腾讯云 TI 平台 TI-ONE

算法手册

产品文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

算法手册
框架及算法说明
输入
数据源

最近更新时间：2021-12-22 12:09:04

腾讯云 TI 平台 TI-ONE 提供两种数据源途径：**本地数据**和 **COS 数据集**。
您可在左侧导航栏的【输入】>【数据源】下，选择需要的数据源。

本地数据

您可以将轻量本地文件（不超过256MB）上传到目标 COS 路径，为下游算法提供输入数据。

- 您需要有上传目标目录的写权限。
- 目标 COS 路径自动生成，如需自定义可修改输入指定路径。

COS 数据集

> ⚠ 注意：
> 使用 COS 数据集的前提是必须要开通 COS 服务，相关详细入门指引请参考 COS 入门。

1. 通过 COS 数据集上传文件的方式：

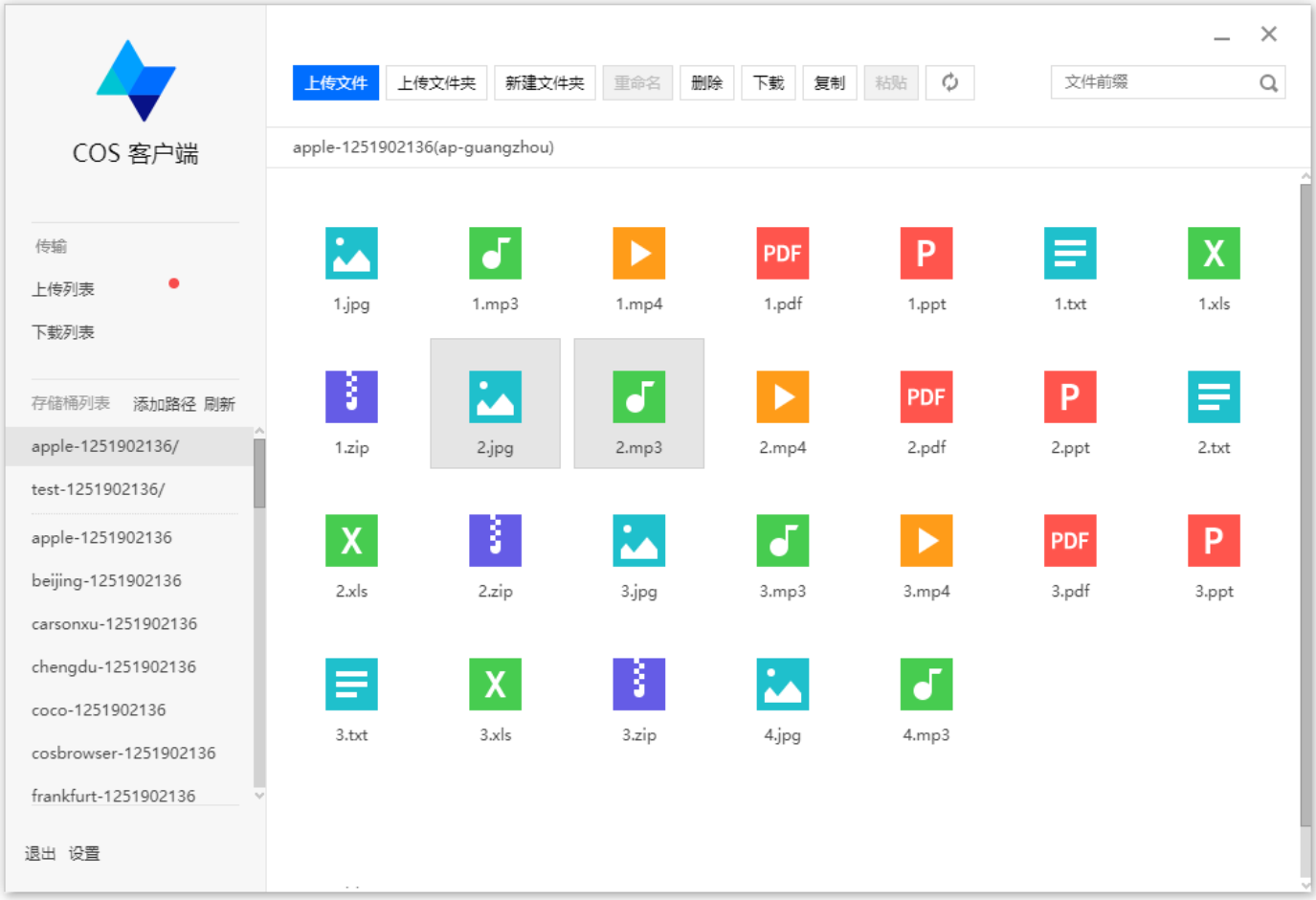
方式一：通过控制台上传数据文件（适合轻量文件上传）

1. 登录 COS 控制台。
2. 在左侧导航栏处，单击【存储桶列表】。
3. 在存储桶列表页，单击左上角【创建存储桶】。详情请参见 创建存储桶 。
4. 上传文件或点选文件夹。详情请参见 上传对象 。

方式二：通过客户端上传数据文件（适合大文件上传，支持批量上传下载）

1. 前往 COSBrowser 工具 下载 COS 客户端。
2. 使用云 API 密钥 SecretId 和 SecretKey 登录 COS 客户端。
   云 API 密钥可在 访问管理控制台 获取，登录后会保留登录信息。

3. 创建存储桶，并上传文件。



方式三：其他上传方式

COS 提供更多开发者上传工具，支持将数据从其他分布式存储迁移至 COS。您可以在 COS 工具概览 中选择合适的上传工具，并按照对应的说明文档进行操作。

2. COS 路径命名规则

- **工作流画布节点路径**（如**本地上传、COS 数据源**和其他算法的 IO 路径中）
  不需要填写存储桶名称，示例：${cos}/data/train.txt
- **通过代码访问 COS 数据文件**（如 **PySpark** 等组件中）
  当用户通过代码访问 COS 上的文件时，需要加前缀/cos_person/ ，与 Spark 相关路径需加前缀 file:///cos_person/。例如，在 Python 中读取上例中的 train.txt 文件：

```
with open("/cos_person/data/forrest/demo/in/train.txt") as f:
for line in f:
print(line)
```

在 PySpark 代码中访问 train.txt 文件：

```
train_data = sc.textFile("file:///cos_person/data/forrest/demo/in/train.txt")
```

3. 数据格式

数据格式总体分为 **Dense** 和 **Libsvm** 两种，您需将数据文件转化为对应的 txt 格式。

Dense

Dense 数据格式每行对应一个样本，每列对应一个特征，在 COS 中每行样本的各列以**空格**连接，如下：

```
10.2 12.8 3.67 ...
25.9 55.9 29.0 ...
7.89 0.89 14.5 ...
```

Libsvm

Libsvm 数据格式是数据的一种稀疏表达方式，仅支持标准的 Libsvm 数据格式。Libsvm 数据格式作为输入数据时无需刻意指明，系统通过模式匹配自动识别数据是 Dense 还是 Libsvm 格式。每行中的元素以**空格**连接，如下：

```
1 1:0.5 3:3.1 7:1.0
0 2:0.1 3:2.3 5:2.0
2 4:0.2 7:1.1 9:0.0
```

- 标准 Libsvm 数据格式要求：数据中的 index 必须是**从1开始计数**，且以**升序**排列。
- 特征数目：稀疏表达的数据需要指定特征数，特征列参数中指定"1-featureSize"。如“1 - 19”表示特征数有19维。若不确定参数数目，可以参数选择列为空。

数据转换

最近更新时间：2020-07-28 10:34:00

中文分词

分词，将中文句子划分成词语，词语之间用空格分隔。

输入参数

输入数据：未分词的中文文本，每行为一个句子。

输出参数

输出数据：分词后的中文文本，每行为一个句子，词与词之间用空格分隔。

算法参数

- 标签分隔符：（可选）如行中有标签分隔符，则只对标签分隔符左侧的文本进行分词。
- 句子分隔符：（可选）句子与句子间的分隔符，用户可输入自定义分隔符，如果其为制表符，则输入“tab”。

实例生成

1. 使用数据节点，上传数据，数据格式见上文“输入数据”部分。
2. 将数据节点连接到分词节点，配置好输出数据路径和标签分隔符，单击【运行】开始分词。

中文去停用词

去除文本中的停用词，文本需要预先分词。

输入参数

- 输入数据：分词后的中文文本，每行为一个句子，词语之间用空格分隔。
- 停用词表：要去除的停用词列表，每行为一个词。

输出参数

输出数据：去除停用词后的中文文本，每行为一个句子。

算法参数

标签分隔符：（可选）如行中有标签分隔符，则只对标签分隔符左侧的文本进行去停用词。

实例生成

1. 使用数据节点，上传输入数据和停用词表数据，数据格式见上文“输入数据”部分。
2. 将数据节点连接到去停用词节点，配置好输出数据路径和标签分隔符，单击【运行】开始去停用词。

句子转向量表示

句子转向量表示可以将句子转换为向量表示，转换方法有三种：

- 转换为词向量平均值。
- 转换为词向量加权平均值，权重由词频决定，词频越高，权重越低。
- 使用论文 A simple but tough-to-beat baseline for sentence embeddings 的方法进行转换。

输入参数

输入数据：文本文件，每一行为一个句子，词语之间使用空格分隔。

输出参数

输出数据：文本文件，每一行为输入数据对应行的句子的向量表示，数字之间用空格分隔。

算法参数

- 预训练词向量文本文件，格式请参考 word2vec 和 glove 算法的【输出参数】部分。
- 转换方式：三选一，average 对应词向量平均值；weighted_average 对应词向量加权平均值；svd 对应论文中的方法。

实例生成

1. 使用数据节点上传数据，数据格式请参考“输入数据”部分。
2. 将数据节点连接到 sentence2vec 节点，配置好预训练词向量路径和转换方式，单击【运行】开始训练。

文本数据切分

该模块将 NLP 任务的输入文件按一定比例切分成训练集和验证集两份文件。输入数据中每行为一个样本，用分隔符区分特征和标签，eg. spiderman rocks.__label__1，此时分隔符为'__label__'。

输入参数

输入数据：要切分的文本文件，每行为一个句子。

输出参数

- 切分后文件1：切分后的第一份文件。
- 切分后文件2：切分后的第二份文件。

算法参数

- 切分比：第一份文件所占的比例。
- 分隔符：分隔一行中句子和标签的分隔符，如果某些标签仅在两份切分后文件之一中出现，则日志中会有提示。
- 确保第一份输出包含所有标签：选择是否确保第一份输出包含所有标签。

实例生成

1. 使用数据节点，上传文本数据，数据格式见上文“输入数据”部分。
2. 将数据节点连接到文本数据切分节点，配置好输出数据路径，切分比和分隔符，单击【运行】开始。

分类检测图像转换

对图片分类或者目标检测的数据集进行切分，并转换成 tfrecord 格式，同时生成表示标签和类别索引对应关系 label_map 文件，供算法节点调用。

算法 IO 参数

- 图像目录：对于图片分类任务，目录中需包含若干个子目录，子目录名字为类别名；对目标检测任务而言，该目录中需要包含所有图像。
- xml 文件目录：仅目标检测任务需要填写。存放 xml 文件的目录，xml 需要为 pascal voc 格式，且为 utf-8 编码。
- 训练集输出：输出训练集的 tfrecord 文件的路径。
- 验证集输出：输出验证集的 tfrecord 文件的路径。
- label_map 文件输出路径：label_map 文件的输出路径。

算法参数

- 任务类型：图片分类或目标检测
- 是否进行切分：是否对数据集进行切分。如果不切分，所有数据将会转换成 tfrecord，并输出到【训练集输出】目录下。
- 验证集比例：如果进行切分，切分成验证集的数据比例。

实例生成

将输入数据连接到算子的输入桩，单击运行进行切分和转换。

图像分割数据转换

算法说明

图像分割数据转换算子可以将 VOC 格式的图像分割数据转换为 tfrecord 格式，供平台的 deeplab 算法训练和验证使用。

输入参数

- JPG 图像目录：存放 JPG 格式的图像的目录，图像和对应标注的文件名（除后缀名外）应该相同。
- PNG 图像目录：存放 PNG 格式的标注的目录，标注文件需要为单通道灰度图，每个像素的取值为类别 ID。
- 列表文件目录：放置 train.txt 和 val.txt 两个文件的目录。两个文件中每一行应为不含后缀的文件名，表明某张图片是否用于训练/验证。

输出参数

- 训练数据输出目录：输出训练数据（tfrecord文件）的目录。
- 验证数据输出目录：输出验证数据（tfrecord文件）的目录。

实例生成

1. 使用数据节点，上传三份所需数据，数据格式见上文“输入数据”部分。
2. 将数据节点连接到图像分割数据转换节点，配置好输出数据路径，单击【运行】开始分词。

视频分类数据转换

算法说明

本算子可以将 UCF101 格式的视频分类数据转换为 tfrecord 格式，可供平台的 I3D 算法训练和验证使用。

算法 IO 参数

- 原始视频文件路径：存放原始视频文件的路径。
- 原始数据处理之后保存路径：存放处理后的输出文件的路径。

算法参数

训练集测试集划分比例：训练集比测试集的倍数，如按 4:1 划分，则填写 4 。

框架
PyCaffe

最近更新时间：2021-12-22 12:09:21

Caffe（Convolutional Architecture for Fast Embedding）是一个高效的深度学习框架，具有易上手、速度快、效率高、社区好等优势。

版本说明

Pycaffe 组件内核是 Caffe 1.0 版本。
Pycaffe 组件中使用的 Python 版本和支持的第三方模块信息如下：

- Python 2.7.12
- SciPy 0.17.0
- NumPy 1.11.0

如果您需要使用其他第三方的 lib，可使用 pip 在代码内安装，示例如下：

```
import pip
pip.main(['install', "package_name"])
```

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择**框架>深度学习**列表下的 **PyCaffe** 节点，将其拖拽至画布中。
2. **配置参数**
   - 脚本及依赖包文件上传：
     将任务脚本上传至程序脚本框。如果需要依赖文件，则压缩为 zip 文件后通过依赖包文件框上传。
   - 程序依赖：
     指定位于 COS 中的用户依赖文件路径，指定内容将被拷贝到程序脚本同一级目录下。支持目录或者文件依赖，若指定多个文件则以英文逗号分隔 。
   - 程序参数：
     指定运行任务脚本的参数。
3. **配置资源**
   在**资源参数**列表框中配置任务的资源参数。
4. **运行**
   单击**保存**并运行工作流。

5. 查看 PyCaffe 控制台和日志

在 PyCaffe 节点上单击右键，可查看任务状态和详细日志。

代码示例

以下代码将向您展示，在 PyCaffe 框架中训练 mnist 手写数字识别的方法。

```
import os
import argparse

import caffe
from caffe import layers as L, params as P
from caffe.proto import caffe_pb2

parser = argparse.ArgumentParser()
parser.add_argument("--save_dir", type=str, default="/cos_person",
help="Directory to save the trained model.")
parser.add_argument("--data_dir", type=str, default=None,
help="Directory which contains two subdirs: "
"`mnist_train_lmdb` and `mnist_test_lmdb`")
parser.add_argument("--max_iter", type=int, default=10000,
help="Number of training steps.")
args = parser.parse_args()


def lenet(lmdb, batch_size, include_accuracy=False, deploy=False):
# our version of LeNet: a series of linear and simple nonlinear
# transformations
n = caffe.NetSpec()
if not deploy:
n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB,
source=lmdb,
transform_param=dict(scale=1. / 255), ntop=2)
else:
n.data = L.Input(
input_param={'shape': {'dim': [batch_size, 1, 28, 28]}})

n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20,
```

```
weight_filler=dict(type='xavier'))
n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50,
weight_filler=dict(type='xavier'))
n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.fc1 = L.InnerProduct(n.pool2, num_output=500,
weight_filler=dict(type='xavier'))
n.relu1 = L.ReLU(n.fc1, in_place=True)
n.score = L.InnerProduct(n.relu1, num_output=10,
weight_filler=dict(type='xavier'))
if not deploy:
if include_accuracy:
n.accuracy = L.Accuracy(n.score, n.label)
n.loss = L.SoftmaxWithLoss(n.score, n.label)

return n.to_proto()


def train():
# Define paths
if not os.path.exists(args.save_dir):
os.makedirs(args.save_dir)
save_path = args.save_dir
train_net_path = '{}/lenet_train.prototxt'.format(save_path)
test_net_path = '{}/lenet_test.prototxt'.format(save_path)
deploy_net_path = '{}/lenet_deploy.prototxt'.format(save_path)
solver_file = '{}/lenet_solver.prototxt'.format(save_path)

lmdb_data_path = args.data_dir
train_lmdb_path = '{}/mnist_train_lmdb'.format(lmdb_data_path)
test_lmdb_path = '{}/mnist_test_lmdb'.format(lmdb_data_path)

lenet_snapshot_prefix = '{}/lenet'.format(save_path)

# Generate net prototxt files
with open(train_net_path, 'w') as f:
f.write(str(lenet(train_lmdb_path, 64)))
with open(test_net_path, 'w') as f:
```

```
f.write(str(lenet(test_lmdb_path, 100, include_accuracy=True)))
with open(deploy_net_path, 'w') as f:
f.write(str(lenet(None, 1, deploy=True)))

# Generate solver prototxt file
s = caffe_pb2.SolverParameter()
# Set a seed for reproducible experiments:
# this controls for randomization in training.
s.random_seed = 0xCAFFE
# Specify locations of the train and (maybe) test networks.
s.train_net = train_net_path
s.test_net.append(test_net_path)
s.test_interval = 500 # Test after every 500 training iterations.
s.test_iter.append(100) # Test on 100 batches each time we test.
s.max_iter = args.max_iter # no. of times to update the net (training iterations)
# EDIT HERE to try different solvers
# solver types include "SGD", "Adam", and "Nesterov" among others.
s.type = "SGD"
# Set the initial learning rate for SGD.
s.base_lr = 0.01 # EDIT HERE to try different learning rates
# Set momentum to accelerate learning by
# taking weighted average of current and previous updates.
s.momentum = 0.9
# Set weight decay to regularize and prevent overfitting
s.weight_decay = 5e-4
# Set `lr_policy` to define how the learning rate changes during training.
# This is the same policy as our default LeNet.
s.lr_policy = 'inv'
s.gamma = 0.0001
s.power = 0.75
# EDIT HERE to try the fixed rate (and compare with adaptive solvers)
# `fixed` is the simplest policy that keeps the learning rate constant.
# s.lr_policy = 'fixed'
# Display the current training loss and accuracy every 1000 iterations.
s.display = 1000
# Snapshots are files used to store networks we've trained.
# We'll snapshot every 5K iterations -- twice during training.
s.snapshot = 5000
```

```
s.snapshot_prefix = lenet_snapshot_prefix
# Train on the GPU
s.solver_mode = caffe_pb2.SolverParameter.GPU
# Write the solver to a file.
with open(solver_file, 'w') as f:
f.write(str(s))

# Train
caffe.set_device(0)
caffe.set_mode_gpu()
solver = caffe.SGDSolver(solver_file)
solver.solve()


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

PyTorch

最近更新时间：2021-12-22 12:09:31

PyTorch 是一个设计精良的深度学习框架。它由 Facebook 的人工智能研究小组在2016年开发，更多详细介绍您可参考 PyTorch 文档。

版本说明

PyTorch 组件中使用的 Python 版本和支持的第三方模块版本信息如下：

- Python 3.6
- SciPy 1.0.0
- NumPy 1.14.3

如果您需要使用其他第三方的 lib，可使用 pip 在代码内安装，示例如下：

```
from pip._internal import main
main(['install', "package_name"])
```

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择**框架>深度学习**列表下的 **PyTorch** 节点，将其拖拽至画布中。
2. **配置参数**
   - 脚本及依赖包文件上传：
     将任务脚本上传至程序脚本框。如果需要依赖文件，则压缩为 zip 文件后通过 依赖包文件 框上传。
   - 程序依赖：
     指定位于 COS 中的用户依赖文件路径，指定内容将被拷贝到程序脚本同一级目录下。支持目录或者文件依赖，若存在多个文件则以英文逗号分隔 。
   - 程序参数：
     指定运行任务脚本的参数。
3. **配置资源**
   在**资源参数**列表框配置任务的资源参数。
4. **运行**
   单击**保存**并运行工作流。
5. **查看 PyTorch 控制台和日志**
   在 PyTorch 节点上单击右键菜单可查看任务状态和详细日志。

代码示例

以下代码展示了在 PyTorch 框架中，调用 torch.nn 构建一个典型神经网络（NN）的方法。
输入：

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

def __init__(self):
super(Net, self).__init__()
# 1 input image channel, 6 output channels, 5x5 square convolution
# kernel
self.conv1 = nn.Conv2d(1, 6, 5)
self.conv2 = nn.Conv2d(6, 16, 5)
# an affine operation: y = Wx + b
self.fc1 = nn.Linear(16 * 5 * 5, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)

def forward(self, x):
# Max pooling over a (2, 2) window
x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
# If the size is a square you can only specify a single number
x = F.max_pool2d(F.relu(self.conv2(x)), 2)
x = x.view(-1, self.num_flat_features(x))
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
x = self.fc3(x)
return x

def num_flat_features(self, x):
size = x.size()[1:] # all dimensions except the batch dimension
num_features = 1
for s in size:
num_features *= s
return num_features
```

```
net = Net()
print(net)
```

输出：

```
Net(
(conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
(fc1): Linear(in_features=400, out_features=120, bias=True)
(fc2): Linear(in_features=120, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

Spark

最近更新时间：2020-12-22 17:16:35

操作场景

Spark 框架面向使用 Scala/Java 的 Spark 用户，用户编写 Spark 应用程序并编译打包成 jar 后，可通过 Spark 框架完成部署。

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择【框架】>【机器学习】列表下的【Spark】节点，并将其拖拽至画布中。
2. **配置参数**
   - 作业 Jar 包：通过该配置框上传您的 Spark 应用程序 Jar 包，必填项。
   - 主类名：指定您的 Spark 应用程序的入口类，即 main 函数所在的类，必填项。
   - 程序参数：您的 Spark 应用程序所需的参数，即传给 main 函数的参数，可选项。
   - 配置文件：指定您的 Spark 应用程序用到的配置文件，可选项。
3. **配置资源**
   在资源参数列表配置任务的资源参数。
   - driver 节点资源类型：请选择合适的 drive 节点机型。
   - executor 节点资源类型：请选择合适的 executor 节点机型。
   - num-executors：分配计算节点数目。
   - spark-conf：指定 Spark 常用参数配置，如压缩、序列化、网络等。
4. **运行**
   单击【保存】并运行工作流。
5. **查看 Spark 控制台和日志**
   在 Spark 节点上单击右键菜单，可查看任务状态和详细日志。

Tensorflow

最近更新时间：2020-08-12 14:28:39

Tensorflow 框架为用户提供了基于 Python API 的 Tensorflow 运行环境，用户可将编写好的脚本及依赖文件上传至该框架进行算法训练。

版本说明

Tensorflow 框架版本及框架中使用的 Python 版本和支持的第三方模块版本信息如下：

Tensorflow 版本	Python 版本	scipy 版本	numpy 版本
tensorflow 2.0	Python 3.5	scipy 1.1.0	numpy 1.18.5
tensorflow 1.14	Python 3.5	scipy 1.1.0	numpy 1.15.4
tensorflow 1.12	Python 3.5	scipy 1.1.0	numpy 1.15.4

如果您需要使用其他第三方的 lib，可使用 pip 在代码内安装，示例如下：

```
from pip._internal import main
main(['install', "package_name"])
```

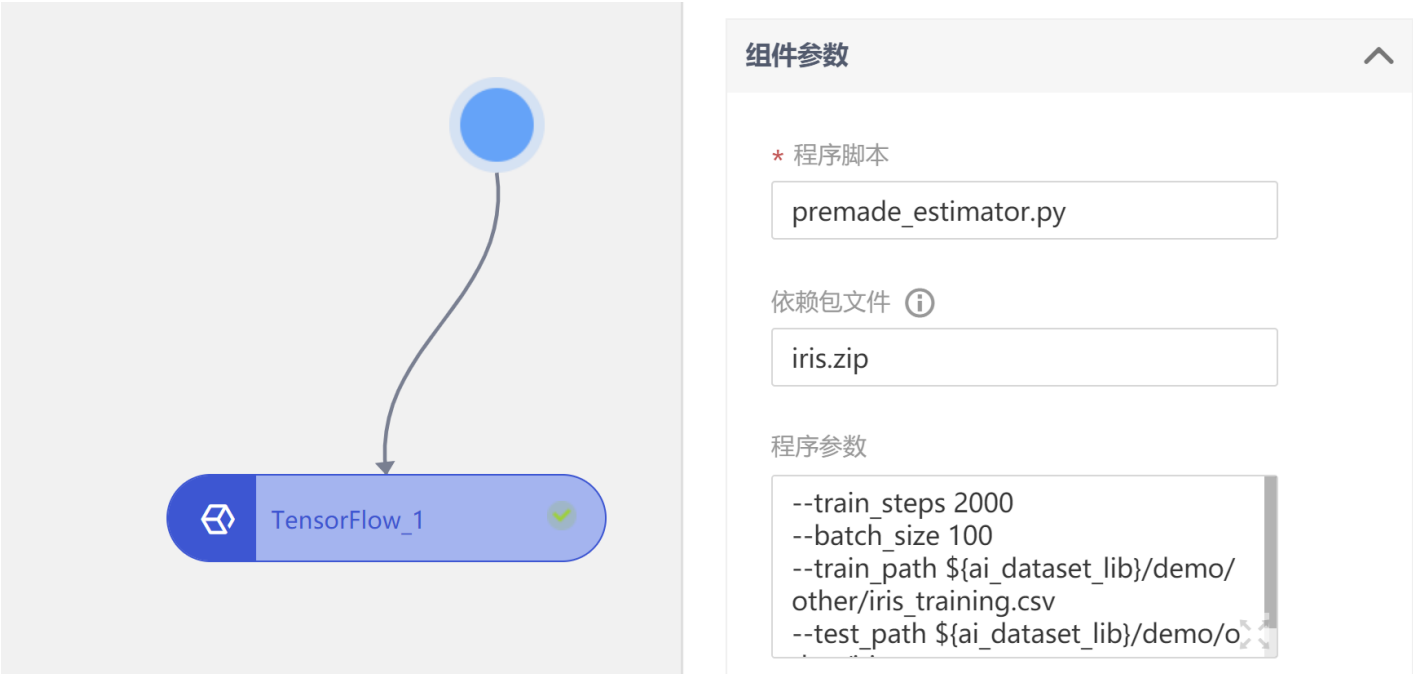
操作步骤

1. **添加组件**
   从左侧菜单栏中，选择【框架】>【深度学习】列表下的【Tensorflow 】节点，并将其拖拽至画布中。
2. **配置参数**
   - 脚本及依赖包文件上传 ： 将任务脚本上传至程序脚本框。如果需要依赖文件，则压缩为 zip 文件后通过 依赖包文件 框上传。
   - 程序依赖：指定位于 COS 中的用户依赖文件路径，指定内容将被拷贝到程序脚本同一级目录下。支持目录或者文件依赖，若存在多个文件则以英文逗号分隔 。
   - 程序参数：指定运行任务脚本的参数。
   - TensorBoard 目录：指定 Tensorboard 保存路径。
3. **配置资源**
   在【资源参数】列表框配置任务的资源参数。
4. **运行**
   单击【保存】并运行工作流。

5. **查看 Tensorflow 控制台和日志**
   在 Tensorflow 节点上单击右键菜单，可查看任务状态和详细日志。
6. **查看 Tensorboard**
   组件处于“运行中”状态时，您可以右键单击任务栏，通过【Tesnorflow 控制台】>【Tensorboard】查看 Tensorboard 信息。

案例说明

本案例提供一段代码，向您演示如何利用 TensorFlow 框架运行自定义代码，如何通过工作流页面向自定义代码传参，如何查看代码日志/报错信息等。
本案例代码修改自 TensorFlow 的官方项目。



1. 程序的入口脚本为 premade_estimator.py。单击【Tensorflow 框架】的【程序脚本】输入框，选择 premade_estimator.py 脚本文件上传。
2. 如果入口脚本需要 import 项目中的其它自己编写的模块，需要将其它模块的代码压缩成 zip 包，并上传到【Tensorflow】组件的【依赖包文件】中，该 zip 包会被添加到 Python 的 path 中。在本 demo 中，我们将 iris_data.py 和 estimator_test.py 两个文件压缩成 iris.zip，上传至【依赖包文件】中，此时，程序脚本中可以使用import iris_data 来引入这一模块。
3. 如果入口脚本需要启动参数，如本 demo 中，入口脚本 premade_estimator.py 可以接收--batch_size，--train_steps，--train_path 和--test_path 四个参数，则将参数及其取值填写到【程序参数】输入框中。
   示例代码为：
   --train_steps 2000
   --batch_size 100

--train_path ${ai_dataset_lib}/demo/other/iris_training.csv
--test_path ${ai_dataset_lib}/demo/other/iris_test.csv

4. 如果自行编写的 Tensorflow 代码会产生用于 Tensorboard 展示的文件（如 events 文件），则可以在自定义的代码中，将这些文件输出到 COS 的某个特定目录，并在【Tensorboard 目录】输入框中填写该目录的路径。如果填写了，训练过程中可以在【Tensorflow 控制台】>【Tensorboard】中查看 Tensorboard。
5. 运行后，可以右键单击算子，并在【Tensorflow 控制台】>【App 详情】中查看 stdout 和 stderr 两个日志。

PySpark

最近更新时间：2020-12-22 17:13:17

PySpark 包含标准 Spark 的功能，同时支持上传 Python 脚本、实时修改脚本和 SQL 功能，更加灵活，我们推荐您使用 PySpark 进行数据预处理。

版本说明

PySpark 框架中使用的 Python 版本和支持的第三方模块版本信息如下：

- Python 2.7/3.5
- SciPy 0.12.1
- NumPy 1.7.1

如果您需要使用其他第三方的 lib，可使用 pip 在代码内安装。
python2 /python3 安装示例如下，您需要将 package_name 换成自己的包名。

```
from pip._internal import main
main(['install', "package_name"])
```

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择【框架】>【机器学习】列表下的【PySpark 】节点，并将其拖拽至画布中。
2. **配置参数**
   - 脚本及依赖包文件上传：
     将任务脚本上传至程序脚本框。如果需要依赖文件，则压缩为 zip 文件后通过依赖包文件框上传。
   - 算法参数：指定您的 PySpark 应用程序所需的参数，即传给 PySpark 脚本的参数，可选项。
   - 配置资源：指定您的 PySpark 应用程序用到的配置文件，可选项。
3. **配置资源**
   在【资源参数】列表框配置任务的资源参数。
   - drver 节点资源类型：请选择合适的 drive 节点机型。
   - executor 节点资源类型：请选择合适的 executor 节点机型。
   - num-executors：分配计算节点数目。
   - spark-conf：指定 Spark 常用参数配置，如压缩、序列化、网络等。
4. **运行**
   单击【保存】并运行工作流。

5. 查看 PySpark 控制台和日志
在 PySpark 节点上单击右键菜单，可查看任务状态和详细日志。

使用建议

使用 PySpark 的目的是更好地借助其分布式计算的优势，解决单机完成不了的计算。如果您在 PySpark 中仍然调用常规的 Python 库做单机计算，那就失去了使用 PySpark 的意义。下面举例说明如何编写 PySpark 分布式计算代码。

使用 Spark 的 DataFrame，而不要使用 Pandas 的 DataFrame

PySpark 本身就具有类似 pandas.DataFrame 的 DataFrame，所以直接使用 PySpark 的 DataFrame 即可，基于 PySpark 的 DataFrame 的操作都是分布式执行的，而 pandas.DataFrame 是单机执行的，例如：

```
...
df = spark.read.json("examples/src/main/resources/people.json")
df.show()
# +----+-------+
# | age| name|
# +----+-------+
# |null|Michael|
# | 30| Andy|
# | 19| Justin|
# +----+-------+

pandas_df = df.toPandas() # 将 PySpark 的 DataFrame 转换成 pandas.DataFrame，并获取'age'列
age = pandas_df['age']
...
```

df.toPandas() 操作会将分布在各节点的数据全部收集到 Driver上，再转成单机的 pandas.DataFrame 数据结构，适用于数据量很小的场景，如果数据量较大时，则此方法不可取。
PySpark的DataFrame 本身支持很多操作，直接基于它实现后续的业务逻辑即可，例如上述代码可以改成age = df.select('age')。

在 Task 里使用 Python 库，而不是在 Driver上 使用 Python 库

下面有段代码，将数据全部 collect 到 Driver 端，然后使用 sklearn 进行预处理。

```
from sklearn import preprocessing
data = np.array(rdd.collect(), dtype=np.float)
normalized = preprocessing.normalize(data)
```

上述代码实际上已退化为单机程序，如果数据量较大的话，collect 操作会把 Driver 的内存填满，甚至 OOM（超出内存），通常基于 RDD 或 DataFrame 的 API 可以满足大多数需求，例如标准化操作：

```
from pyspark.ml.feature import Normalizer

df = spark.read.format("libsvm").load(path)

# Normalize each Vector using $L^1$ norm.
normalizer = Normalizer(inputCol="features", outputCol="normFeatures", p=1.0)
l1NormData = normalizer.transform(dataFrame)
```

如果 RDD 或 DataFrame 没有满足您要求的 API，您也可以自行写一个处理函数，针对每条记录进行处理：

```
# record -> other record
def process_fn(record):
# your process logic
# for example
# import numpy as np
# x = np.array(record, type=np.int32)
# ...

# record -> True or Flase
def judge_fn(record):
# return True or Flase

processed = rdd.map(process_fn).map(lambda x: x[1:3])
filtered = processed.filter(judge_fn)
```

process_fn 或 judge_fn 会分发到每个节点上分布式执行，您可以在 process_fn 或 judge_fn 中使用任何 Python 库（如 numpy、scikit-learn 等）。

更多关于 Spark 的使用可以参考以下文档：

- RDD
- DataFrame
- Python API

TensorFlow PS

最近更新时间：2020-08-14 16:10:21

机器学习系统相较其他系统，有以下特点：

- 迭代性：模型的更新并非一次完成，需要循环迭代多次。
- 容错性：即使在每个循环中产生一些错误，模型最终仍能收敛。
- 参数收敛的非均匀性：有些参数几轮迭代就会收敛，而有的参数却需要上百轮迭代。

工业界需要训练大型的机器学习模型，一些广泛应用的特定的模型在规模上有两个特点：

1. 参数很大，超过单个机器的容纳的能力（大型 LR 和神经网络）。
2. 训练数据太大，需要并行提速（大数据）。

设计一个上述系统时，我们需要解决很多问题，例如频繁访问修改模型参数需要消耗巨大带宽，如何提高并行度，减少同步等待造成的延迟等，在此类场景下，类似 MapReduce 的框架无法满足需求，参数服务器应运而生。Parameter Server 适用于大规模深度学习系统，大规模 Logistic Regression 系统，大规模主题模型，大规模矩阵分解等依赖 SGD 或 L-BFGS 最优化的算法。

参数

组件参数

组件参数与普通 TensorFlow 组件一样，框架提供了支持 py2 或 py3 的 TensorFlow 1.12 的镜像。如下示例：

- 程序脚本：可复制“示例代码”到脚本文件中然后上传到此处
- 程序参数：--data_dir ${ai_dataset_lib}/public/data/cherry/Minist/
- 版本号：TensorFlow1.12-Python3.5

资源参数

配置多机多卡的资源，worker_num 数量即机器数，ps_num 即 parameter server 数。

示例代码

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

```
import json
import math
import os
import sys
import tempfile
import time

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

flags = tf.app.flags
flags.DEFINE_string("data_dir", "",
"Directory for storing mnist data")
flags.DEFINE_boolean("download_only", False,
"Only perform downloading of data; Do not proceed to "
"session preparation, model definition or training")
flags.DEFINE_integer("task_index", None,
"Worker task index, should be >= 0. task_index=0 is "
"the master worker task the performs the variable "
"initialization ")
flags.DEFINE_integer("num_gpus", 1, "Total number of gpus for each machine."
"If you don't use GPU, please set it to '0'")
flags.DEFINE_integer("replicas_to_aggregate", None,
"Number of replicas to aggregate before parameter update"
"is applied (For sync_replicas mode only; default: "
"num_workers)")
flags.DEFINE_integer("hidden_units", 100,
"Number of units in the hidden layer of the NN")
flags.DEFINE_integer("train_steps", 10000,
"Number of (global) training steps to perform")
flags.DEFINE_integer("batch_size", 100, "Training batch size")
flags.DEFINE_float("learning_rate", 0.01, "Learning rate")
flags.DEFINE_boolean(
"sync_replicas", False,
"Use the sync_replicas (synchronized replicas) mode, "
"wherein the parameter updates from workers are aggregated "
"before applied to avoid stale gradients")
flags.DEFINE_boolean(
```

```
"existing_servers", False, "Whether servers already exists. If True, "
"will use the worker hosts via their GRPC URLs (one client process "
"per worker host). Otherwise, will create an in-process TensorFlow "
"server.")
flags.DEFINE_string("ps_hosts", "localhost:2222",
"Comma-separated list of hostname:port pairs")
flags.DEFINE_string("worker_hosts", "localhost:2223,localhost:2224",
"Comma-separated list of hostname:port pairs")
flags.DEFINE_string("job_name", None, "job name: worker or ps")

FLAGS = flags.FLAGS

IMAGE_PIXELS = 28

# Example:
# cluster = {'ps': ['host1:2222', 'host2:2222'],
# 'worker': ['host3:2222', 'host4:2222', 'host5:2222']}
# os.environ['TF_CONFIG'] = json.dumps(
# {'cluster': cluster,
# 'task': {'type': 'worker', 'index': 1}})

def main(unused_argv):
# Parse environment variable TF_CONFIG to get job_name and task_index

# If not explicitly specified in the constructor and the TF_CONFIG
# environment variable is present, load cluster_spec from TF_CONFIG.
tf_config = json.loads(os.environ.get('TF_CONFIG') or '{}')
task_config = tf_config.get('task', {})
task_type = task_config.get('type')
task_index = task_config.get('index')

FLAGS.job_name = task_type
FLAGS.task_index = task_index

mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

if FLAGS.download_only:
sys.exit(0)
```

```
if FLAGS.job_name is None or FLAGS.job_name == "":
raise ValueError("Must specify an explicit `job_name`")
if FLAGS.task_index is None or FLAGS.task_index == "":
raise ValueError("Must specify an explicit `task_index`")

print("job name = %s" % FLAGS.job_name)
print("task index = %d" % FLAGS.task_index)

cluster_config = tf_config.get('cluster', {})
ps_hosts = cluster_config.get('ps')
worker_hosts = cluster_config.get('worker')

ps_hosts_str = ','.join(ps_hosts)
worker_hosts_str = ','.join(worker_hosts)

FLAGS.ps_hosts = ps_hosts_str
FLAGS.worker_hosts = worker_hosts_str

# Construct the cluster and start the server
ps_spec = FLAGS.ps_hosts.split(",")
worker_spec = FLAGS.worker_hosts.split(",")

# Get the number of workers.
num_workers = len(worker_spec)

cluster = tf.train.ClusterSpec({"ps": ps_spec, "worker": worker_spec})

if not FLAGS.existing_servers:
# Not using existing servers. Create an in-process server.
server = tf.train.Server(
cluster, job_name=FLAGS.job_name, task_index=FLAGS.task_index)
if FLAGS.job_name == "ps":
server.join()

is_chief = (FLAGS.task_index == 0)
if FLAGS.num_gpus > 0:
# Avoid gpu allocation conflict: now allocate task_num -> #gpu
```

```
# for each worker in the corresponding machine
gpu = (FLAGS.task_index % FLAGS.num_gpus)
worker_device = "/job:worker/task:%d/gpu:%d" % (FLAGS.task_index, gpu)
elif FLAGS.num_gpus == 0:
# Just allocate the CPU to worker server
cpu = 0
worker_device = "/job:worker/task:%d/cpu:%d" % (FLAGS.task_index, cpu)
# The device setter will automatically place Variables ops on separate
# parameter servers (ps). The non-Variable ops will be placed on the workers.
# The ps use CPU and workers use corresponding GPU
with tf.device(
tf.train.replica_device_setter(
worker_device=worker_device,
ps_device="/job:ps/cpu:0",
cluster=cluster)):
global_step = tf.Variable(0, name="global_step", trainable=False)

# Variables of the hidden layer
hid_w = tf.Variable(
tf.truncated_normal(
[IMAGE_PIXELS * IMAGE_PIXELS, FLAGS.hidden_units],
stddev=1.0 / IMAGE_PIXELS),
name="hid_w")
hid_b = tf.Variable(tf.zeros([FLAGS.hidden_units]), name="hid_b")

# Variables of the softmax layer
sm_w = tf.Variable(
tf.truncated_normal(
[FLAGS.hidden_units, 10],
stddev=1.0 / math.sqrt(FLAGS.hidden_units)),
name="sm_w")
sm_b = tf.Variable(tf.zeros([10]), name="sm_b")

# Ops: located on the worker specified with FLAGS.task_index
x = tf.placeholder(tf.float32, [None, IMAGE_PIXELS * IMAGE_PIXELS])
y_ = tf.placeholder(tf.float32, [None, 10])

hid_lin = tf.nn.xw_plus_b(x, hid_w, hid_b)
```

```
hid = tf.nn.relu(hid_lin)

y = tf.nn.softmax(tf.nn.xw_plus_b(hid, sm_w, sm_b))
cross_entropy = -tf.reduce_sum(y_ * tf.log(tf.clip_by_value(y, 1e-10, 1.0)))

opt = tf.train.AdamOptimizer(FLAGS.learning_rate)

if FLAGS.sync_replicas:
if FLAGS.replicas_to_aggregate is None:
replicas_to_aggregate = num_workers
else:
replicas_to_aggregate = FLAGS.replicas_to_aggregate

opt = tf.train.SyncReplicasOptimizer(
opt,
replicas_to_aggregate=replicas_to_aggregate,
total_num_replicas=num_workers,
name="mnist_sync_replicas")

train_step = opt.minimize(cross_entropy, global_step=global_step)

if FLAGS.sync_replicas:
local_init_op = opt.local_step_init_op
if is_chief:
local_init_op = opt.chief_init_op

ready_for_local_init_op = opt.ready_for_local_init_op

# Initial token and chief queue runners required by the sync_replicas mode
chief_queue_runner = opt.get_chief_queue_runner()
sync_init_op = opt.get_init_tokens_op()

init_op = tf.global_variables_initializer()
train_dir = tempfile.mkdtemp()

if FLAGS.sync_replicas:
sv = tf.train.Supervisor(
is_chief=is_chief,
```

```
logdir=train_dir,
init_op=init_op,
local_init_op=local_init_op,
ready_for_local_init_op=ready_for_local_init_op,
recovery_wait_secs=1,
global_step=global_step)
else:
sv = tf.train.Supervisor(
is_chief=is_chief,
logdir=train_dir,
init_op=init_op,
recovery_wait_secs=1,
global_step=global_step)

sess_config = tf.ConfigProto(
allow_soft_placement=True,
log_device_placement=False,
device_filters=["/job:ps",
"/job:worker/task:%d" % FLAGS.task_index])

# The chief worker (task_index==0) session will prepare the session,
# while the remaining workers will wait for the preparation to complete.
if is_chief:
print("Worker %d: Initializing session..." % FLAGS.task_index)
else:
print("Worker %d: Waiting for session to be initialized..." %
FLAGS.task_index)

if FLAGS.existing_servers:
server_grpc_url = "grpc://" + worker_spec[FLAGS.task_index]
print("Using existing server at: %s" % server_grpc_url)

sess = sv.prepare_or_wait_for_session(server_grpc_url, config=sess_config)
else:
sess = sv.prepare_or_wait_for_session(server.target, config=sess_config)

print("Worker %d: Session initialization complete." % FLAGS.task_index)
```

```
if FLAGS.sync_replicas and is_chief:
# Chief worker will start the chief queue runner and call the init op.
sess.run(sync_init_op)
sv.start_queue_runners(sess, [chief_queue_runner])

# Perform training
time_begin = time.time()
print("Training begins @ %f" % time_begin)

local_step = 0
while True:
# Training feed
batch_xs, batch_ys = mnist.train.next_batch(FLAGS.batch_size)
train_feed = {x: batch_xs, y_: batch_ys}

_, step = sess.run([train_step, global_step], feed_dict=train_feed)
local_step += 1

now = time.time()
print("%f: Worker %d: training step %d done (global step: %d)" %
(now, FLAGS.task_index, local_step, step))

if step >= FLAGS.train_steps:
break

time_end = time.time()
print("Training ends @ %f" % time_end)
training_time = time_end - time_begin
print("Training elapsed time: %f s" % training_time)

# Validation feed
val_feed = {x: mnist.validation.images, y_: mnist.validation.labels}
val_xent = sess.run(cross_entropy, feed_dict=val_feed)
print("After %d training step(s), validation cross entropy = %g" %
(FLAGS.train_steps, val_xent))
```

```
if __name__ == "__main__":
tf.app.run()
```

Analytics Zoo

最近更新时间：2020-12-22 16:43:28

Analytics Zoo 作为一个数据分析 + AI 平台，能够帮助用户利用 Spark 的各种流水线、内置模型、特征操作等，构建基于大数据的深度学习端到端应用。 某种意义上它是 Spark 和 BigDL 的扩充，可以将 Spark、TensorFlow、Keras 和 BigDL 无缝合并到一个集成管道中，方便地扩展到企业已有的大型 Apache Hadoop/Spark 集群，进行分布式训练或推理。

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择【框架】>【机器学习】列表下的 Analytics Zoo 节点，并将其拖拽至画布中。
2. **配置参数**
   - 脚本及依赖包文件上传：
     将执行脚本上传至执行脚本框。如果需要依赖文件，则压缩为 zip 文件后通过依赖包文件框上传。
   - 算法参数：
     指定运行执行脚本的参数。
   - Python 版本：指定 Python 版本，Analytics Zoo 目前支持 Python3.5。
   - 版本号：目前支持 Analytics Zoo。
3. **配置资源**
   在【资源参数】列表框配置任务的资源参数。
   - num-executors：指定分配的计算节点数目。
   - driver-memory：指定主节点内存大小，上限为14GB。
   - executor-cores：指定每个子节点分配的 CPU Core 数，推荐2 - 3。
   - executor-memory：指定每个子节点分配的内存大小，上限为55GB，推荐单个 core 分配2 - 3GB。
   - spark-conf：指定 Spark 常用参数配置，如压缩、序列化、网络等。例如 spark.cores.max = 1000。
4. **运行**
   单击【保存】并运行工作流。
5. **查看 Spark 控制台和日志**
   在该节点上单击右键菜单，单击【Spark 控制台】，可查看任务状态和详细日志。

案例说明

Analytics Zoo 为用户提供基于 LSTM 的算法，用于时间序列数据的异常检测。它将影响当前时间的一系列值（例如最近50个小时的数据）作为模型的输入来训练模型，然后使用训练好的模型预测下一个数据点。当实际值与模型预测值相距较大时，定义为异常。我们将通过一个异常检测的案例向您介绍 Analytics Zoo 的使用方法。

数据准备

我们使用 Numenta Anomaly Benchmark 的一个数据集（NYC taxi passengers）作为案例的数据集。该数据集包含10320条样本，每条样本表示特定时间纽约市的出租车乘客总数。数据格式如下所示：

```
timestamp,value
2014-07-01 00:00:00,10844
2014-07-01 00:30:00,8127
2014-07-01 01:00:00,6210
2014-07-01 01:30:00,4656
2014-07-01 02:00:00,3820
2014-07-01 02:30:00,2873
2014-07-01 03:00:00,2369
2014-07-01 03:30:00,2064
2014-07-01 04:00:00,2221
```

在您运行案例之前，您需要先下载数据，解压压缩包，并将数据文件 nyc_taxi.csv 上传到 COS 上去。

执行脚本准备

参数定义

在脚本的开头，定义算法所需的参数。这里包括数据路径、batch_size、epoch 次数、展开长度四个参数。

```
parser.add_option("--input_dir", dest="input_dir")
parser.add_option("-b", "--batch_size", dest="batch_size", default="1024")
parser.add_option("--nb_epoch", dest="nb_epoch", default="20")
parser.add_option("--unroll_len", dest="unroll_len", default="24")
```

数据预处理

为了准备 AnomalyDetrctor 模型的输入，使用 unroll 方法对时间序列数据进行展开。

```
## 加载数据
def load_and_scale(input_path):
df = pd.read_csv(input_path)
df['datetime'] = pd.to_datetime(df['timestamp'])
df['hours'] = df['datetime'].dt.hour
df['awake'] = (((df['hours'] >= 6) & (df['hours'] <= 23)) | (df['hours'] == 0)).astype(int)
print(df.head(10))
sqlContext = SQLContext(sc)
```

```
dfspark = sqlContext.createDataFrame(df[["value", "hours", "awake"]])
feature_size = len(["value", "hours", "awake"])
return AnomalyDetector.standardScale(dfspark), feature_size

df_scaled, feature_size = load_and_scale(options.input_dir)
data_rdd = df_scaled.rdd.map(lambda row: [x for x in row])
unrolled = AnomalyDetector.unroll(data_rdd, int(options.unroll_len), predict_step=1)
[train, test] = AnomalyDetector.train_test_split(unrolled, 1000)
```

模型训练

您能够使用下面的 Python API 创建一个 AnomalyDetrctor 模型，使用 mse 作为 loss，rmsprop 作为优化器进行训练。

```
from zoo.models.anomalydetection import AnomalyDetector
model = AnomalyDetector(feature_shape=(10, 3), hidden_layers=[8, 32, 15], dropouts=[0.2, 0.2, 0.2])
model.compile(loss='mse', optimizer='rmsprop', metrics=['mae'])
model.fit(train, batch_size=int(options.batch_size), nb_epoch=int(options.nb_epoch))
```

异常检测

模型训练完成之后，您可以使用训练好的模型进行预测，检测数据点是否是异常。

```
y_predict = model.predict(test, batch_per_thread=int(options.batch_size))\
.map(lambda x: float(x[0]))
y_truth = test.map(lambda x: float(x.label.to_ndarray()[0]))
anomalies = AnomalyDetector.detect_anomalies(y_predict, y_truth, 50)
```

传递算法参数

通过算法参数传递参数给执行脚本进行执行。

```
--input_dir ${ai_dataset_lib}/analytics/nyc_taxi.csv
--batch_size 1000
--nb_epoch 2
--unroll_len 24
```

同时，“Python 版本”选择 Python3.5，“版本号”选择 analytics_zoo。

运行工作流

单击【保存】并运行工作流。

机器学习
数据预处理

最近更新时间：2021-04-07 10:25:54

按比例采样

按比例采样是一种常用的数据预处理算法。它提供了从原数据集里随机抽取特定比例的小样本数据的方法。该模块常用于抽取小样本用于数据的可视化。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

- 抽样率：范围是0 – 1.0，表示抽取样本的比例，默认值为0.5。

按样本数采样

按样本数采样是一种常用的数据预处理算法。它提供了从原数据集里随机抽取特定数量小样本数据的方法。该模块常用于抽取小样本用于数据的可视化。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：

- csv：csv 文件。
  - 输入数据包含 header 信息。
  - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

- 采样数量：默认是1000。
- 有放回采样：默认是。可选择是和否。

上采样

上采样是一种常用的处理不平衡数据的预处理方法。它是把小数据量的类别复制多份。上采样后的数据集中会反复出现一些样本，训练出来的模型可能会有一定的过拟合。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

- 标签列：指定标签所在的列，从0开始计数。
- 采样类别：需要采样的类别值（数量少的类别），如类别 0.0。
- 目标类别：数量多的类别，如类别 1.0。
- 类别比率阈值：如果(目标类别 / 采样类别）比类别比率阈值小，那么说明数据是平衡的，不做任何处理。如果(目标类别 / 采样类别）比类别比率阈值大，那么会对采样类别进行采样，采样率为 (目标类别 / 采样类别）/ 类别比率阈值。

下采样

下采样是一种常用的处理不平衡数据的预处理方法。下采样是从大众类中剔除一些样本，或者说只从大众类中选取部分样本。下采样的缺点显而易见，那就是最终的训练集丢失了数据，模型只学到了部分数据的特征。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

- 标签列：指定标签所在的列，从0开始计数。
- 采样类别：需要采样的类别值（数量多的类别），如类别 0.0。
- 目标类别：数量少的类别，如类别 1.0。
- 类别比率阈值：如果(采样类别 / 目标类别）比类别比率阈值小，那么说明数据是平衡的，不做任何处理。如果(采样类别 / 目标类别）比类别比率阈值大，那么会对采样类别进行采样，采样率为 类别比率阈值 / (采样类别 / 目标类别）。

数据切分

数据切分是另外一种常用的数据预处理算法。在机器学习建模过程中，通常需要训练数据集和验证数据集两类数据集。该方法将数据集按照一定的比例切分为训练数据集和验证数据集。

输入

- 输入数据路径：输出文件所在路径。
- 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 第一部分输出结果：第一份数据的输出，如切分比例为0.7，该份结果占总数据的0.7。
- 第二部分输出结果：第二份数据的输出，如切分比例为0.7，该份结果占总数据的0.3。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

切分比例：数据切分的比例。

去除重复行

该算法用于将数据集中的重复样本进行去重处理。

输入

- 输入数据路径：输出文件所在路径。
- 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：

- csv：csv 文件。
  - 输出数据包含 header 信息。
  - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

生成 ID 列

该算法自动生成一列 ID 列，ID 列各行的数据各不相同。生成的 ID 列会放到输出数据的最后一列。

输入

- 输入数据路径：输出文件所在路径。
  - 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据格式：格式包括以下两种：
- csv：csv 文件。
  - 输出数据包含 header 信息。
  - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

参数

生成的列名：ID 列的列名，默认是“ID”。

缺失值填充

该算法对数据中某列数据存在的缺失值进行替换。

输入

- 输入数据路径：输出文件所在路径。
  - 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据格式：格式包括以下两种：
- csv：csv 文件。
  - 输出数据包含 header 信息。
  - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

参数

- 特征列：处理的特征列序号，如0 - 1，从0开始计数。
- 填充方法：
  - zero：填充0值。
  - minimum：填充最小值。
  - maximum：填充最大值。
  - average： 填充均值。
  - median： 填充中位数。
  - value：填充某一指定的固定值（主要针对字符串特征）。

修改列名

该算法修改数据中某一列的列名。

输入

- 输入数据路径：输出文件所在路径。
  - 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据格式：格式包括以下两种：
- csv：csv 文件。
  - 输出数据包含 header 信息。
  - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

参数

- 选择列：选择修改列名的列，从0开始计数。

- 列名：修改后的列名。

自动数据预处理

该算法自动预处理数据。主要做如下自动处理：

1. 去除重复样本。
2. 删除缺失率高的列，删除列值相同的列。
3. 数据规整化（大小写转换，去除两侧空格）。
4. 填充缺失值。
5. 处理异常值。

输入

- 输入数据路径：输出文件所在路径。
  - 输入数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据格式：格式包括以下两种：
- csv：csv 文件。
  - 输出数据包含 header 信息。
  - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

参数

- 缺失值阈值：如果特征列的缺失值比例大于该阈值，特征列会被删除。
- 是否删除异常值：决定异常值的处理方式，选择是，则删除异常值所在的行，否则用合适的值填充。

数据类型转换

该算法提供数据类型转换功能。

输入

- 输入数据路径：输出文件所在路径。
  - 输入数据格式：格式包括以下两种：
  - csv：csv 文件。

        - 输入数据包含 header 信息。
        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

输出

- 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
        - 输出数据包含 header 信息。
        - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

参数

- 选择列：选择修改列名的列，从0开始计数。
- 目标数据类型：选择目标数据类型。

特征转换

最近更新时间：2020-07-24 16:42:48

二值化

二值化是一个将数值特征转换为二值特征的处理过程。阈值参数表示决定二值化的阈值。 值大于阈值的特征二值化为1，否则二值化为0。

输入

- 输入数据路径：输入文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

输出

- 输出数据路径：经转换后的数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 选中的原始特征列会被删除，经过二值化的特征会 append 到数据的最后几列。

参数

- 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 阈值：默认为0.5。值大于阈值的特征二值化为1，否则二值化为0。

正则化

正则化器缩放单个样本让其拥有单位 p 范数。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：经转换后的测试数据存储路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- P：范数。

标准归一化

在原始的数据中，各变量的范围大不相同。对于某些机器学习的算法，若没有做过标准化，目标函数会无法适当的运作。举例来说，多数的分类器利用两点间的距离计算两点的差异， 若其中一个特征具有非常广的范围，那两点间的差异就会被该特征左右，因此，有些特征应该被标准化，可以使各特征按比例影响两点间的距离。另外一个做特征缩放的理由是他能使加速梯度下降法的收敛。标准归一化会使每个特征中的数值平均变为0（将每个特征的值都减掉原始数据中该特征的均值）、标准差变为1。

训练节点

- 输入
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过标准归一化的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
  - 特征是否减去均值：默认选中。
  - 特征是否除以方差：默认不选中。

预测节点

- **输入**
  - 输入数据路径：输入测试文件所在路径。
    - 输入文件类型：格式包括以下两种：
      - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
      - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的测试数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过标准归一化的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

最大绝对值归一化

最大绝对值归一化将每个特征调整到 [-1, 1] 的范围,它通过每个特征内的最大绝对值来划分。 它不会移动和聚集数据，因此不会破坏任何的稀疏性。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> **注意：**
> 最后的结果中，选中的原始特征列会被删除，经过最大绝对值归一化的特征会append到数据的最后几列。

- **参数**
  选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

预测节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。

- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过最大绝对值归一化的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

最小最大归一化

最小最大归一化将每个特征调整到一个特定的范围（通常是（0,1））。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件

        - 输出数据包含 header 信息
        - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过最小最大归一化的特征会 append 到数据的最后几列。

- **参数**
    - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
    - min：默认是0。转换的下界,被所有的特征共享。
    - max：默认是1。转换的上界,被所有特征共享。

预测节点

- **输入**
    - 输入数据路径：输入训练文件所在路径。
    - 输入文件类型：格式包括以下两种：
        - csv：csv 文件
            - 输入数据包含 header 信息
            - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
        - parquet：列式存储格式 parquet
- **输出**
    - 输出数据路径：经转换后的训练数据存储路径。
    - 输出数据格式：格式包括以下两种：
        - csv：csv 文件
            - 输出数据包含 header 信息
            - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
        - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过最小最大归一化的特征会 append 到数据的最后几列。

- **参数**
    - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

主成分分析

主成分分析一种统计学的特征降维方法，将数据从原来的坐标系投影到新的坐标系，通过每个维度的方差大小来衡量该维度的重要性。从中选取重要性排在前K个的特征作为新的特征，达到数据降维的目的。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
      >!最后的结果中，选中的原始特征列会被删除，经过 PCA 的特征会 append 到数据的最后几列。
  - **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
    - k：降维后的特征维度。

预测节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件

- 输出数据包含 header 信息
- 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
- parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过 PCA 的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

向量索引

向量索引把数据集中的类型特征转换为索引。它不仅可以自动的判断哪些特征可以类别化，也能将原有的值转换为类别索引。通过 maxCategories 参数来判断特征是否可以类别化。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

    > ⚠ 注意：
    > 最后的结果中，选中的原始特征列会被删除，经过向量索引的特征会 append 到数据的最后几列。

  - **参数**

- 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- maxCategories：拥有的不同值的数量小于等于 maxCategories 的特征被判断可以类别化。

预测节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过向量索引的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

多项式展开

多项式展开是一个将特征展开到多元空间的处理过程。 它通过 degree（阶）结合原始的维度来定义。例如设置 degree 为2就可以将 (x，y) 转化为 (x，x x，y，x y，y y)。

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息

        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
    - 输出数据路径：经转换后的训练数据存储路径。
    - 输出数据格式：格式包括以下两种：
        - csv：csv 文件
            - 输出数据包含 header 信息
            - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
        - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过多项式展开的特征会 append 到数据的最后几列。

- **参数**
    - 选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
    - 阶（degree）：展开阶数。

独热编码（one-hot）

将离散型特征的每一种取值都看成一种状态，若您的这一特征中有 N 个不相同的取值，那么我们就可以将该特征抽象成 N 种不同的状态，独热编码保证了每一个取值只会使得一种状态处于“激活态”，也就是说这 N 种状态中只有一个状态位值为1，其他状态位都是0。

- **输入**
    - 输入数据路径：输入训练文件所在路径。
    - 输入文件类型：格式包括以下两种：
        - csv：csv 文件
            - 输入数据包含 header 信息
            - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
        - parquet：列式存储格式 parquet
- **输出**
    - 输出数据路径：经转换后的训练数据存储路径。
    - 输出数据格式：格式包括以下两种：
        - csv：csv 文件
            - 输出数据包含 header 信息
            - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
        - parquet：列式存储格式 parquet

- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数。

字符串索引

字符串索引把数据集中的字符串特征转换为索引。字符串索引很多情况下会和独热编码一起使用。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
>
> 最后的结果中，选中的原始特征列会被删除，经过字符串索引的特征会 append 到数据的最后几列。

- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数。
  - 处理非法值的方法：可以选择保持、忽略、报错三种处理方法。
  - 排序方式：对字符串进行索引的顺序，可以选择按频率倒序、按频率正序、按字母倒序、按字母正序四种方法。

预测节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件

        - 输入数据包含 header 信息
        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

> ⚠ 注意：
>
> 最后的结果中，选中的原始特征列会被删除，经过字符串索引的特征会 append 到数据的最后几列。
>
> - **参数**
>   - 选择特征列：表示需要计算的特征所在列，从0开始计数。

索引转字符串

将字符串转换为索引之后，我们很难分辨索引到底代表的是哪个类别，这时候我们可以用序列转字符串算子将序列再转换为原始的字符串。

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数。

离散余弦变换

离散余弦变换将一个在时间域（time domain）内长度为 N 的实值序列转换为另外一个在频率域（frequency domain）内的长度为 N 的实值序列。

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数。
  - 是否反向变换：默认反向变换。

Box-Cox 变换

统计分析中，基础数据的分布可能比较特别，不符合“正态分布”。正态分布的变换，比较经典的就是 box-cox 变换。
box-cox 变换的目标有两个：一个是变换后，可以一定程度上减小不可观测的误差和预测变量的相关性。主要操作是对因变量转换，使得变换后的因变量于回归自变量具有线性相依关系，误差也服从正态分布，误差各分量是等方差且相互独立。第二个是用这个变换来使得因变量获得一些性质，例如在时间序列分析中的平稳性，或者使得因变量分布为正态分布。

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**

- 输出数据路径：经转换后的训练数据存储路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数。
  - lambda：变换参数，当lambda为0时，等价于对数变换。

特征分桶

特征分桶将连续的特征列转换成离散的列。这些离散值由用户指定，通过“切分区间”参数来确定。

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数，如“1 - 4”。
  - 切分区间：如果有 n+ 切分区间,那么将有n个桶。桶将由区间 x和区间 y共同确定,它的值范围为[x，y]，如果是最后一个桶,范围将是 [x，y]。切分区间应该严格递增。例如“0.0，1.0，2.0”。

分位数离散化

分位数离散化输入连续的特征列,输出离散的特征。分桶数是通过参数“桶数量”来指定的。 桶的范围是通过使用近似算法（见approxQuantile）来得到的。桶的上边界和下边界分别是正无穷和负无穷时, 取值将会覆盖所有的实数值。

训练节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

      最后的结果中，选中的原始特征列会被删除，经过字符串索引的特征会 append 到数据的最后几列。
- **参数**
  - 选择特征列：表示需要计算的特征所在列，从0开始计数，如“1-4”。
  - 桶数量：分桶的个数。

预测节点

- **输入**
  - 输入数据路径：输入训练文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- **输出**
  - 输出数据路径：经转换后的训练数据存储路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件
      - 输出数据包含 header 信息
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

      最后的结果中，选中的原始特征列会被删除，经过字符串索引的特征会 append 到数据的最后几列。
- **参数**

- 选择特征列：表示需要计算的特征所在列，从0开始计数，如“1-4”。

特征选择

最近更新时间：2020-06-29 15:33:33

卡方特征选择

卡方检验是一种常用的特征选择方法。卡方用来描述两个事件的独立性或者描述实际观察值与期望值的偏离程度。卡方值越大，则表明实际观察值与期望值偏离越大，也说明两个事件的相互独立性越弱。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过卡方选择的特征会 append 到数据的最后几列。

参数

- 标签列：标签列所在的列号，从0开始计数，如填写0，表示第一列是标签。
- 特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 选择的特征个数：根据卡方值选择的特征个数。

基于方差的特征选择

如果一个特征不发散，例如方差接近于0，也就是说样本在这个特征上基本上没有差异，这个特征对于样本的区分并没有什么用。所以通过对低方差的特征进行过滤，是特征选择常用的方法。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

> ⚠ 注意：
> 最后的结果中，选中的原始特征列会被删除，经过方差选择的特征会 append 到数据的最后几列。

参数

- 标签列：标签列所在的列号，从0开始计数，如填写0，表示第一列是标签。
- 特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 方差阈值：根据方差阈值选择特征。

基于树的特征选择

基于树的集成算法有一个很好的特性，就是模型训练结束后可以输出模型所使用的特征的相对重要度，便于我们选择特征，理解哪些因素是对预测有关键影响。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 标签列：标签列所在的列号，从0开始计数，如填写0，表示第一列是标签。
- 特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 选择的特征个数：根据树模型选择的特征个数。

基于信息的特征选择

基于信息的特征选择是常用的特征选择方法。总共有四种信息值用于特征选择：极小冗余极大相关、互信息最大化、互信息等。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

> ⚠ 注意：
>
> 最后的结果中，选中的原始特征列会被删除，经过树选择的特征会 append 到数据的最后几列。

参数

- 标签列：标签列所在的列号，从0开始计数，如填写0，表示第一列是标签。
- 特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 选择的特征个数：根据信息值选择的特征个数。
- 特征选择方法：可选择的项有极小冗余极大相关、互信息最大化、互信息、联合互信息、交互覆盖、条件互信息最大化、信息碎片。

特征权重分析

学习出特征在模型中所占比重，分析特征重要性。

输入

- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

算法参数

- 任务类别：分类、回归
- 分类方法：
  分类：决策树分类、随机森林分类、梯度提升树分类
  回归：决策树回归、随机森林回归、梯度提升树回归

输出

- 输出参数：colname、weight
- 特征权重分析图：按重要性从高到低排列展示前十个特征，其余特征占比放在第一列“其他”列展示
- 图中默认展示前1000行数据，更多数据请单击【下载】查看。

异常检测

最近更新时间：2020-07-31 09:31:48

孤立森林

基于孤立森林的异常点检测算法，该算法首先构建 n 棵树，每棵树都从原始数据中有放回的采样 m 个样本进行训练，每棵树在训练的时候都完全采用了随机选择特征以及特征分裂的方式，然后再将每棵树的训练结果进行汇总就可以得到每个样本成为异常点的概率（0到1之间的浮点值），该值越大越有可能是异常点。具体算法过程请参考论文 Isolation-based Anomaly Detection。

训练节点

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

- 选择特征列：表示需要计算的特征所在列，例如“1 - 12，15”，表示取特征在表中的1到12列，15列，从0开始计数。
- 树棵树：构建森林需要的树个数，默认100。
- 特征数：用于训练树的随机特征数。
- 样本数：用于训练树的随机样本数。
- 异常点比例：数据集中异常点所占的比例，默认为0.1。
- 树的最大深度：默认为10。
- 是否有放回采样：默认为 false。

预测节点

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

选择特征列：表示需要计算的特征所在列，例如“1-12,15”，表示取特征在表中的1到12列，15列，从0开始计数。

Z-score 异常值检测

Z-score 是一维或低维特征空间中的参数异常检测方法。该技术假定数据是高斯分布，异常值是分布尾部的数据点，因此远离数据的平均值。距离的远近取决于使用公式计算的归一化数据点 z_i 的设定阈值 Z_t：Z_i = (x_i - mu) / std，其中 mu 是均值， std 是标准差。然后经过标准化处理后，异常值也进行了标准化处理，其绝对值大于 Z_t 。本算法中 Z_t 取3。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数

选择特征列：检测的特征列，从0开始计数。

特征提取

最近更新时间：2020-06-29 15:30:01

Word2Vec on SONA

算法简介

Word2Vec 将文本中的词语映射到 k 维向量空间中，同时向量空间上的相似度可以用来表示词语语义上的相似度。

参数说明

- **输入数据格式**
  文档，每行一个句子，如下所示：

```
word1 word2 word3 ...
word4 word5 word6 ...
... ...
... ...
```

- **训练节点**
  - 算法 IO 参数
    - 训练数据：训练数据所在的 COS 路径。
    - 模型输出：训练结果保存 COS 路径。
  - 算法参数
    - Embedding 特征的维度：embedding 向量的维度。
    - 学习率：初始学习率。
    - BatchSize：每个 mini batch 数据大小。
    - 最大 epoch 数：最大迭代轮数。
    - windowSize：滑动窗口大小。
    - 是否对数据进行采样操作：是否对训练数据进行采样。
    - 是否进行 ID 重映射：是否进行 ID 重映射。
    - cbow 或者 skipgram：cbow 方式或者 skipgram 方式。
    - 模型分区个数：参数服务器模型的分区数量。
    - 负采样数：负采样节点个数。
    - checkpoint 轮数：每隔多少轮做一次 checkpoint。

- spark conf 参数
  - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
  - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要设置为 true。
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - executor-memory(g)：每个 executor 需要的内存，单位为 g。
  - executor-cores：每个 executor 需要的 CPU 核数。
  - driver-memory(g)：spark driver 需要的内存，单位为 g。
  - spark.ps.instances：angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 ps 个数越多。
  - spark.ps.cores：每个 angel ps 需要的核数。
  - spark.ps.memory(g)：每个 angel ps 需要的内存，单位为 g。

本算子主要解决了词表示学习的问题

将高维稀疏的词表示嵌入到低维的向量空间。

- 原始数据如下图所示：

```
anarchism originated as a term of abuse first used against early working class radicals including the diggers of the englis
h revolution and the sans culottes of the french revolution whilst the term is still used in a pejorative way to describe a
ny act that used violent means to destroy the organization of society it has also been
taken up as a positive label by self defined anarchists the word anarchism is derived from the greek without archons ruler
chief king anarchism as a political philosophy is the belief that rulers are unnecessary and should be abolished although t
here are differing interpretations of what this means anarchism also refers to
related social movements that advocate the elimination of authoritarian institutions particularly the state the word anarch
y as most anarchists use it does not imply chaos nihilism or anomie but rather a harmonious anti authoritarian society in p
lace of what are regarded as authoritarian political structures and coercive economic institutions anarchists advocate soci
al relations based upon
voluntary association of autonomous individuals mutual aid and self governance while anarchism is most easily defined
by what it is against anarchists also offer positive visions of what they believe to be a truly free society however ideas
about how an anarchist society might work vary considerably especially with respect to economics there is also disagreement
 about how a free society might be brought about origins and predecessors kropotkin and others argue that before recorded
history human society was organized on anarchist principles most anthropologists follow kropotkin and engels in believing t
hat hunter gatherer bands were egalitarian and lacked division of labour accumulated wealth or decreed law
and had equal access to resources william godwin anarchists including the the anarchy organisation and rothbard find anarch
ist attitudes in taoism from ancient china kropotkin found similar ideas in stoic zeno of citium according to kropotkin zen
o repudiated the omnipotence of the state its intervention and regimentation and proclaimed the sovereignty of the moral la
w of the individual the anabaptists of one six th century europe are sometimes considered to be religious forerunners of mo
dern anarchism bertrand russell in his history of western philosophy writes that the anabaptists repudiated all law since t
hey held that the good man
```

- 运行算子后得到的结果有词向量文件和词映射文件。

- 词映射文件格式为：映射后的 ID：原单词。

```
0:fokine
1:mattered
2:intimately
3:reunion
4:harmonies
5:bone
6:glorifying
7:pentoxide
8:arkham
9:albescens
10:transfusion
11:peleus
12:racecars
13:onyx
14:industrialisation
15:mysteroid
16:been
17:fuller
18:modifies
19:ipa
20:kwong
21:pig
22:jade
23:crying
24:breath
25:battering
26:semites
27:accomplished
28:swain
29:ukrainian
```

- 词向量文件格式为：映射后的 ID：词向量。

```
0:5.890932E-4 0.0039711213 9.18858E-5 4.8313546E-4 -0.0035535127 0.0033942712 -8.548046E-5
0.001094785 -0.0010664434 -0.0020063713 -0.0018684975 -0.0034639365 -0.0022227308
-6.5268646E-4 0.0021918614 0.0037843545 6.349852E-4 -0.0037996734 -0.0032630798
9.685887E-5 4.3079612E-4 -0.0029240637 -0.0028981306 1.9581897E-4 0.0015903218
-4.669021E-4 -0.0024416482 0.0023292138 -0.0038829634 -0.003250387 -0.003386422
0.0029003124 0.002824246 -0.0023789285 -0.0035327068 9.579024E-5 0.0018456406 6.4591283E-4
-9.6570153E-4 5.8667467E-4 1.7671482E-4 0.0018733151 -0.0018923845 4.2529823E-4
0.0030963414 0.0033465486 0.0036233077 -0.0025648093 -0.002446964 -0.0036055946
0.0022110161 2.966985E-4 7.5042853E-4 -0.0032370735 -0.0027218119 -6.127785E-4
-0.0018945014 -0.002547899 6.5984833E-4 -0.001254839 0.0022504167 0.002252328 -0.002790278
0.0028389143 0.0022638151 0.0033603131 0.0021212026 3.6142385E-4 6.203796E-4 5.541262E-4
0.002161485 0.0028538373 -0.0017391599 0.0024538261 0.0031673256 4.3207666E-4
-4.4809043E-4 -0.0013000127 0.0035357084 -0.0019916636 -0.0019426235 0.0020503998
0.0011139599 -0.0011370436 -5.5532827E-4 -0.0017496068 -2.1123204E-4 0.0017121066
3.017349E-4 0.0020119825 -5.462527E-4 0.0011839543 0.0039800024 0.0025195205 0.0016699361
-2.6240083E-4 5.595205E-4 0.0024836515 0.0030871194 -0.0017066303 -0.0031065384
-1.8145161E-5 -6.145583E-4 5.410473E-4 0.0018261466 -5.059019E-4 -0.003784447 0.0011087982
0.0024840261 6.257725E-4 0.0028605137 0.0010251418 0.0031615957 -0.0012451047
-0.0023883507 4.8524578E-4 -0.0035458368 8.091281E-5 -0.003228977 2.6641597E-4
-0.001941703 0.0014182758 -2.5636618E-4 5.154852E-4 -9.58437E-4 -7.410791E-4 0.0010008282
-0.0014225176
1:-0.004024367 0.0038609765 -0.0024515798 -0.0013728965 0.0018704536 0.0031438328
0.0024150535 -0.00315246 -0.0016100466 0.0014629547 0.0016889522 -0.0027478538
-7.642371E-4 -9.747202E-4 -0.002067412 2.652061E-6 -0.0036517768 0.0023558848 6.51561E-4
0.00250819 0.0037633702 -0.0022672703 -8.236773E-4 0.0021630542 0.0026306782 -0.0025659492
0.002010112 0.0019294362 -0.0010794078 8.0198277E-4 0.0027767317 -0.0019157645
-0.0022618228 -5.0847436E-4 -6.764415E-4 7.448302E-4 -5.4385833E-4 0.0022963793
-3.5616924E-4 0.0034990378 -0.0014823454 -0.0011686712 -3.6209723E-4 0.0027484274
-0.0020010716 0.0032461027 -0.0020084667 -0.0032737674 -0.001871959 -0.003069962
8.655588E-6 -0.0024964819 0.0032511626 -0.0023781932 0.001520502 -9.5813023E-4 0.002325141
```

[2.0] FeatureCombination

本算子提供基于原有特征列组合、过滤、筛选产生新特征的功能。

参数说明

IO 参数

与普通的机器学习算子类似，需要填写数据路径、文件类型、有无标题栏、数据分隔符，以及特征列。这里的特征列是指所有参与特征组合的列。

算法参数

算法参数有三个基本项，分别是产生新特征的迭代次数、每轮迭代最大增长的特征数以及特征选择规则。特征选择规则可以有方差和树模型。由于树模型是基于监督学习的，所以选择树模型时，需要额外填写标签列和当前的任务类型（分类还是回归）

特征组合结果查看

特征组合算子有两个输出桩，第一个桩是正常的数据输出，即特征组合之后的数据集，可以正常连接后续的机器学习流程算子。第二个输出桩是特征组合结果，不需要额外连接任何算子，只需要鼠标右键单击，查看特征组合结果即可。
例如：新特征 feature0_0，它是由就特征 f_0 和 f_0 自交叉组合而成的，feature0_1 是由 f_0 和 f_1 交叉而出的。特征组合结果的预览支持1000行，全量数据支持下载。

对后续算子的影响

例如原始数据0 – 29列是特征，30列是标签。假如全选特征作为备选，迭代3轮，每轮最多产生10个新特征，则最后会产生30个新特征。如果下面接一个分类器，则它的特征列应该填写 "0-29, 31-60"

分类算法

最近更新时间：2020-07-24 16:43:01

逻辑回归

LogisticRegression（LR）算法是一种常见的分类算法，因其模型简单、可解释性强等特点在工程领域得到广泛应用，该算法支持二分类和多分类。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
- 特征列：作为训练特征的列，从0开始编号。对于库表可直接勾选；对于普通路径，可填形式如 a - b、c 或它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。
- 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - elasticNetParam：弹性网混合参数，0表示L2正则，1表示L1正则，0 - 1之间的值表示两者的混合。
  - family：算法族，可选 auto、binomial 和 multinomial，分别表示自动推断、二分类和多分类。
  - fitIntercept：是否拟合截距。
  - maxIter：最大迭代次数。
  - regParam：正则化系数。
  - standardization：训练前是否对特征标准化（特征值除以标准差）。
  - tol：误差最大容忍界，低于这个界的时候算法停止迭代。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持二分类任务评估跟多分类任务评估。
  - 评估指标：算法的评估指标，选择二分类任务评估时，支持 areaUnderRoc 和 areaUnderPR；选择 多分类任务评估时，支持 f1、weightedPrecision、weightedRecall、accuracy。
  - elasticNetParam：连续范围参数，范围不超过0 - 1。
  - regParam：连续范围参数，下界大于等于0，上界没有限制（太大会导致所有样本划归到“0”类）。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征)。
    - 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

朴素贝叶斯分类

NaiveBayes（朴素贝叶斯）是一种常用的多类分类算法，该算法假设各个特征之间是相互独立的，通过贝叶斯公式计算出某个样本属于某个类别的概率。朴素贝叶斯算法有 multinomial naive Bayes 和 Bernoulli naive Bayes 两种。该算法常用于文本分类，每个特征表示词在一篇文档的出现的次数（multinomial naive Bayes）或者是否出现（Bernoulli naive Bayes，用0，1表示的特征）。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
  - 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。

- 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出
  - ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - modelType：模型类型，支持 multinomial 跟 bernoulli。
  - smoothing：光滑参数。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持 BinaryClassificationEvaluator 和 MulticlassClassificationEvaluator，分别对应二分类评估跟多分类评估。
  - 评估指标：算法的评估指标，选择 BinaryClassificationEvaluator 时，支持 areaUnderRoc 和 areaUnderPR；选择 MulticlassClassificationEvaluator 时，支持 f1、weightedPrecision、weightedRecall、accuracy。
  - smoothing：连续范围参数。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

线性支持向量机分类

LinearSVC 是线性 SVM 分类器，只支持 L2 正则和二分类。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
  - 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为double型，且从0开始连续编号。
  - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - fitIntercept：是否拟合截距。
  - maxIter：最大迭代次数。
  - regParam：正则化系数。
  - standardization：训练前是否对特征标准化（特征值除以标准差）。
  - tol：误差最大容忍界，低于这个界的时候算法停止迭代。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持二分类任务评估跟多分类任务评估。
  - 评估指标：算法的评估指标，选择二分类任务评估时，支持 areaUnderRoc 和 areaUnderPR；选择多分类任务评估时，支持 f1、weightedPrecision、weightedRecall、accuracy。
  - regParam：连续范围参数，下界大于等于0，上界没有限制（太大会导致所有样本划归到“0”类）。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。

- 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

决策树分类

DecisionTreeClassifier（决策树算法）是机器学习中常用的分类/回归算法。决策树算法具有解释性好、可以处理类别特征、支持多分类、不需要做特征 scaling、可以表示非线性模型等优点。平台上的决策树分类算法支持连续、非连续特征的多分类任务，最高可以支持百万级别的样本。

训练节点

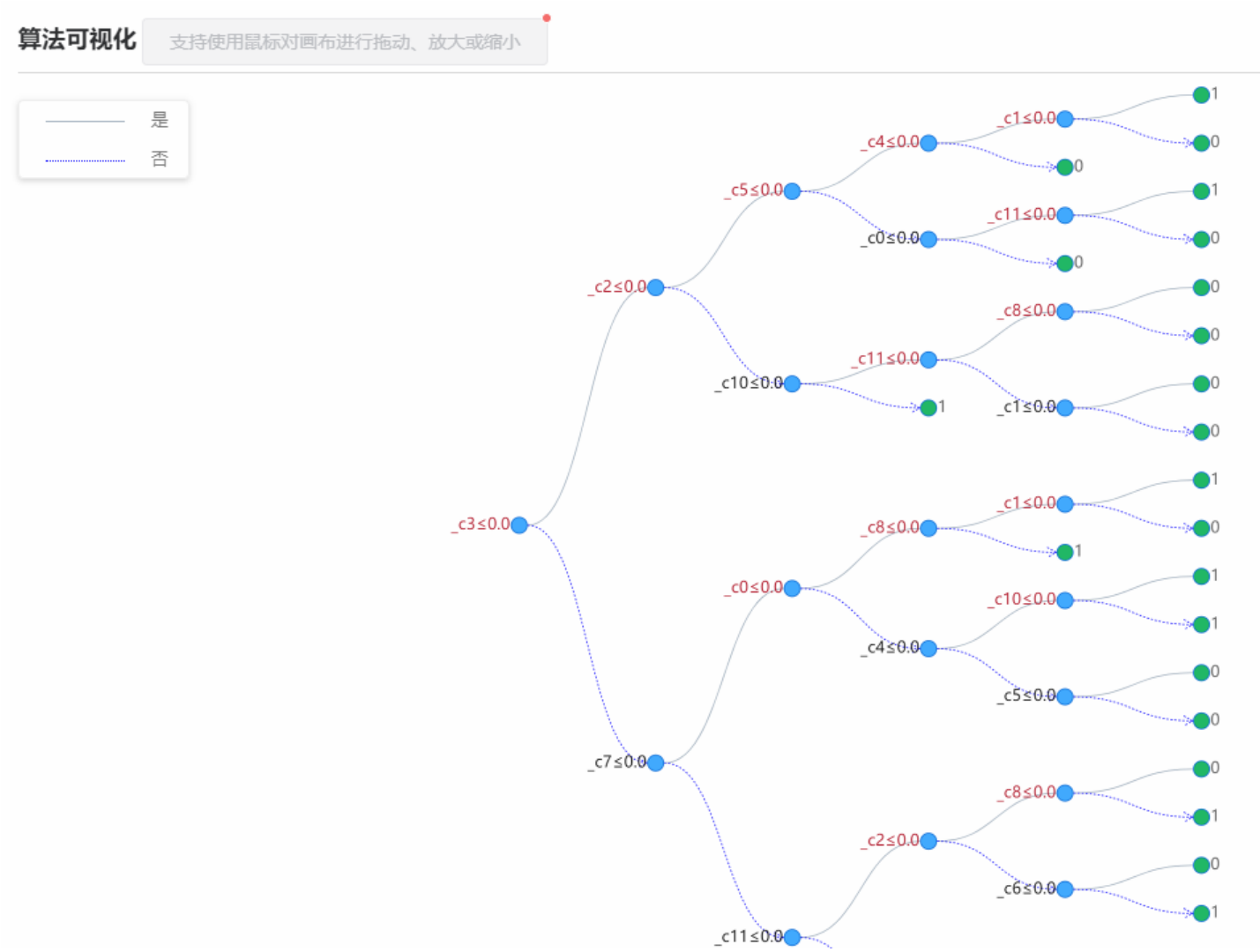
- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
- 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
  - 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。
  - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - maxDepth：决策树最大深度。
  - maxBins：决策树最大分支数。
  - minInfoGain：决策树分裂最小信息增益。
  - impurity：不纯度指标，支持 gini 指数跟 entropy。
  - minInstancesPerNode：决策树节点最小样本数。
  - checkpointInterval：每多少轮设置 checkpoint 一次，在迭代轮数非常多的时候，可以降低因为计算节点失败导致的级联重算风险。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。

- 评估方法：算法的评估方法，支持 BinaryClassificationEvaluator 和 MulticlassClassificationEvaluator，分别对应二分类评估跟多分类评估。
- 评估指标：算法的评估指标，选择 BinaryClassificationEvaluator 时，支持 areaUnderRoc 和 areaUnderPR；选择 MulticlassClassificationEvaluator 时支持 f1、weightedPrecision、weightedRecall、accuracy。
- minInfoGain：连续范围参数，下界大于等于0，上界小于1，一般不超过0.1。
- maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
- maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
- minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
- 树模型可视化

  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判断节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



> **注意：**
>
> 在训练时选择 PMML 格式才能可视化（ML 格式不行）

随机森林分类

RandomForestClassifier 是随机森林分类器，支持二分类和多分类、支持离散和连续特征。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

- 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。
- 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。

- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
  - 算法参数
    - maxDepth：决策树最大深度。
    - maxBins：决策树最大分支数。
    - minInfoGain：决策树分裂最小信息增益。
    - impurity：不纯度指标，支持 gini 指数和 entropy。
    - minInstancesPerNode：决策树节点最小样本数。
    - checkpointInterval：每多少轮设置 checkpoint 一次，在迭代轮数非常多的时候，可以降低因为计算节点失败导致的级联重算风险。
    - numTrees：最大迭代次数。
    - featureSubsetStrategy：特征采样比例策略，支持 auto、all、onethird、sqrt 和 log2，分别表示自动、全部、两分一、特征数的开方和特征数的对数。其中自动策略为：如果 numTrees 为1，设置为 all；如果 numTrees 大于1，设置为 onethird。
    - subsamplingRate：样本数采样比例。
  - 半自动调参
    - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
    - 评估方法：算法的评估方法，支持 BinaryClassificationEvaluator 和 MulticlassClassificationEvaluator，分别对应二分类评估跟多分类评估。
    - 评估指标：算法的评估指标，选择 BinaryClassificationEvaluator 时，支持 areaUnderRoc 和 areaUnderPR；选择 MulticlassClassificationEvaluator 时，支持 f1、weightedPrecision、weightedRecall、accuracy。
    - minInfoGain：连续范围参数，下界大于等于0，上界小于1，一般不超过0.1。
    - maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
    - maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
    - minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。
    - numTrees：离散正整数参数，比较合理的方式是根据默认值20在周围调节。

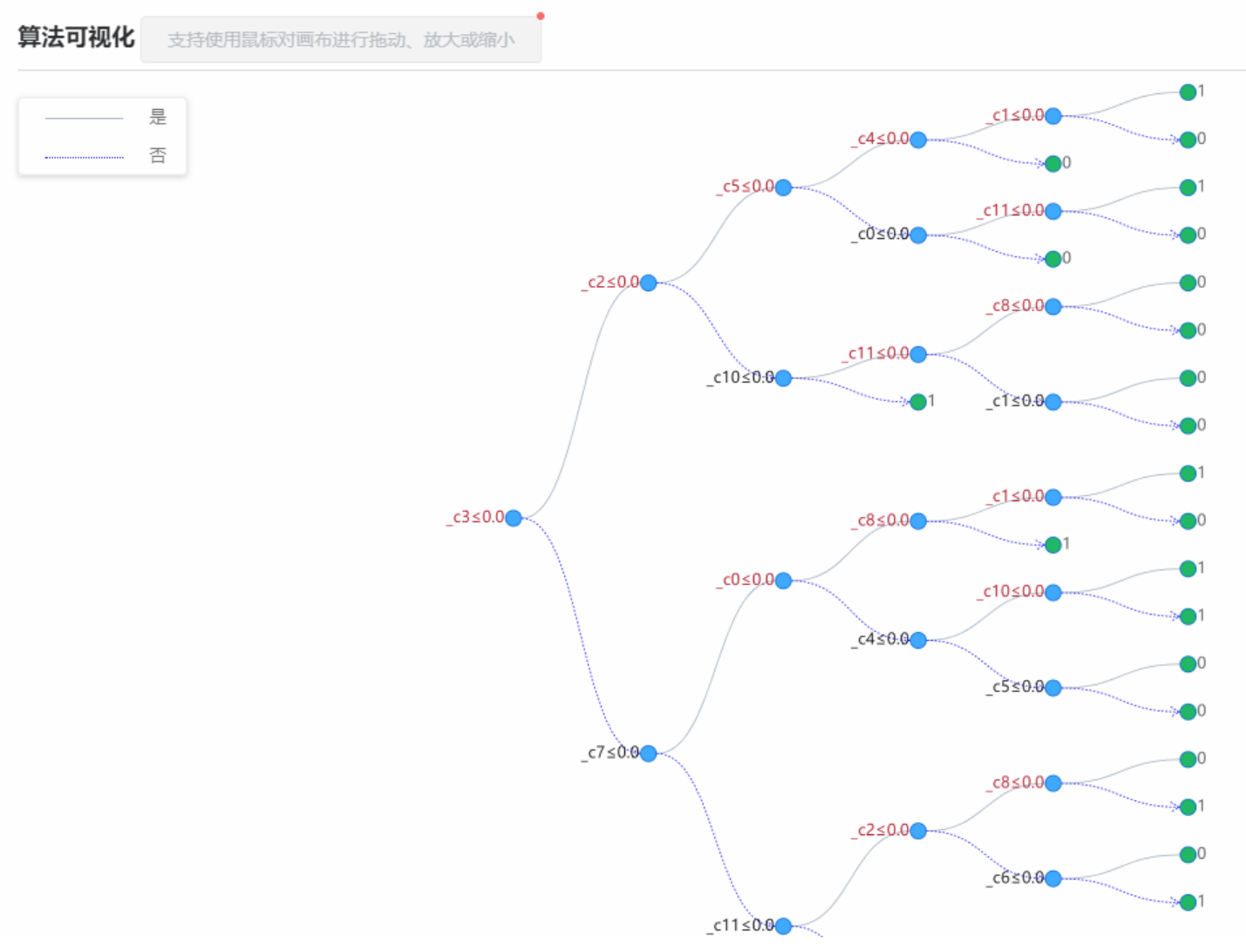
预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。

- 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。
- 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。

- 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
- 树模型可视化

  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判断节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



> ⚠ 注意：
> 在训练时选择 PMML 格式才能可视化（ML 格式不行）

梯度提升树分类

GradientBoostedTrees（梯度提升树）是机器学习中常用的分类算法，这里的实现根据论文 J.H. Friedman. "Stochastic Gradient Boosting." 1999。

训练节点

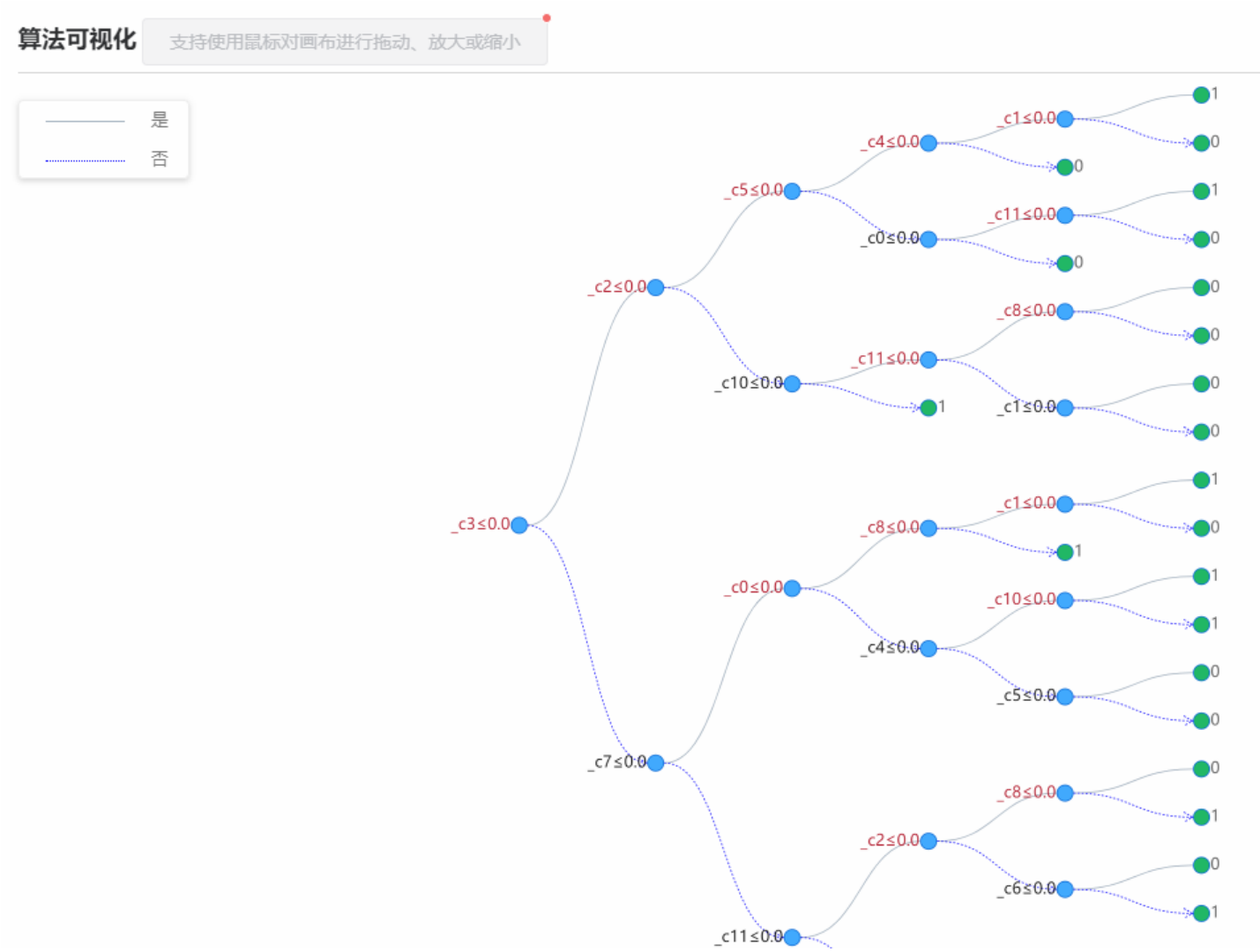
- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
  - 算法参数
    - featureSubsetStrategy：特征采样比例策略，支持 auto、all、onethird、sqrt 和 log2，分别表示自动、全部、三分一、特征数的开方和特征数的对数。其中自动策略为：如果 numTrees 为1，设置为 all；如果 numTrees 大于1，设置为 onethird
    - impurity：不纯度指标，支持 gini 指数和 entropy。
    - maxBins：决策树最大分支数。
    - maxDepth：决策树最大深度。
    - maxIter：最大迭代次数。
    - minInfoGain：决策树分裂最小信息增益。
    - minInstancesPerNode：决策树节点最小样本数。
    - stepSize：步长，范围为(0, 1]。
    - subsamplingRate：样本数采样比例。
  - 半自动调参
    - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
    - 评估方法：算法的评估方法，支持 BinaryClassificationEvaluator 和 MulticlassClassificationEvaluator，分别对应二分类评估跟多分类评估。

- 评估指标：算法的评估指标，选择 BinaryClassificationEvaluator 时，支持 areaUnderRoc 和 areaUnderPR；选择 MulticlassClassificationEvaluator 时，支持 f1、weightedPrecision、weightedRecall、accuracy。
- minInfoGain：连续范围参数，下界大于等于0，上界小于1，一般不超过0.1。
- stepSize：连续范围参数，下界大于0，上界小于等于1
- subsamplingRate：连续范围参数，下界大于0，上界小于等于1。
- maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
- maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
- minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
- 树模型可视化

  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判断节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



> **注意：**
> 在训练时选择 PMML 格式才能可视化（ML 格式不行）

K近邻分类

通过测量不同特征值之间的距离进行分类。它的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别，其中K通常是不大于20的整数。
KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。具体的说明请看 分布式 KNN。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。

        - 输入数据包含 header 信息。
        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，从0开始编号。要求特征列的特征标签为 double 类型，且从0开始连续编号。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
    - 算法参数
        - k：k近邻
        - topTreeSize：顶层树的样本点个数。
        - topTreeLeafSize：顶层树叶子节点包含样本点个数。
        - subTreeLeafSize：子数叶子节点包含样本点个数。

预测节点

- 输入
    - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
    - 输入文件类型：格式包括以下两种：
        - csv：csv 文件。
            - 输入数据包含 header 信息。
            - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 输出
    - 结果路径：路径。
    - 输出文件类型：格式包括以下两种：
        - csv：csv 文件。
            - 输出数据包含 header 信息。
            - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
        - parquet：列式存储格式 parquet。

Angel 算法相关

Angel 算法相关内容请参考以下文档：

- [Angel]LR on SONA

- [Angel]FM on SONA
- [Angel]GBDT on SONA
- [Angel]FTRL-LR on SONA
- [Angel]FTRL-FM on SONA

回归算法

最近更新时间：2020-07-24 16:43:06

随机森林回归

随机森林回归算法是常见的回归算法，支持离散和连续特征。

训练节点

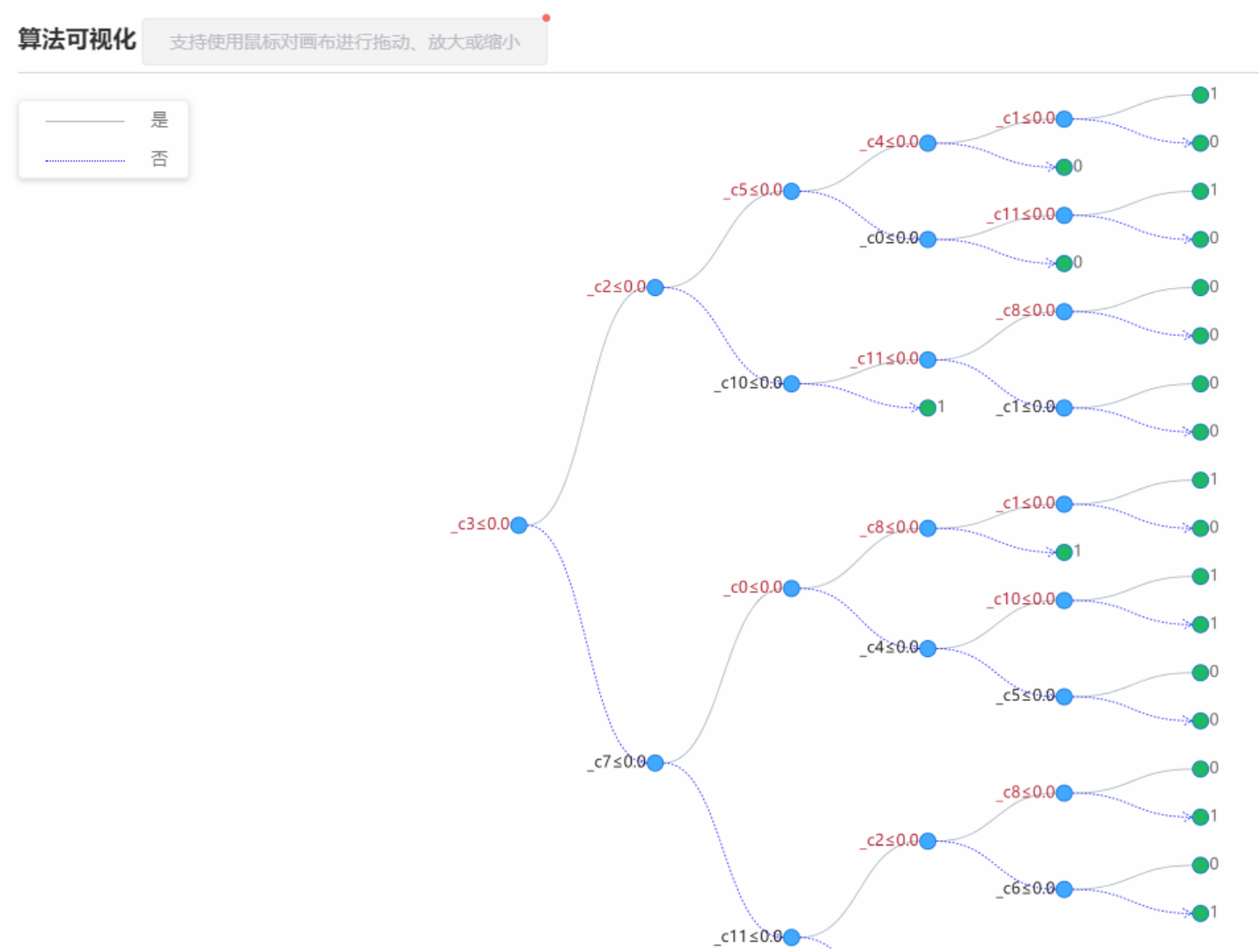
- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - maxDepth：决策树最大深度。
  - maxBins：决策树最大分支数。
  - minInfoGain：决策树分裂最小信息增益。
  - impurity：不纯度指标，支持 variance。
  - minInstancesPerNode：决策树节点最小样本数。
  - checkpointInterval：每多少轮设置 checkpoint 一次，在迭代轮数非常多的时候，可以降低因为计算节点失败导致的级联重算风险。
  - numTrees：最大迭代次数。
  - featureSubsetStrategy：特征采样比例策略，支持 auto、all、onethird、sqrt 和 log2 ，分别表示自动、全部、三分一、特征数的开方和特征数的对数。其中自动策略为：numTrees 为1时，该参数为 all；numTrees 大于1时，该参数为 sqrt。
  - subsamplingRate：样本数采样比例。
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持 RegressionEvaluator。
  - 评估指标：回归评估指标，支持 root mean squared error（均方根误差）、mean squared error（均方误差）、R2 metric（决定系数）和 mean absolute error（平均绝对值误差）。

- minInfoGain：连续范围参数，下界大于等于0，上届小于1，一般不超过0.1。
- maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
- maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
- minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。
- numTrees：离散正整数参数，比较合理的方式是根据默认值20在周围调节。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。
  - 结果格式：结果数据格式，默认为 parquet。
- 数模型可视化

  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判读节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



>!在训练时选择 PMML 格式才能可视化（ML 格式不行）

决策树回归

决策树算法是机器学习中常用的一类分类/回归算法。决策树算法有解释性好、可以处理类别特征、不需要做特征 scaling 等优点，可以表示非线性模型，最高可以支持百万级别的样本。

训练节点

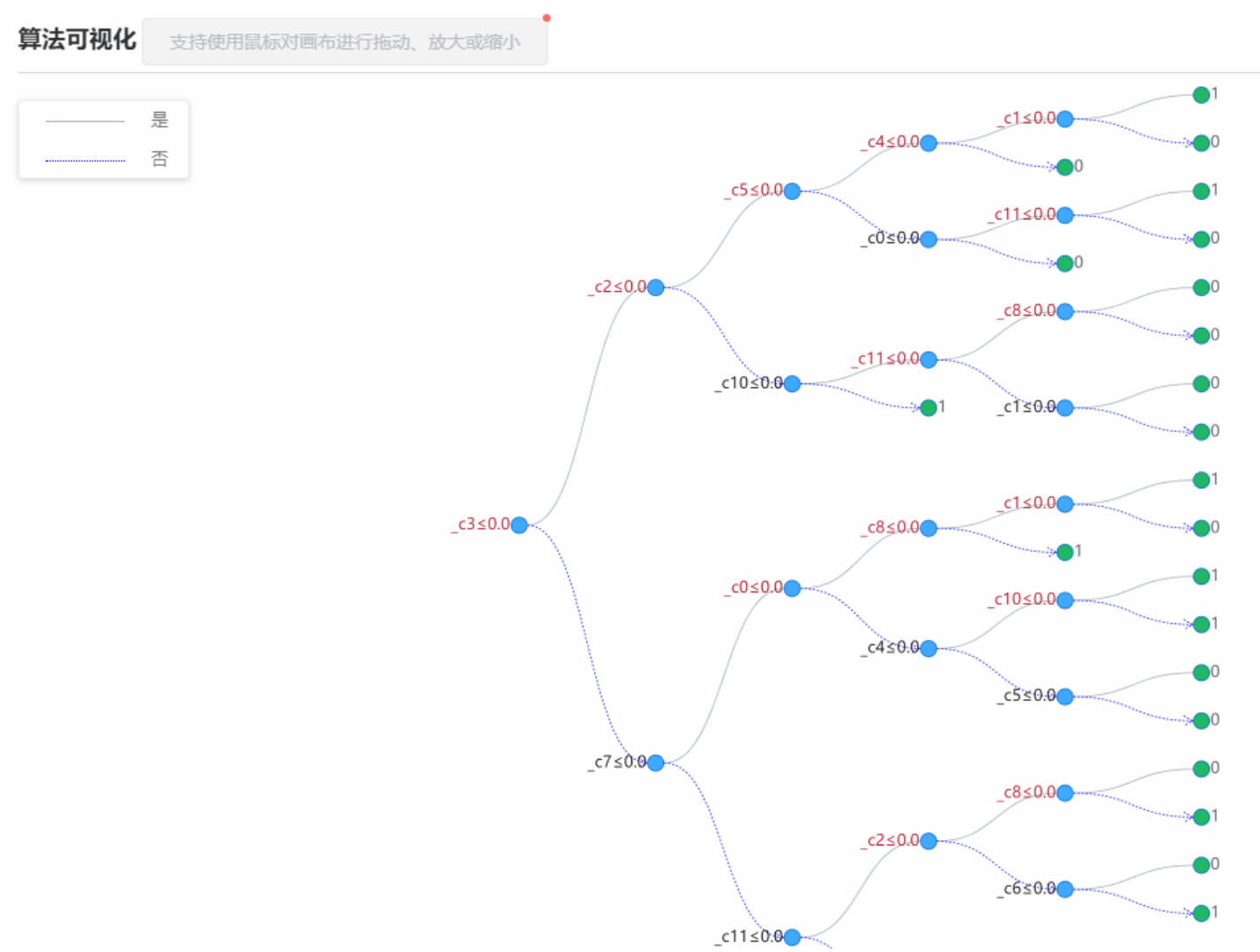
- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a－b、c 或者它们的混合，用英文逗号分割（例如0－10，15，17－19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。

    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
    - maxDepth：决策树最大深度。
    - maxBins：决策树最大分支数。
    - minInfoGain：决策树分裂最小信息增益。
    - impurity：不纯度指标，支持 variance。
    - minInstancesPerNode：决策树节点最小样本数。
    - checkpointInterval：每多少轮设置 checkpoint 一次，在迭代轮数非常多的时候，可以降低因为计算节点失败导致的级联重算风险。
        - 半自动调参
    - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
    - 评估方法：算法的评估方法，支持 RegressionEvaluator。
    - 评估指标：回归评估指标，支持root mean squared error（均方根误差）、mean squared error（均方误差）、R2 metric（决定系数）和 mean absolute error（平均绝对值误差）。
    - minInfoGain：连续范围参数，下界大于等于0，上届小于1，一般不超过0.1。
    - maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
    - maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
    - minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。

预测节点

- 输入
    - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
    - 训练数据格式：文本类型。
    - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
    - 结果路径：路径。
    - 结果格式：结果数据格式，默认为 parquet。
- 树模型可视化

  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判读节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



>!在训练时选择 PMML 格式才能可视化（ML 格式不行）

梯度提升树回归

梯度提升树是一种常用的分类回归算法，这里的实现根据论文 J.H. Friedman. "Stochastic Gradient Boosting." 1999。

训练节点

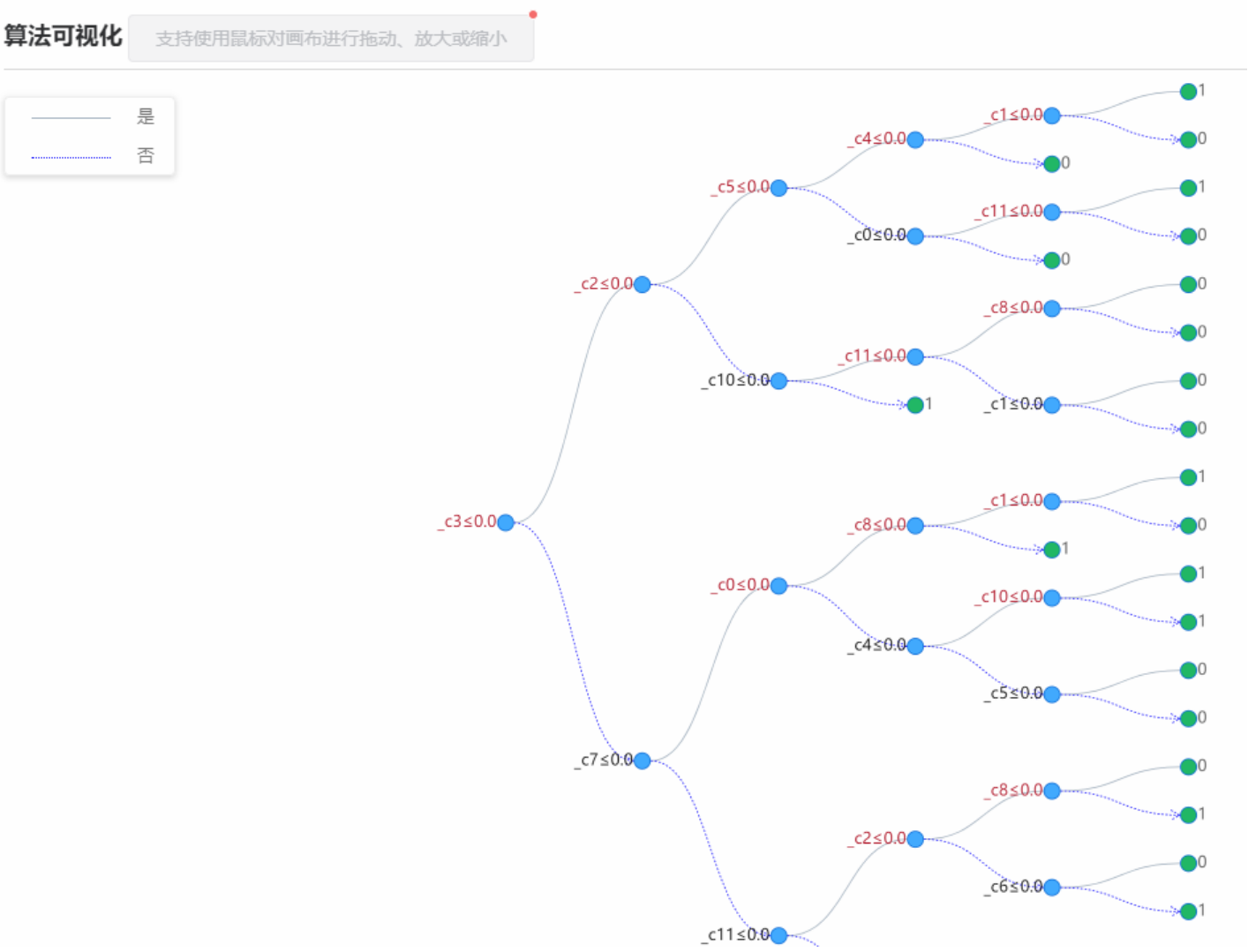
- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a－b、c 或者它们的混合，用英文逗号分割（例如0－10，15，17－19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。

    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
    - featureSubsetStrategy：特征采样比例策略，支持 auto、all、onethird、sqrt 和 log2，分别表示自动、全部、三分一、特征数的开方和特征数的对数。其中自动策略为：numTrees 为1时，该参数 all；numTrees大于1时，该参数为 sqrt。
    - impurity：不纯度指标，支持 variance。
    - maxBins：决策树最大分支数。
    - maxDepth：决策树最大深度。
    - maxIter：最大迭代次数。
    - minInfoGain：决策树分裂最小信息增益。
    - minInstancesPerNode：决策树节点最小样本数。
    - stepSize：步长，范围为（0，1）。
    - subsamplingRate：样本数采样比例。
- 半自动调参
    - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
    - 评估方法：算法的评估方法，支持 RegressionEvaluator。
    - 评估指标：回归评估指标，支持 root mean squared error（均方根误差）、mean squared error（均方误差）、R2 metric（决定系数）和 mean absolute error（平均绝对值误差）。
    - minInfoGain：连续范围参数，下界大于等于0，上界小于1，一般不超过0.1。
    - stepSize：连续范围参数，下界大于0，上界小于等于1。
    - subsamplingRate：连续范围参数，下界大于0，上界小于等于1。
    - maxBins：离散正整数参数，比较合理的方式是根据默认值32在周围调节。
    - maxDepth：离散正整数参数，比较合理的方式是根据默认值5在周围调节。
    - minInstancesPerNode：离散正整数参数，默认值为1，可根据样本数目适当调节。

预测节点

- 输入
    - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
    - 训练数据格式：文本类型。
    - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
    - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
    - 结果路径：路径。
    - 结果格式：结果数据格式，默认为 parquet。

- 数模型可视化
  树形算法在运行完成后，可以支持用户对模型结果进行可视化查看。树形图中蓝色节点为特征的判读节点，实线表示判断为“是”的路径，虚线表示判断为“否”的路径。绿色节点为分类结果。



> **注意：**
> 在训练时选择 PMML 格式才能可视化（ML 格式不行）

保序回归

保序回归是一种非参回归模型（nonparametric regression），该算法只有一个约束条件，即函数空间为单调递增函数的空间。详细说明请参考 保序算法说明。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。

- 训练数据格式：文本类型。
- 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
- 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。**对于这个算法，一般特征列只有1列，如果选择多列，实际生效的也只是选择的第一列。**
  - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。
  - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数：isotonic 表示升序还是降序。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。
  - 结果格式：结果数据格式，默认为 parquet。

聚类算法

最近更新时间：2020-07-14 14:48:38

KMeans

KMeans 是一种常用的聚类算法，将无标签的数据聚成 K 个类。平台提供的 KMeans 算法实现了并行的 k-means++ 的初始化算法。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - k：聚类类别数。
  - maxIter：最大迭代次数。
  - tol：容忍误差下界，低于该值的时候，算法停止迭代。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持 ClustringEvaluator。
  - 评估指标：聚类评估指标 sllhouette。
  - k：离散整正整数参数，取值需要大于等于2。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。

- 输出
  - 结果路径：路径。
  - 结果格式：结果数据格式，默认为 parquet。

高斯混合模型

高斯混合模型（Gaussian Mixture Model）是一种业界广泛使用的聚类算法，该方法使用了高斯分布作为参数模型，并使用了期望最大（Expectation Maximization）算法进行训练。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - k：聚类类别数。
  - maxIter：最大迭代次数。
  - tol：容忍误差下界，低于该值的时候，算法停止迭代。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持 ClustringEvaluator。
  - 评估指标：聚类评估指标 sllhouette。
  - k：离散整正整数参数,取值需要大于等于2。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。

- 输出
  - 结果路径：路径。
  - 结果格式：结果数据格式，默认为 parquet。

二分 KMeans

二分 K 均值算法属于层次聚类，详情可参考 官方文档 。

训练节点

- 输入
  - 训练路径：路径或者库表，Dense 结构，每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
    - 标签列：作为标签的列，要求特征列的特征标签为 double 类型。
    - 验证数据：半自动调参时用于评估的数据，格式与训练数据一致。
- 输出：ML 格式或者 PMML 格式的模型，保存在后台生成的路径下。
- 算法参数
  - k：聚类类别数。
  - maxIter：最大迭代次数。
- 半自动调参
  - 调参算法：默认贝叶斯调参，目前支持贝叶斯调参、网格调参和随机调参。
  - 评估方法：算法的评估方法，支持 ClustringEvaluator。
  - 评估指标：聚类评估指标 sllhouette。
  - k：离散整正整数参数，取值需要大于等于2。

预测节点

- 输入
  - 训练路径：路径或者库表，Dense 结构, 每一列对应一个特征、标签或者不参与计算的字段。
  - 训练数据格式：文本类型。
  - 数据分隔符：数据分隔符，默认为空白符或者逗号，可通过下拉框选择。
  - 特征列：作为训练特征的列，从0开始编号。对于库表可以直接勾选，对于普通路径，可填形式如 a - b、c 或者它们的混合，用英文逗号分割（例如0 - 10，15，17 - 19表示第0到10列、15、17到19列总共15列作为特征）。
- 输出
  - 结果路径：路径。

- 结果格式：结果数据格式，默认为 parquet。

关联规则

最近更新时间：2020-07-31 16:11:50

[2.0]关联规则挖掘

FPGrowth 是关联规则的一种实现方式，该算法将大规模的频繁集构建成 FPTree ，提高了提取频繁集的效率。平台提供的 FPGrowth 是 Li et al.，PFP: Parallel FP-growth for query recommendation 论文的并行实现，支持大规模的频繁集挖掘和关联规则的生成。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

参数

- 并行数：训练数据的分区数、Spark 的并行数。
- 抽样率：输入数据的采样率。
- 支持度：范围：0 - 1.0，频繁集的支持度阈值，item 出现的次数除以整个数据集的样本数。小于该阈值的频繁集将会被过滤。
- 置信度：范围：0 - 1.0，规则的置信度阈值，算法会过滤掉小于该阈值的规则。

测试节点

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：

- csv：csv 文件。
  - 输入数据包含 header 信息。
  - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
- parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。

预测方法

首先根据频繁项集生成关联规则。然后对于 itemsCol 中的每个事务，变换方法将其项与每个关联规则的前因进行比较。 如果记录包含特定关联规则的所有前提，则该规则将被视为适用，并且其结果将被添加到预测项中。 预测方法将所有适用规则的结果总结为预测项。

[2.0]Apriori

Apriori 算法是常用的用于挖掘出数据关联规则的算法，它用来找出数据值中频繁出现的数据集合，找出这些集合的模式有助于我们做一些决策。
Apriori 算法使用支持度来作为判断频繁项集的标准。Apriori 算法的目标是找到最大的K项频繁集。这里有两层意思，首先，要找到符合支持度标准的频繁集。但是这样的频繁集可能有很多。第二层意思就是要找到最大个数的频繁集。
Apriori 算法采用了迭代的方法，先搜索出候选1项集及对应的支持度，剪枝去掉低于支持度的1项集，得到频繁1项集。然后对剩下的频繁1项集进行连接，得到候选的频繁2项集，筛选去掉低于支持度的候选频繁2项集，得到真正的频繁2项集，以此类推，迭代下去，直到无法找到频繁k+1项集为止，对应的频繁 k 项集的集合即为算法的输出结果。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
      - 输出数据包含 header 信息。
      - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet。
  - 最小支持度：范围：0 - 1.0，频繁集的支持度阈值。
  - 最大支持度：范围：0 - 1.0，频繁集的支持度阈值。
  - 频繁项集最小长度：默认为1。
  - 频繁项集最大长度：默认为10。

推荐算法

最近更新时间：2020-07-14 14:47:08

交换最小二乘

ALS（alternating least squares）算法是交换最小二乘的简称。在机器学习中，ALS 特指使用交替最小二乘求解的一个协同推荐算法。它通过观察到的所有用户给商品的打分，来推断每个用户的喜好并向用户推荐适合的商品。

从广义上讲，推荐系统基于两种不同的策略：基于内容的方法和基于协同过滤的方法。Spark 中使用协同过滤的方式。协同过滤分析用户以及用户相关的产品的相关性，用以识别新的用户-产品相关性。协同过滤系统需要的唯一信息是用户过去的行为信息，例如对产品的评价信息。协同过滤是领域无关的，所以它可以方便解决基于内容方法难以解决的许多问题。

推荐系统依赖不同类型的输入数据，最方便的是高质量的显式反馈数据，它们包含用户对感兴趣商品明确的评价。例如，Netflix 收集的用户对电影评价的打分等级数据。但是显式反馈数据不一定总是找得到，因此推荐系统可以从更丰富的隐式反馈信息中推测用户的偏好。 隐式反馈类型包括购买历史、浏览历史、搜索模式甚至鼠标动作。例如，购买同一个作者许多书的用户可能喜欢这个作者。

许多研究都集中在处理显式反馈，然而在很多应用场景下，应用程序重点关注隐式反馈数据。因为可能用户不愿意评价商品或者由于系统限制我们不能收集显式反馈数据。在隐式模型中，一旦用户允许收集可用的数据，在客户端并不需要额外的显式数据。
Spark利用交换最小二乘解决矩阵分解问题分两种情况：数据集是显式反馈和数据集是隐式反馈。

训练节点

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet
- 参数
  - 用户列：User 所在列，从0开始计数。
  - 商品列：Item 所在列的列号，从0开始计数。
  - 打分列：评分所在列的列号，从0开始计数。
  - 隐变量数：矩阵变换中隐变量的个数，推荐值：10 - 200。
  - 正则系数：正则项系数，推荐值：0.01。
  - 最大迭代次数：最大迭代次数，推荐值：10 - 20。
  - 反馈方式：隐性反馈与显性反馈方式。

- 是否使用非负的限制：是否对最小二乘法使用非负的限制。
- 偏好行为强度的基准：专门针对隐性反馈的参数，决定了偏好行为强度的基准。

预测节点

- 输入
  - 输入数据路径：输入文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet
- 输出
  - 输出数据路径：输出文件所在路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet
- 参数
  - 用户列：User 所在列，从0开始计数。
  - 商品列：Item 所在列的列号，从0开始计数。

基于商品的协同过滤

根据所有用户对物品的评价，发现物品直接的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户。例如，用户 A 喜欢物品A和物品 C；用户B喜欢物品 A、B、C；用户 C 喜欢物品 A。从这些用户的历史喜好中，可以认为物品 A 和物品 C 比较类似，基于这个判断，用户 C 也可能喜欢物品 C。

训练节点

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。
- 参数
  - 用户列：User 所在列，从0开始计数。
  - 商品列：Item 所在列的列号，从0开始计数。

- 打分列：评分所在列的列号，从0开始计数。
- 相似产品的个数：和当前产品相似的产品个数。

预测节点

- 输入
  - 输入数据路径：输入文件所在路径。
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet
- 输出
  - 输出数据路径：输出文件所在路径。
  - 输出数据格式：格式包括以下两种：
    - csv：csv 文件。
      - 输入数据包含 header 信息。
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
    - parquet：列式存储格式 parquet
- 参数
  - 用户列：User 所在列，从0开始计数。
  - 商品列：Item 所在列的列号，从0开始计数。

基于用户的协同过滤算法

基于用户的协同过滤算法先使用统计技术寻找与目标用户有相同喜好的邻居，然后根据目标用户邻居的喜好产生向目标用户的推荐。基本原理就是利用用户访问行为的相似性来推荐用户可能感兴趣的资源。假设用户 A 喜好物品 A 和 C，用户B喜欢物品 B，用户 C 喜欢物品A、C 和 D。从这些用户的历史喜好中，我们发现用户 A 和用户 C 的口味和偏好是比较类似的，同时用户 C 还喜欢物品 D，那么我们判断用户 A 也可能喜欢物品 D，从而向用户 A 推荐物品 D。

训练节点

- 输入数据路径：输入文件所在路径。
  - 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet
  - 参数

        - 用户列：User 所在列，从0开始计数。
        - 商品列：Item 所在列的列号，从0开始计数。
        - 打分列：评分所在列的列号，从0开始计数。
        - 相似产品的个数：和当前产品相似的产品个数。

预测节点

- 输入
    - 输入数据路径：输入文件所在路径。
    - 输入文件类型：格式包括以下两种：
        - csv：csv 文件。
            - 输入数据包含 header 信息。
            - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
        - parquet：列式存储格式 parquet
- 输出
    - 输出数据路径：输出文件所在路径。
    - 输出数据格式：格式包括以下两种：
        - csv：csv 文件。
            - 输入数据包含 header 信息。
            - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
        - parquet：列式存储格式 parquet
- 参数
    - 用户列：User 所在列，从0开始计数。
    - 商品列：Item 所在列的列号，从0开始计数。

Angel 算法相关

Angel 算法相关内容请参考以下文档：

- [Angel]LR on PyTONA
- [Angel]FM on PyTONA
- [Angel]DeepFM on PyTONA
- [Angel]DeepAndWide on PyTONA
- [Angel]DCN on PyTONA
- [Angel]AttentionNet on PyTONA
- [Angel]AttentionFM on PyTONA
- [Angel]xDeepFM on PyTONA
- [Angel]PNN on PyTONA

时间序列

最近更新时间：2020-07-14 14:47:02

时间序列特征抽取

时序特征抽取是一个转换器，能对包含时间序列的数据进行特征抽取，产生一系列新的特征。该转换器支持的特征包括：最大值、最小值、均值、方差、标准差、偏度、峰度、中位数、极差、序列各项的平方和、序列相邻两项之差的均值等。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

选择特征列：表示需要计算的特征所在列，例如“1 – 12，15”，表示取特征在表中的1到12列，15列，从0开始计数。

自相关系数

自相关系数衡量 y（t）和 y（t-k）之间相关性。对于一个平稳时间序列，自相关系数（ACF）会快速的下降到接近0的水平，然而非平稳时间序列的自相关系数会下降的比较缓慢。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：

- csv：csv 文件
  - 输入数据包含 header 信息
  - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
- parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 滞后阶数：计算相距k个时间间隔的序列值之间的相关性。

偏自相关系数

偏自相关性是指去除 y（t-1）、y（t-2）、…、y（t-k+1）的影响之后，衡量 y（t）和 y（t-k）之间相关性。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 滞后阶数：计算相距 k 个时间间隔的序列值之间的相关性。
- 是否包含截距：包含或者不包含。

差分

对于非平稳序列，我们可以通过差分法，将其转换为平稳序列。计算相邻观测值之间的差值，这种方法被称为差分法。差分可以通过去除时间序列中的一些变化特征来平稳化它的均值，并因此消除（或减小）时间序列的趋势和季节性。

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 差分阶数：即d值。
- 是否反向：是或否。

KPSS检验

单位根检验是一种更客观的判定是否需要差分的方法。这个针对平稳性的统计假设检验被用于判断是否需要差分方法来让数据更平稳。
KPSS 检验原假设为数据是平稳的，我们要寻找能够证明原假设是错误的证据。因此，很小的 P 值（例如小于

0.05）说明需要进行差分。
具体原理请参考 原始论文。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 常数和趋势选择：回归中的包含项，是否带常数项和/或趋势项。

迪基福勒检验

单位根检验是一种更客观的判定是否需要差分的方法。这个针对平稳性的统计假设检验被用于判断是否需要差分方法来让数据更平稳。
迪基–福勒检验（ Dickey-Fuller test ）也是单位根检验方法，它可以测试一个自回归模型是否存在单位根（unit root）。
具体原理请参考 原始论文。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符

- parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 滞后阶数：向后追溯的观测值的数量。
- 常数和趋势选择：回归中的包含项，是否带常数项和/或趋势项。

Ljung-Box检验

Ljung-Box 检验是对 randomness 的检验,或者说是对时间序列是否存在滞后相关的一种统计检验。
它可以用于两方面的检验：

- 纯随机性检验，p 值小于5%,序列为非白噪声
- 用于检验某个时间段内的一系列观测值是不是随机的独立观测值。如果观测值并非彼此独立，一个观测值可能会在 i 个时间单位后与另一个观测值相关，形成一种称为自相关的关系。自相关可以削减基于时间的预测模型（例如时间序列图）的准确性，并导致数据的错误解释。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件

- 输出数据包含 header 信息
- 输出数据分割符：主要包括逗号、空格、分号、星号等分割符

- parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 滞后阶数：计算相距k个时间间隔的序列值之间的相关性。

ARIMA（差分整合移动平均自回归模型）

许多非平稳序列差分后会显示出平稳序列的性质，这个非平稳序列为差分平稳序列。差分平稳序列使用差分整合移动平均自回归模型（ARIMA）进行拟合。
ARIMA 包含3个部分：AR、I、MA。

- AR 表示 auto regression，即自回归模型。
- I 表示 integration，即单整阶数，时间序列模型必须是平稳性序列才能建立计量模型，ARIMA 模型作为时间序列模型也不例外，因此首先要对时间序列进行单位根检验，如果是非平稳序列，就要通过差分来转化为平稳序列，经过几次差分转化为平稳序列，就称为几阶单整。
- MA 表示 moving average，即移动平均模型。
  可见，ARIMA 模型实际上是 AR 模型和 MA 模型的组合。p 为自回归模型滞后阶数，d 为时间序列单整阶数，q 为移动平均模型滞后阶数。当 p、d 为0时，ARIMA 模型退化为 MA 模型，当 q、d 为0时，ARIMA 模型退化为 AR 模型。仅当 d 为0时，ARIMA 模型退化为 ARMA 模型。

训练节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 自回归项数：p 值，默认为1。
- 差分次数：d值，默认为0。

- 滑动平均项数：q值，默认为1。
- 优化器评价次数：默认为10000。
- 迭代次数：默认为10000。
- 模型是否包括截距项：是否带截距，默认为是。

预测节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。

AutoARIMA(自动差分整合移动平均自回归模型)

自动差分整合移动平均自回归模型给定 maxP、maxD 和 maxQ，它通过搜索获取合适的 p、d 和 q 值，来建立一个差分整合移动平均自回归模型。

训练节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息

        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 最大自回归项数：maxP 值，默认为2。
- 最大差分次数：maxD 值，默认为2。
- 最大滑动平均项数：maxQ 值，默认为2。
- 优化器评价次数：默认为10000。
- 迭代次数：默认为10000。
- intercept：是否带截距，默认为 true。

预测节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
    - csv：csv 文件
        - 输入数据包含 header 信息
        - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
    - csv：csv 文件
        - 输出数据包含 header 信息
        - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。

EWMA（指数加权移动平均法）

指数加权移动平均法（Exponentially Weighted Moving Average，EWMA）是一种常用的序列数据处理方式。在 t 时刻，根据实际的观测值可以求取 EWMA(t)：EWMA(t) = aY(t) + (1−a)EWMA(t−1)，t = 1，2，…..，n；其中，EWMA（t）表示 t 时刻的估计值；Y（t）表示 t 时刻的测量值；n 表示所观察的总的时间；a(0 < a < 1)表示对于历史测量值权重系数。之所以称之为指数加权，是因为加权系数a是以指数式递减的，即各指数随着时间而指数式递减。用 n 表示为 a = 2/(n + 1)。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 历史值的权重系数：默认0.2

GARCH（广义自回归条件异方差模型）

自回归条件异方差模型（ARCH）模型的实质是使用残差平方序列的 q 阶移动平移拟合当期异方差函数值，由于移动平均模型具有自相关系数q阶截尾性，所以 ARCH 模型实际上只适用于异方差函数短期自相关系数。
但是在实践中，有些残差序列的异方差函数是具有长期自关性，这时使用 ARCH 模型拟合异方差函数，将会产生很高的移动平均阶数，增加参数估计的难度并最终影响 ARCH 模型的拟合精度。
为了修正个问题，提出了广义自回归条件异方差模型（GARCH）， 该模型简记为 GARCH（p，q）。GARCH 模型实际上是在 ARCH 的基础上，增加考虑异方差函数的 p 阶自回归性而形成，它可以有效的拟合具有长期记忆性的异方差函数。ARCH 模型是 GARCH 模型的一个特例，p = 0的 GARCH（p，q）模型。
本平台支持 p = 1、q = 1 的广义自回归条件异方差模型。

训练节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 优化器评价次数：默认为100。
- 迭代次数：默认为100。

预测节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。

HoltWinters（三次指数平滑模型）

移动平均模型在解决时间序列问题上简单有效，但它们的计算比较难，因为不能通过之前的计算结果推算出加权移动平均值。此外，移动平均法不能很好的处理数据集边缘的数据变化，也不能应用于现有数据集的范围之外。因此，移动平均法的预测效果相对较差。指数平滑法（exponential smoothing）是一种简单的计算方案，可以有效的避免上述问题。按照模型参数的不同，指数平滑的形式可以分为一次指数平滑法、二次指数平滑法、三次指数平滑法。其中一次指数平滑法针对没有趋势和季节性的序列，二次指数平滑法针对有趋势但是没有季节特性的时间序列，三次指数平滑法则可以预测具有趋势和季节性的时间序列。术语 Holt-Winter 指的就是三次指数平滑。三次指数平滑模型（HoltWinters）按照季节性分量的计算方式不同，可以分为累加式季节性分量和累乘式季节性分量。

训练节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。
- 季节频率：e.g. 月度数据为12。
- 优化器评价次数：默认为10000。
- 迭代次数：默认为10000。
- 模型类型：有累加式和累乘式两种，默认是累加式。

预测节点

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 时间列：时间数据所在的列。
- 时间序列数据列：时间序列数据所在的列。

TS-Box-Cox变换

BoxCox 转换通过 lambda 参数对数值特征列进行变换，将特征数据变换为服从正太分布的数据。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet
  - 输入时间列
  - 输入时间序列数据列

参数

- 变换参数：数值为0时进行对数变换
- 是否反向：正向/反向可供选择

主题模型

最近更新时间：2020-07-14 14:46:57

隐式狄利克雷分布

LDA 是一种概率主题模型：隐式狄利克雷分布（Latent Dirichlet Allocation，简称 LDA）。
LDA 是2003年提出的一种主题模型，它可以将文档集合中每篇文档的主题以概率分布的形式给出。通过分析一些文档，我们可以抽取出它们的主题（分布），根据主题（分布）进行主题聚类或文本分类。
同时，它是一种典型的词袋模型，即一篇文档是由一组词构成，词与词之间没有先后顺序的关系。一篇文档可以包含多个主题，文档中每一个词都由其中的一个主题生成。

训练节点

- 输入
  - 输入数据路径：输入文件所在路径
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- 算法参数
  - 特征列：作为训练特征的列，从0开始编号。可填形式如 a - b、c 或它们的混合，用英文逗号分割（例如0 - 10，15，17-19表示第0到10列、15、17到19列总共15列作为特征）
  - 迭代次数：算法迭代次数
  - 优化方法：online 或者 em

预测节点

- 输入
  - 输入数据路径：输入文件所在路径
  - 输入文件类型：格式包括以下两种：
    - csv：csv 文件
      - 输入数据包含 header 信息
      - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
    - parquet：列式存储格式 parquet
- 输出
  - 结果路径：路径
  - 输出文件类型：格式包括以下两种：
    - csv：csv 文件

- 输出数据包含 header 信息
- 输出数据分割符：主要包括逗号、空格、分号、星号等分割符

- parquet：列式存储格式 parquet

表算子

最近更新时间：2020-07-14 14:46:52

ExceptDistinct

类似 SQL 中的 Except distinct 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 左表特征列：左表输入特征列，从0开始计数，例如“1-12，15”
- 右表特征列：右表输入特征列，从0开始计数，例如“1-12，15”

Intersect

类似 SQL 中的 Intersect 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符

        - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 左表特征列：左表输入特征列，从0开始计数，例如“1-12，15”
- 右表特征列：右表输入特征列，从0开始计数，例如“1-12，15”

Join

类似 SQL 中的 Join 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 左表特征列：左表输入特征列，从0开始计数，例如“1 - 12，15”
- 右表特征列：右表输入特征列，从0开始计数，例如“1 - 12，15”
- join 类别：包括 inner、cross、outer、full、left、right 几类

- on 条件：格式类似于a1，b1，a2，b2。其中a1和a2是左表的字段，b1和b2是右表的字段
- 左右表添加前缀：左表和右表字段是否添加前缀
- 左右表字段前缀名称：左表和右表字段添加的前缀名称

Limit

类似 SQL 中的 Limit 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含header信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

行数：选择的行数。

RegexpExtract

类似 SQL 中的 Regexp Extract 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv： csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 选择特征列：选择的特征列，从0开始计数，支持单列。
- 正则表达式：匹配的正则表达式。
- group id：组别 ID。

RegexpReplace

类似 SQL 中的 Regexp Replace 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 选择特征列：选择的特征列，从0开始计数，支持单列
- 模式串
- 替换串

SelectColumn

类似 SQL 中的 Select 操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

选择特征列：检测的特征列，从0开始计数，例如“1－12，15”。

Union

类似 sql 中的 Union ALL操作。

输入

- 输入数据路径：输入文件所在路径
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件
    - 输入数据包含 header 信息
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

输出

- 输出数据路径：输出文件所在路径
- 输出数据格式：格式包括以下两种：
  - csv： csv 文件

    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符
  - parquet：列式存储格式 parquet

参数

- 左表特征列：左表输入特征列，从0开始计数，例如“1－12，15”
- 右表特征列：右表输入特征列，从0开始计数，例如“1－12，15”

统计分析

最近更新时间：2020-07-14 14:46:48

全表统计

全表统计是对数据全表做的统计，根据不同的数据类型能够输出六种不同内容的统计结果。

参数

- 特征列：输入特征列序号，从0开始计数，如0 - 29。
- 输入数据是否包含 header 信息：是或否。
- 输入数据分隔符：主要包括逗号、空格、分号、星号等分割符。

输出	参数
category	top，counts，uniques，missing，missing_perc， types
unique	counts，uniques，missing，missing_perc，types
date	max，min，range，counts，uniques，missing，missing_perc，types
numerics	mean，std，variance，min，max，5%，25%，50%，75%，95%，iqr，kurtosis，skewness，sum，mad，cv ，zeros_num，zeros_perc，deviating_of_mean，deviating_of_mean_perc，deviating_of_median，deviating_of_median_perc，top_correlations，counts，uniques，missing，missing_perc，types
constant	This is a constant value
bool	true_class_count，true_class_perc，false_class_count，false_class_perc，counts，uniques，missing，missing_perc，types

统计分析

用户可以通过统计分析算子获取对数据做出的统计数据与统计图。目前支持七种统计显示方式：协方差、相关系数矩阵、正态检验、卡方拟合性检验、卡方独立性检验、经验密度图、离散值特征分析图。

任务流连接方式：

- 【COS 数据源】>【SelectColumn】>【StatisticAnalysis】
- 【本地数据】>【SelectColumn】>【StatisticAnalysis】

参数配置时有直接点选字段功能，因此请先将前两步运行成功后，再使用此算子。

参数：

- 输入数据是否包含 header 信息：是或否。
- 输入数据分隔符：主要包括逗号、空格、分号、星号等分割符。
- 统计方式：支持选择不同的统计方式。
- 选择字段：针对不同的统计方式选择字段。

图算法

最近更新时间：2020-08-06 16:50:15

[2.0]CommonFriends

计算两个好友的共同好友数，某种程度上可以刻画两个节点之间的紧密程度。

输入

- 输入数据路径：输入文件所在路径，无权网络数据, 数据格式为两列 srcId(long) | dstId(long), 其中|为分隔符，分隔字段表示空白符或者逗号等。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

参数说明

- src：源节点列。
- dst：目标节点列。
- numPartition：分区数。

资源参数

- drive 节点资源类型：请选择合适的 drive 节点机型。
- executor 节点资源类型：请选择合适的 executor 节点机型。
- num-executors：分配计算节点数目，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
- spark-conf：spark常用参数配置，如压缩、序列化、网络等。

[2.0]HyperAnf

估计网络的平均半径，参考论文 HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget 开发，详细细节请看论文。

输入

- 输入数据路径：输入文件所在路径，无权网络数据, 数据格式为两列 srcId(long) | dstId(long), 其中|为分隔符，分隔字段表示空白符或者逗号等。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。
    >算法结果保存路径，共两列，其中第一列为 round 值，第二列为 anf 值，其中round = -1对应的 anf 为最终估计值。

参数说明

- src：源节点列。
- dst：目标节点列。
- numPartition：分区数。
- maxIter：最大迭代次数。

资源参数

- drive 节点资源类型：请选择合适的 drive 节点机型。
- executor 节点资源类型：请选择合适的 executor 节点机型。
- num-executors：分配计算节点数目，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
- spark-conf：spark 常用参数配置，如压缩、序列化、网络等。

[2.0]LPA

LPA（Label Propagation Algorithm）是最简单的社区发现算法，通过标签扩散发掘网络的社区关系。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。
    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息。
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。
    > 算法结果保存路径，共两列，其中第一列为节点 ID，第二列为节点对应的社区 ID。社区 ID 相同表示属于同一个社区。

参数说明

- src：源节点列。
- dst：目标节点列。
- numPartition：分区数。

资源参数

- num-executors：使用多少个 Spark 节点。
- driver-memory：Spark driver 的内存大小。
- executor-cores：每个 Spark 节点使用多少个 core。
- executor-memory：每个 Spark 节点使用的内存大小。
- spark-conf：Spark 的其他参数。由于权限原因，需要用户额外提供 ugi 参数 spark.hadoop.hadoop.job.ugi=用户名：密码。

[2.0]EffectiveSize

EffectiveSize 是由结构空洞理论得到的网络度量指标，是 ego-network 中节点的重要衡量指标。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：

- csv：csv 文件。
  - 输入数据包含 header 信息。
  - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
- text：本文件。
- parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。
    > 算法结果保存路径，共三列，其中第一列为节点 ID，第二列为 effectiveSize 值，第三列为 redundancyCol 值。

参数说明

- src：源节点列。
- dst：目标节点列。
- numPartition：分区数。

资源参数

- num-executors：使用多少个 Spark 节点。
- driver-memory：Spark driver 的内存大小。
- executor-cores：每个 Spark 节点使用多少个 core。
- executor-memory：每个 Spark 节点使用的内存大小。
- spark-conf：Spark 的其他参数。由于权限原因，需要用户额外提供 ugi 参数 spark.hadoop.hadoop.job.ugi=用户名：密码。

[2.0]PageRank

PageRank 是著名的节点排序算法，由 Google 发表。

输入

- 输入数据路径：输入文件所在路径。
- 输入文件类型：格式包括以下两种：
  - csv：csv 文件。
    - 输入数据包含 header 信息。

    - 输入数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - text：本文件。
  - parquet：列式存储格式 parquet。

输出

- 输出数据路径：输出文件所在路径。
- 输出数据格式：格式包括以下两种：
  - csv：csv 文件。
    - 输出数据包含 header 信息
    - 输出数据分割符：主要包括逗号、空格、分号、星号等分割符。
  - parquet：列式存储格式 parquet。
    > 算法结果保存路径，共三列，其中第一列为节点 ID，第二列为 effectiveSize 值，第三列为 redundancyCol 值。

参数说明

- src：源节点列。
- dst：目标节点列。
- numPartition：分区数。
- maxIter：最大迭代次数。
- tol：最小容忍误差，当误差小于该值时，算法迭代提早结束。

资源参数

- num-executors：使用多少个 Spark 节点。
- driver-memory：Spark driver 的内存大小。
- executor-cores：每个 Spark 节点使用多少个 core。
- executor-memory：每个 Spark 节点使用的内存大小。
- spark-conf：Spark 的其他参数。由于权限原因，需要用户额外提供 ugi 参数 spark.hadoop.hadoop.job.ugi=用户名：密码。

Angel 算法相关

Angel 算法相关内容请参考以下文档：

- [Angel]Closeness on SONA
- [Angel]Common Friends on SONA
- [Angel]FastUnfolding on SONA
- [Angel]Kcore on SONA
- [Angel]LINE on SONA

- [Angel]PageRank on SONA
- [Angel]HIndex on SonA
- [Angel]LPA on SONA
- [Angel]GraphSage_Supervised
- [Angel]Word2Vec on SONA

深度学习
自然语言处理

最近更新时间：2021-12-29 11:41:36

BERT 文本分类

BERT文本分类算法首先使用 BERT 网络，产生要分类的句子的向量表示，再通过全连接层网络对句子进行分类。

算法 IO 参数

- 训练数据：每一行为一个句子或两个句子，词与词之间用空格分隔，句子之间以及句子和标签之间用特定分隔符分隔（分隔符在算法参数中可以设置），**句子在分隔符左侧，标签在分隔符右侧**。
- 验证数据：格式同训练数据。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- bert 参数目录：存储预训练的 bert 参数的目录，对于中文和英文，可以分别下载 压缩包一 和 压缩包二，并填写解压后的目录

算法参数

- 标签分隔符：用于分隔句子和标签的分隔符。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 句子分隔符：对于输入为两个句子的数据，分隔两个句子的分隔符。
- 最大句子长度：BERT 要求固定的输入长度，因此不足这一长度的句子将被补齐，超过的将被截断。

模型 IO 参数

- 预测数据输入：要预测的数据，格式同训练数据，可以没有标签。
- 输出结果路径：存放预测结果的 csv 文件的路径。csv 文件中每一行是一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的类别，第三列是预测的类别。

模型参数：

- 标签分隔符：用于分隔句子和标签的分隔符。
- 句子分隔符：对于输入为两个句子的数据，分隔两个句子的分隔符。

实例生成

1. 使用数据节点，上传数据，数据格式见上文【训练数据】部分。

2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。
3. 将两份数据分别连接到 BERT 文本分类节点的两个输入桩。单击【运行】开始训练。

LSTM 文本分类

LSTM 文本分类算法首先使用双向 LSTM 网络 参考文档 产生要分类的句子的向量表示，再通过全连接层网络对句子进行分类。

算法 IO 参数

- 训练数据：每一行为一个句子，词与词之间用空格分隔，句子和标签之间用特定分隔符分隔（分隔符在算法参数中可以设置），**句子在分隔符左侧，标签在分隔符右侧**。
- 验证数据：格式同训练数据。
- 模型目录：存放模型文件和日志文件的目录。（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录 。

算法参数

- 分隔符：用于分隔句子和标签的分隔符。
- 词向量维度：网络中词向量的维度。
- LSTM 维度：网络中句子的向量表示的维度。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 是否使用预训练好的词向量：如设为 True，可填写词向量文件路径。词向量文件格式与 glove 词向量官方格式相同。**如果使用预训练好的词向量，预训练词向量的维度应等于参数【词向量维度】的值**。

模型 IO 参数

- 预测数据输入：要预测的数据，格式同训练数据，可以没有标签。
- 输出结果路径：存放预测结果的 csv 文件的路径。csv 文件中每一行是一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的类别，第三列是预测的类别。

模型参数

分隔符：预测数据中分隔句子和标签的分隔符。

实例生成

1. 使用数据节点上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。

3. 将两份数据分别连接到 LSTM 句子分类节点的两个输入桩，单击【运行】开始训练。

Fasttext 文本分类

Fasttext 是一种简单有效的句子分类算法， 通过词向量以及 ngram 向量的平均值计算出句子的向量表示，再通过全连接层网络对句子进行分类 参考文档。

算法 IO 参数

- 训练数据：每一行为一个句子，词与词之间用空格分隔，句子和标签之间用特定分隔符分隔（分隔符在算法参数中可以设置）。**句子在分隔符左侧，标签在分隔符右侧**。
- 验证数据：格式同训练数据。
- 模型目录：存放模型文件和日志文件的目录。（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录。

算法参数

- 分隔符：用于分隔句子和标签的分隔符
- 词向量维度：网络中词向量的维度。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 是否使用预训练好的词向量：如设为 True，可填写词向量文件路径。词向量文件格式与 glove 词向量官方格式相同。**如果使用预训练好的词向量，预训练词向量的维度应等于参数【词向量维度】的值**。

模型 IO 参数

- 预测数据输入：要预测的数据，格式同训练数据，可以没有标签。
- 输出结果路径：存放预测结果的 csv 文件的路径。csv 文件中每一行是一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的类别，第三列是预测的类别。

模型参数

- 分隔符：预测数据中分隔句子和标签的分隔符。

实例生成

1. 使用数据节点，上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。
3. 将两份数据分别连接到 Fasttext 节点的两个输入桩，单击【运行】开始训练。

TextCNN 文本分类

算法说明

TextCNN 使用卷积神经网络产生句子的向量表示，再通过全连接层网络对句子进行分类 参考文档。

算法 IO 参数

- 训练数据：每一行为一个句子，词与词之间用空格分隔，句子和标签之间用特定分隔符分隔(分隔符在算法参数中可以设置)。**句子在分隔符左侧，标签在分隔符右侧**。
- 验证数据：格式同训练数据。
- 模型目录：存放模型文件和日志文件的目录。（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

- 分隔符：用于分隔句子和标签的分隔符。
- 词向量维度：网络中词向量的维度。
- 各层网络卷积核大小：即 kernel_size。
- 各层网络卷积核个数：即通道数。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 是否使用预训练好的词向量：如设为 True，可填写词向量文件路径。词向量文件格式与 glove 词向量官方格式相同。**如果使用预训练好的词向量，预训练词向量的维度应等于参数【词向量维度】的值**。

模型 IO 参数

- 预测数据输入：要预测的数据，格式同训练数据，可以没有标签。
- 输出结果路径：存放预测结果的 csv 文件的路径。csv 文件中每一行是一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的类别，第三列是预测的类别。

模型参数

- 分隔符：预测数据中分隔句子和标签的分隔符。

实例生成

1. 使用数据节点上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。
3. 将两份数据分别连接到 TextCNN 节点的两个输入桩，单击【运行】开始训练。

BiLSTM-CRF 序列标注

BiLSTM-CRF 是一种常用的序列标注模型，能用于分词，词性标注，命名实体识别等序列标注任务。模型使用双向 LSTM 网络产生句子中的各个词语的向量表示，并据此计算词语标签的概率分布，然后使用 CRF 计算总概率最大的标签序列 参考文档。

算法 IO 参数

- 训练数据：文本文件，其中每一行是一个句子及其各个词语的对应标签，句子和标签之间由特定分隔符分隔（默认为 __label__），句子中的各个词语之间，以及各个标签之间用空格分隔。由于算法对句子中的每个词语进行标注，词语个数必须等于标签个数。
- 验证数据：文本文件，格式与训练数据相同。
- 模型目录：存放模型文件和日志文件的目录。（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

- 批处理大小：即算法 batch_size。
- 学习率：即算法的 learning_rate。
- 训练次数：即将训练数据训练的 epoch 数。
- 词向量维度：即每个词语向量表示的维度。
- LSTM 维度：即每个词语 LSTM 向量表示的维度。
- 使用预训练好的词向量模型：如设为 True，可填写词向量文件路径。词向量文件格式与 glove 词向量官方格式相同。**如果使用预训练好的词向量，预训练词向量的维度应等于参数【词向量维度】的值。**

模型 IO 参数

- 预测数据输入：要进行预测的数据，格式同训练数据，可以没有标签。
- 预测结果输出路径：存放预测结果的 csv 文件的路径。csv 文件中每行为一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的序列标签，第三列是预测的序列标签。

模型参数

- 批处理大小：预测时的 batch_size。

实例生成

1. 使用数据节点，上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。
3. 将两份数据分别连接到 BiLSTM-CRF 节点的两个输入桩，单击【运行】开始训练。

Word2Vec 词向量

Word2Vec 是一种经典的词向量算法，能够从大量文本中学习出各个词语的向量表示。这一向量表示可以用作其它深度学习模型的初始值 参考文档。

输入参数

- 训练数据：文本文件，词语之间使用空格分隔。
- 验证数据：（可选）包含四元组的数据集，如 word2vec 官方 questions-words.txt，用于评价生成的词向量的质量。

输出参数

保存路径：生成的词向量文件的保存路径。生成的文件为文本文件，每行为一个词语及其向量表示，词语和向量之间，向量中的各个数字之间使用空格分隔。

算法参数

- 词向量维度：要训练的词向量维度。
- 窗口大小：skip-gram 算法中的 window_size 参数。
- 最小出现次数：只训练出现次数大于这一次数的词语的词向量。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate 参数。
- 批处理大小：即训练的 batch_size。
- 负采样个数：skip-gram 算法中的负样本个数。

实例生成

1. 使用数据节点上传数据，数据格式请参见【输入参数】部分。
2. 将数据节点连接到 word2vec 节点，配置好保存路径，单击【运行】开始训练。

Glove 词向量

Glove 是一种经典的词向量算法，能够从大量文本中学习出各个词语的向量表示。这一向量表示可以用作其它深度学习模型的初始值 参考文档。

输入参数

- 训练数据：文本文件，词语之间使用空格分隔。
- 验证数据：（可选）包含四元组的数据集，如 word2vec 官方 questions-words.txt，用于评价生成的词向量的质量。

输出参数

保存路径：生成的词向量文件的保存路径。生成的文件为文本文件，每行为一个词语及其向量表示，词语和向量之间，向量中的各个数字之间使用空格分隔。

算法参数

- 词向量维度：要训练的词向量维度。
- 窗口大小：计算共现矩阵时使用的 window_size 参数。
- 最小出现次数：只训练出现次数大于这一次数的词语的词向量。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate 参数。
- 批处理大小：即训练的 batch_size。
- 最大词汇表大小：如果训练数据中单词总数超过最大词汇量 n，则只训练出现频率最高的 n 个词的词向量。

实例生成

1. 使用数据节点上传数据，数据格式请参考【输入参数】部分。
2. 将数据节点连接到 glove 节点，配置好保存路径，单击【运行】开始训练。

中文文本摘要

中文文本摘要首先使用 BERT 网络，可查看 论文链接，产生文中句子的向量表示，再通过 TextRank 算法抽取权重较高的句子。

输入参数

文本数据输入：要抽取摘要的文本，要求每行为一个句子。

输出参数

输出数据：抽取后的摘要。

算法参数

摘要句子数目：从文本中选为摘要的句子数量。

实例生成

1. 使用数据节点，上传数据，数据格式见上文【文本数据输入】部分。
2. 将输入数据连接到【中文文本摘要】节点的输入桩，单击【运行】开始运行。

中文关键词抽取

中文关键词抽取算法使用 TF-IDF 抽取输入文本中的关键词。

输入参数

文本数据输入：要抽取关键词的文本，无需预先分词。每行为一个句子。

输出参数

输出数据：抽取后的关键词，每行为输入数据对应行的句子中抽取的关键词。

算法参数

关键词个数：从每行文本中抽取的关键词数量。

实例生成

1. 使用数据节点，上传数据，数据格式见上文【 文本数据输入 】部分。
2. 将输入数据连接到【 中文关键词抽取 】节点的输入桩，单击【 运行 】开始运行。

中文词频统计

词频统计节点统计文本中词语的出现频率，并从高到低排列。

输入参数

输入数据：要统计词频的文本。

输出参数

输出文件：词频统计结果。

实例生成

1. 使用数据节点，上传数据，数据格式见上文【 文本数据输入 】部分。
2. 将输入数据连接到【 词频统计 】节点的输入桩，单击【 运行 】开始运行。

中文新词发现

PMI（点互信息）和左右熵能够刻画一个文本片段的凝固程度和灵活运用程度，因此可以用于发现文本中不存在词库中的新词。

输入参数

- 输入数据：未分词的中文纯文本文件。
- 词库：（可选）文本文件，每行为一个词。如发现的新词已经存在这个词库中，则跳过这一新词。

输出参数

输出数据：算法发现的新词，每行为一个词。

算法参数

- 最大词语长度：只考虑长度不超过这一长度的文本片段构成新词的可能性。
- 保留前 n 个新词：只保留可能性最大的前若干个新词。

实例生成

1. 使用数据节点上传数据，数据格式请参考【输入数据】部分
2. 将数据连接到【新词发现】节点的输入桩，设置好输出数据路径，单击【运行】开始发现新词。

BERT-CRF 序列标注

BERT-CRF 是一种表现十分出色的序列标注模型， 能用于分词，词性标注，命名实体识别等序列标注任务。模型使用 BERT 产生句子中各个字的向量表示，并据此计算词语标签的概率分布，然后使用 CRF 计算总概率最大的标签序列。

算法 IO 参数

- 训练数据：文本文件，其中每一行是一个句子及其各个词语的对应标签，句子和标签之间由特定分隔符分隔（默认为 __label__），句子中的各个词语之间，以及各个标签之间用空格分隔。由于算法对句子中的每个词语进行标注，词语个数必须等于标签个数。
- 验证数据：文本文件，格式与训练数据相同，选填。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- bert 参数目录：存储预训练的 bert 参数的目录，对于中文和英文，可以分别下载 压缩包一 和 压缩包二，并填写解压后的目录。

算法参数

- 最大句子长度：BERT 要求固定的输入长度，因此不足这一长度的句子将被补齐，超过的将被截断。
- 标签分隔符：用于分割句子和标签的分隔符。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 输入转化：带标签的训练数据最小单位是词， 而 BERT 的输入数据最小单位应该为字，需要转换原始数据。
- 存储 checkpoint 步数：即存储 checkpoint 的频率。

模型 IO 参数

- 预测数据：文本数据，数据可包含标签信息以备模型评估使用。格式与训练数据相同。
- 预测结果输出：输出预测结果的 csv 文件的路径。csv 文件中每行为一个句子及其预测结果，每一行中，第一列是要预测的句子，第二列是标注的序列标签，第三列是预测的序列标签。

模型参数

标签分隔符：用于分割句子和标签的分隔符。

实例生成

1. 使用数据节点上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据只有一份，可以使用【输入】>【数据转换】中的【文本数据切分】节点，将上传的数据按比例分为训练数据和验证数据。
3. 将两份数据分别连接到 BERT-CRF 节点的前两个输入桩，单击【运行】开始训练。

BERT 中文问答

BERT 可用于机器阅读理解任务，机器阅读理解指问题答案能够在提供文本中找到对应的表示。模型使用 BERT 产生句子中各个字的向量表示，并据此计算出每个字作为答案开头和答案结尾的 logit 值，最后逻辑判断确定文本中答案所在区域。

算法 IO 参数

- 训练数据：json文件，下图表示文件中一个key：value 对。其中 Q_ANN_VAL 表示问题答案对序号、question 表示问题描述、evidences 表示该问题的一至多个证据，其中每一个证据由序号（Q_ANN_VAL），答案（answer）以及证据（evidence）组成。

```
"Q_ANN_VAL_001093": {
    "question": "身体内哪种元素缺失容易导致抽筋?",
    "evidences": {
        "Q_ANN_VAL_001093#00": {
            "answer": [
                "钙"
            ],
            "evidence": "小腿肚抽筋通常是由于钙流失导致骨质疏松引起的。"
        }
    }
},
```

- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- bert 参数目录：存储预训练的 bert 参数的目录，对于中文和英文，对于中文和英文，可以分别下载 压缩包一 和 压缩包二，并填写解压后的目录。

算法参数

- 最大句子长度：BERT 要求固定的输入长度，因此不足这一长度的句子将被补齐，超过的将被截断。
- 最大问题长度：BERT 的输入为问题和证据相连后的句子，因此对问题长度做了限制，超过这一长度的问题将被截断。
- 批处理大小：即训练的 batch_size。
- 训练 epoch 数：训练数据的训练次数。
- 学习率：即 learning_rate。
- 存储 checkpoint 步数：即存储 checkpoint 的频率。

模型 IO 参数

- 预测数据输入：json 文件，格式与训练数据保持一致，文件中可不含 answer 标签。
- 预测结果输出：输出预测结果的 csv 文件的路径。csv 文件中每行为一个问题-证据对及其输出结果，每行中第一列为问题，第二列为证据，第三列为标注的答案，第四列为预测的答案。

实例生成

1. 使用数据节点上传数据，数据格式请参考【训练数据】部分。
2. 将训练数据连接到 BERT 的第一个桩点，单击【运行】开始训练。

文本相似度计算

算法说明

文本相似度计算是一个二分类问题， 模型的输入为一对句子，输出为0或1。0代表两个句子不相似，1代表相似。算法采用 Bag of Words 的思想， 通过对词向量的计算获得两个句子的向量表示，计算句子之间的余弦距离，并与通过训练数据获得的最优阈值比较确定标签。其本质为无监督学习模型。

算法 IO 参数

- 训练数据：文本文件，其中每一行是一对句子及其各个词语的对应标签，句子和句子之间有特定的分隔符分隔（默认为__sentence__），句子与标签之间由特定分隔符分隔（默认为 __label__）。句子必须预先分词，各个词语之间用空格分隔。
- 预训练词向量：预训练词向量文件，其中每一行为词及其对应的词向量。

算法参数

- 句子分隔符：用于分割两个句子的分隔符。
- 标签分隔符：用于分割句子和标签的分隔符。
- 词向量维度：确定预训练词向量的维度，与预训练词向量的维度一致。
- 阈值下界：计算最优阈值时所考虑的下界。
- 阈值上界：计算最优阈值时所考虑的上界。
- 阈值更新步长：计算最优阈值时更新步长。

模型 IO 参数

- 预测数据：文本文件，格式与训练数据保持一致，可不包含标签信息。
- 预测结果输出：输出预测结果的 csv 文件的路径。csv 文件中每行为一个句子对及其输出结果，每行中第一列为待判断的句子对。第二列为句子关系的真实标签（相似为1，不相似为0），第三列为句子关系的预测标签。

模型参数

- 预测数据是否标注： 如果为 True，模型输出文件中会包含数据原标签，可用于后续评估。

- 句子分隔符：用于分割句子和标签的分隔符。
- 标签分隔符：用于分割句子和标签的分隔符。如果预测数据无标注信息，用户可忽略此参数。

实例生成

1. 使用数据节点上传数据，数据格式请参考【训练数据】部分。
2. （可选）如数据没有做中文分词及去关键词处理，则需要拉取中文分词及去关键词算子进行数据预处理。
3. 将训练数据连接到算子的输入桩，单击【运行】开始训练。

BERT 分类模型蒸馏

算法说明

BERT 分类模型准确率较高，但参数量较多，模型较大，推理速度也较低。BERT 分类模型蒸馏算法能够用小型的网络从微调过的 BERT 文本分类模型中学习信息。算法使用双向 LSTM 网络 论文链接 拟合微调过的 BERT 分类模型。

参数设置

算法 IO 参数

- 蒸馏数据：每一行为一个句子，词与词之间用空格分隔，句子和标签之间用特定分隔符分隔（分隔符在算法参数中可以设置）。一般为 BERT 文本分类模型使用的训练集。
- BERT文本分类模型目录：在腾讯云 TI 平台上训练的 BERT 文本分类模型的模型目录。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- 词向量文件路径：用于初始化 LSTM 网络的词向量文件路径，可以为空。
- 验证数据：用于评估蒸馏效果的数据目录，格式与蒸馏数据相同。

算法参数

- 蒸馏训练次数：蒸馏过程遍历蒸馏数据的次数。
- 学习率：即 learning_rate。
- 批处理大小：即训练的 batch_size。
- 标签分隔符：一行中分隔句子和标签的分隔符。如果为\t，请填写 tab。
- 词向量维度：网络中词向量的维度。
- LSTM 维度：网络中句子的向量表示的维度。

模型 IO 参数

- 预测数据：要预测的数据路径，如果有标注，格式与蒸馏数据相同。如果无标注，需要一行为一个句子，单词/词语之间用空格分隔。

- 预测结果输出路径：存放预测结果的 csv 文件的输出路径。csv 文件中第一列为句子，第二列为正确标签（如果数据有标注，否则为空），第三列为预测结果。

模型参数

- 标签分隔符：一行中分隔句子和标签的分隔符。如果为\t，请填写 tab。

实例生成

1. 参照【BERT 文本分类算法】文档，训练一个BERT 文本分类模型。
2. 将【BERT 文本分类算法】的训练集连接到【BERT 分类模型蒸馏】的【蒸馏数据】输入桩，将前者的【模型目录】连接到后者的【BERT 本分类模型目录】输入桩，将验证数据连接到后者的【验证数据】输入桩。
3. 填写其它参数，配置相应资源，开始蒸馏，蒸馏完成后，可以在日志中查看蒸馏效果。

中文词性标注

中文词性标注算子可以对输入的中文句子进行词性标注，并输出标注后的结果。

算法 IO 参数

- 输入数据：要进行词性标注的文本文件路径，每行是一个句子。
- 输出数据：输出标注结果文件的路径，每行是输入数据中对应行的句子及其词性标注结果。

算法参数

分隔符：在输出数据中，分隔句子和标签的分隔符。

实例生成

1. 按照输入数据格式，上传一个要标注的文本文件。
2. 将文本文件连接到【中文词性标注】算子的输入桩，或者在【输入数据】中填写文本文件的路径。
3. 配置输出数据和分隔符，运行算子进行预测。

TF-IDF 句子向量

TF-IDF 句子向量算子可以根据 TF-IDF 值，将输入的句子转换为向量表示。

算法 IO 参数

- 输入数据：要进行词性标注的文本文件路径，每行是一个句子。需要预先进行分词。
- 输出文件路径：输出句子向量文件的路径。每行是输入数据中对应句子的向量表示。

实例生成

1. 按照输入数据格式，上传一个文本文件。

2. 将文本文件连接到【TF-IDF 句子向量】算子的输入桩，或者在【输入数据】中填写文本文件的路径。
3. 配置输出文件路径，运行算子。

依存句法分析

依存句法分析分析对句子中词语之间的依存关系进行分析，并以此来表示句子的句法结构。【依存句法分析】算子可以根据用户上传的数据，训练一个自定义的句法分析模型。

算法 IO 参数

- 训练数据：文本文件，必须为CoNLL格式。
- 验证数据：文本文件，必须为CoNLL格式。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

- 训练次数：训练时遍历训练集的次数。
- 学习率：训练时的学习率。
- 是否使用预训练词向量：训练时是否使用预训练的词向量。
- 词向量维度：模型中词向量的维度。如果使用预训练的词向量，预训练词向量文件的词向量维度必须与该参数的取值匹配。
- 词性标注向量维度：模型中词性标注向量的维度。
- LSTM 输出维度：模型中 LSTM 产生的向量的维度。
- batch size：训练时的 batch size。
- 预训练词向量路径：如果使用预训练词向量，此处填写预训练词向量的路径。预训练词向量格式为Glove格式。

模型 IO 参数

- 预测数据：文本文件，必须为CoNLL格式。
- 预测结果输出：包含预测结果的 csv 文件的路径。csv 文件中每行为一个句子及其预测结果，第一列为被预测的句子，第二列为句子的标注，第三列为句子的预测结果。

模型参数

batch size：预测时的 batch size。

实例生成

1. 按照格式，上传训练、验证和预测数据。
2. 将训练和验证数据连接至【依存句法分析】算子的输入桩，配置模型目录和其它算法参数后，开始训练。
3. 训练成功后，将预测数据连接至【依存句法分析】算子的小尾巴，配置预测结果输出和 batch size 后，开始预测。预测完成后，预测结果将被写入指定的路径中。

计算机视觉

最近更新时间：2021-12-24 15:24:59

R-FCN 目标检测

分类需要特征具有平移不变性，检测则要求对目标的平移做出准确响应。现在的大部分 CNN 在分类上可以做的很好，但用在检测上效果不佳。SPP、Faster R-CNN 类的方法在 ROI pooling 前都是卷积，是具备平移不变性的，但一旦插入 ROI pooling 后，后面的网络结构就不再具备平移不变性了。因此，RFCN 提出来的 Position Sensitive Score Map 可以将目标的位置信息融合进 ROI pooling。

算法 IO 参数

- 配置文件：配置模型具体结构的文件，可参考 配置文件。
- 训练数据输入：训练数据，tfrecord 形式，可由【分类检测图像转换】算子生成。
- 验证数据输入：验证数据，tfrecord 形式，可由【分类检测图像转换】算子生成。
- 标签类别映射：表明类别名和 ID 对应关系的文件，可由【分类检测图像转换】算子生成。
- 初始模型：选填，用于 finetune 的预训练模型。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

训练步数：训练的总步数。

模型 IO 参数

- 预测数据输入：一个包含若干个jpg格式图片的目录。只有jpg格式的图片会被预测。
- 预测结果输出目录：存放预测结果的输出目录。

模型参数

- 是否展示可视化输出：如果设为【是】，则预测结果目录中会产生可视化的预测结果（增加了预测框的图片）与json格式的预测结果，json的格式为：每张图片对应一个json文件，json文件中包括以下字段：
  - boxes：预测框的坐标，顺序为[top, left, bottom, right]。
  - scores：每个预测框的分数。
  - class_idx：每个预测框对应的类别id。
  - class_names：每个预测框对应的类别名称。

如果设为【否】，则预测结果目录中只会产生 json 格式的预测结果。

如果生成可视化的预测结果，可能需要多耗费一些时间。

SSD 目标检测

SSD 算法是一种直接预测 bounding box 的坐标和类别的 object detection 算法，没有生成 proposal 的过程，即一阶段算法。SSD 主要特点在于采用了特征融合，详情请参见下文算法示例。

算法 IO 参数

- 配置文件：配置模型具体结构的文件，可参考 配置文件。
- 训练数据输入：训练数据，tfrecord 形式，可由【分类检测图像转换】算子生成。
- 验证数据输入：验证数据，tfrecord 形式，可由【分类检测图像转换】算子生成。
- 标签类别映射：表明类别名和 ID 对应关系的文件，可由【分类检测图像转换】算子生成。
- 初始模型：选填，用于 finetune 的预训练模型。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

训练步数：训练的总步数。

模型 IO 参数

- 预测数据输入：一个包含若干个jpg格式图片的目录。只有jpg格式的图片会被预测。
- 预测结果输出目录：存放预测结果的输出目录。

模型参数

- 是否展示可视化输出：如果设为【是】，则预测结果目录中会产生可视化的预测结果（增加了预测框的图片）与json格式的预测结果，json的格式为：每张图片对应一个json文件，json文件中包括以下字段：
  - boxes：预测框的坐标，顺序为[top, left, bottom, right]。
  - scores：每个预测框的分数。
  - class_idx：每个预测框对应的类别id。
  - class_names：每个预测框对应的类别名称。

如果设为【否】，则预测结果目录中只会产生 json 格式的预测结果。

如果生成可视化的预测结果，可能需要多耗费一些时间。

Faster RCNN 目标检测

Faster RCNN 是继 RCNN 和 Fast RCCN 之后的改进网络。Faster RCNN 是两阶段目标检测算法。第一阶段：提出区域建议网络 RPN，快速生成候选区域。第二阶段：通过交替训练，使 RPN 和 Fast-RCNN 网络共享参数。详情请参见下文的算法示例。

算法 IO 参数

- 配置文件：配置模型具体结构的文件，可参考 配置文件。
- 训练数据输入：训练数据，tfrecord 形式，可由【分类检测图像转换】转换算子生成。
- 验证数据输入：验证数据，tfrecord 形式，可由【分类检测图像转换】转换算子生成。
- 标签类别映射：表明类别名和 ID 对应关系的文件，可由【分类检测图像转换】算子生成。
- 初始模型：选填，用于 finetune 的预训练模型。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

算法参数

训练步数：训练的总步数。

模型 IO 参数

- 预测数据输入：一个包含若干个jpg格式图片的目录。只有jpg格式的图片会被预测。
- 预测结果输出目录：存放预测结果的输出目录。

模型参数

- 是否展示可视化输出：如果设为【是】，则预测结果目录中会产生可视化的预测结果（增加了预测框的图片）与json格式的预测结果，json的格式为：每张图片对应一个json文件，json文件中包括以下字段：
  - boxes：预测框的坐标，顺序为[top, left, bottom, right]。
  - scores：每个预测框的分数。
  - class_idx：每个预测框对应的类别id。
  - class_names：每个预测框对应的类别名称。

如果设为【否】，则预测结果目录中只会产生 json 格式的预测结果。

如果生成可视化的预测结果，可能需要多耗费一些时间。

Inception 图片分类

从原理上来说更深和更宽的卷积神经网络具有更强的表征能力，但纯粹增大网络的缺点也显而易见，包括参数太多带来的复杂度太大，容易过拟合和梯度弥散等。Inception 的出现就是为了解决如何在增加网络深度和宽度的同时减小网络参数的问题。最开始的 inception 网络由一个个的 Inception 模块堆叠而成，inception 模块由1 x 1、3 x 3、5 x 5的卷积和3 x 3的 pooling 堆叠，一方面增加了网络的 width，另一方面也增加了网络对尺度的适应性。Inception 网络经过多次迭代和更新已经形成了四个版本，分别为 V1、V2、V3 和 V4。在 TI 的 Inception 模块中可以选择不同版本进行训练。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 Inception 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

ResNet 图片分类

ResNet（残差网络）是在2015年为解决训练很深的网络的时候出现的梯度退化的问题而提出的。由于使用了 shortcut，ResNet 把原来需要学习逼近的未知函数 H（x）进行恒等映射，变成了逼近 F（x）= H（x）- x 的一个函数，两种表达的效果相同，但 F（x）的优化难度却比 H（x）小的多。同时针对较深（层数大于等于50）的网络提出了 BottleNeck 结构，可以减少运算的时间复杂度。腾讯云 TI 平台 TI-ONE 上集成的 resnet 模块有8种网络模式供选择，v1 和 v2 在 shortcut 结构体构建方式上有区别，50、100等数字表示不同的网络深度，具体可参见 resnet 论文。详情请参见下文算法示例。

算法 IO 参数

- 训练数据输入：用于训练的tfrecord数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。

- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 ResNet 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

VGG 图片分类

VGG16 继承了 alexnet 的框架，但其通过加深卷积层数的方法获得了比 AlexNet 更高的图像识别率，是一种比 AlexNet 更深的网络结构。
AlexNet 与 VGG 相比有以下不同：

- 有较多的连续的 convolution 块和较小的 filter size（3 x 3），这样做的好处是减少权重个数，利于网络的训练和泛化。
- 通道数（channels）增多，使网络可以对输入提取更丰富的特征。VGG 论文发表时根据网络结构的复杂度提出了三种结构：VGG_a、VGG_16 和 VGG_19，您可以根据需要进行选择。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。

- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 VGG 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

MobileNet 图片分类

MobileNet V2 是一种小巧而高效的 CNN 模型，它的基本单元是深度级可分离卷积（depthwise separable convolution）。MobileNet 训练精度高，推理速度快。可参见 论文文档。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 MobileNet 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

SqueezeNet 图片分类

SqueezeNet 是 Han 等提出的一种轻量且高效的 CNN 模型，它参数比 AlexNet 少50倍，但模型性能（accuracy）与 AlexNet 接近。在可接受的性能下，小模型相比大模型，具有很多优势：
（1）更高效的分布式训练，小模型参数小，网络通信量减少。
（2）便于模型更新，模型小，客户端程序容易更新。
（3）利于部署在特定硬件如 FPGA，因为其内存受限。可参见 论文文档。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 SqueezeNet 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。

- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

ShuffleNet 图片分类

ShuffleNet 是 Face++ 提出的降低深度网络计算量的网络。它利用两个操作：逐点群卷积（pointwise group convolution）和通道混洗（channel shuffle），与现有先进模型相比在类似的精度下大大降低计算量。可参见论文文档。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 模型名称：要使用的 ShuffleNet 版本。
- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

NasNet 图片分类

NasNet 是 google 使用自动机器学习技术搜索出来的图像分类算法。可参见 论文文档。

算法 IO 参数

- 训练数据输入：用于训练的 tfrecord 数据，可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。
- 验证数据输入：用于模型训练时的验证数据，与训练集格式一致。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- label_map 文件所在目录：记录标签和 ID 对应关系的 label_map.txt 文件所在目录。可以使用【输入】>【数据转换】>【分类检测图像转换】算子生成。

算法参数

- 学习率：训练过程的初始学习率。
- batch_size：训练过程中的 batch_size。
- 训练步数：训练过程的总迭代次数。
- 是否模型微调：支持在已有的训练好的模型上进行微调（通常用于迁移学习），如选“是”请在下方填写完整模型路径，并选择是否仅训练全连接层 。
- 优化器：用于优化模型参数的优化算法。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下；如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列是标注的类别，第三列是预测的类别。

模型参数

batch_size：预测时的 batch size。

模型量化压缩

图像分类任务的模型由于其参数量大，模型占用空间大，在 CPU 以及边缘设备上的推理速度较慢。模型量化压缩针对训练完成后的图像分类模型，将模型的权重以及算子的运算（一般为 float32 类型）转换为更紧凑的格式存储（int8 或者 float16）。在保持精度下降不多的条件下，压缩模型的存储所需空间，并且在 CPU 以及边缘设备上的推理速度加快。

算法 IO 参数

- 输入模型目录：待量化的模型目录，对应于图像分类算法的模型目录。（如果目录中有若干个模型文件，算法将加载目录中最新的模型文件。）
- 模型目录：存放量化后的模型文件和日志文件的目录（如果改变了网络结构或更换了数据，请更换或清空模型目录）。
- 典型数据集输入：选填。典型数据集输入与图像分类算法的训练输入格式一致，tfrecord 形式，可由分类检测图像转换算子生成。如果在参数量化数据类型中选择了 Int8，并且在是否使用典型数据集中选择了是的条件下，该数据集才会生效。

算法参数

- 量化数据类型：可选值为 Int8 或 Float16，量化压缩后模型存储的数据类型。Int8 的选项关联了更多参数，包括：是否使用典型数据集、典型数据集输入以及典型数据集选用的数量。
- 是否使用典型数据集：是否使用典型数据集对于 Int8 量化算法进行优化。量化数据类型参数为 Int8 时有效。
- 典型数据集选用的数量：整数，典型数据集使用的样本数量，填-1表示使用全部数据。是否使用典型数据集参数为是时有效。

模型 IO 参数

- 预测数据输入：包含要预测的图片的目录。如果要预测的图片有标签，则将对应类别的图片放在以标签名命名的目录下，如果要预测的图片无标签，则将所有要预测的图片全部放在该目录下。
- 预测结果输出路径：输出预测结果的 csv 文件的目录。csv 文件中，每行为一张图片的预测结果，每行中，第一列为图片的路径，第二列为图片的预测结果，第三列为图片的真实标签。

Deeplab 图像分割

Deeplab 是一种常用而高效的图像分割算法，能够对图像进行语义分割。

算法 IO 参数

- 训练数据目录：存放训练用的tfrecord文件的目录，tfrecord文件由【图像分割数据转换】算子生成。
- 验证数据目录：存放验证用的tfrecord文件的目录，tfrecord文件由【图像分割数据转换】算子生成。
- 模型目录：存放模型文件和日志文件的目录（如果目录中已有模型文件，下次训练时将加载目录中最新的模型文件。因此，如果改变了网络结构或更换了数据，请更换或清空模型目录）。

- 预训练模型路径：存放预训练的 deeplab 模型的路径，基于Mobilenet_V2的预训练模型可以从 此处 下载，基于Xception_65的预训练模型可以从 此处 下载。使用预训练模型时，将预训练模型解压后上传至cos，并填写到model.ckpt的文件路径，如${cos}/checkpoints/deeplab/xception_65/model.ckpt。

算法参数

- 模型类别：可以选择xception_65或者mobilenet_v2。如果使用了预训练模型，模型类别的选择需要和预训练模型相匹配。
- 类别数：模型要预测的类别数（含背景类，如VOC数据集应该填写21）。
- 学习率：模型训练时的学习率。
- 训练步数：模型的训练步数。
- 批处理大小：训练时每张GPU上的batch size。
- 是否加载最后一层：加载预训练模型时，是否加载模型的最后一层（分类层），在自定义数据集上，应设为【否】。
- 训练时裁剪尺寸：训练时，对图像进行裁剪的尺寸。长和宽用英文逗号分隔，需要分别大于图像的最大长和宽，且为16的倍数+1。如图像最大尺寸为500像素，则取值为16*32+1=513。
- 验证时裁剪尺寸：验证时，对图像进行裁剪的尺寸。长和宽用英文逗号分隔，需要分别大于图像的最大长和宽，且为16的倍数+1。如图像最大尺寸为500像素，则取值为16*32+1=513。

模型 IO 参数

- 预测数据目录：存放测试数据的目录，目录中应为要预测的图片文件，jpg格式。
- 预测结果输出目录：输出预测结果的目录，预测结果为png格式的灰度图，图片文件名与预测数据一一对应，每个像素点的取值为类别id。

模型参数

- 预测时输入尺寸：需要为验证时裁剪尺寸的较大值。
- 是否展示可视化输出：如果设为【是】，则在灰度图的预测结果之外，另外生成可视化的预测结果（将分割结果叠加到原图上）。

拉普拉斯金字塔

拉普拉斯金字塔是一种图像金字塔算法，能够进行图像的多尺度变换。

算法 IO 参数

- 图像目录：包含若干jpg格式图片的目录，或者包含若干个前述格式的子目录的目录。
- 输出目录：输出金字塔变换后的图片的目录。

算法参数

- 上采样层数：金字塔算法中，上采样的层数。

- 下采样层数：金字塔算法中，下采样的层数。

高斯金字塔

高斯金字塔是一种图像金字塔算法，能够进行图像的多尺度变换。

算法 IO 参数

- 图像目录：包含若干jpg格式图片的目录，或者包含若干个前述格式的子目录的目录。
- 输出目录：输出金字塔变换后的图片的目录。

算法参数

- 上采样层数：金字塔算法中，上采样的层数。
- 下采样层数：金字塔算法中，下采样的层数。

I3D 视频分类

I3D 是基于深度学习的视频分类算法。通过使用 Inflated 3D 卷积，应对视频分类领域数据集缺乏，避免之前只能从头在小数据上训练的□境，利用 I3D 将在 Imagenet 上训练成功的经典模型迁移学习到 video 数据集上。目前仅支持 one stream RBG 模式。

算法IO参数

- 保存训练 checkpoint 路径：存放模型训练结果的路径。
- 验证集效果最佳模型保存路径：存放最佳模型的路径。
- 训练集和测试集保存路径：用于训练和测试的数据路径，一般使用【 视频分类数据转换 】算子产生。

算法参数

- GPU 数量：训练时使用的 GPU 数量，需要和资源参数匹配。
- 保存 checkpoint 步长：保存 checkpoint 文件的步长间隔。
- 保存summary步长：保存 summary 文件的步长间隔。
- 训练步长：模型的训练步数。
- 优化器：模型使用的优化器。
- 类别数：要预测的类别数。
- 视频片段参与训练的帧数量：每个视频片段参与训练的帧数。
- 批大小：训练时的 batch size。
- 初始化学习率：训练时的初始学习率。
- 初始权重文件路径：使用的预训练模型路径。

模型 IO 参数

- 预测集原始视频路径：要预测的视频文件的保存路径。

- 预测结果保存路径：保存输出的预测结果的路径。

模型参数

视频片段参与预测的帧数量：每个视频片段预测时，参与预测的帧数。

SiamFC 单目标视频追踪

SiamFC 是基于深度学习的单目标追踪算法。算法通过共享的参数同时提取模板特征和目标特征后，通过 cross-correlation 进行相似度度量，计算每个位置的相似度得到 score map，最后计算追踪目标位置。

算法 IO 参数

- 初始权重文件路径：训练时使用的初始权重文件路径。
- 训练集和测试集信息保存路径：训练集和测试集经过切分后，保存的信息的路径。
- 保存训练 checkpoint 路径：存放模型训练结果的路径。
- 验证集效果最佳模型保存路径：存放最佳模型的路径。

算法参数

- 保存 summary 步长：保存 summary 文件的步长间隔。
- 原始图片文件格式：按实际情况填写jpg或png。
- 原始数据标签文件格式：填写xml。
- GPU 数量：训练时使用的 GPU 数量，需要和资源参数匹配。
- 批大小：训练时的 batch size。
- 学习率：训练时的学习率。
- 优化器：可以填写MOMENTUM或SGD。
- 训练步长：训练的总步数。
- 保存 checkpoint 步长：保存 checkpoint 文件的步长间隔。

模型 IO 参数

- 预测集原始视频路径：要预测的视频文件的保存路径。
- 预测结果保存路径：保存输出的预测结果的路径。

模型参数

- 预测集是否包括标签：1代表有标签，0代表无标签。
- 原始图片格式：按实际情况填写jpg或png。
- 预测图片标注格式：填写xml。
- 是否保存预测的 boudingbox 结果：1代表保存，0代表不保存。
- 是否保存预测结果图：1代表保存，0代表不保存。

输出
可视化

最近更新时间：2020-06-30 15:17:28

可视化图主要分为 Relationship（关系）、Distribution（分布）、Comparison（对比）、Composition（组合）四类。

关系

气泡图

输入

- 格式：| 横坐标 | 纵坐标 | 颜色 | 气泡大小 |
- 说明：以空格连接各字段，通过**参数**中的横坐标、纵坐标、颜色、气泡大小所在列分别指定。不指定气泡大小列，气泡图就会变成散点图。

  参数

- 横坐标列：从0开始计数。
- 纵坐标列：从0开始计数。
- 颜色列：从0开始计数；若无该列，不填。
- 气泡大小所对应的列：从0开始计数；若无该列，不填。

输出

鼠标悬停即可查看结果，显示如下：
![](<https: main.qcloudimg.com="" raw="" 7d9d5114f8524293072d156242a04fbd.png="">)

分布

箱线图

指定 beginCol 到 endCol 之间的多个列，将绘制出这多个列的箱线图。

输入

- 数据形式：Dense，以空格连接各字段。

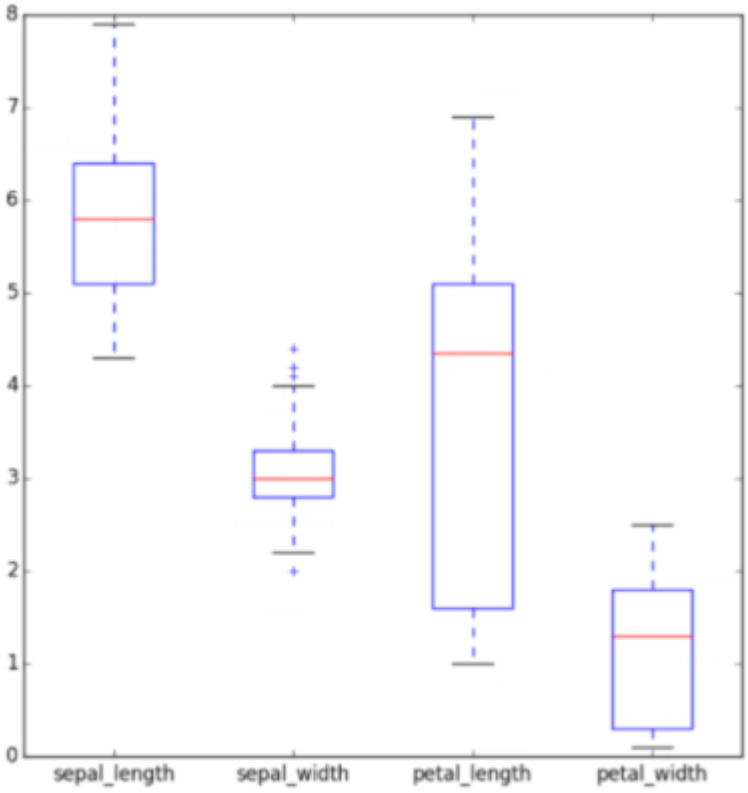
- 格式：| 参与分布的 features | 不参与分布的 features |
  - 参与分布的 features：通过**参数**中的**起始列和终止列**进行指定。
  - 不参与分布的 features：可包括不参与分布的特征。

参数

- 起始列：从0开始计数。
- 终止列：从0开始计数，包括该列。

输出

鼠标悬停即可查看结果，显示如下：



直方图

指定输入数据表的某一列，模块就会统计每个元素出现的频次，并画直方图。

输入

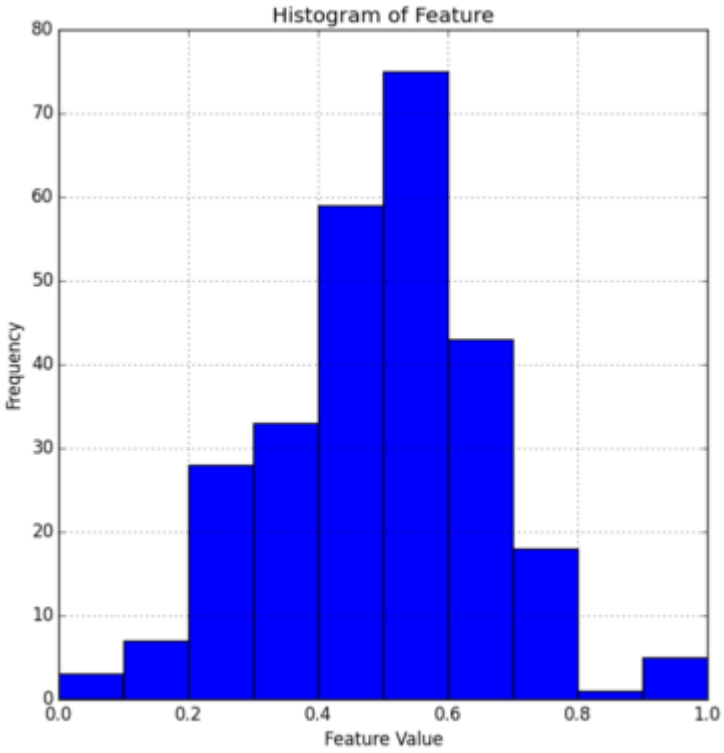
- 数据形式：Dense，以空格连接各字段。
- 格式：| features|
- 说明：features 中包含了需要统计的列。

参数

- 字段号：要统计的字段号，从0开始计数。

输出

鼠标悬停即可查看结果，显示如下：



散点图

输入

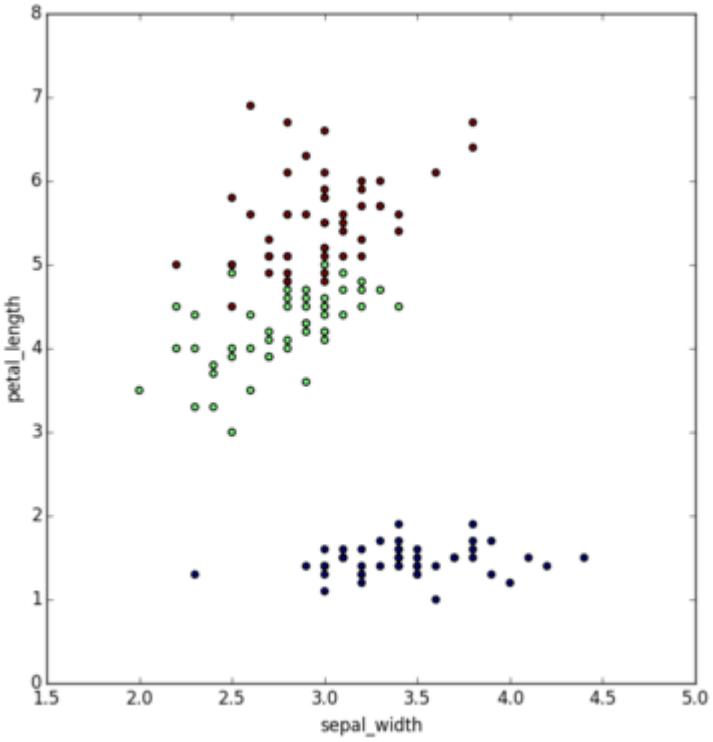
- 数据形式：Dense，以空格连接各字段。
- 格式：| 横坐标 | 纵坐标 | 颜色 |
- 说明：通过**参数**中的横坐标、纵坐标、颜色所在列分别指定。

参数

- 横坐标列：从0开始计数。
- 纵坐标列：从0开始计数。
- 颜色列：从0开始计数，可不填。

输出

鼠标悬停即可查看结果，显示如下：



对比

柱状图

指定横坐标列，然后指定 beginCol 到 endCol 之间的多个列，将会绘制出这多个列的对比。

输入

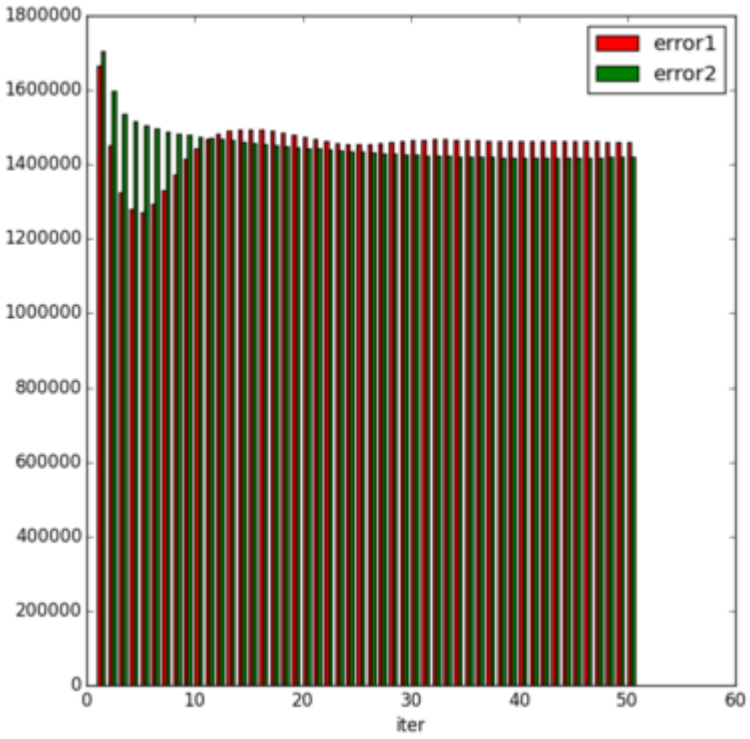
- 数据形式：Dense，以空格连接各字段。
- 格式：| 横坐标 | 参与对比的 features | 不参与对比的 features |
  - 参与对比的 features：通过**参数**中的**起始列和终止列**进行指定。
  - 不参与对比的 features：可包括不参与对比的特征。

参数

- 横坐标列：从0开始计数。
- 起始列：从0开始计数。
- 终止列：从0开始计数。

输出

鼠标悬停即可查看结果，显示如下：



折线图

绘制多个列的折线图对比。

输入

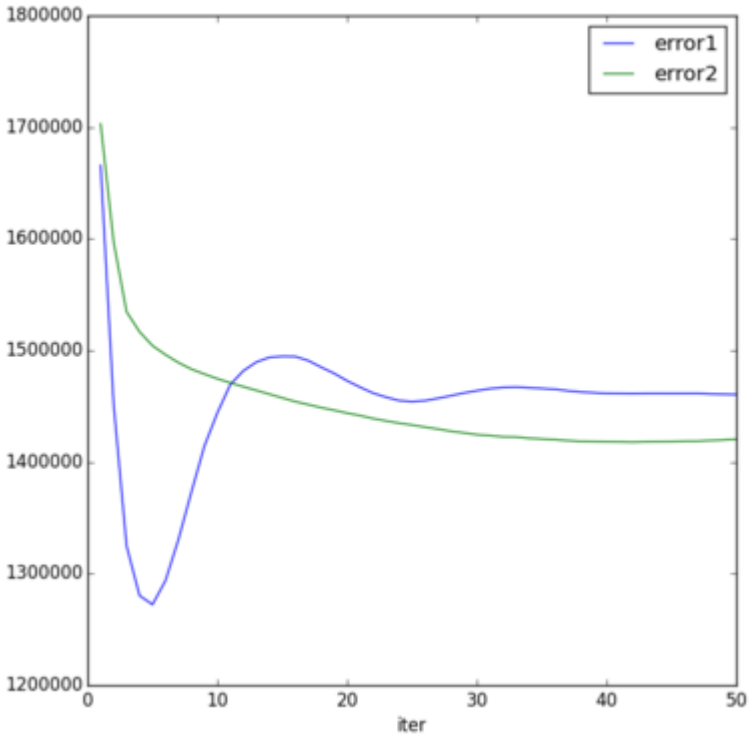
- 数据形式：Dense，以空格连接各字段。
- 格式：| 横坐标 | 参与对比的 features | 不参与对比的 features |
  - 参与对比的 features：通过**参数**中的**起始列和终止列**进行指定。
  - 不参与对比的 features：可包括不参与对比的特征。

参数

- 横坐标列：从0开始计数。
- 起始列：从0开始计数。
- 终止列：从0开始计数。

输出

鼠标悬停即可查看结果，显示如下：



画矩阵

绘制矩阵，矩阵中的内容必须为数值（整数或浮点型）。矩阵中的颜色由对应的数值以及右边的色条决定。该色条的取值范围为矩阵中数值的范围，即矩阵中的最大值和最小值分别对应色条中的最上边（深红）和最下边（深蓝）。

输入

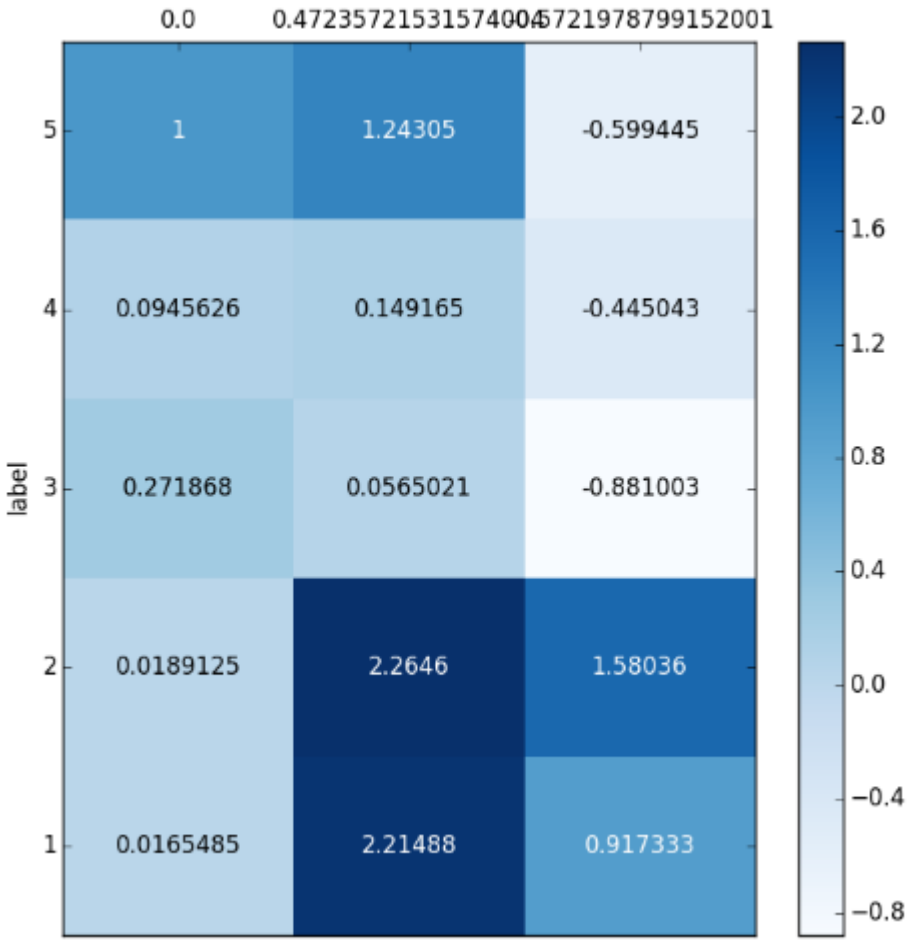
- 数据形式：Dense，以空格连接各字段。
- 格式：| 带横纵坐标的 matrix |
- 说明：矩阵的第一列作为绘制的纵坐标，表示矩阵的 label，矩阵的第一行作为绘制的横坐标。

参数

矩阵中是否显示注：是否显示矩阵中的数值，默认不显示，该注释内容如果过长将影响显示效果。

输出

鼠标悬停即可查看结果，显示如下：



组合

饼状图

类似于直方图，指定输入数据表的某一列，模块就会统计每个元素出现的频次，并绘制饼图。

输入

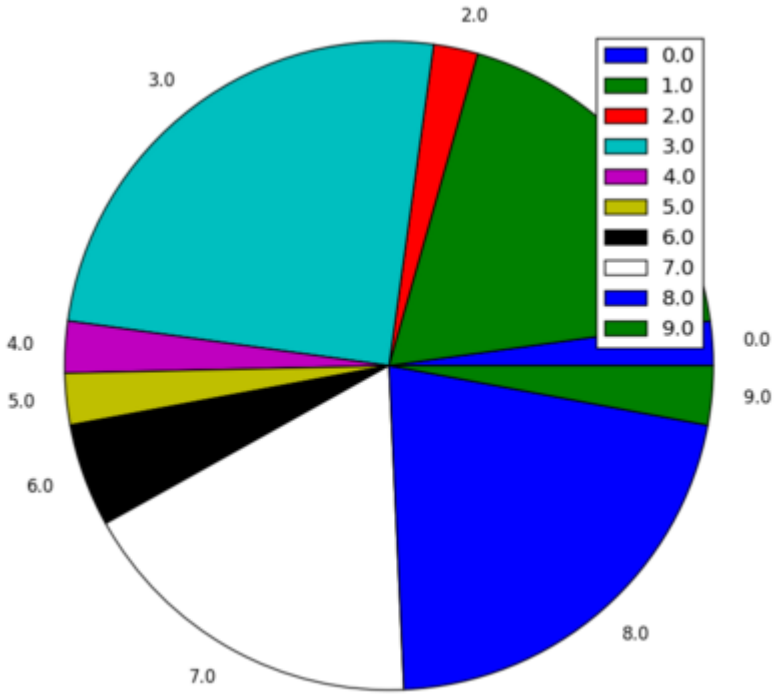
- 格式：| features|
- 说明：features 中包含了需要统计的列。

参数

字段号：要统计的字段号，从0开始计数。

输出

鼠标悬停即可查看结果，显示如下：



瀑布图

指定横坐标列和纵坐标列，绘制累积的趋势图。

输入

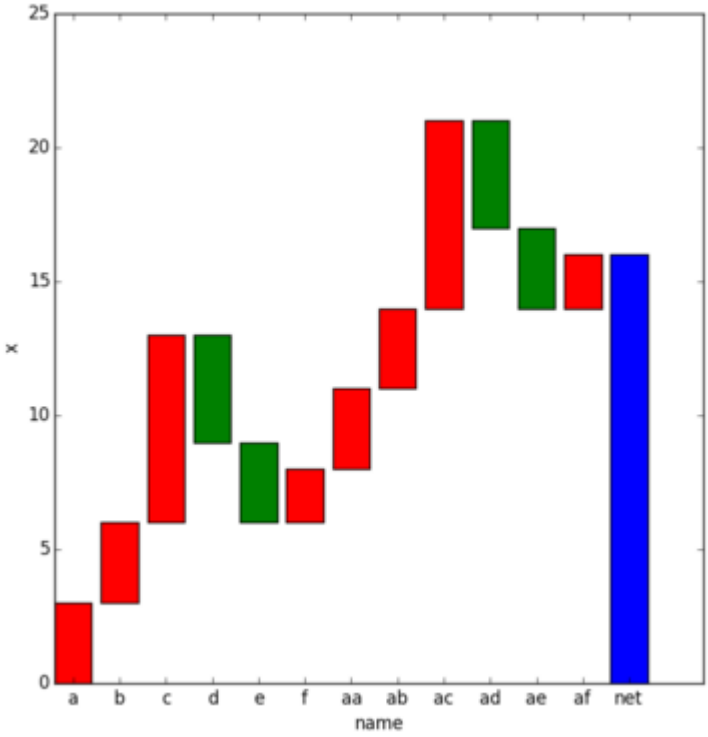
- 格式：| features|
- 说明：features 中包含了需要统计的列。

参数

- 横坐标：从0开始计数。
- 纵坐列：从0开始计数。

输出

鼠标悬停即可查看结果，显示如下：



模型评估

最近更新时间：2021-12-24 15:22:03

腾讯云 TI 平台 TI-ONE 内置多种模型评估可视化组件，您可快速辨别模型的质量，并对该模型进行优化。您只需拖拽组件即可轻松使用本服务。

二分类任务评估

用于评估二分类算法的预测结果。预测的结果必须是0 – 1的概率值，评估结果包括混淆矩阵和 precision，recall，f1等指标。

真实情况	预测结果	
	正例	反例
正例	TP（真正例）	FN（假反例）
反例	FP（假正例）	TN（真反例）

- P（precision）= TP / (TP + FP)
- R（recall）= TP / (TP + FN)
- F1 = 2 * P * R / (P + R)

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0~1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

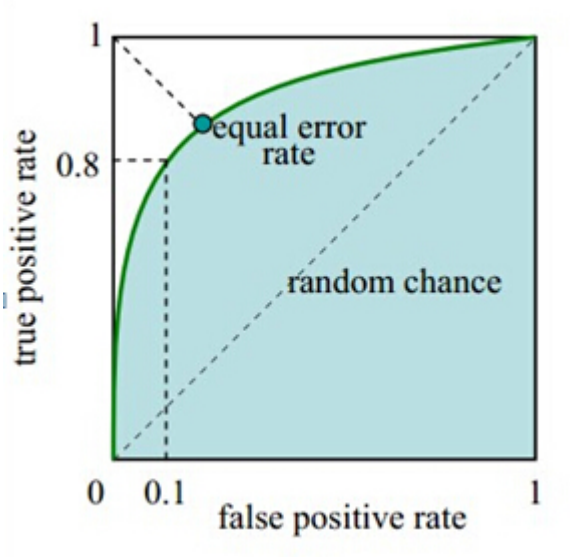
- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值
  - 范围：0 – 1的值，预测值大于该阈值，则预测为1；否则为0。
  - 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

precision，recall，f1 值。

ROC 曲线

ROC（receiver operating characteristic curve），将阈值不同取值下对应的 TPR 和 FPR 结果画于二维坐标系中得到的曲线，就是 ROC 曲线
下图所展示的是 ROC 曲线与 AUC：



- 对角线对应于“随机猜测”的模型。
- 点（0,1）对应于将所有正例排在所有反例之前的“理想模型”。
- AUC 对应于 ROC 曲线下的面积，AUC 越大，则模型效果越好。
- P（precision）= TP / (TP + FP)
- R（recall）= TP / (TP + FN)

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0 - 1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值
  - 范围：0 - 1的值，预测值大于该阈值，则预测为1；否则为0。
  - 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

ROC AUC 值。

PR 曲线

- P（precision）= TP / (TP + FP)
- R（recall）= TP / (TP + FN)

其中，横轴是 precision，纵轴是 recall。

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0~1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值
  - 范围：0 - 1的值，预测值大于该阈值，则预测为1；否则为0。
  - 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

precision recall 值。

KS 曲线

KS（Kolmogorov-Smirnov）检验：K - S 检验主要是验证模型对违约对象的区分能力，通常是在模型预测全体样本的信用评分后，将全体样本按违约与非违约分为两部分，然后用KS统计量来检验这两组样本信用评分的分布是否有显著差异。

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0 - 1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值

- 范围：0 - 1的值，预测值大于该阈值，则预测为1；否则为0。
- 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

KS，threshold 值。

Lift 曲线

Lift chart 是不同阈值下 Lift 和 Depth（正例的比例）的轨迹，它衡量的是，与随机猜测相比，模型的预测能力比随机结果“变好”了多少。lift（提升指数）越大，模型的运行效果越好。
Lift=precision/正例的比例。

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0 - 1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值
  - 范围：0 - 1的值，预测值大于该阈值，则预测为1；否则为0。
  - 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

LIFT 值，正例的比例。

Gain 曲线

Gains (增益) 与 Lift（提升）相当类似：Lift chart 是不同阈值下 Lift 和 Depth（正例的比例）的轨迹，Gains chart 是不同阈值下 Precision 和 Depth（正例的比例）的轨迹。

输入

- 数据形式：Dense。
- 格式：| label | score |
- label：样本真实的label，通过**参数**中的**标签列**指定。
- score：打分值，模型预测的0 - 1.0之间的概率值所在的列，通过**参数**中的**打分列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 打分列（scoreCol）：概率值所在列，从0开始计数。
- 预测阈值
  - 范围：0 – 1的值，预测值大于该阈值，则预测为1；否则为0。
  - 说明：在真实的 CTR 预估中，预测的概率值都比较小，因此要视真实情况设定该参数。

输出

Precision值，正例的比例 。

聚类任务评估

用于评估聚类算法的预测结果。输入的是特征列数据与预测列的模型预测值；模型将计算Silhouette（轮廓系数）。轮廓系数（Silhouette Coefficient），结合了内聚度和分离度两种因素，用来评价算法聚类结果。

输入

- 数据形式：Dense。
- 格式：predict。
- predict：预测值，通过参数中的**预测列**指定。

参数

预测列（scoreCol）：预测值所在列，从0开始计数。

输出

Silhouette 数值

多分类任务评估

用于评估多分类算法的预测结果。样本的预测结果是模型预测的类别；模块将会统计类别真实类列和预测类别的混淆矩阵和各个类别的 truePositiveRates，falsePositiveRate，precision，recall，fMeasure，trueNegativeRate，negativePredictiveValue。

输入

- 数据形式：Dense。
- 格式：| label | predict |
- label：样本真实的 label，保存的是样本真实的类别，通过**参数**中的**标签列**指定。
- score：预测值，保存的是模型预测的类别，通过**参数**中的**预测列**指定。

参数

- 标签列（labelCol）：标签所在列，从0开始计数。
- 预测列（predictCol）：预测的类别所在列，从0开始计数。

输出

- True class 和 Hypothesized class 的混淆矩阵。
- 不同类别的 f1、precision、recall。

回归算法评估

用于评估回归算法的预测结果。输入的是真实的值和模型预测值；模型将计算 explainedVariance，meanAbsoluteError，meanSquaredError，rootMeanSquaredError，r2 等指标。

输入

- 数据形式：Dense。
- 格式：| label | predict |
- 说明：以空格连接各字段。
- label：样本真实的label，通过**参数**中的**标签列**指定。
- predict：预测值，通过**参数**中的**预测列**指定。

参数

- 标签列（labelCol）： 标签所在列，从0开始计数。
- 预测列（scoreCol）：预测值所在列，从0开始计数。

输出

MAE、MSE、rMAE、R2 等指标。

序列标注任务评估

将序列标注任务或依存句法分析任务的预测结果文件作为输入，计算评估指标，包括字级别的准确率，单句中序列标注正确率为100%的句子比例，单句中序列标注正确率大于80%的句子比例，以及单句中序列标注正确率大于50%的句子比例。

算法 IO 参数

预测数据： 序列标注任务或依存句法分析任务的预测结果。

算法参数

- 标签所在列：标签在CSV文件中所在列，从0开始计数。

- 预测结果所在列：预测结果在CSV文件中所在列，从0开始计数。

实例生成

将序列标注相关算子的预测结果连接到算子的输入桩，单击【运行】开始计算指标。对于平台内置的序列标注算子和依存句法分析算子，标签列和预测列的序号应分别填写1和2。

深度学习分类任务评估

将分类任务的预测文件作为输入，计算各项评估指标，包括整体 accuracy 和每个类别的 precision，recall 和 F1。

算法 IO 参数

预测数据：文本分类或图片分类任务的预测结果。

算法参数

- 标签所在列：标签在 CSV 文件中所在列，从0开始计数。
- 预测结果所在列：预测结果在 CSV 文件中所在列，从0开始计数。

实例生成

将分类相关算子的预测结果连接到算子的输入桩，单击【运行】开始计算指标。对于平台内置的图片分类和文本分类算子，标签列和预测列应分别填写1和2。

中文问答评估

将机器阅读理解的预测结果文件作为输入，计算各项评估指标，包括整体 accuracy 和 F1。

算法 IO 参数

预测数据：中文阅读理解任务的预测结果。

算法参数

- 标签所在列：标签在 CSV 文件中所在列，从0开始计数。
- 预测结果所在列：预测结果在 CSV 文件中所在列，从0开始计数。

实例生成

实例生成：将阅读理解算子的预测结果连接到算子的输入桩，单击【运行】开始计算指标。对于平台内置的 BERT 中文问答算子，标签列和预测列应分别填写1和2。

Tensorboard

通过 TensorFlow 将程序输出的日志文件作为输入，提供可视化输出，使得 TensorFlow 程序的理解、调试和优化更加简单高效。

算法 IO 参数

Tensorboard 路径：TensorFlow 程序输出的 event file 所在路径。

算法参数

展示时间（分钟）：Tensorboard 实例运行的时长，默认为60分钟。

资源参数

资源类型：Tensorboard 实例运行所需的机型。

医疗板块

最近更新时间：2021-07-26 10:54:57

医学对比指标

用于计算常用医学指标。

对比指标：相对危险度（RR）、比值比（OR）。

- 相对危险度（Relative Risk，RR）= 暴露组的累积发病率（或死亡率）/ 对照组的累计发病率（或死亡率）。相对危险度表明暴露组发病率或死亡率是对照组发病率或死亡率的多少倍。说明暴露组发病或者死亡的危险性是非暴露组的倍数。RR 值越大，表明暴露的效应越大，暴露与结局的关联的强度越大。
- 比值比（Odds Ratio，OR）= 病例组的暴露比值（a/c）/ 对照组的暴露比值（b/d）= ad/bc。在病例对照研究中，比值比指病例组中暴露与非暴露人数的比值和对照组中暴露与非暴露人数的比值的比。在队列研究中，指的是暴露组中患病与非患病者的比值和非暴露组中患病与非患病者的比值的比。

输入参数

输入数据路径：标签和预测结果的 csv 文件的路径。

输出参数

- 指标输出路径：对比指标值输出路径。
- 混淆矩阵输出路径：对比指标混淆矩阵输出路径。

算法参数

- 输入数据是否包含 header 信息：是，否。
- 输入数据分割符：逗号、分号、空格、Tab。
- 组别列（单列）：组别所在列的列号，从0开始计数。
- 类别列（单列）：类别所在列的列号，从0开始计数。

资源参数

资源类型：选择合适的算力机型。

医学评估指标

用于计算常用医学指标。

评估指标：真阳性、假阳性、真阴性、假阴性、敏感性/度、特异性/度、精准率、准确率。

通常来说，阳性（positive）代表患病或者携带病毒，阴性（negative）代表正常样本。

- 真阳性 TP（True Positive）：预测和真实情况均为阳性，预测正确。
- 假阳性 FP（False Positive）：将不具备阳性症状的人检测出阳性，假阳性检测结果导致误诊。
- 真阴性 TN（True Negative）：预测和真实情况均为阴性，预测正确。
- 假阴性 FN（False Negative）：将阳性症状的样本预测为阴性，假阴性结果导致漏诊。

真实情况	预测结果	
阳性	阳性	阴性
	TP（真阳性）	FN（假阴性）
阴性	FP（假阳性）	TN（真阴性）

- 敏感性/度（Sensitivity）= TP /（TP + FN），即患者被预测为阳性的概率。此值越大，说明诊断试验越灵敏。
- 特异性/度（Specificity）= TN /（TN + FP），即实际上未患病样本被诊断为阴性的概率。此值越大，说明诊断试验越精确。
- 精准率（Precision）= TP /（TP + FP），即所有被预测为阳性的样本中实际为阳性的概率。
- 准确率（Accuracy）=（TP + TN）/（TP + TN + FP + FN），即预测正确的结果占总样本的百分比。

输入参数

输入数据路径：标签和预测结果的 csv 文件的路径。

输出参数

- 指标输出路径：评估指标值输出路径。
- 混淆矩阵输出路径：评估指标混淆矩阵输出路径。

算法参数

- 标签列（单列）：标签所在列的列号，从0开始计数。
- 预测值列（单列）：预测值所在列的列号，从0开始计数。
- 输入数据是否包含 header 信息：是，否。
- 输入数据分割符：逗号、分号、空格、Tab。

资源参数：

资源类型：选择合适的算力机型。

图像转换

对常用医学格式图像进行读取，包括 Dicom、nii、nrrd 等可由 simpleitk 可读取的格式，可转为 nii.gz 格式或者切片保存为 tiff 格式，同时支持对数据进行切分，供算法节点调用。

医学图像常用术语：

- 一层：即一张二维图像。
- 序列：即由多层图像所构建的三维图像。
- Dicom 格式文件：医学图像格式，通常一个 dicom 文件为一层，一个序列为一个文件夹。
- Nii、nrrd 等格式：通常一个文件为一个序列。

输入参数

- 图像输入路径：目录中需包含若干个子目录，子目录名字为类别名。
- 标注输入路径：存放图像目录、label、标注目录以及 bounding box 区域（以 ";" 分隔）的 csv 格式文件，列名分别为 image_path、label、mask_path、bounding_box。

输出参数

- 训练集输出路径：输出训练集的文件的路径。
- 验证集输出路径：输出验证集的文件的路径。

算法参数

- 仅截取 ROI 部分：是，否。若选“是”，进一步选择取 bounding box 或者仅 ROI。
- 输出格式：nii.gz、tiff。
- 数据切分：是，否。是否对数据集进行切分，如果选择“否”，所有数据将会转换成目标格式，并输出到【训练集输出】目录下。
- 验证集比例：0-1，如果数据切分中选择“是”，此处为切分成验证集的数据比例。

资源参数

资源类型：选择合适的算力机型。

影像组学特征提取

从医学影像中提取常用影像组学特征，以 pyradiomics 为参考。

输入参数

图像输入路径：经医学图像转换后的医学图像。

输出参数

影像组学特征输出路径：输出包含特征的 csv 文件路径。

算法参数

- 特征参数设置：可以通过配置 pyradiomics 提供的配置文件进行配置，具体可配置参数如下：
  - 基本设置-binWidth (bin 的宽度标量)：float 类型，大于0，用于设定 bin 的宽度标量。
  - 基本设置-差值方法（interpolator）：用于设置重采样的差值方法。包含：枚举类型，sitkNearestNeighbor (= 1)，sitkLinear (= 2)，sitkBSpline (= 3)，sitkGaussian (= 4)，sitkLabelGaussian (= 5)，sitkHammingWindowedSinc (= 6)，sitkCosineWindowedSinc (= 7)，sitkWelchWindowedSinc (= 8)，sitkLanczosWindowedSinc (= 9)，sitkBlackmanWindowedSinc (= 10)。
  - 基本设置-是否重采样：是，否。[公有云上默认值为(1.0,1.0,1.0)]
  - 图像分量：Original、Wavelet、LoG、Square、SquareRoot、Logarithm、Exponential、Gradient、LBP3D。
  - 特征类型：Shape、Firstorder、glcm、glrlm、glszm、gldm。

资源参数

资源类型：选择合适的算力机型。

数据拼接

将多个模型的结果文件拼接到一个 csv 文件中，目的是对该 csv 进行模型结果评估对比，以此来判断哪个模型输出的结果性能更高。

输入参数

- 待拼接输入路径1：待拼接的文件路径1。
- 待拼接输入路径2：待拼接的文件路径2。

输出参数

拼接输出路径：拼接后文件路径（根据输入1和输入2的指定列进行合并拼接）。

算法参数

- 输入文件类型：csv 文件。
- 输入文件包含 header 信息：是，否。
- 输入文件分隔符：逗号、空格、分号、Tab（“\t”）。
- 输入文件1特征列：表示需要拼接的特征所在列，例如“1-10,12,15”，表示取特征在表中的1到10列，12列，15列，从0开始计数。
- 输入文件1模型名（可选）：输入的模型名作为输入文件1列名的前缀，输出至输出文件，用下划线连接，如不填，则不增加前缀（如模型名填写 “model1”，列0名称 “balance”，输出文件列0为 “model1_balance”）。

- 输入文件2特征列：表示需要拼接的特征所在列，例如“1-10,12,15”，表示取特征在表中的1到10列，12列，15列，从0开始计数。
- 输入文件2模型名（可选）：输入的模型名作为输入文件1列名的前缀，输出至输出文件，用下划线连接，如不填，则不增加前缀（如模型名填写 "model1"，列0名称 “balance”，输出文件对应列为“model1_balance”）。

资源参数

资源类型：选择合适的算力机型。

模型校验

计算常用医学单样本和多样本检验，包含独立 T 检验，配对 T 检验、Delong 检验、Pearson 校验、Spearmanr 检验、Kendalltau 校验。

对比不同模型表现的差异性、对比不同数据集分布差异显著性。

- 独立 T 检验：用于分析定类数据（X）与定量数据（Y）之间的差异情况。独立样本T检验需要样本服从正态分布，且两组样本的总体方差相等。
- 配对 T 检验（paired t-test）：用于分析配对定量数据之间的差异对比关系。配对样本T检验要求样本是配对的，两个样本的样本量要相同，且样本先后的顺序是一一对应的。
- Delong 检验：用于检验 AUC 显著性。
- 卡方检验（Chi-Squared Test）：用于比较两个及两个以上样本率（构成比）以及两个分类变量的关联性分析。
- 显著性检验
  - Mann-whitney：假设两个样本分别来自除了总体均值以外完全相同的两个总体，用于检验这两个总体的均值是否有显著的差别。
- 相关系数
  - 皮尔逊相关系数（Pearson）：用于度量两个变量 X 和 Y 之间的相关程度（线性相关）。
  - 斯皮尔曼等级相关系数（Spearmanr Rank）：用来估计两个变量 X、Y 之间的相关性，其中变量间的相关性可以使用单调函数来描述。
  - 肯德尔等级相关系数（Kendalltau Rank）：用来测量两个随机变量相关性的统计值。

输入参数

输入数据路径：标签和预测结果的 csv 的路径。

输出参数

校验结果输出路径：校验结果输出路径。

算法参数

- 标签列（单列）：标签所在列的列号，从0开始计数。
- 预测结果列（支持多列）：预测结果所在列的列号，从0开始计数。
- 输入数据是否包含 header 信息：是，否。
- 输入数据分割符：逗号，分号，空格，Tab。
- 校验类型（选项）：独立 T 检验，配对 T 检验，Delong 检验，pearson、spearmanr、kendalltau。
- 置信度水平：默认0.95。

Angel 算法指南
Angel 算法简介

最近更新时间：2021-12-24 15:11:36

Angel 是由腾讯自研并开源的高性能分布式机器学习和图计算平台，它提供了用于特征工程、模型构建、参数调优、模型服务和 AutoML 的全栈设施，包括传统机器学习、深度学习、图表示学习和图神经网络等算法。

Spark on Angel

凭借 Angel 强大的 PS Service 能力，Spark on Angel 扩展了 Spark 的参数更新能力，使 Spark 也具备高速训练大模型的能力而不用再顾虑 Spark Driver 的单点性能问题。Spark on Angel 组件一般用来运行用户自己实现的算法。

图算法

图算法基于 Spark-on-Angel 系统，利用 Angel 对高维稀疏数据的存储能力和 Spark 对海量数据的处理能力，搭建了端到端的图计算平台，为您提供节点测度（PageRank、Kcore、Closeness），社区发现 LPA，FastUnfolding 和图表示学习 LINE，Word2Vec 等算法。

PyTONA 算法

PyTONA 算法将 Angel 的参数服务器与深度学习系统 PyTorch 相结合，使用户可以在利用 PyTorch 编写最新推荐模型的同时，利用 Angel 参数服务器的扩展性，进行大规模分布式的推荐模型训练。相比于 TensorFlow，Angel 的参数服务器具备处理高维稀疏离散大模型的能力，PyTorch 则更加轻量，更易于进行新模型的尝试。

机器学习算法

腾讯云 TI 平台 TI-ONE 还提供基于 Spark on Ange 的机器学习算法，如分类算法等。如果需要运行 Spark on Angel 自带算法，建议您使用各个算法对应的算法组件。

Spark on Angel

最近更新时间：2021-12-24 15:10:48

Angel 是由腾讯自研并开源的高性能分布式机器学习和图计算平台，它提供了用于特征工程、模型构建、参数调优、模型服务和 AutoML 的全栈设施，包括传统机器学习、深度学习、图表示学习和图神经网络等算法。

凭借 Angel 强大的 PS Service 能力，Spark on Angel 扩展了 Spark 的参数更新能力，使 Spark 也具备高速训练大模型的能力而不用再顾虑 Spark Driver 的单点性能问题。Spark on Angel 组件一般用来运行用户自己实现的算法，如果需要运行 Spark on Angel 自带算法，建议您使用各个算法对应的算法组件。

操作步骤

1. **添加组件**
   从左侧菜单栏中，选择【框架】>【机器学习】列表下的【Spark on Angel】节点，并将其拖拽至画布中。
2. **配置参数**
   - 作业 Jar 包：通过该配置框上传您的 Spark on Angel 应用程序 Jar 包。
   - 主类名：指定您的 Spark on Angel 应用程序的入口类，即 main 函数所在的类。
   - 程序参数：您的 Spark on Angel 应用程序所需的参数，即传给 main 函数的参数。
3. **配置资源**
   - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
   - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
   - driver 节点资源类型：请选择合适的 drive 节点机型。
   - executor 节点资源类型：请选择合适的 executor 节点机型。
   - master 节点资源类型：请选择合适的 master 节点机型。
   - ps 节点资源类型：请选择合适的 ps 节点机型。
   - spark conf 参数。
4. **运行**
   单击【保存】并运行工作流。
5. **查看 Spark 控制台和日志**
   在 Spark on Angel 节点上单击右键菜单，可查看任务状态。

案例说明

本案例向您阐述如何在腾讯云 TI 平台 TI-ONE 使用 Spark on Angel 组件离线训练自己实现的算法模型。

示例代码

平台内置的 Spark on Angel 版本是 2.3.1，所以用户在本地打包时请引入 Spark on Angel 2.3.1 相关的依赖。使用 maven 作为打包工具，此时打包后的 jar 包我们命名为 spark-on-angel-examples.jar。

代码取自官方 example，LogisticRegression on Spark on Angel。

```
/*
* Tencent is pleased to support the open source community by making Angel available.
*
* Copyright (C) 2017-2018 THL A29 Limited, a Tencent company. All rights reserved.
*
* Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file exc
ept in
* compliance with the License. You may obtain a copy of the License at
*
* https://opensource.org/licenses/Apache-2.0
*
* Unless required by applicable law or agreed to in writing, software distributed under the Licen
se
* is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, eith
er express
* or implied. See the License for the specific language governing permissions and limitations u
nder
* the License.
*
*/


package com.tencent.angel.spark.examples.cluster

import com.tencent.angel.RunningMode
import com.tencent.angel.conf.AngelConf
import com.tencent.angel.ml.core.conf.{MLConf, SharedConf}
import com.tencent.angel.spark.context.PSContext
import com.tencent.angel.spark.ml.core.{ArgsUtil, GraphModel, OfflineLearner}
import org.apache.spark.{SparkConf, SparkContext}

object OfflineRunner {

def main(args: Array[String]): Unit = {
```

```
val params = ArgsUtil.parse(args)

// Train data path
val input = params.getOrElse("input", "")

// Can be used in train/predict mode, it means model output path in train mode and model load path in predict mode
val output = params.getOrElse("modelPath", "")
val actionType = params.getOrElse("actionType", "train")
val network = params.getOrElse("network", "LogisticRegression")

// Model load path in train mode, just use in train mode
val modelPath = params.getOrElse("model", "")

// Predict result save path, just use in predict mode
val predictPath = params.getOrElse("predictPath", "")

// set running mode, use angel_ps mode for spark
SharedConf.get().set(AngelConf.ANGEL_RUNNING_MODE, RunningMode.ANGEL_PS.toString)

// build SharedConf with params
SharedConf.addMap(params)

val dim = SharedConf.indexRange.toInt

println(s"dim=$dim")

// load data
val conf = new SparkConf()

// we set the load model path for angel-ps to load the meta information of model
actionType match {
case MLConf.ANGEL_ML_TRAIN => {
if(modelPath.length > 0)
conf.set(AngelConf.ANGEL_LOAD_MODEL_PATH, modelPath)
}
case MLConf.ANGEL_ML_PREDICT => {
if(output.length > 0)
```

```
conf.set(AngelConf.ANGEL_LOAD_MODEL_PATH, output)
}
}

val sc = new SparkContext(conf)

// start PS
PSContext.getOrCreate(sc)

val className = "com.tencent.angel.spark.ml.classification." + network
val model = GraphModel(className)
val learner = new OfflineLearner

actionType match {
case MLConf.ANGEL_ML_TRAIN => learner.train(input, output, modelPath, dim, model)
case MLConf.ANGEL_ML_PREDICT => learner.predict(input, predictPath, output, dim, model)
}
}
}
```

平台配置

1. 添加组件

从左侧菜单栏中，选择【框架】>【机器学习】列表下的 Spark on Angel 节点，并将其拖拽至画布中。

2. 配置参数

- 作业 Jar 包：通过该配置框上传本地编译后的 spark-on-angel-examples.jar。
- 主类名：指定您的 Spark on Angel 应用程序的入口类，即 main 函数所在的类。
- 程序参数：您的 Spark on Angel 应用程序所需的参数，即传给 main 函数的参数。

3. 配置资源

您可按需选择资源参数。

4. 运行

单击【保存】并运行工作流。

查看 Spark 控制台

在 Spark on Angel 节点上单击右键菜单【Spark 控制台】，可查看任务状态。

运行结果

运行成功后，在用户指定的模型保存目录下会有模型文件生成。

文件名	大小	存储类型
input_bias/	-	-
input_weight/	-	-
graph.json	850B	标准存储

图算法

最近更新时间：2020-12-14 11:33:03

[Angel]Closeness on SONA

算法简介

紧密中心度算法（Closeness Centrality）计算一个节点到所有其他可达节点的最短距离的倒数，进行累积后归一化的值。紧密中心度可以用来衡量信息从该节点传输到其他节点的时间长短。
节点的 Closeness Centrality 越小，其在所在图中的位置越靠近中心。紧密中心度算法（Closeness Centrality）适用于社交网络中关键节点挖掘等场景。

参数说明

- **输入数据格式**
  边表形式，每行一条边，分为带权重和不带权重两种形式，如下所示：

```
边不带权重
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...

边带权重
srcId 分隔符 dstId 分隔符 weight
srcId 分隔符 dstId 分隔符 weight
... ... ...
... ... ...
```

- **算法 IO 参数**
  - 边输入路径：输入的边表所在的 COS 路径，每行一条边。
  - 结果输出路径：输出结果保存 COS 路径。
  - 分区数：Spark RDD 的分区数量。
  - srcIndex：输入数据的源顶点列下标，0是第一列，默认为0。
  - dstIndex：输入数据的目标顶点列下标，默认为1。
  - PS分区个数：参数服务器模型的分区数量。
  - 分隔符：每条边的起始顶点、目标顶点以及权重列之间的分隔符：tab，空格等。
  - 输出额外信息：是否输出节点 cardinality 及以半径加权求和的 cardinality，true or false。

- **算法参数**
  - storageLevel：RDD 存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。
  - 均衡分区：参数服务器对输入数据节点存储划分是否均衡分区，如果输入节点的索引不是均匀的话建议选择否。
  - 精度：HyperLogLog counter 的精度（≥4）。精度会以指数形式影响算法性能，建议在资源允许的情况下尽量设为10。
  - 分批向 PS 更新数值的批数：每个 worker 分批向参数服务器传输数值更新的批次。
  - 图的类型：无向图或者有向图。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 ps 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了网络中节点的重要性评价的问题

- 原始数据如下图所示：

```
1 2 0.381926670577
1 3 0.137141712315
1 4 0.363602521259
1 5 0.327848933153
1 6 0.282351916721
1 7 0.289094152066
1 8 0.911038250669
1 9 0.403538220425
1 794106 0.808741084239
1 908381 0.344581317466
1 1427826 0.406062985725
1 1427827 0.750857815832
1 1750238 0.959075805008
2 3 0.483304527168
2 4 0.0554463878699
2 7 0.290931337689
2 9 0.930012903374
2 23 0.349294675562
2 55 0.625915186232
2 175 0.158129118323
2 187 0.854984536208
2 270 0.969068443219
2 638 0.316151838831
2 645 0.352465985852
```

- 运行算子后得到的结果输出格式为 nodeId(long) | closeness(float) | 节点 cardinality | 半径加权求和的 cardinality， closeness 值越大表示节点越重要。

```
1488985 0.20409247      1468967 7197556
1488986 0.22557351      1468967 6512143
1488987 0.21458225      1468967 6845706
1488988 0.20446524      1468967 7184434
1488989 0.21514168      1468967 6827905
1488990 0.20588484      1468967 7134896
1488991 0.20446524      1468967 7184434
1488992 0.25458002      1468967 5770158
1488993 0.20446524      1468967 7184434
1488994 0.2083924       1468967 7049043
1488995 0.20651776      1468967 7113030
1488996 0.20446524      1468967 7184434
1488997 0.21297738      1468967 6897291
1488998 0.21033262      1468967 6984019
1488999 0.20977367      1468967 7002628
1489000 0.20446524      1468967 7184434
1489001 0.22449532      1468967 6543419
1489002 0.21033262      1468967 6984019
1489003 0.20977362      1468967 7002630
1489004 0.20446524      1468967 7184434
1489005 0.21284814      1468967 6901479
1489006 0.20527977      1468967 7155927
1489007 0.2207164       1468967 6655450
1489008 0.20446524      1468967 7184434
1489009 0.20465615      1468967 7177732
1489010 0.20620626      1468967 7123775
1489011 0.20446524      1468967 7184434
1489012 0.20567487      1468967 7142180
1489013 0.20622325      1468967 7123188
1489014 0.20527416      1468967 7156122
```

[Angel]Common Friends on SONA

算法简介

计算顶点与它的直接（一度）邻居之间的共同邻居数量。在社交网络中，可以全量计算网络中每个顶点的共同好友数量，也可以输入多对不存在好友关系的顶点对，计算这些顶点之间的共同好友数量。

参数说明

- **输入数据格式**

  边表形式，每行一条边，每条边由起始顶点和终止顶点标识，如下所示：

```
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...
```

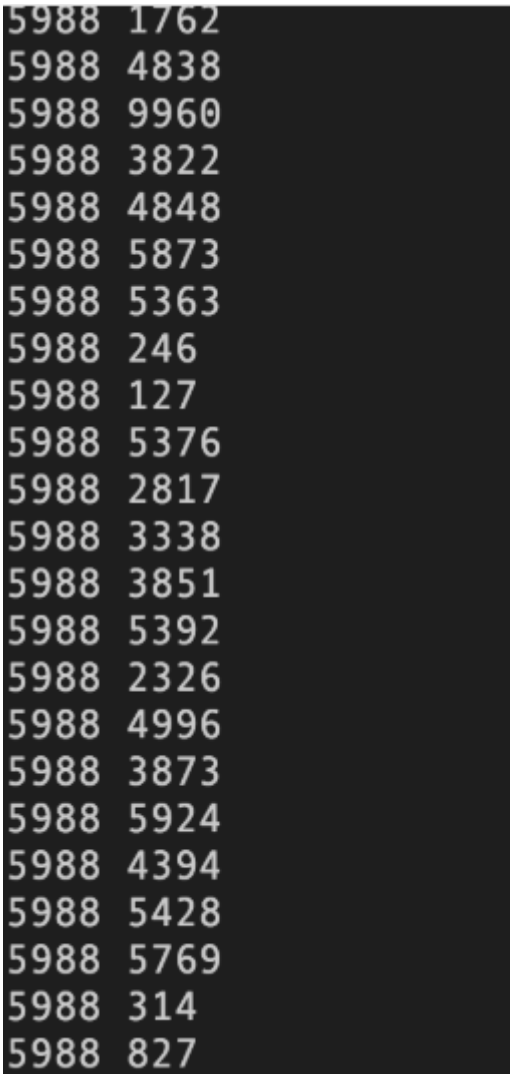
- **算法 IO 参数**
  - 所有好友边：输入的边表所在的 COS 路径，每行一条边，每条边由起始顶点和终止顶点标识。
  - 需要计算的好友边：需要计算共同好友的顶点对（边）COS 路径，每行一条。
  - 结果输出路径：输出结果保存 COS 路径。

- 数据分隔符：每条边的起始顶点和终止顶点之间的分隔符：tab，空格等。
- checkpoint 路径：模型的 checkpoint 保存 COS 路径。
- RDD 存储级别：DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。

- **算法参数**
  - 分区数量：Spark RDD 的分区数量。
  - 参数服务器分区数量：参数服务器模型的分区数量。
  - 数据中源顶点的列：输入数据的源顶点列下标，0是第一列，默认为0。
  - 数据中目标顶点的列： 输入数据的目标顶点列下标，默认为1。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 ps 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了计算两个好友的共同好友数的问题，某种程度上可以刻画两个节点之间的紧密程度

- 原始数据如下图所示：



- 运行算子后得到的结果输出格式为输出的格式为 srcId(long) | dstId(long) | count(int)。

```
6284    1358    6
6284    1797    8
6284    2240    9
6284    2368    5
6284    3560    10
6284    4798    6
6284    4838    13
6284    4983    11
6284    4996    8
6284    6140    9
6284    6468    8
6284    6638    6
6284    6872    10
6284    7622    7
8400    175     41
8400    644     28
8400    791     23
8400    857     21
8400    1225    36
8400    2637    20
8400    3146    14
8400    3406    30
8400    4373    44
8400    4593    21
8400    4651    25
8400    5002    10
8400    5631    12
8400    5680    32
8400    5817    22
8400    6065    18
```

[Angel]FastUnfolding on SONA

算法简介

模块度成为度量社区划分优劣的重要标准，划分后的网络模块度值越大，说明社区划分的效果越好，Fast Unfolding 算法便是基于模块度对社区划分的算法。Fast Unfolding 算法是一种迭代的算法，主要目标是不断划分社区使得划分后的整个网络的模块度不断增大。

参数说明

- **输入数据格式**

  边表形式，每行一条边，分为带权重和不带权重两种形式，如下所示：

```
边不带权重
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...

边带权重
```

```
srcId 分隔符 dstId 分隔符 weight
srcId 分隔符 dstId 分隔符 weight
... ... ...
... ... ...
```

- **算法 IO 参数**
  - 边输入路径：输入的边表所在的 COS 路径，每行一条边。
  - 输出路径：输出结果保存 COS 路径。
  - checkpoint 路径：模型的 checkpoint 保存 COS 路径。
  - srcIndex：输入数据的源顶点列下标，0是第一列，默认为0。
  - dstIndex：输入数据的目标顶点列下标，默认为1。
  - 输入数据分隔符：每条边的起始顶点、目标顶点之间的分隔符：tab，空格等。
  - 是否带权：边是否带权重。
  - weightIndex：输入数据边的权重列下标，默认为2。
- **算法参数**
  - storageLevel：RDD 存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。
  - 数据分区数：Spark RDD 分区数目。
  - psPartitionNum：参数服务器模型的分区数量。
  - batchSize：每个 batch 数据大小。
  - numFold：折叠次数。
  - numOpt：每轮模块度优化次数。
  - eps：每次模块度优化提升下界。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了社团发现的问题，目标是从图数据中挖掘节点之间的社群属性

- 原始数据如下图所示：

```
214328887 34428380
17116707 28465635
380580781 18996905
221036078 153460275
107830991 17868918
151338729 222261763
19705747 34428380
222261763 88323281
19933035 149538028
158419434 17434613
149538028 153226312
364971269 153226312
100581193 279787626
113058991 69592091
151338729 187773078
406628822 262802533
460282402 88323281
280935165 437804658
222261763 27633075
285312927 151338729
279787626 131613362
158419434 17675120
394263193 100581193
254839786 88323281
204317520 21548772
67864340 172883064
270449528 297801196
153226312 187773078
67864340 8088112
153226312 17116707
```

- 运行算子后得到的结果输出的格式为 nodeId(long) | communityId(long)，communityId 一致表示属于同一个社区，编号不连续。

```
282268551       5538692
225490101       13050752
191737201       18091925
257234051       39271212
236143101       14192900
266043601       14299000
221523901       7573752
222090651       11866022
235246701       7309052
273552051       10680412
270548301       14305350
276176651       5906772
272099201       34942750
276270001       11668422
264093451       8627002
239545701       8826702
244154451       23003171
220021101       9785602
243081951       14111650
200259301       7885972
244145701       10632552
198500751       13786252
264727251       37010960
250160301       14102041
271586451       14402018
276912001       14372700
267400351       5848302
249262601       14165300
264403551       12958782
224915001       19324787
```

[Angel]Kcore on SONA

算法简介

对输入的网络数据，获得每个节点的 coreness。Kcore 算法的最直观解释是剥洋葱，即首先将网络中度为1的节点剥离，
此时剩余的网络中可能会产生新的度为1的节点，再依次剥离，直到剩余的网络中所有节点的度都大于等于2。 那么之前被剥离的节点的 coreness 即为1。

之后再依次剥离得到 coreness 为2，3，...，n 的节点，直到网络中所有的节点都得到对应的 coreness。实际的分布式实现没有这么朴素，请参考论文 Distributed k-Core Decomposition。coreness 越大表示节点属于越中心的位置，从而也越重要。

参数说明

- **输入数据格式**
  边表形式，每行一条边，每条边由起始顶点和终止顶点标识，如下所示：

```
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...
```

- **算法 IO 参数**
  - 边输入路径：输入的边表所在的 COS 路径，每行一条边。
  - srcIndex：输入数据的源顶点列下标，0是第一列，默认为0。
  - dstIndex: 输入数据的目标顶点列下标，默认为1。
  - 输入数据分隔符：每条边的起始顶点、目标顶点之间的分隔符：tab，空格等
  - 输出路径：输出结果保存 COS 路径。
  - checkpoint 路径：模型的 checkpoint 保存 COS 路径。
- **算法参数**
  - storageLevel：RDD 存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。
  - 数据分区数：Spark RDD 分区数目。
  - ps 分区数：参数服务器模型的分区数量。
  - 是否使用均衡分区：参数服务器对输入数据节点存储划分是否均衡分区，如果输入节点的索引不是均匀的话建议选择否。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。

- driver 节点资源类型：请选择合适的 drive 节点机型。
- executor 节点资源类型：请选择合适的 executor 节点机型。
- master 节点资源类型：请选择合适的 master 节点机型。
- ps 节点资源类型：请选择合适的 ps 节点机型。
- spark conf 参数
  - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
  - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了网络中节点的重要性评价的问题

- 原始数据如下图所示：

```
2 17820
2 18068
2 18397
2 18592
2 19124
2 19476
2 19520
2 19556
2 19708
2 20635
2 20773
2 20774
2 20789
2 21395
2 22366
2 23230
2 23638
2 23675
2 24637
2 24704
2 25943
2 26978
2 27001
2 27014
2 27023
2 27599
2 28658
2 28837
2 30620
2 32660
2 33309
2 33918
```

- 运行算子后得到的结果输出格式为 nodeId(long) | coreness(int)，coreness 值越大表示节点越重要。

```
639950  1
1512700 1
637100  1
108150  6
369450  4
1055300 1
1114250 2
204100  13
1700250 1
1154200 3
950200  1
1235650 8
1553650 1
1158700 2
720200  3
796550  1
817200  1
94950   11
1794700 2
428150  1
386050  48
360900  10
775900  1
674850  1
1772600 2
1493050 1
284450  1
356800  1
795150  1
1558650 1
```

[Angel]LINE_V2 on SONA

算法简介

LINE Large-scale Information Network Embedding 算法，Network Embedding 领域著名的算法之一，重点在于设计了两个目标函数，分别刻画节点之间的一阶相似性（直接连边）和二阶相似性（相似邻居）。

参数说明

- **输入数据格式**

  无向图，节点需要从0开始连续编号，以空白符或者逗号分隔，如下所示：

```
0 2
2 1
3 1
```

```
3 2
4 1
... ...
... ...
```

- **算法 IO 参数**
  - 训练数据：训练数据所在的 COS 路径。
  - remapping：是否对节点 ID 进行重新映射。
  - 模型保存路径：训练结果保存 COS 路径。
- **算法参数**
  - Embedding 特征的维度：Embedding 向量的维度，是 ps 分区数的倍数。
  - ps 分区数：参数服务器模型的分区数量。
  - 负采样数：负采样节点个数。
  - 学习率：初始学习率。
  - BatchSize：每个 mini batch 数据大小。
  - 最大 epoch 数：最大迭代轮数。
  - Order：一阶或者二阶的 LINE。
  - subSample：是否对训练数据进行采样。
  - 每隔多少个 epoch 保存一次模型：每隔多少个 epoch 保存一次模型。
  - 每隔多少轮做一次 checkpoint：每隔多少轮做一次 checkpoint。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了将难以处理的图数据转换为易于处理的向量型数据（将高维的图数据嵌入到低维的向量空间）的问题

- 原始数据如下图所示：

```
214328887 34428380
17116707 28465635
380580781 18996905
221036078 153460275
107830991 17868918
151338729 222261763
19705747 34428380
222261763 88323281
19933035 149538028
158419434 17434613
149538028 153226312
364971269 153226312
100581193 279787626
113058991 69592091
151338729 187773078
406628822 262802533
460282402 88323281
280935165 437804658
222261763 27633075
285312927 151338729
279787626 131613362
158419434 17675120
394263193 100581193
254839786 88323281
204317520 21548772
67864340 172883064
270449528 297801196
153226312 187773078
67864340 8088112
153226312 17116707
```

- 运行算子后得到的结果有节点向量文件和节点 ID 映射文件。
- ID 映射文件格式为：映射后的节点 ID：原节点 ID。

```
0:28611918
1:145739229
2:353875935
3:229906813
4:138134695
5:217416964
6:56042522
7:43969852
8:49969223
9:14936142
10:62631420
11:43303409
12:305073376
13:732073
14:531775806
15:29158922
16:110415367
17:28000466
18:52451443
19:425124875
20:346882245
21:104169476
22:80022606
23:23853982
24:6080022
25:179510652
26:47187685
27:138014258
28:155620461
29:45904217
```

- 词向量文件格式为：映射后的节点 ID：节点向量。

```
0:0.2898812 -0.047034055 -0.042045355 0.4276733 0.30882192 0.0038887805 0.27694055
-0.12006601 -0.120681144 0.103084706 0.2345341 -0.23847368 0.2526599 0.31838563
-0.16692284 -0.33219445 0.058145545 -0.2509371 0.32117867 -0.03681458 -0.13352285
-0.019652104 -0.26246533 0.21991372 -0.11176241 0.030767515 -0.3209092 0.010717862
-0.0472607 -0.10711371 -0.116032586 0.016764972 -0.10913398 -0.16196716 -0.010743721
0.16318785 -0.0031017836 0.22008748 0.2456535 -0.47097415 0.1810638 -0.22511122
0.051662043 0.061026413 0.09730562 0.022381173 -0.116673164 -0.062071186 0.037573524
0.12565996 -0.14277667 0.11718496 -0.33158565 0.15205628 -0.10584539 0.12724379 0.0837177
0.0633107 0.052425776 -0.12510064 -0.10304372 0.10064604 -0.25814906 -0.043349676
-0.0647264 0.37888378 -0.041122045 -0.15174721 0.31346837 -0.11513387 -0.11248529
-0.08968258 0.17045689 0.10403982 0.3508255 0.122624435 -0.02356654 -0.078666575
0.075153574 -0.12484962 -0.2132088 0.37333682 0.14387383 -0.0043084025 0.0026193284
0.122436084 0.093921825 0.13332695 -0.3528769 0.17361347 0.136092 -0.28001904 -0.065318964
-0.09192268 -0.14953 0.13457565 0.009737076 0.03245861 -0.011260181 -0.17032576
-0.025500191 0.225259 0.010825465 0.08550035 0.18094008 -0.024440603 -0.37060487
-0.111062795 -0.113206975 -0.03076769 -0.30103722 -0.13427535 0.29618007 0.020595286
0.0891129 0.21706551 0.052805796 -0.029851768 -0.010125663 -0.1338474 -0.027037874
0.12828681 -0.085413516 -0.029099409 -0.1473377 -0.25299066 -0.09719495 -0.05066044
1:0.15396145 -0.03399318 -0.025618391 0.24543856 0.16971079 0.006164773 0.14896606
-0.059113406 -0.059358966 0.06554142 0.13003832 -0.109991714 0.13613388 0.1635272
-0.08661964 -0.17715803 0.033492237 -0.13268925 0.18180203 -0.013440184 -0.07691324
-0.023086684 -0.14805198 0.1265929 -0.06372057 0.012285376 -0.17066298 0.016570589
-0.024537923 -0.07299372 -0.06352418 -0.0065579494 -0.05633624 -0.079201914 2.601383E-4
0.08377397 -0.0026846523 0.11296575 0.14766073 -0.2386535 0.09719485 -0.11571432
0.030700749 0.0399174 0.055291213 0.006381305 -0.03225908 -0.028265063 0.01867099
0.05865118 -0.08119735 0.05661325 -0.17788818 0.102336735 -0.056368176 0.059462566
0.04233031 0.04380105 0.030015515 -0.082805514 -0.049679484 0.05570632 -0.1475022
-0.015011237 -0.045077268 0.1983284 -0.03223794 -0.08618669 0.16953442 -0.05217284
-0.061583724 -0.058652967 0.084571466 0.052958667 0.19428891 0.062122174 -0.0038125203
-0.0510118 0.02935238 -0.0852438 -0.11997961 0.20398352 0.09827316 -7.904256E-4
```

[Angel]PageRank on SONA

算法简介

PageRank 算法可能是最著名的节点重要性评价算法，最初由拉里佩奇提出，被应用于 Google 搜索的网页排名。详细算法细节参考论文 The PageRank Citation Ranking:Bringing Order to the Web。

参数说明

- **输入数据格式**

  边表形式，每行一条边，分为带权重和不带权重两种形式，如下所示：

```
边不带权重
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...

边带权重
srcId 分隔符 dstId 分隔符 weight
```

```
srcId 分隔符 dstId 分隔符 weight
... ... ...
... ... ...
```

- **算法 IO 参数**
  - 边输入路径：输入的边表所在的 COS 路径，每行一条边。
  - 结果输出路径：输出结果保存 COS 路径。
  - 分区数：Spark RDD 的分区数量。
  - srcIndex：输入数据的源顶点列下标，0是第一列，默认为0。
  - dstIndex：输入数据的目标顶点列下标，默认为1。
  - weightIndex：输入数据边的权重列下标，默认为2。
  - PS 分区个数：参数服务器模型的分区数量。
  - 分隔符：每条边的起始顶点、目标顶点以及权重列之间的分隔符：tab、空格等。
- **算法参数**
  - storageLevel：RDD 存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。
  - 均衡分区：参数服务器对输入数据节点存储划分是否均衡分区，如果输入节点的索引不是均匀的话建议选择否。
  - 均衡划分比例：0 - 1之间的浮点数，节点索引越不均衡，该值越小，建议0.7。
  - tolerance：停止进行消息更新的精度阈值。
  - 重定向概率：重定向概率。
  - 是否带权：边是否带权重。
  - 版本：图的切割方式，点切或者边切。
  - 存储时分批次数：存储时的分批次数。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了网络中节点的重要性评价的问题

- 原始数据如下图所示：

```
1 2 0.381926670577
1 3 0.137141712315
1 4 0.363602521259
1 5 0.327848933153
1 6 0.282351916721
1 7 0.289094152066
1 8 0.911038250669
1 9 0.403538220425
1 794106 0.808741084239
1 908381 0.344581317466
1 1427826 0.406062985725
1 1427827 0.750857815832
1 1750238 0.959075805008
2 3 0.483304527168
2 4 0.0554463878699
2 7 0.290931337689
2 9 0.930012903374
2 23 0.349294675562
2 55 0.625915186232
2 175 0.158129118323
2 187 0.854984536208
2 270 0.969068443219
2 638 0.316151838831
2 645 0.352465985852
```

- 运行算子后得到的结果输出格式为 nodeId(long) | rank(float)，rank 值越大表示节点越重要。

```
1833327 0.62738
1833328 1.1309646
1833329 1.6518683
1833330 0.3990053
1833331 11.042695
1833332 0.7988026
1833333 0.5726967
1833334 0.26107377
1833335 0.7131747
1833336 0.64276665
1833337 0.2896067
1833338 0.3704346
1833339 0.79982185
1833340 0.26838666
1833341 0.6007695
1833342 0.8122548
1833343 0.70206827
1833344 0.23111989
1833345 0.4796366
1833346 0.4578224
1833347 0.7777402
1833348 0.26449165
1833349 0.8121565
1833350 0.73096466
1833351 0.66768336
1833352 0.64804393
1833353 0.2935462
1833354 0.5862066
1833355 0.22112657
1833356 0.59728956
```

[Angel]Hindex on SONA

算法简介

HIndex 算法是计算一个节点 h-index 指数的算法。在一个 graph 中，一个节点的 h-index 值为 h 时，表示该节点至少有 h 个邻居的度大于或等于 h，通常来说，h-index 值越高，表明该节点的影响力越大，适用于社交网络中关键节点挖掘等场景。

参数说明

- **输入数据格式**

  边表形式，每行一条边，如下所示：

```
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...
```

- **算法 IO 参数**

- 边输入路径：输入的边表所在的 COS 路径，每行一条边。
- 出发节点所在列：输入数据的源顶点列下标，0是第一列，默认为0。
- 目标节点所在列：输入数据的目标顶点列下标，默认为1。
- 输入数据分隔符：每条边的起始顶点、目标顶点以及权重列之间的分隔符：tab，空格等。
- 输出路径：输出结果保存 COS 路径。

- **算法参数**
  - 数据分区数：Spark RDD 数据的分区数量。
  - ps 分区数：参数服务器上模型的分区数量。
  - 是否使用均衡分区：参数服务器对输入数据节点存储划分是否均衡分区，如果输入节点的索引不是均匀的话建议选择否
  - storageLevel：RDD 存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了网络中节点的重要性评价的问题

- 原始数据如下图所示：

```
2 17820
2 18068
2 18397
2 18592
2 19124
2 19476
2 19520
2 19556
2 19708
2 20635
2 20773
2 20774
2 20789
2 21395
2 22366
2 23230
2 23638
2 23675
2 24637
2 24704
2 25943
2 26978
2 27001
2 27014
2 27023
2 27599
2 28658
2 28837
2 30620
2 32660
2 33309
2 33918
```

- 运行算子后得到的结果输出格式为 nodeId(long) | HIndex值(Int) | GIndex值(Int) | WIndex值(Int)，index 值越大表示节点越重要

- GIndex 和 WIndex 值是与 hindex 相近的指标，可自行查阅了解。

```
1297500 2       3       0
903500  2       2       0
1512700 1       1       1
856500  2       2       1
637100  1       1       1
913000  1       1       0
18500   233     358     95
81100   147     186     63
172600  26      28      19
204100  17      21      4
1055300 1       1       1
1154200 3       3       0
1446200 2       3       2
1735500 1       1       1
950200  1       1       1
1022800 1       1       1
1464300 1       1       1
1481700 1       1       1
1158700 2       2       1
1486900 1       1       1
720200  3       6       1
1383700 3       3       2
817200  1       1       0
1858600 1       1       0
477800  1       1       1
1794700 2       2       1
472200  2       2       1
360900  10      15      4
775900  1       1       1
1772600 3       3       2
```

[Angel]LPA on SONA

算法简介

LPA 是一种基于标签传播的局部社区划分。对于网络中的每一个节点，在初始阶段，Label Propagation 算法对于每一个节点都会初始化一个唯一的一个标签。

每一次迭代都会根据与自己相连的节点所属的标签改变自己的标签，更改的原则是选择与其相连的节点中所属标签最多的社区标签为自己的社区标签，这就是标签传播的含义了。随着社区标签不断传播。最终，连接紧密的节点将有共同的标签。

参数说明

- **输入数据格式**

  边表形式，每行一条边，如下所示：

```
srcId 分隔符 dstId
srcId 分隔符 dstId
... ...
... ...
```

- **算法 IO 参数**
  - 边输入路径：输入的-表所在的 COS 路径，每行一条边。
  - 出发节点所在列：输入数据的源顶点列下标，0是第一列，默认为0。
  - 目标节点所在列：输入数据的目标顶点列下标，默认为1。
  - 输入数据分隔符：每条边的起始顶点、目标顶点之间的分隔符: tab，空格等。
  - 输出路径：输出结果保存 COS 路径。
  - checkpoint 路径：模型的 checkpoint 保存 COS 路径。
- **算法参数**
  - storageLevel：RDD存储级别，DISK_ONLY/MEMORY_ONLY/MEMORY_AND_DISK
  - 数据分区数：Spark RDD 分区数目。
  - ps 分区数：参数服务器上模型的分区数量。
  - 是否使用均衡分区：参数服务器对输入数据节点存储划分是否均衡分区，如果输入节点的索引不是均匀的话建议选择否。
- **spark conf 参数**
  - spark.angel.tmp.output.path.prefix：Angel 临时目录的前缀路径，为 COS 路径。
  - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。
- **资源参数**
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - executor-memory(g)：每个 executor 需要的内存，单位为 g。
  - executor-cores：每个 executor 需要的 CPU 核数。
  - driver-memory(g)：spark driver 需要的内存，单位为 g。
  - spark.ps.instances：angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - spark.ps.cores：每个 angel ps 需要的核数。
  - spark.ps.memory(g)：每个 angel ps 需要的内存，单位为 g。

本算子主要解决了社团发现的问题，目标是从图数据中挖掘节点之间的社群属性

- 原始数据如下图所示：

```
2 17820
2 18068
2 18397
2 18592
2 19124
2 19476
2 19520
2 19556
2 19708
2 20635
2 20773
2 20774
2 20789
2 21395
2 22366
2 23230
2 23638
2 23675
2 24637
2 24704
2 25943
2 26978
2 27001
2 27014
2 27023
2 27599
2 28658
2 28837
2 30620
2 32660
2 33309
2 33918
```

- 运行算子后得到的结果输出的格式为 nodeId(long) | communityId(long)、communityId 一致表示属于同一个社区，编号不连续。

```
1750238 1
1427826 1
1427827 1
295908  2
295913  2
295930  2
295939  2
295940  2
356952  2
662454  2
1647569 2
1479973 2
295926  2
1635427 2
295927  2
295936  2
356948  2
1104158 2
153361  4
224     6
135     6
151     6
265     6
284     6
67      6
70      6
180     6
639934  7
1192839 7
1185045 7
```

[Angel]GraphSage_Supervised

算法简介

GraphSAGE 全称是 Graph SAmple and aggreGate，由斯坦福大学提出，该算法旨在从图网络中学习出节点的特征表示，并以此为理论基础，斯坦福和 Pinterest 公司合作提出了第一个工业级别（数十亿节点和数百亿边）基于 GCN 的推荐系统，并在离线评估和 AB 实验选中取得了不错的效果。
Pytorch on Angel 为运行图卷积神经网络算法提供了能力。 我们遵循 Pytorch-Geometric 来定义图卷积神经网络同时使用 Angel 参数服务器来存储网络结构及节点特征。

参数说明

- **输入数据格式**

  有三个输入需要准备：边表，节点特征表和节点标签表。

  边表：是一个文件或者路径，每一行是一条边，由源点和终点组成，中间以空格分开，每个节点被编码成一个 long 型数值。

  特征表：是一个文件或者路径，每一行包含节点 ID 和节点特征，节点的特征支持稀疏和稠密两种格式。

  对于稀疏格式，每一行的格式如下：

```
node\tf1:v1 f2:v2 f3:v3
```

由于空格是特征内部之间的间隔符，因此节点与特征之间的间隔是 tab。
对于稠密格式如下：

```
node\tv1 v2 v3
```

标签表：是一个文件或路径，每一行包含节点 ID 和标签，并以空格间隔。由于该版本 graphsage 是半监督模型，因此标签数据可以不包含全部节点的标签。

> ⚠ 注意：
> 图中出现的每一个点，在特征表中都需要有相应的特征。

- **算法 IO 参数**
  - 边路径：边数据的存储 COS 路径。
  - 特征路径：特征数据的存储 COS 路径。
  - 标签路径：标签数据的存储 COS 路径。
  - 预测值输出路径：预测值输出路径。
  - embedding 输出路径：表示向量输出路径。
  - 模型输出路径：模型保存路径。
  - 验证标签路径：验证集标签数据的存储 COS 路径，为空时表示不使用额外的验证集。
- **算法参数**
  - batchSize：每个 batch 的样本数。
  - 学习率：算法学习率。
  - 数据分区数：数据分区个数。
  - ps 分区数：ps 分区数，只有在使用均衡分区时才会生效。
  - 均衡分区：是否使用 ps 均衡分区，为 true（是）表示使用 ps 均衡分区，此时设置的ps分区数才会生效，否则表示不使用，默认情况下为 false（否）。
  - epoch：模型训练迭代的轮数。
  - 验证集比例：当验证标签路径为空时，使用验证集比例对输入数据进行划分，一部分做训练，一部分做验证；当验证标签路径不为空时，该值将不起作用。
  - 特征格式：稠密，稀疏。
  - 采样邻居个数：每阶采样的邻居的个数。
  - RDD 存储级别：可选 MEMORY_ONLY、DISK_ONLY、MEMORY_AND_DISK。
  - 初始化特征分批次数：ps 分批初始化时的分批次数 。
  - Pytorch模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装-pytorch(1.3.1版本)，然后到 python/graph 下载算法文件 graphsage.py，执行如下命令生成可用于分布式训练的 Pytorch 模型文件：

```
python graphsage.py --input_dim 602 --hidden_dim 128 --output_dim 41 --output_file sage_reddit_1.3.pt
```

其中，input_dim 为节点特征的维度，hidden_dim 为中间隐藏层的维度，output_dim 为类别数，output_file 是指生成脚本文件的名称，以'.pt'结尾，该名称可自己指定。该脚本是使用 TorchScript 生成的一个生成一个模型文件，其中包含 graphsage 的数据流图，生成的 pt 文件上传到自己的 COS 存储路径。

- 特征维度：输入数据节点特征的维度。
- 任务类型：训练，预测。
- 保存模型的周期 epoch 数：每隔多少 epoch 保存一次模型。
- 学习率衰减系数：学习率衰减参数。
- 评估指标：可选择 acc，binary_acc（二分类场景），auc，recall，precision，f1-score，rmse 等。
- 验证周期数：每隔 X 轮验证一次。

- **资源参数**
- num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的问题是学习网络图中节点的 embedding 向量

- 原始数据如下图所示：
  边数据

```
245 28279
245 80832
245 102428
245 123853
245 158276
245 69152
245 211193
245 70937
245 28674
245 124223
245 175627
245 79431
245 232554
245 92388
245 97811
245 131274
245 217580
245 17260
245 67753
245 185258
245 29052
245 35530
245 482
```

特征数据

```
4 2.50204934926 -4.01671331019 -1.00679252498 -2.50024283806 -1.1792767053 -3.30
589 -0.681190146881 1.54305142447 0.673432050178 -1.49874658463 -0.459586949029
365323226078 -0.221782306315 0.219591050163 0.135294180314 -0.551515617432 0.130
0768642 -^C0.425165177497 0.268237555042 0.638801943663 0.327269427101 0.5960401
556 -0.126204092671 0.0419745378135 0.46669898091 0.32411994766 -0.0590080670556
.379109111215 -0.452565765992 -0.199471893602 -0.00386883808895 -0.279107234974
044473089405 0.187744919185 0.163462639204 0.281057021875 -0.0481714562741 0.194
37835819 -0.212283596882 0.319407873991 0.54549051336 -0.0527479075331 -0.221197
61052 0.338168426718 -0.188279383999 -0.3829499401 0.462814629243 -0.53687632144
.458701434747 0.167887287342 -0.0488542084144 0.184067272501 -0.131879573968 -0.
0268579085679 0.264263538495 0.287586432468 -0.819554954265 -0.187418067851 0.04
289105625 0.365001182577 -0.461945356627 0.645013231094 -0.000809116328693 -0.46
639602838 -0.282501090409 -0.140720990896 -0.0635983874276 -0.57078114869 0.1190
28931189 0.20375864948 0.407865823618 0.196450915501 -0.114589680537 0.036636948
0.394395616583 0.554558967204 -0.29948213885 0.064262555428 -0.392600693767 0.18
2478261802 -0.668896696321 -0.346840314596 -0.31999344416 -0.389280260584 0.0158
03563371 -0.307831802696 -0.315548923635 -0.420671554543 -0.592232297412 0.09839
```

标签数据

```
126541 1
126542 38
126543 30
126544 25
126545 0
126546 34
126547 15
126548 15
126549 3
126550 15
126551 0
126552 3
126553 18
126554 28
126555 15
126556 0
126557 34
126558 35
126559 31
126560 22
126561 6
126562 25
126563 18
126564 5
126565 0
126566 3
126567 15
126568 15
```

- 训练完成后会得到每个节点的 embedding 向量

```
59300
-0.101006076,0.09672009,0.09066432,-0.022397662,-0.16015106,-0.072940014,0.0891726,0.03593
9407,-0.057052888,-0.020299077,-0.027996391,0.094235554,-0.051969808,0.07112672,-0.0032085
092,0.15696296,0.063309915,-0.09726555,-0.11374468,-0.076717936,0.21870388,-0.12281935,-0.
18608066,0.044033702,0.032213334,0.08759572,-0.05351148,0.04096452,0.014474234,0.13503271,
-0.067725725,-0.093879744,0.07039475,0.034715604,0.010091716,-0.0012690376,0.1503294,0.004
228998,0.16033168,0.024118496,0.04271365,-0.07002662,0.043193426,0.08971406,-0.049680527,-
0.016690338,0.07007755,0.016271466,-0.047587052,0.0324686,0.038266566,0.15195204,0.1081394
,
0.15196237,-0.20705844,-0.09236143,0.030038413,-0.060288493,-0.030284125,0.04107925,0.0390
1295,0.08350218,-0.037643075,-0.031110674,0.008384633,-0.10941429,-0.006639789,-0.06951745
,-0.1447602,-0.019906431,0.12125606,-0.010718306,-0.020589568,0.057079744,0.039215006,0.06
359325,-0.15569362,0.048392445,0.03716639,-0.042304195,0.048667595,-0.1274348,0.077101864,
0.023447646,-0.03065622,-0.036637045,-0.19770111,0.067883156,0.25927782,-0.051332943,0.115
98023,0.024827417,0.091641925,0.033324912,0.023655733,-0.024690801,-0.11311579,-0.19217727
,-0.07706465,-5.5763277E-4,-0.008889718,-0.105431646,0.032512758,-0.023962565,-0.029739305
,
0.018607289,0.053781062,-0.0457167,0.2406749,0.14200471,0.014628827,0.015306584,0.04567098
2,0.042671066,0.030222762,0.017063893,-0.12884402,-0.0649811,0.073540375,0.13032538,-0.027
523166,0.027986303,0.10826148,-0.025458463,0.10685,0.0077786613,-0.07150696,-0.084982954
228300
-0.15029502,0.12509347,0.06813762,-0.06256564,-0.06559508,-0.15062883,0.06018788,0.0070343
427,-0.08855551,0.009586039,-0.11661325,-0.039669827,-0.10115201,0.043847464,0.017123833,0
.
06872124,-0.02949688,-0.07560799,0.020790845,-0.053463444,0.13357982,-0.17106728,-0.201189
16,0.046715528,0.044287793,0.10635565,0.030950498,0.04421844,0.017359894,0.001258285,-0.03
8121056,-0.12792142,0.06534402,-0.012319515,-1.8073515E-4,-0.010829915,-0.012875969,0.0361
87842,0.11782508,-0.091740005,-0.07956342,0.022308093,0.030652884,0.14755526,-0.0750109,-0
.
059926126,0.11485239,0.028636292,-0.032113016,0.1943619,0.03432464,0.18588307,4.022269E-4,
```

[Angel]Word2Vec on SONA

算法简介

Word2Vec 将文本中的词语映射到 k 维向量空间中，同时向量空间上的相似度可以用来表示词语语义上的相似度。

参数说明

- **输入数据格式**

  文档，每行一个句子（单词或者节点 ID），如下所示：

```
word1 word2 word3 ...
word4 word5 word6 ...
... ...
... ...
```

- 算法 IO 参数
  - 训练数据：训练数据所在的 COS 路径。

- 模型保存路径：训练结果保存 COS 路径。
- 模型加载路径：增量训练需要。

- 算法参数
  - Embedding 特征的维度：embedding 向量的维度。
  - ps 分区数：参数服务器模型的分区数量。
  - RDD 数据分区数：训练数据的分区数。
  - 负采样数：负采样节点个数。
  - 窗口大小：滑动窗口大小。
  - 最大 epoch 数：最大迭代轮数。
  - BatchSize：每个 mini batch 数据大小。
  - 学习率：初始学习率。
  - 学习率衰减率：学习率衰减率系数。
  - 每隔多少个 epoch 保存一次模型：您可按需设置。
  - 每隔多少轮做一次 checkpoint：您可按需设置。
  - 是否对数据进行采样操作：是否对训练数据进行采样。
  - 是否进行 ID 重映射：是否进行 ID 重映射，如果是单词需要重映射到 ID。
  - RDD 存储级别：选择 RDD 存储级别，DISK_ONLY、MEMORY_ONLY、MEMORY_AND_DISK。

- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决了词表示学习的问题，将高维稀疏的词表示嵌入到低维的向量空间

- 原始数据如下图所示：

```
anarchism originated as a term of abuse first used against early working class radicals including the diggers of the englis
h revolution and the sans culottes of the french revolution whilst the term is still used in a pejorative way to describe a
ny act that used violent means to destroy the organization of society it has also been
taken up as a positive label by self defined anarchists the word anarchism is derived from the greek without archons ruler
chief king anarchism as a political philosophy is the belief that rulers are unnecessary and should be abolished although t
here are differing interpretations of what this means anarchism also refers to
related social movements that advocate the elimination of authoritarian institutions particularly the state the word anarch
y as most anarchists use it does not imply chaos nihilism or anomie but rather a harmonious anti authoritarian society in p
lace of what are regarded as authoritarian political structures and coercive economic institutions anarchists advocate soci
al relations based upon
voluntary association of autonomous individuals mutual aid and self governance while anarchism is most easily defined
by what it is against anarchists also offer positive visions of what they believe to be a truly free society however ideas
about how an anarchist society might work vary considerably especially with respect to economics there is also disagreement
 about how a free society might be brought about origins and predecessors kropotkin and others argue that before recorded
history human society was organized on anarchist principles most anthropologists follow kropotkin and engels in believing t
hat hunter gatherer bands were egalitarian and lacked division of labour accumulated wealth or decreed law
and had equal access to resources william godwin anarchists including the the anarchy organisation and rothbard find anarch
ist attitudes in taoism from ancient china kropotkin found similar ideas in stoic zeno of citium according to kropotkin zen
o repudiated the omnipotence of the state its intervention and regimentation and proclaimed the sovereignty of the moral la
w of the individual the anabaptists of one six th century europe are sometimes considered to be religious forerunners of mo
dern anarchism bertrand russell in his history of western philosophy writes that the anabaptists repudiated all law since t
hey held that the good man
```

- 运行算子后得到的结果有词向量文件和词映射文件。
- 词映射文件格式为：映射后的 ID：原单词。

```
0:fokine
1:mattered
2:intimately
3:reunion
4:harmonies
5:bone
6:glorifying
7:pentoxide
8:arkham
9:albescens
10:transfusion
11:peleus
12:racecars
13:onyx
14:industrialisation
15:mysteroid
16:been
17:fuller
18:modifies
19:ipa
20:kwong
21:pig
22:jade
23:crying
24:breath
25:battering
26:semites
27:accomplished
28:swain
29:ukrainian
```

- 词向量文件格式为：映射后的 ID：词向量。

```
0:5.890932E-4 0.0039711213 9.18858E-5 4.8313546E-4 -0.0035535127 0.0033942712 -8.548046E-5
0.001094785 -0.0010664434 -0.0020063713 -0.0018684975 -0.0034639365 -0.0022227308
-6.5268646E-4 0.0021918614 0.0037843545 6.349852E-4 -0.0037996734 -0.0032630798
9.685887E-5 4.3079612E-4 -0.0029240637 -0.0028981306 1.9581897E-4 0.0015903218
-4.669021E-4 -0.0024416482 0.0023292138 -0.0038829634 -0.003250387 -0.003386422
0.0029003124 0.002824246 -0.0023789285 -0.0035327068 9.579024E-5 0.0018456406 6.4591283E-4
-9.6570153E-4 5.8667467E-4 1.7671482E-4 0.0018733151 -0.0018923845 4.2529823E-4
0.0030963414 0.0033465486 0.0036233077 -0.0025648093 -0.002446964 -0.0036055946
0.0022110161 2.966985E-4 7.5042853E-4 -0.0032370735 -0.0027218119 -6.127785E-4
-0.0018945014 -0.002547899 6.5984833E-4 -0.001254839 0.0022504167 0.002252328 -0.002790278
0.0028389143 0.0022638151 0.0033603131 0.0021212026 3.6142385E-4 6.203796E-4 5.541262E-4
0.002161485 0.0028538373 -0.0017391599 0.0024538261 0.0031673256 4.3207666E-4
-4.4809043E-4 -0.0013000127 0.0035357084 -0.0019916636 -0.0019426235 0.0020503998
0.0011139599 -0.0011370436 -5.5532827E-4 -0.0017496068 -2.1123204E-4 0.0017121066
3.017349E-4 0.0020119825 -5.462527E-4 0.0011839543 0.0039800024 0.0025195205 0.0016699361
-2.6240083E-4 5.595205E-4 0.0024836515 0.0030871194 -0.0017066303 -0.0031065384
-1.8145161E-5 -6.145583E-4 5.410473E-4 0.0018261466 -5.059019E-4 -0.003784447 0.0011087982
0.0024840261 6.257725E-4 0.0028605137 0.0010251418 0.0031615957 -0.0012451047
-0.0023883507 4.8524578E-4 -0.0035458368 8.091281E-5 -0.003228977 2.6641597E-4
-0.001941703 0.0014182758 -2.5636618E-4 5.154852E-4 -9.58437E-4 -7.410791E-4 0.0010008282
-0.0014225176
1:-0.004024367 0.0038609765 -0.0024515798 -0.0013728965 0.0018704536 0.0031438328
0.0024150535 -0.00315246 -0.0016100466 0.0014629547 0.0016889522 -0.0027478538
-7.642371E-4 -9.747202E-4 -0.002067412 2.652061E-6 -0.0036517768 0.0023558848 6.51561E-4
0.00250819 0.0037633702 -0.0022672703 -8.236773E-4 0.0021630542 0.0026306782 -0.0025659492
0.002010112 0.0019294362 -0.0010794078 8.0198277E-4 0.0027767317 -0.0019157645
-0.0022618228 -5.0847436E-4 -6.764415E-4 7.448302E-4 -5.4385833E-4 0.0022963793
-3.5616924E-4 0.0034990378 -0.0014823454 -0.0011686712 -3.6209723E-4 0.0027484274
-0.0020010716 0.0032461027 -0.0020084667 -0.0032737674 -0.001871959 -0.003069962
8.655588E-6 -0.0024964819 0.0032511626 -0.0023781932 0.001520502 -9.5813023E-4 0.002325141
```

PySONA 算法

最近更新时间：2020-12-14 11:29:53

[Angel]LR on PyTONA

算法简介

LogisticRegression（LR）算法是一种常见的分类算法，因其模型简单、可解释性强等特点在工程领域得到广泛应用。本算子是 LR 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ 注意：
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**
- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。
  - 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
  - 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
  - PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。
- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务 。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 Pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 lr.py，执行如下命令：

```
python lr.py --input_dim 148
```

其中，input_dim 为输入数据的特征的维度，执行完该命令后会生成用于分布式训练的 lr.pt 模型文件，用户需手动上传到自己的 cos 路径。

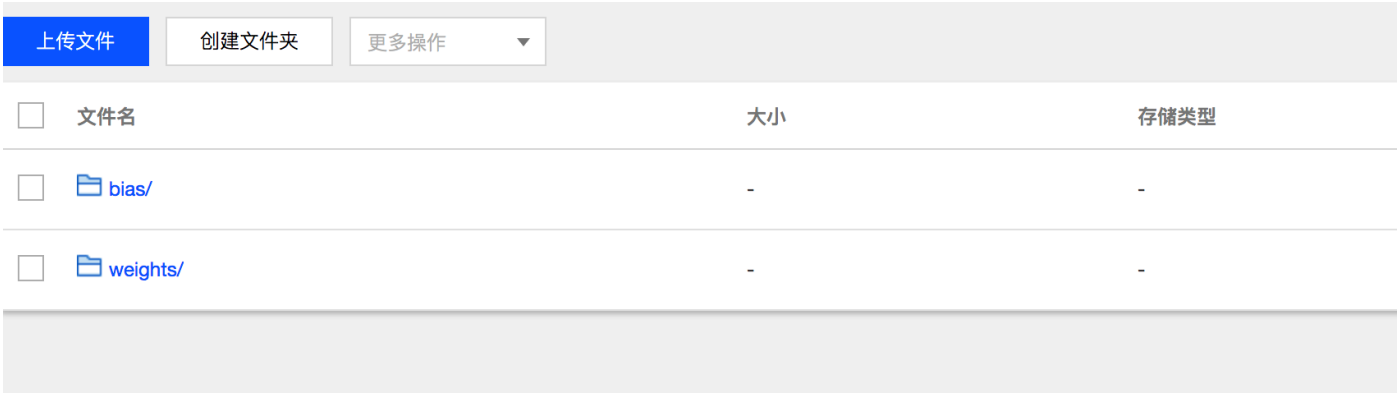
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决二分类问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]FM on PyTONA

算法简介

FactorizationMachine（FM）是一种基于矩阵分解的机器学习算法，它可对任意的实值向量进行预测。其主要优点包括：

（1）可用于高度稀疏数据场景。

（2）具有线性的计算复杂度。本算子是 FM 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**

  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
>
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。
  - 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。

- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 pytorch(1.3.1版本)，然后到 python/recommendation 下载算法文件 fm.py，执行如下命令：

```
python fm.py --input_dim 148 --embedding_dim 10
```

其中，input_dim 为输入数据的特征的维度，embedding_dim 为特征 embedding 向量的长度，执行完该命令用户会生成用于分布式训练的 fm.pt 模型文件，用户需手动上传到自己的 COS 路径。

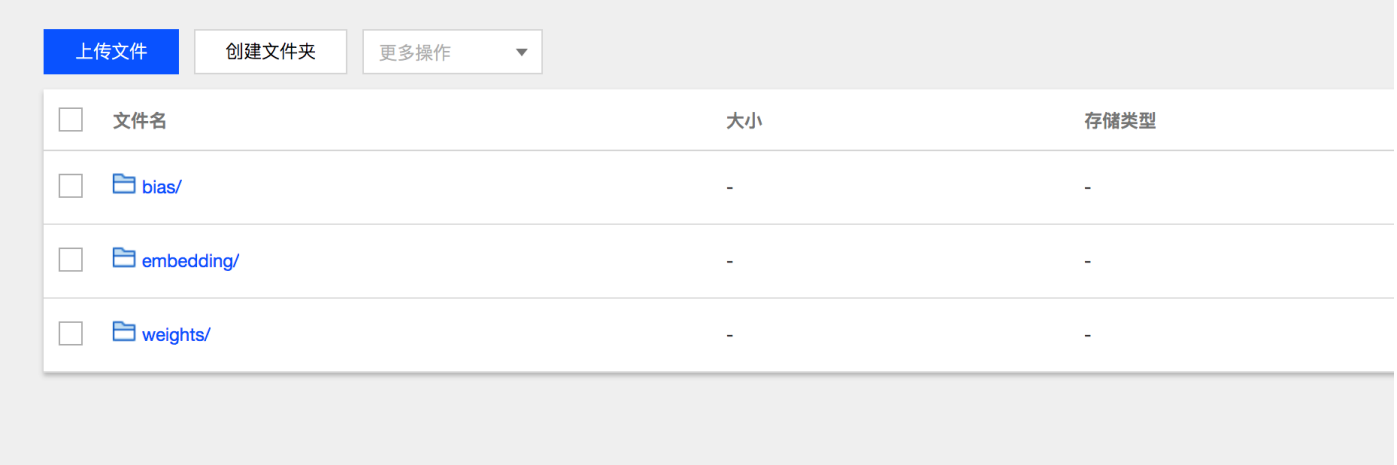
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]DeepFM on PyTONA

算法简介

DeepFM 算法是在 FM（Factorization machine）的基础上加入深度层构成。与 PNN，NFM 算法相比，它保留了 FM 的二阶隐式特征交叉的同时，又用深度网络来获取高阶特征交叉。本算子是 DeepFM 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm 格式: label index:value index:value index:value ......
libffm 格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
> index 是从1开始的, 以空格分隔；libffm 格式的 field 是从0开始的

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye为predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 Pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 deepfm.py，执行如下命令：

```
python deepfm.py --input_dim 148 --n_fields 13 --embedding_dim 10 --fc_dims 10 5 1
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，fc_dims为fc-layers 每层的节点数。执行完该命令会生成用于分布式训练的 deepfm.pt 模型文件，用户需手动上传到自己的 COS 路径。

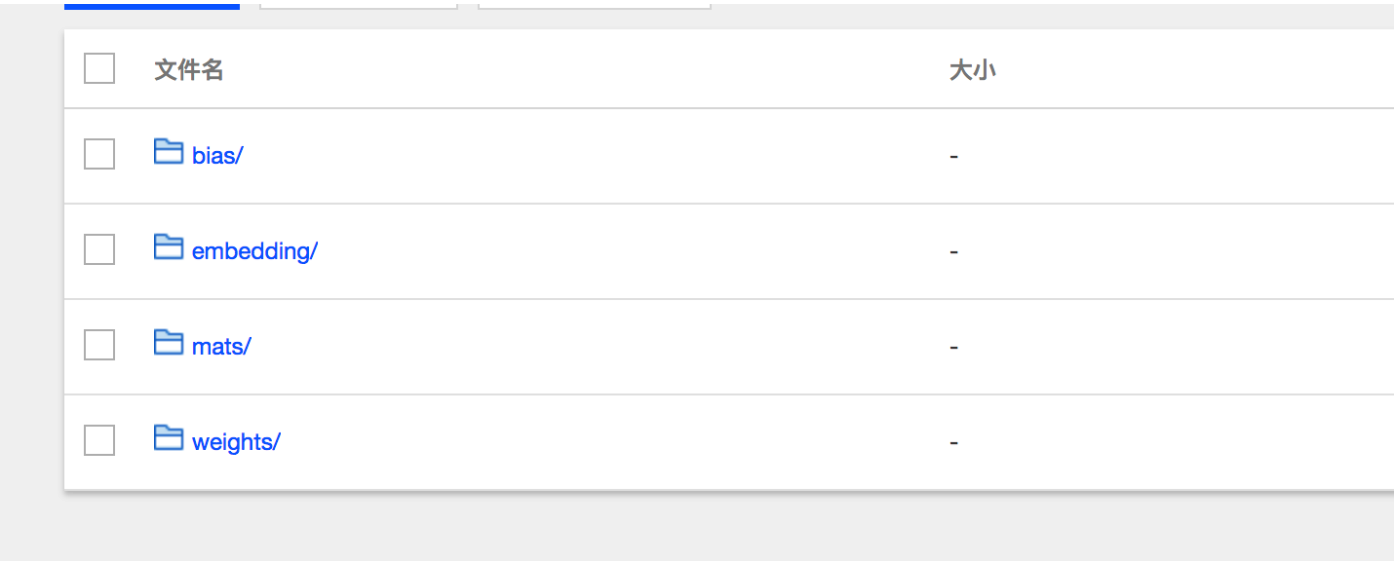
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练



[Angel]DeepAndWide on PyTONA

算法简介

DeepAndWide 算法是将 Embedding 的结果直接输入 DNN 进一步提取高阶特交叉，最后将一阶特征与高阶特征组合起来进行预测，本算子是 DeepAndWide 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 Pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 deepandwide.py，执行如下命令：

```
python deepandwide.py --input_dim 148 --n_fields 13 --embedding_dim 10 --fc_dims 10 5 1
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，fc_dims为fc-layers 每层的节点数。执行该命令后会生成用于分布式训练的 deepandwide.pt 模型文件，用户需手动上传到自己的 COS 路径。

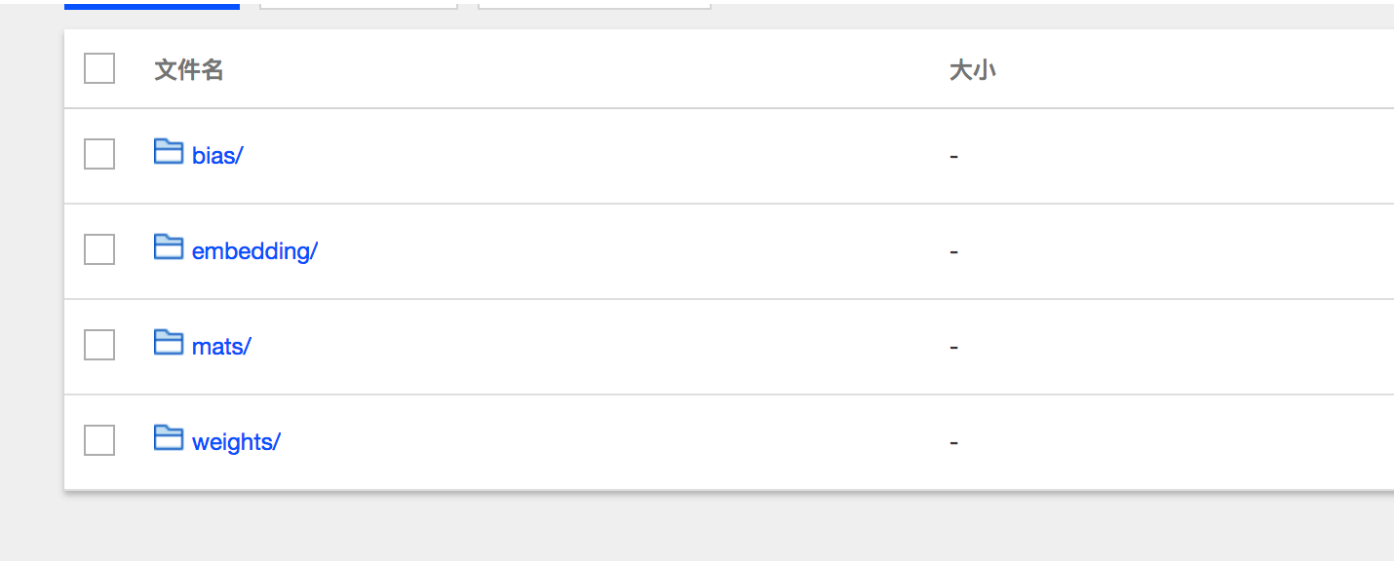
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]DCN on PyTONA

算法简介

DCN 由两个结构组成，一个是 cross network，一个是很基础的 deep network，而 stack layer 只是简单的拼接了 cross network 和 deep network 的输出结果。本算子是 DCN 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ 注意：
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 dcn.py，执行如下命令：

```
python dcn.py --input_dim 148 --n_fields 13 --embedding_dim 10 --cross_depth 3 --fc_dims 10 5
5
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，cross_depth 为 cross 网络的深度，fc_dims为fc-layers 每层的节点数。执行该命令后会生成用于分布式训练的 dcn.pt 模型文件，用户需手动上传到自己的 COS 路径。

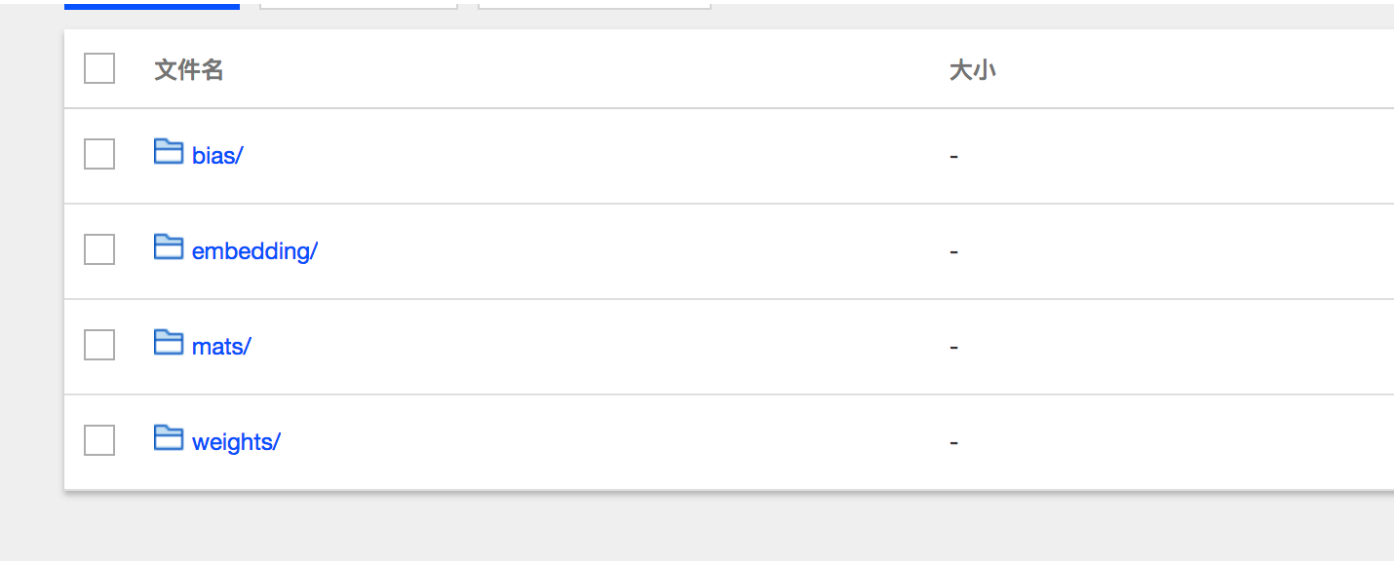
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]AttentionNet on PyTONA

算法简介

Attention 给模型赋予了区分辨别的能力，例如，在机器翻译、语音识别应用中，为句子中的每个词赋予不同的权重，使神经网络模型的学习变得更加灵活（soft）本算子是 AttentionNet 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 attention_net.py，执行如下命令：

```
python attention_net.py --input_dim 148 --n_fields 13 --embedding_dim 10 --fc_dims 10 5 1
```

其中，input_dim 为输入数据的特征的维度，n_fields为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，fc_dims 为 fc-layers 每层的节点数，执行该命令后会生成用于分布式训练的 attention_net.pt 模型文件，用户需手动上传到自己的 COS 路径。

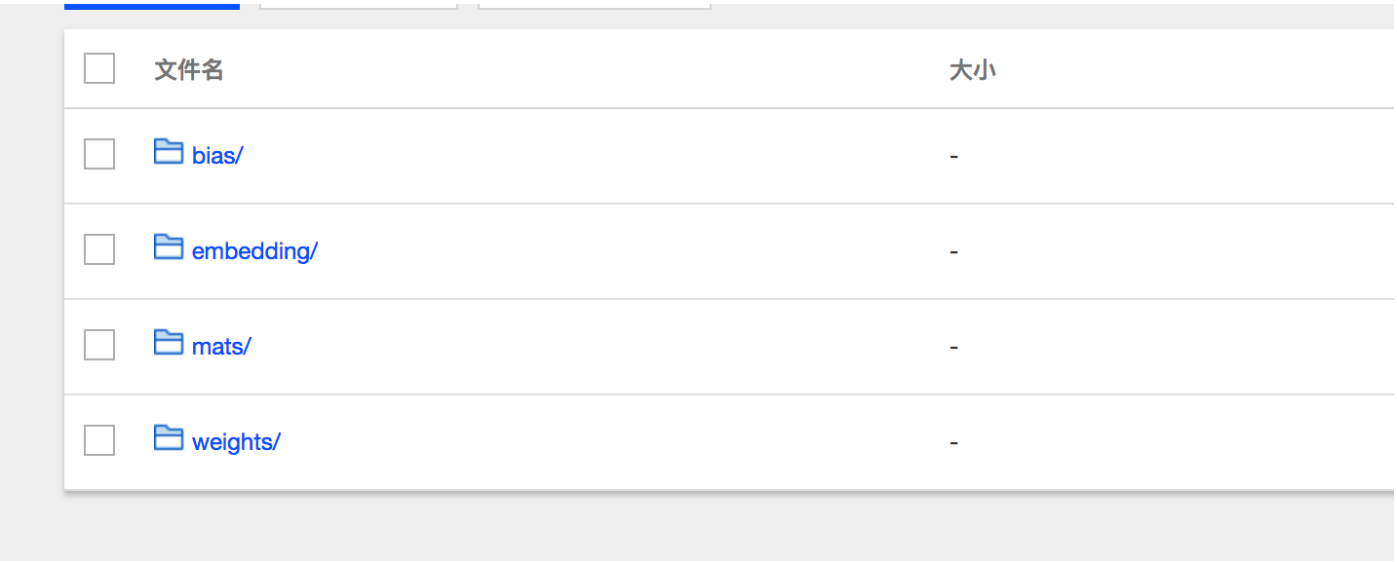
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]AttentionFM on PyTONA

算法简介

AttentionFM（Attentional Factorization Machine），和 NFM 是同一个作者。AFM 是在 FM 上的改进，它最大的特点就是使用一个 attention network 来学习不同组合特征的重要性
本算子是 AttentionFM 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 attention_fm.py，执行如下命令：

```
python attention_fm.py --input_dim 148 --n_fields 13 --embedding_dim 10 --attention_dim 10
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，attention_dim为attention 的维度，执行该命令用户会生成用于分布式训练的 attention_fm.pt模型文件，用户需手动上传到自己的 COS 路径。

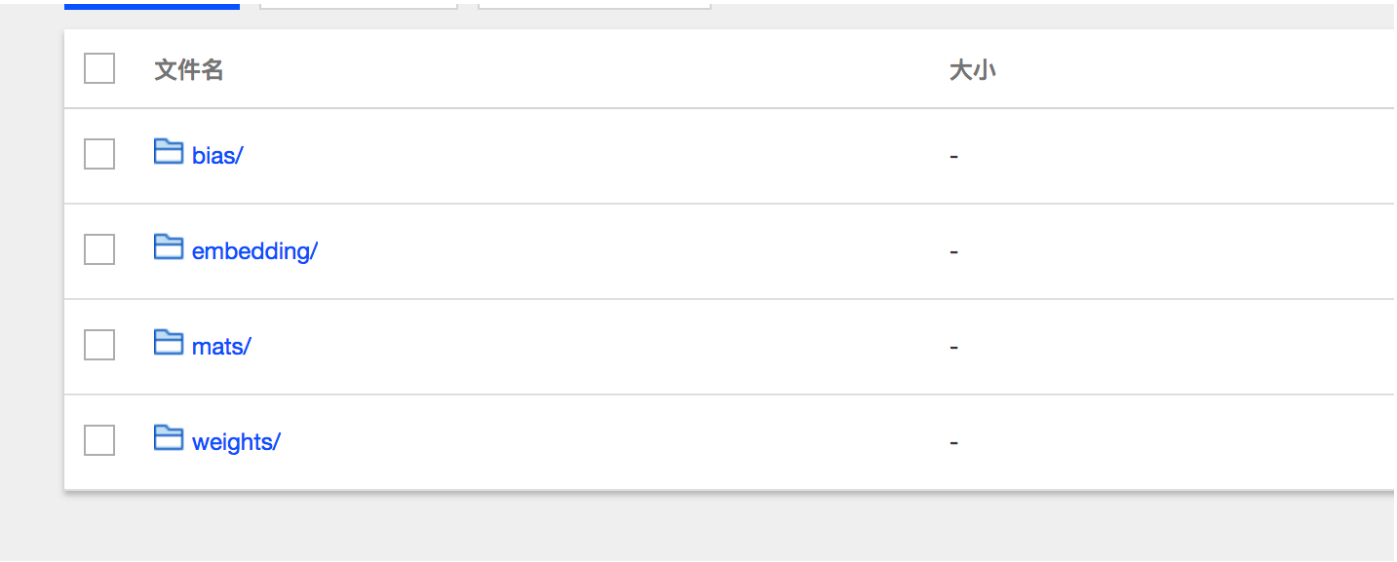
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]xDeepFM on PyTONA

算法简介

为了实现自动学习显式的高阶特征交互，同时使得交互发生在向量级上，xDeepFM 提出了一种新的名为压缩交互网络（Compressed Interaction Network，简称 CIN）的神经模型。
本算子是 xDeepFM 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ **注意：**
> index 是从1开始的, 以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。
  - 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。

- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数
  - batchSize：训练用的 mini batch 大小
  - actionType：训练或者预测任务
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型

用户需在自己本地安装 pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 xdeepfm.py，执行如下命令生成可用于分布式训练的 Pytorch 模型文件：

```
python xdeepfm.py --input_dim 148 --n_fields 13 --embedding_dim 10 --fc_dims 10 5 5 --cin_dims 5 5
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，fc_dims 为 fc-layers 每层的节点数，cin_dims 为 cin 网络每层的节点数。执行该命令后会生成用于分布式训练的 xdeepfm.pt 模型文件，用户需手动上传到自己的 cos 路径。

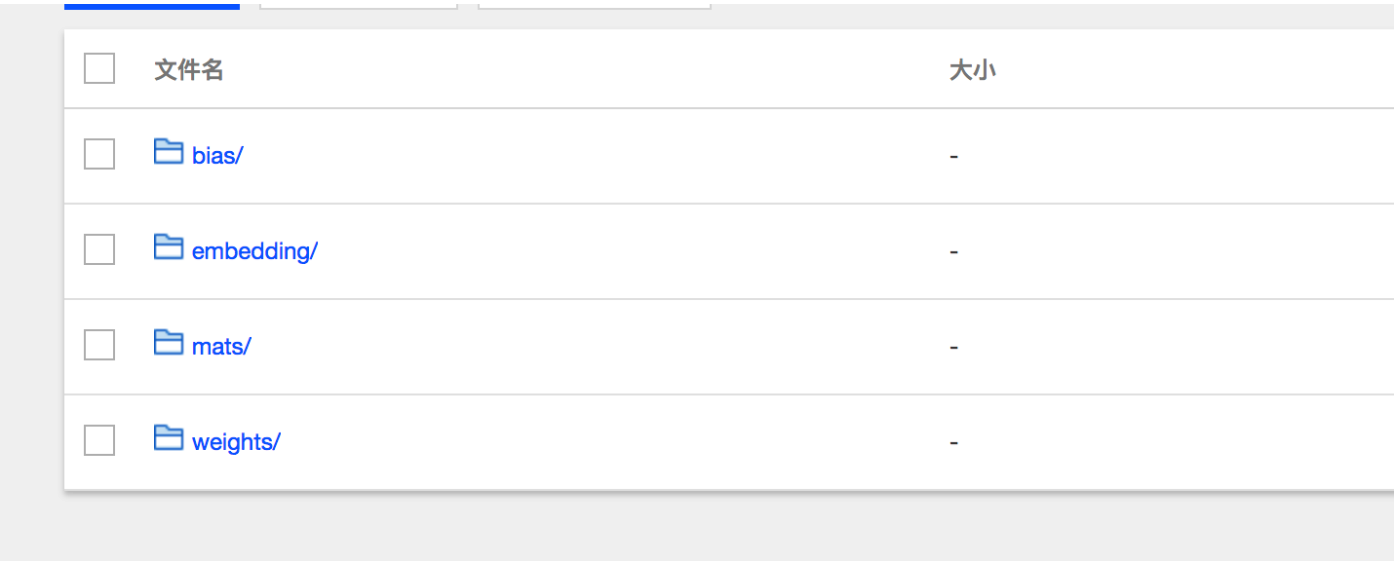
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



[Angel]PNN on PyTONA

算法简介

PNN（Product-Based Neural Networks）算法是在 Embedding 的基础上，对 Embedding 的结果进行两两内积或外积，然后将内/外积结果与原始的 Embedding 结果拼接起来输入 DNN 进一步提取高阶特交叉，值得注意的是，PNN 并没有放弃一阶特征，最后将一阶特征与高阶特征组合起来进行预测，本算子是 PNN 算法在 Pyorch on Angel 的实现。

参数说明

- **输入数据格式**
  目前支持 libsvm、libffm 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
libffm格式: label field:index:value field:index:value field:index:value ......
```

> ⚠ 注意：
> index 是从1开始的，以空格分隔；libffm 格式的 field 是从0开始的。

- **节点**

- 算法 IO 参数
  - 数据输入路径：训练数据输入路径。
  - 验证数据输入路径：actionTpye 为 train 时，该参数生效。

- 模型保存路径：训练完后 angel 模型保存路径，可用于批量预测或增量训练。
- 模型加载路径：actionTpye 为 train 时，表示增量训练预加载的模型路径，为 predict 是表示预测模型加载路径。
- 预测结果输出路径：actionTpye 为 predict 时该参数生效，表示预测结果存储路径。
- PyTorch 模型文件保存路径：训练好的 Pytorch 模型保存路径，可用于在线 serving。

- 算法参数
  - numEpoch：训练总的迭代轮数。
  - batchSize：训练用的 mini batch 大小。
  - actionType：训练或者预测任务。
  - PyTorch 模型文件：用户使用 Python 脚本生成的 Pytorch 脚本模型。

用户需在自己本地安装 pytorch（1.3.1版本），然后到 python/recommendation 下载算法文件 pnn.py，执行如下命令：

```
python pnn.py --input_dim 148 --n_fields 13 --embedding_dim 10 --fc_dims 10 5 1
```

其中，input_dim 为输入数据的特征的维度，n_fields 为输入数据 field 的个数，embedding_dim 为特征 embedding 向量的长度，fc_dims 为 fc-layers 每层的节点数。执行该命令后会生成用于分布式训练的 pnn.pt 模型文件，用户需手动上传到自己的 COS 路径。

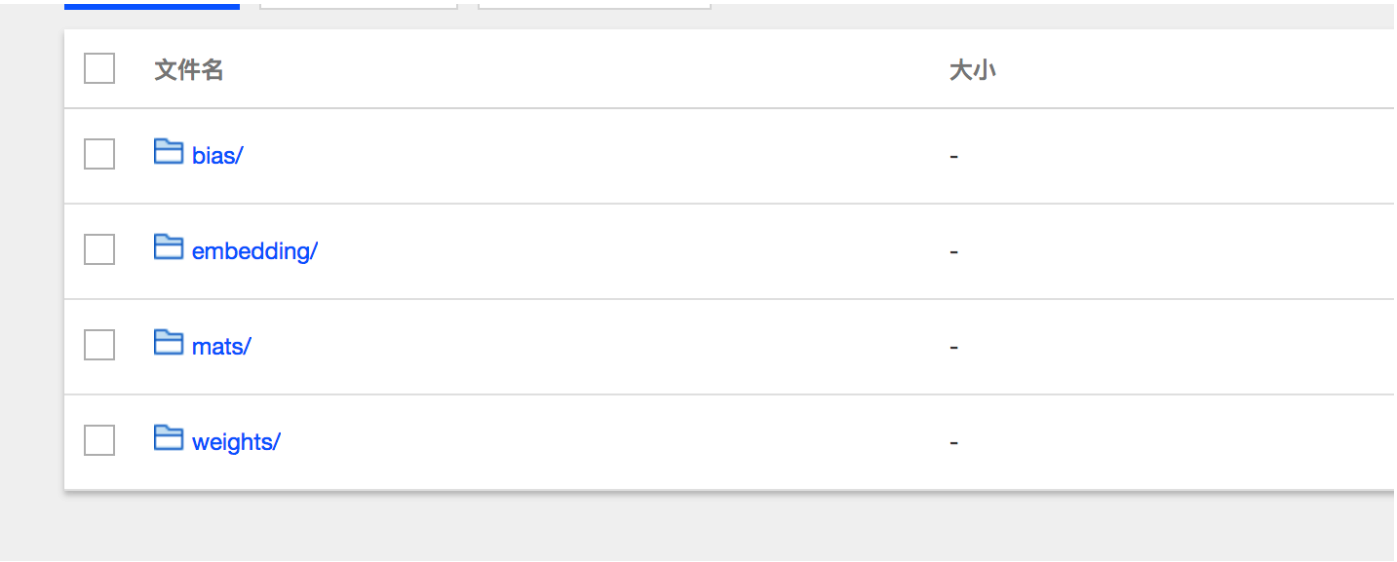
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

本算子主要解决的是二分类的问题，训练的模型可用于做分类预测

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后会得到分类模型，可用于预测或再训练。



机器学习算法

最近更新时间：2020-07-31 15:30:25

[Angel]LR on SonA

算法简介

LogisticRegression（LR）算法是一种常见的分类算法，因其模型简单、可解释性强等特点在工程领域得到广泛应用。

参数说明

- **输入数据格式**
  目前支持 libsvm、dummy 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
dummy格式: index,index,index,......
```

> ⚠ 注意：
> libsvm 格式的 index 是从1开始的，以空格分隔，dummy 格式的 index 是从0开始的，以逗号分隔。

- **训练节点**
- 算法 IO 参数
  - 数据输入路径：训练数据存储 COS 路径。
    带预测的算子模块模型保存路径不需要用户填写，平台会自动生成。
- 算法参数
  - 训练迭代次数：模型训练迭代的轮数。
  - 学习率：模型训练初始学习率，默认为0.5。
  - 特征数目：数据特征个数（最大的 featureID 值+1）。
  - validate 数据集比例：每次 validation 的样本比率，设为0时不做 validation。
  - 模型类型：模型的数据类型。
  - 学习率衰减速度：学习速率衰减系数, 默认值为 0.1。
  - 正则项系数：L2惩罚项系数。
  - 每个 mini-batch 样本采样率：每个 mini-batch 样本采样率，1.0表示不采样。
  - 数据标签转换类：数据标签转换类。

- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。
- **预测节点**
  - 模型 IO 参数
    - 数据输入路径：预测数据输入路径，cos 路径。
  - 模型参数
    - 特征数目：数据特征个数（最大的 featureID 值+1）。
    - 资源参数：与训练节点意义相同。

本算子主要解决了二分类问题

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后模型保存路径下会生成分类模型。

文件名	大小	存储类型
input_bias/	-	-
input_weight/	-	-
.graph.json.crc	16B	标准存储
graph.json	850B	标准存储

- 使用分类模型进行新的数据预测，得到分类结果，分类结果展示的是每个类别的概率。

```
-1 0.44907671213150024
-1 0.46622851490974426
-1 0.04085632786154747
-1 0.09479375928640366
-1 0.7462939620018005
-1 0.8910619020462036
-1 7.328957435674965E-4
+1 0.5286999940872192
+1 0.5334902405738831
+1 0.9565372467041016
+1 0.7629945874214172
+1 0.3772147595882416
-1 0.0071544889360666275
-1 0.23927532136440277
+1 0.28980204463005066
-1 0.06192959472537041
-1 0.005914544221013784
-1 0.021107962355017662
-1 0.2649117708206177
+1 0.20948977768421173
+1 0.9262672066688538
-1 0.004387388937175274
-1 0.04830632358789444
-1 0.3640007972717285
-1 0.02431498095393181
+1 0.5481274724006653
-1 0.009707165881991386
+1 0.303445041179657
-1 0.2564375400543213
-1 0.28424742817878723
```

[Angel]FM on SonA

算法简介

FactorizationMachine（FM）一种基于矩阵分解的机器学习算法。它可对任意的实值向量进行预测。其主要优点包括：

（1）可用于高度稀疏数据场景。

（2）具有线性的计算复杂度。

参数说明

- **输入数据格式**
  目前支持 libsvm、dummy 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
dummy格式: index,index,index,.....
```

> ⚠ 注意：
> libsvm 格式的 index 是从1开始的，以空格分隔；dummy 格式的 index 是从0开始的，以逗号分隔。

- **训练节点**
- 算法 IO 参数
  - 数据输入：训练数据存储 cos 路径。
  - validate数据集比例：每次 validation 的样本比率，设为0时不做 validation。
  - ml.data.type：数据格式，支持 dummy、libsvm。

> ⚠ 注意：
> 带预测的算子模块模型保存路径不需要用户填写，平台会自动生成。

- 算法参数
  - 迭代次数：模型训练迭代的轮数。
  - 学习率：模型训练初始学习率，默认为 0.05。
  - FM 因子维度：embedding 维度。
  - 正则项参数：L2惩罚项系数。
  - 模型类型：模型的数据类型。
  - 特征数目：数据特征个数（最大的 featureID 值+1）。
  - 学习率衰减速度：学习速率衰减系数，默认值为 0.001。
  - 一个 epoch 中更新参数的次数：在实现时为了加速，每个 minibatch 只更本地参数，多个 minibatch 后会更新一次 PS 上的参数，该参数用于指定一个 epoch 中更新 PS 参数的次数。
- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。

- master 节点资源类型：请选择合适的 master 节点机型。
- ps 节点资源类型：请选择合适的 ps 节点机型。
- spark conf 参数
  - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
  - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

- **预测节点**

- 模型 IO 参数
  - 输入数据：预测数据输入路径，cos 路径。
  - 预测结果保存路径：预测结果保存路径，cos 路径。

- 模型参数
  - 特征数目：数据特征个数（最大的 featureID 值+1）。
  - ml.data.type：数据格式，支持 dummy、libsvm。

- 资源参数：与训练节点意义相同。

本算子主要解决二分类问题

- 原始数据如下图所示：

```
-1 3:1 11:1 14:1 19:1 39:1 42:1 55:1 64:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 14:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 3:1 6:1 17:1 22:1 36:1 41:1 53:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 17:1 21:1 35:1 40:1 53:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 2:1 6:1 18:1 19:1 39:1 40:1 52:1 61:1 71:1 72:1 74:1 76:1 80:1 95:1
-1 3:1 6:1 18:1 29:1 39:1 40:1 51:1 61:1 67:1 72:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 16:1 26:1 35:1 45:1 49:1 64:1 71:1 72:1 74:1 76:1 78:1 101:1
+1 5:1 7:1 17:1 22:1 36:1 40:1 51:1 63:1 67:1 73:1 74:1 76:1 81:1 83:1
+1 2:1 6:1 14:1 29:1 39:1 42:1 52:1 64:1 67:1 72:1 75:1 76:1 82:1 83:1
+1 4:1 6:1 16:1 19:1 39:1 40:1 51:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
+1 3:1 6:1 18:1 20:1 37:1 40:1 51:1 63:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 2:1 11:1 15:1 19:1 39:1 40:1 52:1 63:1 68:1 73:1 74:1 76:1 80:1 90:1
-1 1:1 6:1 15:1 19:1 39:1 42:1 55:1 62:1 67:1 72:1 74:1 76:1 78:1 83:1
-1 2:1 6:1 17:1 24:1 38:1 42:1 50:1 64:1 71:1 73:1 74:1 76:1 82:1 83:1
+1 3:1 6:1 15:1 25:1 38:1 40:1 48:1 63:1 68:1 73:1 74:1 76:1 80:1
-1 3:1 6:1 17:1 27:1 35:1 40:1 57:1 63:1 69:1 73:1 74:1 76:1 81:1 103:1
-1 1:1 7:1 16:1 22:1 36:1 42:1 56:1 62:1 67:1 73:1 74:1 76:1 79:1 83:1
-1 2:1 6:1 16:1 22:1 36:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 3:1 6:1 14:1 21:1 35:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
+1 4:1 7:1 18:1 29:1 39:1 41:1 51:1 66:1 67:1 72:1 74:1 76:1 81:1 83:1
+1 3:1 6:1 16:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 5:1 6:1 18:1 22:1 36:1 43:1 49:1 66:1 71:1 72:1 74:1 76:1 78:1 83:1
-1 3:1 9:1 14:1 26:1 35:1 40:1 56:1 63:1 71:1 73:1 74:1 76:1 80:1 83:1
-1 4:1 6:1 15:1 21:1 35:1 40:1 57:1 63:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 5:1 6:1 15:1 22:1 36:1 41:1 47:1 66:1 67:1 72:1 74:1 76:1 80:1 83:1
+1 5:1 10:1 17:1 19:1 39:1 40:1 47:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 16:1 22:1 36:1 42:1 48:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
+1 5:1 16:1 20:1 37:1 40:1 63:1 68:1 73:1 74:1 76:1 82:1 93:1
-1 3:1 6:1 18:1 22:1 36:1 41:1 51:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 4:1 6:1 16:1 22:1 36:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 10:1 16:1 24:1 38:1 42:1 59:1 64:1 67:1 73:1 74:1 76:1 82:1 83:1
-1 1:1 6:1 18:1 20:1 37:1 42:1 50:1 62:1 71:1 73:1 74:1 76:1 81:1 83:1
-1 4:1 6:1 18:1 19:1 39:1 41:1 51:1 62:1 67:1 73:1 74:1 77:1 80:1 83:1
-1 2:1 9:1 14:1 20:1 37:1 40:1 55:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 11:1 18:1 20:1 37:1 40:1 49:1 63:1 71:1 73:1 74:1 76:1 78:1 83:1
-1 4:1 6:1 17:1 21:1 35:1 42:1 54:1 66:1 67:1 73:1 74:1 76:1 80:1 86:1
-1 1:1 6:1 17:1 20:1 37:1 42:1 54:1 62:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 1:1 6:1 18:1 22:1 36:1 46:1 55:1 61:1 67:1 72:1 74:1 76:1 78:1 83:1
+1 2:1 6:1 14:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 74:1 76:1 79:1
```

- 训练完成后模型保存路径下会生成分类模型。

文件名	大小	存储类型
embedding_embedding/	-	-
wide_bias/	-	-
wide_weight/	-	-
.graph.json.crc	20B	标准存储
graph.json	1.26KB	标准存储

- 使用分类模型进行新的数据预测，得到分类结果，分类结果展示的是每个类别的概率。

```
+1 0.5892754197120667
+1 0.3882182240486145
+1 0.5647069215774536
+1 0.540843665599823
-1 0.4810137152671814
+1 0.4873451292514801
-1 0.8671833276748657
-1 0.4757355749607086
-1 0.25080475211143494
-1 0.5218862295150757
-1 0.2079264372587204
-1 0.16579324007034302
-1 0.593603789806366
+1 0.5182967185974121
-1 0.20008273422718048
-1 0.5195870399475098
-1 0.1958072930574417
-1 0.18222826719284058
-1 0.5453251004219055
+1 0.6981514096260071
-1 0.3687385320663452
+1 0.4956549406051636
-1 0.4568973481655121
-1 0.22829033434391022
+1 0.4939494729042053
-1 0.5450286865234375
-1 0.20025043189525604
-1 0.4924597442150116
+1 0.8603235483169556
-1 0.6111857295036316
```

[Angel]GBDT on SONA

算法简介

GBDT（Gradient Boosting Decision Tree）：梯度提升决策树是一种集成使用多个弱分类器（决策树）来提升分类效果的机器学习算法，在很多分类和回归的场景中，都有不错的效果。

参数说明

- **输入数据格式**
  目前支持 libsvm、dummy 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
dummy格式: index,index,index,.....
```

> ⚠ 注意：
> libsvm 格式的 index 是从1开始的，以空格分隔；dummy 格式的 index 是从0开始的，以逗号分隔。

- **训练节点**

- 算法 IO 参数
  - 训练数据路径：训练数据存储 cos 路径。
  - 验证数据路径：验证数据存储 cos 路径。

- 算法参数
  - 任务类型：分类，回归任务可选。
  - 损失函数：二分类 logistic， 多分类 logistic，均方根误差（用于回归任务）。
  - 评估指标：二分类 AUC（仅二分类使用），分类错误率，多分类交叉熵，均方根误差（用于回归任务）。
  - 树的数量：构建的树的个数（轮数）。
  - 候选分裂点数量：每个特征可分成的候选分裂点的个数。
  - 树的最大深度：每棵树的最大深度。
  - 学习速度：默认为0.1。
  - 类别数量：分类任务的类别数。
  - 特征数量：输入数据特征数+1。
  - 特征下采样比例：取值范围（0,1），为1时默认不采样。
  - 多分类策略：多分类训练策略，one-tree（一轮一棵树）或者 multi-tree（一轮多棵树），多分类时可选每轮建单棵树或多棵树（其中多棵树的输的个数是指类别个数）。
  - 树节点上训练数据的最少个数：每个树节点上最少的样本数。
  - 最大叶子权重：为0时表示无限制。
  - 节点最小权重：节点最小的权重。
  - 最小分裂损失：效果同 xgboost 中的 gamma。

- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。

- spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
- driver 节点资源类型：请选择合适的 drive 节点机型。
- executor 节点资源类型：请选择合适的 executor 节点机型。
- master 节点资源类型：请选择合适的 master 节点机型。
- ps 节点资源类型：请选择合适的 ps 节点机型。
- spark conf 参数
  - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
  - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

- **预测节点**

- 模型 IO 参数和模型参数。
  - 预测数据路径：预测数据输入 cos 路径。
  - 预测结果保存地址：预测结果保存 cos 路径。

- 资源参数：与训练节点意义相同。

本算子主要解决二分类或者多分类的问题

- 原始数据如下图所示：

```
1 3:1 10:1 11:1 21:1 30:1 34:1 36:1 40:1 41:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 3:1 10:1 20:1 21:1 23:1 34:1 36:1 39:1 41:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 1:1 10:1 19:1 21:1 24:1 34:1 36:1 39:1 42:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
1 3:1 9:1 19:1 21:1 30:1 34:1 36:1 40:1 42:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 3:1 10:1 14:1 22:1 29:1 34:1 37:1 39:1 41:1 54:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 3:1 9:1 20:1 21:1 23:1 34:1 36:1 39:1 42:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 1:1 10:1 19:1 21:1 23:1 34:1 36:1 39:1 45:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
1 3:1 9:1 19:1 21:1 30:1 34:1 36:1 40:1 48:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 1:1 10:1 20:1 21:1 23:1 34:1 36:1 39:1 45:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 3:1 9:1 20:1 21:1 24:1 34:1 36:1 39:1 45:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 3:1 9:1 20:1 21:1 23:1 34:1 36:1 39:1 42:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 1:1 10:1 20:1 21:1 23:1 34:1 36:1 39:1 51:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 3:1 7:1 11:1 22:1 29:1 34:1 37:1 39:1 42:1 54:1 58:1 65:1 66:1 77:1 86:1 88:1 92:1 95:1 9
0 6:1 7:1 14:1 22:1 29:1 34:1 36:1 40:1 41:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
0 4:1 7:1 19:1 22:1 29:1 34:1 37:1 39:1 41:1 54:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 9
1 3:1 10:1 11:1 21:1 30:1 34:1 36:1 40:1 42:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
1 3:1 9:1 19:1 21:1 30:1 34:1 36:1 40:1 42:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1 1
1 3:1 10:1 11:1 21:1 30:1 34:1 36:1 40:1 41:1 53:1 58:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
0 1:1 10:1 20:1 21:1 23:1 34:1 36:1 39:1 41:1 53:1 56:1 65:1 69:1 77:1 86:1 88:1 92:1 95:1
```

- 训练完成后模型保存路径下会生成特征重要度文件和分类模型文件。

- 使用分类模型进行新的数据预测，得到分类结果，分类结果形式为：原始 label 预测 score 值。

```
0 0 -0.8525053
1 1 0.8613856
0 0 -0.8525053
0 0 -0.8525053
0 0 -0.8490381
0 0 -0.7985194
1 1 0.8613856
0 0 -0.8525053
1 1 0.8613856
0 0 -0.8327054
1 1 0.8613856
0 0 -0.8525053
0 0 -0.8490381
0 0 -0.8525053
0 0 -0.8490381
0 0 -0.8327054
0 0 -0.8525053
1 1 0.8613856
0 0 -0.8490381
0 0 -0.8525053
0 0 -0.8525053
0 0 -0.8327054
0 0 -0.8525053
0 0 -0.8525053
0 0 -0.8327054
1 1 0.8613856
0 0 -0.7985194
0 0 -0.8525053
0 0 -0.8525053
0 0 -0.8490381
```

[Angel]FTRL-LR on SONA

算法简介

FTRL 是一种在线优化算法，这种优化算法不光更能适应海量数据的要求，同时还能比较轻松的学习到一个有效且稀疏的模型，自问世以来在学术界和工业界都倍受关注和好评。
我们在 Spark on Angel 平台实现了以 FTRL 进行优化的分布式 LR 算法。

参数说明

- **输入数据格式**
  目前支持“libsvm”、“dummy”两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
dummy格式: index,index,index,.....
```

> ⚠ **注意：**
> libsvm 格式的 index 是从1开始的，以空格分隔；dummy 格式的 index 是从0开始的，以逗号分隔。

- **训练节点**
- 算法 IO 参数
  - 数据的输入路径：训练数据存储 cos 路径。
  - 输入数据的维度：数据特征个数（最大的 featureID 值+1）。
  - 迭代轮数：模型训练迭代的轮数。
  - batchSize：每个 batch 的样本数。
  - 模型加载地址：预训练的模型加载路径，cos 路径。
  - 任务类型：训练。
  - 输入数据格式：libsvm 或者 dummy。

> ⚠ 注意：
> 带预测的算子模块模型保存路径不需要用户填写，平台会自动生成。

- 算法参数
  - alpha：w 更新公式中的 alpha。
  - beta：w 更新公式中的 beta。
  - lambda1：w 更新公式中的 lambda1。
  - lambda2：w 更新公式中的 lambda2。

- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

- **预测节点**

- 模型 IO 参数
  - 输入数据的维度：数据特征个数（最大的 featureID 值+1）。
  - 数据的输入数据：预测数据输入路径，cos 路径。
  - 预测结果保存路径：预测结果保存路径，cos 路径。

- 输入数据格式：libsvm 或者 dummy。
- 数据是否有标签：预测数据是否带标签，true 或者 false。
- 任务类型：预测。

- 资源参数：与训练节点意义相同。

本算子主要解决了二分类问题

- 原始数据如下图所示：

```
0 4:1 10:1 19:1 40:1 44:1 51:1 66:1 72:1 77:1 81:1 94:1 100:1 107:1
0 6:1 11:1 19:1 40:1 45:1 52:1 67:1 72:1 77:1 79:1 94:1 99:1 107:1
0 4:1 12:1 20:1 37:1 46:1 53:1 66:1 72:1 77:1 79:1 94:1 100:1 107:1
0 7:1 12:1 21:1 36:1 45:1 53:1 67:1 73:1 77:1 79:1 94:1 100:1 107:1
0 2:1 12:1 19:1 40:1 45:1 54:1 68:1 73:1 78:1 79:1 94:1 100:1 108:1
0 4:1 12:1 22:1 41:1 45:1 52:1 68:1 72:1 78:1 79:1 94:1 100:1 107:1
0 6:1 12:1 23:1 35:1 47:1 55:1 66:1 73:1 78:1 79:1 94:1 99:1 109:1
1 7:1 11:1 20:1 37:1 45:1 52:1 67:1 72:1 77:1 79:1 94:1 101:1 107:1
1 3:1 12:1 22:1 41:1 44:1 54:1 66:1 72:1 78:1 89:1 94:1 101:1 107:1
1 5:1 12:1 19:1 40:1 45:1 52:1 67:1 72:1 77:1 84:1 94:1 100:1 107:1
1 4:1 12:1 24:1 38:1 45:1 52:1 67:1 73:1 77:1 79:1 94:1 104:1 107:1
1 2:1 10:1 19:1 40:1 45:1 54:1 67:1 74:1 77:1 79:1 94:1 100:1 110:1
0 1:1 12:1 19:1 40:1 44:1 51:1 69:1 72:1 78:1 79:1 94:1 100:1 107:1
0 3:1 12:1 25:1 39:1 44:1 56:1 66:1 73:1 77:1 79:1 94:1 101:1 107:1
1 4:1 12:1 26:1 38:1 45:1 57:1 67:1 74:1 77:1 79:1 94:1 100:1 111:1
0 3:1 12:1 27:1 35:1 45:1 58:1 67:1 75:1 77:1 79:1 94:1 101:1 112:1
0 1:1 11:1 20:1 37:1 44:1 59:1 69:1 72:1 77:1 79:1 94:1 100:1 107:1
0 3:1 12:1 20:1 37:1 44:1 60:1 70:1 72:1 77:1 79:1 94:1 100:1 107:1
0 4:1 12:1 21:1 36:1 45:1 56:1 67:1 72:1 77:1 79:1 94:1 101:1 107:1
```

- 训练完成后模型保存路径下会生成分类模型。

文件名	大小	存储类型
back/	-	-
weight/	-	-

- 使用分类模型进行新的数据预测，得到分类结果，分类结果展示的是每个样本属于 label 类的概率。

```
-1.0 0.3419125542504
-1.0 0.38359382547300375
-1.0 0.018408318476705897
-1.0 0.06658325965624243
-1.0 0.6752134409759735
-1.0 0.8543909787580126
-1.0 0.001781148657691602
1.0 0.3965742371799506
1.0 0.41525546019306786
1.0 0.9444405599912905
1.0 0.6404916004498611
1.0 0.3155890391702023
-1.0 0.0025977153946314664
-1.0 0.04688815982657244
1.0 0.2499093859366886
-1.0 0.018902131436522255
-1.0 8.20684237451898E-4
-1.0 0.009407662011386627
-1.0 0.17514926971926495
1.0 0.22869286143817608
1.0 0.8905396640946988
-1.0 0.0036897059706793617
-1.0 0.03729956634018075
-1.0 0.3628251168938015
-1.0 0.03232311209756578
1.0 0.6774610219029031
-1.0 0.0035985976301669336
1.0 0.26610164425527427
-1.0 0.16190011624835868
-1.0 0.27298994759153083
```

[Angel]FTRL-FM on SONA

算法简介

FTRL 是一种在线优化算法，这种优化算法不光更能适应海量数据的要求，同时还能比较轻松的学习到一个有效且稀疏的模型，自问世以来在学术界和工业界都倍受关注和好评。

我们在 Spark on Angel 平台实现了以 FTRL 进行优化的分布式 FM 算法。

参数说明

- **输入数据格式**

  目前支持 libsvm、dummy 两种数据格式，分别如下所示：

```
libsvm格式: label index:value index:value index:value ......
dummy格式: index,index,index,.....
```

> ⚠ **注意：**
>
> libsvm 格式的 index 是从1开始的，以空格分隔；dummy 格式的 index 是从0开始的，以逗号分隔。

- **训练节点**
- 算法 IO 参数
  - 数据的输入路径：训练数据存储 cos 路径。
  - 输入数据的维度：数据特征个数（最大的 featureID 值+1）。
  - 迭代轮数：模型训练迭代的轮数。
  - batchSize：每个 batch 的样本数。
  - 模型加载地址：预训练的模型加载路径，cos 路径（如果没有预训练的模型，则为空）。
  - 任务类型：训练。
  - 输入数据格式：libsvm 或者 dummy。

> ⚠ 注意：
> 带预测的算子模块模型保存路径不需要用户填写，平台会自动生成。

- 算法参数
  - alpha：w 更新公式中的 alpha。
  - beta：w 更新公式中的 beta。
  - lambda1：w 更新公式中的 lambda1。
  - lambda2：w 更新公式中的 lambda2。
  - 因子数：因子分解超参。

- 资源参数
  - num-executors：任务启动的 spark executor 个数，可根据数据量来配置，一般训练数据量越大，需要的 worker 个数越多。
  - spark.ps.instances：Angel ps 个数，可根据模型大小来配置，一般模型越大，需要的 PS 个数越多。
  - driver 节点资源类型：请选择合适的 drive 节点机型。
  - executor 节点资源类型：请选择合适的 executor 节点机型。
  - master 节点资源类型：请选择合适的 master 节点机型。
  - ps 节点资源类型：请选择合适的 ps 节点机型。
  - spark conf 参数
    - spark.angel.tmp.output.path.prefix：angel 临时目录的前缀路径，为 COS 路径。
    - saprk.angel.output.path.deleteonexist：为了防止误删除模型，默认不自动删除模型输出路径的文件。如果需要，可设置为 true。

- **预测节点**

- 模型 IO 参数
  - 输入数据的维度：数据特征个数（最大的 featureID 值+1）。
  - 数据的输入数据：预测数据输入路径，cos 路径。

- 预测结果保存路径：预测结果保存路径，cos 路径。
- 输入数据格式：libsvm 或者 dummy。
- 数据是否有标签：预测数据是否带标签，true 或者 false。
- 任务类型：预测。

- 资源参数：与训练节点意义相同。

本算子主要解决了二分类问题

- 原始数据如下图所示：

```
0 4:1 10:1 19:1 40:1 44:1 51:1 66:1 72:1 77:1 81:1 94:1 100:1 107:1
0 6:1 11:1 19:1 40:1 45:1 52:1 67:1 72:1 77:1 79:1 94:1 99:1 107:1
0 4:1 12:1 20:1 37:1 46:1 53:1 66:1 72:1 77:1 79:1 94:1 100:1 107:1
0 7:1 12:1 21:1 36:1 45:1 53:1 67:1 73:1 77:1 79:1 94:1 100:1 107:1
0 2:1 12:1 19:1 40:1 45:1 54:1 68:1 73:1 78:1 79:1 94:1 100:1 108:1
0 4:1 12:1 22:1 41:1 45:1 52:1 68:1 72:1 78:1 79:1 94:1 100:1 107:1
0 6:1 12:1 23:1 35:1 47:1 55:1 66:1 73:1 78:1 79:1 94:1 99:1 109:1
1 7:1 11:1 20:1 37:1 45:1 52:1 67:1 72:1 77:1 79:1 94:1 101:1 107:1
1 3:1 12:1 22:1 41:1 44:1 54:1 66:1 72:1 78:1 89:1 94:1 101:1 107:1
1 5:1 12:1 19:1 40:1 45:1 52:1 67:1 72:1 77:1 84:1 94:1 100:1 107:1
1 4:1 12:1 24:1 38:1 45:1 52:1 67:1 73:1 77:1 79:1 94:1 104:1 107:1
1 2:1 10:1 19:1 40:1 45:1 54:1 67:1 74:1 77:1 79:1 94:1 100:1 110:1
0 1:1 12:1 19:1 40:1 44:1 51:1 69:1 72:1 78:1 79:1 94:1 100:1 107:1
0 3:1 12:1 25:1 39:1 44:1 56:1 66:1 73:1 77:1 79:1 94:1 101:1 107:1
1 4:1 12:1 26:1 38:1 45:1 57:1 67:1 74:1 77:1 79:1 94:1 100:1 111:1
0 3:1 12:1 27:1 35:1 45:1 58:1 67:1 75:1 77:1 79:1 94:1 101:1 112:1
0 1:1 11:1 20:1 37:1 44:1 59:1 69:1 72:1 77:1 79:1 94:1 100:1 107:1
0 3:1 12:1 20:1 37:1 44:1 60:1 70:1 72:1 77:1 79:1 94:1 100:1 107:1
0 4:1 12:1 21:1 36:1 45:1 56:1 67:1 72:1 77:1 79:1 94:1 101:1 107:1
```

- 训练完成后模型保存路径下会生成分类模型。

- 使用分类模型进行新的数据预测，得到分类结果，分类结果展示的是每个样本属于 label 类的概率。

```
-1.0 0.3419125542504
-1.0 0.38359382547300375
-1.0 0.018408318476705897
-1.0 0.06658325965624243
-1.0 0.6752134409759735
-1.0 0.8543909787580126
-1.0 0.001781148657691602
1.0 0.3965742371799506
1.0 0.41525546019306786
1.0 0.9444405599912905
1.0 0.6404916004498611
1.0 0.3155890391702023
-1.0 0.0025977153946314664
-1.0 0.04688815982657244
1.0 0.2499093859366886
-1.0 0.018902131436522255
-1.0 8.20684237451898E-4
-1.0 0.009407662011386627
-1.0 0.17514926971926495
1.0 0.22869286143817608
1.0 0.8905396640946988
-1.0 0.0036897059706793617
-1.0 0.03729956634018075
-1.0 0.3628251168938015
-1.0 0.03232311209756578
1.0 0.6774610219029031
-1.0 0.0035985976301669336
1.0 0.26610164425527427
-1.0 0.16190011624835868
-1.0 0.27298994759153083
```

内置算法的预训练模型

最近更新时间：2021-03-25 15:09:46

操作场景

TI-ONE 中的图像分类、目标检测和 BERT 类算法支持迁移学习功能，您可以在预训练模型的基础上，在特定目标任务上进行微调。使用预训练模型进行迁移学习可以显著加快模型训练速度，提升模型效果。
TI-ONE 内置算法的预训练模型您可以从以下地址下载，请注意使用时和算法相匹配，如 resnet_v1_50 模型需要使用 resnet_v1_50 的预训练模型，不能使用其它预训练模型。

图像分类

ResNet

- resnet_v1_50
- resnet_v1_101
- resnet_v1_152
- resnet_v2_50
- resnet_v2_101
- resnet_v2_152

解压后将对应 .ckpt 文件上传至 cos ，并在 resnet 算子的 预训练模型路径 参数中填写对应 .ckpt 文件在 cos 中的路径即可。工程对应的 cos 桶的根目录以 ${cos} 表示。

VGG

- vgg_16
- vgg_19

解压后将对应 .ckpt 文件上传至 cos ，并在 vgg 算子的 预训练模型路径 参数中填写对应 .ckpt 文件在 cos 中的路径即可。工程对应的 cos 桶的根目录以 ${cos} 表示。

Inception

- inception_v1
- inception_v2
- inception_v3
- inception_v4

解压后将对应 .ckpt 文件上传至 cos ，并在 inception 算子的 预训练模型路径 参数中填写对应 .ckpt 文件在 cos 中的路径即可。工程对应的 cos 桶的根目录以 ${cos} 表示。

MobileNet

- mobilenet_v2
- mobilenet_v2_140

解压后的目录中出现以 ckpt.data-00000-of-00001 ， .ckpt.index 和 .ckpt.meta 结尾的三个文件，将它们上传到 cos 的目录下，并在 mobilenet 算子的 预训练模型路径 参数中填写这个目录的路径加上 文件名.ckpt ，如 ${cos}/checkpoints/mobilenet_v2_140/mobilenet_v2_1.4_224.ckpt 。工程对应的 cos 桶的根目录以 ${cos} 表示。

NasNet

nasnet_mobile

解压后的目录中出现 model.ckpt.data-00000-of-00001 和 model.ckpt.index 两个文件，将它们上传到 cos 的目录下，并在 nasnet 算子的 预训练模型路径 参数中填写这个目录的路径加上 model.ckpt ，如 ${cos}/checkpoints/nasnet/model.ckpt 。工程对应的 cos 桶的根目录以 ${cos} 表示。

ShuffleNet

shufflnet_v2_050

解压后的目录中出现 model.ckpt.data-00000-of-00001 、 model.ckpt.meta 和 model.ckpt.index 三个文件，将它们上传到 cos 的目录下，并在 shufflenet 算子的 预训练模型路径 参数中填写这个目录的路径加上 model.ckpt ，如 ${cos}/checkpoints/shufflenet_v2_050/model.ckpt 。工程对应的 cos 桶的根目录以 ${cos} 表示。

目标检测

使用内置目标检测算法时，用户需要指定配置文件和预训练模型。

SSD

- ssd_inception_v2 配置文件
- ssd_inception_v2 预训练模型

使用时，将预训练模型解压后的文件上传到 cos 的目录下，并在 SSD 算子的 预训练模型路径 参数中填写这个目录的路径加上 model.ckpt 即可，如 ${cos}/checkpoints/ssd_inception_v2_coco/model.ckpt 。也可以不下载预训练模型，在参数中直接填写 ${ai_dataset_lib}/checkpoints/ssd_inception_v2_coco/model.ckpt 。

FasterRCNN

- faster_rcnn_resnet50 配置文件
- faster_rcnn_resnet50 预训练模型

使用时，将预训练模型解压后的文件上传到 cos 的目录下，并在 SSD 算子的 预训练模型路径 参数中填写这个目录的路径加上 model.ckpt 即可，如 ${cos}/checkpoints/faster_rcnn_resnet50_coco/model.ckpt 。也可以不下载预训练模型，在参数中直接填写 ${ai_dataset_lib}/checkpoints/faster_rcnn_resnet50_coco/model.ckpt 。

RFCN

- rfcn_resnet101 配置文件
- rfcn_resnet101 预训练模型

使用时，将预训练模型解压后的文件上传到 cos 的目录下，并在 SSD 算子的 预训练模型路径 参数中填写这个目录的路径加上 model.ckpt 即可，如 ${cos}/checkpoints/rfcn_resnet101_coco/model.ckpt 。也可以不下载预训练模型，在参数中直接填写 ${ai_dataset_lib}/checkpoints/rfcn_resnet101_coco/model.ckpt 。

BERT 类算法

TI-ONE 内置的 BERT 类算法（ BERT 文本分类 、 BERT-CRF 、 BERT 中文问答 ）需要填写 BERT 目录 这一参数。对于中文和英文，可以分别下载以下两个压缩包，解压后上传到 cos ，并填写 cos 中的目录。

- 中文 BERT 下载地址
- 英文 BERT 下载地址