AI4EU Experiments Onboarding Tutorial: Sentiment Analysis

In this tutorial, we will be training a Keras Sentiment Analysis Model, writing the corresponding protobuf definitions, gRPC server and client and later dockerizing them.

Step 1: Training the keras models



Here, we wish to build a sentiment analysis model by using imdb dataset to classify a movie review as positive or negative. We use the imbd dataset from keras.datasets library for this purpose.

The trained model is then saved to the disk as model.h5 and loaded back when we need it to test a new review for the movie.

Step2: Write the service. The file should be named as model.py according to Acumos conventions.



In model.py we write a service which takes in a movie review as an argument and return its sentiment as being positive or negative. The trained model is loaded from the model.h5 file for this purpose.

Step 3: Make the Proto file

//Define the used version of proto

syntax = "proto3";

package fraunhofer.sentimentanalysis;

//Define a message to hold the features input by the client

message Text {

string query = 1;

}

//Define a message to hold the classification result

message Review\_Classify {

float review = 1 ;

}

//Define the service

service sentiment\_analysis\_model {

rpc classify\_review(Text) returns (Review\_Classify);

}

Step 4: Generate gRPC classes for python:

Open the terminal, change the directory to be in the same folder that the proto file is in.

To generate the gRPC classes we have to install the needed libraries first:

**#Install gRPC:**

python3 -m pip install grpcio

**#To install gRPC tools, run:**

python3 -m pip install grpcio-tools googleapis-common-protos

**Now, run the following command:**

python3 -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. model.proto

This command used model.proto file to generate the needed stubs to create the client/server.

The files generated will be as follows:

**model\_pb2.py — contains message classes**

* **model\_pb2**.**Text** for the input movie review
* **model\_pb2**.**Prediction** for the sentiment of the movie(i.e. positive (1) or negative (0))

**model\_pb2\_grpc.py — contains server and client classes**

* **model\_pb2\_grpc**.**PredictServicer** will be used by the server
* **model\_pb2\_grpc**.**PredictStub** the client will use it

Step 5: Creating a Server:

The server will import the generated files and the function that will handle the predictions.

Then we will define a class that will take a request from the client and uses the prediction function to return a respond.

The request gives us the five features, the response is a prediction.

After that, we will use **add\_PredictServicer\_to\_server** function from (**model\_pb2\_grpc.py**) file that was generated before to add the class PredictSevicer to the server.

Once you have implemented all the methods, the next step is to start up a gRPC server so that clients can actually use your service.



Step 6: Creating a Client:

In the client file we will do the following:

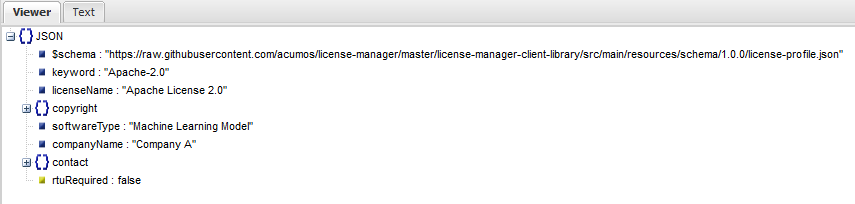
* Open a gRPC channel
* Create a [stub](https://whatis.techtarget.com/definition/stub)
* Create a request message
* Use the stub to call the service

Below is the code snippet for client:



Step 7: Include the license file

We need to include a license file before building a docker image. I have added a sample apache license in this example.



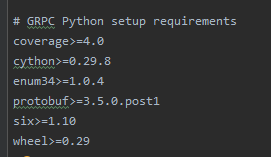
Step 8: Prepare the Docker file



Whenever any layer is re-built all the layers that follow it in the Dockerfile need to be rebuilt too. It's important to keep this fact in mind while creating Docker files.

The dockerfile here separates out the gRPC specific requirements in a separate file called requirements.txt. The reason for doing this is to separate the application dependency from the gRPC dependency. gRPC dependency in requirements.txt will be built as a separate layer when the Docker image is built. This avoids rebuild of this layer every time a change is made in the application. Below is the contents of gRPC requirement.txt.

We are also copying the license file along with all other required files to the container.



Build the Docker image

1. docker build -t sentiment\_analysis .

Run the docker image

2) docker run -p 50053:50053 --rm -ti sentiment\_analysis /bin/bash

The -p option maps the port on the container to the host.

The Docker run internally executes sentiment\_analysis\_server.py.

Open one more terminal and run the client which now can access the docker server

3) python3 sentiment\_analysis\_client.py