

Custom named entity recognition

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/1-introduction>

Introduction

Completed

- 1 minute

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 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 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.

2. Understand custom named entity recognition

Understand custom named entity recognition

Completed

- 7 minutes

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 Language in Foundry Tools.

Custom vs built-in NER

Azure 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	<code>https://<your-resource>.cognitiveservices.azure.com</code>
<API-VERSION>	The version of the API you are calling	<code>2023-05-01</code>

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.

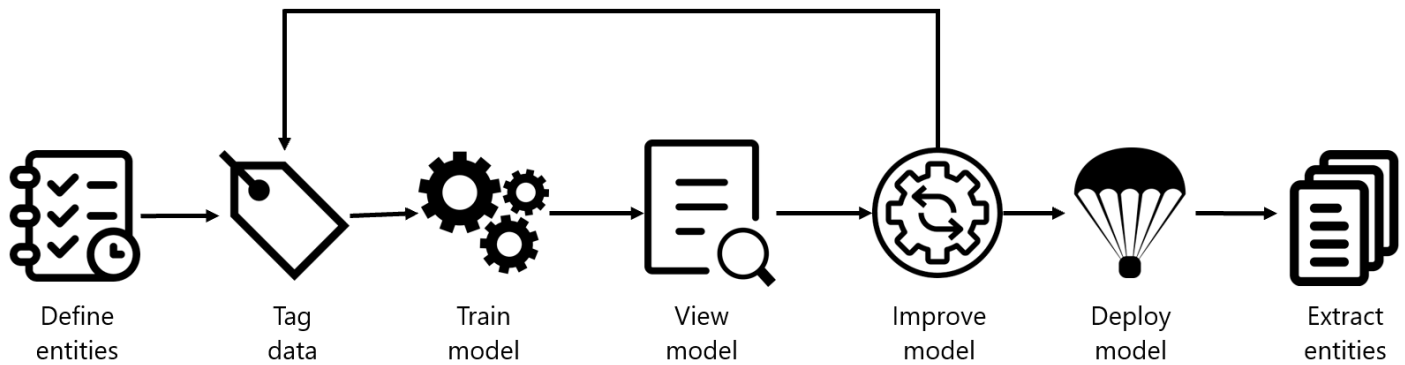
Tip

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 Language project life cycle



Creating an entity extraction model typically follows a similar path to most Azure 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 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 Language](#) page for detailed information.

3. Label your data

<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/3-tag-your-data>

Label your data

Completed

- 4 minutes

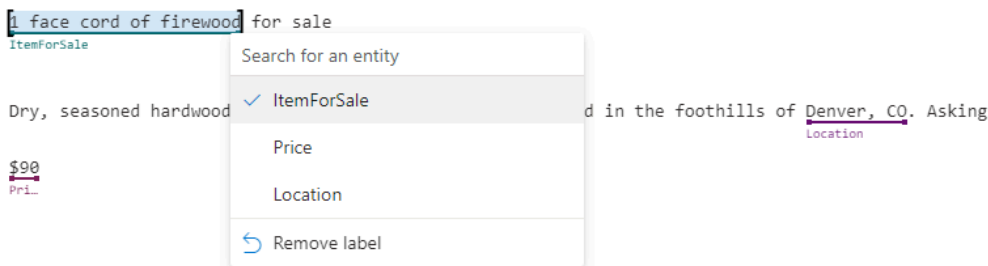
Labeling, or tagging, your data correctly is an important part of the process to create a custom entity extraction model. Labels identify examples of specific entities in text used to train the model. Three things to focus on are:

- **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



Labels Distribution

Ready for training

+ Add entity

ItemForSale ...

Price (12)

Location (12)

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

4. Train and evaluate your model

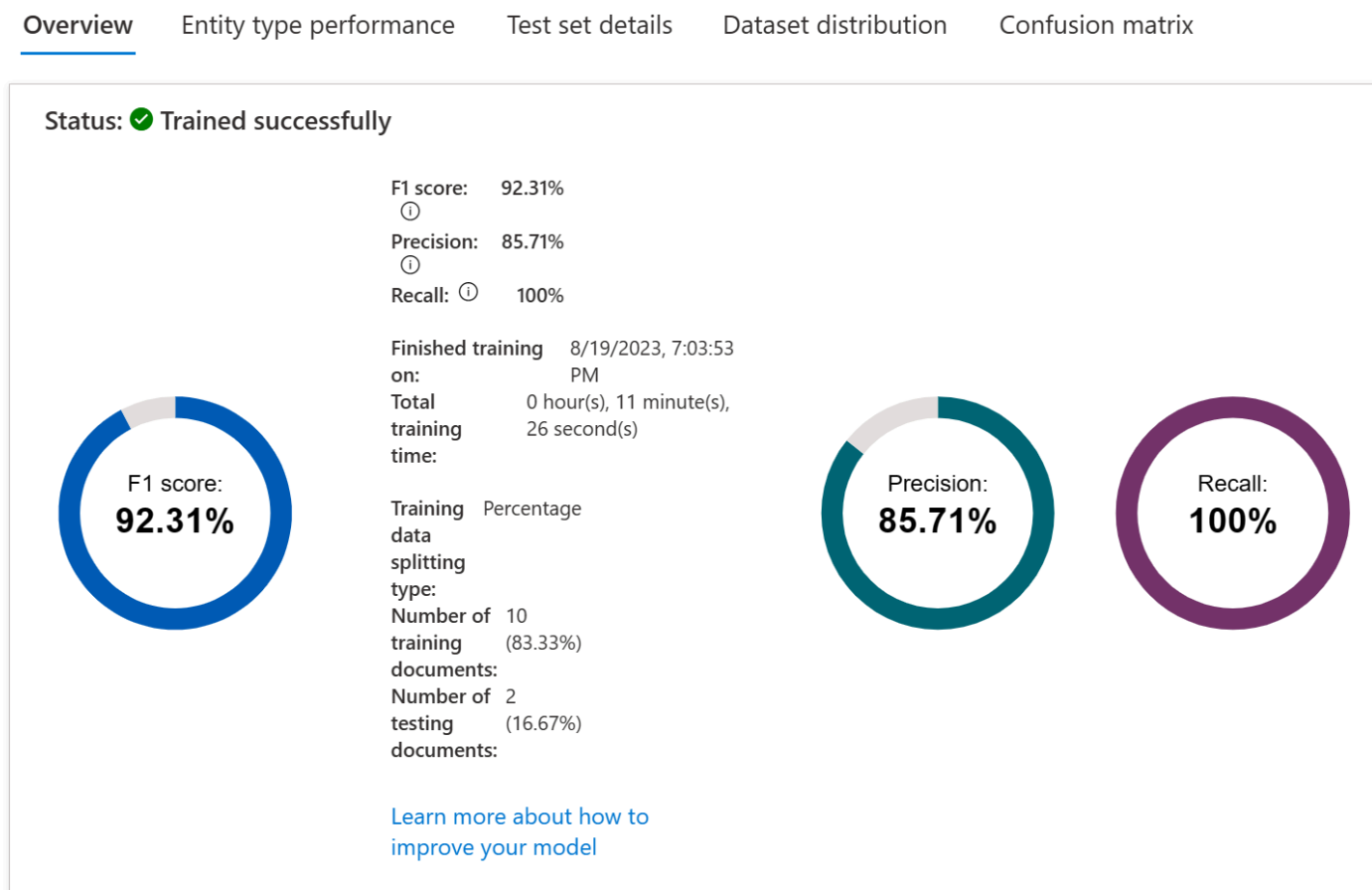
<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/4-train-evaluate-your-model>

Train and evaluate your model

Completed

- 3 minutes

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.



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

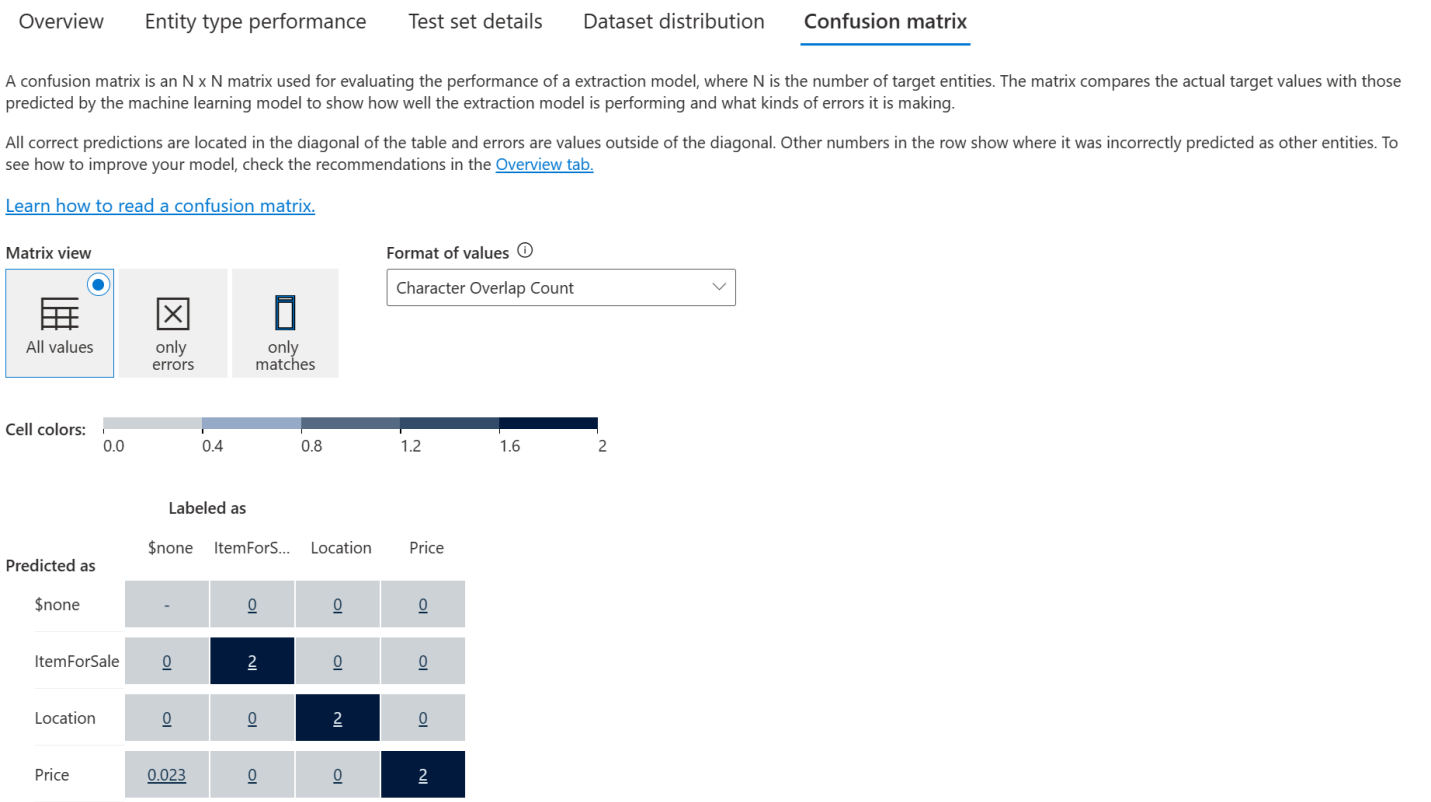
Ideally we want our model to score well in both precision and recall, which means the entity recognition works well. If both metrics have a low score, it means the model is both struggling to recognize entities in the document, and when it does extract that entity, it doesn't assign it the correct label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

5. Exercise - Extract custom entities

<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/5-exercise-extract-custom-entities>

Exercise - Extract custom entities

Completed

- 35 minutes

In this exercise, you use Azure Language to build a custom named entity recognition model.

Note

To complete this lab, you need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

Tip

After completing the exercise, if you've finished exploring Foundry Tools, delete the Azure resources that you created during the exercise.

6. Module assessment

<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/6-knowledge-check>

Module assessment

Completed

- 2 minutes

7. Summary

<https://learn.microsoft.com/en-us/training/modules/custom-name-entity-recognition/7-summary>

Summary

Completed

- 1 minute

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 Language, see the [Azure Language documentation](#).