分布式系统作业

远程过程调用训练

第三次作业

姓名：郭梓煜

班级：计科2班

学号：17341046

问题描述

根据所学的RPC相关原理，实现客户端-服务器通信，并进行简单的计算如数据库查询、算术计算、数据挖掘、深度学习推导等。

- 要求：
  - 采用gRPC；
  - 采用Protobuf作为C-S数据传输格式；
  - 服务器端采用线程池，支持并发；
  - 支持至少两种的计算服务如简单的算术运算+数据挖掘算法（K-means、KNN等）；
- 这里我实现的是简单的算术运算+，-，*，/四种运算以及数据挖掘K-means、KNN两种算法。

解决方案

由于一开始对环境grpc的编程不是十分熟悉，所以阅读了官方文档，运行了第一个程序helloworld之后，逐渐了解了如何使用grpc以及protobuf，并用之进行实现通信以及进行算术运算以及数据挖掘。

- 官方代码example可直接使用git clone获取

```
git clone -b v1.24.0 https://github.com/grpc/grpc
```

环境配置

- vscode
- win10
- python3.5
- numpy
- grpc安装

```
python -m pip install grpcio
```

- grpc-tools安装

```
python -m pip install grpcio-tools
```

- sklearn安装(用于K-means)

```
python -m pip install sklearn
```

- vscode-proto3
  - 便于proto3文件编写

实验代码及解析

代码文件

主要的编程文件包括三个：

- algorithm_server.py 服务器程序
- algorithm_client.py 客户端程序
- algorithm.proto 协议

另外两个文件algorithm_pb2_grpc.py，algorithm_pb2.py均由algorithm.proto文件使用

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

命令即可生成。

实验代码解析

algorithm.proto文件：

- 使用message定义request和response的数据，字段以及其名称类型。
  - 类型包括string，int，float，stream等
  - 分配字段编号：每个字段都有一个唯一的编号。注意：最小是1，不可重复。 只有在枚举时会出现0，其余最小均是1。
  - 其中若需要使用数组，如K-means算法中，则使用repeated，具体使用方法详见客户端。
  - 不同data，result的用处在文件中有明确注释

- 使用Service定义服务，分别有Greeter，Calculate，Knn，K_means。
  - 在其中用rpc语句定义输入输出，具体服务函数。
  - 其中具体的功能将在服务器端实现
- 具体代码如下：

```
syntax = "proto3";
// Message
// The request message containing the user's name.
message HelloRequest { string name = 1; }
// The response message containing the greetings
message HelloReply { string message = 1; }
//简单算术运算的data，result
message data {
  int32 a = 1;
  int32 b = 2;
}
message result { float result = 1; }
//KNN算法的data，result
message Knn_data {
  int32 k = 1;
  string testdata = 2;
  string traindata = 3;
  string labels = 4;
}
message Knn_result { string Knn_result = 1; }
//K-means算法的data，result
message K_means_data {
  int32 num = 1;
  int32 kind = 2;
  repeated string km_data = 3;//数组
}
message K_means_result { string km_result = 1; }

//Service
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello(HelloRequest) returns (HelloReply) {}
}
//简单算术运算
service Calculate {
  // add algorithm
  rpc add(data) returns (result) {}
```

```
  // multi algorithm
  rpc multi(data) returns (result) {}
  // substract algorithm
  rpc sub(data) returns (result) {}
  // divide algorithm
  rpc div(data) returns (result) {}
}
//KNN
service Knn {
  rpc knn(Knn_data) returns (Knn_result) {}
}
//K-means
service K_means {
  rpc k_means(K_means_data) returns (K_means_result) {}
}
```

algorithm_server.py文件：

- 服务器端程序主要实现在proto中定义的服务功能以及定义serve函数，并启动rpc服务，将实现的服务加入进去。
- 不同服务的具体实现：
  - Greeter
    - 赋值proto文件中HelloReply的message其中获取来自request的用户名，然后进行返回

```
class Greeter(algorithm_pb2_grpc.GreeterServicer):
  def SayHello(self, request, context):
      print("%s join!" % request.name)
      return algorithm_pb2.HelloReply(message='Welcome to %s!' % request.name)
```

  - Calculate
    - 分别实现加减乘除四个功能
    - 其中加减乘都和前面的Greeter差不多进行赋值并返回即可，但是除要注意除数为0的情况。
    - 我原本想直接排除这种情况，但不知道如何报错，也对其中函数parameter中的context十分奇怪，上网学习一番后发现可以借助context的set_code，set_details进行报错。

```
```

```
class Calculate(algorithm_pb2_grpc.CalculateServicer):
  # 加
  def add(self, request, context):
      return algorithm_pb2.result(result=request.a+request.b)
  # 减
  def sub(self, request, context):
      return algorithm_pb2.result(result=request.a-request.b)
  # 乘
  def multi(self, request, context):
      return algorithm_pb2.result(result=request.a*request.b)
  # 除
  def div(self, request, context):
      if request.b == 0:
          context.set_code(grpc.StatusCode.INVALID_ARGUMENT)
          context.set_details("Cannot divide when the denominator is 0!")
          return algorithm_pb2.result()
      return algorithm_pb2.result(result=request.a/request.b)
```

- Knn
  - Knn算法即K最邻近分类算法
  - 实现这个功能首当其冲的难题是如何将训练集，测试集以及训练集的label从客户端输入，在服务器进行运算。我这里在proto文件中使用string来记录训练集等list，并用'_'分割开。
    - 如[[0 0 2 1] [1 2 5 4] [1 2 3 4]]表示为0021_1254_1234
  - 在服务器中先把在request获取的数据进行处理，把str化为list再化为array。
  - 之所以化为array是因为在下面的计算中可以对两个array直接相加减
  - shape--取得训练集个数，tile--将测试数据转变个数，sum(axis=1)--横向相加
  - 最后返回Knn_result，注意应转化为string类型
  - 注意：当第K邻近的两种类别数量一致时，这里会输出类别代号大的。样例中类别2，0次数均为1，这里输出2。

```
  class Knn(algorithm_pb2_grpc.KnnServicer):
      def knn(self, request, context):
          # 处理数据
          testdata = array(list(map(int, request.testdata)))
          labels = array(list(request.labels))
          traindata_l = []
```

```
            tmp = request.traindata.split('_')
            for i in range(len(tmp)):
                #str->int
                traindata_l.append(list(map(int, tmp[i])))
            traindata = array(traindata_l)
            # 训练数据个数
            # shape取的事训练数据的第一维，即其行数，也就是训练数据的个数
            traindatasize = traindata.shape[0]  # 3
            # 将测试数据转成和历史数据一样的个数  然后和训练数据相减
            # tile()的意思是给一维的测试数据转为与训练数据一样的行和列的格式
            # [[ 0  1  0  2] [-1 -1 -3 -1] [-1 -1 -1 -1]]
            dif = tile(testdata, (traindatasize, 1)) - traindata
            sqdif = dif ** 2  # [[0 1 0 4] [1 1 9 1] [1 1 1 1]]
            # axis=1 ----> 横向相加的意思
            sumsqdif = sqdif.sum(axis=1)  # [ 5 12  4]
            # 此时sumsqdif以成为一维数组
            # [2.23606798 3.46410162         2.         ]
            distance = sumsqdif ** 0.5
            # sortdistance为测试数据各个训练数据的距离按近到远排序之后的结果
            sortdistance = distance.argsort()  # [2 0 1]
            count = {}
            for i in range(0, request.k):
                vote = labels[sortdistance[i]]  # 2 0
                # vote测试数据最近的K个训练数据的类别
                count[vote] = count.get(vote, 0) + 1
            sortcount = sorted(count.items(), key=operator.itemgetter(
                1), reverse=True)  # [(2, 1), (0, 1)]
            # 2
            return algorithm_pb2.Knn_result(Knn_result=str(sortcount[0][0]))
```

- K_means
  - 即k均值聚类算法
  - 这里调用sklearn库的KMeans进行实现
    - fit()方法是对Kmeans确定类别以后的数据集进行聚类，predict()是根据聚类结果，确定所属类别。
  - 使用request.kind得到分类类数，request.num得到数据总数

```
class K_means(algorithm_pb2_grpc.K_meansServicer):
  def k_means(self, request, context):
      num = request.num
      # 处理数据
      X_tmp = []
      for i in range(0, num):
          tmp = request.km_data[i].split('_')
          X_tmp.append(list(map(eval, tmp)))
      X = array(X_tmp)
      # 对Kmeans确定类别以后的数据集进行聚类
      kmeans = KMeans(n_clusters=request.kind).fit(X)
      # 根据聚类结果，确定所属类别
      pred = kmeans.predict(X)
      print(pred)
      return algorithm_pb2.K_means_result(km_result=str(pred))
```

- serve
  - 主要实现启动rpc服务，加入服务等功能
  - 设置客户端可连接服务器的最大数量以及可连接的最长时间

```
def serve():
  # 启动rpc服务
  server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
  algorithm_pb2_grpc.add_GreeterServicer_to_server(Greeter(), server)
  algorithm_pb2_grpc.add_CalculateServicer_to_server(Calculate(), server)
  algorithm_pb2_grpc.add_KnnServicer_to_server(Knn(), server)
  algorithm_pb2_grpc.add_K_meansServicer_to_server(K_means(), server)
  server.add_insecure_port('[::]:50051')
  server.start()
  try:
      while True:
          time.sleep(60*60*24)  # one day in seconds
  except KeyboardInterrupt:
      server.stop(0)
```

algorithm_client.py文件：

- 客户端程序主要实现run()函数
- 在run函数中主要实现连接rpc服务器，调用rpc服务，并读取用户输入，发送运算数据等操作。
- rpc服务：
  - Greeter：与官方文档中的helloworld类似

```
stub = algorithm_pb2_grpc.GreeterStub(channel)
name = str(input("Please input your user name:"))
response = stub.SayHello(algorithm_pb2.HelloRequest(name=name))
print("Greeter client received: ", response.message)
```

  - 对用户进行的操作进行分类

```
op1 = int(input("Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-bre
```

  - Calculate：
    - 需要对用户执行的操作(加减乘除)进行分类
    - 不同的操作调用不同的rpc服务

```
if(op1 == 0):
  stub = algorithm_pb2_grpc.CalculateStub(channel)
      while(True):
          op = int(input(
              "Service Guide: 0-add 1-substract 2-mutiply 3-divideAny_other_keys-
          if(op != 0 and op != 1 and op != 2 and op != 3):
                  break
          num1 = int(input("The frist number: "))
          num2 = int(input("The second number: "))
          if(op == 0):# 加
              response = stub.add(algorithm_pb2.data(a=num1, b=num2))
              print("Result: ", response.result)
          elif(op == 1):# 减
              response = stub.sub(algorithm_pb2.data(a=num1, b=num2))
               print("Result: ", response.result)
```

```
            elif(op == 2):# 乘
                response = stub.multi(algorithm_pb2.data(a=num1, b=num2))
                print("Result: ", response.result)
            elif(op == 3):# 除
                response = stub.div(algorithm_pb2.data(a=num1, b=num2))
                 print("Result: ", response.result)
            else:
                break
```

- Knn
  - 同样调用rpc服务，将输入的字符串，发送给服务器进行运算，打印返回结果。
  - 一开始我想直接发送二维数组，可是发现repeated二维的赋值太过复杂，需要进行多次add()操作，所以改为字符串输入。
  - 如代码中给出的例子，输出的结果应为2。可在实验结果处加以验证。

```
elif(op1 == 1):
  stub = algorithm_pb2_grpc.KnnStub(channel)
  k = int(input("K(int): for example:2\n"))
  testdata = str(input("testdata(str): for example: 0123\n"))
  traindata = str(input("traindata(str): for example: 0021_1254_1234\n"))
  labels = str(input("labels(str): for example: 012\n"))

  response = stub.knn(algorithm_pb2.Knn_data(k=k, testdata=testdata,traindata=tra
  print("Result: ", response.Knn_result)
```

- K-means
  - 同样调用rpc服务，并得到返回结果
  - 不一样的地方在于这里的数据是一个数组和数组的大小
    - 需要用循环一步一步将字符串append进proto文件中用repeated定义的数组中
  - 最后，输入代码注释中的测试样例，结果应该为[0 1 1 1 0 1]或[1 0 0 0 1 0]，分为两类，可在实验结果验证
  - 可以自行选择list的个数以及分类类数

```
elif(op1 == 2):
      stub = algorithm_pb2_grpc.K_meansStub(channel)
      K_means_tmp = algorithm_pb2.K_means_data()
      k = int(input("num of kinds(int): for example:2\n"))
```

```
        K_means_tmp.kind = k
        k = int(input("num of data(int): for example:6\n"))
        K_means_tmp.num = k
        for i in range(0, k):
            data = str(
                input("Listdata(str): for example: 80_90_99\n"))
            K_means_tmp.km_data.append(data)
        response = stub.k_means(K_means_tmp)
        print("Result: ", response.km_result)
        '''
        6
        88_74_96_85
        92_99_95_94
        91_87_99_95
        78_99_97_81
        88_78_98_84
        100_95_100_92
        '''
```

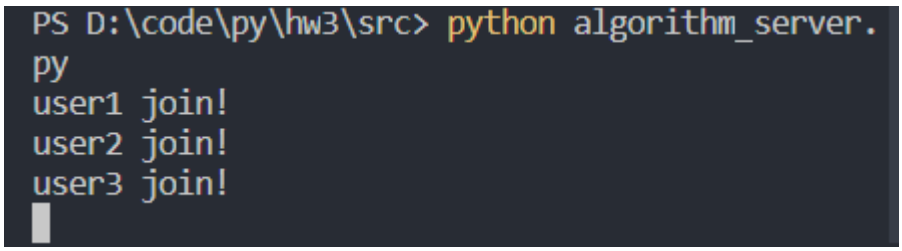
实验过程&结果

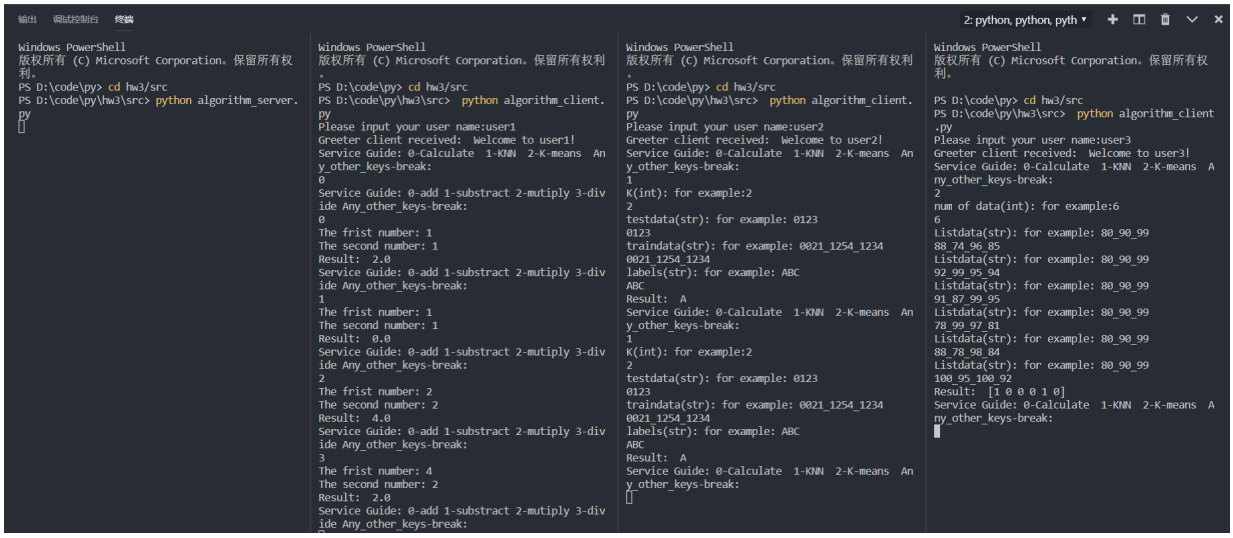
- 首先用algorithm.proto文件生成algorithm_pb2_grpc.py，algorithm_pb2.py

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权
利。
PS D:\code\py> cd hw3/src
PS D:\code\py\hw3\src> python -m grpc_tools.pro
toc --python_out=. --grpc_python_out=. -I. algo
rithm.proto
PS D:\code\py\hw3\src>
```

- 为测试是否可以多个客户端这里采用三个客户端
  - 一个进行基本算术运算
  - 一个运行Knn算法
  - 一个运行K-means算法

如下图：





- 现在对每一个算法进行验证
  - Calculate

```
PS D:\code\py> cd hw3/src
PS D:\code\py\hw3\src>  python algorithm_client.py
Please input your user name:user1
Greeter client received:  Welcome to user1!
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
0
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
0
The frist number: 1
The second number: 1
Result:  2.0
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
1
The frist number: 5
The second number: 2
Result:  3.0
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
2
The frist number: 2
The second number: 5
Result:  10.0
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
3
The frist number: 5
The second number: 2
Result:  2.5
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
```

- 测试特殊情况，除数为0

```
Service Guide: 0-add 1-substract 2-mutiply 3-divide Any_other_keys-break:
3
The frist number: 4
The second number: 0
Traceback (most recent call last):
  File "algorithm_client.py", line 76, in <module>
    run()
  File "algorithm_client.py", line 44, in run
    response = stub.div(algorithm_pb2.data(a=num1, b=num2))
  File "C:\Users\Administrator\AppData\Local\Programs\Python\Python35\lib\site-packages\grpc\_channel.py", line 604, in __call__
    return _end_unary_response_blocking(state, call, False, None)
  File "C:\Users\Administrator\AppData\Local\Programs\Python\Python35\lib\site-packages\grpc\_channel.py", line 506, in _end_unary_response_blocking
    raise _Rendezvous(state, None, None, deadline)
grpc._channel._Rendezvous: <_Rendezvous of RPC that terminated with:
        status = StatusCode.INVALID_ARGUMENT
        details = "Cannot divide when the denominator is 0!"
        debug_error_string = "{"created":"@1571932046.542000000","description":"Error received from peer ipv6:[::1]:50051","file":"src/core/lib/surface/call.cc","file_line":1055,"grpc_message":"Cannot divide when the denominator is 0!","grpc_status":3}"
>
```

会出现`Cannot divide when the denominator is 0!`的报错

- Knn

```
PS D:\code\py\hw3\src>  python algorithm_client.py
Please input your user name:user2
Greeter client received:  Welcome to user2!
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
1
K(int): for example:2
2
testdata(str): for example: 0123
0123
traindata(str): for example: 0021_1254_1234
0021_1254_1234
labels(str): for example: 012
012
Result:  2
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
```

得出结果为2，经过具体计算验证正确，具体计算过程可见代码注释。

- K-means

```
Greeter client received:  Welcome to user3!
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
2
num of kinds(int): for example:2
2
num of data(int): for example:6
6
Listdata(str): for example: 80_90_99
88_74_96_85
Listdata(str): for example: 80_90_99
92_99_95_94
Listdata(str): for example: 80_90_99
91_87_99_95
Listdata(str): for example: 80_90_99
78_99_97_81
Listdata(str): for example: 80_90_99
88_78_98_84
Listdata(str): for example: 80_90_99
100_95_100_92
Result:  [1 0 0 0 1 0]
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
2
num of kinds(int): for example:2
2
num of data(int): for example:6
6
Listdata(str): for example: 80_90_99
88_74_96_85
Listdata(str): for example: 80_90_99
92_99_95_94
Listdata(str): for example: 80_90_99
91_87_99_95
Listdata(str): for example: 80_90_99
78_99_97_81
Listdata(str): for example: 80_90_99
88_78_98_84
Listdata(str): for example: 80_90_99
100_95_100_92
Result:  [0 1 1 1 0 1]
Service Guide: 0-Calculate  1-KNN  2-K-means  Any_other_keys-break:
```

如图，输出[0 1 1 1 0 1]或[1 0 0 0 1 0]，代表第0，4元素为一类其余为另一类。经验证，结果正确。

遇到的问题及解决方法

1. 各个文件的用途以及如何编译，编写是遇到的第一个难题。总共五个文件，三个需要编写。后来通过阅读老师作业链接中的grpc官方文档，并运行helloworld程序，逐步了解到了如何编写各个文件以及如何进行编译，正式开始了编程。
2. import问题
   我使用的python版本为python3.5。一开始使用import出现了import失败的问题。到网上搜

索原因发现是python2和python3版本不同，import也不相一致的原因。

3. 数据表示问题
   在Knn算法以及K-means算法中如何表示数据也是一个重要的问题。这里用到了str的操作，在proto文件中声明string类型。还有一个难点在于如何在proto中表示数组，经过查阅，使用repeated即可实现。
4. repeated的赋值以及读取问题
   在proto中声明了数组，如何在客户端对其进行赋值以及如何在服务器端对其进行读取也是一个问题。经查阅，赋值需要先"实例化"algorithm_pb2.K_means_data()，然后对其中的kind，num，以及km_data数组用append操作赋值。具体操作如下：

```
K_means_tmp = algorithm_pb2.K_means_data()
...
K_means_tmp.kind = k
...
K_means_tmp.num = k
...
K_means_tmp.km_data.append(data)
```

   而读取时只需要直接将request.km_data当成数组使用即可。
5. 字符串处理问题
   在服务器端获取到的数据是str，需要将之转化为list，这里首先使用split以"_"分割字符串生成list，再转化为array即可。
   在读取到数据，并用之于Knn算法中时出现了bug。报错显示不能对array中是str类型的直接相加减，最后使用`map(eval, tmp)`将tmp这个list中的str转为int，然后再转为array解决了问题。
6. Knn、K-means算法实现问题
   一开始对Knn、K-means算法并不了解，都是后来到网上查阅相关blog，以及在python中的实现，然后学习其中的实现方法。其中Knn算法用到了许多之前没了解过的函数来处理array，K-means则直接调用sklearn库的KMeans，fit，predict来聚类，确定类别。
7. context问题
   一开始在server端看到helloworld中函数parameter中出现的context十分奇怪，不知道有什么用，不知道其存在的意义。后来上网学习一番后发现可以借助context的set_code，set_details进行报错。另外，报错不能像原来python中使用raise进行弹出错误，在这里应该使用context.set_code,set_details进行报错。
8. 这一次作业在学习过程中也遇到小小的困难。在网上搜索基本找不到解决方法，只有学会阅读相关的官方使用文档才能进行进一步的编程。所以学会阅读官方文档也十分重要。

心得体会

这一次作业做起来不难，只是一开始入手比较难，需要阅读官方文档，运行example才有进一步的了解。不过后来就容易许多了，由于使用python作为编程语言，许多都可以调库，轻松的实现算法。经过这次实验，我对grpc的了解更加深入了，还有对数据挖掘算法也有深入的了解，以及如何使用protobuf传输数据也更加熟悉了，总的来说还算是获益匪浅。