## AutoML Translation Tutorial

This tutorial demonstrates how to create your own custom translation model and translate text using Google AutoML Translation API.

The example covers custom model training using English to Japanese dataset, model performance evaluation and translation by following the below steps.

1. Prepare training data
2. Create a dataset
3. Import sentence pairs into a dataset
4. Create (Train) a model
5. List model evaluations
6. Make a prediction
7. Delete a model

## Installation

The AutoML Translation client library for Node.js is required to run the application on your local workstation. Download the Node.js client library, unzip it and copy it in your project folder. Run following commands to install the client library

npm init

npm install --save nodejs-automl

Install following additional libraries “argparse”

npm install argparse

This application is tested with Node.js v10.4.1

Set the following environment variables.

* export GOOGLE\_APPLICATION\_CREDENTIALS=**/path/to/service\_account.json**
* export PROJECT\_ID=**project-id**
* export REGION\_NAME=**region-name** e.g. "us-central1"

### Node.js Files

The tutorial consists of the following Node.js programs. These files should be available in your project folder.

* automl\_translation\_tutorial\_dataset.js – It has functionalities related to dataset.
* automl\_translation\_tutorial\_model.js – It has functionalities related to model.
* automl\_translation\_tutorial\_predict.js – It has functionalities related to predictions.

## Running the Application

### Step 1: Preparing Training Data

Model customization requires training data and optionally validation and test data in Tab-separated values (.tsv) or Translation Memory eXchange (.tmx) format. A csv file is created to define the Google Storage bucket locations of these three datasets.

* en-ja.csv: There are two columns. First column has the type of data and can have value “TRAIN”, “VALIDATION” or “TEST”. Second column has the actual path of the dataset file. An example is given below.
  + TRAIN,gs://**{project\_id}**-vcm/en-ja-train.tmx
  + VALIDATION,gs://**{project\_id}**-vcm/en-ja-validation.tmx
  + TEST,gs://**{project\_id}**-vcm/en-ja-test.tmx
* en-ja-train.tmx: This file has the English and Japanese sentence pairs for training data.
* en-ja-validation.tmx: Similar to the training data file, this has the sentence pairs for evaluating the model created using the training data.
* en-ja-test.tmx: This file has the sentence pairs and the model uses them as test data.

Copy the above four files from “resources\dataset” folder to Google Storage bucket *gs://****{PROJECT\_ID}****-vcm.* Make sure that the [PROJECT\_ID] is replaced with your Project ID in the en-ja.csv**.**

### Step 2: Create a Dataset

Create an empty dataset to facilitate importing the sentence pairs for training the model.

##### Request

Make a request to create an empty dataset and name it ‘TD\_ENJA\_DEV\_0626\_1330’ by executing the following request. Parameters ‘en’ and ‘ja’ denote the source and target language respectively.

node automl\_translation\_tutorial\_dataset.js create\_dataset "TD\_ENJA\_DEV\_0626\_1330" "en" "ja"

##### Code

The createDataset() function takes in the following parameters and returns the newly created dataset object.

* projectId: Id of the project.
* computeRegion: Region name.
* datasetName: Name of the dataset to be created
* source: Language code of the text to be translated. Example: en
* target: Language code of the translated text. Example: ja

/\*\*

\* Create a dataset.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} datasetName

\* Name of the dataset.

\* @param {string} source

\* Source language code.

\* @param {string} target

\* Target language code.

\*/

function createDataset (projectId, computeRegion, datasetName, source, target) {

const automl = require('@google-cloud/automl');

const client = new automl.v1beta1.AutoMlClient();

// A resource that represents Google Cloud Platform location.

const projectLocation = client.locationPath(projectId, computeRegion);

// Specify the source and target language.

const datasetSpec = {

sourceLanguageCode: source,

targetLanguageCode: target

};

// Set dataset name and dataset specification.

const dataset = {

displayName: datasetName,

translationDatasetMetadata: datasetSpec

};

// Create a dataset with the dataset specification in the region.

client.createDataset({parent: projectLocation, dataset: dataset})

.then(responses => {

const dataset = responses[0];

// Display the dataset information

console.log('Dataset name:', dataset.name);

console.log('Dataset id: ', dataset.name.split('/').pop(-1));

console.log('Dataset display name: ', dataset.displayName);

console.log('Dataset example count: ', dataset.exampleCount);

console.log('Translation dataset specification:');

console.log('\tSource language code: ', dataset.translationDatasetMetadata.sourceLanguageCode);

console.log('\tTarget language code: ', dataset.translationDatasetMetadata.targetLanguageCode);

console.log('Dataset create time:');

console.log('\tseconds: ', dataset.createTime.seconds);

console.log('\tnanos: ', dataset.createTime.nanos);

})

.catch(err => {

console.error(err);

});

}

##### Response

Dataset name: projects/216065747626/locations/us-central1/datasets/7230871261092518862

Dataset id: 7230871261092518862

Dataset display name: TD\_ENJA\_DEV\_0626\_1330

Dataset example count: 0

Translation dataset specification:

Source language code: en

Target language code: ja

Dataset create time:

seconds: 1530000062

nanos: 454725000

The response displays the dataset ID of the newly created dataset object.

### Step 3: Import Sentence Pairs into Dataset

We will now import the sentence pairs into the dataset.

##### Request

Make a request to import the data(training data, validation data and test data) into dataset by executing the following request with operation type ‘import\_data’. The path of the CSV file in Google Storage bucket and the dataset ID (created from the previous step) are passed as arguments.

node automl\_translation\_tutorial\_dataset.js import\_data "7230871261092518862" "gs://{PROJECT\_ID}-vcm/en-ja.csv"

##### Code

The importData() function takes in the following parameters, imports dataset text files and displays the import status.

* projectId: Id of the project.
* computeRegion: Region name.
* datasetId: ID of the dataset into which the training, validation and test data are to be imported.
* path: Path of the CSV file.

/\*\*

\* Import sentence pairs to the dataset.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} datasetId

\* Id of the dataset to which the training data will be imported.

\* @param {string} path

\* Path of the training data csv file.

\*/

function importData (projectId, computeRegion, datasetId, path) {

const automl = require('@google-cloud/automl');

const client = new automl.v1beta1.AutoMlClient();

// Get the full path of the dataset.

const datasetFullId = client.datasetPath(projectId, computeRegion, datasetId);

// Get the multiple Google Cloud Storage URIs.

const inputUris = path.split(',');

const inputConfig = {

gcsSource: {

inputUris: inputUris

}

};

// Import data from the input URI.

client.importData({name: datasetFullId, inputConfig: inputConfig})

.then(responses => {

const operation = responses[0];

console.log('Processing import...');

// Operation#promise starts polling for the completion of the LRO.

return operation.promise();

})

.then(responses => {

// The final result of the operation.

console.log('Dataset imported.');

})

.catch(err => {

console.error(err);

});

}

##### Response

Processing import...

Dataset imported.

### Step 4: Create (Train) Model

Create a new custom translation model.

##### Request

Make a request to create a new model with the imported training data by executing the following request with operation type ‘create\_model’. Dataset ID and Model name are passed as arguments.

node automl\_translation\_tutorial\_model.js create\_model "7230871261092518862" "TM\_ENJA\_DEV\_0626\_1443"

##### Code

The createModel() function takes in the following parameters, creates model and displays the model creation status.

* projectId: Id of the project.
* computeRegion: Region name.
* datasetId: Dataset that has the training, validation and test data.
* modelName: Name of the model to be created.

/\*\*

\* Create a model.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} datasetId

\* Id of the dataset to which model is created.

\* @param {string} modelName

\* Name of the model.

\*/

function createModel (projectId, computeRegion, datasetId, modelName) {

const automl = require('@google-cloud/automl');

const client = new automl.v1beta1.AutoMlClient();

// A resource that represents Google Cloud Platform location.

const projectLocation = client.locationPath(projectId, computeRegion);

// Set model name and dataset.

const myModel = {

displayName: modelName,

datasetId: datasetId,

translationModelMetadata: {}

};

// Create a model with the model metadata in the region.

client.createModel({parent: projectLocation, model: myModel})

.then(responses => {

const operation = responses[0];

const initialApiResponse = responses[1];

console.log('Training operation name: ', initialApiResponse.name);

console.log('Training started...');

// Operation#promise starts polling for the completion of the LRO.

return operation.promise();

})

.then(responses => {

// The final result of the operation.

const model = responses[0];

console.log(model);

// Retrieve deployment state.

var deploymentState = '';

if (model.deploymentState === 1) {

deploymentState = 'deployed';

} else if (model.deploymentState === 2) {

deploymentState = 'undeployed';

}

// Display the model information.

console.log('Model name: ', model.name);

console.log('Model id: ', model.name.split('/').pop(-1));

console.log('Model display name: ', model.displayName);

console.log('Model create time:');

console.log('\tseconds: ', model.createTime.seconds);

console.log('\tnanos: ', model.createTime.nanos);

console.log('Model deployment state: ', deploymentState);

})

.catch(err => {

console.error(err);

});

}

Model ID is printed once the model is trained successfully.

##### Response

The model creation takes a few hours to complete. Once the model is created successfully, it returns the model ID.

Training operation name: projects/216065747626/locations/us-central1/operations/3677877269117914435

Training started...

Model name: projects/216065747626/locations/us-central1/models/574599445431604111

Model id: 574599445431604111

Model display name: TM\_ENJA\_DEV\_0626\_1443

Model create time:

seconds: 1530010686

nanos: 477478000

Model deployment state: undeployed

Note that the operation name is also displayed which can be used independently to check the model creation status using following command.

node automl\_translation\_tutorial\_model.js get\_operation\_status "projects/216065747626/locations/us-central1/operations/3677877269117914435"

### Step 5: List Model Evaluation

Evaluate the performance of the custom translation model.

##### Request

Make a request to display the evaluation performances of the model by executing the following request with operation type ‘listModelEvaluations’. Model ID and filter are passed as arguments.

node automl\_translation\_tutorial\_model.js list\_model\_evaluations "574599445431604111"

##### Code

The listModelEvaluations() function takes in the following parameters, retrieves a list of model evaluations and displays them.

* projectId: Id of the project.
* computeRegion: Region name.
* modelId: Model ID.
* filter\_: filter expression (Optional, default-''). Currently it is not implemented

/\*\*

\* List model evaluations.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} modelId

\* Id of the model.

\* @param {string} [filter\_]

\* Filter expression.

\*/

function listModelEvaluations (projectId, computeRegion, modelId, filter\_) {

const automl = require('@google-cloud/automl');

const client = new automl.v1beta1.AutoMlClient();

// Get the full path of the model.

const modelFullId = client.modelPath(projectId, computeRegion, modelId);

// List all the model evaluations in the model by applying filter.

client.listModelEvaluations({parent: modelFullId, filter: filter\_})

.then(responses => {

const element = responses[0];

console.log('List of model evaluations:');

for (let i = 0; i < element.length; i += 1) {

console.log(element[i]);

}

})

.catch(err => {

console.error(err);

});

}

##### Response

List of model evaluations:

{ name:

'projects/216065747626/locations/us-central1/models/574599445431604111/modelEvaluations/8046485181691436473',

annotationSpecId: '',

createTime: { seconds: '1530010719', nanos: 661558000 },

evaluatedExampleCount: 502,

translationEvaluationMetrics:

{ bleuScore: 2.581087499856949,

baseBleuScore: 3.314509615302086 },

metrics: 'translationEvaluationMetrics' }

### Step 6: Make a Prediction

Translate an input text content from English to Japanese using the custom translation model.

The text content that is to be translated is placed in the “resources\input.txt”.

##### Request

Make a translation request by executing the following request with operation type ‘predict’. Model ID and translationAllowFallback (‘True’) are passed as arguments.

node automl\_translation\_tutorial\_predict.js predict "574599445431604111" "resources\\input.txt"

##### Code

The predict() function takes in the following parameters and displays the translated text.

* projectId: Id of the project.
* computeRegion: Region name.
* modelId: Model ID of the custom translation model
* filePath: Input file path of the text content to be translated
* translationAllowFallback: Use ‘True’ if AutoML will fall-back to use a Google translation model for translation requests if the specified AutoML translation model cannot serve the request. Use ‘False’ to not use Google translation model. (Optional, default-’False’)

/\*\*

\* Get model details.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} modelId

\* Id of the model.

\* @param {string} filePath

\* Input file path of the content to be translated.

\* @param {string} [translationAllowFallback]

\* Use true if AutoML will fallback to use a Google translation model for

\* translation requests if the specified AutoML translation model cannot

\* serve the request. Use false to not use Google translation model.

\*/

function predict (projectId, computeRegion, modelId, filePath, translationAllowFallback) {

const automl = require('@google-cloud/automl');

const fs = require('fs');

// Create client for prediction service.

const client = new automl.v1beta1.PredictionServiceClient();

// Get the full path of the model.

const modelFullId = client.modelPath(projectId, computeRegion, modelId);

// Read the file content for translation.

const content = fs.readFileSync(filePath, 'utf8');

// Set the payload by giving the content of the file.

const payload = {

textSnippet: {

content: content

}

};

// params is additional domain-specific parameters.

// translationAllowFallback allows to use Google translation model.

var params = {};

if (translationAllowFallback) {

params = {

translationAllowFallback: true

};

}

client.predict({name: modelFullId, payload: payload, params: params})

.then(responses => {

const response = responses[0];

console.log('Translated Content: ', response.payload[0].translation.translatedContent.content);

})

.catch(err => {

console.error(err);

});

}

##### Response

Translated content: これがどのように終わるか教えて

Above is the Japanese translation for the English sentence: “Tell me how this ends”

### Step 7: Delete a Model

Delete permanently the model when it is no more required.

##### 

##### Request

Make a request with operation type ‘deleteModel’ to delete the model we created. Model ID is passed as argument.

node automl\_translation\_tutorial\_model.js delete\_model "574599445431604111"

##### Code

The deleteModel() function takes in the following parameters and permanently deletes the model.

* projectId: Id of the project.
* computeRegion: Region name.
* modelId: Model ID to delete

/\*\*

\* Delete a model.

\*

\* @param {string} projectId

\* Id of the project.

\* @param {string} computeRegion

\* Region name.

\* @param {string} modelId

\* Id of the model.

\*/

function deleteModel (projectId, computeRegion, modelId) {

const automl = require('@google-cloud/automl');

const client = new automl.v1beta1.AutoMlClient();

// Get the full path of the model.

const modelFullId = client.modelPath(projectId, computeRegion, modelId);

// Delete a model.

client.deleteModel({name: modelFullId})

.then(responses => {

const operation = responses[0];

// Operation#promise starts polling for the completion of the LRO.

return operation.promise();

})

.then(responses => {

// The final result of the operation.

console.log('Model deleted.');

})

.catch(err => {

console.error(err);

});

}

##### Response

Model deleted.