AI4EU Experiments Onboarding Tutorial: House Sales Price Prediction using gRPC

This tutorial provides a basic Python programmer’s introduction to working with gRPC.

By walking through this example you’ll learn how to:

* Define a service in a .proto file.
* Generate server and client code using the protocol buffer compiler.
* Use the Python gRPC API to write a simple client and server for your service.

It assumes that you have read the [Overview](https://grpc.io/docs/guides/#overview) and are familiar with [protocol buffers](https://developers.google.com/protocol-buffers/docs/overview). You can find out more in the [proto3 language guide](https://developers.google.com/protocol-buffers/docs/proto3) and [Python generated code guide](https://developers.google.com/protocol-buffers/docs/reference/python-generated).

What is gRPC?

With gRPC you can define your service once in a .proto file and implement clients and servers in any of gRPC’s supported languages, which in turn can be run in environments ranging from servers inside Google to your own tablet - all the complexity of communication between different languages and environments is handled for you by gRPC. You also get all the advantages of working with protocol buffers, including efficient serialization, a simple IDL, and easy interface updating.

This example is a Machine Learning Regression example that lets clients get the house sales prediction based on chosen attributes.

Steps

1. Write the service to be served.
2. Make a proto file to define the messages and services.
3. Use the proto file to generate gRPC classes for Python
4. Create the server.
5. Create the client.
6. Include a license
7. Prepare the docker file, run the docker and the run the client in a new tab.

Step 1: Write the Service:

In our case, the service is predicting house pricing. Below is code snippet.



This model has 5 input arguments (features of the house that we want to predict its sale price).

Since our motive here is to understand the gRPC, I have taken a simple DecisionTreeRegressor from sklearn.tree.

Step 2: Make the Proto File:

//Define the used version of proto

syntax = "proto3";

package fraunhofer.houseprediction;

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

message Features {

float MSSubClass = 1 ;

float LotArea = 2 ;

float YearBuilt = 3 ;

float BedroomAbvGr = 4 ;

float TotRmsAbvGrd = 5 ;

}

//Define a message to hold the predicted price

message Prediction {

float salePrice = 1 ;

}

//Define the service

service Predict {

rpc predict\_sale\_price(Features) returns (Prediction);

}

Here, we did not give values to the features, those numbers indicate the order of serializing the features.

## Step 3: 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**.**Features** for the input features
* **model\_pb2**.**Prediction** for the prediction price

**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 4: Creating the 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.

Below is the house\_sale\_prediction\_client.py



## Step 5: Creating the Client:

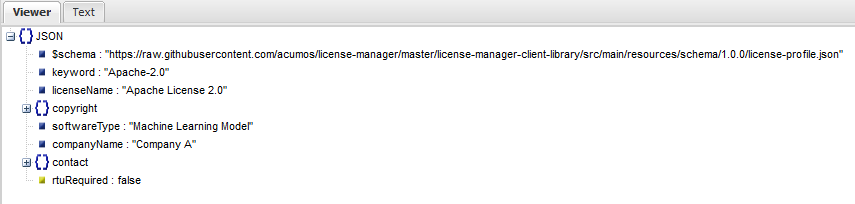
In the client file we will do the following:

* 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 6: Include a license File

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



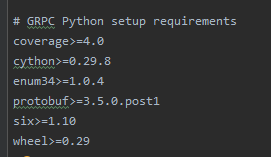
Step 7: Prepare the Docker file



In the docker file, we copy the license file along with other files required to the container.

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 Dockerfiles.

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.



Build the docker image

1. docker build -t house-price-prediction:v1 .

Run the docker image

2) docker run –p 8061:8061 --rm –ti house-price-prediction:v1 /bin/bash

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

The Docker run internally executes house\_price\_prediction\_server.py.

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

3) python3 house\_price\_prediction\_client.py