100天搞定机器学习：模型训练好了，然后呢?

原创 student老胡 机器学习算法与Python实战 2021-04-08 20:16

收录于话题
#100天搞定机器学习

25个

↑↑↑点击上方蓝字，回复**资料**，10个G的惊喜

100天搞定机器学习|Day1-62 合集

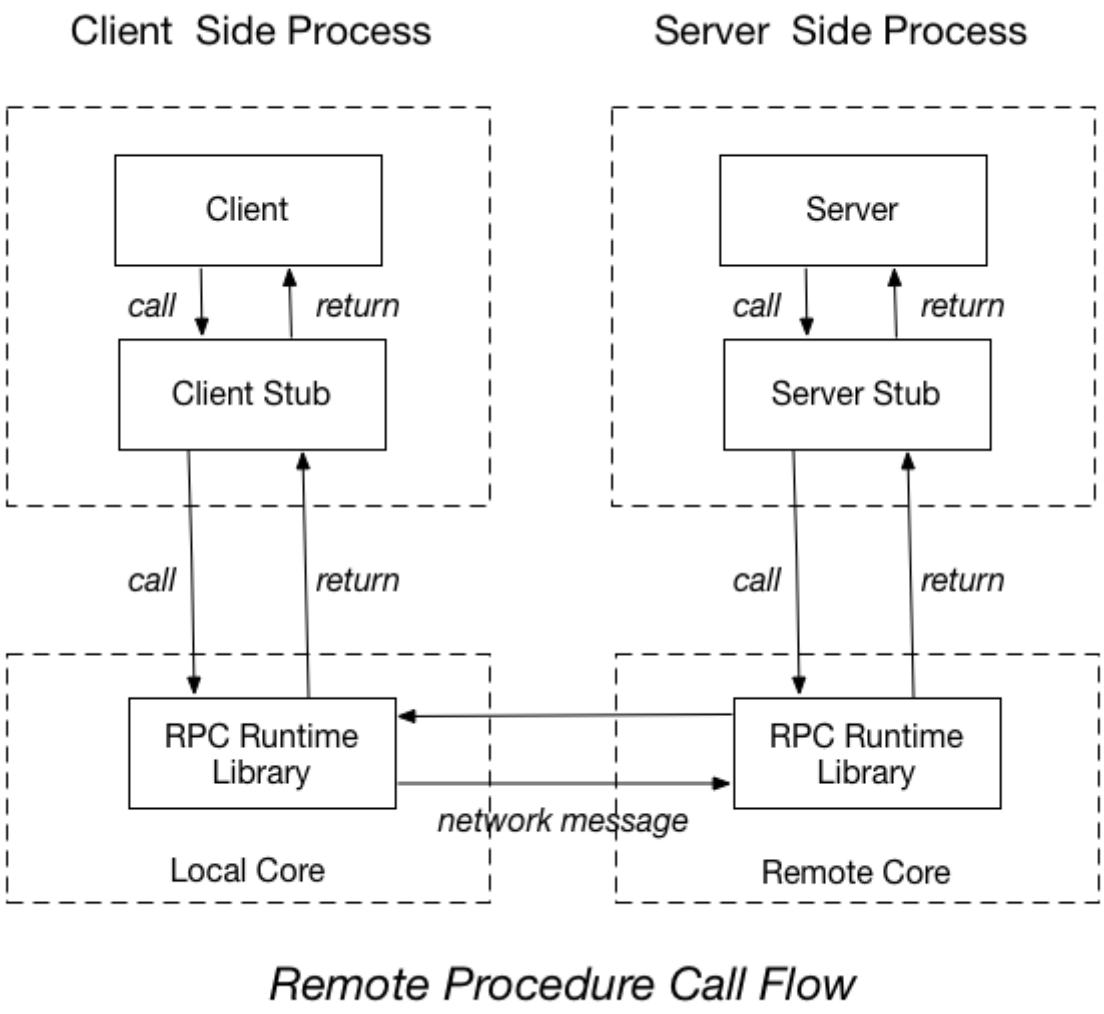
大家好，我是老胡。

许久没有更新100天搞定机器学习系列了，最近在看一个开源框架，其中有用到 gRPC ，它可以用于机器学习模型的部署，也可用于深度学习框架的开发，本文就当是《100天搞定机器学习》的番外篇吧，gRPC ，我们一起探个究竟。

gRPC（Remote Procedure Call）

gRPC 由 Google 开发，是一款语言中立、平台中立、开源的 RPC 框架。
RPC（Remote Procedure Call）即：远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。使用的时候，客户端调用server端提供的接口就像是调用本地的函数一样。

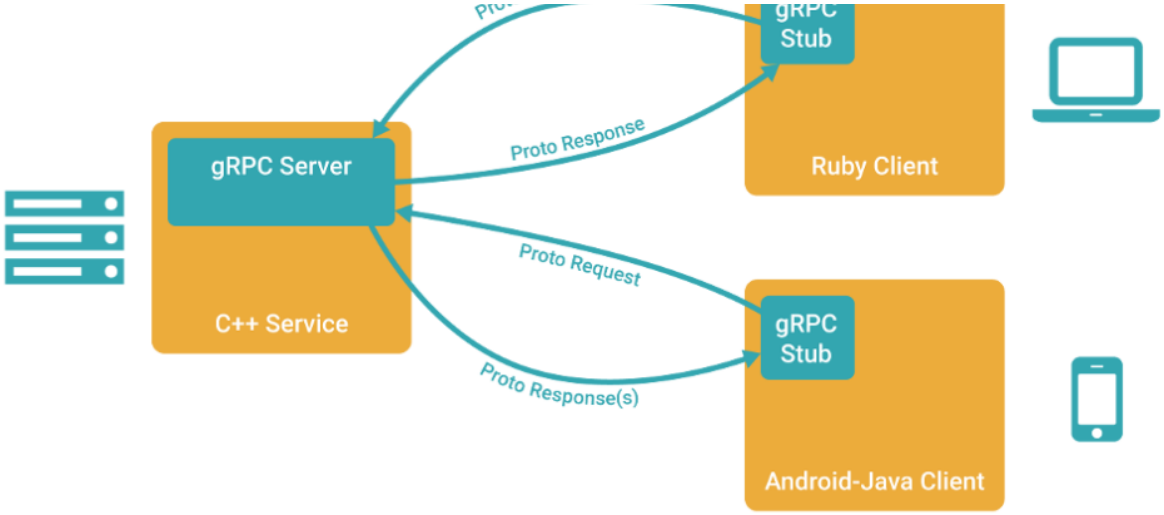
比如：有两台服务器A，B，一个应用部署在A服务器上，想要调用B服务器上应用提供的函数/方法，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义和传达调用的数据。



Remote Procedure Call Flow

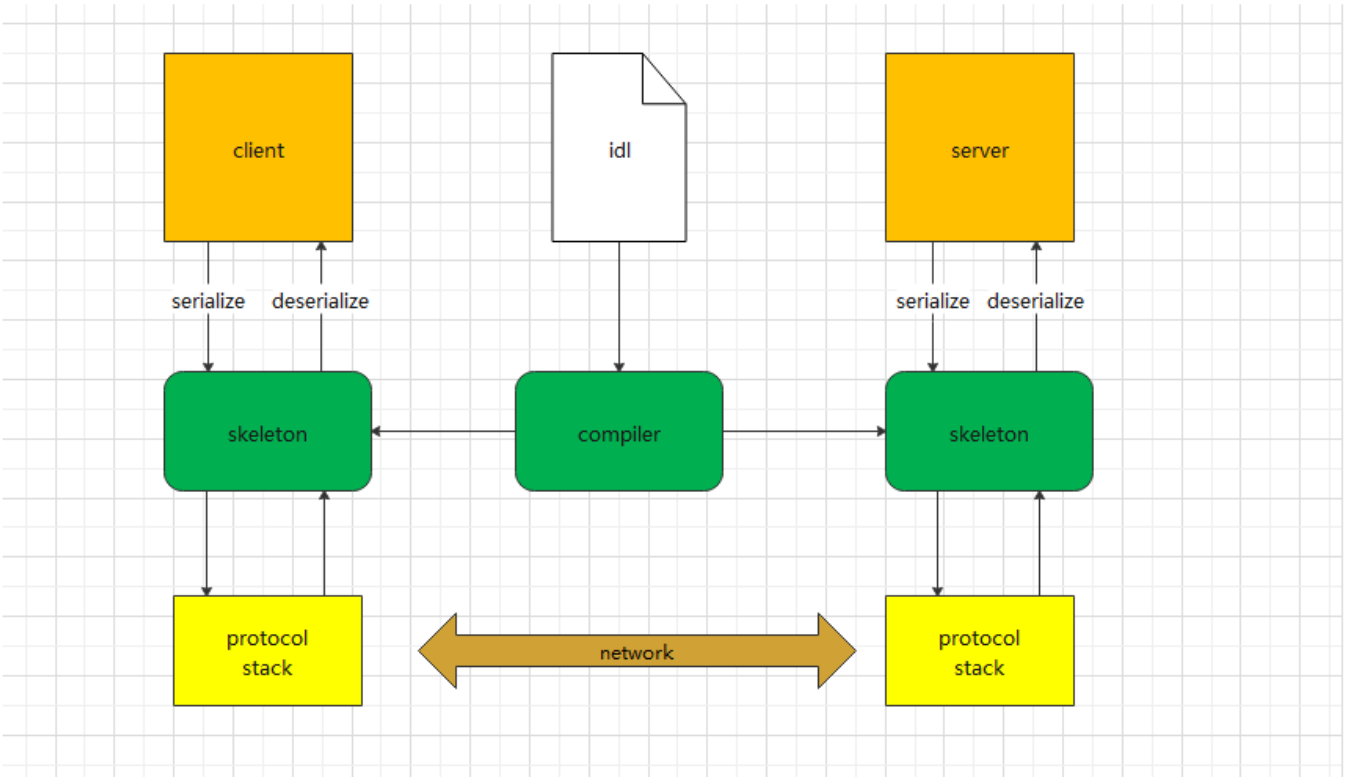
RPC更像是一种思想或机制，其实现方式有很多，除了gRPC ,还有阿里巴巴的 Dubbo、Facebook 的 Thrift、Twitter 的 Finagle 等。

gRPC 基于以下理念：定义一个服务，指定其能够被远程调用的方法（包含参数和返回类型）。在服务端实现这个接口，并运行一个 gRPC 服务器来处理客户端调用。在客户端拥有一个存根能够像服务端一样的方法。你可以很容易地用 c++ 创建一个 gRPC 服务端，用 Go、Python、Ruby 来创建客户端。



上图中的 Protocbuf 是gRPC的数据序列化工具，使用 Protobuf 将数据序列化成二进制的数据流，即可让用不同语言(proto3支持C++, Java, Python, Go, Ruby, Objective-C, C#)编写并在不同平台上运行的应用程序交换数据。ps：Protocbuf 也是 Google 开源的。

Protocol Buffer 官方提供了编译工具来对 proto 文件进行编译并生成语言相关的代码文件，可以极大地减少编码的工作量。对于序列化协议来说，使用方只需要关注业务对象本身，即 idl 定义，序列化和反序列化的代码只需要通过工具生成即可。



ProtoBuf 协议的工作流程

gRPC 实例详解——机器学习模型部署

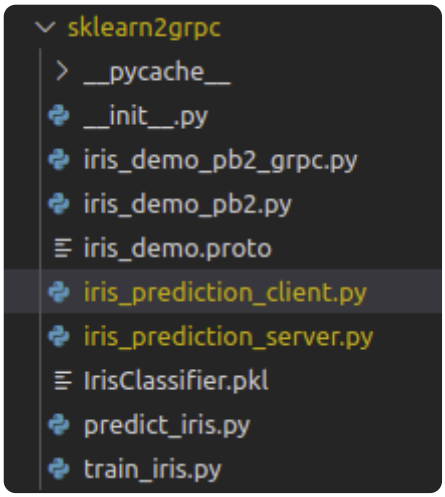
开始实例之前，需要安装 gRPC 及相关工具

```
pip install -U grpcio
pip install -U grpcio-tools
pip install -U protobuf
```

gRPC的使用通常包括如下几个步骤：

- 通过protobuf来定义接口和数据类型
- 编写gRPC server端代码
- 编写gRPC client端代码

下面我们就以Iris数据集为例，**用 gRPC server端部署一个随机森林分类器，client 端发起请求预测鸢尾花类型。**



0、训练一个随机森林分类模型，把训练好的模型保存为pkl文件。

```
# train_model.py
from sklearn import datasets
from sklearn.pipeline import Pipeline
import joblib
from sklearn.ensemble import RandomForestClassifier

def main():
    clf = RandomForestClassifier()
    p = Pipeline([('clf', clf)])
    p.fit(X, y)

    filename_p = 'IrisClassifier.pkl'
    joblib.dump(p, filename_p)
    print('Model saved!')


if __name__ == "__main__":
    iris = datasets.load_iris()
    X, y = iris.data, iris.target
    main()
```

1、通过protobuf定义接口和数据类型

新建一个iris_demo.proto文件

```
syntax = "proto3";

package iris;

message IrisPredictRequest {// 定义参数1
    float sepal_length = 1;//参数字段1
    float sepal_width = 2;//参数字段2
    float petal_length = 3;//参数字段3
    float petal_width = 4;//参数字段4
}

message IrisPredictResponse {// 定义参数1
    int32 species = 1;
}
```

```
service IrisPredictor{// 定义服务
    rpc predict_iris_species(IrisPredictRequest) returns (IrisPredictResponse){}
}
```

proto文件格式一般三部分组成，

- 头部的syntax 注明版本号为 "proto3"，必须写，没理由。
- 中间的 message 定义了predict_iris_species方法的参数IrisPredictRequest和IrisPredictResponse，还有参数字段的类型。
- 尾部的 service 定义一个服务IrisPredictor，其中包括 1 个predict_iris_species的RPC方法。这里可以定义多个RPC方法，在 message 中定义对应的参数即可。

2、使用gRPC protobuf生成Python的库函数

```
python -m grpc_tools.protoc -I=. --python_out=. --grpc_python_out=. ./iris_demo.proto
```

其中：

-I指定了源文件的路径

--python_out， 指定 xxx_pb2.py的输出路径，如果使用其它语言请使用对应语言的option

--grpc_python_out 指定xxx_pb2_grpc.py文件的输出路径

--*.proto是要编译的proto文件。

运行成功后，会自动生成iris_demo_pb2.py（里面有消息序列化类）和iris_demo_pb2_grpc.py（包含了服务器 Stub 类和客户端 Stub 类，以及待实现的服务 RPC 接口）。我们无需关心这两个py文件的细节，只需要直到在服务端和客户端怎么调用即可。

本例中，我们会用到的方法如下：

xxx_pb2.py

├── xxx_pb2.IrisPredictRequest 用于传入特征数据

├── xxx_pb2.IrisPredictResponse 用于预测

xxxx_pb2_grpc.py
├── xxx_pb2_grpc.IrisPredictorServicer 服务器 Stub 类
├── xxx_pb2_grpc.IrisPredictorStub 客户端 Stub 类

3、写一个服务器

这里的重点是定义 IrisPredictor 类的 predict_iris_species 方法，然后用 iris_demo_pb2_grpc.py 中的 add_IrisPredictorServicer_to_server 方法将 IrisPredictor 添加到 server。serve 函数里定义了 gRPC 的运行方式，使用 4 个 worker 的线程池。

```
# iris_prediction_server.py
import grpc
from concurrent import futures
import time
import joblib
import iris_demo_pb2
import iris_demo_pb2_grpc
import predict_iris
from sklearn.ensemble import RandomForestClassifier


class IrisPredictor(iris_demo_pb2_grpc.IrisPredictorServicer):

    @classmethod
    def get_trained_model(cls):
        cls._model = joblib.load('IrisClassifier.pkl')
        return cls._model

    def predict_iris_species(self, request, context):
        model = self.__class__.get_trained_model()
        sepal_length = request.sepal_length
        sepal_width = request.sepal_width
        petal_length = request.petal_length
        petal_width = request.petal_width
        result = model.predict(
            [[sepal_length, sepal_width, petal_length, petal_width]])
        response = iris_demo_pb2.IrisPredictResponse(species=result[0])
        return response  # not sure


def run():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=4))
    iris_demo_pb2_grpc.add_IrisPredictorServicer_to_server(
        IrisPredictor(), server)
    server.add_insecure_port('[::]:50055')
    server.start()
```

```
    print("grpc server start...")
    print("Listening on port 50055")
    server.wait_for_termination()


if __name__ == '__main__':
    run()
```

4、写一个客户端

客户端的逻辑更加简单，连上gRPC服务，然后发起调用。

```
# iris_prediction_client.py
import grpc
import iris_demo_pb2
import iris_demo_pb2_grpc


def run():
    channel = grpc.insecure_channel('localhost:50055')
    stub = iris_demo_pb2_grpc.IrisPredictorStub(channel)
    request = iris_demo_pb2.IrisPredictRequest(
        sepal_length=6.7,
        sepal_width=3.0,
        petal_length=5.2,
        petal_width=2.3)
    response = stub.predict_iris_species(request)
    print('The prediction is :', response.species)


if __name__ == '__main__':
    run()
```

5、调用 RPC

先开启服务端

```
$ python iris_prediction_server.py
```

```
grpc server start...

Listening on port 50055
```

另起一个terminal执行客户端代码，调用gRPC服务，预测结果如下：

```
$ python iris_prediction_client.py

The prediction is : 2
```

referance

https://grpc.io/
http://doc.oschina.net/
https://zhuanlan.zhihu.com/p/148139089 https://yu-ishikawa.medium.com/machine-learning-as-a-microservice-in-python-16ba4b9ea4ee

也可以加一下老胡的微信
围观朋友圈~~~

推荐阅读 （点击标题可跳转阅读）

论机器学习领域的内卷
机器学习博士自救指南
机器学习必知必会的 6 种神经网络类型

你见过的最全面的Python重点知识汇总

老铁，三连支持一下，好吗？↓↓↓

喜欢此内容的人还喜欢

保姆级教程：Anaconda 安装及使用

机器学习算法与Python实战