, performance metrics, and when it was trained. You’ll be able to see if any testing documents failed, and these failures help you understand where to improve.

Deploy your model

When you’re satisfied with the training of your model, it’s time to deploy it, which allows you to start extracting entities through the API.

1. In the left pane, select **Deploying a model**.
2. Select **Add deployment**, then enter the name `AdEntities` and select the **ExtractAds** model.
3. Click **Deploy** to deploy your model.

Prepare to develop an app in Visual Studio Code

To test the custom entity extraction capabilities of the Azure AI Language service, you’ll develop a simple console application in Visual Studio Code.

Tip: If you have already cloned the **mslearn-ai-language** repo, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/mslearn-ai-language` repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.
4. Wait while additional files are installed to support the C# code projects in the repo.

   > **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

Configure your application

Applications for both C# and Python have been provided, as well as a sample text file you'll use to test the summarization. Both apps feature the same functionality. First, you'll complete some key parts of the application to enable it to use your Azure AI Language resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/05-custom-entity-recognition** folder and expand the **CSharp** or **Python** folder depending on your language preference and the **custom-entities** folder it contains. Each folder contains the language-specific files for an app into which you're you're going to integrate Azure AI Language text classification functionality.
2. Right-click the **custom-entities** folder containing your code files and open an integrated terminal. Then install the Azure AI Language Text Analytics SDK package by running the appropriate command for your language preference:

   C#:

   CodeCopy

   ```
   dotnet add package Azure.AI.TextAnalytics --version 5.3.0
   ```

   Python:

   CodeCopy

   ```
   pip install azure-ai-textanalytics==5.3.0
   ```

3. In the **Explorer** pane, in the **custom-entities** folder, open the configuration file for your preferred language

    - **C#**: appsettings.json
    - **Python**: .env
4. Update the configuration values to include the **endpoint** and a **key** from the Azure Language resource you created (available on the **Keys and Endpoint** page for your Azure AI Language resource in the Azure portal). The fil should already contain the project and deployment names for your custom entity extraction model.
5. Save the configuration file.

Add code to extract entities

Now you're ready to use the Azure AI Language service to extract custom entities from text.

1. Expand the **ads** folder in the **custom-entities** folder to view the classified ads that your application will analyze.
2. In the **custom-entities** folder, open the code file for the client application:
    - **C#**: Program.cs
    - **Python**: custom-entities.py
3. Find the comment **Import namespaces**. Then, under this comment, add the following language-specific code to import the namespaces you will need to use the Text Analytics SDK:

   C#: Programs.cs

   C#Copy

   ```
   // import namespaces
   using Azure;
   using Azure.AI.TextAnalytics;
   ```

   Python: custom-entities.py

   CodeCopy

   ```
   # import namespaces
   from azure.core.credentials import AzureKeyCredential
   from azure.ai.textanalytics import TextAnalyticsClient
   ```

4. In the **Main** function, note that code to load the Azure AI Language service endpoint and key and the project and deployment names from the configuration file has already been provided. Then find the comment **Create**

client using endpoint and key, and add the following code to create a client for the Text Analysis API:

C#: Programs.cs

C#Copy

```
 // Create client using endpoint and key
 AzureKeyCredential credentials = new(aiSvcKey);
 Uri endpoint = new(aiSvcEndpoint);
 TextAnalyticsClient aiClient = new(endpoint, credentials);
```

Python: custom-entities.py

CodeCopy

```
 # Create client using endpoint and key
 credential = AzureKeyCredential(ai_key)
 ai_client = TextAnalyticsClient(endpoint=ai_endpoint,
credential=credential)
```

5. in the **Main** function, note that the existing code reads all of the files in the **ads** folder and creates a list containing their contents. In the case of the C# code, the a list of **TextDocumentInput** objects is used to include the file name as an ID and the language. In Python a simple list of the text contents is used.
6. Find the comment **Extract entities** and add the following code:

C#: Program.cs

C#Copy

```
 // Extract entities
 RecognizeCustomEntitiesOperation operation = await
aiClient.RecognizeCustomEntitiesAsync(WaitUntil.Completed,
batchedDocuments, projectName, deploymentName);

 await foreach (RecognizeCustomEntitiesResultCollection documentsInPage in
operation.Value)
 {
     foreach (RecognizeEntitiesResult documentResult in documentsInPage)
     {
         Console.WriteLine($"Result for \"{documentResult.Id}\":");

         if (documentResult.HasError)
         {
             Console.WriteLine($"  Error!");
```

```
                Console.WriteLine($"  Document error code:
{documentResult.Error.ErrorCode}");
                Console.WriteLine($"  Message:
{documentResult.Error.Message}");
                Console.WriteLine();
                continue;
            }

            Console.WriteLine($"  Recognized {documentResult.Entities.Count}
entities:");

            foreach (CategorizedEntity entity in documentResult.Entities)
            {
                Console.WriteLine($"  Entity: {entity.Text}");
                Console.WriteLine($"  Category: {entity.Category}");
                Console.WriteLine($"  Offset: {entity.Offset}");
                Console.WriteLine($"  Length: {entity.Length}");
                Console.WriteLine($"  ConfidenceScore:
{entity.ConfidenceScore}");
                Console.WriteLine($"  SubCategory: {entity.SubCategory}");
                Console.WriteLine();
            }

            Console.WriteLine();
        }
 }
```

Python: custom-entities.py

CodeCopy

```
 # Extract entities
 operation = ai_client.begin_recognize_custom_entities(
     batchedDocuments,
     project_name=project_name,
     deployment_name=deployment_name
 )

 document_results = operation.result()

 for doc, custom_entities_result in zip(files, document_results):
     print(doc)
     if custom_entities_result.kind == "CustomEntityRecognition":
         for entity in custom_entities_result.entities:
             print(
                 "\tEntity '{}' has category '{}' with confidence score of
'{}'".format(
                     entity.text, entity.category, entity.confidence_score
                 )
             )
     elif custom_entities_result.is_error is True:
         print("\tError with code '{}' and message '{}'".format(
             custom_entities_result.error.code,
custom_entities_result.error.message
             )
```

```
            )
```

7. Save the changes to your code file.

Test your application

Now your application is ready to test.

1. In the integrated terminal for the **classify-text** folder, and enter the following command to run the program:
    - **C#**: `dotnet run`
    - **Python**: `python custom-entities.py`

    > **Tip**: You can use the **Maximize panel size** (**^**) icon in the terminal toolbar to see more of the console text.

2. Observe the output. The application should list details of the entities found in each text file.

Clean up

When you don't need your project anymore, you can delete if from your **Projects** page in Language Studio. You can also remove the Azure AI Language service and associated storage account in the Azure portal.