3.4 - Create a custom text classification solution

Overview

The Azure AI Language service enables processing of natural language to use in your own app. Learn how to build a custom text classification project.

Learning objectives

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

- Understand types of classification projects
- Build a custom text classification project
- Tag data, train, and deploy a model
- Submit classification tasks from your own app

Introduction

Natural language processing (NLP) is one of the most common AI problems, where software must interpret text or speech in the natural form that humans use. Part of NLP is the ability to classify text, and Azure provides ways to classify text including sentiment, language, and custom categories defined by the user.

In this module, you'll learn how to use the Azure AI Language service to classify text into custom groups.

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

- Understand types of classification projects.
- Build a custom text classification project.
- Tag data, train, and deploy a model.
- Submit classification tasks from your own app.

Understand types of classification projects

Custom text classification assigns labels, which in the Azure AI Language service is a *class* that the developer defines, to text files. For example, a video game summary might be classified as "Adventure", "Strategy", "Action" or "Sports".

Custom text classification falls into two types of projects:

- **Single label classification** - you can assign only one class to each file. Following the above example, a video game summary could only be classified as "Adventure" or "Strategy".
- **Multiple label classification** - you can assign multiple classes to each file. This type of project would allow you to classify a video game summary as "Adventure" or "Adventure and Strategy".

When creating your custom text classification project, you can specify which project you want to build.

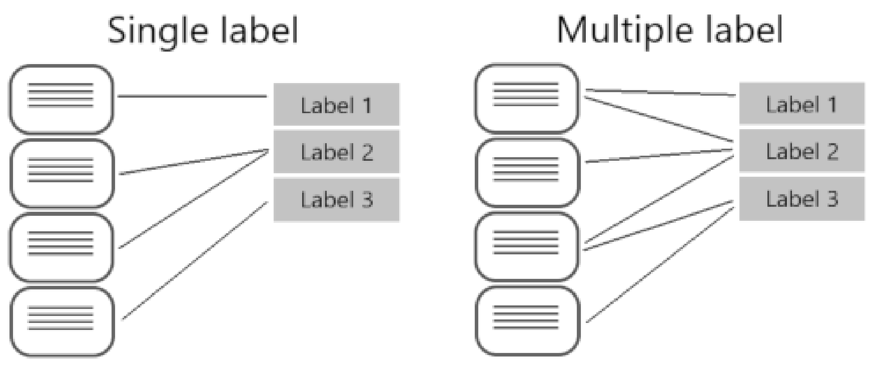
Single vs. multiple label projects

Beyond the ability to put files into multiple classifications, the key differences with multiple label classification projects are labeling, considerations for improving your model, and the API payload for classification tasks.

Labeling data

In single label projects, each file is assigned one class during the labeling process; class assignment in Azure AI Language only allows you to select one class.

When labeling multiple label projects, you can assign as many classes that you want per file. The impact of the added complexity means your data has to remain clear and provide a good distribution of possible inputs for your model to learn from.



Labeling data correctly, especially for multiple label projects, is directly correlated with how well your model performs. The higher the quality, clarity, and variation of your data set is, the more accurate your model will be.

Evaluating and improving your model

Measuring predictive performance of your model goes beyond how many predictions were correct. Correct classifications are when the actual label is *x* and the model predicts a label *x*. In the real world, documents result in different kinds of errors when a classification isn't correct:

- False positive - model predicts *x*, but the file isn't labeled *x*.
- False negative - model doesn't predict label *x*, but the file in fact is labeled *x*.

These metrics are translated into three measures provided by Azure AI Language:

- **Recall** - Of all the actual labels, how many were identified; the ratio of true positives to all that was labeled.
- **Precision** - How many of the predicted labels are correct; the ratio of true positives to all identified positives.
- **F1 Score** - A function of *recall* and *precision*, intended to provide a single score to maximize for a balance of each component

> Tip: Learn more about the Azure AI Language evaluation metrics, including exactly how these metrics are calculated

With a single label project, you can identify which classes aren't classified as well as others and find more quality data to use in training your model. **For multiple label projects, figuring out quality data becomes more complex due to the matrix of possible permutations of combined labels.**

For example, let's your model is correctly classifying "Action" games and some "Action and Strategy" games, but failing at "Strategy" games. To improve your model, you'll want to find more high quality and varied summaries for both "Action and Strategy" games, as well at "Strategy" games to teach your model how to differentiate the two. **This challenge increases exponentially with more possible classes your model is classifying into.**

API payload

Azure AI Language provides a REST API to build and interact with your model, using a **JSON body to specify the request.** This API is abstracted into multiple language-specific SDKs, however for this module we'll focus our examples on the base REST API.

To submit a classification task, the API requires the JSON body to specify which task to execute. You'll learn more about the REST API in the next unit, but worth familiarizing yourself with parts of the required body.

Single label classification models specify a project type of `customSingleLabelClassification`:

```
{
  "projectFileVersion": "<API-VERSION>",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    "projectName": "<PROJECT-NAME>",
    "storageInputContainerName": "<CONTAINER-NAME>",
    "projectKind": "customSingleLabelClassification",
    "description": "Trying out custom multi label text classification",
    "language": "<LANGUAGE-CODE>",
    "multilingual": true,
    "settings": {}
  },
  "assets": {
    "projectKind": "customSingleLabelClassification",
        "classes": [
            {
                "category": "Class1"
            },
            {
                "category": "Class2"
            }
        ],
        "documents": [
            {
                "location": "<DOCUMENT-NAME>",
                "language": "<LANGUAGE-CODE>",
                "dataset": "<DATASET>",
                "class": {
                    "category": "Class2"
                }
            },
            {
                "location": "<DOCUMENT-NAME>",
                "language": "<LANGUAGE-CODE>",
                "dataset": "<DATASET>",
                "class": {
                    "category": "Class1"
                }
            }
        ]
    }
}
```

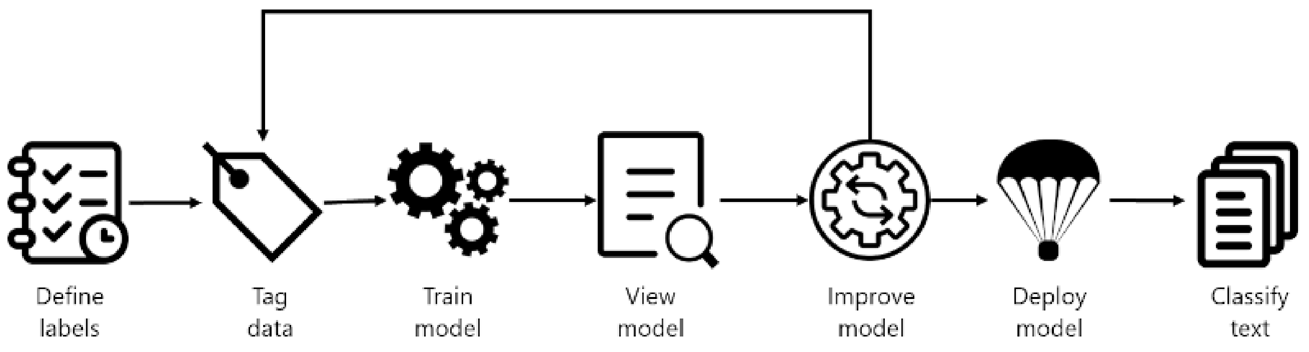
Multiple label classification models specify a project type of `CustomMultiLabelClassification`

```
{
  "projectFileVersion": "<API-VERSION>",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    "projectName": "<PROJECT-NAME>",
    "storageInputContainerName": "<CONTAINER-NAME>",
    "projectKind": "customMultiLabelClassification",
    "description": "Trying out custom multi label text classification",
    "language": "<LANGUAGE-CODE>",
    "multilingual": true,
    "settings": {}
  },
  "assets": {
    "projectKind": "customMultiLabelClassification",
    "classes": [
      {
        "category": "Class1"
      },
      {
        "category": "Class2"
      }
    ],
    "documents": [
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "classes": [
          {
            "category": "Class1"
          },
          {
            "category": "Class2"
          }
        ]
      },
      {
        "location": "<DOCUMENT-NAME>",
        "language": "<LANGUAGE-CODE>",
        "dataset": "<DATASET>",
        "classes": [
          {
            "category": "Class2"
          }
        ]
      }
    ]
  }
}
```

Understand how to build text classification projects

Custom text classification projects are your workspace to build, train, improve, and deploy your classification model. You can work with your project in two ways: through **Language Studio** and via the REST API. Language Studio is the GUI that will be used in the lab, but the REST API has the same functionality. Regardless of which method you prefer, the steps for developing your model are the same.

Azure AI Language project life cycle



- **Define labels**: Understanding the data you want to classify, identify the possible labels you want to categorize into. In our video game example, our labels would be "Action", "Adventure", "Strategy", and so on.
- **Tag data**: Tag, or label, your existing data, specifying the label or labels each file falls under. Labeling data is important since it's how your model will learn how to classify future files. Best practice is to have clear differences between labels to avoid ambiguity, and provide good examples of each label for the model to learn from. For example, we'd label the game "Quest for the Mine Brush" as "Adventure", and "Flight Trainer" as "Action".
- **Train model**: Train your model with the labeled data. Training will teach our model what types of video game summaries should be labeled which genre.
- **View model**: After your model is trained, view the results of the model. Your model is scored between 0 and 1, based on the precision and recall of the data tested. Take note of which genre didn't perform well.
- **Improve model**: Improve your model by seeing which classifications failed to evaluate to the right label, see your label distribution, and find out what data to add to improve performance. For example, you might find your model mixes up "Adventure" and "Strategy" games. Try to find more examples of each label to add to your dataset for retraining your model.

- **Deploy model**: Once your model performs as desired, deploy your model to make it available via the API. Your model might be named "GameGenres", and once deployed can be used to classify game summaries.
- **Classify text**: Use your model for classifying text. The lab covers how to use the API, and you can view the API reference

How to split datasets for training

When labeling your data, you can specify which dataset you want each file to be:

- **Training** - The training dataset is used to actually train the model; the data and labels provided are fed into the machine learning algorithm to teach your model what data should be classified to which label. The training dataset will be the larger of the two datasets, recommended to be about 80% of your labeled data.
- **Testing** - The testing dataset is labeled data used to verify you model after it's trained. Azure will take the data in the testing dataset, submit it to the model, and compare the output to how you labeled your data to determine how well the model performed. The result of that comparison is how your model gets scored and helps you know how to improve your predictive performance.

During the **Train model** step, there are two options for how to train your model.

- **Automatic split** - Azure takes all of your data, splits it into the specified percentages randomly, and applies them in training the model. This option is best when you have a larger dataset, data is naturally more consistent, or the distribution of your data extensively covers your classes.
- **Manual split** - Manually specify which files should be in each dataset. When you submit the training job, the Azure AI Language service will tell you the split of the dataset and the distribution. This split is best used with smaller datasets to ensure the correct distribution of classes and variation in data are present to correctly train your model.

To use the automatic split, put all files into the *training* dataset when labeling your data (this option is the default). To use the manual split, specify which files should be in testing versus training during the labeling of your data.

Deployment options

Azure AI Language allows each project to create both **multiple models and multiple deployments**, each with their own unique name. Benefits include ability to:

- Test two models side by side
- Compare how the split of datasets impact performance
- Deploy multiple versions of your model

> Note: **Each project has a limit of ten deployment names**

During deployment you can choose the name for the deployed model, which can then be selected when submitting a classification task:

```
<...>
  "tasks": [
    {
      "kind": "CustomSingleLabelClassification",
      "taskName": "MyTaskName",
      "parameters": {
        "projectName": "MyProject",
        "deploymentName": "MyDeployment"
      }
    }
  ]
<...>
```

Using the REST API

The REST API available for the Azure AI Language service allows for CLI development of Azure AI Language projects in the same way that Language Studio provides a user interface for building projects. Language Studio is explored further in this module's lab.

Pattern of using the API

The API for the Azure AI Language service operates asynchronously for most calls. In each step we submit a request to the service first, then check back with the service via a subsequent call to get the status or result.

With each request, a header is required to authenticate your request:

Key	Value
`Ocp-Apim-Subscription-Key`	The key to your Azure AI Language resource

Submit initial request

The URL to submit the request to varies on which step you are on, but all are prefixed with the endpoint provided by your Azure AI Language resource.

For example, to train a model, you would create a **POST** to the URL that would look something like the following:

```
<YOUR-ENDPOINT>/language/analyze-text/projects/<PROJECT-NAME>/:train?api-version=<API-VERSION>
```

Placeholder	Value	Example
`<YOUR-ENDPOINT>`	The endpoint for your API request	`https://<your-custom-resource>.cognitiveservices.azure.com`
`<PROJECT-NAME>`	The name for your project (value is case-sensitive)	`myProject`

The following body would be attached to the request:

```
    {
        "modelLabel": "<MODEL-NAME>",
        "trainingConfigVersion": "<CONFIG-VERSION>",
        "evaluationOptions": {
            "kind": "percentage",
            "trainingSplitPercentage": 80,
            "testingSplitPercentage": 20
        }
    }
```

Key	Value
`<YOUR-MODEL>`	Your model name.
`trainingConfigVersion`	The model version to use to train your model.
`runValidation`	Boolean value to run validation on the test set.
`evaluationOptions`	Specifies evaluation options.
`kind`	Specifies data split type. Can be `percentage` if you're using an automatic split, or `set` if you manually split your dataset
`testingSplitPercentage`	Required integer field only if `type` is *percentage*. Specifies testing split.
`trainingSplitPercentage`	Required integer field only if `type` is *percentage*. Specifies training split.

The response to the above request will be a `202`, meaning the request was **successful**. Grab the `location` value from the response headers, which will look similar to the following URL:

```
<ENDPOINT>/language/analyze-text/projects/<PROJECT-NAME>/train/jobs/<JOB-ID>?api-version=<API-VERSION>
```

Key	Value
`<JOB-ID>`	Identifier for your request

This URL is used in the next step to get the training status.

Get training status

To get the training status, use the URL from the header of the request response to submit a **GET** request, with same header that provides our Azure AI Language service key for authentication. The response body will be similar to the following JSON:

```
{
  "result": {
    "modelLabel": "<MODEL-NAME>",
    "trainingConfigVersion": "<CONFIG-VERSION>",
    "estimatedEndDateTime": "2023-05-18T15:47:58.8190649Z",
    "trainingStatus": {
      "percentComplete": 3,
      "startDateTime": "2023-05-18T15:45:06.8190649Z",
      "status": "running"
    },
    "evaluationStatus": {
      "percentComplete": 0,
      "status": "notStarted"
    }
  },
  "jobId": "<JOB-ID>",
  "createdDateTime": "2023-05-18T15:44:44Z",
  "lastUpdatedDateTime": "2023-05-18T15:45:48Z",
  "expirationDateTime": "2023-05-25T15:44:44Z",
  "status": "running"
}
```

Training a model can take some time, so periodically check back at this status URL until the response `status` returns `succeeded`. Once the training has succeeded, you can view, verify, and deploy your model.

Consuming a deployed model

Using the model to classify text follows the same pattern as outlined above, **with a POST request submitting the job and a GET request to retrieve the results.**

Submit text for classification

To use your model, submit a **POST** to the *analyze* endpoint at the following URL:

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

Placeholder	Value	Example
`<YOUR-ENDPOINT>`	The endpoint for your API request	`https://<your-custom-resource>.cognitiveservices.azure.com`

> Important: Remember to include your **resource key in the header** for `Ocp-Apim-Subscription-Key`

The following JSON structure would be attached to the request:

```
{
  "displayName": "Classifying documents",
  "analysisInput": {
    "documents": [
      {
        "id": "1",
        "language": "<LANGUAGE-CODE>",
        "text": "Text1"
      },
      {
        "id": "2",
        "language": "<LANGUAGE-CODE>",
        "text": "Text2"
      }
    ]
  },
  "tasks": [
     {
      "kind": "<TASK-REQUIRED>",
      "taskName": "<TASK-NAME>",
      "parameters": {
        "projectName": "<PROJECT-NAME>",
        "deploymentName": "<DEPLOYMENT-NAME>"
      }
    }
  ]
}
```

Key	Value
`<TASK-REQUIRED>`	Which task you're requesting. The task is `CustomMultiLabelClassification` for multiple label projects, or `CustomSingleLabelClassification` for single label projects
`<LANGUAGE-CODE>`	The language code such as `en-us`.
`<TASK-NAME>`	Your task name.
`<PROJECT-NAME>`	Your project name.
`<DEPLOYMENT-NAME>`	Your deployment name.

The response to the above request will be a `202`, meaning the request was successful. **Look for the `operation-location` value in the response headers, which will look something like the following URL:**

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

Key	Value
`<YOUR-ENDPOINT>`	The endpoint for your API request
`<JOB-ID>`	Identifier for your request

This URL is used to get your task results.

Get classification results

Submit a **GET** request to the endpoint from the previous request, with the same header for authentication. The response body will be similar to the following JSON:

```
{
  "createdDateTime": "2023-05-19T14:32:25.578Z",
  "displayName": "MyJobName",
  "expirationDateTime": "2023-05-19T14:32:25.578Z",
  "jobId": "xxxx-xxxxxx-xxxxx-xxxx",
  "lastUpdateDateTime": "2023-05-19T14:32:25.578Z",
  "status": "succeeded",
  "tasks": {
    "completed": 1,
    "failed": 0,
    "inProgress": 0,
    "total": 1,
    "items": [
      {
        "kind": "customSingleClassificationTasks",
        "taskName": "Classify documents",
        "lastUpdateDateTime": "2022-10-01T15:01:03Z",
        "status": "succeeded",
        "results": {
          "documents": [
            {
              "id": "<DOC-ID>",
              "class": [
                  {
                      "category": "Class_1",
                      "confidenceScore": 0.0551877357
                  }
              ],
              "warnings": []
            }
          ],
          "errors": [],
          "modelVersion": "2022-04-01"
        }
      }
    ]
  }
}
```

The classification result is within the items array's `results` object, for each document submitted.

Exercise - Classify text

Azure AI Language provides several NLP capabilities, including the key phrase identification, text summarization, and sentiment analysis. The Language service also provides custom features like custom question answering and custom text classification.

To test the custom text classification of the Azure AI Language service, we'll configure the model using Language Studio then use a small command-line application that runs in the Cloud Shell to test it. The same pattern and functionality used here can be followed for real-world applications.

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 search field at the top of the portal, search for `Azure AI services`, and create a **Language Service** resource.
3. Select the box that includes **Custom text classification**. Then select **Continue to create your resource**.
4. Create a 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.
5. Select **Review + create,** then select **Create** to provision the resource.
6. Wait for deployment to complete, and then go to the deployed resource.
7. View the **Keys and Endpoint** page. You will need the information on this page later in the exercise.

Upload sample articles

Once you've created the Azure AI Language service and storage account, you'll need to upload example articles to train your model later.

1. In a new browser tab, download sample articles from `https://aka.ms/classification-articles` 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**. In the Configuration screen enable the option to **Allow Blob anonymous access** then select **Save**.
4. Select **Containers** in the left menu, located below **Data storage**. On the screen that appears, select **+ Container**. Give the container the name `articles`, 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 you've created the container, select it then select the **Upload** button. Select **Browse for files** to browse for the sample articles you downloaded. Then select **Upload**.

Create a custom text classification project

After configuration is complete, create a custom text classification project. This project provides a working place to build, train, and deploy your model.

> **NOTE**: This lab utilizes **Language Studio**, but 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:

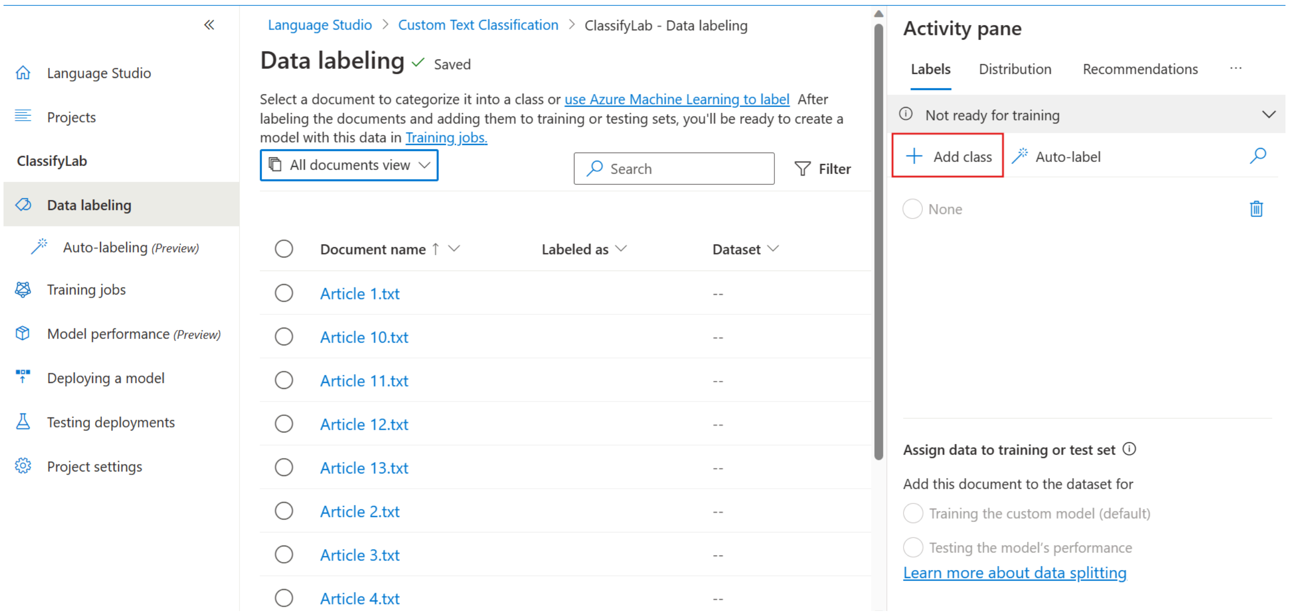
    - **Azure Directory**: The Azure directory containing your subscription.
    - **Azure subscription**: Your Azure subscription.
    - **Resource type**: Language.
    - **Language resource**: The Azure AI Language resource you created previously.

    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:
    1. On the bar at the top if 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 text classification**.
4. The **Connect storage** page appears. All values will already have been filled. So select **Next**.
5. On the **Select project type** page, select **Single label classification**. Then select **Next**.
6. On the **Enter basic information** pane, set the following:
    - **Name**: `ClassifyLab`
    - **Text primary language**: English (US)
    - **Description**: `Custom text lab`
7. Select **Next**.
8. On the **Choose container** page, set the **Blob store container** dropdown to your *articles* container.
9. Select the **No, I need to label my files as part of this project** option. Then select **Next**.
10. Select **Create project**.

Label your data

Now that your project is created, you need to label, or tag, your data to train your model how to classify text.

1. On the left, select **Data labeling**, if not already selected. 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 class**. The articles in this lab fall into four classes you'll need to create: `Classifieds`, `Sports`, `News`, and `Entertainment`.


Language Studio > Custom Text Classification > ClassifyLab - Data labeling

Data labeling ✓ Saved

Select a document to categorize it into a class or use Azure Machine Learning to label 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 jobs.

Document name	Labeled as	Dataset
Article 1.txt		--
Article 10.txt		--
Article 11.txt		--
Article 12.txt		--
Article 13.txt		--
Article 2.txt		--
Article 3.txt		--
Article 4.txt		--

Activity pane

Not ready for training

Assign data to training or test set

Add this document to the dataset for

Training the custom model (default)

Testing the model's performance

Learn more about data splitting

3. After you've created your four classes, select **Article 1** to start. Here you can read the article, define which class this file is, and which dataset (training or testing) to assign it to.
4. Assign each article the appropriate class and dataset (training or testing) using the **Activity** pane on the right. You can select a label from the list of labels on the right, and set each article to **training** or **testing** using the options at the bottom of the Activity pane. You select **Next document** to move to the next document. For the purposes of this lab, we'll define which are to be used for training the model and testing the model:

Article	Class	Dataset
Article 1	Sports	Training
Article 10	News	Training
Article 11	Entertainment	Testing
Article 12	News	Testing
Article 13	Sports	Testing
Article 2	Sports	Training
Article 3	Classifieds	Training
Article 4	Classifieds	Training
Article 5	Entertainment	Training
Article 6	Entertainment	Training
Article 7	News	Training
Article 8	News	Training
Article 9	Entertainment	Training

> **NOTE** Files in Language Studio are listed alphabetically, which is why the above list is not in sequential order. Make sure you visit both pages of documents when labeling your articles.

5. Select **Save labels** to save your labels.

Train your model

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

1. Select **Training jobs** on the left side menu.
2. Select **Start a training job**.
3. Train a new model named `ClassifyArticles`.
4. Select **Use a manual split of training and testing data**.

    > **TIP** In your own classification projects, the Azure AI Language service will automatically split the testing set by percentage which is useful with a large dataset. With smaller datasets, it's important to train with the right class distribution.

5. Select **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 of text classification, it's important to evaluate and improve your model to verify it's performing as you expect.

1. Select **Model performance**, and select your **ClassifyArticles** model. There you can see the scoring of your model, performance metrics, and when it was trained. If the scoring of your model isn't 100%, it means that one of the documents used for testing didn't evaluate to what it was labeled. These failures can help you understand where to improve.
2. Select **Test set details** tab. If there are any errors, this tab allows you to see the articles you indicated for testing and what the model predicted them as and whether that conflicts with their test label. The tab defaults to show incorrect predictions only. You can toggle the **Show mismatches only** option to see all the articles you indicated for testing and what they each of them predicted as.

Deploy your model

When you're satisfied with the training of your model, it's time to deploy it, which allows you to start classifying text through the API.

1. On the left panel, select **Deploying model**.
2. Select **Add deployment**, then enter `articles` in the **Create a new deployment name** field, and select **ClassifyArticles** in the **Model** field.
3. Select **Deploy** to deploy your model.
4. Once your model is deployed, leave that page open. You'll need your project and deployment name in the next step.

Prepare to develop an app in Visual Studio Code

To test the custom text classification 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, 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/04-text-classification** folder and expand the **CSharp** or **Python** folder depending on your language preference and the **classify-text** 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 **classify-text** 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 **classify-text** 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 text classification model.
5. Save the configuration file.

Add code to classify documents

Now you're ready to use the Azure AI Language service to classify documents.

1. Expand the **articles** folder in the **classify-text** folder to view the text articles that your application will classify.
2. In the **classify-text** folder, open the code file for the client application:
    - **C#**: Program.cs

- **Python**: classify-text.py (View here: classify-text.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: classify-text.py (View here: classify-text.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
   AzureKeyCredential(aiSvcKey); Uri endpoint = new Uri(aiSvcEndpoint); TextAnalyticsClient
   aiClient = new TextAnalyticsClient(endpoint, credentials);
   ```

   Python: classify-text.py (View here: classify-text.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 **articles** folder and creates a list containing their contents. Then find the comment **Get Classifications** and add the following code:

   C#: Program.cs
   ```
   // Get Classifications ClassifyDocumentOperation operation = await
   aiClient.SingleLabelClassifyAsync(WaitUntil.Completed, batchedDocuments, projectName,
   deploymentName); int fileNo = 0; await foreach (ClassifyDocumentResultCollection
   documentsInPage in operation.Value) { ``` foreach (ClassifyDocumentResult documentResult
   in documentsInPage) { Console.WriteLine(files[fileNo].Name); if
   (documentResult.HasError) { Console.WriteLine($" Error!"); Console.WriteLine($" Document
   error code: {documentResult.Error.ErrorCode}"); Console.WriteLine($" Message:
   {documentResult.Error.Message}"); continue; } Console.WriteLine($" Predicted the
   following class:"); Console.WriteLine(); foreach (ClassificationCategory classification
   in documentResult.ClassificationCategories) { Console.WriteLine($" Category:
   {classification.Category}"); Console.WriteLine($" Confidence score:
   {classification.ConfidenceScore}"); Console.WriteLine(); } fileNo++; } ``` }
   ```

   Python: classify-text.py (View here: classify-text.py)
   ```
   # Get Classifications operation = ai_client.begin_single_label_classify(
   batchedDocuments, project_name=project_name, deployment_name=deployment_name )
   document_results = operation.result() for doc, classification_result in zip(files,
   document_results): if classification_result.kind == "CustomDocumentClassification":
   classification = classification_result.classifications[0] print("{} was classified as
   '{}' with confidence score {}.".format( doc, classification.category,
   classification.confidence_score) ) elif classification_result.is_error is True: print("
   {} has an error with code '{}' and message '{}'".format( doc,
   classification_result.error.code, classification_result.error.message) )
   ```

6. 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 classify-text.py`

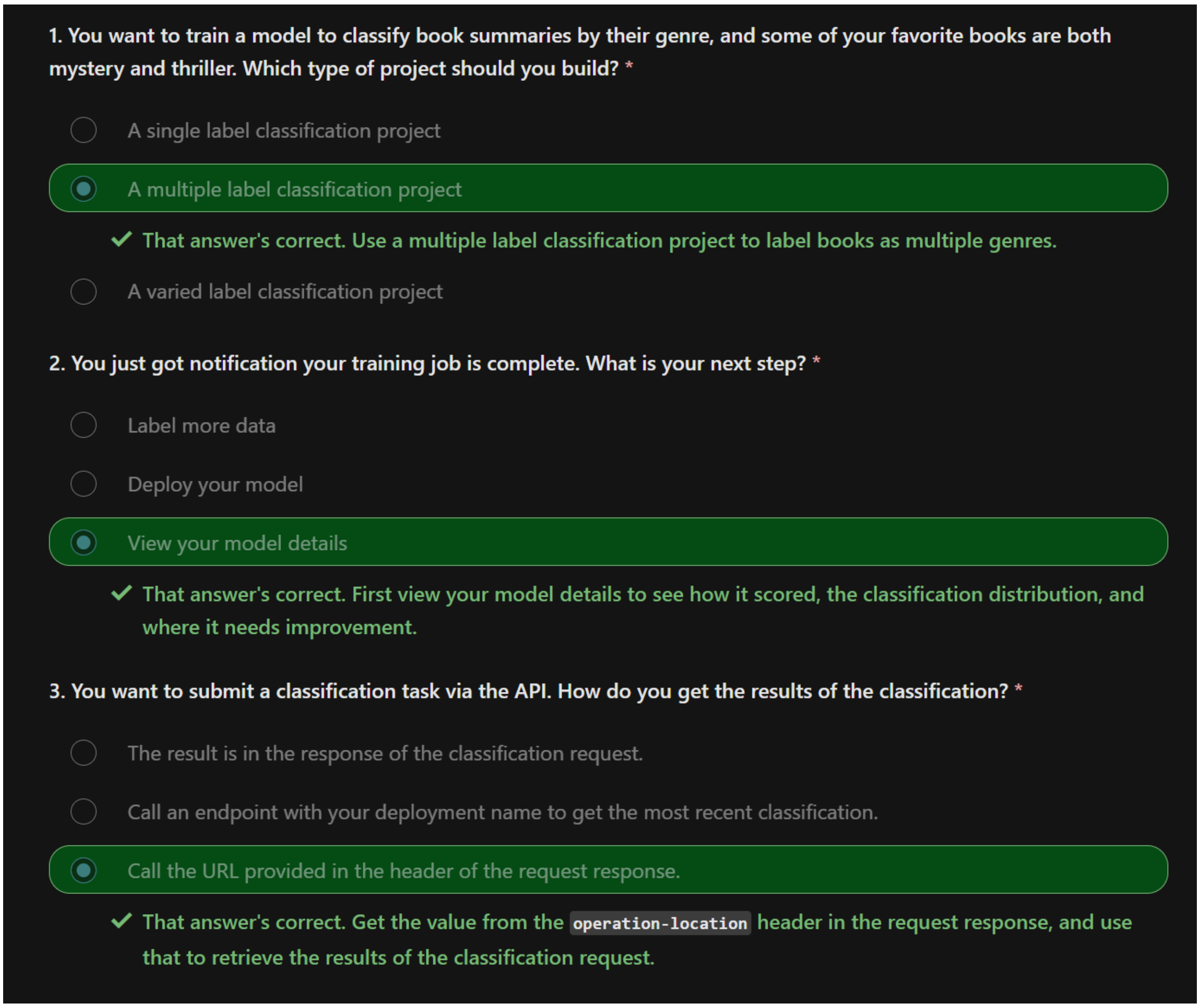
    > **Tip**: You can use the **Maximize panel size** (**^**) icon in the terminal toolbar to see more of the console text.

2. Observe the output. The application should list a classification and confidence score for 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.

Knowledge Check


1. You want to train a model to classify book summaries by their genre, and some of your favorite books are both mystery and thriller. Which type of project should you build? *

- ○ A single label classification project
- [✓ A multiple label classification project]
  ✓ That answer's correct. Use a multiple label classification project to label books as multiple genres.
- ○ A varied label classification project

2. You just got notification your training job is complete. What is your next step? *

- ○ Label more data
- ○ Deploy your model
- [✓ View your model details]
  ✓ That answer's correct. First view your model details to see how it scored, the classification distribution, and where it needs improvement.

3. You want to submit a classification task via the API. How do you get the results of the classification? *

- ○ The result is in the response of the classification request.
- ○ Call an endpoint with your deployment name to get the most recent classification.
- [✓ Call the URL provided in the header of the request response.]
  ✓ That answer's correct. Get the value from the `operation-location` header in the request response, and use that to retrieve the results of the classification request.

Summary

In this module, you learned about custom text classification and how to build a text classification service.

Now that you've completed this module, you can:

- Understand types of classification projects.
- Build a custom text classification project.
- Tag data, train, and deploy a model.
- Submit classification tasks from your own app.

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

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