easynlp&easytext 源码阅读打报告

甘铠荣 2021K8009922008

一、 Easynlp & Easytext 简介

Easynlp 是由阿里巴巴的 PAI 团队推出的一款中文开源 NLP 算法框架，它的前身是 EasyTexMiner。

易用且兼容开源：支持常见的中文 NLP 数据与模型，提供用户自定义模块 AppZoo 以及 ModelZoo，兼容 EasyTransfer 模型并且自带分布式训练框架。

大模型小样本落地技术：集成了多种小样本学习算法，提出了 Contrastive Prompt Tuning 方案。

大模型知识蒸馏技术： 提供知识蒸馏功能帮助蒸馏大模型从而得到高效的小模型来满足线上部署服务的需求；提供 MetaKD 算法，支持元知识蒸馏；支持数据增强，通过预训练模型来增强目标领域的数据。

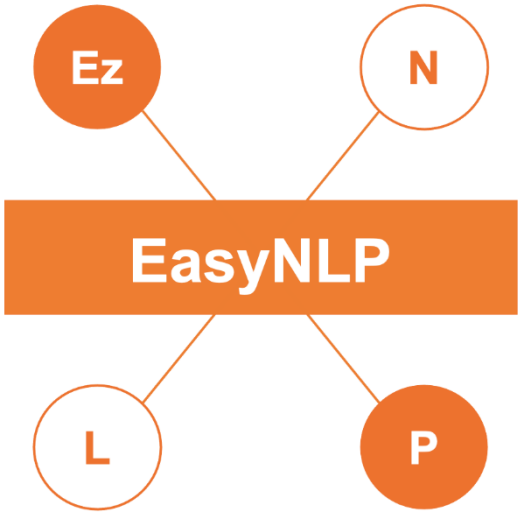

图 1 Easynlp

Easytext 是由百度潘老师开发的一款开源自然语言模型训练框架。最后一次更新是在两年前。包含了命名实体识别，事件识别以及属级情感分析的相关模型以及对应的配置文件。很好的体现了面向对象编程的思想。

二、 Easynlp & Easytext 训练模块分析

在 Easynlp 中，若要训练一个 BERT 模型，需要用到：

ClassficationDataset 数据加载与预处理模块：用于对数据集进行处理，其属性包含了数据集的信息，方法用于处理和转换文本数据。

Application 训练框架模块：这是训练框架的基类模块。

Optimizer 优化器模块：用于更新和管理模型参数。

Trainer 训练控制模块：这个类用于进行训练，其属性包含了训练模型时用到的参数，方法包括设置模型和优化器、从检查点恢复模型训练、设置训练数据加载器等模型训练常用

操作。

Evaluator 性能评估模块：用于评估模型的性能。
Exporter 参数转换模块：用于在不同的深度学习框架之间迁移模型。
Losses 损失函数模块：用于定义不同的损失函数。

在 Easytext 中，需要用到：
ConfigFactory 是一个工厂类，用于创建和管理模型的配置。
MetriccTracker 模块主要用于跟踪每一个 epoch 的 metric 记录，保存 best 等指标。
Model 是一个包含预定义结构和参数的实体，用于从输入的数据中进行学习和预测。
Loss 用于评估预测结果和真是值之间的差异。
ModelMetricAdapter 适配器，使 model 和 metric 能一起工作。
Optimizerfactory 是一个工厂类，用于创建和配置优化器。
LRSchedulerfactory 是一个工厂类，用于创建和配置学习率调度器。
Trainer 用于执行训练，评估等任务。

三、 工作流程与类间关系分析

```
initialize_easynlp()
args = get_args()
user_defined_parameters = parse_user_defined_parameters(args.user_defined_parameters)
pretrained_model_name_or_path = get_pretrain_model_path(user_defined_parameters.get('pretrain_model_name_or_path', None))

train_dataset = ClassificationDataset(
    pretrained_model_name_or_path=pretrained_model_name_or_path,
    data_file=args.tables.split(",")[0],
    max_seq_length=args.sequence_length,
    input_schema=args.input_schema,
    first_sequence=args.first_sequence,
    second_sequence=args.second_sequence,
    label_name=args.label_name,
    label_enumerate_values=args.label_enumerate_values,
    user_defined_parameters=user_defined_parameters,
    is_training=True)

valid_dataset = ClassificationDataset(
    pretrained_model_name_or_path=pretrained_model_name_or_path,
    data_file=args.tables.split(",")[-1],
    max_seq_length=args.sequence_length,
    input_schema=args.input_schema,
    first_sequence=args.first_sequence,
    second_sequence=args.second_sequence,
    label_name=args.label_name,
    label_enumerate_values=args.label_enumerate_values,
    user_defined_parameters=user_defined_parameters,
    is_training=False)

model = get_application_model(app_name=args.app_name,
    pretrained_model_name_or_path=pretrained_model_name_or_path,
    num_labels=len(valid_dataset.label_enumerate_values),
    user_defined_parameters=user_defined_parameters)

trainer = Trainer(model=model, train_dataset=train_dataset,user_defined_parameters=user_defined_parameters,
    evaluator=get_application_evaluator(app_name=args.app_name, valid_dataset=valid_dataset,user_defined_parameters=user_define
    eval_batch_size=args.micro_batch_size))

trainer.train()
```

图 2 BERT 模型训练代码

在 easynlp 中，训练一个 BERT 模型需要：

初始化 Easynlp 库——解析用户定义的参数——获取预训练模型的路径——创建训练和验证数据集——获取应用模型——创建训练器实例（包含了模型、训练集以及用户定义的参数和评估器）——调用 train 方法进行训练。

Easytext 也需要先解析用户自定义参数，构建数据集，然后实例化模型和训练器再进行训练。

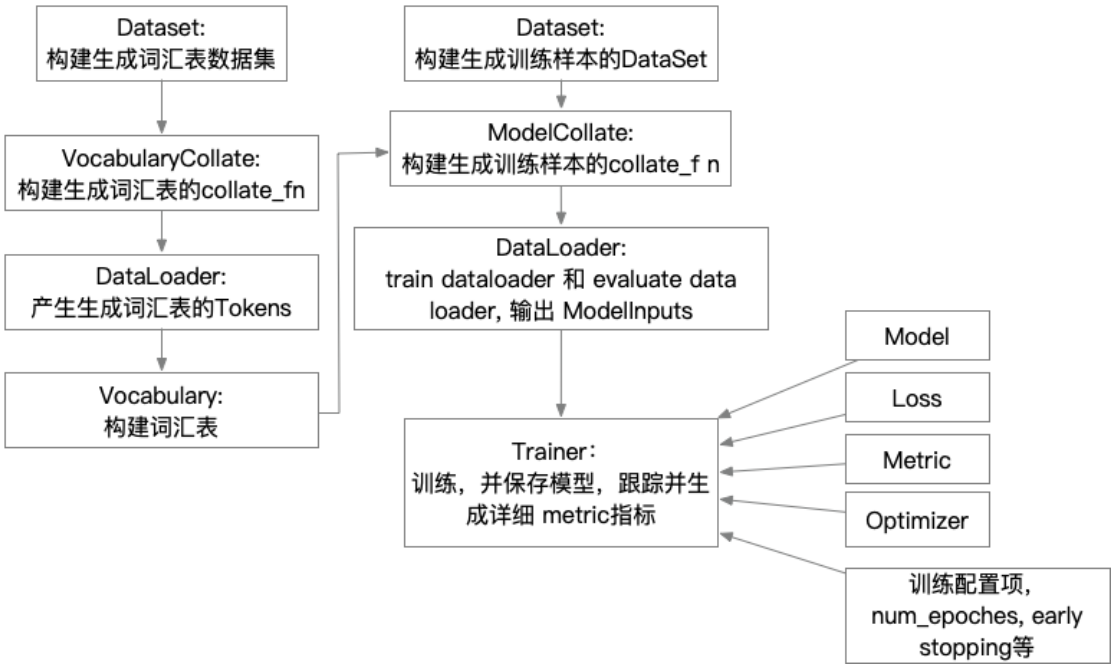

图 3 easytext 工作流程

Easynlp 类间关系：

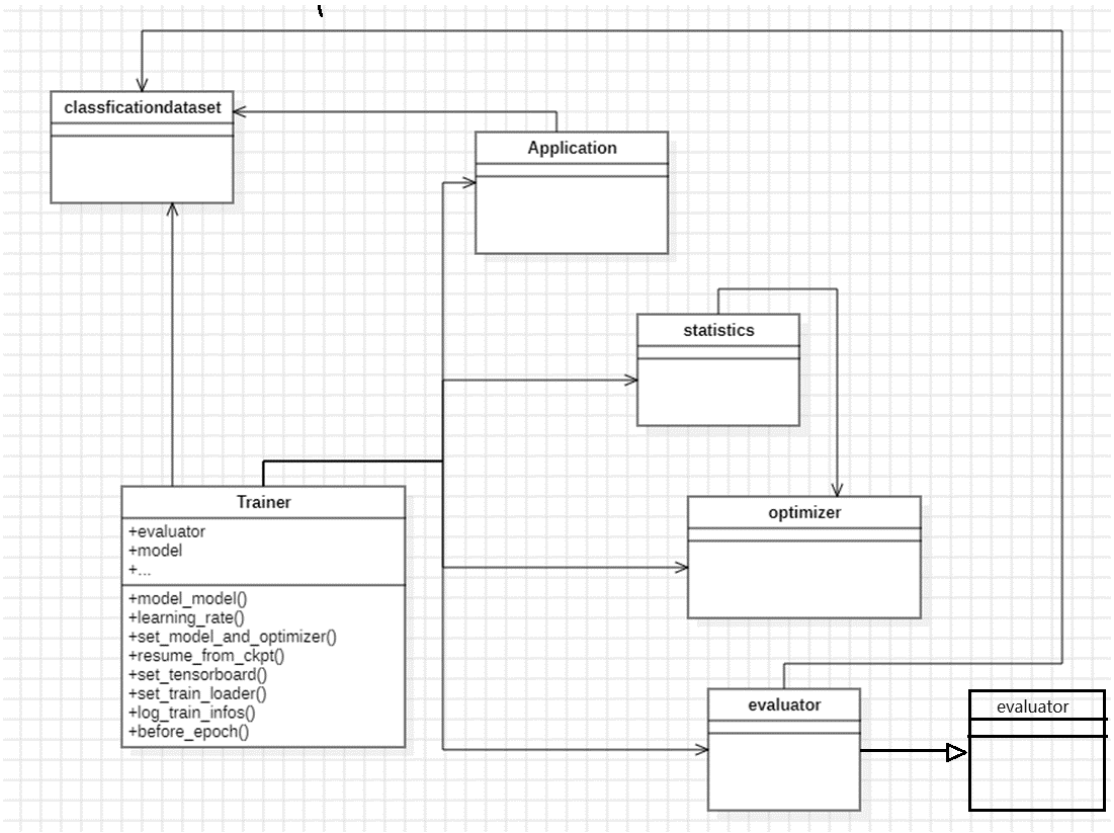

图 4 easynlp 类间关系

模型的建立需要验证集中的数据；

评估器也需要验证集中的数据进行评估；
模型的训练需要训练集的数据。
因此这三类都对数据处理类有依赖关系。
优化器需要使用到训练过程中 statistics
获取到的数据，因此对其有依赖关系。
训练类在训练过程中需要通过调用各种
模块完成训练过程，因此它对其它的模块都有依赖关系
其中的 Application，classificationdataset 和 evaluator 类都是来源于不同模型的类对基类的继承，这样就实现了在不同的模型上进行训练。

Easytext 类间关系：

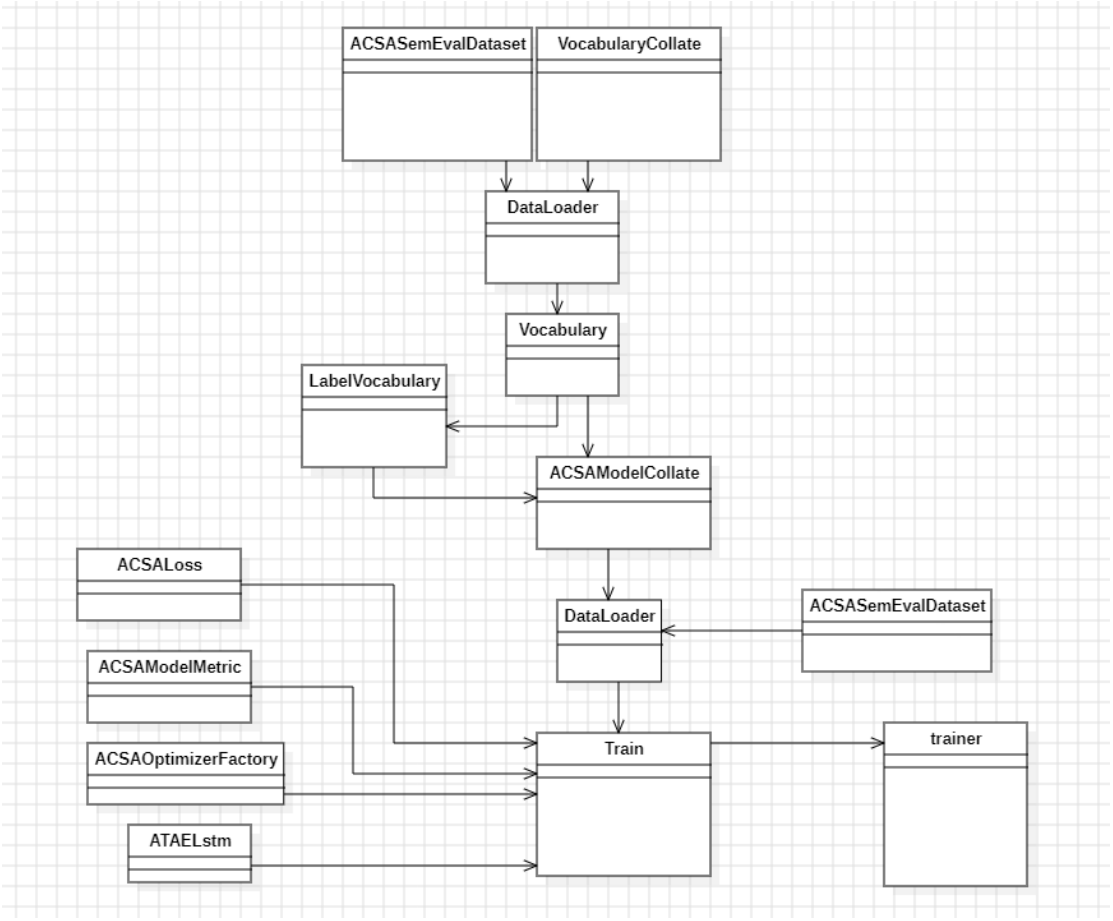

图 5 easytext 类间关系

在 easytext 里，每个模型有一个单独的训练模块，即 Train 类，它接受其它模块处理生成的字典、数据集以及各种参数，并实例化 Loss、metric 等模块最后调用共用的训练器对这些数据进行训练。

各个模型也有其对应的数据集与字典等数据，这些数据分别被字典生成模块和数据集生成模块处理后发送给 dataloader，在 train 中进行加载。

其它的功能模块如 metric，optimizer 会在 train 类中获取数据及信息并被实例化。

四、 设计模式分析

工厂模式：

在 easytext 的训练模块里 ACSAOptimizerFactory 是一个工厂类 ACSAOptimizerFactory 类的作用是根据给定的模型和配置来创建 Adam 优化器对象。这使得创建优化器的过程更加模块化且可扩展，可以方便添加新的优化器类型和配置。

```python
class ACSAOptimizerFactory(OptimizerFactory):

    def __init__(self, config: Dict):
        self.config = config

    def create(self, model: Model) -> "Optimizer":

        return Adam(params=model.parameters(),
                    lr=0.01)
```

图 6 easytext 的优化器工厂

同样的，easynlp 中也使用一个工厂类来实例化 pytorch 中的 adam 优化器。

同时，在 easynlp 中使用 appzoo 中的 api 去实例化不同模型的结构，比如在其中的 get_application_model 函数中它就可以根据 app_name 以及用户自定义参数来创建模型。

在 easytext 中还有一个功能强大的工厂类，component_factory。这个类可以根据配置信息创建和管理 Component 对象。它可以通过已经创建好的对象来获取并创建对象，通过注册表信息获取并创建对象，通过字典来创建对象……

```python
class ComponentFactory:
    """
    Component Factory
    """

    def __init__(self, is_training: bool):
        """
        初始化
        :param is_training: True: training 状态创建 component; False: 非training状态创建 component
        """
        self._is_training = is_training
        self._registry = Registry()
```

图 7 easytext 的 component 工厂

装饰器模式：

```python
class ComponentRegister:
    """
    Component 注册器，也是类装饰器
    """

    @classmethod
    def register(cls, name_space: str, typename: str = None, is_allowed_exist: bool = False) -> T:
        """
        用作在类或函数上的装饰器
        :param typename: 注册的类或者函数的类型名称，如果为 None 或者 ""，那么，将会默认使用类或者函数的名字作为名字，
                         在配置文件中直接使用类名字或者函数名字即可。
        :param name_space: 注册的类或者函数的 name space
        :param is_allowed_exist: True: 允许名字重复，那么，后面的名字会将前面的名字覆盖，正常来讲不应该出现这样的设置；
                                 False: 不允许名字重复，如果出现重复，自己定义的名字需要进行修改
        :return:
        """
```

图 8 easytext 的装饰器

在 easytext 中对象的注册器使用了装饰器模式的设计思想。它们使得我们可以在定义类的同时将类注册到注册表中，这样就可以在程序的其他地方通过注册表来获取这些类。这种

方式使得代码更加清晰，也更容易管理。

五、 总结

在 easynlp 中，包含了很多模型，将这些模型封装好以便于用户调用时十分重要的，将训练器的各种模块如评估器作为父类，各类模型的模块再去继承，最后通过一个统一的工厂将它们实例化很好的符合了面向对象的思想。这既方便了用户使用，也方便对模型种类进行扩展。

在 easytext 中使统一使用 instance 工厂和 component 工厂去创建和管理数据实例和各种模块，并且通过一个注册表管理对象的创建，这样的统一管理更有利于让开发者对程序行为进行管理，也能让用户更好地理解程序在干什么。