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

```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding, Conv1D, GlobalMaxPooling1D
from keras.datasets import imdb
import keras
from keras.models import Sequential
from keras.layers import Dense , Flatten ,Embedding,Input
from keras.models import Model
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)
max_features = 5000
maxlen = 100
batch_size = 64
embedding_dims = 32
filters = 128
kernel_size = 3
#hidden_size = 128
epochs = 5

(x_train,y_train),(x_test,y_test) = imdb.load_data(num_words=max_features)

# restore np.load for future normal usage
np.load = np_load_old

x_train = sequence.pad_sequences(x_train,maxlen = maxlen)
model = Sequential()
model.add(Embedding(max_features,embedding_dims, input_length = maxlen))
model.add(Conv1D(filters,kernel_size, padding ='valid',activation= 'relu',strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(128,activation='sigmoid'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer= 'adam', loss= 'binary_crossentropy', metrics=['acc'])

model.fit(x_train,y_train,batch_size=batch_size, epochs=epochs)

model.save("model.h5")
print("Saved model to disk")
```

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.

```
from keras.preprocessing.text import one_hot,Tokenizer
from keras.models import load_model
import warnings
from keras.datasets import imdb
from keras.preprocessing import sequence

def classify_review(review):
   maxlen = 100
   model = load_model('model.h5')
   d = imdb.get_word_index()
   words = review.split()
   review = []
   for word in words:
      if word not in d:
         review.append(2)
      else:
         review.append(d[word] + 3)

   review = sequence.pad_sequences([review],
                           truncating='pre', padding='pre', maxlen=maxlen)
   prediction = model.predict(review)
   return prediction[0][0]
```

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

The proto file must be explicitly named model.proto as Acumos expects it that way.

Also note that package names must be globally unique to let AI4EU Experiments distinguish the protobuf definitions for all onboarded models.

```
//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.

```
import grpc
from concurrent import futures
import time

# import the generated classes :
import model_pb2
import model_pb2_grpc

# import the function we made :
import model as psp

port = 50053

# create a class to define the server functions, derived from
# usingSKlearn_pb2_grpc.PredictServicer :
class sentiment_analysis_modelServicer(model_pb2_grpc.sentiment_analysis_mod-
elServicer):
    def classify_review(self, request, context):
        # define the buffer of the response :
        response = model_pb2.Review_Classify()
        # get the value of the response by calling the desired function :
        response.review = psp.classify_review(request.query)
        return response


# creat a grpc server :
server = grpc.server(futures.ThreadPoolExecutor(max_workers = 10))

model_pb2_grpc.add_sentiment_analysis_modelServicer_to_server(sentiment_analysis_mod-
elServicer(), server)

print('Starting server. Listening on port :' + str(port))
server.add_insecure_port("[::]:{}".format(port))
server.start()

try:
    while True:
        time.sleep(86400)
except KeyboardInterrupt:
    server.stop(0)
```

Step 6: Creating a Client:

In the client file we will do the following:

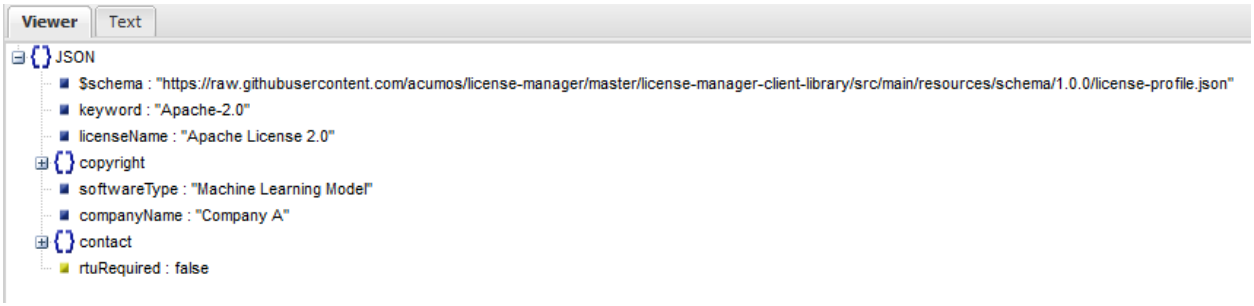
- Open a gRPC channel
- Create a stub
- Create a request message
- Use the stub to call the service

Below is the code snippet for client:

```
import grpc
from timeit import default_timer as timer

# import the generated classes
import model_pb2
import model_pb2_grpc

start_ch = timer()
port_address = 'localhost:50053'
# open a gRPC channel
channel = grpc.insecure_channel(port_address)

# create a stub (client)
stub = model_pb2_grpc.sentiment_analysis_modelStub(channel)
end_ch = timer()

text = "the movie was a great waste of my time"
ans_lst = []

start = timer()

# create a valid request message
requestPrediction  = model_pb2.Text(query = text)

print("Make the call")
# make the call
responsePrediction = stub.classify_review(requestPrediction)


print('The prediction is :',responsePrediction.review)
print('Done!')
```

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

```
MAINTAINER Tejas  "tejas.morbagal.harish@iais.fraunhofer.de"

RUN apt-get update -y
RUN apt-get install -y python3-pip python3-dev
RUN pip3 install --upgrade pip
RUN pip3 install numpy
RUN pip3 install pandas
RUN pip3 install tensorflow==1.13.1

RUN pip3 install keras

COPY ./requirements.txt /requirements.txt
COPY license-1.0.0.json model.h5 model.proto model_pb2.py model_pb2_grpc.py senti-
ment_analysis_server.py model.py ./


WORKDIR /

RUN pip3 install -r requirements.txt

ENTRYPOINT [ "python3","sentiment_analysis_server.py" ]
```

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.

```
# GRPC Python setup requirements
coverage>=4.0
cython>=0.29.8
enum34>=1.0.4
protobuf>=3.5.0.post1
six>=1.10
wheel>=0.29
```

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