



架构师

ARCHITECT

特刊

机器学习实践



SPECIAL ISSUE
January, 2017

架构师特刊



卷首语

机器学习，趁着近年来深度学习的热潮强势复苏，很快地从一个很少被大众关注的技术主题，转变为被很多人使用的管理工具和开发工具。其有效性被无数企业成功验证并推广应用，为了避免错失良机，企业需要设计自己的机器学习项目，比如在电商平台的推荐、排序业务中。在业务的多样性大的时候企业就需要考虑将机器学习系统平台化。对于学术界来说，学者们除了看重算法性能和运算效率之外，也希望机器学习平台容易调试、灵活性要强、迭代要快；而对于工业界更看重的是平台的稳定性强、处理大数据量、容易进行数据整合、高效率、低开发成本等。

InfoQ 实际上已经积累了不少企业机器学习平台构建的内容，包括百度的 PaddlePaddle，腾讯的 Angel 等。还有大规模机器学习平台，如第四范式的“先知”和 TalkingData 的 Fregata，各有优点。

Tensorflow 深度学习框架也应该开源一年了，经过了广泛应用和验证，本电子书里也收集了

一篇将 Tensorflow 应用于企业实践中的内容。另外还有将 DeepLearning4j 部署到生产环境。

机器学习企业实践也是少不了的，这些是将机器学习研究转化为真正的生产力，为用户带来实际价值，如：Twitter 机器学习平台的设计与搭建、机器排序学习在 1 号店电商搜索中的实战、百分点基于机器学习方法对销售预测的研究等等。

InfoQ 由社区推动，这里的内容源自像你一样的专业技术人员，欢迎大家投稿机器学习的相关实践内容：editors@cn.infoq.com，促进大家共同进步。

InfoQ 编辑 **Tina**

目录



05 深度学习框架 TensorFlow 在 Kubernetes 上的实践

13 百度 PaddlePaddle 深度学习平台介绍

25 第四范式大规模机器学习先知平台的整体架构和实现细节

53 轻量级大规模机器学习算法库 Fregata：快速，无需调参

67 Twitter 机器学习平台的设计与搭建

83 机器排序学习在电商搜索中的实战

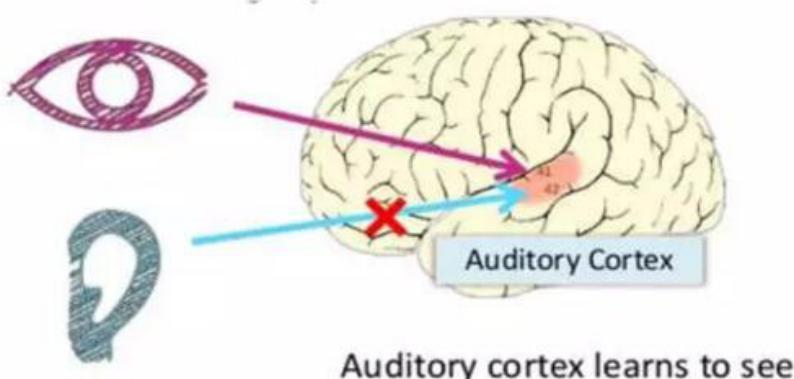
95 基机器学习方法对销售预测的研究

深度学习框架 TensorFlow 在 Kubernetes 上的实践

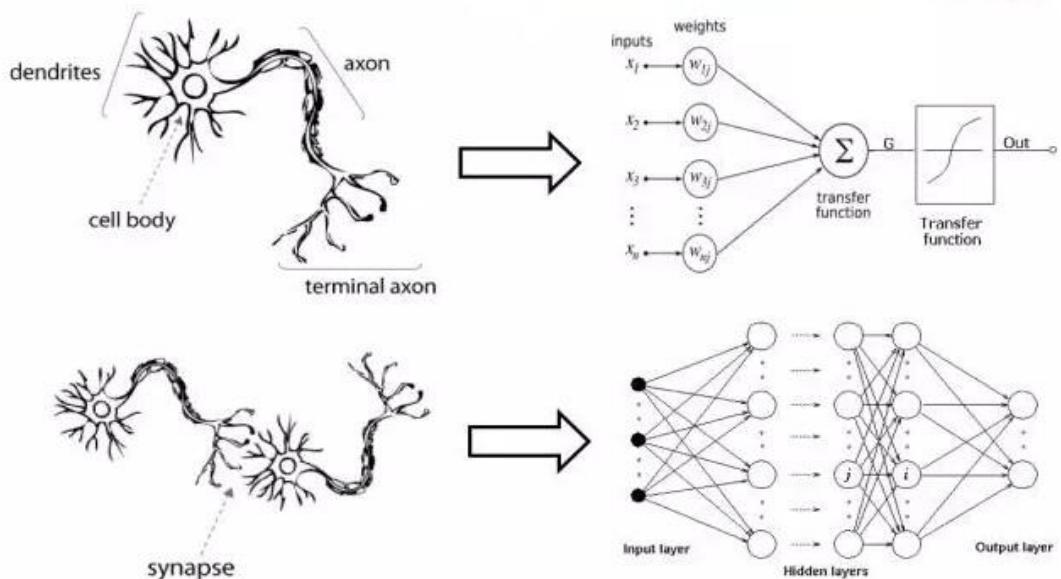
作者 郑泽宇

什么是深度学习？

深度学习这个名词听了很多次，它到底是什么东西，它背后的技术其实起源于神经网络。神经网络最早受到人类大脑工作原理的启发，我们知道人的大脑是很复杂的结构，它可以被分为很多区域，比如听觉中心、视觉中心，我在读研究中心的时候，做视频有计算机视觉研究室，做语言有语言所，语音有语音所，不同的功能在学科划分中已经分开了，这个和我们人类对大脑理解多多少少有一些关系。之后科学家发现人类大脑是一个通用的计算模型。



科学家做了这样一个实验，把小白鼠的听觉中心的神经和耳朵通路剪断，视觉输入接到听觉中心上，过了几个月，小白鼠可以通过听觉中心处理视觉信号。这就说明人类大脑工作原理是一样的，神经元工作原理一样，只是需要经过不断的训练。基于这样的假设，神经学家做了这样的尝试，希望给盲人能够带来重新看到世界的希望，他们相当于是把电极接到舌头上，通过摄像机把不同的像素传到舌头上，使得盲人有可能通过舌头看到世界。对人类神经工作原理的进一步理解让我们看到深度学习有望成为一种通用的学习模型。



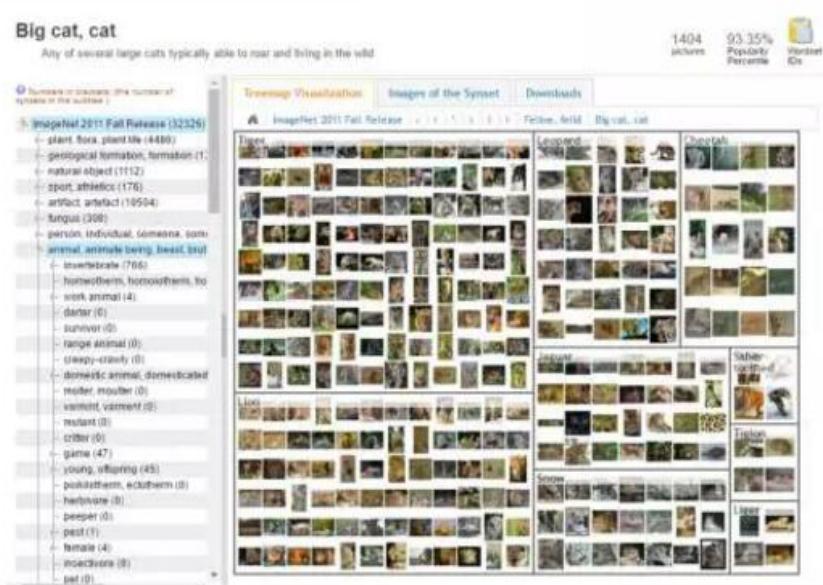
上图给出了神经网络的大致结构。图中左侧是人类的神经元，右侧是神经网络的神经元。神经网络的神经元最早受到了人类神经元结构的启发，并试图模型人类神经元的工作方式。具体的技术这里不做过深的讨论。上图中下侧给出的是人类神经网络和人工神经网络（Artificial Neural Network）的对比，在计算机神经网络中，我们需要明确的定义输入层、输出层。合理的利用人工神经网络的输入输出就可以帮助我们解决实际的问题。

神经网络最核心的工作原理，是要通过给定的输入信号转化为输出信号，使得输出信号能够解决需要解决的问题。比如在完成文本分类问题时，我们需要将文章分为体育或者艺术。那么我们可以将文章中的单词作为输入提供给神经网络，

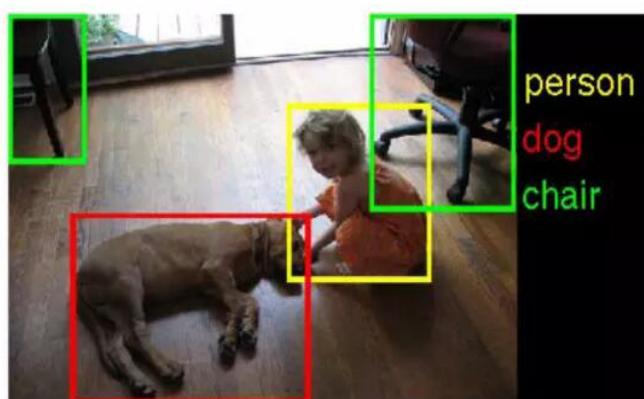
而输出的节点就代表不同的种类。文章应该属于哪一个种类，那么我们希望对应的输出节点的输出值为 1，其他的输出值为 0。通过合理的设置神经网络的结构和训练神经网络中的参数，训练好的神经网络模型就可以帮助我们判断一篇文章应该属于哪一个种类了。

深度学习在图像识别中的应用

深度学习，它最初的应用，在于图像识别。最经典的应用就是 Imagenet 的数据集。



ImageNet 是一个非常大的数据集，它里面有 1500 万张图片。下图展示了数据集中一张样例图片。



在深度学习算法被应用之前，传统的机器学习方法对图像处理的能力有限。在 2012 年之前，最好的机器学习算法能够达到的错误率为 25%，而且已经很难再有新的突破了。在 2012 年时，深度学习首次被应用在在 ImageNet 数据集上，直接将错误率降低到了 16%。在随后的几年中，随着深度学习算法的改进，错误率一直降低到 2016 年的 3.5%。在 ImageNet 数据集上，人类分类的错误率大概为 5.1%。我们可以看到，机器的错误率比人的错误率更低，这是深度学习带来的技术突破。

什么是TensorFlow

TensorFlow 是谷歌在去年 11 月份开源出来的深度学习框架。开篇我们提到过 AlphaGo，它的开发团队 DeepMind 已经宣布之后的所有系统都将基于 TensorFlow 来实现。TensorFlow 一款非常强大的开源深度学习开源工具。它可以支持手机端、CPU、GPU 以及分布式集群。TensorFlow 在学术界和工业界的應用都非常广泛。在工业界，基于 TensorFlow 开发的谷歌翻译、谷歌 RankBrain 等系统都已经上线。在学术界很多我在 CMU、北大的同学都表示 TensorFlow 是他们实现深度学习算法的首选工具。

```
import tensorflow as tf
sess = tf.InteractiveSession()
with tf.name_scope('input'):
    input1 = tf.constant([1.0, 2.0, 3.0], name="input1")
    input2 = tf.Variable(tf.random_uniform([3]), name="input2")
tf.initialize_all_variables().run()
with tf.name_scope("add"):
    output = tf.add(input1, input2, name="add")
writer = tf.train.SummaryWriter("/log/demo-hello", sess.graph)
print output.eval()
```

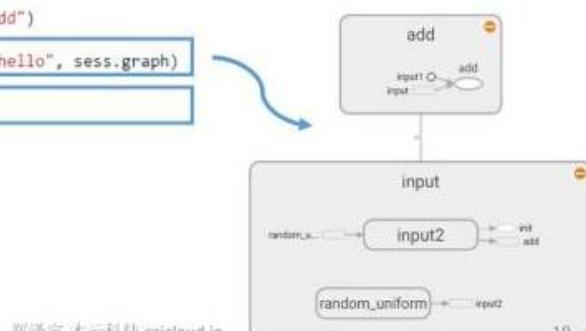
所有结果都需要先运行才能获取

通过 Sessions 维护上下文，所有执行都需要通过 Session

数据都存储在 "tensor" 中

变量都存储在 Variables 中

在运行前所有变量都需要初始化



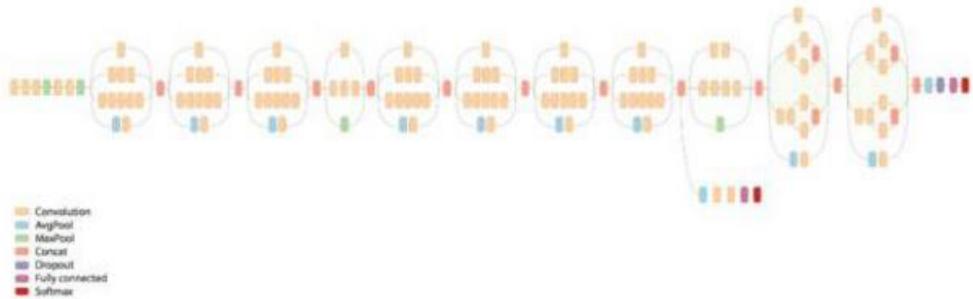
上面的 ppt 给出了一个简单的 TensorFlow 程序样例，这个样例实现了向量加法的功能。TensorFlow 提供了 Python 和 C++ 的 API，但 Python 的 API 更全面，所以大部分 TensorFlow 程序都是通过 Python 实现的。在上面程序的第一行我们通过 import 将 TensorFlow 加载进来。在 TensorFlow 中所有的数据都是通过张量（Tensor）的方式存储，要计算张量中数据的具体取值，我们需要通过一个会话（session）。

上面代码中的第二行展示了如何生成会话。会话管理运行一个 TensorFlow 程序所需要的计算资源。TensorFlow 中一个比较特殊的张量是变量（tf.Variable），在使用变量之前，我们需要明确调用变量初始化的过程。在上面的代码最后一行，我们可以看到要得到结果张量 output 的取值，我们需要明确调用计算张量取值的过程。

- 全连接层
 - `tf.nn.relu(tf.matmul(input_tensor, weights) + biases)`
- 卷积层
 - `tf.nn.conv2d(data, conv1_weights, strides=[1, 1, 1, 1])`
- 池化层
 - `tf.nn.max_pool(relu1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')`
- LSTM结构 + 循环神经网络
 - `lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(hidden_nodes)`
 - `cell = tf.nn.rnn_cell.MultiRNNCell([lstm_cell] * layers)`

通过 TensorFlow 实现神经网络是非常简单的。通过 TFLearn 或者 TensorFlow-Slim 可以在 10 行之内实现 MNIST 手写体数字识别问题。上面的 ppt 展示了 TensorFlow 对于不同神经网络结构的支持，可以看出，TensorFlow 可以在很短的代码内支持各种主要的神经网络结构。

虽然 TensorFlow 可以很快的实现神经网络的功能，不过单机版的 TensorFlow 却很难训练大规模的深层神经网络。



- **Inception-v3 model for ImageNet**
 - 2500万参数
 - 每次inference/forward prorogate 50亿 乘法/加法 操作

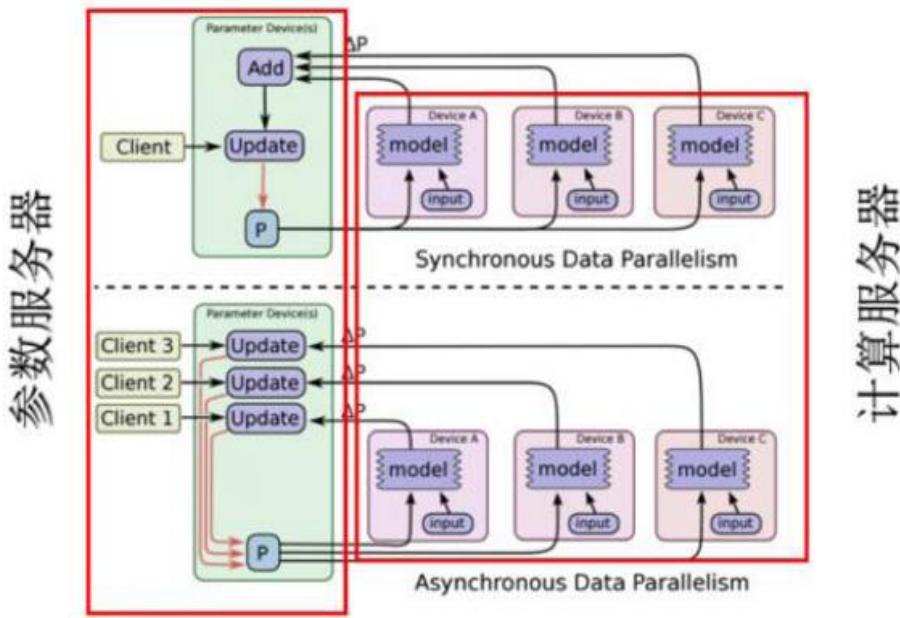
这张图给出了谷歌在 2015 年提出的 Inception-v3 模型。这个模型在 ImageNet 数据集上可以达到 95% 的正确率。然而，这个模型中有 2500 万个参数，分类一张图片需要 50 亿次加法或者乘法运算。即使只是使用这样大规模的神经网络已经需要非常大的计算量了，如果需要训练深层神经网络，那么需要更大的计算量。神经网络的优化比较复杂，没有直接的数学方法求解，需要反复迭代。在单机上要把 Inception-v3 模型训练到 78% 的准确率大概需要 5 个多月的时间。如果要训练到 95% 的正确率需要数年。这对于实际的生产环境是完全无法忍受的。

TensorFlow on Kubernetes

如我们上面所介绍的，在单机环境下是无法训练大型的神经网络的。在谷歌的内部，Google Brain 以及 TensorFlow 都跑在谷歌内部的集群管理系统 Borg 上。我在谷歌电商时，我们使用的商品分类算法就跑在 1 千多台服务器上。在谷歌外，我们可以将 TensorFlow 跑在 Kubernetes 上。在介绍如何将 TensorFlow 跑在 Kubernetes 上之前，我们先来介绍一下如何并行化的训练深度学习的模型。

深度学习模型常用的有两种分布式训练方式。一种是同步更新，另一种是异步更新。如下图所示，在同步更新模式下，所有服务器都会统一读取参数的取值，计算参数梯度，最后再统一更新。而在异步更新模式下，不同服务器会自己读取参数，计算梯度并更新参数，而不需要与其他服务器同步。同步更新的最大问题

在于，不同服务器需要同步完成所有操作，于是快的服务器需要等待慢的服务器，资源利用率会相对低一些。而异步模式可能会使用陈旧的梯度更新参数导致训练的效果受到影响。不同的更新模式各有优缺点，很难统一的说哪一个更好，需要具体问题具体分析。



无论使用哪种更新方式，使用分布式 TensorFlow 训练深度学习模型需要有两种类型的服务器，一种是参数服务器，一种是计算服务器。参数服务器管理并保存神经网络参数的取值；计算服务器负责计算参数的梯度。

在 TensorFlow 中启动分布式深度学习模型训练任务也有两种模式。一种为 In-graph replication。在这种模式下神经网络的参数会都保存在同一个 TensorFlow 计算图中，只有计算会分配到不同计算服务器。另一种为 Between-graph replication，这种模式下所有的计算服务器也会创建参数，但参数会通过统一的方式分配到参数服务器。因为 In-graph replication 处理海量数据的能力稍弱，所以 Between-graph replication 是一个更加常用的模式。

最后一个问题，我们刚刚提到 TensorFlow 是支持以分布式集群的方式运行的，那么为什么还需要 Kubernetes？如果我们将 TensorFlow 和 Hadoop 系统做一个简单的类比就可以很清楚的解释这个问题。大家都知道 Hadoop 系统主要

- 启动服务器
 - 通过Kubernetes的DNS机制设置服务器地址
 - 通过replica controller控制失败重启
 - Kubernetes提供监控、调度等功能
- 存储解决方案
 - 使用nfs、ceph等分布式存储

可以分为 Yarn、HDFS 和 mapreduce 计算框架，那么 TensorFlow 就相当于只是 Hadoop 系统中 Mapreduce 计算框架的部分。

TensorFlow 没有类似 Yarn 的调度系统，也没有类似 HDFS 的存储系统。这就是 Kubernetes 需要解决的部分。Kubernetes 可以提供任务调度、监控、失败重启等功能。没有这些功能，我们很难手工的去每一台机器上启动 TensorFlow 服务器并时时监控任务运行的状态。除此之外，分布式 TensorFlow 目前不支持生命周期管理，结束的训练进程并不会自动关闭，这也需要进行额外的处理。

讲师介绍

郑泽宇，谷歌高级工程师。从 2013 年加入谷歌至今，郑泽宇作为主要技术人员参与并领导了多个大数据项目，拥有丰富机器学习、数据挖掘工业界及科研项目经验。

【延伸阅读】

深度学习在 Spark 平台上如何进入生产环境



PaddlePaddle 深度学习平台介绍

作者 于洋

我们是谁？

PaddlePaddle 来自于百度深度学习研究院 (IDL)，百度深度学习研究院可能是中国第一家以深度学习为核心的大数据人工智能研发机构，而 PaddlePaddle 是百度 IDL 最早与 INF (基础架构部) 和 SYS (系统部) 开发的深度学习平台，所以这个项目非常有历史。我们的老大是徐伟老师，现任百度深度学习研究院“杰出科学家”，负责深度学习平台的开发以及算法的研究。我们目前的团队是比较小的，大概 10 人左右，主要以系统工程师为主，都是由 Github 办公，另外百度硅谷 AI 实验室资深科学家王益也加入了我们的团队。

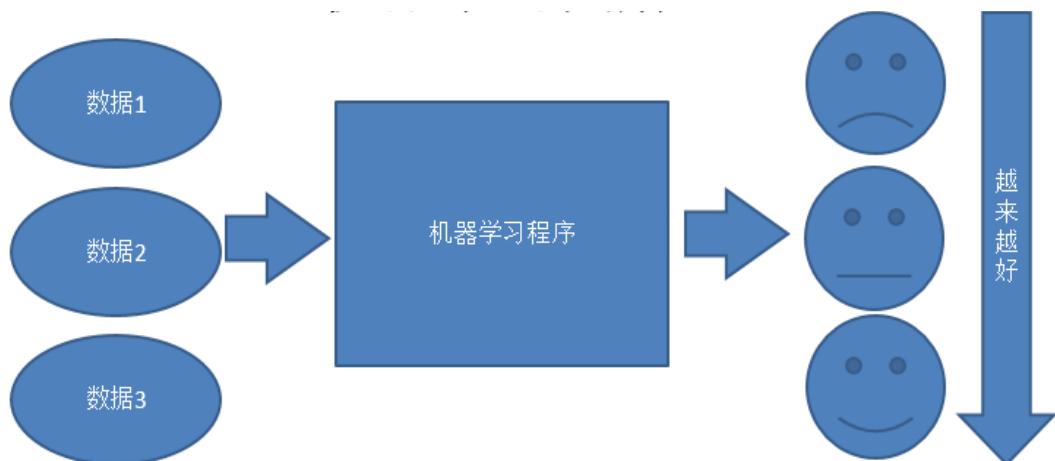
PaddlePaddle 是啥？

PaddlePaddle 是百度自主研发的深度学习的平台，是解决深度学习训练问题的平台。它的出发点，就是性能是第一优先，兼顾灵活易用性。PaddlePaddle 本身是一个非常务实的平台，这和其他的一些平台不太一样，我们本质是一个面向工程师的平台，是一个已经解决和将要解决一些实际问题的平台，而不是一个

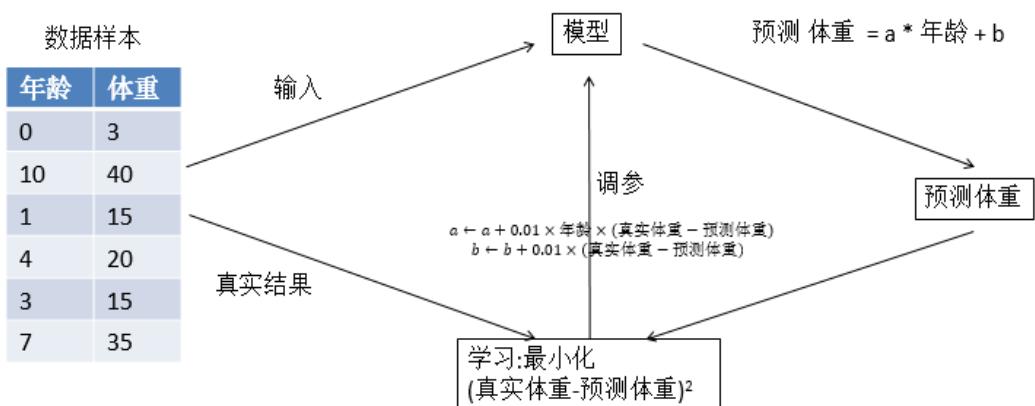
专业做科研的平台。目前百度有超过 30 个主要产品都有应用到 PaddlePaddle，比如搜索、杀毒、作业帮。

1. 什么是深度学习

这里首先简单介绍一下机器学习，什么是深度学习，深度学习是机器学习得一个分支，是通过多层的计算结构做的机器学习，机器学习又是人工智能的分支，所以这三者的关系是层层递进的。所谓机器学习就是不需要显示的通过编程告诉机器怎样做，而是随着数据增加机器的能力越来越强。



如图，我们先确定一个评价指标，机器学习就是通过数据总结规律，使这个评价指标越来越好，PaddlePaddle 就是中间的机器学习程序。

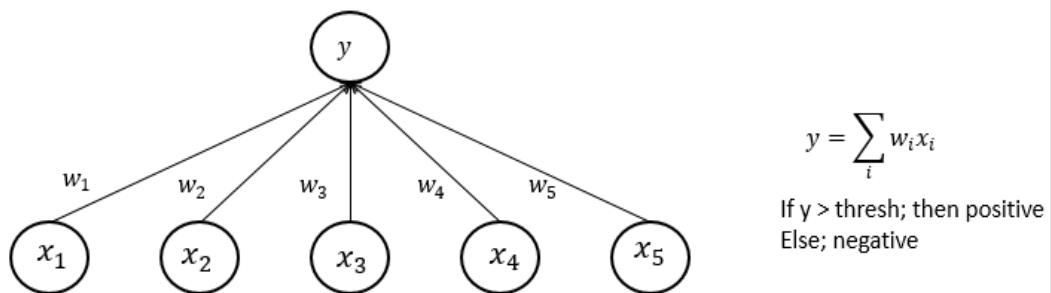


这是一个监督学习的例子，监督学习是指有一个数据样本，将数据样本输入

到一个初始模型中，它会给出一个预测结果，将预测结果跟真实结果做一个对比，通过学习让真实结果和预测结果更接近，然后学习的表现就是给模型调参。比如人的年龄和体重的关系，左边的数据是年龄和体重，比如说 0 岁是 3 公斤，10 岁是 40 公斤。首先先把样本数据输给模型，然后这个模型根据这个公式会有一个预测结果。然后根据这个预测结果和真实结果，也就是说体重之间的差，可以确定一个最小化的学习目标。这个学习目标就是让真实体重和预测体重的差最小，通过学习目标来调节模型的参数进而得到一个最优的模型，这就是一个最简单的监督学习的框架。

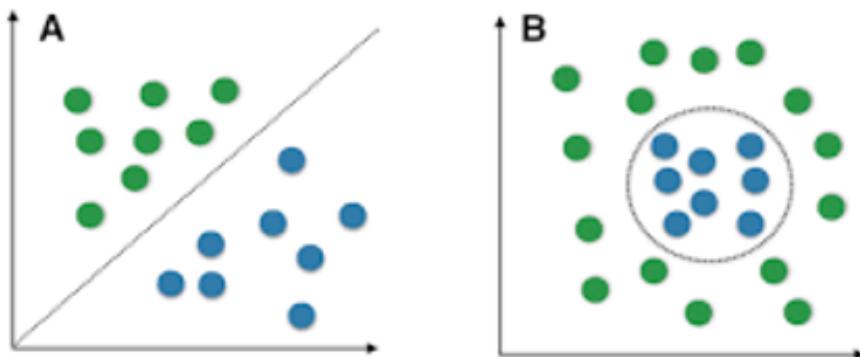
机器学习主要有四个要素。第一个要素就是数据，数据的训练场景和使用场景要一致。比如上面那个例子中人的年龄和体重，数据标签中并没有性别，可能都是女性的年龄和体重，但是预测的时候或者使用的时候是不在乎性别或者只有男性，这个时候预测肯定不准，因为一般而言男性比女性稍微重一些。第二个要素是模型结构，主要体现在对问题的理解。刚才预测年龄和体重的模型结构是一个线性的模型，就是一条直线，但是这肯定是不对的，因为人的体重肯定不会随着年龄无限的增长，所以模型结构和具体问题有关。而且模型结构还有一个问题就是输入特征是什么？刚才这个模型年龄应该是一个特征，当然还可能有别的特征，比如说收入情况，比较穷的人可能体重不会很大，比较富的人体重也不会很大，因为有时间锻炼，但是中产阶级可能比较胖。第三个要素是优化目标函数，就是如何衡量预测和真实的差异。刚才这个例子用的优化目标就是平方差。这是比较简单的一个优化目标，但如果是分类问题，比如如果要做一个手写字符的识别，两个类别之间做差肯定是不行的，那么应该如何设计这个优化目标呢，这也是一个比较重要的问题。第四个要素就是优化算法，也就是调参。实际上优化算法可以选择的很多，包括现在有一些自适应学习率的学习算法，都可以用 PaddlePaddle 来实现。

那么什么又是浅层学习呢？浅层学习的典型算法就是 SVM 和逻辑回归。图中 x_1-x_5 是五个特征， w_1-w_5 是五个权重，预测值 y 就等于权重和对应特征的乘积和。



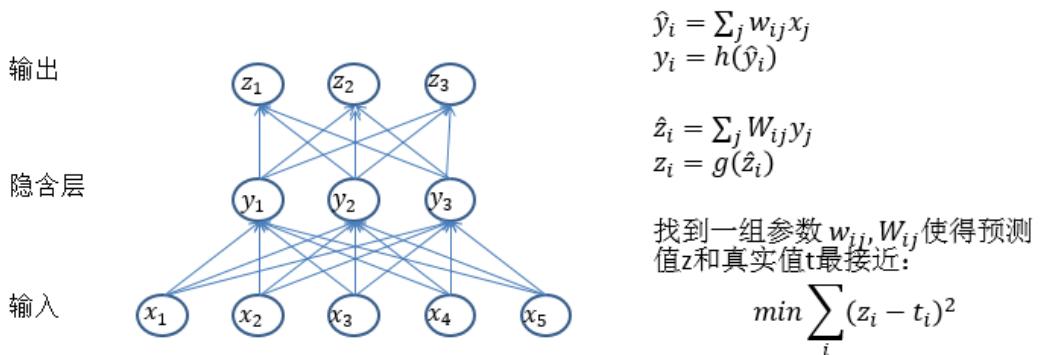
典型算法：支持向量机（SVM），逻辑回归（logistic regression）
 假如说这是一个分类问题，那么 y 大于阈值就是正例，小于阈值就是负例，这就是一个浅层学习的基本模型。如果把这个浅层模型投影到二维情况下，那么 y 就可以表示成 x 和 w 的一条直线，这条直线是一个分界线，直线两边分别表示正例和负例。但是对于非线性问题，也就是线性不可分的情况下传统的浅层学习是做不了的。

Linear vs. nonlinear problems



浅层学习的局限性一是依赖于特征选择，如果特征选择特别准，那么确实可能线性可分或者说一刀切。比如说人类性别分类问题，如果我们选择头发长度或者颜色，身高或者体重作为特征，这些特征不太可能一刀能够切开这个分割平面。但是如果选择 DNA 作为特征，显然这个问题是线性可分的。或者可以通过核函数，将原来的空间扭曲成一个线性可分的空间。这些情况的话需要对输入数据的分布

有比较清楚的了解，而且要有非常好的数学背景。但是如果没有那么好的数学背景呢？这个时候就需要神经网络了，传统的统计学习就是分析数据，调整核函数，而神经网络比较简单的玩法就是看看有没有效果，如果有效果再说为什么这么做。



上图是一个两层模型，两层模型的话其实就是一个浅层模型的堆叠。从 x 到 y 是一个浅层学习，从 y 到 z 也是浅层学习。它的需求点就是 y 到 z 应该是线性可分，但是 y 是通过 x 学习出来的特征。所以深度学习的特点就是叠很多层，然后在中间每一层对特征进行变换，直到最后变成一个可以分类的空间。中间的这些特征都是可学的，所以就不用显式地去编程中间应该怎么做了。

深度学习有以下几个特点，一是比较灵活，网络可以连接成不同结构，可以是环路或者跳层的连接，也可以选择不同的训练目标、激活函数或者正则化。二是可以学习高层次的抽象、分解变化因素。因为只有最后一层是做分类的，所以之前的都可以看作是特征提取的过程。多任务学习也是神经网络非常好的一个特点，比如说现在有一个机器翻译数据，是一个英文到法文，还有一个英文到中文数据，那么这两个机器翻译任务可以在一个神经网络里面学习，只是两个不同的任务，前面英文的相关网络都是可以共享的，这也是神经网络比较灵活的一点。四是转移学习或者迁移学习，这也是深度学习中非常常用的一点。我们可以从一个任务里得到一个神经网络然后直接应用到另外一个任务。迁移学习的好处就是对于一个新的领域，在数据量比较少的情况下，也可以得到比较好的效果。

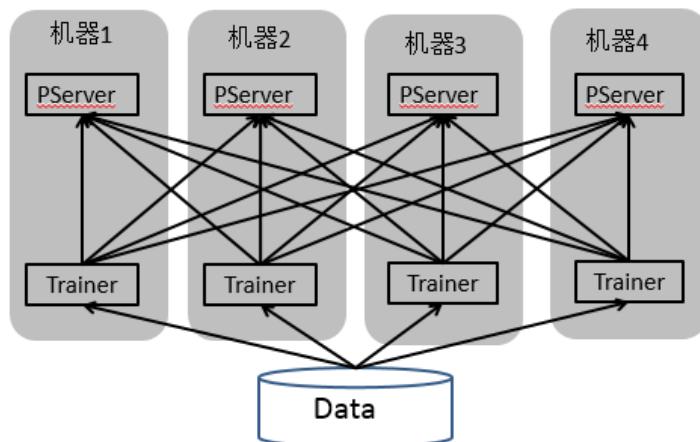
那么到底什么样的任务适合于深度学习？其实也是目前机器学习的一个问题。主要有两点，第一点是一个明确目标。这个目标最好可以用数学表示出来。

比如说优化目标可以是正确率或者某一种误差函数。第二点就是大量的数据，或者虽然这个任务里面没有大量数据，但是另一个任务有大量数据可以用，这两个任务之间可以迁移学习。所以目前来讲机器学习是一个很有局限性的问题。这个局限性，比如如果把围棋棋盘变成三角形或者六边形的事情，而直接用围棋棋盘来训练 AlphaGo 的话，它是适应不了棋盘的变化的，但是人是适应的。所以目前的 AI 还是一个特定领域下的 AI，应用场景非常窄。一个类似《西部世界》中能学习的智能体，目前还是一个需要探索的问题，我们还没有完全了解，我们也正在尝试了解。

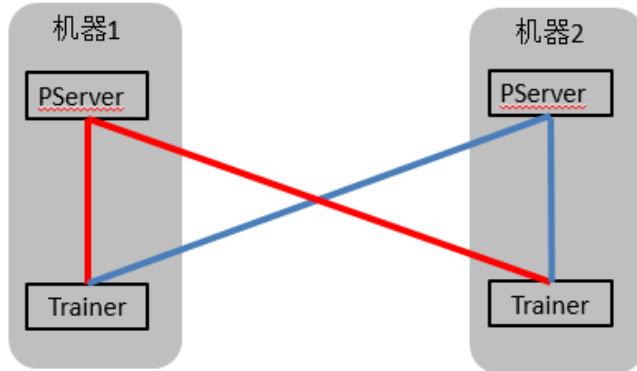
2. PaddlePaddle整体架构

下面介绍一些 PaddlePaddle 的整体框架。主要介绍多机的并行架构、多 GPU 的并行架构、序列模型的实现以及大规模稀疏训练的实现。这些问题都是实现神经网络最复杂的问题，具体怎么实现一个 layer 或者全连接，卷积这些，其实各大框架实现的原理都差不多，但是这些都是比较干货的东西。

- 参数分块
- 点对点通信
- 数据分布

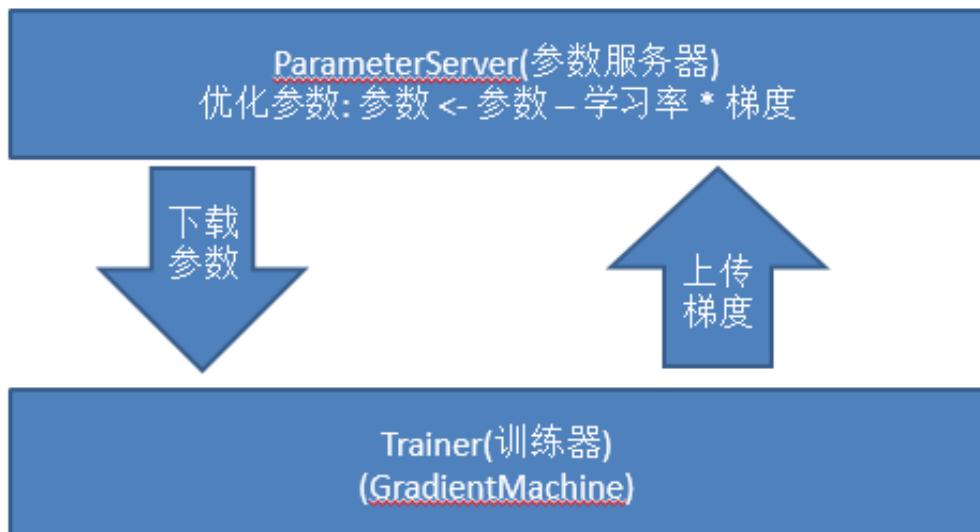


多机并行架构，图中是我们的实现方法，Paddle2013 年启动的时候，当时比较流行的架构就是 Pserver 和 Trainer 的架构。多机并行架构，首先是数据，数据分配到不同节点，就是简单的数据并行。灰色方框是一个机器，Pserver 和 Trainer 分布在两个独立的进程里，中间划线的部分是网络通信连接，可以看到 Pserver 之间是没有连接的。

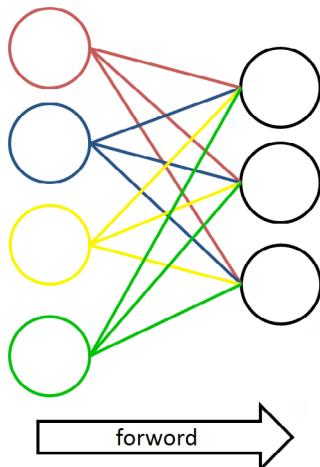


每一个机器的网络通信是均衡的

我们为什么做这么一个拓扑结构？我们的 Pserver 逻辑上可以用在任何机器上，中间是 PServer，然后许多 Trainer 去连接也是可以的，但是为什么每个机器都跑一个 PServer，每个机器都跑一个 Trainer，这样做的原因就是网络是均衡的。因为参数是被切分到不同的 PServer 上，这样我们网卡的出路和入路是一样的，网络是可以满载的。



多机并行架构的大致流程就是 Trainer 首先计算梯度，Pserver 接收上传的梯度，然后通过一个优化方法去学习参数的优化，然后 Trainer 在下一个 mini-batch 上直接从 Parmeterserver 上下载一个参数过来，这个算法就是一个简单的 SGD。

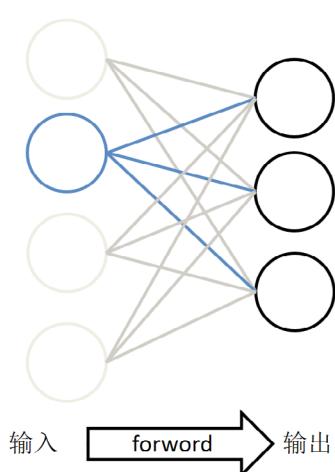


左侧表示神经网络的两层的连接。两层中间的连接(边)上，为Parameter。每一个Parameter均切分到不同的PServer上。

假设有四个PServer，那么参数的划分为

- 红色一组、蓝色一组、黄色一组、绿色一组
- 如果划分成三个PServer的话，则是红色一组、蓝色一组，黄色绿色一组(简单的除法)

这也是 PServer 比较有意思的地方，左边是一个神经网络，它会把神经网络每一层参数分割开，具体怎么分割，这也是一个比较有技巧的地方。左边表示两层神经网络中间的连接，每个连接上是有一个参数的。我们要做的就是把中间这些连接参数均匀的分配到不同的 PServer 上。如果假设我们有四个 PServer，那么参数分的话，Paddle 的实现是这么做的，就是在左边靠数据那边是做切割的。红色一组蓝色一组，红黄蓝绿四组，如果是四个 PServer 的话就是每一组放一个 PServer，如果划分为三个，最小粒度是左侧彩色的粒度。第一组是红色第二组是蓝色第三是黄色和绿色，这么做其实就可以实现一个稀疏训练的神经网络。



稀疏模型训练是说输入数据是稀疏的

由于稀疏输入，那么灰色的神经元和连接在训练中都没有作用

- 灰色神经元的输出是0
- 灰色连接的梯度是0
 - 梯度是0的话，简单的SGD不更新权重

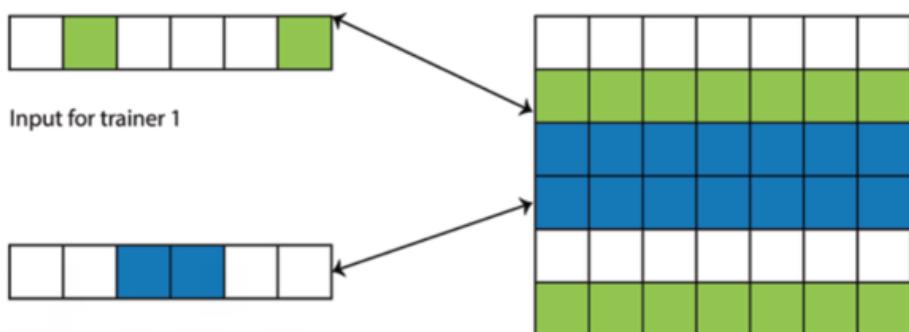
所以只有蓝色的连接有价值

- 需要从PServer服务器获得最新参数
- 需要计算梯度，并将梯度传回参数服务器

什么是稀疏模型的训练？首先输入数据是稀疏的，左侧（输入）到右侧（输出）

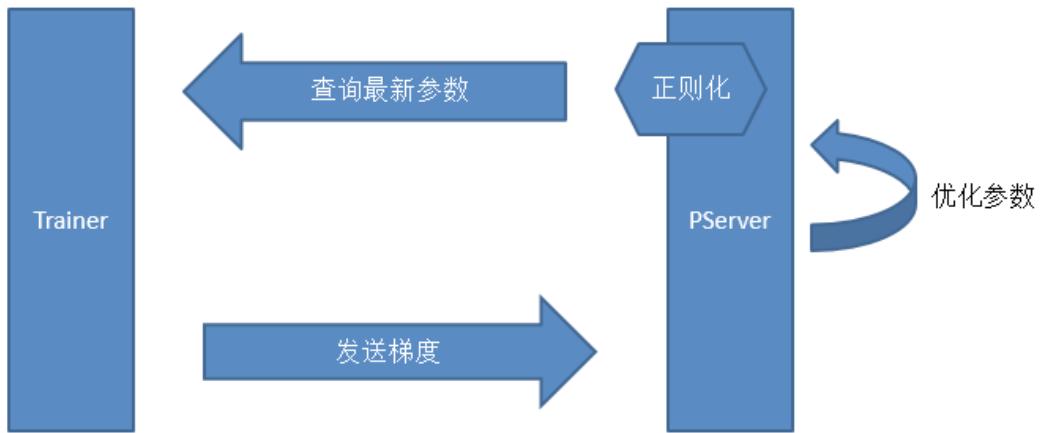
出），左侧是稀疏的话就是左侧有一些是实心的，剩下全是虚的。所以对稀疏来讲灰色神经元就是输出是 0，简单来讲就是梯度是 0，通过简单的 SGD 可以推出权重是不更新的。所以图中只有蓝色的是有价值的，需要计算的梯度是蓝色部分。所以按右侧划分 PServer 参数的时候，右侧划分三组这个功能不能实现，所以一般来说是从左侧划分成一些 Parameter block 给 PServer。稀疏训练的话，PaddlePaddle 的实现分两个部分。第一个部分就是做预取，预取就是读一遍训练数据标记好哪些神经元是有用的，哪些是稀疏的，然后从服务器上查询最新的参数。第二部分就是经典的神经网络方式，就是前馈和反向传播，先计算梯度，然后逐层返回梯度给服务器。这两个部分实现的时候是并行的，就是异步的。

- 每个 Trainer Prefetch 出自身需要的参数和服务器通信



前面是单机的模型，多机的大规模稀疏模型就是每一个 Trainer 找自身需要的参数给服务器，所以参数不存在单点上，这个集群里有整体所有的参数。这样设计还有一个特点，我们可以训练出来一个模型，这个模型占的内存比每个节点大。比如在百度我们用的机器内存有 200G，但是训练得到的网络占据的空间大于 200G。所以我们把参数所有的操作都放在 PServer 上，Trainer 没有任何参数，每次都是从 server 上直接获取。

正则化是大规模稀疏模型的另一个问题。简单的 SGD 在梯度是 0 的时候，是不需要更新参数，但是如果加上正则化就不一样，比如 L2 正则化的时候，就要求参数的 L2 范数持续减小。稀疏模型正则化，第一点是从 PServer 上查参数，

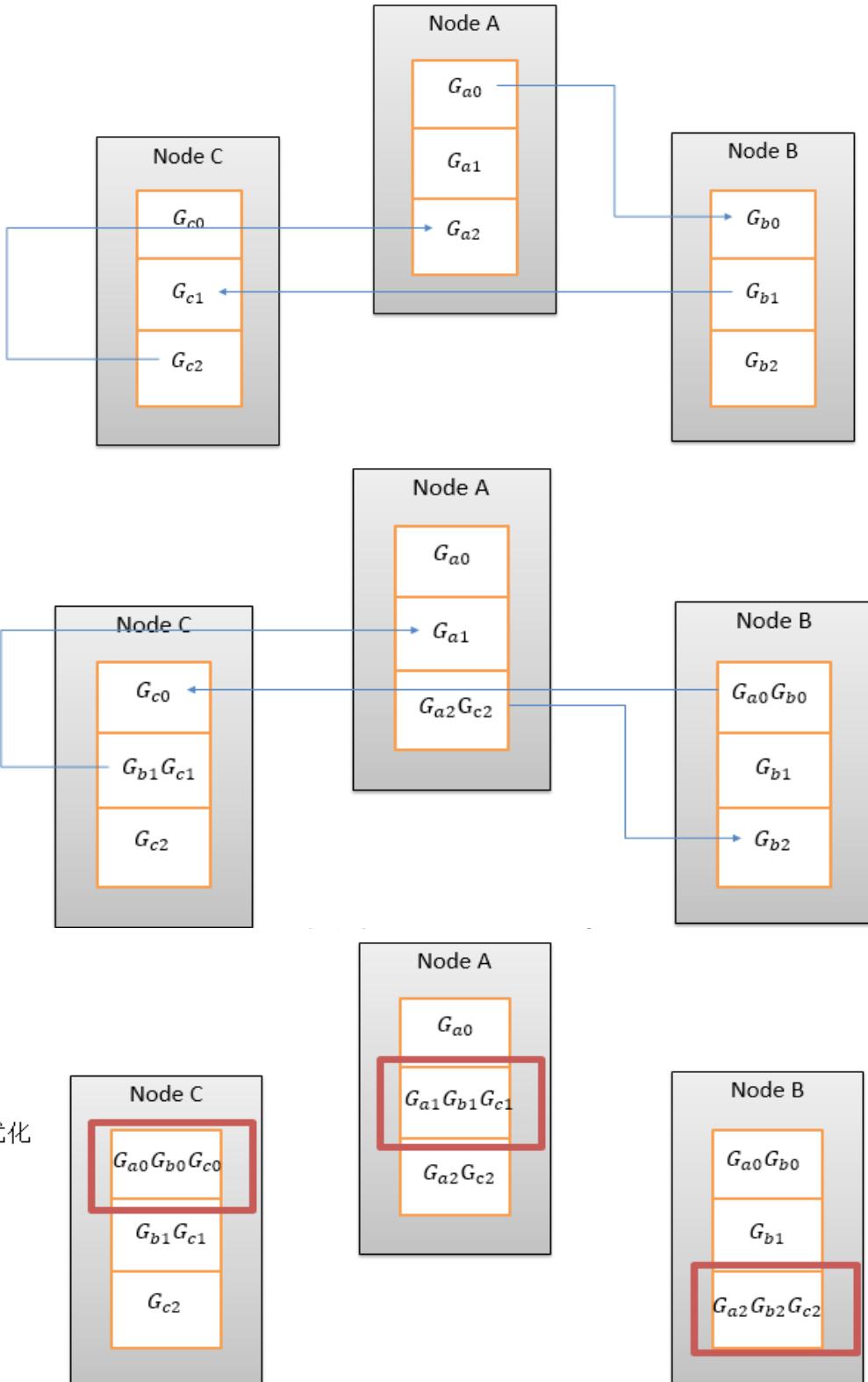


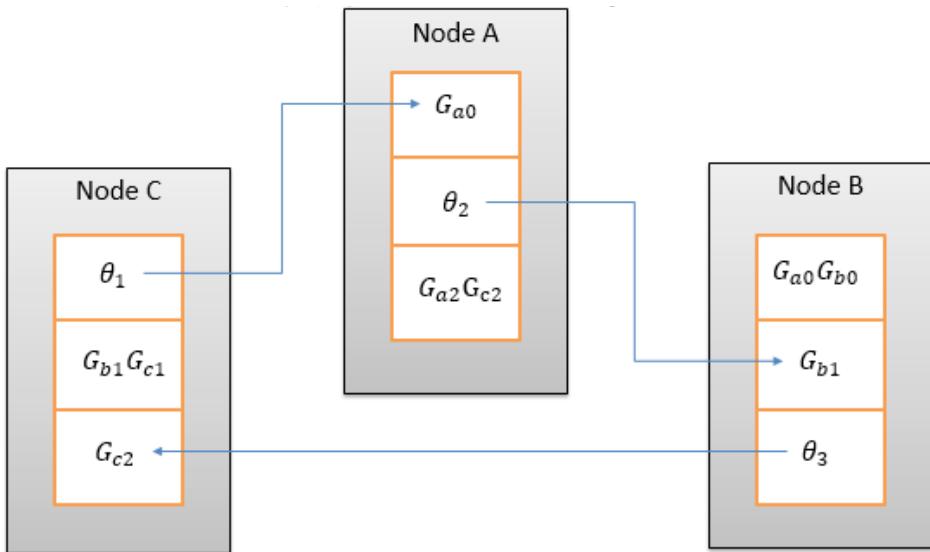
第二点 Trainer 去计算梯度然后发送给 PServer，第三点是做参数优化，这个优化是不包括正则化的优化。我们在查询参数之前去做正则化，用这个参数的时候正则化再开始做。正则化是每一轮都要做，要持续的让这个参数越来越好。因为是一个稀疏模型，不一定每一次都会访问这个参数，所以我们就会记录下来访问的次数，然后最后一次访问的时候把之前没有做的正则化补齐，这个就是大规模稀疏模型正则化的做法。

当然这是 2013 年的趋势，目前有一个流行趋势就是不需要 PServer 这个架构，而是完全 P2P 的更新。P2P 的更新一般是用一个环形网络，这个框架主要针对语言或者图像任务。比如说百度的语音识别就是基于这个框架。这个框架的多机通信算法的描述比较简单，这里有三个节点，节点 NodeA 就是机器 A，然后机器 B、机器 C，对于一个参数，首先在每个节点上切分成三份，我们要做的就是求和以后做一个优化。它的具体算法就是，比如说 NodeA 的时候，从 A0 发给 B0，NodeB 的时候直接从 B1 发给 C1 这样环形走，这是第一步。

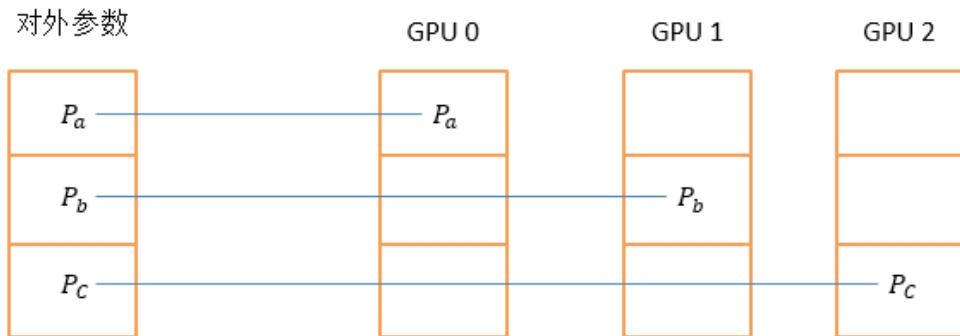
第二步求完和，然后再把求和的结果发给下一步。第二步这些和已经全了，全了以后画方框这一点，就是这个梯度已经和全部同步完毕，那么我可以优化到参数上，然后变成了 theta1、theta2、theta3 然后环形的发送回去。

环形网络通信的优势在于，首先实现同步算法的时候比较简单，和 Pserver 相比少了一部分通讯，而且同步力度非常大，到最后把参数算出来之后再做同步，





它中间每一个机器都可以随便分发。但是它不能实现异步 SGD，不过异步 SGD 通常收敛效果不是最佳的。所以不实现异步 SGD 这一点不是非常重要。第二缺点在于，实现稀疏更新的情况下，网络的开销变得越来越大。因为它新的节点会累加之前节点的信息，传递给下一个节点。因为是稀疏越累加越多，最后一个节点会有一个全量的数据。



神经网络由多个参数块构成，每一个参数块的参数都有一个主设备。

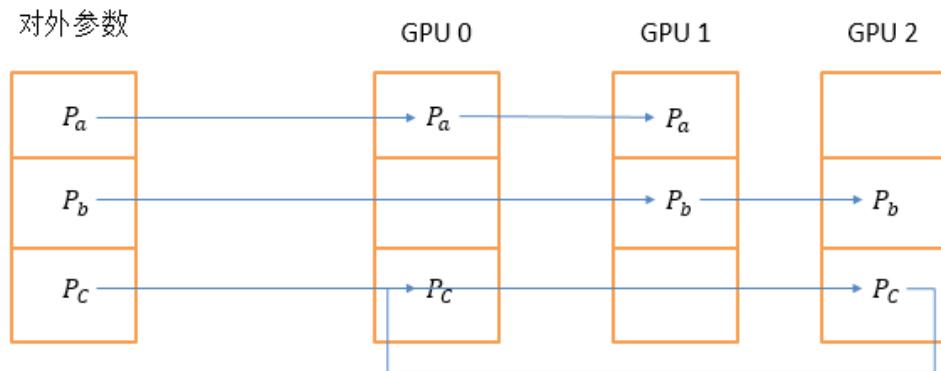
- 如果 paddle 配置使用多 GPU，那么参数会分在不同的设备上

PaddlePaddle 的单机多 GPU 通信是环形通信，这比别的计算框架要快的一点，比如 caffe 的算法快一倍到两倍。单机多 GPU 的时候是环形通信，那为什么是环形呢？因为 GPU 是一个 SPMD ((Single Program/Multiple Data)) 的设备，一次会处理很多数据，但是处理很多数据和处理很少的数据对于 GPU 来讲是一样的，

GPU 一般训练时都是稠密的数据。但是单机多 GPU 一般也不会做异步 SGD，这与多 GPU 通信和网络多节点通信不一样，不需要把参数聚合到 CPU，而是用一个逻辑的对外参数即可。

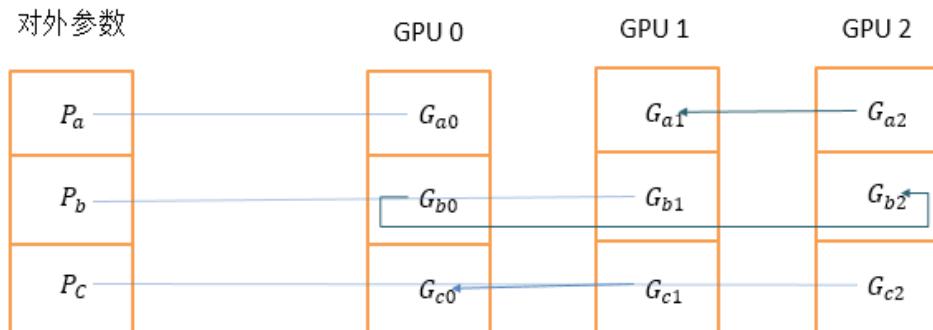
而多 GPU 架构，如上图，左侧是对外参数，是可以被 PaddlePaddle 其他的程序访问，然后这一个对外参数分成三个参数块，把三个参数块分别放到三个不同的主卡上。

多 GPU 架构的参数分发采用的是从主设备分发给下一个设备，再由下一个设备分发给下下个设备，依次往最远端分发的方式。



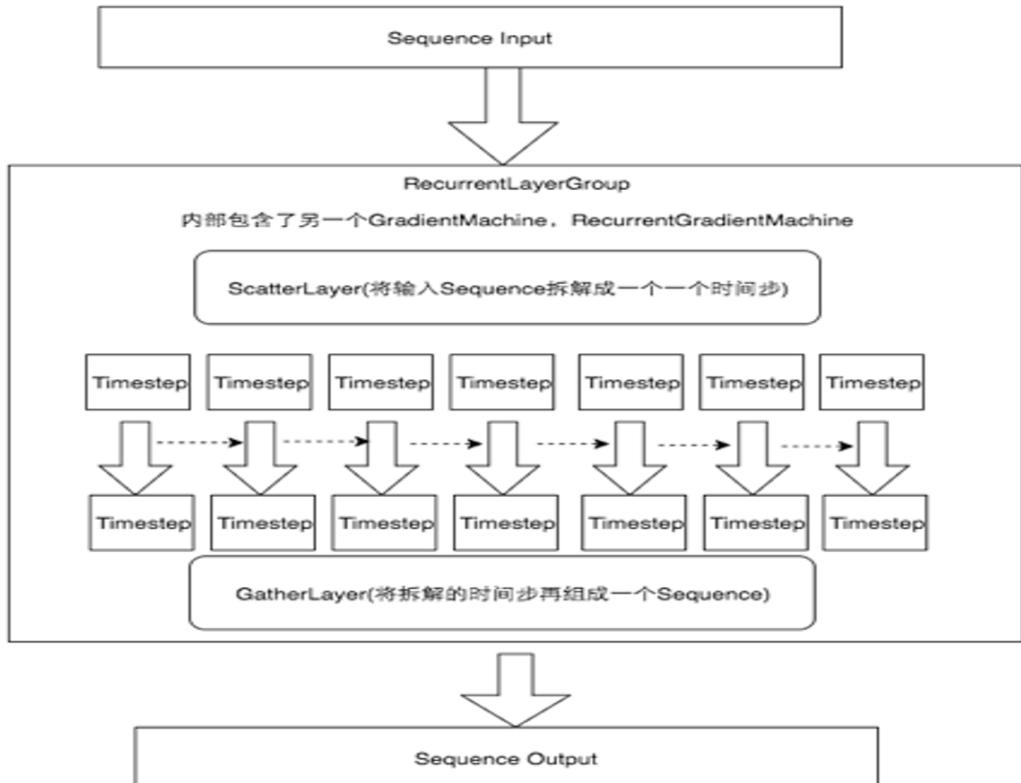
环形分发，将参数从主设备分发给下一个设备，再由下一个设备分发给下下个设备。

而梯度聚合就是沿着分发的反方向，将计算得到的梯度聚合到每个参数的主卡上。这样做好处就是只利用显卡之间的 P2P 通信，没有经过 CPU 和内存。



环形聚合，沿着分发的反方向，将梯度聚合到每个 Parameter 的主卡上

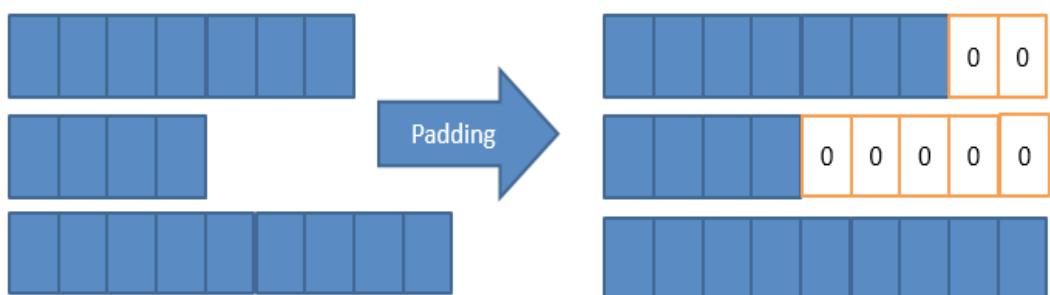
PaddlePaddle 还实现了对一些序列模型的训练，序列模型是指训练数据是一个序列（sequence），具有一定的顺序（order）的概念。就是序列模型的特征是 a vector of features 而不是 a set of features，比较常见的序列模型任务就是自然语言处理或者说音视频的处理。



PaddlePaddle 中对于序列模型的处理也和其他的神经网络框架不一样。但是这也充分说明我们的框架是比较务实的。具体实现是这样的，在其他层来看，sequence 都是一整条输入，但是对于某些特定的层，PaddlePaddle 会将输入序列打散成不同的时间步，然后在这个层内部对每个时间步进行处理，当然这个处理都是自定义的。这个时间部处理完了以后我们再用一个 GatherLayer 将拆解的时间步重新组合成一个 sequence 再输出出来，就是一个打散再聚合的过程。每个时间步之间是有通信的，比如做语音识别前一秒的语音和后一秒的语音有一定关系，这种通信 Paddle 是采用一种叫 Memory 的机制，就是在特殊的 RNN 的层里，定义一种 Memory，上一个时间步的神经网络，可以将需要记住的东西放到

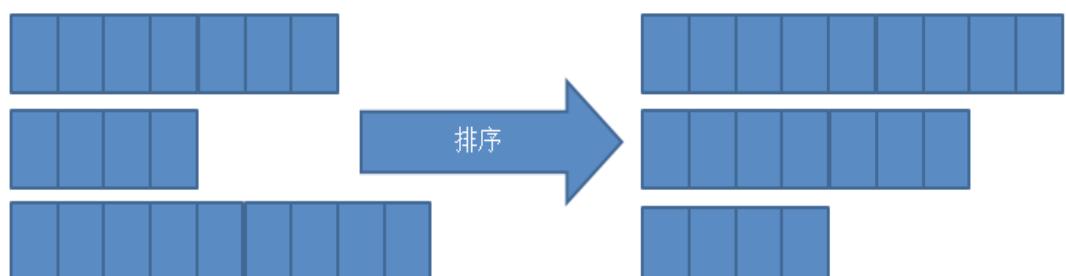
Memory 中，下一个时间步可以引用这个 Memory，这样就实现了 Memory 的共享。Paddle 是支持任意复杂的 RNN 结构，你知道的任何的 RNN 结构都是支持的，当然也可以自己搭配不同的结构。

Sepuence 序列模型最难的一点在于 batch 计算，为什么要做 batch 计算呢？因为矩阵维度要足够大，才能充分发挥 GPU 的能力，才能加速运算（SPMD）。但是 Sepuence 序列模型有一个问题就是输入是参差不齐的。那么怎么对这种输入进行计算？



Tensorflow、MXNet的实现

不同的网络框架的做法不一样，Tensorflow 和 MXNet 会做 Padding，将三个 Sepuence 全部 Pading 一样长，不够的后面补 0，然后再依次计算。当然 Padding 非常有技巧，比如把 Sepuence 和相近长度的 Padding 起来，使补 0 补得最少。但是本质上这种实现还是一个需要多计算的方法。



PaddlePaddle的实现

而 PaddlePaddle 的实现是做一个排序，不是物理排序，而是一个逻辑排序。

逻辑上排序成右侧的形式我们依次去计算。但是计算到一定程度以后，空间可能会变少。这是 PaddlePaddle 的实现，这个优点就是没有 Padding 不会增加计算量，但是缺点就是排序需要时间。这两个实现并不能说谁比谁，因为是这个问题终究没有非常好的解决，我们目前还在探索这个问题。

3. PaddlePaddle实现时的一些思考

目前主流的神经网络框架可以分为两种，一种是基于 OP（操作）的，就是从矩阵乘法配起，一步一步对应一个一个数学运算，配各种连接，然后大家可以直接根据数学公式把梯度推出来。另一种是基于层的，每一层，把一堆数学公式封装好，只提供一个最简单的概念。基于 OP（比如 TensorFlow）的框架的优势是更灵活，因为它可以让研究人员构造新的东西。而基于 Layer（caffe）的框架会让细节暴露的更少，会让优化变得更容易，计算更快一些。

那么 PaddlePaddle 是基于 OP 还是基于 layer ? PaddlePaddle 是一个混合的系统，首先 PaddlePaddle 支持大部分 Layer，但是也支持从 OP 开始配网络（比如矩阵乘法，加法，激活等等）。对于一些成型的 layer，比如 LSTM，我们使用 C++ 对其进行重新优化。我们刚开始做的时候 LSTM 还没有一个专门的 Layer，是从 OP 开始配，但是配完以后发现 LSTM 使用的非常广泛，所以就将其作为一个 Layer 重新在 C++ 实现一遍。所以在 PaddlePaddle 中我们把这些常用的 OP 的组合再优化成一个 Layer，这样做的好处就是可以兼顾灵活性，但是最重要的还是性能。这样做好处就是 PaddlePaddle 的 LSTM 是业界最快的，而且是从原理上就比其他框架快。

还有一个实现思考的话就是，PaddlePaddle 的多机通信到底是基于 MPI 还是 Spark 还是 k8s + Docker ? 首先 PaddlePaddle 的任务比较简单，因为我们不是一个网站，或者说要求很高的东西，如果运行几周挂一次还是可以接受的。比如说每个小时存一个数据点，再恢复的话并不是耗费非常大的事情。所以 PaddlePaddle 的网络任务需求相对简单，我们可以自己做，并不依赖于任何网络框架。第二点就是 PaddlePaddle 的网络是需要高性能的网络，从头手写网络

库更方便性能调优，并且 RDMA 可以更好的支持。同理，PaddlePaddle 底层不依赖任何 GPU 通信框架。当然，我们未来会提供一些配置脚本，比如说在多机上更容易实现的配置脚本。

当然这些都是 PaddlePaddle 实现上的一些东西，目前还有一些更新的东西在做，等做完了以后再和大家分享。

用PaddlePaddle做一些有意思的事

神经网络可以做什么？下面用 PaddlePaddle 来做一些有趣的事情为大家演示一下。

1. 手写识别

首先是手写识别的例子，代码大家可以从 github 上直接下载。数据集是图中右侧的手写字符数据集，每一个数据都是 256 色的灰度图，分辨率是 28×28 。

- 数据集—MNIST



- 示例代码均可在 https://github.com/reyoung/paddle_mnist_demo 下载

实际上，PaddlePaddle 对用户的接口是 Python，不需要很深刻的编程基础，这里展示一下 PaddlePaddle 数据读取的代码。数据读取的话首先有两个东西，pixel 和 label，PaddlePaddle 的数据读取只要考虑每一条数据就可以了。

网络配置的话其实比较简单。首先数据从哪儿读，从两个文件列表里读，然后文件名，函数名。这个就是一个简单的网络，这个叠的层数多一些。

```

3   data_dir = './data/'
4   define_py_data_sources2(train_list=data_dir + 'train.list',
5                           test_list=data_dir + 'test.list',
6                           module='provider',
7                           obj='process')
8   settings(
9     batch_size=128,
10    learning_rate=1e-3,
11    learning_method=AdaGradOptimizer()
12  )
13
14  img = data_layer(name='pixel', size=28 * 28)
15
16  hidden1 = fc_layer(input=img, size=200, act=TanhActivation())
17  hidden2 = fc_layer(input=hidden1, size=200, act=TanhActivation())
18  predict = fc_layer(input=hidden2, size=10, act=SoftmaxActivation())
19
20  outputs(classification_cost(input=predict, label=data_layer(name='label', size=10)))
21

```

2. 情感分类

情感分类的 demo 也是在 PaddlePaddle 的主版本库里，所谓的情感分类就是判断情感趋向，比如显示器很棒就是一个好评，用了两个月以后显示器碎了，那就是差评。但是这个数据集不太好展示，因为这是一个英文的数据集，我们其实很想做一个开放的中文数据来做演示，因为中文比较有意思。

- 该Demo在
https://github.com/baidu/Paddle/tree/master/demo/quick_start
- 数据集 <http://jmcauley.ucsd.edu/data/amazon/> 亚马逊商品评论
 - 这个显示器很棒！（好评）
 - 用了两个月之后这个显示器屏幕碎了。（差评）
- 英文

训练数据读取比刚才简单，因为是英语句子，直接读词就可以了，也不用分词。

网络配置的话，这个网络也比刚才更简单一些，首先经过一个 embedding_layer，把词变成一个向量，然后输入到 LSTM 里，输出一个最大的值，用这个值来做分类。

```
from paddle.trainer.PyDataProvider2 import *

UNK_IDX = 0

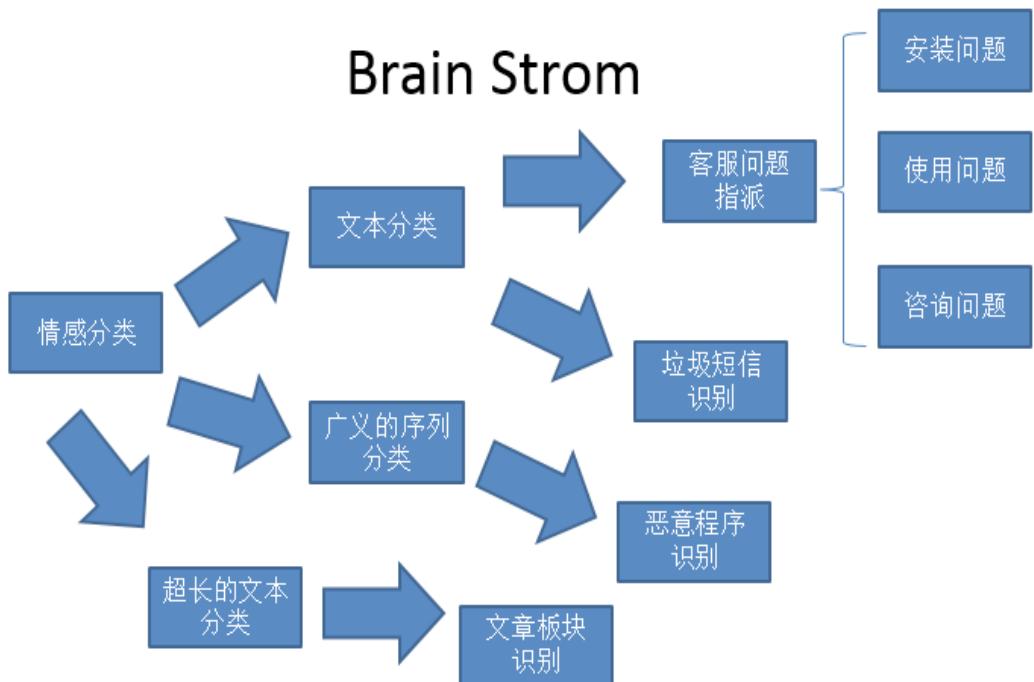
def initializer(settings, dictionary, **kwargs):
    settings.word_dict = dictionary
    settings.input_types = [
        # Define the type of the first input as sequence of integer.
        # The value of the integers range from 0 to len(dictionary)-1
        integer_value_sequence(len(dictionary)),
        # Define the second input for label id
        integer_value(2)]

@provider(init_hook=initializer, cache=CacheType.CACHE_PASS_IN_MEM)
def process(settings, file_name):
    with open(file_name, 'r') as f:
        for line in f:
            label, comment = line.strip().split('\t')
            words = comment.split()
            word_slot = [settings.word_dict.get(w, UNK_IDX) for w in words]
            yield word_slot, int(label)
```

通过情感分类我们还可以做什么？

现在我们有情感分类的 demo，情感分类最简单的拓展就是文本分类，文本分类可以做智能客服，比如客服有很多不同类型的问题，但是我也有很多客服把这些不同类型问题指派给相应的客服。实际上问题分类和情感分类也是一样的，目前百度上也有产品，是夜莺团队做的。再说文本分类的应用，比如说是一个手机助手的 APP，我要识别垃圾短信。那么怎么让模型有持续的学习能力能够识别不断进化的垃圾短信？我们可以再给他新的数据，他可以知道新的垃圾是什么，也可以用文本分类。再说情感分类，从文本这个角度展开成一个更广义的序列分类。然后我们可以做恶意程序的识别，因为我们写的 Code 也是文本序列，或者类似的文本序列，或者对 Code 进行一些简单的提取，总归还是可以做恶意程序的识别。再比如情感分类的文本很短，可能就是一句话，但是可以变成一个文章的识别，就是超长文本的识别。超长文本可以做文章板块，比如新闻类的 APP，投稿的新闻没有分板块，用这个模型可以预先将新闻分配到一个板块里，先让机器分一遍。PaddlePaddle 中文档的演示非常多，而且是非常实用的。PaddlePaddle 中的 demo 分这么几种，一种是图像分类一种，是自然语言处理，

还有推荐系统的演示。这些都可以演示，比如机器翻译的话，可以实现从广告词到另一个广告词的翻译，这些都是我们实际已经完成的东西。



除了 PaddlePaddle 主版本库中的一些 demo，目前 PaddlePaddle 还对外征集 demo 的创意。比如说一个想要的东西，并提供数据集，它可以是开源或者开放的，我们去做。在我们的 Github 上，IM 的平台上已经有人反馈说让我们做一个语音识别的演示，我们也已经放到这里面。或者说一个特别有意思的事情，比如说智能家居或者之类的东西，我们可以做一下。当然最后一点，因为 PaddlePaddle 是开源的，你们也可以直接贡献自己的 demo。

PaddlePaddle 从 9 月份开源到现在，大家的评价都比较好。但是还是有一些质疑开源的版本不是最新的。但是 PaddlePaddle 的话，目前来看有两点，首先确实不是所有的东西都开源，有一些东西不是我们的部门写的，是别的部门写的，我们没有做协调，所以没有开源，还有就是涉及业务的代码，有保密性，比如数据是百度内部的，但是剩下的不涉及业务的代码、不涉及其他部门的代码均已经开源。而且 Github 的版本库是 PaddlePaddle 的主版本库，现在所有的开发都是基于 Github 开发，Github 也是我们目前唯一的版本库。

联系我们

如果有什么问题或者建议都可以直接联系我们，首先可以直接通过 Github 的地址。因为我是工程人员，不一定是做模型做的非常专业，如果有关于模型的问题或者科研的问题都可以在这两个链接上交流，一个是英文一个是中文的。大家有任何的意见或者建议的话，Bug 和功能直接提交到 Issue 上，其他问题可以发邮件到 paddle-dev 这个邮箱里。谢谢大家！

答疑环节

提问 1：我想问一下目前 PaddlePaddle 支持的深度学习的模型，它的网络结构都有哪些？

于洋：PaddlePaddle 支持什么样的模型？ 常见的都支持，RNN 支持所有的 RNN，你也可以自己定义内部怎么连接，包括在 RNN 后面加一个卷积，输入数据中一些不常用的格式我们也支持。我们可以有两层的序列模型，比如对一个文章分类，每个词是一个 Sepuence，这也是支持的。自然语言处理中所有的问题都支持的，而且目前做的比较好。图像问题在我们开源之前，支持力度比较薄弱，现在一些其他的图像问题我们也做，下周会开始做所有的图像东西。语音我们也有相应的产品。

提问 2：PaddlePaddle 有没有做移动端上面的优化计算？

于洋：这个问题是这样的。我们之前在手机百度上面有 PaddlePaddle 的模型跑，但是之前的模型比较小，所以都是开发一个简单的框架。移动端的开发我们会在一个半月之内放一个版本出来，那个版本是支持 ARM 的训练和预测。预测框架的话，可能年底会有手机端的预测框架。

提问 3：我们这个 PaddlePaddle 是不是支持模型并行？具体怎么实现？把某一个放到单独节点？

于洋：支持。首先可以用稀疏来做，如果模型在某一层比较大，这个使用稀疏更新。还有多块显卡之间的模型并行我们也做了，可以在某些显卡上跑一些层，

其他显卡跑另一些层。但是在某些节点做模型的一部分，在另一些节点做模型的下一部分，这个还没有做。在百度暂时没有更大的神经网络需求，去做多机的模型并行。

提问 4：百度的神经网络框架有没有使用专用处理器，或者 FPGA？

于洋：我们的预测有专门的 FPGA，但是那也是另外一个团队做的。我个人理解专用的处理器在神经网络中应用最大的一点应该是在预测里，训练的时候大家可以等一下，但是预测需要非常快，需要非常非常快的时间里有个反馈。其实预测最大的市场应该是手机，但是在手机上用专用处理器和 FPGA 应该不太靠谱。所以百度现在目前来讲有做专用硬件的，但也是在硬件上做预测，训练的话我们没有做尝试，可能目前对于我们来讲这不是一个非常关键的问题。但是预测对手机或者说对于其他的加速还是非常重要的。

讲师介绍

于洋，百度工程师，从事百度深度学习平台 PaddlePaddle 开发工作。硕士毕业于天津大学，15 年毕业后加入百度深度学习实验室。随后一直从事深度学习系统的研发，主要负责深度学习系统的性能优化和功能开发工作。

【延伸阅读】

大规模机器学习在蚂蚁 + 阿里的应用



第四范式大规模机器学习先知平台的整体架构和实现细节

作者 胡时伟 涂威威

本文主要从如下几个方面来讲述 AI 开发平台“先知”产品的一些经验：

- 为什么人工智能系统需要高维大规模机器学习模型；
- 训练高维大规模机器学习模型算法的工程优化；
- 机器学习产品的架构实践；

首先先从人工智能发展说起，人工智能并不是一个最近出现的概念，早在 60 年代就有著名学者曾经预言二十年内机器将能够完成人能做到的一切工作。到今天我们又听到说 20 年之内，一大半的工作岗位将被机器人替代。那么 60 年代到今天发生的最大的区别是什么？这其中发生了两个重大的变化，第一个是计算能力的突飞猛进，今天的手机一个核的计算能力就足以秒杀当年的超级计算机。第二个是我们拥有了大数据，TB 级的数据存储、处理在今天已经不再困难，而 20 年前，GB 级的硬盘才刚刚兴起。

因此我们说今天人工智能 = 机器学习 + 大数据，那么什么样是好的人工智能呢？这里引入一个“VC 维”的概念。“VC”维是 1960 年代到 1990 年代由

Vapnik 及 Chervonenkis 建立的一套统计学习理论。VC 维反映了函数集的学习能力，VC 维越大则模型或函数越复杂，学习能力就越强。之前，统计建模曾经进入过一个误区，就是去追求经验风险最小化，什么意思呢？就是说我希望建立一个模型，在给定的样本上不要有误差，这样感觉非常好，但是往往这么一来，在实际的预测中非常糟糕，这是为什么呢？是因为采用了一些 VC 维很高的模型，虽然函数集学习能力是强了，但是由于数据不够，所以导致置信风险变大产生了一些类似过拟合的情况，最后这个模型还是不好用。

但是今天我们进入了大数据时代，样本的数量，包括样本的特征丰富程度有了极大提升，这就又带来了提升 VC 维的新机会。我们经常说经验主义害死人，过去的建模就是害怕经验主义，所以呢就把这个大脑变笨，降低 VC 维，使得模型更有效。但是今天的大数据情况下，可以通过补充更多的阅历（数据），来避免经验主义，那么一个阅历丰富的聪明的人，自然是要比一个笨的记不住东西的人要好的。因此我们说大数据人工智能时代，提升 VC 维变成了一个好的人工智能系统的关键因素。

那么机器学习中的高维度从何而来？传统方法只能利用可以放在特征矩阵这个平面中的数据，对于立体的数据，多维度的数据，因为它们多不是数字，所以传统机器学习模型无法处理，只能选择舍弃。但在实际工业应用中，这类非数字化的数据所包含的信息，往往信息价值很高，比如它可能对个性化推荐很有影响，可能对泛化处理有帮助。为了能充分利用这些数据，我们对特征矩阵外的立体的数据通过切片等算法进行变换，使得变换后的数据成为特征矩阵的一部分，同时还对不同特征之间进行交叉组合等操作，这样特征矩阵的每一行的列数就从原始数据的列数，变成了每一行都是一个巨大（比如 2 的 64 次方）的向量，形成超高维度的模型。

高维度模型真正的意义何在？通过对原来立体数据切片处理，可以使得某些过去只能有简单线性表达的数据，比如年龄等，获得更接近真实情况的细腻体现；此外，原来机器学习不能利用的非数字、没有排序关系的数据，比如姓名等，也

可以发挥其价值所在。举个例子，在个性化推荐的场景中，体现个性化信息的数据之间通常是不可比的，比如，我们先只考虑热度、推荐序号和用户 ID 三个变量，其中用户 ID 这个变量就是传统机器学习模型所不能利用的，只有通过将这个数据切片处理，获得一个高维度模型，才可以真正将用户信息这个数据发挥出价值。

要获得一个成功的高维度模型，就需要好的机器学习系统，这个系统应当具备两个特征：横向可扩展的高计算性能和算法本身在收敛过程中的正确性。为此，我们的算法工程团队开发了一系列的基础设施组件，组成了大规模分布式机器学习框架 GDBT (General Distributed Brain Technology)。GDBT 是一个由 C++ 编写的，完全分布式的适合于机器学习计算场景的计算框架，可以运行在单机、MPI、Yarn、Mesos 等多个分布式环境。

大规模分布式机器学习框架GDBT

机器学习是一种数据驱动的实现人工智能的方式，机器学习在实际应用中的大数据、高维度背景导致需要一个高效计算的平台，同时，监督学习领域著名的 No Free Lunch 定理指出，没有一个机器学习模型能够对所有的问题都是最有效的。所以在不同的实际问题里，需要使用不同的机器学习算法或者对机器学习算法做适应性地调整，去达到更好的实际效果。因此在实际的应用中，需要能够非常容易地开发出适应实际问题的机器学习算法。相比于传统的 ETL 计算，机器学习算法的计算过程有很多自身的要求和特点。

在框架设计上，没有普适的最好的框架，只有最适合自身应用场景的框架。GDBT 框架的设计初衷主要就是打造一个专门为分布式大规模机器学习设计的计算框架，兼顾开发效率和运行效率。GDBT 框架不是某一种算法，而是一种通用的机器学习算法计算框架，使算法工程师可以基于 GDBT 开发各种传统或者创新算法的分布式版本，而不用过多地关心分布式底层细节。

目前比较流行的计算框架比如 Hadoop、Spark 其重点任务大多是 ETL 类的任务。前面提到机器学习计算任务相比于传统的 ETL 计算任务有很多自身的特点，

时间有限这里可以简单地展开一下。

在计算方面：相比于 ETL 会做的一些相对“简单”的运算，机器学习算法会对数据做相对复杂的运算，一些非线性的模型，比如深度学习模型会需要比较密集的计算；在实际的应用中，需要考虑不同计算资源的特性；分布式计算中，由于分布式带来的通讯、同步、灾备等等的 overhead 需要调整计算模式来尽可能地降低。

在通讯方面：很多机器学习算法在计算过程中会频繁使用到全局或者其他节点的信息，对网络吞吐和通讯延迟的要求要远高于 ETL 任务。同时，很多机器学习任务对于一致性的要求要低于 ETL 任务，在系统的设计上可以使用放松的一致性要求。

在存储方面：ETL 需要处理来源不同的各种数据，比较少的反复迭代运算，很多机器学习算法会对数据做反复的迭代运算，可能会有大量的不断擦写的中间数据产生，对存储的使用效率、访问效率有着更高的需求。

在灾备和效率的权衡方面：容灾策略有两方面的额外开销，一方面来自于执行过程中为了容灾所要做的额外的诸如状态保存之类的操作，另一方面来自于灾难恢复时所需要进行的额外重复计算开销。这两方面的开销是此消彼长的。与 ETL 计算任务不同，机器学习计算任务流程相对复杂，中间状态较多，在较细的粒度上进行容灾会增加执行过程中的额外开销。因此在容灾策略和容灾粒度上，机器学习计算任务和 ETL 计算任务之间的权衡点不一样。

GDBT 计算框架针对机器学习任务在计算、通讯、存储、灾备等方面做了深入的优化。时间有限，这里可以简单的讲一点 GDBT 的工作，有兴趣了解更多的同学可以会后再和我们联系，或者加入第四范式一起解决这些有趣的问题：）

在计算方面，计算硬件发展到今天，提升计算能力的主要方式是堆积计算能力的分布式并行计算，比如现在做深度学习非常流行的 GPU 本身也是一种并行计算硬件，针对不同需求的计算硬件的种类也在变多，比如 FPGA 等等。对于大数据、高维度、复杂的机器学习计算任务，GDBT 框架充分考虑了分布式并行计算

的特点，针对不同的硬件资源、不同的算法场景做了调度、计算模式、机器学习算法部件的抽象等的优化。GDBT 框架本身在设计上也刻意避免了现在一些框架设计容易走入的误区。比如：为了分布式而分布式，但是忘记了分布式带来的 overhead。GDBT 框架针对单机、分布式模式分别进行了优化，框架本身会对不同规模的计算任务、不同的计算环境做自适应的调整选择更高效的实现。

然后在通讯方面，通信效率是分布式机器学习系统中至关重要的部分。首先，相比于 ETL 任务，很多机器学习算法在计算过程中会频繁使用到全局或者其他节点的信息，对网络吞吐和通讯延迟的要求都很高。其次，一些机器学习算法在通信时可能有很多可以合并处理的通讯需求。再次，很多机器学习算法并不要求在所有环节保证强一致性。最后，对于不同的网络拓扑，最优的通讯方式也会不一样。因为存在可能的合并处理、非强一致性需求、网络拓扑敏感等，就会存在有别于传统通讯框架的优化。先知的 GDBT 计算框架内部，针对机器学习计算任务单独开发了一套通信框架，为机器学习任务提供了更高效、更易用的支持点对点异步、点对点同步、深入优化的组通讯（比如类 MPI 的 Broadcast、AllReduce、Gather、Scatter 等等）的通信框架，同时为应用层提供了简便易用的合并、本地缓存 等等功能。

然后介绍下我们在参数服务器（Parameter Server）方面的工作，很多机器学习算法的学习过程都是在“学习”一组参数，对于分布式机器学习任务，不同的节点需要对“全局”的这组参数进行读取和更新，由于是分布式并行的计算任务，因此存在着一致性的问题，不同的机器学习算法或者同一个机器学习算法的不同实现对一致性的要求会不尽相同，不同的 一致性策略对整体算法的效率会产生很大的影响。参数服务器就是为了解决分布式并行机器学习任务中多机协同读取、更新同一组参数设计的，设计上需要提供简单易操作的访问接口方便机器学习专家开发算法，同时也需要提供可选择的一致性策略、灾备等等。

另外，由于是为机器学习任务设计，参数服务器的设计目标是高吞吐、低延迟。GDBT 中的参数服务器针对不同的应用场景做了更加深入的优化，结合高效缓存、

智能合并等优化策略以及基于 GDBT 自带的高效异步通讯框架，同时，还针对稀疏、稠密等不同的参数场景进行了针对性优化；内置提供了丰富的一致性策略可供用户选择或自定义一致性策略；GDBT 将参数服务器看成一种特殊的 Key-Value 存储系统，对其进行独立的灾备设计，同时提供不同粒度的灾备选项，便于在实际的部署中选择合适的灾备策略提升效率。

这次我主要讲计算框架方面的工作，以后有机会可以再详细分享我们在机器学习算法框架以及机器学习算法方面的设计工作。针对机器学习算法计算，GDBT 框架做了进一步的抽象，比如数据流、优化算子、多模型框架等等，这里简单介绍下数据流的设计。对于研究并实现机器学习算法的专家而言，算法的核心就是数据的各种变换和计算。GDBT 框架为了让机器学习专家更容易、更快速地开发出不同的机器学习算法提供了数据流的抽象，使得机器学习专家通过描述数据流 DAG 图的方式编写机器学习算法。机器学习专家只需要关注数据的核心变换和计算逻辑，GDBT 计算框架将机器学习专家描述的数据流图进行高效的分布式计算。

数据流计算框架的抽象一方面有助于降低机器学习专家的开发门槛、提升开发速度（他们不需要关注底层分布式、并行细节），另一方面也为 GDBT 框架提供了更大的空间去进行数据流执行优化，能够进一步提升执行效率（如果 GDBT 框架只在底层模块进行优化，将会非常乏力；但是数据流的抽象使得 GDBT 框架能够更全面地了解计算任务的 pattern，可以做更多的优化工作）。

在存储、灾备设计等方面，GDBT 也做了深入的优化，比如多级缓存的设计、框架层面对存储读取写入、网络访问、计算过程等等的灾备设计，对不同计算规模采取不同的灾备粒度等等，这里时间有限，就不做过多介绍了。机器学习算法部分的工作以后有机会可以再一起交流。

GDBT 定制的通信框架、算法框架以及参数服务器，为进行大规模机器学习训练提供了基石。当然 GDBT 还有一个很大的特性是算法开发者友好，对于算法开发来说，学术研究和工业应用之间存在一定的取舍。一些其他的算法框架比如 Tensorflow，比较注重研究上的易用性，从而在效率上有所舍弃，而一些注重于

生产应用的算法框架特别是分布式框架，在算法二次开发和扩展上则捉襟见绌。GDBT 提供的是工业级的开发者易用性，从语言级别，GDBT 整体基于 C++ 14 标准，为算法的开发提供了更大的自由。从功能抽象上，GDBT 提供了对参数服务器和算子的良好包装，在 GDBT 上，只需要数百行代码就可以实现像逻辑回归、矩阵分解等算法的分布式版本。

机器学习算法框架的语言选择问题

关于机器学习算法框架的语言选择问题，因为今天很多的大数据框架都是基于 Hadoop/Spark 这一套的，都是 JVM 上的东西，因此基于 JVM 从整体来说接入生态是比较容易的，但其实有三点理由让我们选择 C++ 而不是基于 JVM 做这件事情。

首先我们前面讲过，目前虽然计算能力对传统应用来说井喷，我们经常说 CPU 过剩、GPU 过剩。但是对于高维机器学习过程来说，依然是资源非常紧张的，这里面包括计算、网络、内存等多个方面，我们看 Spark 的 Project Tungsten 也做了大量堆外内存管理的工作以提升数据的处理效率，但是对机器学习来说，基本上整个过程都需要对内存和计算过程进行精细控制，以及避免不可控的 GC 过程。

其次，C++ 的一些语言特性，比如运算符重载机制也可以使得框架上层的算法应用的语法比较简单优雅，不会变成巨大的一坨，而大规模机器学习很多时候和字符串组成的离散特征打交道，C++ 对字符串处理的效率也要高出 JVM-Based 语言非常多。

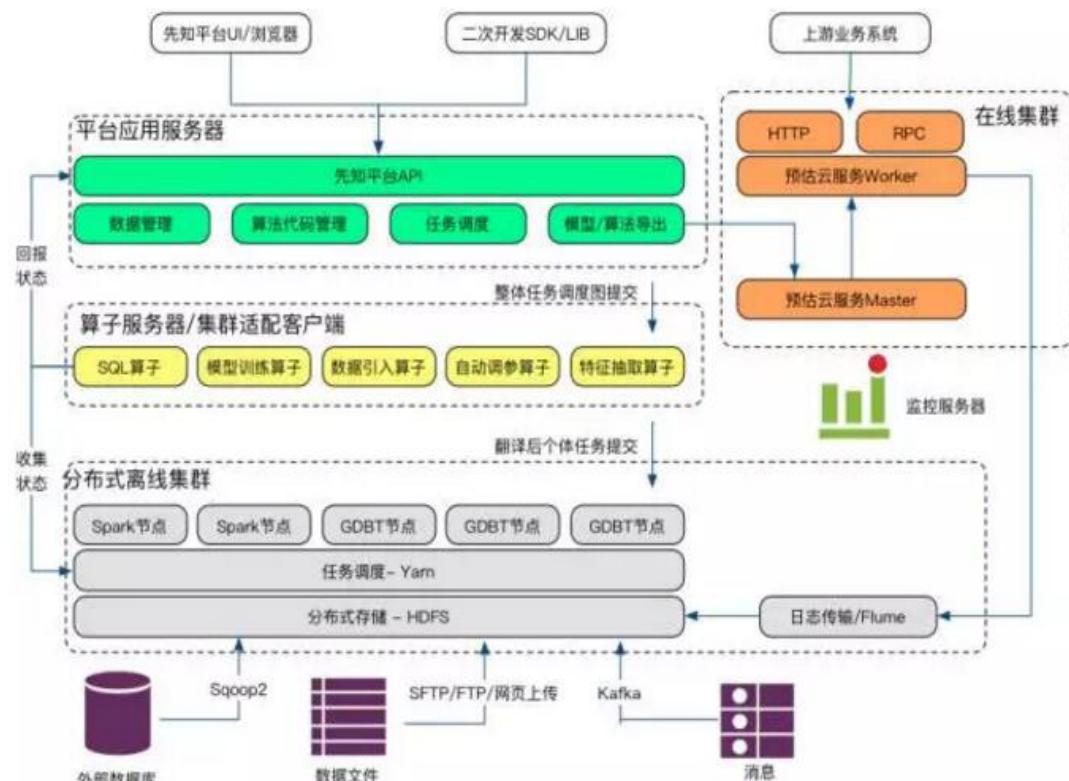
再次，Spark 目前的机制和 Parameter Server 的结合很难做到优雅和完美，强行对接 PS 会破坏 Spark 自身的灾备、任务调度等特性。如果不对接，那么就基本上只能靠降低模型大小来确保效率，和高维度的目标南辕北辙。所以综合考虑，我们还是选择 C++ 来实现整个一套的系统，而将和大数据框架的对接以及开发门槛降低这个任务交给整个机器学习系统的架构。

整体架构

上面讲了很多机器学习算法框架，但有过实践经验的朋友都知道，光有一个算法框架只相当于汽车有了发动机，离开起来还很远。我们总结一个完整的机器学习系统需要有如下部分：

- 数据引入和预处理；
- 特征工程；
- 模型训练算法（支持参数灵活调整和二次开发）；
- 模型评估；
- 模型上线（批量预估、实时API调用、线上特征实时计算）。

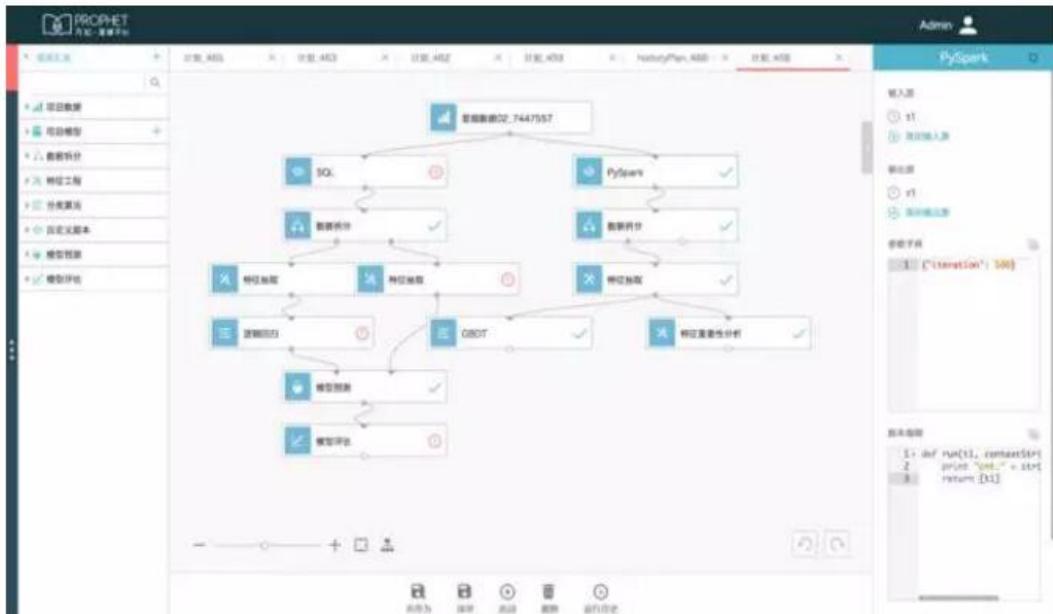
那么对于这所有的部分，我们开发了一个叫『先知』的整体平台产品，我们先看一下先知的整体架构图：



整体来说，先知平台由界面层、调度管理层、集群适配层、集群任务执行层，另外外加一个线上的预估服务云。

这套架构能够给我们带来如下几个优势：

1. 把整个机器学习过程作为一个整体来看待，做到各模块的深度整合。下面是我们产品的一个截图。



可以看到，通过一个 DAG 图的描述，可以将大数据的处理过程（SQL/PySpark/ 数据拆分）和机器学习的过程（特征工程、模型训练、模型评估）整体结合起来，用户不再需要去开发脚本去处理复杂的模块和模块之间的对接以及输入输出的各种判断。

同时呢，这个 DAG 图的背后我们还做了很多工作，一个比较有意思的就是对存储的抽象和适配。我们有的客户是用 AWS 作为基础设施，而有的客户用阿里云做基础设施，有的客户则选择在 IDC 内构建自己的机房。这里面就有块存储、云盘、SSD、内存盘等多种存储服务。

另外，一个完整的机器学习过程，往往还要从数据库里面导入数据，最终要把数据导出到某个指定的位置。导数据在逻辑上的复杂性和由于某些开源模块不支持内存缓存和 SSD 导致的性能的低下都是困扰数据科学家的问题。

对此设计了一个两层的存储抽象层 API，第一层是一个开源项目 alluxio，就是以前的 tachyon，alluxio 这个东西是非常好的，好在哪里呢？他用一套解

解决方案一下子解决了几个问题，第一是可以支持很多分布式的文件系统，包括 S3、Ceph 等。第二还能够比较透明的解决内存和 SSD 加速的问题。针对 Alluxio 这个项目，我们将整个的数据处理、模型训练、预估体系，包括我们自己开发的 GDBT 框架都和 alluxio 做了深度的融合集成，也针对开源的产品在访问调度、吞吐、SSD 优化上做了一些增强和修补的工作。

第二层是 DataManager，我们知道企业的应用，特别在意安全性，那么如果我们直接把底层的文件系统暴露给使用者，会导致权限隔离等个方面的隐患。Datamanager 提供了一个数据的逻辑沙箱，使得每个使用者在自己的 DAG 里面只能用一个唯一的名字来访问到自己安全容器内部的数据，这样不仅保证了安全，也避免了使用者直接去处理各种各样的长的数据文件的 URI。

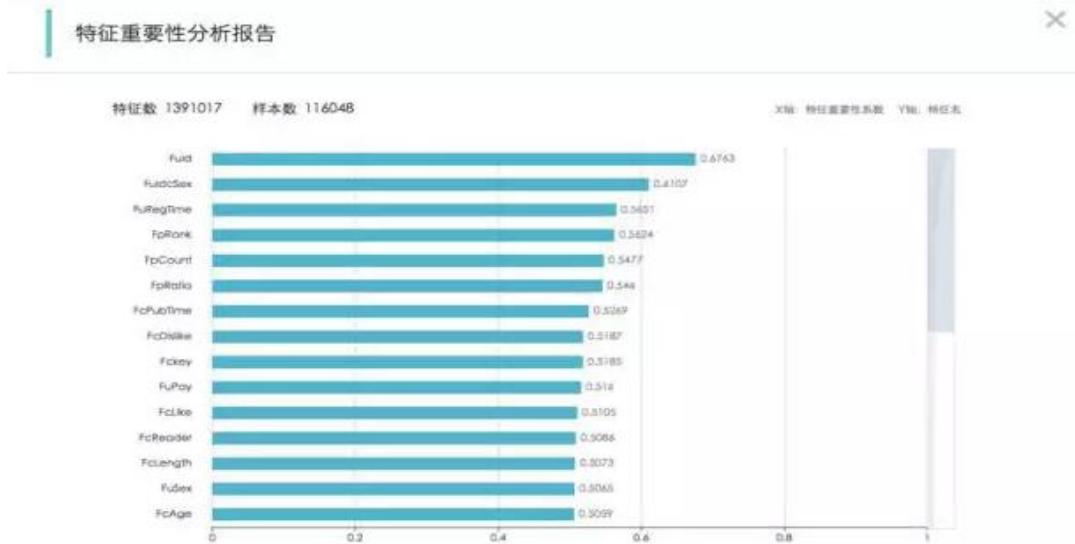
2. 给开发者提供比较好的体验，相信使用大数据系统的朋友都有一个感受，就是在分布式时代，调试变得难了。一方面很多时候日志太过于复杂，不知道错误在哪里，另外一方面经常出现跑了几个小时报一个错前功尽弃的情况。先知平台从产品上做了很多工作，比如说 Schema 推断。

脚本编辑

The screenshot shows a code editor interface. On the left, there is a toolbar with icons for file operations like new, open, save, and close. The main area contains a line of SQL code: `1 select col_3, col_5, cc`. To the right of the code, a red error icon with a white 'X' is displayed. Below the code, a large error message box is shown with the text: `field col_x6 is not found` and the Chinese translation: `字段不存在，请修改`.

我们可以在运行这个 SQL 之前就在界面上交互式的看到哪里出了错误，以便及时改正，而不需要整个 DAG 图跑了几个小时，运行到这个地方的时候，可能人

都去睡觉了报个错，只能第二天再来。另外比如我们还提供了特征重要性评估的功能：



这个功能可以让使用者很快发现自己模型里面的问题，比如某个特征重要性极高，需要考虑自己的模型是不是有看答案（用结果去训练模型，再去评估结果，导致模型线下评估效果非常好，线上无效）的现象。值得一提的是，这个特征重要性评估的算法也是基于 GDBT 框架开发的，GDBT 不仅可以高效的支撑各种模型算法开发，也能够快速的支持各种其他的大规模分布式计算逻辑。

3. 支持模型的上线，模型上线有几个技术上的难点，一是线上线下的一致性，而是性能。线上线下一致性为什么难，因为你看到训练过程是那么复杂一个 DAG 图，那么对应的线上多个数据源，也要经过一个组合、变换的过程，那如果每次都去手动开发，这个代价就不得了。

先知的架构支持从线下的 DAG 导出线上服务的核心部分，那么特征工程、模型计算就可以直接获得一个可用的 API，可以极大的节省开发者的代价。另外一个难题就是性能，因为线下批量的训练，可以花很长时间，而线上如果是实时预估，那么就要毫秒级的响应，也要求很高的 QPS。

我们在线上有一个叫 Cannon 的分布式 KV 框架，可以支持到数十 T 分布式内存的模型高性能存储和查询。而模型计算的部分也可以复用 GDBT 的代码，既减

少了开发量，又为一致性提供了保证。这里面也有很多有意思的工程优化，比如说如何解决机器数变大、网络条件变差情况下的可用性塌方式下降。今天时间有限就不展开说了，有机会可以再跟大家分享。

小结

非常感谢大家耐心听我们分享了上面我们做的这些微小的工作。第四范式是一个以产品研发为主的公司，我们有数十个来自于各大国内外公司顶尖机器学习团队的优秀工程师，各路 ACM 冠军选手每天在各个方向上做很多深入的产品和工程优化工作，希望能够促进机器学习和 AI 在各行各业的发展，也希望能够给从业者们带来更好用的平台和工具。

当然做出好的机器学习解决方案，最关键的还是要和行业结合，工具和平台系统只是其中的一小部分，所以还希望能够向各位同仁多多学习。这里打个小广告，现在先知平台公有云版本已经向业务场景成熟的企业客户有限名额的开放合作，如果有风控、内容推荐、客户经营等场景，数据量较大的朋友，我们也可以一起互相切磋，互相学习。谢谢大家。

Q&A

Q1：请问先知平台用到深度学习框架了么？

答：现有的开源的深度学习框架不能完全解决先知平台客户的问题，因为很多实际的问题，除了包含稠密的连续特征之外，还有大规模的稀疏离散特征，目前大部分的开源框架大多 focus 在稠密连续特征的深度学习问题上，对学术界比较关心的同学，可能可以发现 google 最近有一篇 wide&deep learning 的论文是做类似的事情的，这个解决方案和我们三年前在百度做的解决方案很类似，但是很可惜的是开源的 tensorflow 在这个问题上效率非常糟糕。实际问题的难点在于它是 IO 密集（大规模稀疏）且计算密集的（稠密），而且这样的模型是极其难调的，我们有很多新的算法在解决这方面的问题，不仅仅是深度学习。

Q2：是通过何种机制做到数百倍的加速的？

答：从算法设计到 GDBT 平台设计和底层优化，都有很多工作，今天的讲座里面有涉及到一些，可以翻看记录；

Q3：有没有提出一些最优化框架，比如 SVRG 等来加速收敛？

答：对于不同的问题，不同的场景，会对优化算法有不同的需求，比如收敛速度、稀疏性、稳定性、是否便于实现、是否便于分布式并行、是否访存友好等等，先知支持多种优化算法（batch/stochastic 的都有），比如 lbfsgs、FOBOS、RDA、FTRL、SVRG、Frank-wolfe 等等都会有涉及，同时针对不同的应用场景也需要做一定的改进和调整。

Q4：超参数学习这块是通过 bayes 还是强化学习呢？

答：目前产品中的是 bayesian 的，其他方式也正在尝试。

Q5：在何种情况下如何判定自动特征不能起作用而改为人工特征呢？

答：有很多特征选择的算法和策略，同时，对于实际的问题，可以先用自动特征处理取得一个初步的效果，再从实际模型效果去看是否需要人工介入做进一步的优化，自动特征工程是一个很难的问题，目前其实是很难做到 100% 全自动的，我们平台期望能够尽最大可能的降低人力。

Q6：先知在实现过程中用到参数服务器了没有？模型的训练，是采用异步 asp，还是同步 bsp，还是半同步的 ssp 这种？如果是异步，如何解决收敛困难的问题？

答：前面有提到，我们用到了 parameter server。模型训练支持多种模式，对于不同的算法，不同的计算环境，会采用不同的同步方式。其实目前大部分情况下，如果数据切分 计算调度得当，异步和同步差别没有特别大，计算资源有较大差别的时候，还是建议带版本控制；更好的方式是采用类似的计算资源做好数据切分和计算调度。

Q7：第四范式的先知和谷歌的 Tensor flow 有什么区别？

答：从覆盖范围上，先知是一个机器学习应用全生命周期管理的平台，而 Tensorflow 对应的是先知的 GDBT 部分。而对于 GDBT 和 Tensorflow 的区别来说，

前面也讲到，首先 Tensorflow 更多是为算法研究目的而存在的，已经有一些 benchmark 说明 tensorflow 比目前很多开源的深度机器学习框架都要慢，所以也有个外号叫 Tensorslow。当然我个人觉得这并不公平，因为 Tensorflow 本身也不是为大规模工业应用设计的，只是现在好的框架的确太贫乏了。Tensorflow 的一些设计理念比如高兼容性，跨平台也是很好的。这方面 GDBT 也都有考虑和规划。

Q8：GDBT 有没有解决模型并行训练问题？还是只依靠数据并行？

答：也分不同的算法，GDBT 同时支持模型分布式和数据分布式。

Q9：GDBT 怎么能够同时支持连续、离散的这两种数据的融合训练？

答：解决这个问题，一个是效率，首先框架底层上要能够支持很灵活的调度，能够根据连续和离散的数据的计算特性做针对性的设计，比如连续复杂模型是计算密集，可能需要调度到 GPU 运算，离散数据可能是 IO 密集，需要做好计算调度，资源异步调度；更重要的是在算法上做更多的新的改进，因为学术界大多情况下是分别考虑，我们有一系列自己重新设计的算法。对算法细节有兴趣的同学，可以考虑加入到第四范式，或者等我们的专利公开。

Q10：请问 GDBT 对于异构计算的支持情况如何？

答：GDBT 目前支持 CPU 和 GPU 的异构计算。在百度的时候，我们有过在模型预测时使用 FPGA，不过，最近 GPU 进展不错，比如 nvidia 有一些新的芯片比如 Tesla P4 的出现，可能会对 FPGA 有一定的冲击，对于 FPGA 的使用，目前我们研究上还是跟随状态，并没有集成到先知产品中。

Q11：第四范式的人工智能平台先知可以直接替代 Spark 吗？

答：Prophet 诞生的原因是因为我们为各种行业提供服务，每个行业的差异化乍一看都不少，按照传统的方式，我们需要 case by case 的去应对，给出对应的方法，然后进行实施。但在这个 case by case 的过程中，大部分的精力是花在如何把对领域专有知识的理解（业务理解）转换为机器学习过程的具体操作（数据科学家的工作），对于端到端的两端，数据 和 服务，反而是比较通用的。

那么如果能够利用技术和算法，解决专有知识到对应机器学习过程的映射问题，我们就可以建设一个通用的平台来使得 AI 应用到不同场景的代价变小，实现人工智能的傻瓜机。Prophet 就是这个目标的第一步。

首先 Prophet 定位在一套完整的平台，包括核心机器学习算法框架 GDBT（没错不是 GBDT，这是个算法框架，其作者起名为 General Distributed Brilliant Technology），以及机器学习任务调度框架 TM，以及人机接口 Lamma，还有架设在整个框架上的一系列算子。当然这些都是内部名字无所谓，总的来说 Prophet 提供的是端到端的机器学习能力，进来是数据，出去是 Service。

然后关于 GDBT 和 Spark，应该说对比的是 基于 GDBT 的算法 以及 基于 Spark 的算法 (MLLib 实现)，由于计算架构的不同，所以简单的来说多少多少倍是没有太多的意义，因为如果特征纬度多到一定程度，MLLib 在不做数据采样的情况下是无法完成某些训练的。但是具体在几千万行，几十个核的场景下，快几百倍是实测结果。

另外，我们在做的事情，算法框架是一个部分，性能也是很重要的，但是做这些的目的是为了降低机器学习应用于具体行业的门槛和先决要求。这个先决要求既包含硬件上的，也包含人在机器学习方面知识的要求。拥有更强大的计算能力和特征处理能力，意味着我们可以更少的让人输入信息，而更多的依靠计算机自身的理解和计算来找到机器学习算法在具体问题上应用的最佳结合点，这其中甚至还需要包括如何去利用计算资源的投入避免机器学习常见的一些缺陷。

因此 Prophet 不会替代 Spark，Prophet 里面的很多组件也是基于 Spark 的，Prophet 的目标是把 AI 的能力较为容易的带到各个应用场景，为了这个目标，我们会利用好 GDBT，也会极致的利用好 Spark，也会利用硬件技术的最新进展。一切为了 AI for everyone。（以上部分来源于知乎：<https://www.zhihu.com/question/48743915#>）

简单的来说，先知的设计范围超出了 Spark，包含了 Spark，所以不能说是替代。

Q12：什么样的企业用得起机器学习来辅助运营？使用你们机器学习系统的门槛是什么？

答：从目前来看，需要有一个好的业务场景和足够的数据。互联网的 APP 或者非常大的传统行业里面的推荐、营销、定价等场景都比较适合。数据量小的就要看，通常来说 10 万多样本分布均匀就有这个可能。用这个机器学习系统目前的门槛是首先要能理解数据和业务，有一定的统计的背景和思路，然后就是能够导入导出数据，最后就是阅读一下先知的使用手册和培训视频。

Q13：电商推荐平台，怎么样能最快地应用机器学习的精准推荐？

答：对于推荐场景，我们有相对比较成熟的接入方案，可以快速通过数据和 API 接入，通过公有云的 SaaS 服务享受到 GDBT 的能力以及先知的整体效果。有需求的朋友可以关注我们的官方网站和公众号（NextParadigm），我们会近期放出先知推荐的试用邀请。

Q14：机器学习目前哪些企业和行业应用比较广泛？国内有哪些成功案例。

答：大规模机器学习，BAT 今日头条等，广告推荐为主，成不成功请看他们收钱增长的速度特别是百度 09 年之后的增长。我们在银行最近的探索也有很多成功的例子，比如在营销和定价、反欺诈方面。另外风控一向是机器学习的主战场。

Q15：自动特征这套做法，跟百度凤巢的那套是一样的对吧？百度有公开论文，是 gradient boosting factorization machine，这个方法比深度学习那个自动特征相比如何。

答：做法和夏粉老师的那套不一样；夏老师和张潼老师这篇文章和 nn-based 的各有优劣。其实 NN 没有大家想的那么万能，“人工”的很多 feature combination 是 NN 学不出来的，其中有很多有趣的问题，这里就不赘述了，可以再交流。

另外：对于 FPGA 和 GPU 的未来我们有一段简单的思考，之前有准备过一段，之前没用上，现在贴这里：FPGA 是作为专用集成电路领域中的一种半定制电路

而出现的，既解决了全定制电路的不足，又克服了原有可编程逻辑器件门电路数有限的缺点。机器学习尤其是深度学习是计算密集型的，比如深度学习里面有大量的浮点矩阵运算这种并行浮点运算需求，传统的 CPU 从设计上而言已经很难满足这种大规模浮点计算密集型任务。目前针对这种机器学习任务，CPU 主流的替代选择是 FPGA 和 GPU。

GPU 是固定的计算架构的计算设备，有着良好的软件编程接口，但是对于特定的计算模式和模型结构不一定是最优的选择。FPGA 本身是一种可编程的硬件，对于有研发能力的厂商而言，深度优化过的 FPGA，相比 GPU，能够提供更专有的硬件加速，更重要的是 FPGA 在单位能耗上能提供的计算能力要高于 GPU。

另外，企业级 FPGA 也比企业级 GPU 便宜很多。但是 FPGA 的缺点也非常明显，对相关专业人才的要求非常高，需要能够将复杂的机器学习算法映射到硬件逻辑上，同时提供高吞吐和低延迟，开发难度很大。对于中小公司而言，如果没有相应的 FPGA 研发能力，或者无法支撑高昂的研发成本，在机器学习尤其是深度学习的训练 / 预估硬件解决方案上可以选择架构固定的 GPU。

对于有相应能力的大中型企业而言，FPGA 带来的硬件成本的大幅降低和能耗的大幅降低，能够轻易覆盖研发团队的费用，在机器学习尤其是深度学习预估的硬件解决方案上 FPGA 会是更合适的选择。

同时，在制造工艺上，相比于 FPGA，GPU 是固定的专用计算架构且不提供可编程能力，可以通过不断的芯片优化，提供更强的计算能力，对于计算更加密集的深度学习训练任务，GPU 现在还是更合适的选择。随着 GPU 在芯片上不断优化提升和能耗的降低，以及 FPGA 不断提升芯片可提供的计算能力，两者的差距在不断缩小，未来两者的地位是否会有大的变革，值得期待。

讲师介绍

胡时伟，第四范式联合创始人，研发副总裁。在大规模机器学习、广告、搜索、行业垂直应用、系统运维、研发团队管理等领域拥有丰富经验。曾主持架构

了百度“知心”系统和链家网系统，在百度任职期间作为系统架构负责人，主持了百度商业客户运营、凤巢新兴变现、“商业知心”搜索、阿拉丁生态等多个核心系统的架构设计工作。在担任链家网研发负责人期间，从 0 开始完成了链家网新主站、经纪人新作业系统、绩效变革系统的整体架构设计以及研发团队的建设管理，参与规划及推动了链家系统和研发体系的互联网化转型。现任第四范式研发总工程师，负责产品技术团队以及第四范式核心产品『先知』机器学习平台的研发工作。

涂威威，第四范式数据建模专家。在大规模分布式机器学习系统架构、大规模机器学习算法设计和应用、在线营销系统方面有深厚积累。百度最高奖 trinity 发起人之一。首次将 FPGA 应用于在线营销深度学习预估系统。设计开发百度机器学习计算框架 ELF。

【延伸阅读】

腾讯大数据宣布开源第三代高性能
计算平台 Angel：支持十亿维度



轻量级大规模机器学习算法库 Fregata： 快速，无需调参

作者 张夏天

一. 大规模机器学习的挑战

随着互联网，移动互联网的兴起，可以获取的数据变得越来越多，也越来越丰富。数据资源的丰富，给机器学习带来了越来越多，越来越大创造价值的机会。机器学习在计算广告，推荐系统这些价值上千亿美元的应用中起到的作用越来越大，创造的价值也越来越大。但是越来越大的数据规模也给机器学习带来了很多挑战。

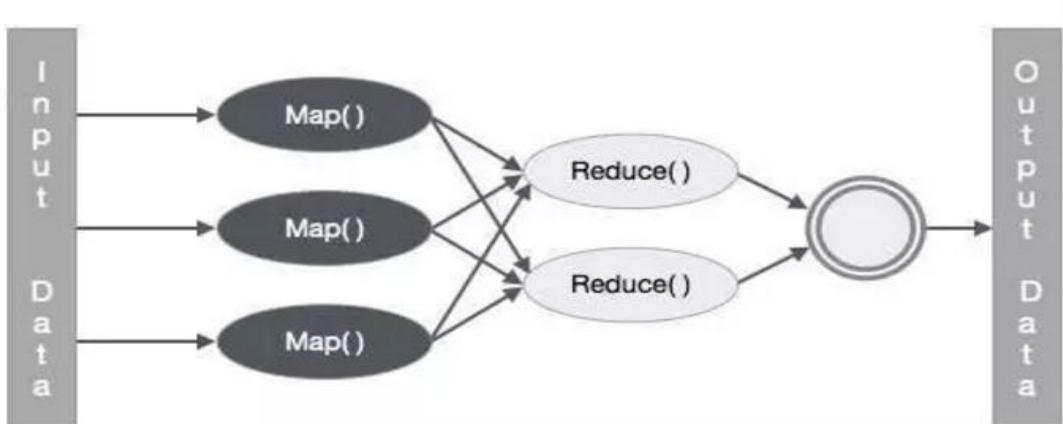
最大的挑战就是庞大的数据量使得对计算资源的需求也急剧增长。首先经典的机器学习算法其计算量基本上都是与训练数据条数或者特征数量呈二次方甚至是三次方关系的^[1]。即是说数据量或者特征数每翻一倍，则计算量就要增加到原来的四倍，甚至是八倍。这样的计算量增长是十分可怕的，即使是采用可扩展的计算机集群也难以满足这样的计算量增长。好在对于很多依赖于凸优化方法的算法，可以采用随机梯度下降方法，将计算量的增长降到基本与数据量和特征数

呈线性关系。但是，大规模机器学习在计算上依然有三个比较大的困难。

第一，因为几乎所有的机器学习算法都需要多次扫描数据，对于大规模数据无论在什么平台上，如果不能全部存储在内存中，就需要反复从磁盘存储系统中读取数据，带来巨大的 I/O 开销。在很多情况下，I/O 开销占到整个训练时间的 90% 以上。

第二，即使有足够的资源将所有数据都放到内存中处理，对于分布式的计算系统，模型训练过程中对模型更新需要大量的网络通信开销。无论是同步更新还是异步更新，庞大的数据量和特征数都足以使得通信量爆炸，这是大规模机器学习的另外一个瓶颈。

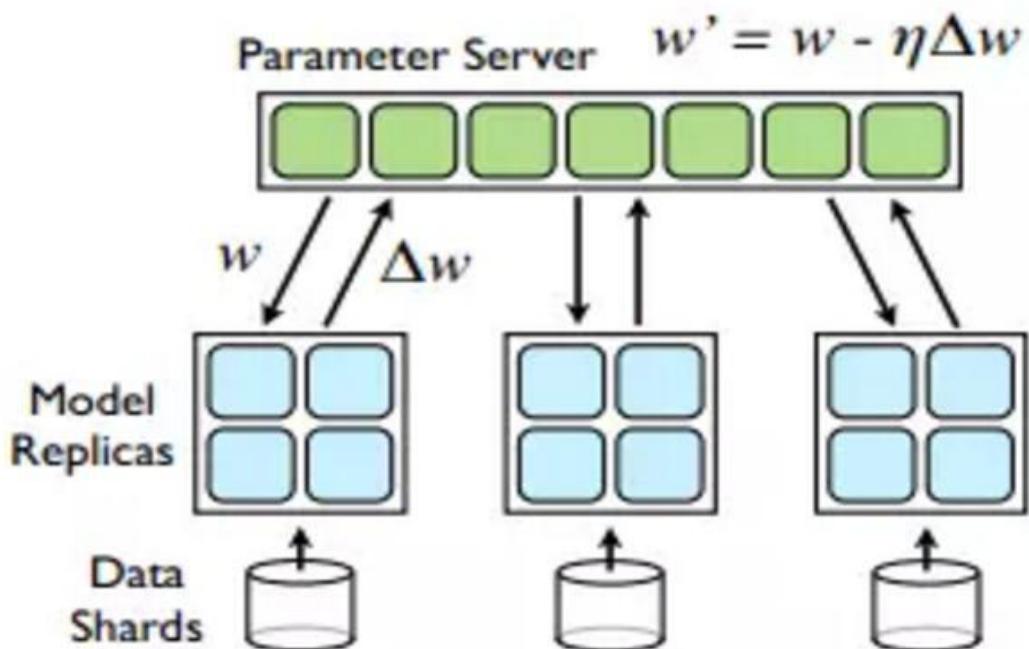
第三，大规模的模型使得无法在一个节点上存储整个模型，如何将模型进行分布式的管理也是一个比较大的挑战。



目前的主流大数据处理技术都是以 Map Reduce 计算模式为核心的（包括 Hadoop 和 Spark）。而 Map Reduce 计算模式下对第一个问题只能通过增加内存，SSD 存储来解决或者缓解，但同时也需要付出高昂的成本。对于第二，和第三个问题，Map Reduce 模式的局限是很难克服这两个问题的。

而 Parameter Server^[2] 作为目前最新的大规模机器学习系统设计模式，主要解决的其实是第三个问题，通过参数服务器以及模型参数的分布式管理来实现对超大规模模型的支持。同时在模型更新过程中的通信模式可以采用异步更新模

式，来减小数据同步的开销，提高通信效率，但是 Parameter Server 模式下模型的更新量计算和模型的更新是分离的，其庞大的通信开销在原理上就是不可避免的。幸运的是，常见的大规模机器学习问题，都是高维稀疏问题，在很大程度上缓解了通信开销的问题，而 Parameter Server 突破了模型规模限制的优点是 Map Reduce 模式无法取代的，所以 Parameter Server 成为了目前大规模机器学习最先进，最受认可的模式。



虽然 Parameter Server 解决了模型分布式管理的瓶颈，异步通信模式和问题本身的稀疏性大大降低了通信的压力。但是机器学习算法本身对数据的多次扫描带来的计算和通信开销依然是大规模机器学习效率的很大瓶颈。

除此之外还有一个很大的挑战就是算法的调参工作，一般机器学习算法都会依赖一个或者多个参数，对于同一问题，不同的参数设定对模型精度的影响是很大的，而同一参数设定在不同的问题上的效果也有很大的不同。对于从事机器学习工作的人来说，调参始终是一个令人头疼的问题。知乎上有个问题是“调参这事儿，为什么越干越觉得像老中医看病？”^[3]，里面有不少关于机器学习调参的经验，心得，吐槽和抖机灵。

对于大规模机器学习问题，调参的难度显然是更大的：

首先，一次训练和测试过程的时间和计算资源开销都是庞大的，不管采用什么调参方法，多次实验都会带来很大的时间和计算资源消耗。

其次，大规模机器学习问题通常都是数据变化很快的问题，如计算广告和推荐系统，之前确定好的参数在随着数据的变化，也有劣化的风险。

目前来说大规模机器学习存在的主要挑战是两个：第一是计算资源的消耗比较大，训练时间较长的问题，第二是调参比较困难，效率较低。TalkingData 在大规模机器学习的实践中也深受这两个问题的困扰，特别是公司在早起阶段硬件资源十分有限，这两个问题特别突出。我们为了解决这个问题，做了很多努力和尝试。TalkingData 最近开源的 Fregata 项目^[4]，就是我们在这方面取得的一些成果的总结。

二. Fregata的优点

Fregata 是 TalkingData 开源的大规模机器学习算法库，基于 Spark，目前支持 Spark 1.6.x，很快会支持 Spark 2.0。目前 Fregata 包括了 Logistic Regression, Softmax, 和 Random Decision Trees 三中算法。

三种算法中 Logistic Regression, Softmax 可以看作一类广义线性的参数方法，其训练过程都依赖于凸优化方法。我们提出了 Greedy Step Averaging^[5] 优化方法，在 SGD 优化方法基础上实现了学习率的自动调整，免去了调参的困扰，大量的实验证明采用 GSA 优化方法的 Logistic Regression 和 Softmax 算法的收敛速度和稳定性都是非常不错的，在不同数据规模，不同维度规模和不同稀疏度的问题上都能取得很好的精度和收敛速度。

基于 GSA 优化方法，我们在 Spark 上实现了并行的 Logistic Regression 和 Softmax 算法，我们测试了很多公开数据集和我们自己的数据，发现在绝大部分数据上都能够扫描一遍数据即收敛。这就大大降低了 IO 开销和通信开销。

其中 Logistic Regression 算法还有一个支持多组特征交叉的变种版本，其

不同点是在训练过程中完成维度交叉，这样就不需要在数据准备过程中将多组特征维度预先交叉准备好，通常这意味着数据量级上的数据量膨胀，给数据存储和 IO 带来极大压力。而这种多组特征交叉的需求在计算广告和推荐系统中又是非常常见的，因此我们对此做了特别的支持。

而 Random Decision Trees^{[6][7]} 算法是高效的非参数学习方法，可以处理分类，多标签分类，回归和多目标回归等问题。而且调参相对也是比较简单的。但是由于树结构本身比较复杂而庞大，使得并行比较困难，我们采用了一些 Hash Trick 使得对于二值特征的数据可以做到扫描一遍即完成训练，并且在训练过程中对内存消耗很少。

总结起来，Fregata 的优点就两个，第一是速度快，第二是算法无需调参或者调参相对简单。这两个优点降低了减少了计算资源的消耗，提高了效率，同时也降低了对机器学习工程师的要求，提高了他们的工作效率。

三. GSA算法介绍

GSA 算法是我们最近提出的梯度型随机优化算法，是 Fregata 采用的核心优化方法。它是基于随机梯度下降法 (SGD) 的一种改进：保持了 SGD 易于实现，内存开销小，便于处理大规模训练样本的优势，同时免去了 SGD 不得不人为调整学习率参数的麻烦。

事实上，最近几年关于 SGD 算法的步长选取问题也有一些相关工作，像 Adagrad, Adadelta, Adam 等。但这些方法所声称的自适应步长策略其实是把算法对学习率的敏感转移到了其他参数上面，并未从本质上解决调参的问题，而且他们也引入了额外的存储开销。GSA 和这些算法相比更加轻量级，易于实现且易于并行，比起 SGD 没有额外的内存开销，而且真正做到了不依赖任何参数。

我们把 GSA 算法应用于 Logistic 回归和 Softmax 回归，对 libsvm 上 16 个不同类型规模不等的数据集，和 SGD, Adadelta, SCSG (SVRG 的变种) 这些目前流行的随机优化算法做了对比实验。结果显示，GSA 算法在无需调任何参数的情

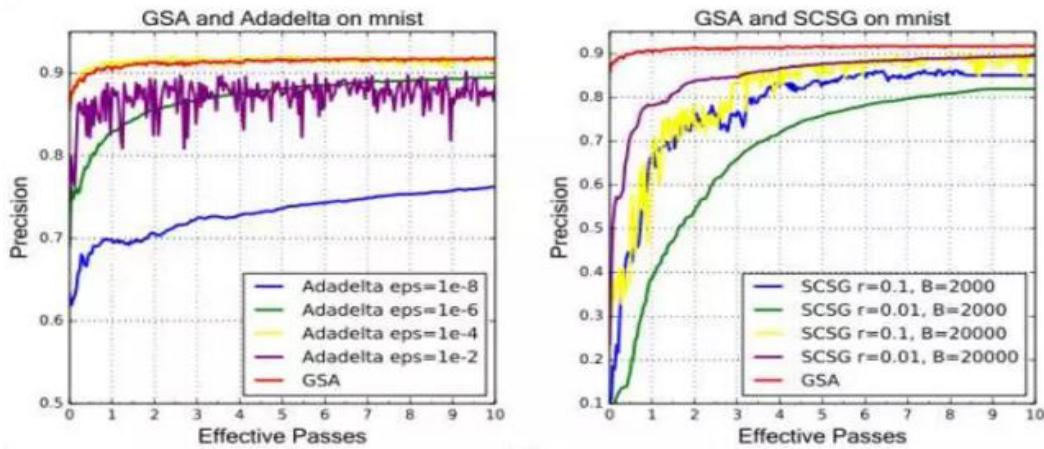


图 3

况下，和其他算法做参数调优后的最佳表现不相上下。此外，GSA 比起这些流行的方法在计算速度和内存开销方面也有一定的优势。

GSA 算法的核心原理非常简单：在迭代的每一步对单个样本的损失函数做线搜索。具体来说，我们对逻辑回归和 softmax 回归的交叉熵损失函数，推导出了一套仅用当前样本点的梯度信息来计算精确线搜索步长的近似公式。我们把利用这套近似公式得到的步长做时间平均来计算当前迭代步的学习率。这样做有两方面的好处：基于精确线搜索得到的步长包含了当前迭代点到全局极小的距离信息——接近收敛时步长比较小，反之则更大，因而保证收敛速度；另一方面平均策略使算法对离群点更鲁棒，损失下降曲线不至剧烈抖动，为算法带来了额外的稳定性。

四. GSA算法Spark上的并行化实现

GSA 算法是基本的优化方法，在 Spark 上还需要考虑算法并行化的问题。机器学习算法的并行化有两种方式，一种是数据并行，另一种是模型并行。但是 Spark 只能支持数据并行，因为模型并行会产生大量细粒度的节点间通信开销，这是 Spark 采用的 BSP 同步模式无法高效处理的。

数据并行模式下进行机器学习算法的并行化又有三种方法，分别是梯度平均，模型平均，以及结果平均。梯度平均是在各个数据分片上计算当前的梯度更新量

然后汇总平均各分片上的梯度更新量总体更新模型。模型平均是各分片训练自己的模型，然后再将模型汇总平均获得一个总体的模型。而结果平均实际上就是 Ensemble Learning，在大规模问题上因为模型规模的问题，并不是一个好的选择。

实际上是目前采用得最多的是梯度平均，当前 Parameter Server 各种实现上主要还是用来支持这种方式，Spark MLLib 的算法实现也是采用的该方式。但是在 Spark 上采用梯度平均在效率上也有比较大的瓶颈，因该方法计算当前的梯度更新量是要依赖于当前的最新模型的，这就带来了在各数据分片之间频繁的模型同步开销，这对 Map Reduce 计算模式的压力是较大的。

模型平均一直被认为其收敛性在理论上是没有保证的，但是最近 Rosenblatt^[8] 等人证明了模型平均的收敛性。而我们在大量的测试中，也发现模型平均通常能取得非常好的模型精度。考虑到模型平均的计算模式更适合 Map Reduce 计算模型，我们在 Fregata 中对于 GSA 算法的并行方法采用的就是模型平均方法。模型平均的并行方法中，每个数据分片在 Map 阶段训练自己的模型，最后通过 Reduce 操作对各个分片上的模型进行平均，扫描一次数据仅需要做一次模型同步。而且在大量的实验中，该方法在扫描一次数据后，模型的精度就可达到很高的水平，基本接近于更多次迭代后的最好结果。

五. Fregata与MLLib对比

Fregata 是基于 Spark 的机器学习算法库，因此与 Spark 内置算法库 MLLib 具有很高的可比性。我们这里简要介绍了三个数据集，两种算法（Logistic Regression 和 Softmax）上的精度和训练时间对比。精度指标我们采用的是测试集的 AUC。对于精度和训练时间，算法每扫描完一次数据记录一次。

Fregata 的算法不需要调参，因此我们都只做了一次实验。而对于 MLLib 上的算法，我们在各种参数组合（包括优化方法的选择）中进行了网格搜索，选取了测试集 AUC 能达到最高的那组参数作为对比结果。

Lookalike 是一个基于 Fregata 平台运用比较成熟的服务，其目标在于根据种子人群进行人群放大以寻找潜在客户。我们将 Lookalike 作为二分类问题来处理，因此可以采用 Logistic Regression 算法来处理。在训练模型时以种子人群为正样本，其他为负样本，通常种子人群的数量不会很多，因此 Lookalike 通常是正样本比例非常少的 class imbalance 问题。在一个大规模数据集（4 亿样本，2 千万个特征）上的 Lookalike 问题的模型训练中，我们对比了 Fregata LR 和 MLLib LR 的性能。

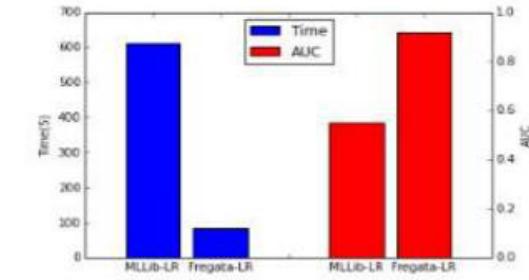
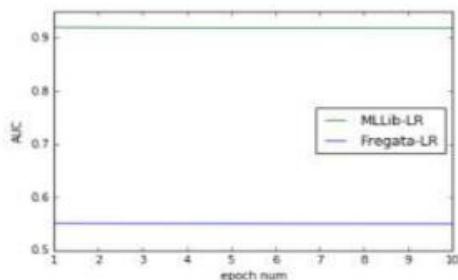


图 4

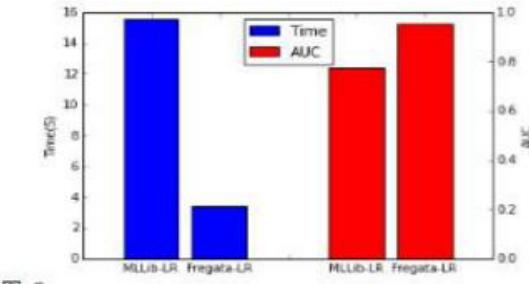
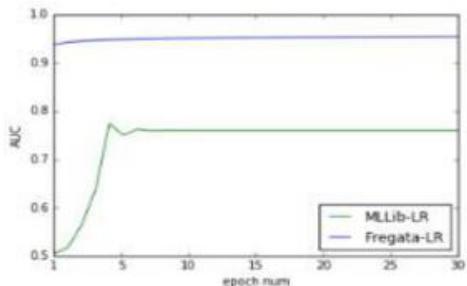
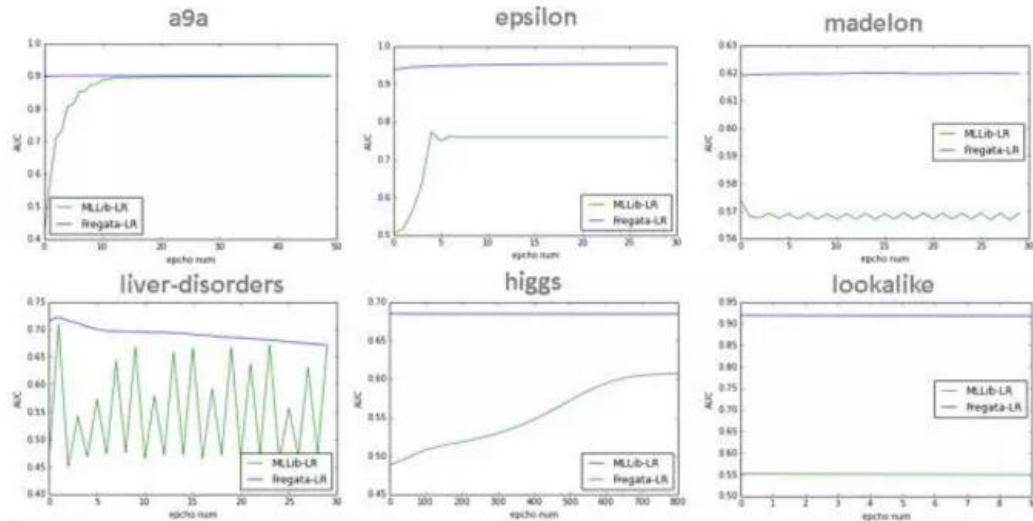


图 5

从图 4 中可以看到 Fregata 的 LR 算法扫描一次数据即收敛达到 AUC（测试集上）的最高值 0.93。在这个数据样本中 而 MLLib 的 LR 算法，即使我们通过调参选取了最好的 AUC 结果，其 AUC 也仅为 0.55 左右。模型预测精度差别非常大。另外，MLLib 的训练时间（达到最高精度的时间）也是 Fregata 大约 6 倍以上。

在公开数据集 epsilon^[9] 上（40 万训练集，2000 特征），Fregata LR 无论从收敛速度还是模型效果与 MLLib LR 相比也有较大的优势。从图 5 中可以看到，在这个数据集上 Fregata LR 在迭代一次以后就在测试集上非常接近最好的结果了，而 MLLib LR 需要 5 次迭代而且最高的精度比 Fregata LR 相差很大，而训练

时间 MLLib LR 也是 Fregata LR 的 5 倍以上。



另外图 6 展示了 Fregata LR 与 MLLib LR 在 6 个不同问题上的测试集 AUC 曲线，可以看到 Fregata LR 算法在不同问题上收敛速度和稳定性相较于 MLLib LR 都是有较大的优势。Fregata LR 在第一次迭代后，AUC 就已经基本收敛，即使与最高值还有一些差距，但是也是非常接近了。

我们也在公开数据集 MNIST 上测试了 Softmax 算法。从图 7 中可以看到，Fregata Softmax 也是一次扫描数据后在测试集上的 AUC 就非常接近最好的结果，而 MLLib Softmax 需要扫描数据多达 40 多次以后才接近 Fregata Softmax 一次扫描数据的结果。对比两个算法在达到各自最优结果所花的时间，MLLib Softmax 是 Fregata Softmax 的 50 倍以上。

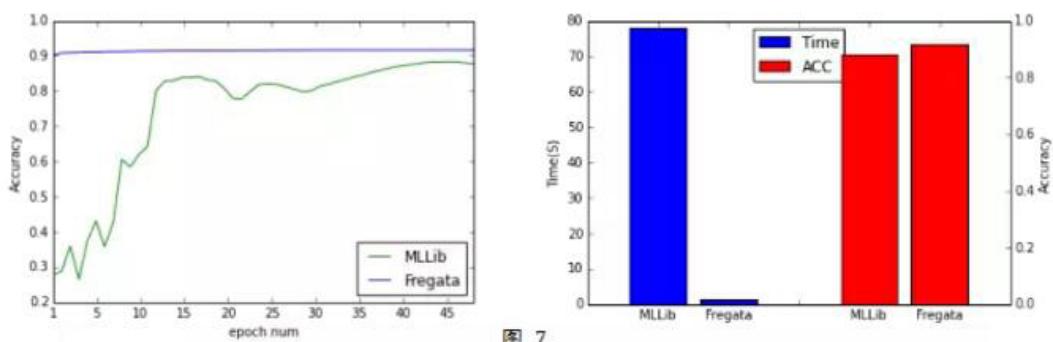


图 7

Fregata 致力于在 Spark 上解决大规模机器学习的问题，Fregata 目前已经公开发布的版本已经能支持亿级维度的模型，而目前内部最新版本已经在一个月内连续突破了 10 亿，100 亿，1000 亿和 10000 亿 4 个台阶。在模型规模提高了 4 个数量级的同时保持了训练的高效性。下面是 Fregata 的 Logistic Regression 算法在 511412394 个样本的训练集下的训练时间。

维度	Spark 配置（2.0）	训练时间(秒)
20 Millions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	469
400 Millions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	455
800 Millions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	449
1 Billions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	487
2 Billions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	449
10 Billions	48 Executors, 1 Core/Executor, 1G/Executor&Driver	473
100 Billions	48 Executors, 1 Core/Executor, 2G/Executor&Driver	481
1000 Billions	48 Executors, 1 Core/Executor, 8G/Executor&Driver	814

从上表可以看出，对于 5 亿多样本的训练集，在仅使用 48 个 Executor 的情况下，千亿维度以内的问题，都可在 500 秒内完成，而且每个 Executor 仅需最多 2G 内存。对于万亿维度的问题，训练时间也仅需 800 秒多一点，只是 Executor 的内存加到了 8G。Fregata 最近的突破，打破了在 Spark 上无法支持超大规模模型的瓶颈，将进一步降低大规模机器学习的使用门槛和成本。

六. Fregata的使用简介

前面简要介绍了 Fregata 算法库涉及到的一些技术原理和性能对比，我们再来看看 Fregata 的使用方式。可以通过 3 种不同的方式来获取 Fregata 如果使用 Maven 来管理工程，则可以通过添加如下代码在 pom.xml 中进行引入，

```
<dependency>
    <groupId>com.talkingdata.fregata</groupId>
    <artifactId>core</artifactId>
    <version>0.0.1</version>
</dependency>
<dependency>
```

```
<groupId>com.talkingdata.fregata</groupId>
<artifactId>spark</artifactId>
<version>0.0.1</version>
</dependency>
```

如果使用 SBT 来管理工程，则可以通过如下代码在 build.sbt 中进行引入，

```
// 如果手动部署到本地maven仓库，请将下行注释打开
// resolvers += Resolver.mavenLocal
libraryDependencies += "com.talkingdata.fregata" % "core" %
"0.0.1"
libraryDependencies += "com.talkingdata.fregata" % "spark" %
"0.0.1"
```

如果希望手动部署到本地 maven 仓库，可以通过在命令中执行如下命令来完成：

```
git clone https://github.com/TalkingData/Fregata.git
cd Fregata
mvn clean package install
```

接下来，让我们以 Logistic Regression 为例来看看如何快速使用 Fregata 完成分类任务：

1. 引入所需包

```
import fregata.spark.data.LibSvmReader
import fregata.spark.metrics.classification.{AreaUnderRoc,
Accuracy}
import fregata.spark.model.classification.LogisticRegression
import org.apache.spark.{SparkConf, SparkContext}
```

2. 通过 Fregata 的 LibSvmReader 接口加载训练及测试数据集，训练及测试数据集为标准 LibSvm 格式，可参照^[10]

```
val (_, trainData) = LibSvmReader.read(sc, trainPath,
numFeatures.toInt)

val (_, testData) = LibSvmReader.read(sc, testPath,
numFeatures.toInt)
```

3. 针对训练样本训练 Logistic Regression 模型

```
val model = LogisticRegression.run(trainData)
```

4. 基于已经训练完毕的模型对测试样本进行预测

```
val pd = model.classPredict(testData)
```

5. 通过 Fregata 内置指标评价模型效果

```
val auc = AreaUnderRoc.of( pd.map{
    case ((x,l),(p,c)) =>
    p -> l
})
```

在Fregata中，使用breeze.linalg.Vector[Double]来存储一个样本的特征，如果数据格式已经是LibSvm，则只需通过Fregata内部的接口LibSvmReader.read(...)来加载即可。否则，可以采用如下的方法将代表实例的一组数据封装成breeze.linalg.Vector[Double]即可放入模型中进行训练及测试。

```
// indices  Array类型，下标从0开始，保存不为0的数据下标
// values   Array类型， 保存相当于indices中对应下标的数据值
// length   Int类型， 为样本总特征数
// label    Double类型， 为样本的标签。如果是测试数据，则不需该字段
sc.textFile(input).map{
    val indices  = ...
    val values   = ...
    val label    = ...
    ...
    (new SparseVector(indices, values, length).asInstanceOf[Vector],
     asNum(label))
}
```

七. Freagata的发展目标

Fregata 目前集成的算法还不多，未来还会继续扩充更多的高效的大规模机器学习算法。Fregata 项目追求的目标有 3 个：轻量级，高性能，易使用。

轻量级是指 Fregata 将尽可能在标准 Spark 版本上实现算法，不另外搭建计算系统，使得 Fregata 能够非常容易的在标准 Spark 版本上使用。虽然 Spark 有

一些固有的限制，比如对模型规模的限制，但是作为目前大数据处理的基础工具，Fregata 对其的支持能够大大降低大规模机器学习的应用门槛。毕竟另外搭建一套专用大规模机器学习计算平台，并整合到整个大数据处理平台和流程中，其成本和复杂性也是不可忽视的。

高性能就是坚持高精度和高效率并举的目标，尽可能从算法上和工程实现上将算法的精度和效率推到极致，使得大规模机器学习算法从笨重的牛刀变成轻快的匕首。目前对 Fregata 一个比较大的限制就是模型规模的问题，这是基于 Spark 天生带来的劣势。未来会采用一些模型压缩的方法来缓解这个问题。

易使用也是 Fregata 追求的一个目标，其中最重要的一点就是降低调参的难度。目前的三个算法中有两个是免调参的，另一个也是相对来说调参比较友好的算法。降低了调参的难度，甚至是免去了调参的问题，将大大降低模型应用的难度和成本，提高工作效率。

另一方面我们也会考虑某些常用场景下的特殊需求，比如 LR 算法的特征交叉需求。虽然通用的 LR 算法效率已经很高，但是对于特征交叉这种常见需求，如果不把特征交叉这个过程耦合到算法中去，就需要预先将特征交叉好，这会带来巨大的 I/O 开销。而算法实现了对特征交叉的支持，就规避了这个效率瓶颈。未来在集成更多的算法的同时，也会考虑各种常用的场景需要特殊处理的方式。

Fregata 项目名称的中文是军舰鸟，TalkingData 的开源项目命名都是用的鸟名，而军舰鸟是世界上飞得最快的鸟，最高时速达到 418km/ 小时，体重最大 1.5 公斤，而翼展能够达到 2.3 米，在全球分布也很广泛。我们希望 Fregata 项目能够像军舰鸟一样，体量轻盈，但是能够支持大规模，高效的机器学习，而且具有很强的适用性。目前 Fregata 还是只雏鸟，期望未来能够成长为一只展翅翱翔的猛禽。

引用

1. Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R.

Bradski, Andrew Y. Ng, Kunle Olukotun, Map–Reduce for Machine Learning on Multicore, NIPS, 2006.

2. <https://www.zhihu.com/question/48282030>
3. <https://github.com/TalkingData/Fregata>
4. <http://arxiv.org/abs/1611.03608>
5. <http://www.ibm.com/developerworks/cn/analytics/library/ba-1603-random-decisiontree-algorithm-1/index.html>
6. <http://www.ibm.com/developerworks/cn/analytics/library/ba-1603-random-decisiontree-algorithm-2/index.html>
7. Rosenblatt J D, Nadler B. On the optimality of averaging in distributed statistical learning[J]. Information and Inference, 2016: iaw013 MLA
8. <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon>
9. <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

作者介绍

张夏天, TalkingData 首席数据科学家, 12 年大规模机器学习和数据挖掘经验, 对推荐系统、计算广告、大规模机器学习算法并行化、流式机器学习算法有很深的造诣; 在国际顶级会议和期刊上发表论文 12 篇, 申请专利 9 项; 前 IBM CRL、腾讯、华为诺亚方舟实验室数据科学家; KDD2015、DSS2016 国际会议主题演讲; 机器学习开源项目 Dice 创始人。

Twitter 机器学习平台的设计与搭建

作者 郭晓江

本文简单介绍一下 Twitter 机器学习平台的设计与搭建，也希望从规模化机器学习平台的角度来主要讲一些我们在这个过程中所遇到的各种坑，以及我们做的各种的努力，也希望能对大家有一点用处。

咱们今天下午的专题是“大数据”专题，机器学习和大数据是密不可分的。如果我们将数据比作一座金矿，机器学习就是挖掘金矿的工具。俗话说：顺势而为。那么机器学习在近些年来也是发展越来越好，应用越来越广，我认为主要得益于以下几个趋势：

1. Data Availability

我们可以获得的数据量越来越大，一会在下一张 slide 中也会讲到如果数据量越大，我们模型的质量会有显著提高；

2. Computation Power越来越强

比如最近出现的云计算、GPU、TPU 等等。在上世纪九十年代其实神经网络的理论就已经有了，也就是深度学习的这些理论已经有了，但是当时并没有

火起来，甚至一度被学术界认为这是一个没有前途的方向，就是因为当时这个 computation power 没有到位。

随着近年来这些方面的不断提高，使得我们可以训练更复杂的模型。大家都知道如果你没有太多的数据或者 computation power 不够多，则只能在一个小数据集上做训练，如果模型非常复杂就会出现过度拟合（Overfit）。所以只有当我们把这些问题全都克服之后，我们才可以训练更复杂的模型，得到一些更好的结果；

3. Development in Algorithms

整个算法的发展，会有无数机器学习的研究者，他们不断地去 push the boundary of machine learning。

从大数据和模型的表现关系来看，在十几年前有两个研究者他们将当时几个机器学习比较常用的算法，在一个具体的机器学习的问题上做了一个实验：

More Data or Better Model?

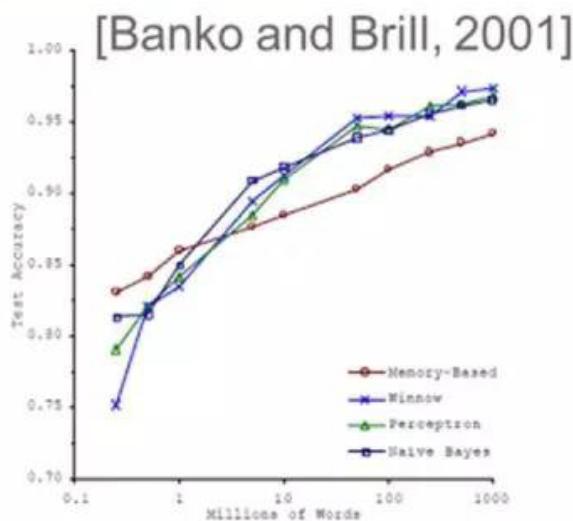


Figure 1. Learning Curves for Confusion Set Disambiguation

这张图的横轴是数据量，即训练数据的数据量，它是一个指数的规模(Scale)。最左边的刻度应该是 10 万个数据点、100 万个数据点和 1000 万个数据点以此类

推；纵轴是模型的表现，即训练出来模型的质量。

大家可以非常清楚地在图中看到，当数据量很小的时候，例如 10 万个数据点时这几个算法的质量非常差，当数据量逐渐增大的时候，模型的质量显著地提高，而且任何一个算法在大数据量时的表现都比任何一个算法在小数据级的表现下要好很多。当然这是在某一个具体的机器学习问题上面做的实验，但是我觉得它有一定的推广价值。它给我们的启示是：如果机器学习的平台架构不够规模化，只能在小数据级上做训练，哪怕你算法做得再好也是徒劳，不如先解决规模化的问题，先在大数据上能够做这样一个训练，然后在算法上再做提高。

说到 Twitter，机器学习在 Twitter 是非常重要的。我们有内部的研究表明：大概 80% 的 DAU 都是直接和机器学习相关产品相关的，90% 的营收来源于广告，而广告完全是由机器学习来支持的。我们现在做的机器学习平台支持了 Twitter 很核心的业务，包括：

- ads ranking (广告排序) ;
- ads targeting;
- timeline ranking (feed ranking) ;
- anti-spam;
- recommendation;
- moments ranking;
- trends

Twitter 的机器学习规模也非常大，我们拿广告来举例子，每天在 Twitter 大概是做 10 个 trillion 量级的广告预测，每个模型的 weights 个数大概是 10 个 million 的量级，每个 training example 大概是有几千到 1 万个 features，每一个数据点上有这么多，整个 Feature Space 大概是百亿的量级，训练的数据也是 TB 量级，所以大家可以看到对机器学习平台的挑战是非常大的。

机器学习在 Twitter 有比较独特的一点是 Realtime (实时性)，Twitter 本身的产品非常的 realtime，Twitter is all about realtime, like news、

events、videos、trends，比如大家去 Twitter 上更多地是更新自己的状态，或者是看一些新闻，去了解一些最新的动态；广告商也会根据产品的特点去投放一些广告，他们往往投放的广告持续的时间都非常短，比如就是一个事件，如 NBA 总决赛，三个小时内做一个广告投放，所以要求我们机器学习的模型就必须根据实时的 traffic 的情况来不断地做调整和变化。否则，如果我们每天训练和更新一次模型，这样的速度就实在是太慢了，所以我们也是投入非常多精力做了一个规模化的在线学习的系统。你在 Twitter 上点击任何一个广告，那么在百毫秒的量级的延迟内，我们的模型就会更新。

下面我简单地过一下机器学习在 Twitter 上几个具体的产品应用。

1. Ads Ranking

它的具体问题是当你上了 Twitter，我后面有 1 千个或者 1 万个广告可以展示给你，我到底展示哪个广告给你你最可能感兴趣？因为 Twitter 采取的是 CPC (Cost Per Click) Model，只有你点击了广告，广告商才会给我们钱，如果你只是看了广告，不感兴趣没有点，广告商是不会给我们钱的，所以选取最合适广告不光是给用户更好的用户体验，同时也是 Twitter 盈利的关键；

2. Timeline Ranking (Feed Ranking)

将你的时间轴进行排序，把最好的 Tweet 能放在比较靠上的位置，这样容易被看得到；

3. Recommendation

推荐你可能最感兴趣的人；

4. Anti-Spam

比如抓僵尸粉，或者是 Abuse Detection，例如大家在 Twitter 上骂起来了，要做一些检测并且把它隐藏掉，还有 NSFW Detection 基本上是鉴别一些黄色图片之类的。

大家也可以看到，机器学习平台面临的挑战其实主要是规模化的挑战。规模化我认为主要是两方面：

- 一方面是组织架构上的规模化，我们作为机器学习平台的组如何更好地去支持这样七八个 Twitter 的核心产品，并且能够让我们的 client team（我们的用户）能够非常快地进行 prototype（产品迭代）；

- 另一方面是整个系统的规模化：一方面你的离线训练如何能更快，在线预测如何能够更快；还有一方面，当我们的用户把整个 pipeline 搭建起来之后，他们要不断优化整个 pipeline，我们有没有足够的工具支持我们这些用户做到这一点（我说的用户是 Twitter 内部的这些产品团队）。我们怎么让我们的用户非常快速地进行迭代和实验？

一、组织结构的规模化

我们相信我们的用户真正了解他们具体做的事情、他们的产品和他们的问题，所以我们采取的合作模式是：

- 我们开发各种的工具、很多的框架，去定义 feature（特征）、transform（变换）、model（模型）等等的格式，然后把工具交给我们的用户，让他们从特征提取到离线训练、如果这个模型好再推到在线生产环境当中、以至后面持续不断地优化提高，在整个过程中我们希望把每一步都做到足够的简便。同时我们还对于一些新的用户提供一些 onboarding 的支持；

- 我们的 client team 负责做特征提取的，因为只有他们了解具体的自己的问题，知道什么样的信号可以对他们的问题有更好的提升。

我们也会把整个特征的共享做到非常好，比如其他 team 有一个很好的特征，你可以非常快地加入你的模型中进行实验。同时我们的 client team 他们负责去 own 和 maintain training pipeline 和 serving runtime，例如 on call 完全不是我们的事，完全由 client team 来做。

二、系统的规模化

主要分几个方面：

1. 准备数据，既然要进行模型训练当然要把数据准备好；

2. 离线的训练，会有 workflow management；
3. Online Serving(在线服务)，比如模型训练好了，要推到市场环境中去，要可以承受这种 high QPS、low latency 这些要求，还要做 A/B testing，在 1% 的数据上先做一些实验，然后过一段时间，真正它在实际的 traffic 上更好的话我们就把它 launch 到 100%。与此同时还做很多工具，来帮助我们的用户更好地去理解他们的数据和模型，以及还有一些工具去做比如参数的扫描、特征的选择等等。

三、准备数据

首先，我们要做的是统一数据格式，定义一个数据格式很简单，但是我觉得意义非常重大，就像秦始皇统一六国以后先统一度量衡，因为只有家用一样的格式大家才能彼此互相沟通、交流和分享。

举个例子：比如某个产品团队在他们在模型中加了某一种信号或特征，结果特别好，我是做广告的，我想把数据拿过来用，如果数据的格式都不一样，我还得过去去研究你们这个组的数据格式到底是什么样子的，我们怎么转换成我们的格式，有非常非常多时间浪费在这个地方，这是我们希望解决的，Enable feature sharing across teams and make machine-learning platform iteration very easy.

我们的特征向量的格式，其实本质上是 feature identifier to feature value mapping，它支持 4 种 dense types：

- Binary；
- Continuous；
- Categorical；
- Text

2 种 sparse feature types：

- SparseBinary；

- SparseContinuous

为了去优化效率，我们 feature identifier 是用 64 位 feature id 存的，这个 feature id 是 feature name 的一个 hash。之所以这样去做，是因为如果你在训练的过程或者在你的生产环境中，你去操作很多个 string 的话，特别费 CPU，所以我们采取的方式是使用 feature id，而在别的地方存一个 feature id to feature name 的 mapping。比如我们存在数据仓库中的数据都是 feature id 的，但是每个机器学习的数据集旁边我们都会存一个 metadata，就是 feature id to feature name 的 mapping。

说到数据准备，大家可以想象一下：如果让一个数据科学家用语言描述怎么样准备他的数据，往往这个数据科学家在非常短的时间比如 1 分钟时间之内就可以描述清楚：比如我们把这个 production 中 scribe 的数据拿过来，然后和另外一个数据集做 join，然后做一些 sampling 和 transform，然后写到 persistent storage 里面去。

我们开发了一套 DataAPI，对机器学习的数据集以及数据集的操作在很高的层次上做的一些抽象，当你用我们这一套 API 去描述你想对数据进行操作过程时，这个代码就跟你描述出来你要做什么事情和我们期望达到的效果一样的简单，我们希望这使得我们大部分的 machine-learning task 在训练过程中的数据准备都能够通过 20 行或者 30 行代码就搞定。

它是基于 Scala 的 API，一个 fluent 的 interface，并且在整个过程中去保证我们的数据正确，以及刚才我们说的 feature id to feature name mapping 的 metadata 是 keep consistency。之后我们简单看一小段代码，举一个例子来让大家感受一下，这是用 Scala 写的，看这个代码的话，其实从代码上你完全就能明白我要做一件怎样的事情：

首先我是从 FeatureSource 里面读出了我机器学习里的数据集，并存在了 tweetTopic 这样一个变量里，然后我再从另外一个地方，从我的一个 input path 里读出另外一个数据集，并且把他 filter/sample by 10% randomly，然后

Example

```
val tweetTopic = TweetMediaClassificationFeatureSource().read  
DailySuffixFeatureSource(args("input"))  
.read  
.filter(0.1)  
.transform(discretizer)  
.joinWithSmaller(SharedFeatures.TWEET_ID, tweetTopic, new LeftJoin)  
.write(DailySuffixFeatureSink(args("output")))
```

1. Take my dataset whose path given by “input”
2. Sample it by 10% randomly
3. Discretize with the given discretizer
4. Left join with media label on tweet id
5. Dump the result to path given by “output”

用我给定的 discretizer 来进行 transform，然后把它和我刚才的 tweetTopic 数据集进行 join，它们的 join key 是 tweet id，并且使用的是 LeftJoin，最后我再把这个数据集写出去，这样我整个过程就准备好了。

其实读一下代码你会发现整个代码其实是非常好懂的，在写代码的过程其实就像在描述。我们的目标就是希望算法工程师在写这个代码的时候就像描述他们想做的事情一样。比如我们对数据的位置、格式等进行抽象。不管你的数据到底在哪里，比如你的数据可以在 hdfs 上，可以在 database 里，可以在很多其他地方，但是在这个 API 里，其实都抽象在 FeatureSource 里，然后用 read 就可以把它读出来，所以用户在使用的时候是不需要操心数据到底存在哪里等等这类事情。

四、Trainer

我们也提供了很多的 trainer，让我们的用户把这些 trainer 进行一定的组合作为他们的 offline training pipeline。首先是 large scale logistic regression learner，我们有两个解决方案：

1.Vowpal Wabbit

是 John Langford 开源的 C++ trainer；

2.Lolly

是 Twitter 内部开发的基于 JVM 的 online learning trainer，因为 Twitter 整个 stack（技术栈）都是基于 JVM 的，比如 Java、Scala，所以我们开发了这个 learner 会和 Twitter Stack 会结合地更好一些。

在 discretizer 方面我们都比较标准了，像 Boosting tree（GBDT、AdaBoost）、Random forest、MDL discretizer 等；

在 Deep Learning 方面，我们是基于 torch 做的，也有一些 Deep Learning 的 libraries。

五、PredictionEngine

刚刚提到 Twitter 这种实时性是非常非常重要的，所以我开发了一个在线学习的一个引擎，叫 PredictionEngine，这是专门为 Large scale online SGD learning 来做的，我们整个广告包括我们的 Feeds Ranking 都是用的这个 PredictionEngine。

在 offline training 其实整个 PredictionEngine 简单地包一层 application layer；在 online serving 的时候，PredictionEngine 包一层 online service layer，加这个 layer 去处理一些像 RPC 等等这方面的东西，它基本的架构是：

- 第一层是 Transform，用户可以去定义或者用我们提供的 transform 来对 feature vector（特征向量）进行一定的变换；

- 第二层是 Cross，Cross 的意思是我可以把我的特征分组，比如分成四五组，然后第一组和第二组所有特征进行 Cross，比如在广告上这个好处是可以把 advertiser id，即把每个广告商的 id 分到一组，把其它的 features 分到第二组，然后第一组和第二组一 Cross，其实 effectively 给每一个广告商一个 personalized feature，这是非常有效的；

- 第三层是 Logistic Regression；

这个 Architecture 一方面很方便地让我们进行在线的学习，同时在

transform layer 和 cross layer 我们也加进去了足够多的这种 nonlinearity(非线性) 元素。如果只是简单的 logistic regression, 那是线性的, 效果并没有那么好, 于是我们加了 transform layer 和 cross layer 会解决一些非线性变换的问题。

那么对 PredictionEngine 我们做了非常多的优化, 在这个地方我会详细地讲:

1. 我们希望减少序列化和反序列化的代价

第一点是 model collocation, model collocation 是什么意思呢? 就比如在广告的预测中, 我们预测的不是一个概率, 即用户有多少可能性去点击这个广告, 我们可能是预测很多个概率, 比如用户可能转发这个 tweet 的概率, 用户点击这个的 tweet 里面的链接的概率, 或者是用户点击了这个链接还购买的概率, 或者用户直接把这个广告叉掉的概率。

对于一个用户和广告的这么一个 pair, 我们会预测很多个概率, 比如你要预测 5 个概率, 本来是应该去做 5 次 RPC call 的, 但是我们会把这五个模型都放在一个 physical container 里面, 这样的话一个 call 过去, 我可以在 5 个模型中都进行计算并把 5 个 prediction 都给你返回, 这是第一个优化。

第二点是 Batch request API, 还是拿广告问题举例, 对于一个用户我要去评估可能成百上千甚至上万的广告的数量, 对于任何一个用户和这个广告的 pair, 其实用户的特征其实都是一样的, 所以有一个 Batch 的 API 的话, 我可以 amortise cost for user feature;

2. 我们希望减少 CPU 的 Cost

也是做了几方面的优化:

- 所有的 feature identifier 全都是用 id 而不是 feature name;
- Transform sharing: 刚才可以看到 PredictionEngine 里面, 第一步是做 transform, 由于我们有 model collocation 可能有五六个模型, 但其实可能有些模型他们的 transform 是一样的, 所以在这个层面上我们不要做重复的

transform，如果不同的 model 的 Transform 都是一样的话，我们就把它识别出来并且只做一次；

- 最后是 feature cross done on the fly，因为 feature cross 其实是特征从几百个变到几千个甚至几万个的过程，比如原始特征几百个，cross 之后特征数量可能大量增加。如果这时候我们把 cross 完的 feature 的再存到我们的内存中去，这个 cross 就太大了，即使只是对这个 cross 后的结果扫描一遍的代价都非常地大，所以要做成 on the fly 的 cross。

3. Training/Serving throughput

在整个在线学习过程之中，它的瓶颈在于最后 trainer 的模型 update，在 update 模型的时候就要对这个模型加锁。如果不优化的话，只能有一个线程来对整个模型进行更新。如果你的模型特别大，比如我们每一个模型都是上 GB (Gigabyte) 的，在这个情况下就会严重的影响 training throughput。

所以我们的优化会对整个模型进行 sharding，比如用多线程。比如有 10 个线程，每个线程分别负责这个模型的十分之一，算出来整个模型的 update 的时候把它切成 10 块，扔到 10 个 queue 或 buffer 里面去，让这 10 个线程来更新自己相应的模型的那一块，所以只是需要每一块的 worker 自己更新自己那块的时候对那块进行加锁就可以了；

第二个是把 training 和 prediction 分离，因为在线学习的话，我们需要在线去响应很多的请求，如果每一个模型、每一个 instance 里面都有一个 training 都在做在线学习其实是很重复的。比如你有 1 千个 instances 都在做在线学习，并且都在做实时响应请求，1 千个 instances 里面的 training 部分是冗余的，所以我们会把 training 这部分单独拿出来作为 training service，定期会把这个模型的更新去放到一个 queue 里面，然后 fanout 到所有的 prediction service instance 里面去；

第三个是弹性负载，比如我们的 client 端要 call 我们的 prediction service 的时候，我们会在 client 端加一个检测请求延迟，当我们在 client

端检测到 prediction service 不堪重负，这个时候我们会动态地减少对 prediction service 的请求，以保证我们的 prediction service 是良性和健康地运转的。

因为大家知道每天的流量会有周期变化，比如某些时段流量特别高，某一些时段比如在夜里流量相对比较低。通过弹性负载动态调整的机制，比如等到白天上午十点或者晚上八点特别忙的时候，我们可以做到对每一个用户评估少一点的广告，比如评估 2000 个广告；如果是到半夜，每一个用户可以评估多一点的广告，如 1 万个广告。这样动态地去保证 CPU 的使用率都是在固定的 level 上。这个 Level 的制定是要考虑不同数据中心之间的 failover，比如数据中心挂了，所有的这些 traffic 都要 failover 到某一个数据中心，然后还要留一点余量，所以我们一般 CPU 的 utilization 是在 40% 左右。

4. Realtime feedback

在线学习很重要一点是 feedback 一定要及时，但是有一个很不好解决的问题，如果用户他点了这个广告，这是正向的反馈你马上能知道，但是用户没有点这个广告你这事就不能马上知道，说不定用户过五分钟以后才点呢。

常用的解决方式是：我先把这个广告先存起来，然后等十五分钟，看看用户有没有点，如果用户在十五分钟内点了，我们就说这个用户点了，这是一个 positive training example，然后把它发到在线学习服务中去；如果用户没有点，这就是 negative training example。

这样的问题就是说我们会有十五分钟的延时，这一点是非常不好的，所以我们做了一个优化：每当我们展示一个广告的时候，我们马上给在线学习服务发一个 negative training example，当成一个用户没有点击的事件，然后当用户后面真正去点了这个广告的话，那时我们会对这个事情进行一定的修正，这样就保证所有的事件实时性都非常高，是没有延迟的。

5. Fault tolerance

我们的模型可能有几千个 instances，这些 instances 经常地挂。我们需要

每隔一段时间对我们的模型进行一个 snapshot，如果某一个 instance 挂了，另外一个重新启动的时候，它会去把最新最近的 model snapshot load 进来，然后再开始进行在线学习。

还有一个问题是 anomaly traffic detection，因为在线学习十分危险，因为上游数据任何的错误都会马上影响到这个模型的质量。举个例子，比如你有个 pipeline，有两个 queue，一个 queue 专门发 positive training example，另一个是发 negative training example，结果你的 positive 的 queue 给挂了，这样在线学习的模型一直只能接到 negative training example，于是模型的预测在非常短的时间内整个模型就全乱了，像 Twitter 这种公司肯定都是有非常严格的 on call 制度，但是 on call 不能解决问题，为什么呢？当 on call 被 page 的时候，5 分钟之后打开电脑去进行干预那个时候就已经晚了，所以我们是需要做 anomaly traffic detection 做到在线学习当中，如果一旦发现 traffic 这个构成发生了很严重的变化，我们会马上停止训练。当然还是要 page on call 啦，然后让 on call 来进行人工干预解决。

六、Tooling

刚才说的是在线学习，我们还给用户提供很多工具，这些工具是为了帮助我们用户很方便地区对整个模型进行研究或者做一些改进。这个工具叫 Auto Hyper-parameter Tuning，就是变量的自动选择。机器学习的模型尤其包括像深度学习模型都有很多的变量。往往大家选变量的时候都是拍脑袋，比如我觉得这个 learning-rate 应该是多少然后放进去，好一点的呢就暴力搜一下，或者有些就是 Random Search。

但是我们基于贝叶斯做了自动的 hyper-parameter 选择，我们会根据之前不同的 parameter setting 所跑出来的模型的结果去计算：我下一个 parameter 选择什么使得在期望意义下我对目标函数的提高会做到最大，而不像无头苍蝇一样到处去搜，而是充分地利用已经模型跑出来的数据和 performance 来选择下一步

尝试的参数。

其他的 tooling，比如：

- workflow management：就是整个 offline 的训练，你需要对它进行监测，需要可复现，可以互相地分享；
- Insight 和 Interpretation：我们不希望我们的用户用我们的东西是一个黑盒，所以我们也会搞一些 tool 帮助他们看数据、看模型、去分析特征的权重和贡献等等；
- Feature selection tool：进行简单地 forward/backward 的 greedy search。

七、Work in Progress

我们的机器学习也是在不断的探索之中，这是我们努力的一些方向：

1. 最主要的方向是我们要平衡规模化和灵活性的问题。因为规模化和灵活性往往是非常矛盾的，如果你用一些 R、Matlab、Scikit-Learn 等等一些工具，它们很多东西做得不错，灵活性是可以的，但是在这些工具上要搞规模化，这个事情是非常困难的。

反过来如果要规模化，你的系统要做的非常非常专，要针对某些情况做出很多优化，但是这种情况下就会影响算法的发挥。举个例子，我们的 PredictionEngine 分三层，第一层 Transform、第二层 Cross、第三层是 Logistic Regression，如果说我想试一点别的框架和步骤，和这个假设如果不是那么一样的话，可能你就没有办法用我们的这个工具。

所以，规模化和灵活性一直是一个非常冲突和难以平衡的问题，也是大家在不断在这方面做更多的努力，我们也期望用一些 torch-based 的 large scale 机器学习，因为 torch 在灵活性方面是足够的，如果我们能够解决规模化的问题就会非常好。

2. 我们也会尝试把深度学习的一些东西在广告或者是 feeds 流上做一些实验，虽然在业界现在成功的并不多，只有 Google 他们声称在这个方面做得还可以；

3. 为我们的用户提供更好的工具，比如 visualization 和 interactive exploration。

Q&A

主持人：好，谢谢晓江，今天你讲的这一场非常爆满。我想替大家问你几个问题，第一个问题是，你们做数据和推荐这一块，能直观的给几个数据来度量一下对你们 Twitter 业务的价值吗？

郭晓江：刚才我可能提到了，Twitter 90% 的营收来自广告，所有广告都是机器学习支撑的。我四年前进 Twitter 的时候，广告组的规模的确刚刚起步，当时那一代机器学习的架构也非常简单，模型也非常的简陋。在我们上了大规模的在线学习的东西之后，把整个 Twitter 的营收提高了大概 30% 左右，这对于 Twitter 是几十亿美金的 business，30% 是一个非常非常可观的数字，一般模型能提高 1%、2% 就已经非常不错了。

在一些基础架构的革新使得我们可以在大规模的数据上面进行训练，比如模型的复杂度和 feature 的规模都扩大了好几个数量级，的确会对我们整个 business 和整个模型的质量都有显著地提高。

主持人：30% 的提升，很酷的数字，很困难的动作，所以在场的 CTO、架构师加油整，我们数据的价值很大。第二个问题，你提到在架构上踩过很多的坑，这四年 来整个过程中，你觉得最难的地方是什么？

郭晓江：因为我们是一个和业务结合蛮紧密的组织，我觉得其实最难的事情是要统一所有组的机器学习的架构。因为在最开始，可能每个组都有自己的一套机器学习的东西，但是如果大家都自己一套东西，互相的 share 就非常的困难，所以我们花了非常多努力，比如就非常简单的数据特征格式的定义，要推到所有的组都相当困难。

我觉得很多时候这个挑战更多还是来自于非技术的那一块，也就是说服大家用我们的这一套系统。也许因为我做技术，我觉得技术的东西还好，但是非技术

的东西要推的话确实会稍微难一些。我们现在大家都用一套机器学习平台，之后我们把学习平台进行一些更新换代，其实基本上代码一般的生命周期也就是两年左右，过两年我们又有一套新的东西来更好地替代，这时候大家都用这一套，我们替代的时候也会方便很多，所以我觉得这是一个很大的挑战吧。

主持人：好，最后一个问题是，你们 Twitter 的这件事情做得很牛，架构也很牛，你们有没有更多的文档或者知识能在网上跟大家分享吗？

郭晓江：我们现在并没有太多对外开放的文档，我们也考虑过有一些东西能不能 open source。因为机器学习和业务结合实在太紧密了，它和整个公司的技术战略也非常紧密，我要开源这个东西，可能所有依赖的东西都得开源，而且机器学习这个东西更新实在太快了，所以我们花了好多时间，把这套开源了，实际上更好的东西已经出来了，所以暂时我们这方面还没有考虑。

讲师介绍

郭晓江，四年前加入 Twitter，先后供职于广告组和机器学习平台组。在广告组设计和构建了 ads ranking 后端平台，之后从无到有领导团队搭建了 Twitter 的机器学习平台，应用在广告推荐、Timeline Ranking、反欺诈等多个产品中，是每年几十亿美元的营收与内容推荐背后的核心。本科毕业于清华电子工程系，硕士毕业于斯坦福电子工程系。

【延伸阅读】

携程实践：当你愉快入住酒店，有想过机器学习在帮忙么



1号店 11.11：机器排序学习在电商搜索中的实战

作者 张志浩

背景

1号店的搜索 Ranking Model 一直在朝着精细化方向深化，我们希望在提升用户满意度的同时，也能提升网站的流量转化率。在实践机器排序学习之前，1号店网站的搜索 Ranking Model 已经经历了 4 个阶段：通用排序模型 (Universal Ranking Model)、基于区域的排序模型 (Region-based Ranking Model)、基于品类的排序模型 (Category-based Ranking Model) 和基于用户的排序模型 (User-based Ranking Model)。

在这个过程中，需要对搜索、浏览、点击、购买，评论等几十个行为特征进行细粒度的分解和重组，这就需要人工去分析用户的搜索行为和流量效率之间的关系，并通过人工调整排序模型来优化效果。在细分了区域和品类之后，为了更有效地提升排序模型的优化效率，我们开始考虑使用机器排序学习算法。

机器排序学习的一般分为两个流程，其中 “training data → learning

algorithm → ranking model”是一个离线训练过程，包含数据清洗、特征抽取、模型训练和模型优化等环节。而“user query → top-k retrieval → ranking model → results page”则是在线应用过程，表示利用离线训练得到的模型进行预估。

机器排序学习有如下优点：

- 人工规则排序是通过构造排序函数，并通过不断的实验确定最佳的参数组合，以此形成排序规则对搜索结果进行排序。但是在数据量较大、排序特征较多的情况下，依靠人工很难充分发现数据中隐藏的信息，而机器排序则可以较好地解决这个问题；
- 机器学习可以基于持续的数据反馈进行自我学习和迭代，不断地挖掘业务价值，对目标问题进行持续优化。

设计原则

从一开始，我们就没有考虑将机器排序学习作为人工规则排序的替代者，而是致力于将两者作为互为补充的排序模型，不同品类采用不同的排序策略，这也体现在了我们的搜索排序架构上。

图1表示目前1号店现在生产环境上机器排序学习的框架，包含两个部分：离线训练和在线应用。离线部分主要是根据历史数据以及训练目标，产生一个可

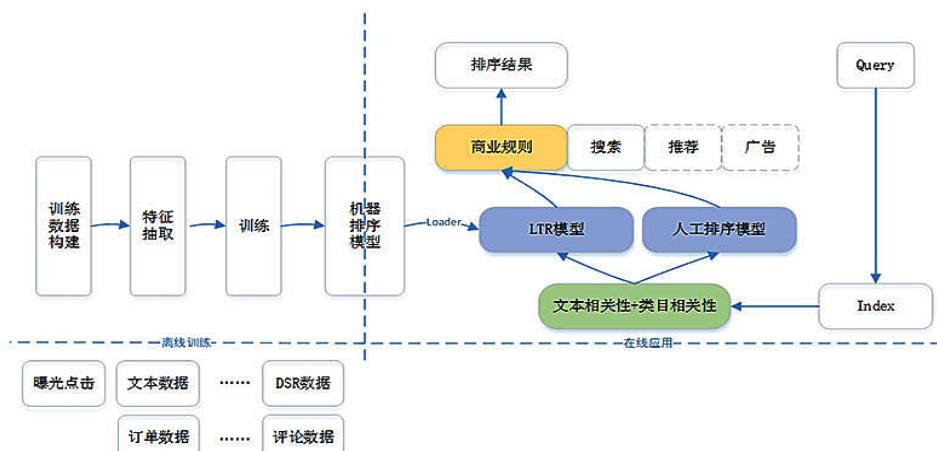


图1 机器排序学习框架

用于在线预测的排序模型；而在线部分则是利用离线产生的排序模型，根据在线用户的 Context 完成实际的排序。

在整个机器排序框架中，整个在线排序模块分为 3 层。

第一层称为一排（也称为粗排），主要是根据用户 Query 从索引库中召回所有相关商品，然后依据文本相关性和类目相关性得分，完成第 1 轮商品筛选，形成上层排序模型的候选商品集。

第二层称为二排（也称为精排），主要是以一排的候选商品集为基础，应用 LTR 模型或者人工排序模型，完成基于排序特征的重新排序。

第三层称为三排，主要是根据各种商业需求对二排的排序结果进行调整，如类目打散、商品推广等。

在整个机器排序框架中，离线部分需要模型评测环境，而在线部分更需要数据收集模块和 A/B Test 模块。这两点并没有在图 1 的框架中列出，但是在后续的介绍中会给出相应的说明。

机器排序学习离线训练过程

我们选择开源的 RankLib 作为机器排序学习的工具包。在离线训练过程中，我们测试了 RankNet，LambdaMART 和 Random Forests，根据评测结果，我们选择了 LambdaMART（具体采用哪种模型，需要根据实际业务场景和训练结果而定）。该算法是一种有监督学习（Supervised Learning）的迭代决策回归树排序算法，目前已经被广泛应用到数据挖掘的诸多领域。

LambdaMART 模型可以分成 Lambda 和 MART 两部分，底层模型训练用的是 MART (Multiple Additive Regression Tree)，也叫 GBDT(Gradient Boosting Decision Tree)，它的核心是每一棵树学习的是之前所有树结论和的残差，这个残差 + 当前的预测值就能得到真实值。而 Lambda 是 MART 求解过程使用的梯度，其物理含义是一个待排序的文档下一次迭代应该排序的方向和强度。具体 LambdaMART 的算法和工作原理可以参考。

下面我们以 LambdaMART 为基础算法来介绍机器排序学习的整个过程。

训练目标数据

离线训练目标数据的获取有 2 种方法，人工标注和点击日志。两者都是为了构建出 $\langle q, p, r\text{-score} \rangle$ 的数值 pair 对，作为机器学习的训练数据集。这里 q 表示用户的查询 query， p 表示通过 q 召回的商品 product， $r\text{-score}$ 表示商品 p 在查询 q 条件下的相关性得分。

其中人工标注一般步骤为，根据给定的 query 商品对，判断商品和 query 是否相关，以及相关强度。该强度值可以用数值序列来表示，常见的是用 5 档评分，如 1- 差，2- 一般，3- 好，4- 优秀，5- 完美。人工标注一方面是标注者的主观判断，会受标注者背景、爱好等因素的影响，另一方面，实际查询的 query 和相关商品数量比较多，所以全部靠人工标注工作量大，一般很少采用。

因此，在我们的实践探索中，寻找获取方便且具有代表性的相关性的度量指标则成为重中之重。经过初期的探索，我们确定以 CTR 为基础，实现了低成本的 query-product 相关性标注（虽然不完美，但在实际工程中切实可行）。具体步骤如下：从用户真实的搜索和点击日志中，挖掘出同一个 query 下，商品的排序位置以及在这个位置上的点击数，如 query_1 有 3 个排好序的商品 a, b 和 c，但是 b 得到了更多的点击，那么 b 的相关性可能好于 a。点击数据隐式地反映了相同 query 下搜索结果相关性的好坏。

在搭建相关性数据的过程中，需要避免“展示位置偏见”position bias 现象，即在搜索结果中，排序靠前的结果被点击的概率会大于排序靠后的结果；在我们的训练模型中，如图 2 所示，第 6 位开始的商品点击数相比前 5 位有明显的下降，所以我们会直接去除搜索结果前 5 个商品的点击数。

同时在实际场景中，搜索日志也通常含有噪音，只有足够多的点击次数才能体现商品相关性的大小。因此为了提升

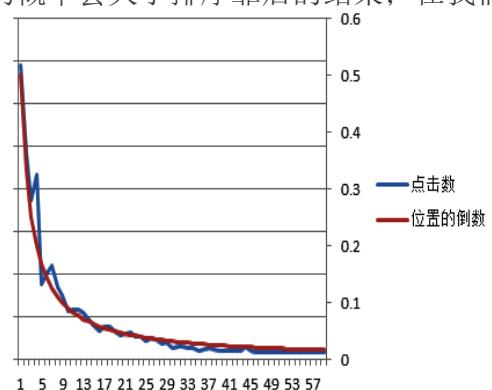


图 2 位置偏见的示意图

训练效率和训练效果，我们也针对 Click 数少于某个阈值的情况（即少于某个阈值的点击，我们就直接忽略）进行了测试，分别为 2, 3, 4, 5, 6, 7, 8。经过离线分析，在阈值为 4 的情况下，NDCG 分别有 20%-30% 的提升（这是 NDCG@60 on validation data 和当前线上的 NDCG 相比），效果提升比较明显（见表 1）。

特征抽取

LTR 使用的排序特征（也称为 Feature），和人工规则排序使用的特征基本相同。按照数据特性的差异，我们将其分为 2 大类，如表 1 所示，当然不同类型的数据价值和反映用户意图的强弱也不尽相同（见表 2）。

表格 1 Click 数对训练目标的影响

	2	3	4	5	6	7	8
AverageOverlapScore	0.2779563	0.2622574	0.2927366	0.2605555	0.3292106	0.2570833	0.2815816
RBOScore	0.3660776	0.3278322	0.3587360	0.3362893	0.3906409	0.3456386	0.3614653
LikeNDCG1	3918.920	4529.955	4345.691	4550.222	3970.641	3555.782	3418.661
LikeNDCG2	203.5713	250.1454	235.3356	244.135	209.4407	217.9441	207.5903
NDCG@60 on training data	0.4734	0.4479	0.502	0.4846	0.4983	0.621	0.643
NDCG@60 on validation data	0.4896	0.4727	0.536	0.4743	0.4928	0.5015	0.4864

表格 2 特征抽取部分样例

Feature类别	Feature项	
查询相关性特征	查询相关性	text relevance, category relevance
商品特征	商品静态特征	title, sub-title, attribute (brand, category, size, color, flavor)等
	商品动态特征	price, promotion等
	订单特征	sales volume by day/week/month, GMV by day/week/month等
	行为特征	CTR (list->detail, detail->shopping)
	售后特征	number of review, positive rate of review, complain rate, return rate等
	商家服务特征	DSR, online duration, online instant response等

特征数据分几类或者怎么分类都不是重要问题，这里主要是说明要尽可能抽取多的特征数据，供后续训练使用。同时对于机器排序学习而言，这里的特征都需要跟踪到 Query 维度，所以在数据采集的时候，需要预先处理好，这样才能体现 Query 对应的排序影响。如果能将订单特征也区分到 Query 维度会更好，但是我们现在还没有做到这一点，以后还会继续实验。

除了商品静态特征以外，商品动态特征、订单特征、行为特征、售后特征、商家服务特征都需要细分到品类和区域，幸运的是，在人工规则的排序模型阶段，就已经为我们准备好了 profile-based 的特征数据。为了能在人工规则排序和机器排序学习之间共享这些特征数据，我们将这些特征数据存储在基于 HBase 的 Item Feature Repository 数据库中。目前这些特征数据还是以天为单位进行批处理，而非实时完成。

在得到相关度数据和特征数据后，就可以根据 LambdaMART 训练数据的格式（如下所示），构建完整的训练数据集。每一行代表一个训练数据，项与项之间用空格分隔，其中 <target> 是相关性得分，<qid> 是每个 query 的序号，<feature> 是特征项的编号，<value> 是对应特征项归一化后的数值，<info> 是注释项，不影响训练结果。

```
<target> qid:<qid> <feature>:<value> <feature>:<value> ...
<feature>:<value> # <info>
```

图 3 表示项目中使用的实际训练数据（这里选取了其中 10 个特征作为示例，# 后面可以增加 Query 和商品名称，方便分析时的查看）。

离线训练

LambdaMART 学习过程的主要步骤可以参考，数学推导不是本文的重点：

- 先遍历所有的训练数据，计算每个 pair 互换位置导致的指标变化 deltaNDCG 以及 lambda；
- 创建回归树拟合第一步生成的 lambda，生成一颗叶子节点数为 L 的回归树；

```

1 0 qid:983 1:0.582 2:0.353 3:0.553 4:0.582 5:0.451 6:1.762 7:0.891 8:0.998 9:0.778 10:0.543 #维达蓝色经典系列3层140g卷筒卫生纸
2 0 qid:983 1:0.583 2:0.422 3:0.553 4:0.583 5:0.452 6:1.758 7:0.902 8:1.012 9:0.765 10:0.545 #维达3层280节卷筒卫生纸*12卷
3 0 qid:983 1:0.582 2:0.413 3:0.552 4:0.583 5:0.439 6:1.689 7:0.901 8:0.986 9:0.782 10:0.476 #维达至有份量系列3层200g
4 2 qid:379 1:0.713 2:0.318 3:0.716 4:0.672 5:1.103 6:0.997 7:0.913 8:0.778 9:0.614 10:0.123 #vinda维达 无芯卫生卷纸 无香 85g*12粒
5 0 qid:379 1:0.121 2:0.322 3:0.709 4:0.657 5:1.112 6:0.879 7:0.924 8:0.775 9:0.324 10:0.132 #心相印mind act upon mind心相印
6 2 qid:379 1:0.032 2:0.345 3:0.804 4:0.649 5:1.004 6:0.645 7:0.935 8:0.679 9:0.527 10:0.127 #清风 卷纸 马蹄莲系列3层100g
7 0 qid:677 1:0.981 2:0.262 3:0.777 4:0.465 5:0.000 6:0.000 7:0.758 8:0.567 9:0.435 10:0.110 #3m/思高 超洁净百洁布3片装
8 1 qid:677 1:0.980 2:0.259 3:0.769 4:0.465 5:0.000 6:0.000 7:0.762 8:0.587 9:0.435 10:0.157 #3m/思高 防刮擦百洁布3片装
9 2 qid:677 1:0.981 2:0.253 3:0.782 4:0.465 5:0.000 6:0.000 7:0.773 8:0.587 9:0.435 10:0.130 #3m/思高 防刮擦海绵百洁布2片装
10 0 qid:672 1:1.311 2:0.724 3:0.000 4:0.443 5:1.000 6:0.217 7:0.773 8:0.593 9:0.378 10:0.654 #妮飘 手帕纸 60包3层10片/包 纸面印花
11 0 qid:672 1:1.311 2:0.723 3:0.000 4:0.443 5:1.000 6:0.218 7:0.765 8:0.582 9:0.412 10:0.655 #彩竹 波斯猫 名典迷你手帕纸 (加香)
12 0 qid:672 1:1.312 2:0.645 3:0.000 4:0.553 5:1.000 6:0.223 7:0.767 8:0.576 9:0.412 10:0.652 #清风 清香型纸手帕10包 原生木浆纸巾
13 1 qid:655 1:1.608 2:0.100 3:0.627 4:1.092 5:0.667 6:0.225 7:0.671 8:1.001 9:0.523 10:0.716 #3m/思高 易洁8层耐用抹布
14 0 qid:655 1:1.607 2:0.112 3:0.628 4:1.094 5:0.652 6:0.226 7:0.669 8:0.988 9:0.489 10:0.011 #3m/思高 耐用型天然橡胶手套
15 0 qid:655 1:1.608 2:0.123 3:0.618 4:1.094 5:0.624 6:0.261 7:0.701 8:0.789 9:0.495 10:0.087 #3m/思高 随手粘衣物用56张+15张替换

```

图 3 LambdaMART 训练样本

- 对这棵树的每个叶子节点通过预测的regression lambda计算每个叶子节点的输出值；
- 更新模型，将当前学习到的回归树加到已有的模型中，用学习率 shrinkage系数做regularization。

RankLib 提供了简单的命令行，只需根据实际的需要配置好参数，就可以实现训练模型、保存模型的功能。下面是我们在工程中使用的参数配置示例：

```

java -jar ~/bin/RankLib.jar -train ~/train_all.csv -gmax 4 -tvs
0.8 -norm zscore -ranker 6 -metric2t NDCG@60 -tree 1000 -leaf
10 -shrinkage 0.1 -save ~/models/learned_lambdamart_model.mod
参数说明

```

Algorithm: LambdaMART

```

set number of trees  $N$ , number of training samples  $m$ , number of leaves per tree  $L$ ,
learning rate  $\eta$ 
for  $i = 0$  to  $m$  do
     $F_0(x_i) = \text{BaseModel}(x_i)$  //If BaseModel is empty, set  $F_0(x_i) = 0$ 
end for
for  $k = 1$  to  $N$  do
    for  $i = 0$  to  $m$  do
         $y_i = \lambda_i$ 
         $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$ 
    end for
     $\{R_{lk}\}_{l=1}^L$  // Create  $L$  leaf tree on  $\{x_i, y_i\}_{i=1}^m$ 
     $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$  // Assign leaf values based on Newton step.
     $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$  // Take step with learning rate  $\eta$ .
end for

```

图 4 LambdaMART 训练过程

```

<ensemble>
  <tree id="1" weight="0.1">
    <split>
      <feature> 10 </feature>
      <threshold> 0.05859375 </threshold>
      <split pos="left">
        <feature> 11 </feature>
        <threshold> 0.015625 </threshold>
        <split pos="left">
          <feature> 15 </feature>
          <threshold> 0.4140625 </threshold>
          <split pos="left">
            <output> -0.8295480608940125 </output>
          </split>
          <split pos="right">
            <output> 0.6033934950828552 </output>
          </split>
        </split>
        <split pos="right">
          <feature> 11 </feature>
          <threshold> 0.5390625 </threshold>
          <split pos="left">
            <output> 0.032923102378845215 </output>
          </split>
          <split pos="right">
            <output> 1.0059690475463867 </output>
          </split>
        </split>
      </split>
    </tree>
  </ensemble>

```

图 5 LambdaMART 训练产生的模型

- `-ranker`是必须的，表示指定的机器学习算法，而6就是LambdaMART；
- `-gmax`是可选的，指定训练目标相关性的最大等级，默认是4，代表5档评分，即 $\{0, 1, 2, 3, 4\}$ ；
- `-tvs`是可选的，设置样本中用于训练的数据比例，即train数据:validation的比是0.8:0.2；
- `-norm`是可选的，指定特征归一化的方法，默认没有归一化，`zscore`代表采用均方误差来归一化；
- `-metric2t`是可选的，训练数据的评测方法，默认是ERR@10；
- `-tree`是可选的，指定lambdamart使用的树的数目，默认是1000；
- `-leaf`是可选的，指定lambdamart每棵树的叶节点数，默认是10；
- `-shrinkage`是可选的，指定lambdamart的学习率，默认是0.1；
- `-save`是可选的，用于保存模型。

上述命令执行结束后会在`-save`指定的目录下产生一个如图5的模型文件，该模型文件也就是可以用于生产环境的机器排序模型。

离线评测

在排序模型训练完成之后，我们需要准备与训练样本相同数据格式的离线测

试样本，这个测试样本尽量选取和训练样本不同的数据集。

```
java -jar ~/bin/RankLib.jar -load ~/models/learned_lambdamart_
model.mod -test ~/test_samples.csv -metric2T NDCG@60 -score ~/
rerank_scores.txt
```

这里 -load 对应的参数就是在离线训练中得到的排序模型，-test 对应的参数就是测试样本集，-metric2T 使用和训练过程相同的评价方式，-score 对应的参数就是保存测试样本集中每个 query 下商品的得分。

【实验样本】为了离线测试 LTR 的模型效果，选择了两个流量差异较大的类目，暂且称为类目 A 和类目 B，其中类目 A 的流量大约是类目 B 的 40 倍。同时选择不同平台、不同时间段内的数据作为训练样本。

【评测指标】除了使用评价排序效果的 NDCG 以外，出于商业因素的考虑，我们还选择了 4 个评价排序位置变动情况的指标，其中 AverageOverlapScore 和 RBOScore 指标越大表示正常排序和 LTR 排序越接近，LikeNDCG1 和 LikeNDCG2 指标越小表示正常排序和 LTR 排序越接近。

可以看出，同样用一周的数据，PC+IOS+Android 的评测效果在多个指标上要好于只用 PC 的效果；而同样在 PC 端，两周数据的评测效果也好于一周；但是这个结论并不是普遍性，对于不同类目，需要具体问题具体分析，进而确定表现较好的模型。

另外，为了验证 LambdaMART 的不同训练参数对预测效率和预测效果的差异，我们也进行了其它 4 组实验，分别是如下：

1. 默认参数设置；
2. 在1的基础上调整tvs参数；
3. 在2的基础上调整leaf参数；
4. 在3的基础上调整tree和shrinkage参数。

4 个模型的排序结果如图 6 所示，其中横坐标 P01 表示线上排序 1–8，P02 表示线上排序 9–16，纵坐标表示这 8 个位置与 LTR 排序下商品位置变动数量。

表格 3 类目 A 的离线评测效果

类目	A				
训练数据平台	PC	PC + IOS + ANDROID	PC	PC	PC
训练数据时间段	一周	一周	两周	一周	一周
NDCG@60 on training data	0.5617	0.5365	0.5313	0.5253	0.5778
NDCG@60 on validation data	0.5391	0.5411	0.5605	0.5138	0.5372
AverageOverlapScore	0.3535	0.3751	0.3710	0.4254	0.3442
RBOScore	0.4236	0.42823	0.4306	0.4591	0.4005
LikeNDCG1	3001.8343	2997.8116	2957.3804	3591.0862	3108.4491
LikeNDCG2	148.5160	147.1333	141.9098	176.5259	156.8397

表格 4 类目 B 的离线评测效果

类目	B				
训练数据平台	PC	PC + IOS + ANDROID	PC	PC	PC + IOS + ANDROID
训练数据时间段	一周	一周	两周	一周	一周
NDCG@60 on training data	0.6834	0.4882	0.6148	0.7887	0.4955
NDCG@60 on validation data	0.5193	0.5356	0.5549	0.5522	0.5042
AverageOverlapScore	0.2273	0.3438	0.3100	0.2207	0.1965
RBOScore	0.2597	0.3636	0.3495	0.2573	0.2344
LikeNDCG1	3166.5950	2765.4566	2022.0802	3202.8392	3142.5835
LikeNDCG2	208.9072	156.8297	110.9762	201.1636	188.5675

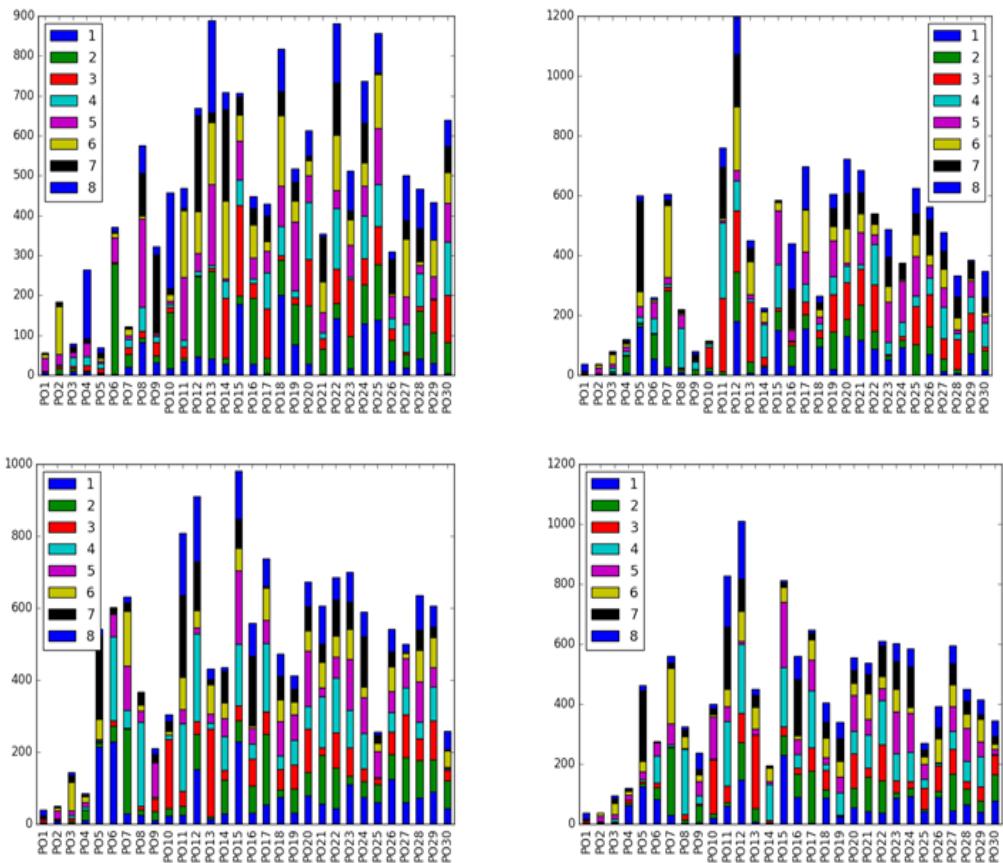


图 6 4 个模型的排序结果

不同模型的指标对比如下，模型 2 的 4 个指标均为最佳。

综合 LambdaMART 在 Training Data 和 Validation Data 的迭代次数和训练时间来看，随着参数的增加，训练时间和迭代次数都有增加；但是当去除噪音 Click 的数据后，训练时间和迭代次数有明显的降低。

在线应用

【模型加载】 离线测评时，可以用命令行的方式加载模型，读取数据文件，进而测试模型性能。显然这种方式并不适合在线应用，因此我们需要提取 RankLib 加载模型和在线打分的代码。为了有效进行系统地管理，我们将模型文件统一存放在 MySQL 表中，并利用字段区分类目和平台等信息。

【在线测试】 为了排除用户固有行为特性的影响，从而更加准确地比较 LTR 排序和人工排序的各项指标，在进行 A/B Test 之前，我们首先进行了一段时间

的 AA 测试，从而选出各项指标较为接近的桶，再比较这些桶在 LTR 排序和人工排序之间差异。

将训练后的 LTR 模型按照图 2 的框架应用到生产环境中，在整个测试期间，我们观察了不同时间周期、不同平台数据产生的多个模型。总体来讲，LTR 模型对“搜索导航页到商详页的 CTR”在 2016 年 11.11 活动期间有约 8.7% 的提升，并且“订单转化率”在 11.11 活动期间有约 4.5% 的提升。（其中 LTR_1 和 LTR_2 是我们设置的不同流量分桶，用于观测不同分桶对模型的影响）

总结

总的来说，人工规则与机器排序是紧密结合的。如果排序的量化目标不太明确，则人工规则就更适合。电商搜索不仅要考虑查询相关性，还要考虑销售额和订单转化率，因此电商搜索往往有较多的业务规则。如 eBay 和 Google 在搜索排序方面就结合了人工业务规则，而广告排序一般更依赖于机器排序，因为其优化目标比较明确。所以我们也会在当前成果的基础上，将机器排序学习进一步应用到推荐和广告系统上。

【延伸阅读】

深度学习在搜狗无线搜索广告中的应用



基于机器学习方法对销售预测的研究

作者 唐新春

首先我先自我介绍一下，在加入百分点之前，曾在生物信息公司中负责生物大数据的分析和数据挖掘；在百分点负责在金融领域的征信模块开发、销售预测领域预测模型研究，以及零售类用户画像的研发等工作。

销售预测的基本情况

在开始今天的分享之前，我首先跟大家简单的聊一下，刚刚过去的双十一，大家可能更关心的是双十一的折扣，什么商品打了什么折扣。但是对于天猫而言，他们可能更关心的是双十一当天的销售额是多少，因为知道销售额，他就能提前做一个准备，做到未雨绸缪。

我们这边有三组数据，第一组是在双十一的前十天，网上有一个专家预测，双十一是 1180 亿，7 天以后马云放出豪言，说今年的双十一可能要突破 1500 亿，去年是 920 多亿。在双十一的前一天，网上有一个专家预测了今年的双十一是 1200 亿，最后双十一是 1207 亿。

这里有两个问题，第一个问题，预测是怎么做出来的？第二个问题是：对于

2016年：1180亿元？

获地榜 2016-10-21 10:46:30 支付宝 淘宝 阅读(1386) 评论(0)

2016年天猫双十一交易额净增加了268亿，相当于2013年天猫双十一的总量，2015年双十一销售额已经是2009年第一次双十一的1824.34倍，是2013年的2.6倍。今年天猫双十一销售额将突破1000亿，或达1180亿元。今年看张勇能否创造奇迹，我们拭目以待！

马云：2016“双十一”破1500亿，天猫淘宝双11玩法

花萌 2016-10-28 13:35:18 阅读(1574) 评论(0)

声明：本文由入驻搜狐公众平台的作者撰写，除搜狐官方账号外，观点仅代表作者本人，不代表搜狐立场。

举报

搜狐科技 > 互联网 > 天猫

专家估计，2016年天猫双11销售量有望超1200亿

马继华 2016-11-10 13:34:55 天猫 双11 阅读(10432) 评论(1)

同一件事情它是预测出不同的结果，什么结果是好与坏？第一个问题是怎么预测的问题，第二个问题是预测的效果好与坏的问题。这就引出了我的主题，《机器学习对销售预测的研究》。

机器学习是常用的日常分析的方法，另一方面机器学习在海量数据中挖掘其中的规律效果非常好。

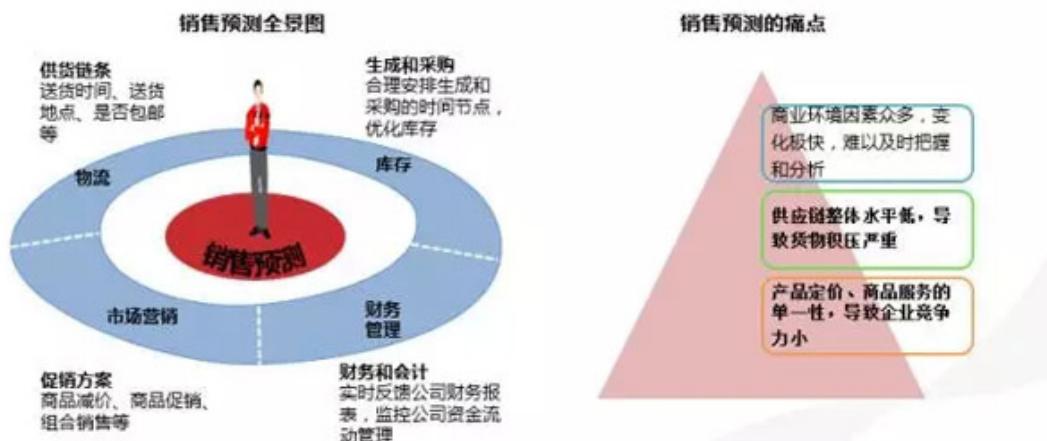
首先，说说，销售预测的现状和痛点。销售只是一个商业问题，要做的是满足用户的需求，同时对后续的运营做主导。而且它的目的并不仅是增加企业的销量，而是能够让企业能够获利，利润增加，所以它是一个商业问题。

对于这样一个商业问题，它在商业环境里面地位是显而易见的，这里有物流、库存、促销、财务等等四个方面的作用。对于销售预测的痛点，有三方面

- 商业环境变化莫测，要做到预测非常准确可能会比较困难；
- 销售预测并不是一个纯粹的销售预测，它与企业的整体的反应链相关的；
- 有企业产品比较单一，或者是服务比较单一，想要通过这个销售预测来做这个指导，来指导研发新的产品，或者是通过价格进行动态定价。

销售预测是完善客户需求管理、指导运营、以提高企业利润为最终目的商业问题。

而**预测的精确性**是销售预测的核心痛点。



在我看来，我觉得核心的痛点就是预测的精确性的问题，也就是第一个痛点。如果第一个痛点很好的解决掉，后面痛点就很好解决。预测的精确性为什么是核心的痛点呢？这里面就要从预测开始讲起，我们对一个事件进行预测是这样一个过程，就是基于历史的情况进行推演出一个规律，通过这个规律来进行推演到未来。它的特点就是我的短期的预测的精度要远远高于长期的预测的精度，即，未来一周的预测要比未来三个月的精度要高。

同时对于预测的话，我们会有几个方面的假设，第一方面是变化模式，其实就是数据里面的规律，它是我们对于这个待预测事物的了解因素。比如说你要去预测明天彩票的号码，我们已经通过对彩票的分析我们知道没有因素是可以影响它的，所以我们是预测不了明天的彩票。这是第一个变化模式。

第二方面，我们要基于数据来做，你的数据量太少，我们这边也做不了，就是一定要达到数据量的级别我们才能做数据挖掘，或者机器学习。

第三方面，我们做机器学习，我们做预测，都会有一个理论框架，在整个理论框架下我们才能做，才能落地。所以这三方面就构成了我们今天要讲的销售预测的体系框架。销售预测体系框架到底是什么呢？

其框架就是销售预测的基本步骤，即确定预测目标、收集和理解数据、建立

模型和评价指标。

销售预测的基本步骤



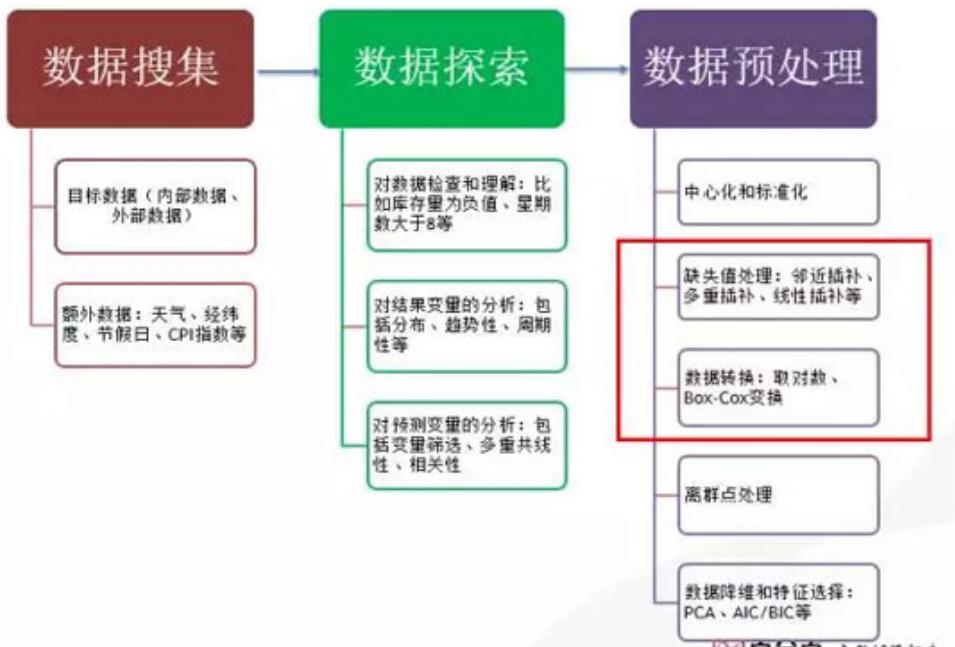
第一步，确定预测目标：首先要研究一下它的预测的对象，比如我要预测衣服，衣服是冬天的衣服，肯定是有季节性的，预测对象不一样，场景不一样，我们预测的方法也会不一样。

同时，做销售预测一般会有一个预期，我们希望这预测做未来 4 周的预测，还是未来 5 天的预测，这是一个短期的预测和长期的预测。如果是希望太高的话我们可能达不到，希望太小，我们花费了这么多的时间和精力。通常而言，你要预测未来一个月的销量，至少需要两年的数据。

此外，因为需要跟业务进行对接，所以业务目标也很重要。一方面是模型的精确性，即，我们可以给出一个精确度，另一方面是模型的可解释性，即，我们的结果更多是要根据后面的业务部门来进行交接，进行沟通，这时候你就不能解释，好与不好为什么，这要带有可解释性。

综上所需，确定预测目标要分两方面进行权衡，即预测精确性和模型可解释性。

第二步，收集数据或理解数据阶段。而这一步通常包括三个小的子步骤，即：



收集数据、数据探索和数据预处理。要尽可能多的获得数据，同时，还要理解数据背后的故事与含义。这里有一个小故事：我当时在做项目的时候，我当时电商的销售预测的时候，我看了一下库存，一般来说你卖出一件商品库存就会减，但是我在数据库里看库存为什么是负值呢？我就和业务部门进行沟通，原来他们把库存的默认值就是负值。这就是对于我们在做预测的时候，对数据背后的含义一定要理解清楚。这是一个数据探索。

而数据探索，其目的是为了更好地发现数据的规律，对应用建模提供一个指导方案。

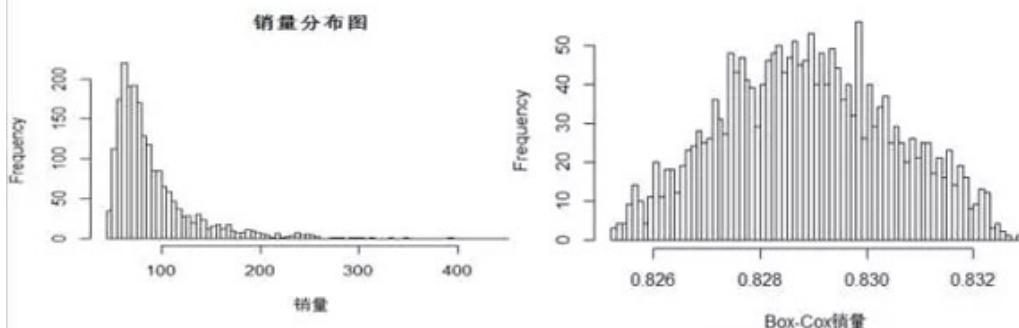
数据预处理是因为同时收集的原始数据可能非常脏、质量非常差，需要做数据清洗。数据预处理的方法很多，这里主要介绍两个数据预处理方法，缺失值处理和数据转换。缺失值处理是指，发现缺失值的时候，需要理解缺失背后的原因是什么，是数据库的技术问题还是真正业务的原因导致它缺失？如果是后者业务原因导致缺失，我们再来考虑怎么处理缺失值，处理缺失值的方法大体有两类：直接删除法和插补法。直接删除法是将缺失率较高的特征或样本数据进行删除，而插补法是通过已有的数据对缺失值进行填补。而数据变换是指数据的分布与我们假设的数学模型会不一样，这样的数据将会对预测精度会有一定的影响。因此，

导入模型之前我们就要把数据进行变换，一般有两种方法：直接对数据进行变换、Box-Cox 变换。

一个需要进行数据变换的原因是去除分布的偏度。一个无偏分布是大致对称的分布，这意味着随机变量落入分布均值两侧的概率大体一致。

数据变换一般有两种方法：

- (1) 对数据做变换，如取对数、平方根或倒数
- (2) Box-Cox 变换



第三步是应用建模。销售预测的方法有很多种类，本报告主要是介绍以下三类：第一类就叫主观预测法，即，专家法；第二类是时间序列法，即指数平滑法和自回归移动模型；第三类是机器学习中的回归算法。

销售预测的基本方法

专家法是指通过人的判断，人的经验对于未来的销售做一个预测，它的优点是比较快速，比较简单，就是很快就能够给一个结果。缺点就是我们预测的结果跟你预测的结果都不一样，带有差异性。这个专家法其实很多公司在前期他们都

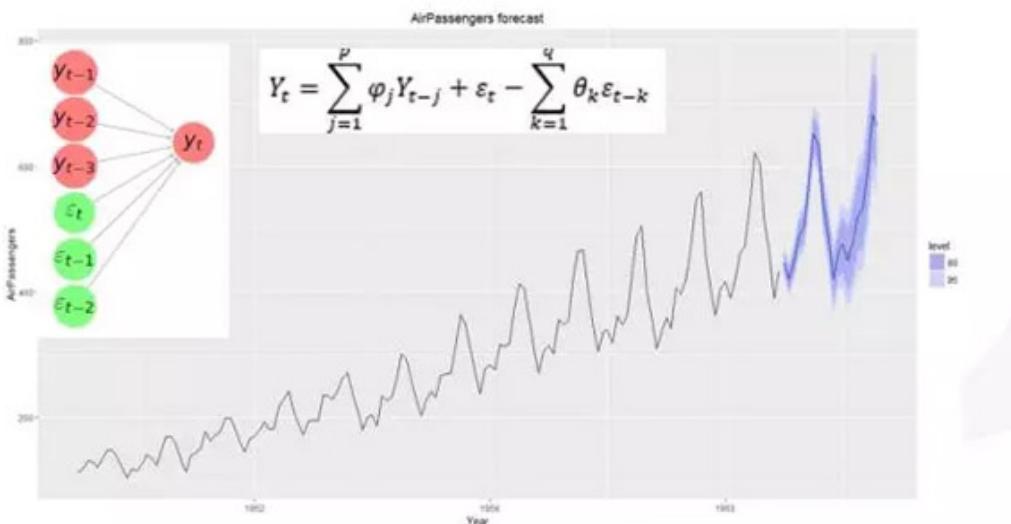


是用专家法来进行预测的。还有我们的客户他们在刚成立的电商，他也是用专家法来做的。

时间序列的方法中最简单是指数平滑法。它的特点是“重近轻远”，即通过不同的权重来控制预测的精度。优点是简单、适合于趋势预测；缺点是精确率不高。

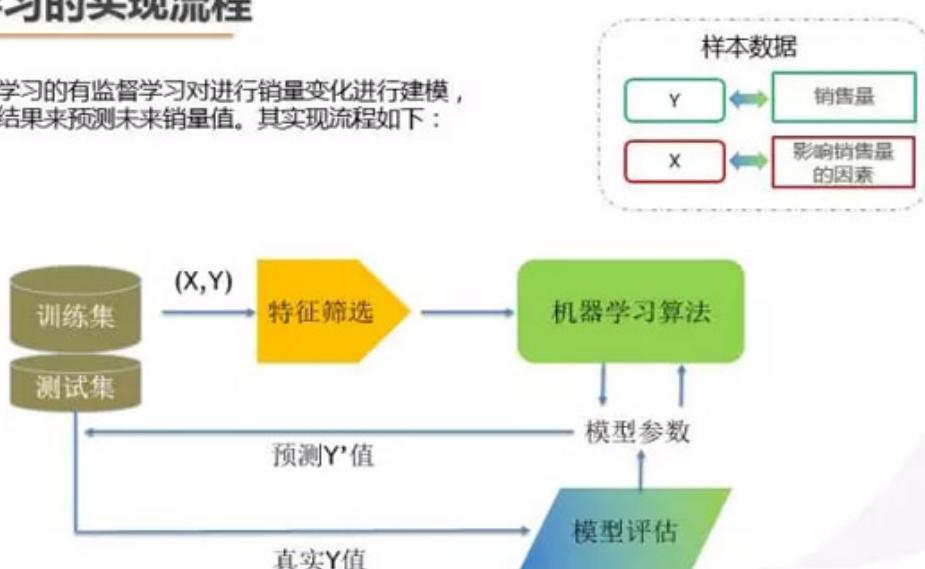
ARIMA 模型是相对比较复杂一点，其原理是用销量去预测未来销量。

无论是指数平滑还是 ARIMA 模型，其预测对于趋势性较强的数据集效果比较



机器学习的实现流程

使用机器学习的有监督学习对进行销量变化进行建模，依据建模结果来预测未来销量值。其实现流程如下：



线性模型

随机分布

>指数分布、泊松分布等

线性回归

$$Y = \theta_0 + \theta_1 * X_1 + \theta_2 * X_2 + \dots + \theta_N * X_N$$

Y为销量值，X为预测变量，N为预测变量个数， θ 为参数

链接函数

>链接函数为log函数

好，但如果遇到趋势不那么强的数据集，则效果不太理想，这时，可以考虑用机器学习的方法进行销售预测。

机器学习的整体流程为：首先，将数据集划分为训练集和测试集，其次，对于训练集做特征筛选，提取有信息量的特征变量，而筛除掉无信息等干扰特征变量，再次，应用算法建立模型，最后，结合测试集对算法模型的输出参数进行优化。

这里主要介绍线性回归模型、决策树（回归树）模型、随机森林、xgboost、神经网络、支持向量回归等六种算法模型。

线性回归模型：假设销量与影响销量的因素是线性关系的，包括误差分布、线性方程和激活函数等。

决策树（回归）：其原理是通过 if-then 规则对特征变量进行逐步决策来构建的模型。此处，可以举一个例子来简单讲解决策树算法的思想是什么？比如说我想给一个妹纸进行颜值评分，分值范围为 [0, 10]。评分的第一轮判断是五官是否端正？如果为否，打 3 分；如果为是，则进行第二轮判断，即身材，身材不好则打 5 分。身材好的话再进入第三轮判断，即是否有钱，有钱就是典型的白富美，就是 9 分。没钱则为 7 分。从图中可以看出，其判断决策的过程倒过来看是一个树，红色是它的叶子，叶子对应他的分值，黄色是变量。

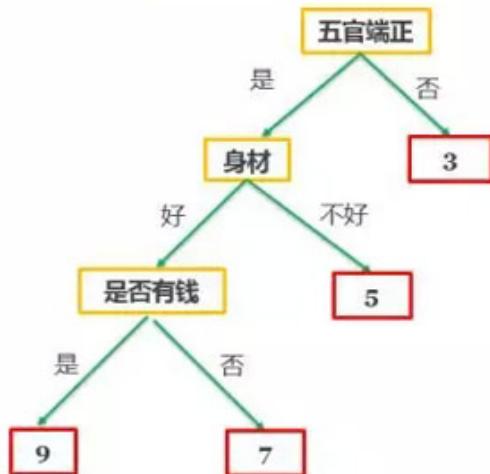
原理简介：

- 通过自变量与因变量直接建立线性关系
- 数值型回归

优(缺)点：

- 模型可解释性强
- 只适用于线性规律

妹纸评分 : [0,10]



原理简介 :

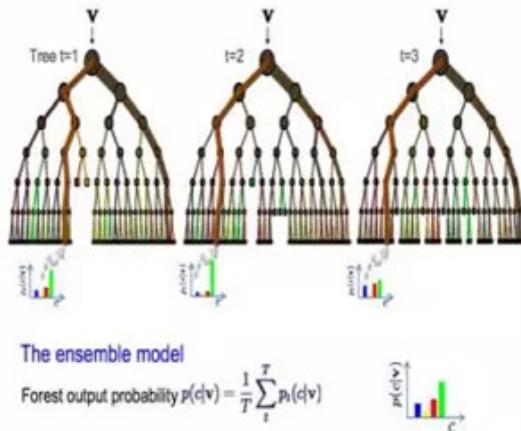
- 通过训练数据，形成if-then规则集合
- 由根节点到叶节点的每一条路径构成规则
- 对结果变量有主要解释作用的特征会先分裂形成规则
- 回归树用平方误差最小化准则，节结点为单元内数值的平均值

优点 :

- 可拟合非线性规律，计算复杂度较低

缺点 :

- 容易出现过拟合



原理简介 :

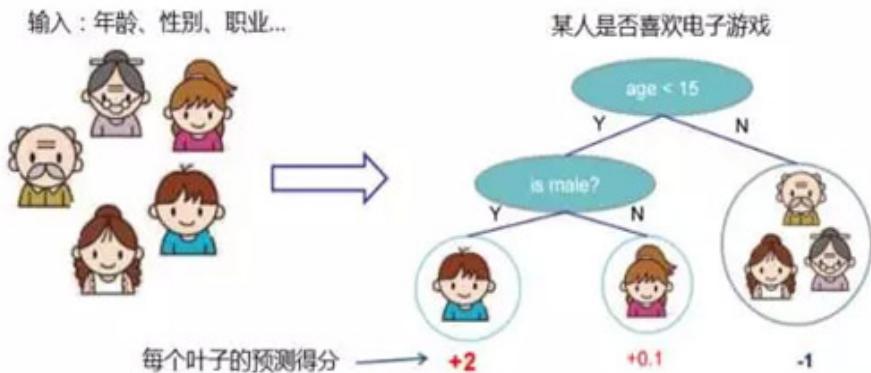
- 是包含多个决策树的组合分类器
- 输出的类别是由个别树输出的类别的众数而定

优(缺)点:

- 准确度高
- 训练速度快
- 容易做出并行算法
- 可处理大量变量并评估变量重要性
- 不会产生过拟合问题

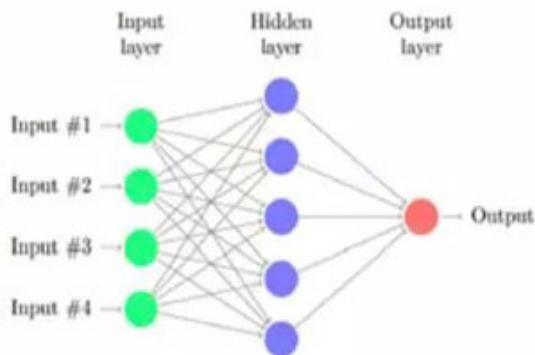
随机森林是从决策树演变而成的一个算法，但其思想与决策树相比增加了集成思想。同时，其“随机”具有两层含义，第一层是对特征变量进行随机选择。第二层是，对训练集样本进行随机选择。

xgboost 是基于传统的 GBDT 算法进行了优化的集成算法，它是数据挖掘大赛上面得分非常高的算法。它的思想是这样的，我给一个数据集，我现在有一个问题，就是要看他一家人当中是否会喜欢电子游戏，也是通过构建树的情况进行判断，比如年龄、性别进行判断，它会反映这个家庭成员对应的我们的样本会打一个分，最后男孩给 2 分，女孩给 1 分。有时候我们一棵树确定不了，我们就规



定多棵树，树 1 和树 2 之间并不是独立的，第一棵树的时候对样本做第一次判断，判断的时候有对和错，但是我会更关注于我判断错的那一部分，我在规定第 2 棵树的时候，我把预测错的更多的考虑一下，就会变成第 2 棵树，我会过多的关注那些预测错的，再依次的来进行优化。

$$Y = f(X) \quad (\text{非线性映射})$$



原理简介：

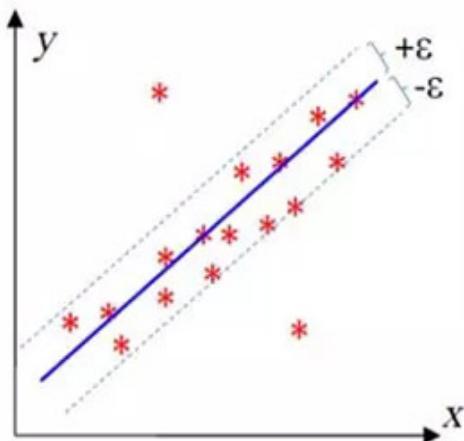
- 是利用一系列非线性回归，将预测变量映射到结果变量的一种方法。

优(缺)点：

- 准确度高
- 训练速度快
- 并行处理能力强
- 需要大量的参数
- 不能观察学习的过程，对结果难以解释

神经网络是指模拟大脑神经元的工作的非线性模型，神经网络是现在最火的一个深度学习的基础。其包括三个部分：输入层、隐藏层和输出层。输入层在销售预测中则为影响销量变换的各相关因素变量；输出层为销量；中间隐藏层为各相关因素变量到销量之间的一个非线性映射关系，通常为一个函数。

神经网络是在反欺诈领域用得比较多，像现在的银行、互联网金融，有的人进行欺骗性的贷款，就用神经网络可以很快的把他发现出来。还有检测病人也可以用到神经网络。



原理简介：

- 是通过寻求结构化风险最小来提高学习泛化能力，实现经验风险和置信范围最小化，从而达到获得良好统计规律的目的

优点：

- 可以解决小样本情况下的机器学习问题
- 可以解决高维、非线性问题

缺点：

- 对非线性问题没有通用解决方案，对核函数的选择非常敏感

SVR最本质与SVM类似，都有一个margin，只不过SVM的margin是把两种类型分开，而SVR的margin是指里面的数据不会对回归有任何帮助。

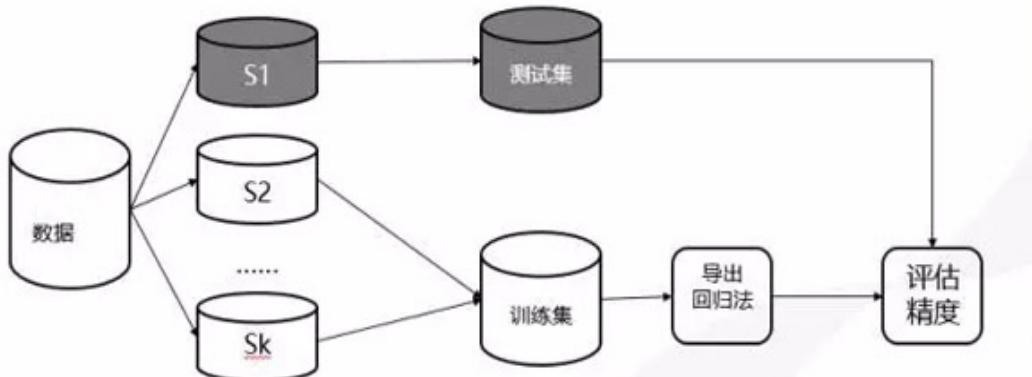
支持向量回归其本质是跟 SVM 是一样，即寻找能使回归局域更大的 margin，其适用于小数据集和高维数据集。

到目前为止，已经简单介绍了六种常用于销售预测的机器学习方法，这些算法也就很好地回答了前面“双十一”销售额例子的第一个问题，即如何进行销售预测？

销售预测效果评估

第四步是评价指标，即预测效果好与坏的问题。对此问题，我想从两个方面跟大家进行分享，即评估方法论和评估的定量指标。第一方面，方法论 K 折交叉验证。其基本思想为：将总数据集均匀划分为 k 等份（假设取 k=10），第一次对数据集进行划分过程为：第一份作为测试值，验证这个模型，剩下第 2 到第 10 个做训练集。第二次划分过程为：把第 2 个作为测试值，剩下 9 个作为训练集，然后依次进行训练集和数据集划分，一共会，得到 10 个模型，选择最小的作为

- 在k-折交叉验证中，初试数据被划分成k个互不相交的子集或“折”，每个折的大小大致相等。训练和测试k次。在第*i*次迭代中，第*i*折用作测试集，其余的子集都用于训练分类法。
- 准确率估计是k次迭代正确分类数除以初始数据中的样本总数。



- 与分类模型不同，回归模型是对连续的因变量进行预测，因此判断回归模型的准确率需要考虑的是预测值与真实值之间差异的大小。

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

- 其中， y_i 为第*i*个样本的真实值， \hat{y}_i 为第*i*个样本的预测值，*n*为样本量。
- 有时也用 $MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$ 来评估回归模型的准确率，与RMSE效果相同。

我们最终的模型。

第二方面是评估指标 RMSE，值越小，说明预测值与真实值之间的差异就越小，模型效果就越好。

至此，已经把销售预测的四大步骤均已经介绍完毕了，下面就以某电商网站的销售预测案例作为理论实战分享。

项目案例

下面我们进入机器学习的实战部分。我会大家看一下案例在上述理论框架的效果怎么样。

第一步，确定预测目标是为某类商品历史销量排行前 20 的单个商品进行未来 7 天的预测。

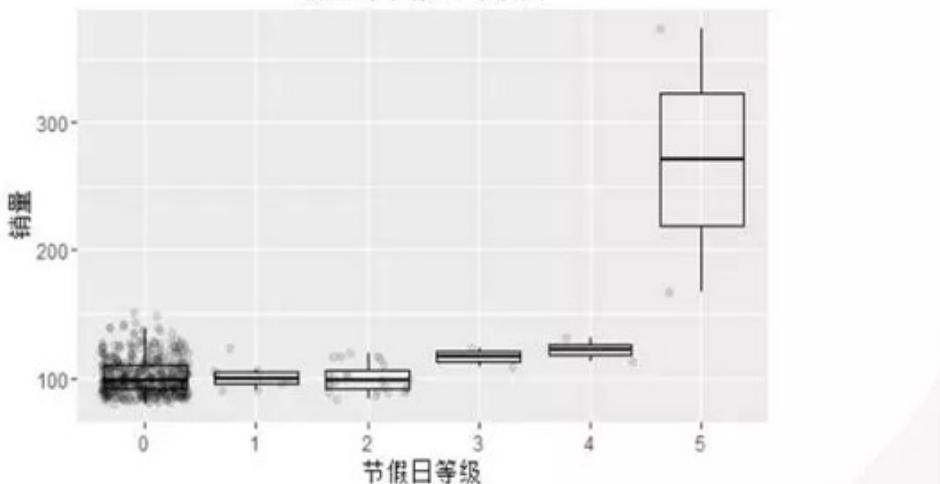
深圳某电商平台主营海外代购业务，由于海外代购物流时间长、发货时间慢等因素导致该电商平台存在大量库存积压情况，想通过销量预测模型改善安排进货、提高发货速度以及优化库存。



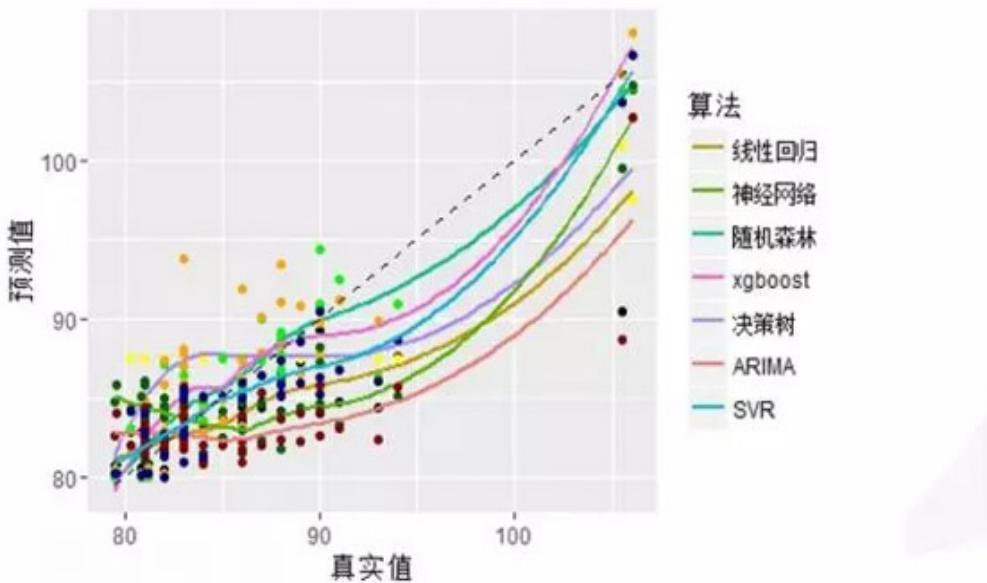
第二步，收集和理解数据，本案例一种为六大维度 72 个指标。其实，影响销量变化的因素错综复杂，除了本身历史销量外，还有一些：比如说竞争对手的因素、促销因素、新闻热点因素、口碑因素、随机事件因素、非技术因素等等，但是，对于算法建模而言，我们需要把有数据支持的、并且能够可控的影响销量的相关因素考虑进去，因此，才得出了六大维度 72 个指标（特征变量）体系。

我们对 72 个指标（特征变量）进行了数据探索，这里以节假日等级与销量

销量与节假日等级图

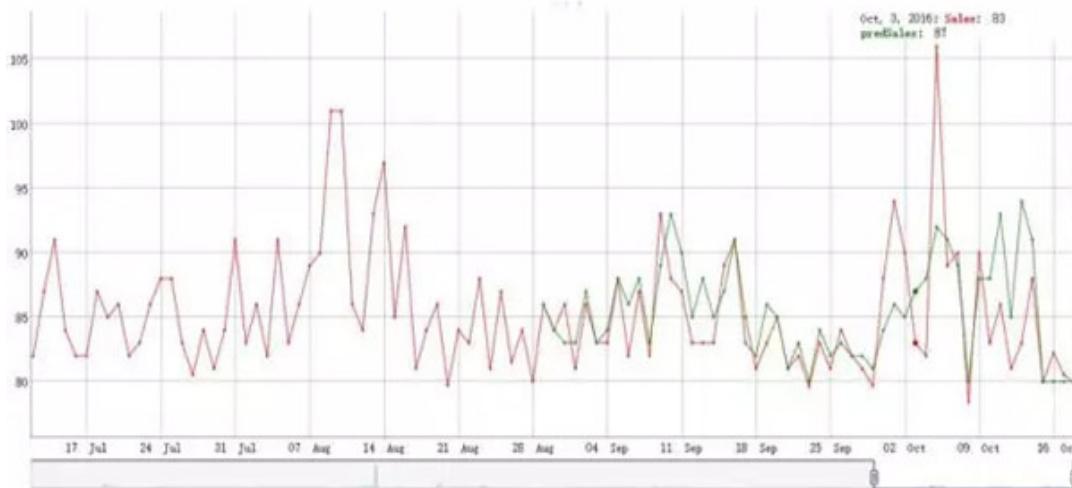


节假日变量对销量的影响明显



的关系为例，通过对数据探索发现了一条规律，即：节假日等级越高，销量会越好。

第三步为应用建模，我们使用了 6 个机器学习模型和 1 个 ARIMA 模型对该案例进行了预测建模，其结果如上图所示，该结果对比图横坐标是真实销量值，纵坐标为算法模型预测值。不同颜色的线对应不同的算法预测结果拟合线，中间 45° 虚线为参考线，与参考线越近的算法，其效果越好。从图可以看出 xgboost 和随机森林两个算法的效果是比较好的。



红色点线：某商品的真实销量

绿色点线：某商品的预测销量

RMSE（均方根误差）：3.68837

算法名称	预处理	变量选择	可解释性	准确性
ARIMA	缺失值/变量筛选	stepBIC	低	低
线性回归	缺失值/标准化/变量筛选	stepBIC	高	低
决策树	缺失值/标准化	信息增益率	高	低
随机森林	缺失值/标准化	模型选择	高	高
xgboost	缺失值/标准化/变量筛选	stepBIC	低	高
神经网络	缺失值/标准化/变量筛选	stepBIC	低	低
支持向量回归	缺失值/标准化/变量筛选	stepBIC	低	高

第四步为评价指标，这里不仅输出 RMSE 值，而且还将历史销量与未来预测销量进行可视化展示，即如上图所示。

我们对所有的预测算法进行了总结，包括预处理、变量筛选方法、可解释性和精确性等内容。

同时，我们也对基于机器学习对销售预测的研究进行了总结，主要分机器学习、数据、效果和业务四个方面。

机器学习	是场景局限性，机器学习 不是万能的 ； 研究的是 相关关系 ，而不是因果关系。
数据	是核心， 无数据或数据质量低 ，会影响模型预测效果； 是模型选择的先决条件， 先数据，后模型 。
效果	评估需要参考业务对接、预测精度、模型可解释性和产业链整体能力等因素 综合考虑 ； 不能简单作为企业利润增加的 唯一标准 。
业务	对建模提供 业务理论基础 ； 算法问题要回归到 业务问题 。

第一方面是机器学习层面，即机器学习可能更多的是关注相关关系。做销售预测，我们只用到了跟销量变化有关的因素变量，而不是因果变量。有些客户会问我们：现在我要提高商品销量，请你们算法人员告诉我调哪些参数能够让销量增加？其实这就是一个不合理的需求，因为我是做预测的时候我们用到机器学习的模型，我只是用到像库存、价格、节假日，是跟销量无关系的，但是并不是默认认为库存高就一定会影响销量的增加，库存低就一定影响销量减小。因为后者它是一个因果关系，而我们做机器学习用的是一个相关性的关系。

第二方面是数据的层面，没有数据或数据质量差，效果就会较差，业界流行一句话“垃圾进垃圾出”就是指数据。

第三方面是效果层面。其实，效果是评估一个模型对于好与坏。对于销售预测而言，其效果可能主要体现在是否增加企业的利润。但是这样的效果不太好评论，因为这不仅仅要考虑预测的精度、模型的可解释性等算法效果，还要考虑企业的供应链、整体能力等。不能将机器学习的预测效果作为衡量企业是否增加利润的唯一标准。

第四方面是业务层面，即在机器学习训练之前的数据预处理、训练之中以及训练之后的模型评估都需要一定的业务理论作为指导。尝若业务理论偏弱，则可

能会影响整个建模过程，其效果也会受到影响。而且我们是从算法的角度，解决的是算法的问题，但是算法的问题，最终还是要回到我们业务问题，在销售预测上我们要回到怎么样提高业绩。

销售预测终究还是一个商业问题，我们只是从数据层面，从算法层面很难能够很好的把这个商业问题解决掉。我们认为这个效果已经比较好了，其实是比那种大数据的精准营销、精准预测还是有一段的距离要走。这是我们对于用机器学习来做产业落地的一个展望和规划。

答疑环节

提问 1：销售预测的销售指标如何定？通过哪些维度预测？预测周期大概多长？

答：销量预测的指标体系的整理主要是通过业务逻辑和待预测商品本身的数据情况而确定的。比如电商类某商品的销量预测，从业务上进行商业分析可以得出，库存因素、价格因素、口碑因素、节假日因素、促销广告因素、新闻热点因素（比如三鹿奶粉被新闻曝光，导致国产奶粉在线上销售受阻）、国家政策因素（主要是指跨境电商，国家税率的高低）等因素，同时，也需要结合待预测商品本身的数据情况，如果以上的几个业务上分析的因素，只有库存因素和节假日因素，那就只能用这两个因素进行预测了，后期等获取到其他因素的数据再进行补充。

而预测周期的确定需要业务部门结合企业的整体供应链能力以及盘点本身历史数据情况而定，如果企业的数据质量较好，历史数据量又较多（比如超过 2 年以上的历史数据），那么预测周期可以设置长一点。但是对于销量预测而言，有一个特点，即短期预测的精度要远高于长期预测的精度，也就是说预测未来一周的精确率要比预测未来三个月要高得多。

提问 2：电商类销量预测，有什么特殊的要求和使用的场景？比如样本量，准确性？

答：电商类的销量预测与实体的销量预测的主要区别在于用户体验上，我们称电商销售预测为线上预测，而实体的销量预测，比如衣服的销量预测、药店的销量预测和文具的销量预测等都属于线下预测；而对于线上预测，除了本身的商业环境影响（价格、库存、质量、评价等等）外，还有一部分影响销量的因素是互联网上的用户行为数据，主要包括浏览、点击和收藏等数据，这些用户行为数据主要是通过布码技术活动的（通过将抓取的代码部署在 web 端或 PC 端），同时，还需要考虑用户行为数据的滞后性，即某用户收藏了该商品，但是需要经过一定时间才能下单购买，这个时候需要把具有滞后性的变量进行拆分，再加入到机器学习模型中去。

对于样本量的多少，需要以预测的目标而定，预测目标较短，则所需要的训练集就少一些，如果预测目标较长（超过 5 天），一般则需要 1 年以上的历史数据。至于准确性，需要结合数据情况而定，准确性的指标通常用 RMSE，RMSE 越小则准确性越好。

讲师介绍

唐新春，百分点数据科学家，清华大学硕士学历，曾负责完成中国某兵器研究院的大型激光器相关算法的研发项目，在加入百分点之前，曾在生物信息公司中负责生物大数据的分析和数据挖掘；在百分点负责在金融领域的征信模块开发、销售预测领域预测模型研究，以及零售类用户画像的研发等工作。现研究兴趣为运用机器学习、数据挖掘等技术在产业界的落地实践，为不同行业提供大数据服务。

版权声明

InfoQ 中文站出品

架构师特刊：机器学习实践

©2016 极客邦控股（北京）有限公司

本书版权为极客邦控股（北京）有限公司所有，未经出版者预先的书面许可，不得以任何方式复制或者抄袭本书的任何部分，本书任何部分不得用于再印刷，存储于可重复使用的系统，或者以任何方式进行电子、机械、复印和录制等形式传播。

本书提到的公司产品或者使用到的商标为产品公司所有。

如果读者要了解具体的商标和注册信息，应该联系相应的公司。

出版：极客邦控股（北京）有限公司

北京市朝阳区洛娃大厦 C 座 1607

欢迎共同参与 InfoQ 中文站的内容建设工作，包括原创投稿和翻译，请联系 editors@cn.infoq.com.

网 址：www.infoq.com.cn