第5章

算法选择和开发

正如第1章所述，人工智能技术包含多个领域，每个领域的算法都非常多，而且每年还有层出不穷的新算法出现。在开发一个人工智能应用之前，有必要结合具体业务场景和数据的可获取情况，快速锁定一些合适的算法，这样可以大大提升应用开发效率。

本章节重点围绕几个常用的人工智能技术领域，介绍一些经验方法，辅助人工智能应用开发者选择合适的算法，在找到合适的算法之后，可以直接订阅 ModelArts 预置算法开始训练，也可以自行开发和调试算法代码，然后再训练。

5.1 算法选择

通常在开发人工智能应用时，都需要依赖算法工程师的个人经验选择某个算法。随着人工智能各技术领域的逐渐成熟，很多经典算法的优缺点及发展路线都已经较为清晰，开发者可以结合具体业务问题及其他一些限制条件（如业务数据的现状，以及业务方对训练速度、训练精度、推理速度的要求等）定性地选择某一个或某一类算法，这对于快速试错非常重要。快速获取性能基线，可以帮助开发者进一步针对性地做迭代优化。下面将针对几个常用的人工智能技术领域的典型算法展开介绍，以辅助应用开发者进行算法选择。

5.1.1 基础层算法选择

对于应用开发者而言，平时接触最多的基础算法应该是机器学习（包括深度学习）和强化学习。下面将以这两个为重点展开介绍。

1. 机器学习算法选择

常用的机器学习算法分为分类、聚类、时序预测、异常检测、关联分析、推荐等，如图 5-1 所示。下面将分别按照任务维度简要介绍一些常用的算法，旨在为开发者提供算法选择的参考。

在分类任务中，逻辑回归算法实现简单，经常被作为性能基线与其他算法比较。逻辑回归算法也可以被看作是一层神经网络，由于其计算复杂度低，经常被拓展到更加复杂的问题上，如大规模推荐。而支持向量机擅长解决高维度非线性分类问题。支持向量机模型的计算复杂度是数据集大小的二次方，因此不适合处理大规模数据。逻辑回归、支持向量机在多分类场景下的应用需要依赖一些额外的技巧，如与集成学习相结合等。

最初的 KNN 算法不需要训练，它直接根据邻近的有标签数据的投票来对未知签数据进行分类。然而，在实际应用中，由于数据样本的距离度量方式是不可知的，所以 KNN 算法需要在常用的几个距离度量方式中去选择并学习合适的度量方式，这时就需要训练。度量学习的目的是学习一个度量矩阵，使得在某度量方式下，数据中同类样本之间的距离尽可能减小，而不同类别样本之间的距离尽可能增大。常用的度量学习方法分为全局度量学习和局部度量学习。深度学习也可以与度量学习相结合，利用深度神经网络自适应学习特征表达，当数据量较多时，推荐使用深度度量学习。深度度量学习已经成功用于人脸识别等领域。

决策树通过递归划分样本特征空间并在每个得到的特征空间区域定义局部模型来做预测。决策树方法的优点是易于理解，数据预处理过程比较简单，同时在相对短的时间内就可以在大数据集上得到可行且效果良好的结果。决策树是非常基础的算法，可解释性强，但它缺点也比较明显，对连续性的特征比较难预测。当数据特征关联性比较强时，决策树的表现不会太好。通常，决策树需要与集成学习方法一起使用，才会有较好的精度。随机森林、GBDT 等算法已经在工业界广泛使用。

由上可知，当处理的问题是二分类问题且数据集规模不大时，支持向量机是首选算法；如果支持向量机的效果不是很理想，则可能是因为该矩阵不能很好地度量样本之间的相似性，因此可以尝试度量学习算法。对于数据集比较大的情况，则首先选择基于决策树的集成学习方法来解决。当然，其他不同的模型也都可以与不同的集成学习策略（如 Bagging、Boosting、Stacking）相结合，进一步提升模型效果，但集成学习通常也会使模型更加复杂，增加训练和推理的计算成本。

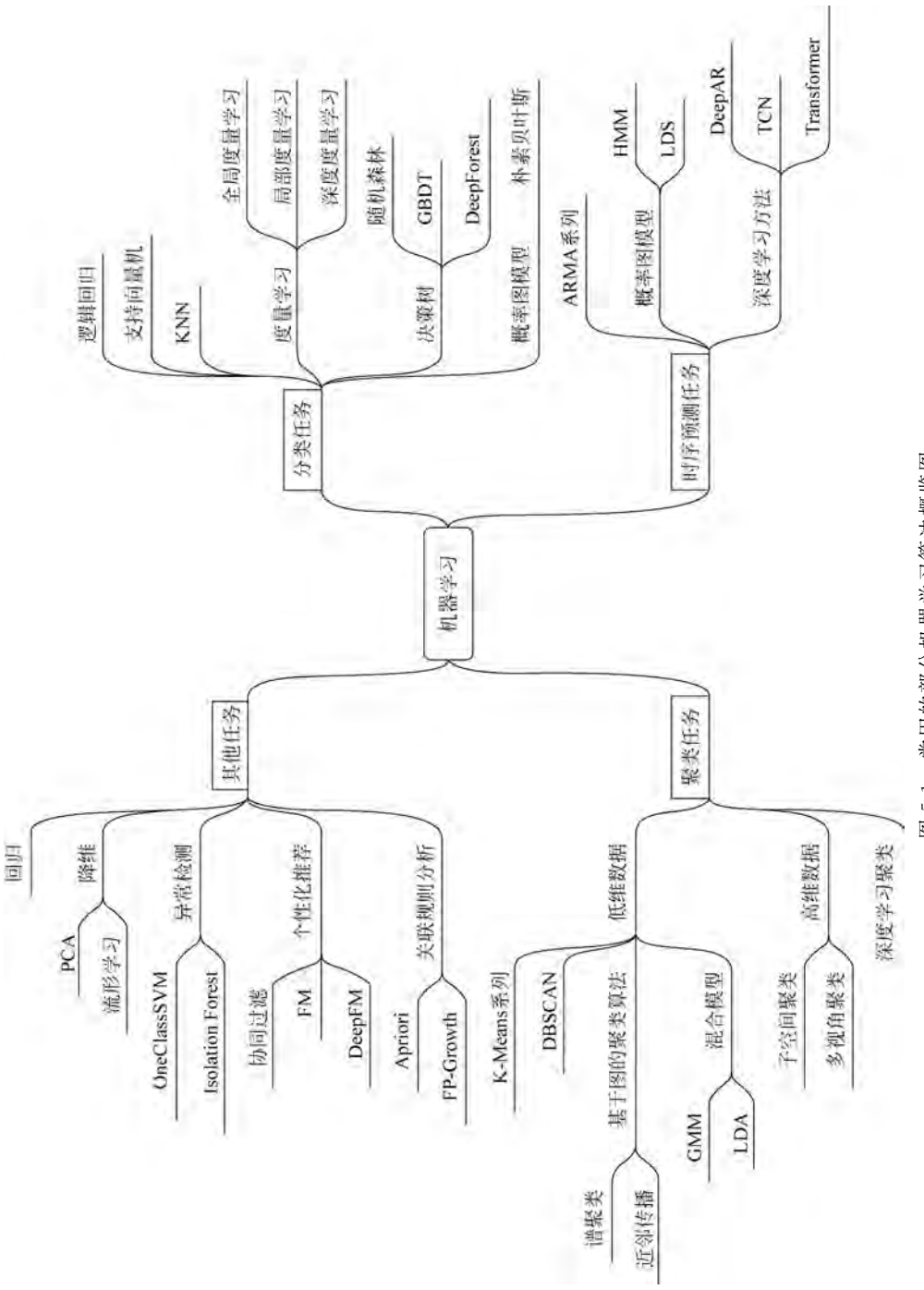


图 5-1　常用的部分机器学习算法概览图

聚类任务中最常用的算法是 K-Means、基于图的聚类算法及混合模型。K-Means 算法已在第 1 章介绍过，其存在两个问题：不同的初始中心点对最后聚类结果的影响非常大；聚类簇数量不容易提取判断。K-Means++通过改进初始中心点的选择来改善 K-Means 算法的聚类效果。基于密度的聚类算法，如 DBSCAN 算法，不需要提前知道聚类簇数量。基于图的聚类算法有谱聚类和近邻传播。谱聚类利用相似度矩阵的特征向量进行聚类。近邻传播算法的基本思想是将数据样本点看作网络的节点，然后通过网络中各边的消息传递计算出各样本的聚类中心。与传统的聚类算法相比，近邻传播算法特别适合高维、多类数据的快速聚类，在聚类性能和效率方面都有大幅度的提升。基于混合模型的聚类方法则是假设每组数据都可以通过一个模型来拟合，该方法的好处是最后的聚类结果会给出样本属于每个聚类簇的概率。常见的混合模型有 GMM(Gaussian Mixture Model，高斯混合模型)和 LDA(Latent Dirichlet Allocation，隐狄利克雷分布模型)。以上聚类算法在处理高维数据时会面临很多问题。为了解决这些问题，建议采用子空间聚类和多视角聚类的方法。由上可知，在提前知道聚类簇的数量时，以上聚类算法都可适用，否则可选的算法只有 DBSCAN 和近邻传播算法；如果聚类结果需要知道样本属于每个聚类簇的概率，则选择基于混合模型的聚类方法；对于高维数据的聚类，子空间聚类和多视角聚类是首选方法。

时序预测任务中的传统算法有 ARMA 和 NARMA 等，随着机器学习和深度学习的发展，基于 SVM、神经网络等的方法也开始流行起来。近几年基于深度学习的时间序列预测主要以循环神经网络为主(例如 DeepAR 等)，其提高了多变量时间序列的精度，但是在大规模分布式并行方面有不少的挑战。基于 CNN 架构的时间卷积网络 TCN(Temporal Convolutional Nets)的计算复杂度要更低，性能更好。另外，如果要解决长时间的序列预测问题，建议采用基于注意力机制的 Transformer 模型。

概率图模型通过在模型中引入隐变量，增强了模型的建模能力。混合高斯模型、隐马尔可夫模型、条件随机场都属于概率图模型。概率图模型可以用于分类、聚类、时序预测任务，如相关向量机和朴素贝叶斯可以用于分类任务；隐马尔可夫模型和线性动力系统可以对序列化数据进行建模；而混合高斯模型常用于聚类任务。概率图模型可以为模型和预测结果提供概率解释。由于经典机器学习在实际应用过程中需要结合业务领域知识构建特征工程，这个过程中有很多手工工作，因此深度学习方法在不同任务的算法中使用深度多层神经网络从原始数据中学习更好的特征表示，如分类任务中的深度度量学习，聚类任务中的深度学习聚类方法等，取得了比原始算法更好的

效果。传统决策树也可以与深度学习思想(不是深度神经网络)相结合,如DeepForest。机器学习领域目前正在朝着 AutoML 的方向发展,很多著名的机器学习算法库(如 Scikit-learn)都演进出了自动版(如 Auto-sklearn 等)。

其他机器学习任务还包括关联规则分析、异常检测和个性化推荐等。关联规则分析常用的经典算法主要有 Apriori 算法和 FP-Growth(Frequent Pattern-Growth,频繁项增长)算法,后者在计算速度上更快。异常检测、新样本检测算法用于发现新的数据点和异常数据点,常用算法有 OneClassSVM、Local Outlier Factor 等。OneClassSVM 适用于数据量少的情况,对于高维度特征和非线性问题可以体现其优势。Local Outlier Factor 算法对数据分布的假设较弱,对于数据分布不满足假设(如高斯分布等)的情况,建议使用这种算法。推荐场景下,一般都是高维稀疏数据,可以采用特征学习与逻辑回归相结合的方法,也可以尝试 FM(Factorization Machine,因式分解机)及其深度学习版本 DeepFM。

此外,还需要从数据标注量的角度来考虑采用哪些算法。在有些场景下,标签数据是自动可以获取的。例如,销售量预估场景下,随着时间的推移,真实的销量结果会不断产生,可以用于时序模型的持续迭代。在很多场景下,标注未必是准确的,比如对于某网站的评论区文本分类问题,用户的反馈可能是带有不准确性的。还有很多时候,标注量严重不足,尤其在医疗等行业中。针对这些问题,就需要采用半监督、弱监督学习方法。但是,半监督、弱监督也都代表的是学习策略,本质上还是要与每类算法(机器学习、计算机视觉、自然语言处理等)相结合才可以发挥作用。

2. 强化学习算法选择

在机器学习中,数据不同会导致算法表现不同。同样地,在强化学习中,由于目标环境的多样性,算法在不同的环境中表现截然不同。另外,结合业务场景,开发者在其他维度(如算法输出动作的连续性或离散性、算法的学习效率等)上可能还有不同的要求。因此,选择合适的强化学习算法是一个很重要的工作。

如第 1 章所述,根据环境是否由模型直接描述,强化学习算法可以分为 Model-free 算法和 Model-based 算法(见图 5-2)。Model-based 算法包括 Dagger(Data Aggregation)、PILCO(Probabilistic Inference for Learning Control)、I2A(Imagination-Augmented Agents)、MBMF(Model-Based RL with Model-Free Fine-Tuning)、STEVE(STochastic Ensemble Value Expansion)、MB-MPO(Model-Based Meta Policy Optimization)、MuZero、AlphaZero、Expert Iteration 等。

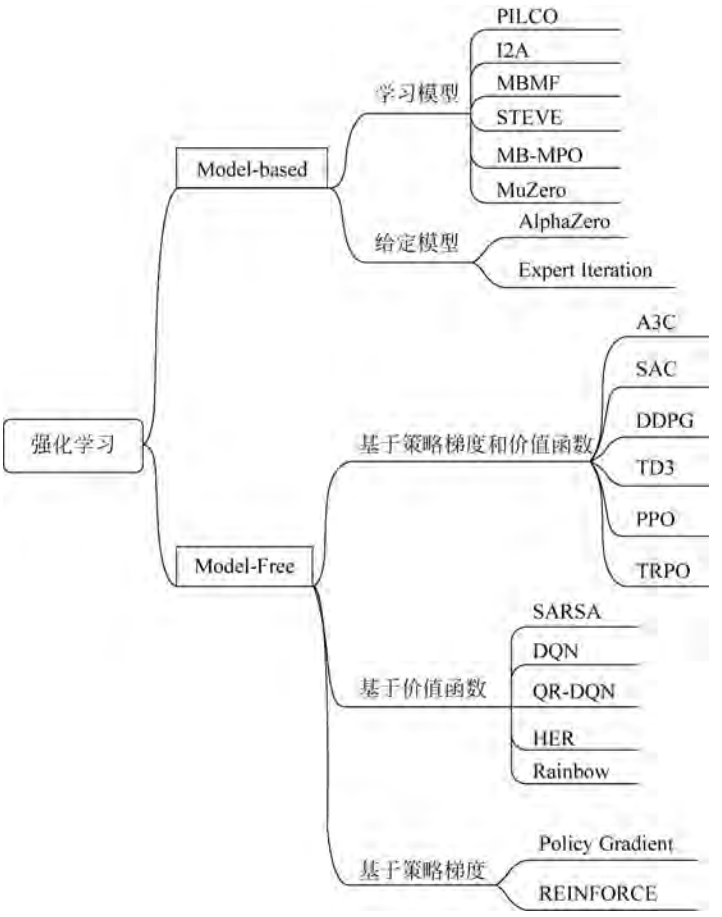


图 5-2　常用的部分强化学习算法概览图

当智能体所处环境是确定性的、不随时间变化，且开发者对环境建模并不困难时，建议选择 Model-based 算法。Model-based 算法先从强化学习主体与环境交互得到的数据中通过监督学习的方式学习环境模型，然后基于学习到的环境模型进行策略优化。在环境简单、观测状态的维度较低时，PILCO 可以显著提升采样效率，但由于其使用高斯过程回归模型对环境进行建模，模型复杂度随着状态维度指数增长，因此难以应用于复杂环境中。STEVE 和 MB-MPO 使用模型集成的方式来表征模型的不确定性，能够有效地推广到高维状态空间。STEVE 使用值展开的方式将环境模型与 Model-free 算法相结合，当环境无法学习时能够退化为 Model-free 算法。MB-MPO 利用元学习的方法在环境模型的集成中学习到足够鲁棒的自适应策略，该方法的策略优化过程完全基于环境模型生成想象样本，因此 MB-MPO 采样效率很高，但当环境难

以建模时策略将无法学习。

Model-based 算法适合对象环境相对简单明确、能够进行机理建模的系统，如机器人、工业制造系统，这类算法在采样效率、收敛速度等关键性能上体现出了优势。而 Model-free 算法则表现出更广泛的适应性，绝大部分环境不需要对算法进行适配，基本上只要满足接口，就可以做到即插即用。因此，对于大多数环境，特别是复杂性较高、难以建模的环境，如游戏、金融等，可以直接用 Model-free 算法尝试。

在 Model-free 算法当中，如第 1 章所述，根据动作取决于策略函数的输出，还是值函数输出的最大值，可以分出策略梯度和价值函数拟合两大类算法。如第 1 章所述，基于策略梯度的算法包括 REINFORCE、PG，基于值函数拟合的算法主要包括 DQN 及其一系列衍生算法，如 QR-DQN(Quantile Regression DQN)、HER(Hindsight Experience Replay)、Raninbow 等。A3C、SAC(Soft Actor-Critic)、DDPG(Deep Deterministic Policy Gradient)、TD3(Twin Delayed DDPG)、TRPO(Trust Region Policy Optimization)、PPO(Proximal Policy Optimization)这些最近几年出现的算法都结合了策略梯度和价值函数拟合两类算法的优点，同时学习价值函数和策略梯度函数。

DQN 等基于价值函数拟合的算法大多采用离轨策略，即采用单独的策略来更新价值函数，通常可以从历史上积攒下来的样本经过采样后进行价值函数的更新。其优点是：①可以从人类示教样本中学习；②可以重用旧策略生成的经验；③可以同时使用多个策略进行采样；④可以使用随机策略采样而优化确定性策略。

但是，DQN 存在着价值高估的固有缺陷。DQN 有一系列优化后的版本，其中 Rainbow 算法是 DQN 系列的集大成者，使用了各种 DQN 变体中的改进方法。Rainbow 相较于其他 DQN 系列算法，性能有显著提升，但正是由于使用了过多的技巧，其单步训练时间较长。由于历史原因，DQN 系列多用于 Atari 等以图像作为状态输入的环境。需要注意的是，由于 DQN 系列算法属于基于价值函数拟合的方法，所以仅适用于动作空间离散的场景，当动作空间维度很高或是连续时将无法求解。另外，这些算法都是确定性策略方法，无法学习随机策略。

基于策略梯度的强化学习算法能够很好地解决连续动作空间问题。此外，这些算法大多采用在轨策略，且学习的策略都是随机策略，所以学习效率较低。正因如此，目前主流的策略梯度算法都会与值函数拟合算法相结合。当对算法的迭代步长非常敏感时，建议采用 TRPO(Trust Region Policy Optimization)和 PPO(Proximal Policy Optimization)。这两种算法都采用在轨策略，并且都同时适用于连续动作空间和离散动作空间的决策问题，并输出随机策略。TRPO 需要求解约束优化问题，计算复杂。

为解决这一问题，PPO 对 TRPO 进行一阶近似，使用裁剪或惩罚的方式限制了新旧策略间的分布差距。一般情况下，建议直接使用 PPO 即可。

上述策略梯度算法虽然能够解决高维动作空间问题，但它们产生的都是随机策略，即输入同一状态，输出的动作可能会不一样。在某些场景下，如果期望得到的是确定性动作（如机械臂控制场景对动作有严格要求），则建议采用 DDPG 算法，其最大特点是策略函数是一个确定性映射函数。由于确定性策略不再需要对动作空间进行积分，因此采样效率相较于其他方法都有提高，非常适合动作空间很大的情况。

此外，当强化学习中环境的奖励函数很难设计时，或者需要利用专家数据给一个较好的起点时，可以使用模仿学习、逆强化学习等方法；当环境奖励稀疏时，需要提升强化学习算法的探索能力，需要采用分布式架构、Reward Shaping 等方式以鼓励智能体去探索未知状态，也可以使用分层强化学习方法对任务进行分解。

5.1.2　应用层算法选择

对于应用开发者而言，计算机视觉和自然语言处理是最常见的两个应用层算法领域，下面将围绕这两个领域展开介绍。

1. 计算机视觉算法选择

正如第 1 章所述，目前常用的几种计算机视觉任务（图像分类、目标检测、图像分割等）大多数以深度学习为基础。常用的部分计算机视觉算法如图 5-3 所示，当开发者需要在时延-精度要求方面做出权衡时，可以考虑不同的深度神经网络架构设计；当需要在数据量和学习效果方面做出权衡时，则可以考虑不同的学习方式，如半监督学习、弱监督学习等。

深度卷积神经网络在发明之初就是用来解决图像分类问题的，到目前为止，深度学习与图像分类的结合愈加紧密，并且出现了很多经典的算法模型。自 2015 年之后，ResNet 也基本上成了很多业务场景下开发者快速尝试的标杆算法。后期出现的 DenseNet、Xception、ResNext、ResNeSt 等算法都以 ResNet 为对比对象。另外，典型的面向移动端的小型网络有 MobileNet、ShuffleNet、GhostNet 等，当开发者对于模型的推理时延要求较高时，需要直接采用这类神经网络进行训练；或者先训练一个大网络，再利用大网络产生的标签对小网络进行训练，这种方式也叫作模型蒸馏。随着神经网络结构搜索技术的不断演进（详情见第 6 章），机器搜索出的网络结构 NASNet、AmoebaNet、EfficientNet 比人工设计的网络结构更好（要么精度更高，要么推理时延更低），其中 EfficientNet 是目前较为流行的一种卷积神经网络结构。

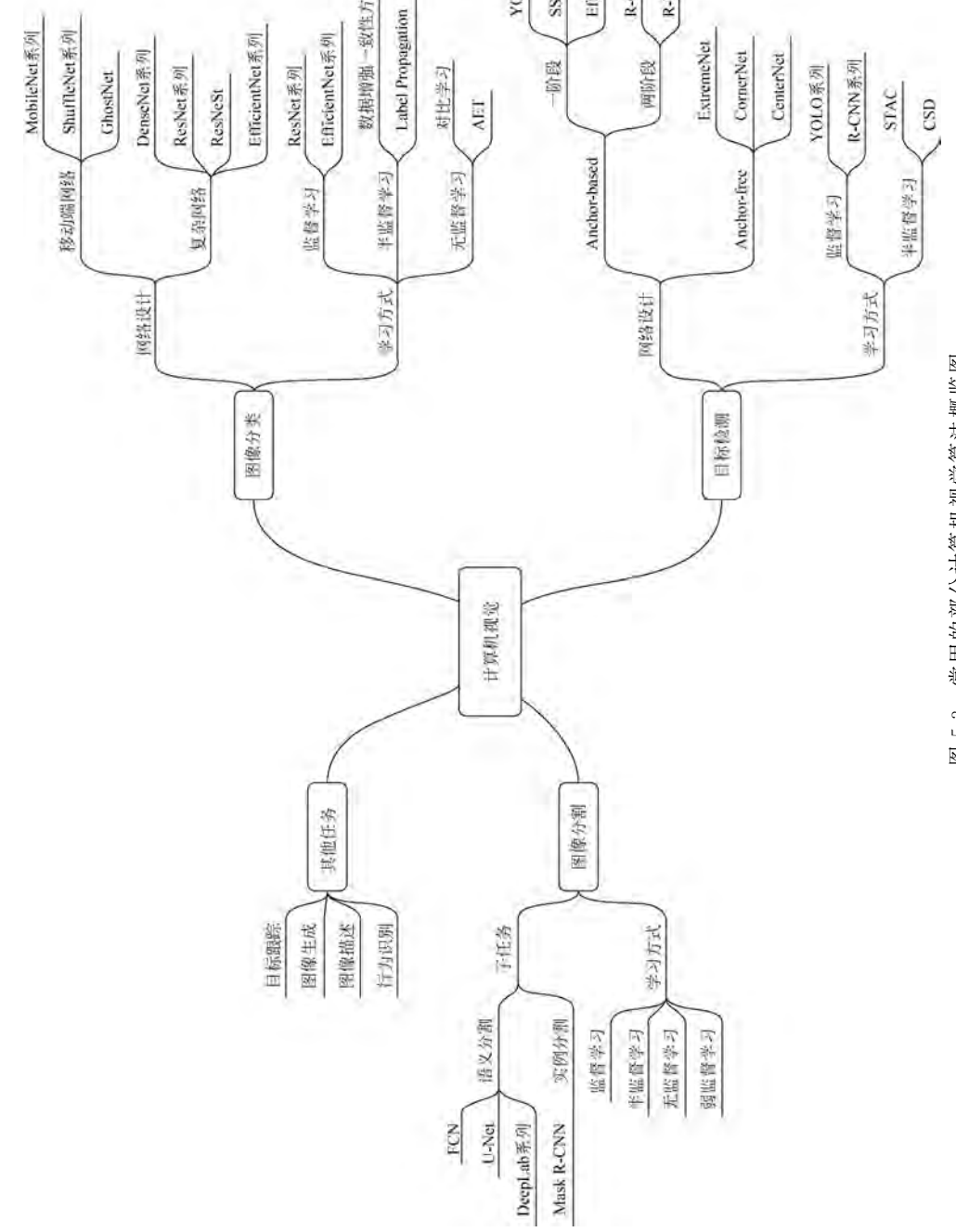


图 5-3　常用的部分计算机视觉算法概览图

当数据集中含有大量无标签数据时，开发者可以选择基于对比学习或者 AET（Auto-Encoding Transformation）等无监督学习方法得到一个预训练模型。最近一两年视觉无监督学习的进步非常大，已经非常接近于全监督学习的水平。当得到预训练模型之后，就可以用少量有标签的数据在此预训练模型之上微调即可，这就会大大减少算法对标注数据量的需求。另外，还可以选择相对成熟一些的半监督算法，如 Mixmatch、Remixmatch 及 Label Propagation 等，这些算法在标注量较少时可以获得不错的精度。

对于目标检测任务，算法可以分为 Anchor-based 和 Anchor-free 两种。Anchor-based 算法需要额外设置一个超参数——Anchor 用以引导目标框的回归，主要可以分为两类：一阶段（One-Stage）算法和两阶段（Two-Stage）算法。一阶段算法使用回归的方式输出这个目标的边框和类别，常见模型有 SSD、RetinaNet、YOLO 系列、EfficientDet 等。其中，YOLO 系列不断进化，已经有了 4 个版本。YOLOv4 在 YOLOv3 的基础上进一步大幅提高了模型的精度、降低了推理时延。EfficientDet 也充分利用了模型自动架构搜索带来的优势。常见的两阶段算法有 R-CNN 系列和 R-FCN 系列，由于引入了候选目标框提取网络，所以误检率一般比 One-Stage 算法低一些。

Anchor-free 方法将人体姿态估计中的关键点检测思想引入通用目标检测中，通过检测关键点来确定目标的位置，常见的模型有 CornerNe、CenterNet、ExtremeNet 等。对于推理时延要求比较高的场景，建议选择 One-Stage 方法，特别是 YOLOv4 和 EfficientDet；对于物体长宽比和尺寸变化比较大并且不太会设置 Anchor 的场景，建议选择 Anchor-free 方法。此外，基于半监督算法的物体检测也开始出现，如基于数据增强一致性的目标检测算法 CSD（Consistency-based Semi-supervised learning for object Detection）。然而，与半监督图像分类算法相比，半监督目标检测还不足够成熟，未来还有较大的改善空间。

如第 1 章所述，图像分割可以根据具体任务主要分为语义分割、实例分割。语义分割的典型算法有 FCN（Fully Convolutional Network）、U-Net、DeepLab 系列。实例分割的代表算法是 Mask-RCNN。还有一种不常用的全景分割，是要将前背景都进行实例分割，可以看作实例分割的一种拓展。

半监督图像分割算法利用少量标注数据构建初级模型，然后在无标签数据上获取伪标签进行分割模型的优化。无监督分割算法利用可自动生成语义标注的计算机合成数据进行无监督训练。不过，弱监督学习更加常用，即使有的图像上有目标框甚至图像级标签，也可以作为弱标注信息用来训练图像分割模型。

其他计算机视觉任务还包括目标跟踪、图像生成、图像描述和行为识别等。目标跟踪是利用摄像机在一段时间内定位移动的单个目标或者多个目标的过程；图像生成顾名思义就是利用现有数据生成新数据的过程；图像描述任务旨在生成文字来描述图像内容；行为识别是视频分类的一种，其根据一系列视频帧判断视频中目标的动作类别，在此不再一一展开。

2. 自然语言处理算法选择

正如第 1 章所述，自然语言处理常见的任务包括文本分类、序列标注、机器翻译、文本摘要等，如图 5-4 所示。总体的算法选择策略是，当对精度要求更高时，建议采用“无监督预训练＋Softmax”的方式；如果对训练时间或推理时延要求更高时，建议采用传统的经典算法。

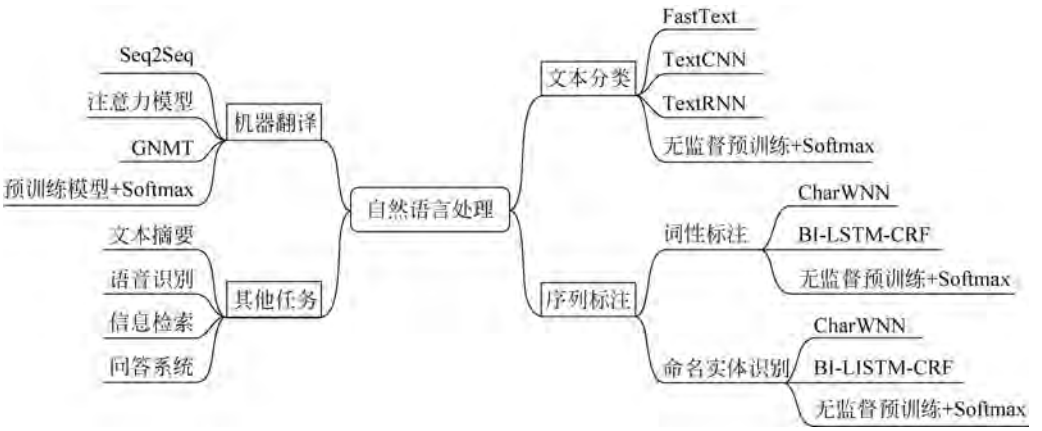


图 5-4　常用的部分自然语言处理算法概览图

文本分类任务中线性分类器是最常用的分类器。FastText 算法对句子中所有的词向量进行平均，然后连接一个 Softmax 层进行分类。由于只用到一层网络，FastText 算法的训练速度特别快。2014 年，Kim 提出 TextCNN 算法将卷积神经网络引入文本分类任务中。该算法首先对每个句子进行 Padding 以保证模型的输入大小是固定值，然后利用卷积神经网络来提取句子中类似 *N*-Gram 的关键信息。不过，卷积神经网络中卷积核的大小是固定的，这导致 TextCNN 模型无法对更长的序列信息进行建模。与卷积神经网络相比，循环神经网络天生就是为处理序列化数据而设计的。TextRNN 利用双向循环神经网络 LSTM 替换 TextCNN 中的卷积神经网络结构以捕获变长且双向的 *N*-Gram 信息。卷积神经网络所有输入的单词是等同对待的，而

循环神经网络中越靠后的单词对输出结果的影响越大，这与人类的阅读习惯不一致。人类在阅读时，句子中对语义影响最大的是中间的某几个单词。注意力机制模型用于解决这个问题。简单来说，注意力机制就是在卷积神经网络、循环神经网络等对文本序列进行建模的过程中加入的一个组件，它使得模型可以给每个单词赋予不同的权重。

在词性标注任务中，CharWNN 算法是一类经典算法，其通过卷积神经网络提取字符的嵌入表示。在得到每个单词的词向量和对应的字符向量之后，将其拼接在一起作为对应单词的最终词向量表示。BI-LSTM-CRF（Bidirectional-LSTM-Conditional Random Field）算法提出将双向长短记忆网络与条件随机场结合起来用于词性标注。这种结构可以通过长短记忆网络有效利用过去和未来的输入信息，同时通过条件随机场用于学习语句的标注信息。通过结合双向 LSTM 和条件随机场的优势，BI-LSTM-CRF 模型极大地提升了词性标注的准确度。

在机器翻译方面，2014 年 Sutskever 等人提出的 Seq2Seq 模型首次实现了端到端的机器翻译。该模型利用编码器-解码器框架，首先使用一个多层神经网络（LSTM 等）将原始句子“编码”为一个中间向量，然后再用类似的网络结构将该向量“解码”，还原出目标句子。对于长句子而言，Seq2Seq 模型也将其压缩在一个固定长度的中间向量中，有可能会丢失一些前后跨度较大的语义信息，这时就需要注意力机制来优化该模型。注意力机制通过学习联合对齐和翻译来扩展编码-解码器框架，并将其应用于多语言翻译任务中。

近几年的大量工作也在不断推进循环语言模型和编码器-解码器结构发展。但是循环神经网络固有的顺序使得模型不能进行并行计算。这样一来，内存约束也限制了模型处理很长的序列。此外，注意力机制已经成为序列建模任务的一个组成部分，注意力机制经常与循环神经网络一起使用。2017 年出现的 Transformer 算法避开了这种难以并行化的循环结构，完全依赖注意力机制来描绘输入与输出序列之间的全局依赖关系。在这之后，基于 Transformer 结构的算法或神经网络，如 Transformer-Big、BERT 等，成为机器翻译任务的首选模型。

近几年，随着 Transformer、BERT、GPT-3、NEZHA 等无监督学习算法在自然语言处理方面的兴起，基本上所有自然语言处理任务的经典算法都被超越了。在无监督模型训练之前，首先需要准备大量的语料。以 BERT 为例，其可以构造多种训练所需的目标函数，如让模型自动预测文本序列中被人为掩盖掉的词，让模型根据一段文本中的前一个句子预测下一个句子等。这个过程是不需要任何标注信息的，因为无论是

掩盖掉的词还是下一个句子，这种被预测的对象在原始语料中天然存在。当预训练结束之后，就可以基于预训练模型在其他任务上微调，如文本分类、序列标注、阅读理解等。例如，在文本分类任务中，输入的句子经过 BERT 进行编码后，将模型最后一层的第一个节点作为句子的向量表示，后边接一个 Softmax 层即可完成分类任务。在序列标注任务中，由于 BERT 采用了 Transformer 的结构，所以能够很好地融合上下文的信息。与 BI-LSTM-CRF 类似，将 BERT 的词向量与 CRF 层拼接可以完成序列标注的任务。因此，如果相比于训练时间和推理时延更在乎模型的精度，则在很多自然语言任务中都应该考虑基于无监督预训练和后期微调的方式；否则，可以适当考虑前面所提到的几种经典算法。

其他自然语言处理任务还包括文本摘要、信息检索和问答系统等。文本摘要通过识别文本内容将其缩减为简明而精确的摘要；信息检索通过文本匹配、知识关联等技术搜索出最合适的信息，常见的应用是搜索引擎；问答系统自动将最优答案匹配到输入的问题，也会用到一部分信息检索相关的技术。可以预见的是，未来更多的自然语言处理任务都可能会依赖无监督预训练。

5.1.3 ModelArts 预置算法选择

ModelArts 预置算法是指 ModelArts 平台自带的算法，仅需提供数据即可自动训练。在采用预置算法训练之前，开发者仅需要按照规范准备好数据集，无须关心具体的训练代码及训练启动后镜像容器的上传、下载等其他工作，预置算法会自动将训练好的模型和 TensorBoard 日志文件上传到开发者定的 OBS 中供查看。

预置算法的性能和精度均经过专业调优，能给开发者提供很快的训练速度和很高的训练精度。对于不熟悉算法原理的人，或者不希望调优而希望开箱即用的人，可以从 AI 市场订阅预置算法，并启动训练。预置算法在性能和精度方面，有以下主要特点。

（1）对于一些相对成熟的任务，AI 市场上预置了很多华为自研的高精度算法（如 CAKD-EfficientNet、DeepFM、NEZHA 等）及其预训练模型。

（2）预置算法在精度方面，预置了很多调优技巧和训练策略，如数据增强、数据平衡、标签平滑、SyncBN、蒸馏、增量训练等。

（3）预置算法结合软硬件优化，采用了多种技术手段实现训练加速（具体在第 6 章介绍）。

(4) 预置算法支持高阶能力(如弹性训练等,具体在第 6 章介绍),当训练资源丰富时可达 10 倍以上加速能力,并提供极致性价比。

预置算法在易用性方面,有以下主要特点。

(1) 自动分布式,只需要选择不同的规格和节点数,就可以自动运行单机单卡、单机多卡、多机多卡模式,且多节点加速比接近线性。

(2) 自动设备切换:只需要改动配置,无须修改代码即可将算法运行在其他设备(如 Ascend)上。

(3) 支持多种数据格式读取。

(4) 支持多种模型格式的同时导出。

(5) 输出模型一键部署推理服务,无须额外开发推理代码。

总体而言,ModelArts 预置算法的性能比开源版本提升了 30%～100%,精度比开源版本提升了 0.5%～6%。ModelArts 预置算法提供了图像分类、目标检测、图像分割、声音分类、文本分类、推荐、时序预测、强化学习等几大方向的经典算法,下面将依次进行介绍。

(1) 图像分类算法。包括 ResNet 系列、Inception 系列、MobileNet 系列、EfficientNet 系列等,且部分算法支持使用 Ascend-910 训练设备和 Ascend-310 推理设备,部分算法还支持华为自研深度学习引擎 MindSpore。在图像分类算法中,resnet_v1_50 使用了大量的优化方案。在训练性能方面,当批大小(Batch Size)为 256 时,该模型在 V100 卡上训练时每秒处理的图像数量为 1220 张,在 P4 卡上处理单张图像的时间为 11ms。训练后的模型还可以自动转换成 OM 格式,并在 Ascend-310 上部署推理服务。Ascend-310 处理单张图像只需要 3.2ms。

(2) 目标检测算法。包括 Faster R-CNN 系列、SSD 系列(包括 RetinaNet)、YOLO 系列(YOLOv3、YOLOv4 及不同的 Backbone)、EfficientDet(以 EfficientNet 为 Backbone 的检测网络)。部分算法也支持 Ascend-910 训练设备、Ascend-310 推理设备、MindSpore 引擎。

(3) 图像分割算法。当前支持 DeepLab 系列和 UNet。

(4) 声音分类算法。当前支持华为自研的声音分类算法 Sound-DNN,可使用 Ascend-910 设备训练。

(5) 文本分类算法。当前支持以 BERT、NEZHA(华为自研的自然语言预训练算法)为基础的中文文本分类算法。

(6) 推荐算法。当前支持华为自研的深度因式分解机 DeepFM,可使用 Ascend-

910 设备训练。DeepFM 模型结合了广度和深度模型的优点，联合训练 FM 模型和 DNN 模型，可同时学习低阶特征组合和高阶特征组合，从而能够学习各阶特征之间的组合关系。

(7) 时序预测算法。支持基于经典机器学习和深度学习两大类主流时序预测算法，通过参数化的配置，可以选择不同的算法，如 ARIMA、LSTM 等，并输出可视化的预测结果。

(8) 强化学习算法。支持多种主流的强化学习算法，如 DQN、PPO 等。针对强化学习环境接入困难的特点，提供了大量预置环境，如常用的 ClassicControl、Atari 等，均可以零代码调用，同时也提供连接自定义环境和自动训练的能力。

5.2 算法开发

当根据数据准备情况、业务要求、已有技术能力等各方面因素综合判断并选择好算法之后，开发者如果没有在 AI 市场找到匹配的预置算法，那就只能自行开发算法代码了。

5.2.1 开发语言

对于人工智能应用开发者而言，如果要做算法开发，那么 Python 编程就是一项必备技能。Kaggle 之前对于机器学习、数据科学领域内的开发语言现状进行了调查与分析，结论是：Python 毫无疑问是该领域最常用的语言。63.1%的受访者选择 Python 作为其主要数据探索工具，24%的受访者依然认为 R 是当前数据分析场景中最有效率的工具，两者几乎占到了 90%。

Python 对于中小型的数据分析及模型构建工程比较适合，生态工具非常丰富，特别适合初创团队。R 在偏研究类的数据分析及图表化展示方面比较有优势。Scala 和 Java 具备非常成熟的工程化套件，适合大型工程类开发，而在数据分析、机器学习领域的工具较少。Julia 提供针对数据概念更为友好的语法、并行编程执行方式和运行速度。正如第 4 章所述，工业界的数据科学家们在工作中无论使用哪种编程语言，都要面临巨大的数据准备工作，很多时候也需要编写数据处理和分析的代码。因此，优秀

的开发语言要能够覆盖项目开发全流程的各个环节，而不只是机器学习算法本身的开发。

Python 诞生于 20 世纪 80 年代，近期的流行主要得益于机器学习、深度学习及数学统计等应用的兴起。Python 在开发效率及社交化传播上有明显的优势。

(1) Python 语言的语法较为简单、易于理解。现在很多开源的数据分析、算法库，都直接基于 Python 开发，或者基于其他语言(如 C/C++)开发并优先提供 Python 的 API，算法工程师借助 Python 的生态能够完成很多复杂的工作。R、Scala 也能够满足数据分析与计算的诉求，但是在真实的生产化场景中，Python 工程化的工具有更多选择，能更好适应生产化场景诉求，因此也更加流行。

(2) 现在有很多优秀的开发工具支持 Python 开发，如 IDE(Integrated Development Environment，集成开发环境)、Jupyter Notebook 等著名的交互式编程环境。这些开发工具和云服务结合非常紧密，现在的主流云厂商基本提供了云上的 Jupyter Notebook，用户在使用时不仅省掉了配置的开销，而且能够直接利用云上的人工智能计算资源，从而提高了生产效率。Jupyter Notebook 在设计之初就考虑到了交互式探索、协作分享等关键要素，因此很多 Notebook 可以方便地进行分享和传播，让他人很方便地学习和复现之前的研究工作，这对 Python 的流行也起到了非常大的帮助。

5.2.2　开发库

每种类型的人工智能算法都有各自特点，难以用统一的库来完成所有人工智能算法的开发。下面将主要介绍目前常用的一些人工智能算法开发库。

1. 机器学习和深度学习开发库

TensorFlow 是由 Google 开发的人工智能计算框架，基于内部的 DistBelief 深度学习框架改进而来，并于 2015 年开源，是当前最受欢迎和最广泛采用的深度学习框架之一。TensorFlow 采用的基本数据存储单元称为张量(Tensor)，不同的张量通过一系列计算组成了一张图(Graph)，就形成了对机器学习算法的抽象。

起初，TensorFlow 复杂的接口和图定义及图运行分离的工作模式对于新手并不十分友好。后来 TensorFlow 社区提供了一些高层的接口，如早期的 Slim 框架预置了很多分类算法，并且提供了一套多卡和分布式的接口。另外，Tensorflow 社区中有非常丰富的算法库，如目标检测算法库等，TensorFlow 庞大的社区是 TensorFlow 的一

大亮点。后来 TensorFlow 还提供了 Estimator 接口，在 Estiamtor 中提供了简单易用的 distributed strategy 接口来支持多卡和分布式训练。TensorFlow 最初的定位不只是一个计算引擎，而更是一个面向机器学习领域的编程语言，但对开发者的冲击过大，不够易用。后来 TensorFlow 也意识到了问题所在，开始发布 TensorFlow 2.x 版本，去掉了以往先定义再执行的编程方式，提供更易用的命令式编程接口，便于开发和调试。

PyTorch 是当前在科研学术领域越来越流行的一种深度学习计算引擎，由 Facebook 团队开发并且于 2017 年开源。PyTorch 的易用性让研究者可以快速实现和验证自己的想法，PyTorch 同时提供了简单易用的多卡和分布式训练接口 DataParallel 和 DistributedDataParallel。PyTorch 的梯度通信直接使用底层 NCCL 或者 Gloo 接口，让分布式通信性能也得到保障。PyTorch 官方和社区同时也提供了大量的算法库，如 Torchvision 的 Models 库、OpenMMlab 的 MMDetection 库及 Facebook 的 Detectron 库。在易用性方面，PyTorch 略胜 TensorFlow 一筹，但是在稳定性、全流程完整性方面还是 TensorFlow 做得更好。

MindSpore 是华为自研的深度学习计算引擎，底层在支持高性能计算的同时，还在易用性上有很多独特之处，如支持自动分布式并行、自动微分、二阶优化等高级能力，使得算法工程师更加聚焦解决实际业务问题。ModelArts 也预置了很多 MindSpore 的算法库，并且可以一键式训练和部署。

Ray 是由美国加州大学伯克利分校开发的支持分布式任务调度的分布式机器学习框架，支持异步多节点并行。Ray 最初主要面向强化学习的使用场景，后来也被用于超参搜索。与 TensorFlow、PyTorch、MindSpore 等不同的是，Ray 更侧重资源管理和任务管理，而非机器学习模型的计算。因此，Ray 可以与 TensorFlow、PyTorch、MindSpore 等联合使用。同时，Ray 也有自带的强化学习库 RLlib。

Scikit-learn 是基于 Python 语言的机器学习算法库，提供简单高效的数据挖掘和数据分析工具，并内置了各种常用的监督和无监督机器学习算法。

XGBoost 是专注 Boosting 类算法的机器学习算法库，因其优秀的设计和高效的训练速度而获得广泛的关注，在很多 AI 算法竞赛中得到非常广泛的应用。

2. 强化学习开发库

正如深度学习中的 TensorFlow 和 PyTorch 一样，在强化学习这个大领域当中，也有许多算法库可被开发者使用。特别是 2016 年之后，各个在该领域领先的高校、科研机构，以及在人工智能领域发力的大公司，都推出了自己的强化学习库（有时候也称

为强化学习平台)。

需要明确的是,主流的强化学习库一般都可集成 TensorFlow、PyTorch 等深度学习计算库。这些库提供模型的前后向计算等能力,而强化学习库专注提供算法、分布式调度等其他方面的能力。

Baselines 是 OpenAI 公司于 2017 年启动的一个开源项目,旨在为科研人员提供高质量的强化学习算法。作为强化学习在商业领域的先行者,OpenAI 为强化学习社区提供了很多非常有价值的开源项目,包括测试环境基准库 Gym 等。Baselines 作为算法库为强化学习的代码实现提供了非常有价值的参考。但由于架构层面的先天不足,对于分布式框架的支持并不好,因此也没有包含当前的高性能分布式强化学习训练算法。此外,Baselines 中不同的算法缺乏统一标准,导致编码风格迥异,易读性受到了一定的影响。

Coach 是 Intel 公司推出的一款开源强化学习项目,相比于 Baselines,Coach 在模块化和组件化的层面有很大的提升,将算法、模型、探索等模块都做了解耦,并且针对性地开发了分布式组件,还对云服务做了适配。在预置的仿真环境方面,除了基础的 Gym、Atari 等,Coach 还支持多个高阶的仿真环境,更加方便用户使用。

RLlib 是 Ray 框架上层的一个强化学习算法库。上文提到的 Ray 能够非常灵活地组建计算集群并进行多进程的任务调度,该能力非常适合进行强化学习的大规模分布式训练,对于 IMPALA 等算法非常友好。同时,RLlib 也提供了很多高级 API,允许用户进行模型、算法流程、环境等幅度较大的自定义。RLlib 的短板主要在于其极致工程化、模块化所带来的代码冗长。如基本的 DQN Agent,由于需要兼容 DQN 的多个变体,而在算法代码中加入了大量配置项和逻辑判断,造成算法可读性和修改性不佳,不适合在科研或者算法研究中使用。

刑天是华为诺亚人工智能系统工程实验室自研的高性能分布式强化学习平台,针对强化学习 Learner-Worker 异构的架构,为用户提供了一个简单高效的分布式框架,方便用户在上面实现自己分布式算法,其支持多种异构的人工智能计算设备,分布式计算性能高。

3. 运筹优化主流求解器

作为运筹优化在产业落地的核心组件,运筹优化求解器已经经过了几十年的发展,经过了充分的竞争并且相对成熟。针对在生产中实际需求的问题类型,运筹优化求解器分为支持整数规划(LP)求解器和混合整数规划(MILP)求解器两类,针对来源

分为商用求解器和开源求解器两大类。

三大商用求解器 Cplex、Gurobi、FICO Xpress 均来自美国公司，经过 10 年以上的开发，对于线性规划、混合整数规划及部分非线性规划均有完整的支持，并且支持 C、C++、C#、Java、Python 等多种主流代码编写建模。其中，Gurobi 从 2018 年被权威基准测试评为最快的求解器。

开源求解器则主要包括 SCIP、MIPCL 等。其中，SCIP 由德国 ZIB 研究所学术团队开发，在学术研究中使用最为广泛，求解速度也很快；另外，MIPCL 也是最快的开源求解器之一，同时支持多线程并行，但受制于开源属性，其团队规模一般都不大，甚至由单人开发。开源求解器在大规模问题的求解能力、稳定性、功能全面性、架构统一性等层面还需要加强。

5.2.3 交互式开发环境

在谈到人工智能开发环境时，我们指的已经不仅仅是 IDE 及对应的开发库，还包括硬件资源的配置。人工智能开发可以分为两大类：①研究探索类，以快速验证、实现原型、教学等为目标，并且需要对于实验进行解释和传播；②生产工程类，以软件工程化交付为目标，进行软件项目管理，需要较强的工程管理能力与问题调优、定位工具。这两大类开发者的目标差别非常明显，因此对于开发环境的诉求完全不同，研究探索类对于开发环境的诉求是：轻量、快速，能方便给别人解释和重现。通常来说，一个人工智能算法研究人员，可能需要在 MATLAB 上进行原型数据开发，然后在 IDE 中用高级语言进行算法代码开发与调试，在数据分析工具上进行数据探索，并且在报表工具上进行数据可视化图表开发，最终将这些图表放到论文或者胶片中进行展示。在一段时间后，又希望能够方便地更新之前工作流中的一些内容，然后重现实验，最终生成新的数据分析报告。而生产工程类则需要践行软件工程生命周期的每个步骤，最终产出生产级的代码，这其中包括代码的静态检查、代码版本管理、集成测试等。

在人工智能工程生产化过程中，以现代化的软件工程方法论为基础，通用的 IDE（如 VSCode、PyCharm 等）配搭对应的开发插件，再配搭云上人工智能计算资源，会是比较合适的选择。而在人工智能研究探索场景中，Jupyter Notebook 则能够在其各个阶段满足开发者诉求并覆盖这些关键点，另外，在云化场景下借助 Jupyter Notebook 云服务则能够进一步提升开发效率。

Jupyter 起始于 IPython 项目，IPython 最初是专注于 Python 的项目，但随着项目

发展壮大，已经不仅仅局限于 Python 这一种编程语言了。按照 Jupyter 创始人的想法，最初的目标是做一个能直接支持 Julia(Ju)、Python(Py)及 R 三种科学运算语言的交互式计算工具平台，所以将平台命名为 Jupyter(Ju-Py-te-R)。现在，Jupyter 已经成为一个几乎支持所有语言，能够把代码、计算输出、解释文档、多媒体资源整合在一起的多功能科学运算平台。

这里需要提到的另外一个概念就是“文学编程”。文学编程是一种由 Donald Knuth 提出的编程范式。这种范式强调用自然语言来解释程序逻辑。简单来说，文学编程的读者不是机器，而是人。从写出让机器读懂的代码，过渡到向人解说如何让机器实现开发者的想法，其中除了代码，更多的是叙述性的文字、图表等内容。文学编程中间可以穿插宏片段和传统的源代码，从中可以生成可编译的源代码。

如果我们将人工智能研究的全生命周期分解为个人探索、协作与分享、生产化运行环境、发表与教学等几个阶段，那么 Jupyter Notebook 可以很好地满足这些阶段的需求。ModelArts 内置的 Jupyter Notebook 界面如图 5-5 所示。

1. Jupyter Notebook 的优点

1）贯串整个人工智能开发和探索的生命周期

在实际的软件和算法开发过程中，上下文切换占用了大量的时间，特别是工具间的切换也是影响效率的重要因素。而 Jupyter Notebook 将所有和软件编写有关的资源放在一起，当开发者打开了 Jupyter Notebook 时，就可以看到相应的文档、图表、视频、代码及解释说明。只要看一个文件，就可以获得项目的所有信息。

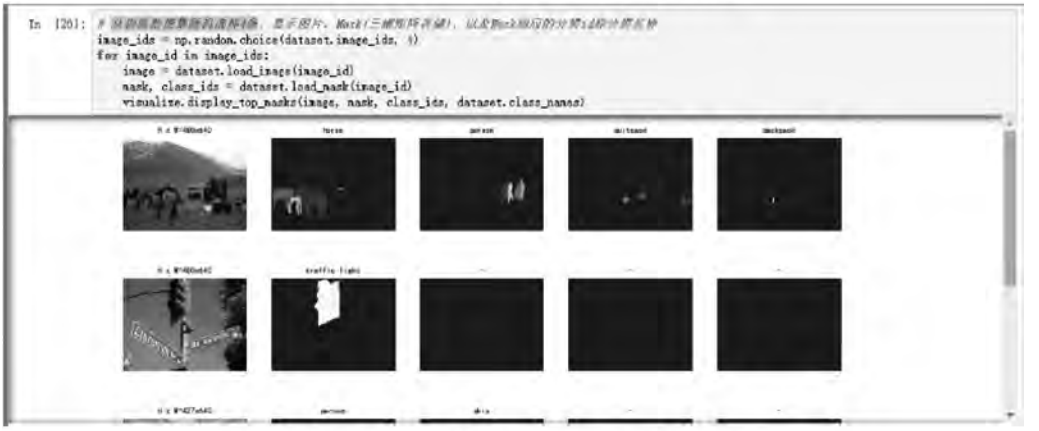
2）交互式探索

Jupyter Notebook 不仅可以输出图像、视频、数学公式，甚至还可以呈现一些互动的可视化内容，如可以缩放的地图和可以旋转的三维模型，这就需要交互式插件来支持。针对大型的数据集或者复杂的仿真算法，Jupyter Notebook 本身也支持将其运行在远端集群(如云上资源)。

3）结果分享与快速重现

Jupyter Notebook 支持以网页的形式分享，GitHub 等开发者社区天然支持 Jupyter Notebook 展示，也可以通过 nbviewer 分享 Jupyter Notebook 文档。Jupyter Notebook 还支持导出成 HTML、Markdown、PDF 等多种格式的文档。开发者不仅可以在 Jupyter Notebook 上完成图表等可视化展示，还可以将 Jupyter Notebook 附在论文或者报告中，便于对外交流。

(a) 某文本情感分析的Jupyter Notebook案例简介界面



(b) 某图像分割的Jupyter Notebook案例打开后的内容预览界面

图 5-5　ModelArts 内置的 Jupyter Notebook 界面

4）可扩展与可定制

好的开发环境一定是可扩展和可定制的。Jupyter Notebook 上通过各种各样的插件和 Magic Command 允许开发者定制出自己的开发环境。Jupyter Notebook 的升级版——JupyterLab 在插件化可扩展方面的能力得到了进一步提升。

5）良好的生态

从 2017 年开始，很多顶级的计算机课程开始完全使用 Jupyter Notebook 作为工具，比如李飞飞的“计算机视觉与神经网络”，一些专业领域的人工智能应用开发教程

也都采用 Jupyter Notebook。

2. Jupyter Notebook 的缺点

首先，Jupyter Notebook 不是一个真正意义上的集成开发环境。如果开发者追求的是产品化代码开发，如代码格式、依赖管理、产品打包、单元测试等功能，那么 Jupyter Notebook 当前有一些插件可以做，但是相比重型 IDE，它的功能还是比较弱。在 Jupyter Notebook 中，目前都是基于单向顺序的方式实现代码单元执行，而如果要非顺序执行，就会产生不可预期的效果。其次，在代码版本的管理方面，由于 Jupyter Notebook 内容结构通过 JSON 的方式进行组织，所以一旦有冲突时，很难进行冲突处理与合并。最后，Jypyter Notbeook 对于分布式调测、重型异步任务的支持不够友好。Jupyter Notebook 定义为研究类调试环境，一方面，对于分布式的训练可以通过单机多进程的方式进行模拟，另一方面，Jupyter Notebook 的架构并不适合运行非常大规模的训练作业。对于较大规模的人工智能应用产品化开发诉求，还是需要在 IDE 中进行工程化代码开发，并配搭测试逻辑，将任务部署在集群中进行批量运行。

3. ModelArts 云上开发环境

ModelArts 云上开发环境服务，让人工智能应用开发者、数据科学家能够充分利用云端的计算资源快速获得可以进行人工智能探索的 Jupyter Notebook 实例。针对研究探索类场景，ModelArts 还提供 JupyterLab，方便与 Github 等知识社区对接，方便完成论文复现、可视化分析等操作；针对生产工程类开发场景，ModelArts 除了提供开发和部署平台，在 Notebook 实例上还提供 WebIDE 的能力以支持生产级软件开发。

ModelArts 云上开发环境主要有以下特点。

1）即开即用

开发者在 ModelArts 页面中创建 Notebook 实例，根据自己的诉求配搭对应的计算和存储资源，一个实时可用的 Jupyter Notebook 实例在几秒钟就可以创建完成，开发者可以直接通过本地浏览器进行访问，并按需使用，随时可以关闭。并且基础 CPU 版本是免费的，能够满足大多数开发者编写代码和调试的诉求。另外，ModelArts 还提供了免费 AI 算力（包含 GPU、Ascend 设备）供开发者进行试用和体验。Notebook 中针对 GPU 驱动及 CUDA 开发库都进行了预先的配置和测试，保证在使用时顺畅、高效。Notebook 自带免费的云上数据存储，开发者不用担心数据丢失的问题。

2）预置能力

ModelArts 内置的 Jupyter Notebook 为了提升开发效率，不仅完全继承了开源 Jupyter Notebook 的基础能力，而且在此基础上预置了多种 AI 计算引擎或开发库，并且安装了常用的工具与依赖库，包括 TensorFlow、PyTorch、MindSpore 在内的预置 AI 引擎或开发库在 Jupyter Notebook 中以多 Kernel 的方式呈现。开发者可以根据自己的需要进行引擎或库之间的切换，甚至在一个 Jupyter Notebook 实例中同时使用多种引擎或库。每个引擎或库安装在一个独立的 Conda 环境下，互相不受影响。

ModelArts 还在 Jupyter Notebook 中内置了 ModelArts Python SDK，实现常用的人工智能应用开发流程中各个环节的封装。开发者不仅可以通过 SDK 进行远程作业（如训练等）的提交和管理，而且还可以在 SDK 中进行本地推理部署和调测、分布式训练模拟等。

3）智能问答

在 ModelArts 开发环境中，平台对开发者常见的开发问题进行了积累与总结，通过交互的方式进行问题提示和处理，并在此基础上引入了问答机器人及 ModelArts 开发者社区，开发者可以对常见问题进行在线搜索。

4）社区与分享

AI 市场预置丰富、精致的学习案例，并且支持 Github 上优质的学习和实践资源的导入。

5）安全可信

ModelArts 所提供的开发环境，完全构建在华为云基础设施平台上，在网络、数据存储、计算安全等方面通过了主流合规性认证，并且针对所用到的操作系统、软件等都进行了安全加固以确保使用的安全可信。

5.2.4 ModelArts 云上云下协同开发

ModelArts 提供了 PyCharm 插件、Python SDK，可以方便地实现云上和云下协同开发。

1. ModelArts PyCharm 插件

虽然 Jupyter Notebook 在交互式探索方面有明显的优势，但是对于生产工程类的开发，建议使用传统 IDE，如 PyCharm、VSCode 等工具，能更高效地完成代码的开发与调试。很多开发者也习惯于本地安装 PyCharm 进行 Python 算法开发，能够方便地

进行断点调试、静态代码扫描、格式化、单元测试等，这些在 Jupyter Notebook 中都是比较难做到的。然而，本地开发的劣势是本地计算资源不足，导致开发环境难以共享，每个开发人员都需要自己的环境，多人协同开发时经常会有依赖的冲突、版本不一致等问题，效率低下。

针对这类使用本地 IDE 的开发用户，ModelArts 提供了一个 PyCharm 插件，开发者可以将本地调试完成的代码，通过插件直接上传到 ModelArts 云上集群，通过云端丰富的计算资源来完成模型的训练和评估等操作。该插件协助用户自动完成代码上传、训练作业提交及日志获取，用户只需要专注本地的代码开发即可。目前，该插件支持 Windows、Linux 或 Mac 版本的 PyCharm，具体功能如表 5-1 所示。

表 5-1　ModelArts PyCharm 插件支持的功能

支持的功能	说　　明	对应操作指导
模型训练	支持将本地开发的代码快速提交至 ModelArts 并自动创建训练作业，在训练作业运行期间获取训练日志并展示到本地	提交训练作业 查看训练作业详情 启动或停止训练作业 查看训练日志 提交不同名称的训练作业
部署上线	支持将训练好的模型快速部署上线为在线服务	部署上线
OBS 文件操作	上传本地文件或文件夹至 OBS，从 OBS 下载文件或文件夹到本地	OBS 文件上传与下载

2. ModelArts Python SDK

ModelArts 提供了一套完整的 Python SDK，通过 SDK 可以直接对接 ModelArts 的功能完成集成。通过 ModelArts SDK，用户可以在任意 Python 开发环境中通过代码调用的方式与云上 ModelArts 进行交互，调用云端能力和计算资源。该 SDK 通过提供 Python 语义抽象，更方便地让用户在 ModelArts 中进行训练、部署、模型管理、流程编排。

ModelArts Notebook 提供了丰富的 SDK 使用样例和资料，让开发者快速掌握 SDK 的使用方式。下面是用 SDK 创建训练作业的示例代码，这段代码在 ModelArts Notebook 中可以直接被执行。

```
from modelarts.session import Session
from modelarts.estimator import Estimator
session = Session()
```

```
estimator = Estimator(
    modelarts_session = session,
    framework_type = 'PyTorch',                              #AI 引擎名称
    framework_version = 'PyTorch - 1.0.0 - python3.6',   #AI 引擎版本
    code_dir = '/bucket/src/',                               #训练脚本目录
    boot_file = '/bucket/src/pytorch_sentiment.py',      #训练启动脚本目录
    log_url = '/bucket/log/',                                #训练日志目录
    hyperparameters = [
                {"label":"classes",
                 "value": "10"},
                {"label":"lr",
                 "value": "0.001"}
                ],
    output_path = '/bucket/output/',                         #训练输出目录
    train_instance_type = 'modelarts.vm.gpu.p100',       #训练环境规格
    train_instance_count = 1,                                #训练节点个数
    job_description = 'pytorch - sentiment with SDK')    #训练作业描述
job_instance = estimator.fit(inputs = '/bucket/data/train/', wait = False, job_name = 'my_
training_job')
```

如果在本地执行，需要在 Session 对象初始化时配置华为云用户鉴权信息，通过定义 Estimator 对象将训练的配置项进行描述，最终通过 fit 方法将训练作业提交到 ModelArts 远程集群中执行。详细的参数信息可以在 ModelArts 的官方文档中找到，这里不再赘述。ModelArts Python SDK 封装了基本的云端资源请求操作，如权限校验、资源申请、数据读写等，以减少开发者工作量。