

## 3.5 - Custom named entity recognition

- [Overview](#)
  - [Learning objectives](#)
  - [Introduction](#)
  - [Understand custom named entity recognition](#)
    - [Custom vs built-in NER](#)
    - [Azure AI Language project life cycle](#)
    - [Considerations for data selection and refining entities](#)
    - [How to extract entities](#)
    - [Project limits](#)
  - [Label your data](#)
    - [How to label your data](#)
  - [Train and evaluate your model](#)
    - [How to interpret metrics](#)
    - [Confusion matrix](#)
  - [Exercise - Extract custom entities](#)
    - [Provision an Azure AI Language resource](#)
    - [Upload sample ads](#)
    - [Create a custom named entity recognition project](#)
    - [Label your data](#)
  - [Train your model](#)
    - [Evaluate your model](#)
    - [Deploy your model](#)
    - [Prepare to develop an app in Visual Studio Code](#)
    - [Configure your application](#)
    - [Add code to extract entities](#)
    - [Test your application](#)
    - [Clean up](#)
  - [Knowledge Check](#)
  - [Summary](#)
- 

### Overview

Build a custom entity recognition solution to extract entities from unstructured documents

### Learning objectives

After completing this module, you'll be able to:

- Understand tagging entities in extraction projects
  - Understand how to build entity recognition projects
-

# Introduction

Custom *named entity recognition* (NER), otherwise known as custom entity extraction, is one of the many features for *natural language processing* (NLP) offered by Azure AI Language service. Custom NER enables developers to extract predefined entities from text documents, without those documents being in a known format - such as legal agreements or online ads.

An entity is a person, place, thing, event, skill, or value.

In this module, you'll learn how to use the Azure AI Language service to extract entities from unstructured documents.

After completing this module, you'll be able to:

- Understand custom named entities and how they're labeled.
- Build a custom named entity extraction project.
- Label data, train, and deploy an entity extraction model.
- Submit extraction tasks from your own app.

---

## Understand custom named entity recognition

Custom NER is an Azure API service that looks at documents, identifies, and extracts user defined entities. These entities could be anything from names and addresses from bank statements to knowledge mining to improve search results.

Custom NER is part of Azure AI Language in Azure AI services.

### Custom vs built-in NER

Azure AI Language provides certain built-in entity recognition, to recognize things such as a person, location, organization, or URL. Built-in NER allows you to set up the service with minimal configuration, and extract entities. To call a built-in NER, create your service and call the endpoint for that NER service like this:

```
<YOUR-ENDPOINT>/language/analyze-text/jobs?api-version=<API-VERSION>
```

Placeholder	Value	Example
<YOUR-ENDPOINT>	The endpoint for your API request	https://<your-resource>.cognitiveservices.azure.com
<API-VERSION>	The version of the API you are calling	2023-05-01

The body of that call will contain the document(s) the entities are extracted from, and the headers contain your service key.

The response from the call above contains an array of entities recognized, such as:

```
<...>
"entities":[
  {
    "text":"Seattle",
    "category":"Location",
    "subcategory":"GPE",
    "offset":45,
    "length":7,
    "confidenceScore":0.99
  },
  {
    "text":"next week",
    "category":"DateTime",
    "subcategory":"DateRange",
    "offset":104,
    "length":9,
    "confidenceScore":0.8
  }
]
<...>
```

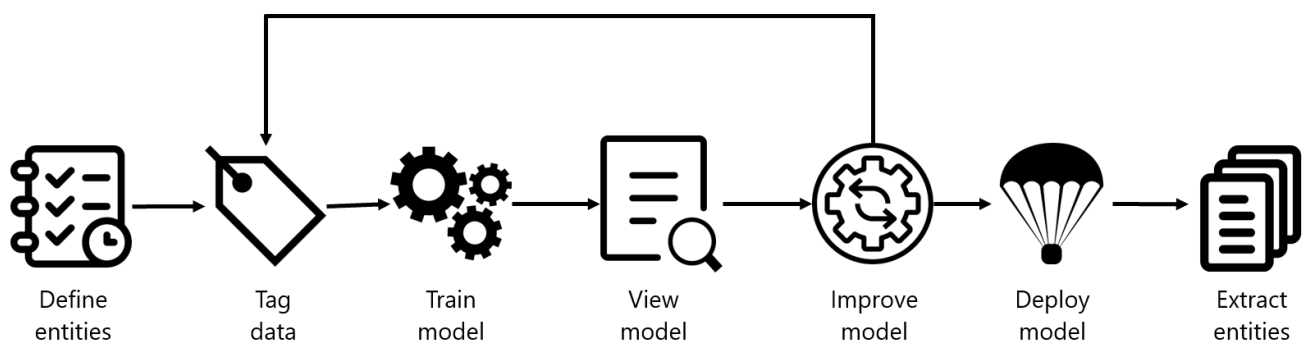
Examples of when to use the built-in NER include finding locations, names, or URLs in long text documents.

Tip: A full list of recognized entity categories is available in the [NER docs](#).

Custom NER, which is the focus of the rest of this module, is available when the entities you want to extract aren't part of the built-in service or you only want to extract specific entities. You can make your custom NER model as simple or complex as is required for your app.

Examples of when you'd want custom NER include **specific legal or bank data, knowledge mining to enhance catalog search, or looking for specific text for audit policies**. Each one of these projects requires a specific set of entities and data it needs to extract.

## Azure AI Language project life cycle



Creating an entity extraction model typically follows a similar path to most Azure AI Language service features:

1. **Define entities:** Understanding the data and entities you want to identify, and try to make them as clear as possible. For example, defining exactly which parts of a bank statement you want to extract.
2. **Tag data:** Label, or tag, your existing data, specifying what text in your dataset corresponds to which entity. This step is important to do accurately and completely, as any wrong or missed labels will reduce the effectiveness of the trained model. A good variation of possible input documents is useful. For example, label bank name, customer name, customer address, specific loan or account terms, loan or account amount, and account number.
3. **Train model:** Train your model once your entities are labeled. Training teaches your model how to recognize the entities you label.
4. **View model:** After your model is trained, view the results of the model. This page includes a score of 0 to 1 that is based on the precision and recall of the data tested. You can see which entities worked well (such as customer name) and which entities need improvement (such as account number).
5. **Improve model:** Improve your model by seeing which entities failed to be identified, and which entities were incorrectly extracted. Find out what data needs to be added to your model's training to improve performance. This page shows you how entities failed, and which entities (such as account number) need to be differentiated from other similar entities (such as loan amount).
6. **Deploy model:** Once your model performs as desired, deploy your model to make it available via the API. In our example, you can send to requests to the model when it's deployed to extract bank statement entities.
7. **Extract entities:** Use your model for extracting entities. The lab covers how to use the API, and you can view the [API reference](#) for more details.

## Considerations for data selection and refining entities

For the best performance, you'll need to use both high quality data to train the model and clearly defined entity types.

High quality data will let you spend less time refining and yield better results from your model.

- **Diversity** - use as diverse of a dataset as possible without losing the real-life distribution expected in the real data. You'll want to use sample data from as many sources as possible, each with their own formats and number of entities. It's best to have your dataset represent as many different sources as possible.
- **Distribution** - use the appropriate **distribution of document types**. A more diverse dataset to train your model will help your model avoid learning incorrect relationships in the data.
- **Accuracy** - use data that is as close to real world data as possible. Fake data works to start the training process, but it likely will differ from real data in ways that can cause your model to not extract correctly.

Entities need to also be carefully considered, and defined as distinctly as possible. Avoid ambiguous entities (such as two names next to each other on a bank statement), as it will make the model struggle to differentiate. If having some ambiguous entities is required, make sure to have more examples for your model to learn from so it can understand the difference.

**Keeping your entities distinct will also go a long way** in helping your model's performance. For example, trying to extract something like "Contact info" that could be a phone number, social media handle, or email address would require several examples to correctly teach your model. Instead, **try to break them down into more specific entities** such as "Phone", "Email", and "Social media" and let the model label whichever type of contact information it finds.

## How to extract entities

To submit an extraction task, the API requires the JSON body to specify which task to execute. For custom NER, the task for the JSON payload is `CustomEntityRecognition`.

Your payload will look similar to the following JSON:

```
{
  "displayName": "string",
  "analysisInput": {
    "documents": [
      {
        "id": "doc1",
        "text": "string"
      },
      {
        "id": "doc2",
        "text": "string"
      }
    ]
  },
  "tasks": [
    {
      "kind": "CustomEntityRecognition",
      "taskName": "MyRecognitionTaskName",
      "parameters": {
        "projectName": "MyProject",
        "deploymentName": "MyDeployment"
      }
    }
  ]
}
```

## Project limits

The Azure AI Language service enforces the following restrictions:

- **Training - at least 10 files**, and not more than 100,000
- **Deployments** - 10 deployment names per project
- **APIs**
  - **Authoring** - this API creates a project, trains, and deploys your model. Limited to 10 POST and 100 GET per minute
  - **Analyze** - this API does the work of actually extracting the entities; it requests a task and retrieves the results. Limited to 20 GET or POST
- **Projects** - only 1 storage account per project, 500 projects per resource, and 50 trained models per project
- **Entities** - each entity can be up to 500 characters. You can have up to **200** entity types.

See the [Service limits for Azure AI Language](#) page for detailed information.

# Label your data

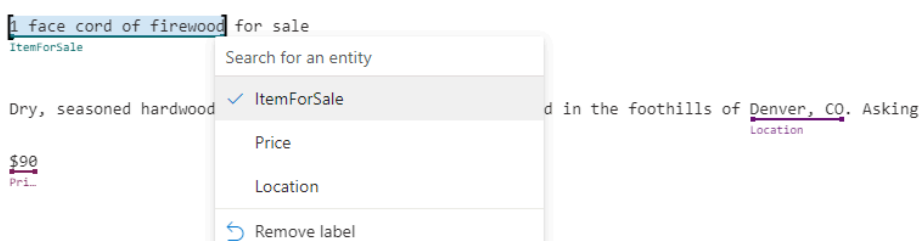
Labeling, or tagging, your data correctly is an important part of the process to create a custom entity extraction model. Labels identify examples of specific entities in text used to train the model. Three things to focus on are:

- **Consistency** - Label your data the same way across all files for training. Consistency allows your model to learn without any conflicting inputs.
- **Precision** - Label your entities consistently, without unnecessary extra words. Precision ensures only the correct data is included in your extracted entity.
- **Completeness** - Label your data completely, and don't miss any entities. Completeness helps your model always recognize the entities present.

## Data labeling ✓ Saved

Select a document to annotate its text with entity labels. After labeling the documents and adding them to training or testing sets, you'll be ready to create a model with this data in [Training](#).

Single document view Document name: Ad 1.txt



## How to label your data

Language Studio is the most straight forward method for labeling your data. Language Studio allows you to see the file, select the beginning and end of your entity, and specify which entity it is.

Each label that you identify gets saved into a file that lives in your storage account with your dataset, in an auto-generated JSON file. This file then gets used by the model to learn how to extract custom entities. It's possible to provide this file when creating your project (if you're importing the same labels from a different project, for example) however it must be in the [Accepted custom NER data formats](#). For example:

```
{
  "projectFileVersion": "{DATE}",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    "projectKind": "CustomEntityRecognition",
    "storageInputContainerName": "{CONTAINER-NAME}",
    "projectName": "{PROJECT-NAME}",
    "multilingual": false,
    "description": "Project-description",
    "language": "en-us",
    "settings": {}
  },
  "assets": {
    "projectKind": "CustomEntityRecognition",
```

```
"entities": [
  {
    "category": "Entity1"
  },
  {
    "category": "Entity2"
  }
],
"documents": [
  {
    "location": "{DOCUMENT-NAME}",
    "language": "{LANGUAGE-CODE}",
    "dataset": "{DATASET}",
    "entities": [
      {
        "regionOffset": 0,
        "regionLength": 500,
        "labels": [
          {
            "category": "Entity1",
            "offset": 25,
            "length": 10
          },
          {
            "category": "Entity2",
            "offset": 120,
            "length": 8
          }
        ]
      }
    ]
  },
  {
    "location": "{DOCUMENT-NAME}",
    "language": "{LANGUAGE-CODE}",
    "dataset": "{DATASET}",
    "entities": [
      {
        "regionOffset": 0,
        "regionLength": 100,
        "labels": [
          {
            "category": "Entity2",
            "offset": 20,
            "length": 5
          }
        ]
      }
    ]
  }
]
```

```
}
]
}
}
```

Field	Description
documents	Array of labeled documents
location	Path to file within container connected to the project
language	Language of the file
entities	Array of present entities in the current document
regionOffset	Inclusive character position for start of text
regionLength	Length in characters of the data used in training
category	Name of entity to extract
labels	Array of labeled entities in the files
offset	Inclusive character position for start of entity
length	Length in characters of the entity
dataset	Which dataset the file is assigned to

---

## Train and evaluate your model

Training and evaluating your model is an iterative process of adding data and labels to your training dataset to teach the model more accurately. To know what types of data and labels need to be improved, Language Studio provides scoring in the **View model details** page on the left hand pane.



Status: ✔ Trained successfully

F1 score: 92.31%

①

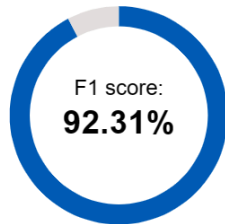
Precision: 85.71%

①

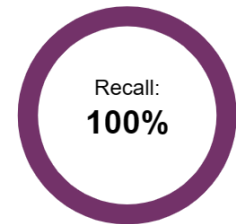
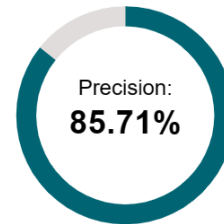
Recall: ① 100%

Finished training on: 8/19/2023, 7:03:53 PM

Total training time: 0 hour(s), 11 minute(s), 26 second(s)



Training data splitting type: Percentage  
Number of training documents: 10 (83.33%)  
Number of testing documents: 2 (16.67%)



[Learn more about how to improve your model](#)

Individual entities and your overall model score are broken down into three metrics to explain how they're performing and where they need to improve.

Metric	Description
Precision	The ratio of successful entity recognitions to all attempted recognitions. A high score means that as long as the entity is recognized, it's labeled correctly.
Recall	The ratio of successful entity recognitions to the actual number of entities in the document. A high score means it finds the entity or entities well, regardless of if it assigns them the right label
F1 score	Combination of precision and recall providing a single scoring metric

Scores are available both per entity and for the model as a whole. You may find an entity scores well, but the whole model doesn't.

## How to interpret metrics

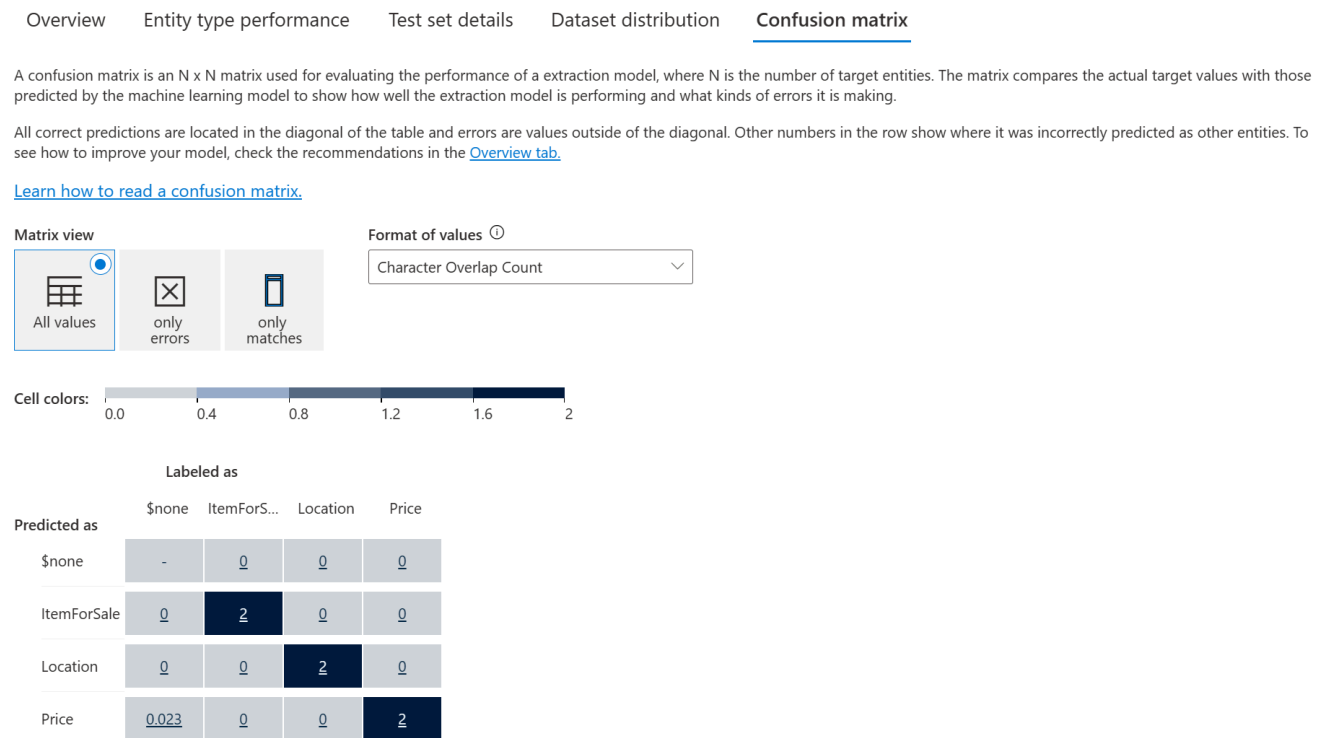
Ideally we want our model to score well in both precision and recall, which means the entity recognition works well. If both metrics have a low score, it means the model is both struggling to recognize entities in the document, and when it does extract that entity, it doesn't assign it the correct label with high confidence.

**If precision is low but recall is high, it means that the model recognizes the entity well but doesn't label it as the correct entity type.**

**If precision is high but recall is low, it means that the model doesn't always recognize the entity, but when the model extracts the entity, the correct label is applied.**

## Confusion matrix

On the same **View model details** page, there's another tab on the top for the *Confusion matrix*. This view provides a visual table of all the entities and how each performed, giving a complete view of the model and where it's falling short.



The confusion matrix allows you to visually identify where to add data to improve your model's performance.

## Exercise - 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 a **Language Service** resource. When on the page for *Select additional features*, select the custom feature containing **Custom named**

## entity recognition extraction.

### Select additional features ...



By default, Azure AI service for Language comes with several pre-built capabilities like sentiment analysis, key phrase extraction, pre-built question answering, etc. Some customizable features below require additional services like Azure AI Search, Blob storage, etc. to be provisioned as well. Select the custom features you want to enable as part of your Language service.

<ul style="list-style-type: none"><li>✓ Sentiment analysis</li><li>✓ Key phrase extraction</li><li>✓ Pre-built question answering</li><li>✓ Conversational language understanding</li><li>✓ Named entity recognition</li><li>✓ Text Summarization</li><li>✓ Text analytics for Health</li></ul>	<div><div>✓</div><div><b>Custom question answering</b> Use this feature to answer user's questions over your data corpus. Requires Azure AI Search. <a href="#">Learn more.</a></div><div>Select</div></div>
	<div><div>✓</div><div><b>Custom text classification, Custom named entity recognition, Custom summarization, Custom sentiment analysis &amp; Custom Text Analytics for health</b> ⓘ Use these customization features to tailor our products for your specific requirements. Requires Azure Storage. <a href="#">Learn more.</a></div><div>Unselect</div></div>

### 3. 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.

[Home](#) > [Azure AI services | Language service](#) > [Select additional features](#) >

### Create Language ...

#### Instance Details

Region ⓘ	East US
Name * ⓘ	ai-learn-ner-1 ✓
Pricing tier * ⓘ	S (1K Calls per minute) ✓

[View full pricing details](#)

Custom text classification, Custom named entity recognition, Custom summarization, Custom sentiment analysis & Custom Text Analytics for health

You need to create your own storage when using these customization features to host and upload all your training and labeled data and have it saved in your own Azure subscription. You get to associate only one storage account with one language resource that cannot be changed moving forward.

[Learn more](#)

New/Existing storage account *	<input checked="" type="radio"/> New storage account <input type="radio"/> Existing storage account
Storage account name *	ailearnstorage1 ✓
Storage account type *	Standard LRS ✓

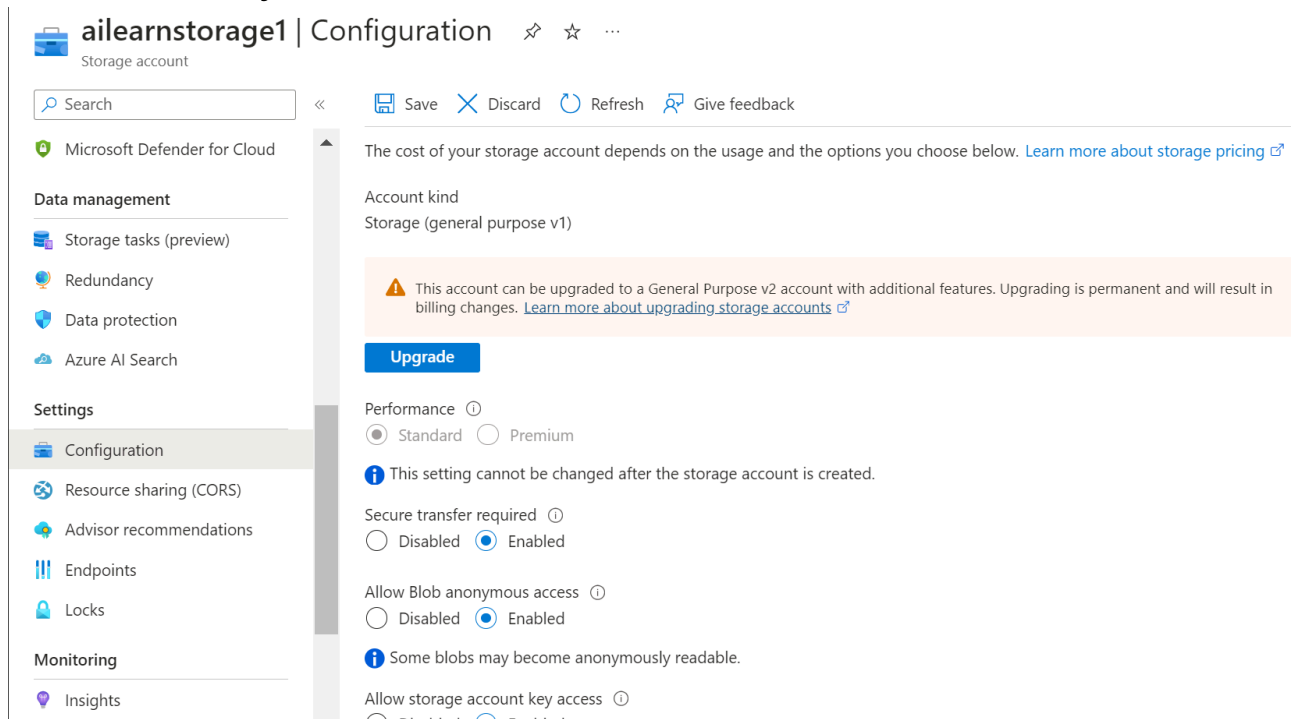
4. Select **Review + create**, then select **Create** to provision the resource.
5. Wait for deployment to complete, and then go to the deployed resource.

6. View the **Keys and Endpoint** page. You will need the information on this page later in the exercise.

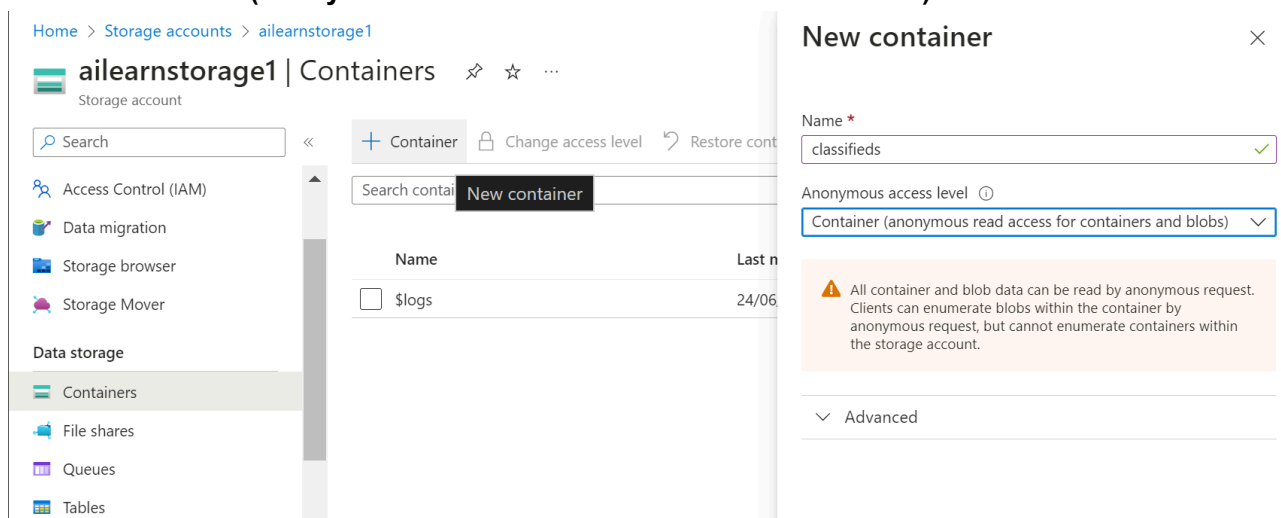
## Upload sample ads

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 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.

Select an Azure resource

Default Directory

Note: Switching directory will cause the page to refresh.

Azure subscription \*

Select an existing Azure subscription or [create a free account](#) and then refresh

Subscription 1

Resource type \*

Language

Language or Azure Cognitive Services resources can be used for any capability except for translation, which requires a Translator resource

Resource name \*

Select an existing resource or create a new one

ai-learn-ner-1

Pricing tier: Standard (S)

Managed identity: Enabled

[Create a new Language resource](#)

Done Cancel

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:

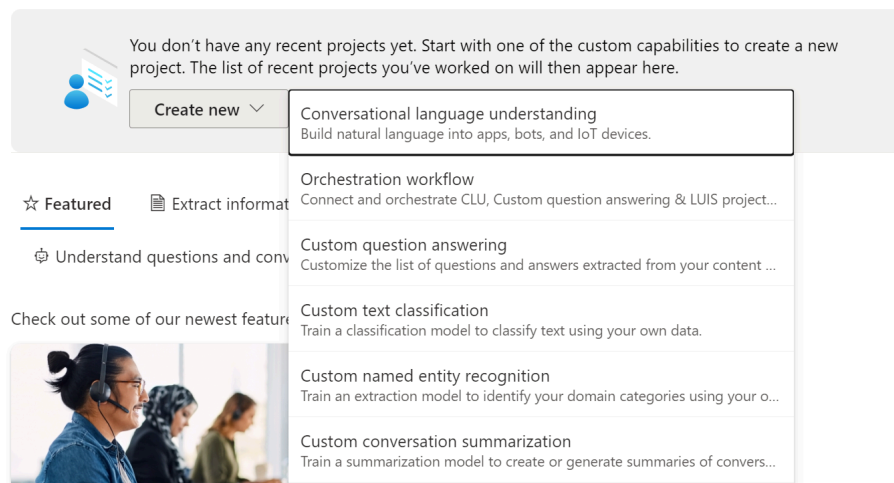
1. On the bar at the top of the page, select the **Settings (⚙)** button.
2. On the **Settings** page, view the **Resources** tab.
3. Select the language resource you just created, and click **Switch resource**.
4. At the top of the page, click **Language Studio** to return to the Language Studio home page.

3. At the top of the portal, in the **Create new** menu, select **Custom named entity recognition**.

Language Studio

## Welcome to Language Studio

Recent custom projects you've worked on



You don't have any recent projects yet. Start with one of the custom capabilities to create a new project. The list of recent projects you've worked on will then appear here.


Create new ▾

- Conversational language understanding  
Build natural language into apps, bots, and IoT devices.
- Orchestration workflow  
Connect and orchestrate CLU, Custom question answering & LUIS project...
- Custom question answering  
Customize the list of questions and answers extracted from your content ...
- Custom text classification  
Train a classification model to classify text using your own data.
- Custom named entity recognition  
Train an extraction model to identify your domain categories using your o...
- Custom conversation summarization  
Train a summarization model to create or generate summaries of convers...

☆ Featured   ▢ Extract informat

🔍 Understand questions and conv

Check out some of our newest feature



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:**

Create a project

×

☒ Connect storage

☒ Enter basic information

☒ Choose container

☐ Review and finish

Choose the Blob store container which contains the dataset you wish to use with this project.

Subscription: Subscription 1

Storage account: ailearnstorage1

Blob store container \* ⓘ

classifieds ▾ ↻

Are your documents already labeled with entities?

To create a custom NER model, your documents need to be labeled with entities. You can label documents after you create the project or import an existing labels file. Labels files can either be a JSON file following [Language Studio formatting](#) or a file following [Azure Machine Learning formats](#)

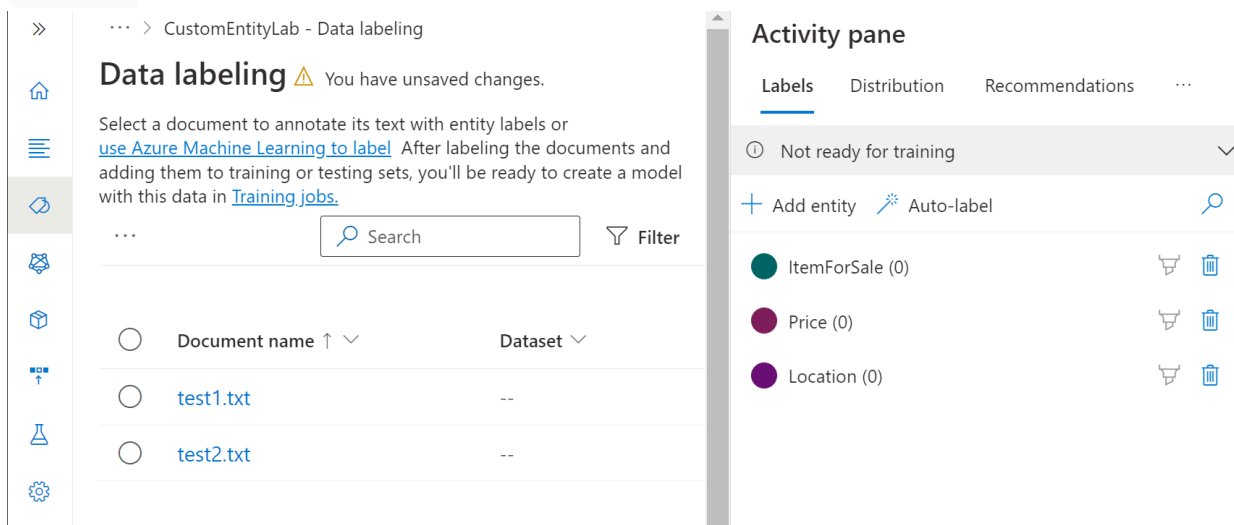
☒ No, I need to label my documents as part of this project

☐ Yes, my documents are already labeled and I have a correctly formatted JSON labels file

## 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**

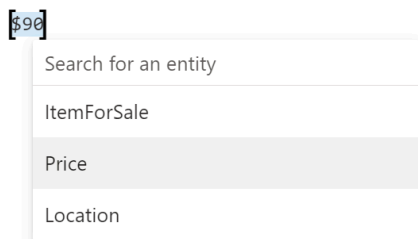


4. After you've created your three entities, select **Ad 1.txt** so you can read it.
5. In **Ad 1.txt**:
  1. Highlight the text *face cord of firewood* and select the **ItemForSale** entity.
  2. Highlight the text *Denver, CO* and select the **Location** entity.
  3. Highlight the text *\$90* and select the **Price** entity. 1. In the **Activity** pane, note that this document will be added to the dataset for training the model.

Single document view Document name: Ad 1.txt

1 face cord of firewood for sale  
ItemForSale

Dry, seasoned hardwood ready to burn! Pickup only, located in the foothills of Denver, CO. Asking  
Location



6. 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.
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.

**Note:** If Visual Studio Code shows you a pop-up message to prompt you to trust the code you are opening, click on **Yes, I trust the authors** option in the pop-up.

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. 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 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#:**

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

**Python:**

```
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 file 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 (View here: [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

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

**Python:** custom-entities.py (View here: [custom-entities.py](#))

```
# 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

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

**Python:** custom-entities.py (View here: [custom-entities.py](#))

```
# 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, 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

```
// 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 (View here: [custom-entities.py](#))

```
# 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 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 it from your **Projects** page in Language Studio. You can also remove the Azure AI Language service and associated storage account in the [Azure portal](#).

## Knowledge Check

1. You've trained your model and you're seeing that it doesn't recognize your entities. What metric score is likely low to indicate that issue? \*

☒ Recall

✓ That answer's correct. Recall indicates how well the model extracts entities, regardless of which entity that is.

☐ Precision

☐ F1 score

2. You just finished labeling your data. How and where is that file stored to train your model? \*

☒ JSON file, in my storage account container for the project

✓ That answer's correct. The JSON file lives next to the dataset in your container for the model to use during training.

☐ XML file, in my local project folder

☐ YAML file, anywhere in my Azure account

3. You train your model with only one source of documents, even though real extraction tasks will come from several sources. What data quality metric do you need to increase? \*

☐ Distribution

☐ Accuracy

☒ Diversity

✓ That answer's correct. Having the right data diversity will lead to better extraction performance.

## Summary

In this module, you learned about custom named entity recognition and how to extract entities.

In this module, you learned how to:

- Understand custom named entities and how they're labeled.
- Build a Language service project.
- Label data, train, and deploy an entity extraction model.
- Submit extraction tasks from your own app.

To learn more about Azure AI Language, see the [Azure AI Language documentation](#).

---

👉 Compiled by [Kenneth Leung](#) (2025)