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 Python application to test it.

While this exercise is based on Python, you can develop text classification applications using multiple language-specific SDKs; including:

- Azure AI Text Analytics client library for Python
- Azure AI Text Analytics client library for .NET
- Azure AI Text Analytics client library for JavaScript

This exercise takes approximately **35** minutes.

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 a **Language Service** resource. When on the page for *Select additional features*, select the custom feature containing **Custom named entity recognition extraction**. Create the resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Select or create a resource group*
    - **Region**: *Choose from one of the following regions**
        - Australia East
        - Central India
        - East US
        - East US 2
        - North Europe
        - South Central US
        - Switzerland North
        - UK South
        - West Europe
        - West US 2
        - West US 3
    - **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.

Configure role-based access for your user

> **NOTE**: If you skip this step, you'll have a 403 error when trying to connect to your custom project. It's important that your current user has this role to access storage account blob data, even if you're the owner of the storage account.

1. Go to your storage account page in the Azure portal.
2. Select **Access Control (IAM)** in the left navigation menu.
3. Select **Add** to Add Role Assignments, and choose the **Storage Blob Data Contributor** role on the storage account.
4. Within **Assign access to**, select **User, group, or service principal**.
5. Select **Select members**.
6. Select your User. You can search for user names in the **Select** field.

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 create 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:

   a. On the bar at the top of the page, select the **Settings (⚙)** button.
   b. On the **Settings** page, view the **Resources** tab.

   c. Select the language resource you just created, and click **Switch resource**.
   d. 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 new 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

> **Tip**: If you get an error about not being authorized to perform this operation, you'll need to add a role assignment. To fix this, we add the role "Storage Blob Data Contributor" on the storage account for the user running the lab. More details can be found on the documentation page

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 step to 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.
6. In the **Activity** pane, note that this document will be added to the dataset for training the model.
7. Use 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.
8. 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 Cloud Shell

To test the custom entity extraction capabilities of the Azure AI Language service, you'll develop a simple console application in the Azure Cloud Shell.

1. In the Azure Portal, use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

   > **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

2. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

3. In the PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

   Code Copy

   ```
   rm -r mslearn-ai-language -f
   git clone https://github.com/microsoftlearning/mslearn-ai-language
   ```

   > **Tip**: As you paste commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

   Code Copy

```
```

4. After the repo has been cloned, navigate to the folder containing the application code files:

Code Copy

```
cd mslearn-ai-language/Labfiles/05-custom-entity-recognition/Python/custom-entities
```

Configure your application

1. In the command line pane, run the following command to view the code files in the **custom-entities** folder:

Code Copy

```
ls -a -l
```

The files include a configuration file (**.env**) and a code file (**custom-entities.py**). The text your application will analyze is in the **ads** subfolder.

2. Create a Python virtual environment and install the Azure AI Language Text Analytics SDK package and other required packages by running the following command:

Code Copy

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-ai-textanalytics==5.3.0
```

3. Enter the following command to edit the application configuration file:

Code Copy

```
code .env
```

The file is opened in a code editor.

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 file should already contain the project and deployment names for your custom entity extraction model.

5. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes and then use the **CTRL+Q** command or **Right-click > Quit** to close the code editor while keeping the cloud shell command line open.

Add code to extract entities

1. Enter the following command to edit the application code file:

Code Copy

```
code custom-entities.py
```

2. Review the existing code. You will add code to work with the AI Language Text Analytics SDK.

Tip: As you add code to the code file, be sure to maintain the correct indentation.

3. At the top of the code file, under the existing namespace references, find the comment **Import namespaces** and add the following code to import the namespaces you will need to use the Text Analytics SDK:

Code Copy

```
# 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 text analytics client:

Code Copy

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

5. Note that the existing code reads all of the files in the **ads** folder and creates a list containing their contents. Then find the comment **Extract entities** and add the following code:

Code Copy

```
# 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
            )
        )
```

6. Save your changes (CTRL+S), then enter the following command to run the program (you maximize the cloud shell pane and resize the panels to see more text in the command line pane):

Code Copy

```
python custom-entities.py
```

7. 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.