KDD 竞赛基准系统（Benchmark）

一、该基准系统使用 python3 进行开发，需要安装以下包：

- numpy
- sklearn
- pandas
- *pyprind: （*可选）用于显示进度条。位于 make_feature_file.py 中，用于显示抽取特征的进度，如果不需要，可以注释掉相应的行（默认不需要）

二、运行方式：

1. 下载 data 数据，解压后放到项目的根目录下

2. 修改配置文件 config.py 中的 CWD（Current Working Directory）变量的值，将其改成当前项目所在的目录，如：

```
#根据具体的系统，设置当前工作目录
CWD = "/home/username/KDD/KDD_benchmark"  # Linux 系统
CWD = "D:\KDD\KDD_Benchmark"  # Windows 系统
```

3. 进入 model_trainer 文件夹，使用以下命令运行程序：

```
python trainer.py
```

程序将对 config.py 中 TRAIN_FILE 对应的训练文件，构建训练正负样本，设计并抽取特征，构建分类器。对 TEST_FILE 变量对应的测试文件，构建测试样本，抽取特征，并使用在训练集上训练得到的模型，对测试集进行预测。（**注意：**在模型训练阶段，可以使用 valid_set 作为测试数据，也可以使用交叉验证法但注意取样的代表性。）

```
# 训练和测试文件（训练阶段有验证数据，测试阶段使用测试数据）
TRAIN_FILE = os.path.join(DATASET_PATH, "train_set", "Train.csv")
TEST_FILE = os.path.join(DATASET_PATH, "valid_set", "Valid.csv")
```

模型对测试集的预测结果文件，对应于 config.py 的 TEST_PREDICT_PATH 变量所指的文件。

```
TEST_PREDICT_PATH = os.path.join(CWD, "predict", "test.predict")
```

4. 评估脚本

Accuracy 为最终的评估标准，使用 model_trainer 文件夹下面的 evaluation.py 获取评估结果，命令如下：

```
python  evalution.py  gold_file_path  pred_file_path

例如：
python evalution.py ..\data\dataset\valid_set\Valid.gold.csv   ..\predict\test.predict
```

gold_file_path（.csv 文件）为标准答案所在的路径，pred_file_path（.predict 文件）为预测文件所在的路径。

三、目录介绍

KDD_Benchmark：基准系统目录

data：数据目录

dataset: 数据集目录

train_set: 训练集文件夹

- Train.authorIds.txt: 训练集的所有作者列表
- Train.csv：训练集

valid_set: 验证集文件夹

- Valid.authorIds.txt: 验证集的所有作者列表
- Valid.csv：验证集
- Valid.gold.csv：验证集的标准答案

test_set: 测试集文件夹（各个小组不同的测试集）

- Test.authorIds.txt: 测试集的所有作者列表
- Test.csv：测试集，如Test.01.csv 是第一小组的测试集

Author.csv: 作者数据集

coauthor.json: 共作者数据

Conference.csv: 会议数据集

Journal.csv: 期刊数据集

Paper.csv：论文数据集

PaperAuthor.csv: 论文-作者 数据集

paperIdAuthorId_to_name_and_affiliation.json：包含论文-作者对 (paperId，AuthorId) 与 名字-单位(name1##name2; aff1##aff2)的映射关系

feature: 特征文件夹

- train.feature: 存放训练数据集抽取得到的特征
- test.feature: 存放测试数据集抽取得到的特征

model: 模型文件夹

- kdd.model：训练好的分类模型

model_trainer: 训练模型

- coauthor.py: 获取共作者
- data_loader.py: 加载数据
- evalution.py: 评估脚本
- feature_functions.py: 特征函数
- make_feature_file.py: 生成特征文件
- stringDistance.py: 获取字符串距离信息
- trainer.py: 模型训练器，**主函数**

predict: 预测结果文件夹

- test.predict: 转化为 KDD 提交格式的测试结果
- test.result：模型的直接预测结果

authorIdPaperId.py: (作者，论文) 对类定义

classifier.py： 分类器，使用了策略模式。

config.py：配置文件

confusion_matrix.py： 评估脚本所使用的包

example.py： 样本类定义

feature.py：特征类定义

README.md: 说明文件

util.py: 小工具类

任务介绍

1. 目标：给定作者 ID 和论文 ID，判断该作者是否写了这篇论文。

2. 数据集描述：

1. 作者数据集: **Author.csv**。包含作者的编号（Id），名字（Name），单位（affiliation）信息。相同的作者可能在 Author.csv 数据集中出现多次，因为作者在不同会议 / 期刊上发表论文的名字可能有多个版本。例如：J. Doe, Jane Doe, 和 J. A. Doe 指的均是同一个人。此外，Affiliation 信息可能为空。

字段名称	数据类型	注释
Id	int	作者编号
Name	string	作者名称
Affiliation	string	隶属单位

2. 论文数据集: **Paper.csv**。包含论文的标题(title)，会议 / 期刊信息，关键字(keywords)。同一论文可能会通过不同的数据来源获取，因此在Paper.csv 中会存在多个副本。此外，Keyword 信息可能为空。

字段名称	数据类型	注释
Id	int	论文编号
Title	string	论文标题
Year	int	论文年份
ConferenceId	int	论文发表的会议Id
JournalId	int	论文发表的期刊Id
Keywords	string	论文关键字

3. (论文-作者)数据集: **PaperAuthor.csv**。包含 (论文 Id-作者 Id)对 的信息。该数据集包含噪声(noisy)，即存在不正确的(论文 Id-作者 Id)对，意味着 PaperAuthor.csv 包含的(论文 Id-作者 Id)对 中的作者 Id 并不一定写了该论文 Id。这是因为，作者名字存在歧义，可能存在同名的不同人或作者名字有多个版本（如上面的例子：J. Doe, Jane Doe 和 J. A. Doe 指的均是同一个人）。此外，Affiliation 信息可能为空。

字段名称	数据类型	注释
PaperId	int	论文编号

字段名称	数据类型	注释
AuthorId	int	作者编号
Name	string	作者名称
Affiliation	string	隶属单位

4. 会议和期刊数据集：**Conference.csv**, **Journal.csv**。每篇论文发表在会议或者期刊上。

字段名称	数据类型	注释
Id	int	会议 / 期刊 编号
ShortName	string	简称
Fullname	string	全称
Homepage	string	主页

5. 共同作者的信息：**coauthor.json**。该文件内容是从 PaperAuthor.csv 中抽取出来共同作者的信息，该文件的生成可以通过运行 model_trainer 下的 coauthor.py：

```
python coauthor.py
```

coauthor.json 文件的内容格式形如：

```
{"A 作者 ID": {"B1 作者 ID": 合作次数, "B2 作者 ID": 合作次数}}
```

第一层的 key 为作者的 ID，对应的 value 为共同作者信息（同样为 key-value 形式，key 为共同作者的 ID，value 为合作次数）。

目前，coauthor.json 文件给出每个作者合作频率最高的 10 个共同作者，该文件的格式为 json。可以通过修改 coauthor.py 中 get_top_k_coauthors (paper_author_path, k, to_file)方法中的 k 值来获取最高的 k 个共同作者，即 top k：

```
k = 10
get_top_k_coauthors(
    os.path.join(config.DATASET_PATH, "PaperAuthor.csv"),
    k,
    os.path.join(config.DATA_PATH, "coauthor.json"))
```

例如，获取作者 ID 为 ‘742736’ 的共同作者信息，可以通过以下代码获取，coauthor["742736"] 值对应的是 ID 为 ‘742736’ 作者的共同作者信息。u'823230': 3 表示 ID 为 ‘742736’ 的作者 与 ID 为 ‘823230’ 的作者共合作过 3 次：

```
>>> import json
>>> coauthor = json.load(open("coauthour.json"))
>>> coauthor["742736"]
{u'823230': 3, u'647433': 3, u'1691202': 3, u'891164': 3,        u'1910552': 3, u'607259': 3,
u'2182818': 7, u'1355775': 4,     u'2097154': 3, u'1108518': 3}
```

6. 论文&作者 pair 字符串信息：**paperIdAuthorId_to_name_and_affiliation.json**。该文件内容是从 Paper-Author.csv 提取的，该文件可以通过运行 model_trainer 文件夹下的 stringDistance.py 来获取：

```
python stringDistance.py
```

将 Paper-Author.csv 中相同的论文 ID 和作者 ID 对的 name 和 affiliation 合并，文件内容为 key-value 形式，key 为论文 ID 和作者 ID 对：'paperid|authorid', value 为 {"name": "name1##name2##name3", "affiliation": "aff1##aff2##aff3"}。

例如，获取 ID 为 ‘1156615’ 的论文和 ID 为 ‘2085584’ 的作者 name 和 affiliation 信息：

```
>>> import json
>>> pa_name_aff = json.load(open("paperIdAuthorId_to_name_and_affiliation.json"))
>>> pa_name_aff['1156615|2085584']
  {u'affiliation': u'Huawei##Microsoft Research Asia', u'name': u'Hang Li##Hang Li'}
```

7. 训练集：**Train.csv**。ComfirmedPaperIds 列对应的表示该作者写了这些论文的列表，DeletedPaperIds 列对应的表示该作者没有写这些论文论文。

字段名称	数据类型	注释
AuthorId	int	作者ID
ComfirmedPaperIds	string	以空格分割的论文(PaperId) 列表
DeletedPaperIds	string	以空格分割的论文(PaperId) 列表

8. 验证集：验证集**Valid.csv** 文件的格式如下：

字段名称	数据类型	注释
AuthorId	int	作者ID
PaperIds	string	以空格分割的论文(PaperId) 列表，待测的论文列表

9. 验证集答案：**Valid.gold.csv** 是验证集的标准答案，文件格式与训练集Train.csv 格式 相同。

10. 测试集：**Test.csv**。测试集Test.csv 文件的格式与验证集Valid.csv 格式相同，将在之 后发布。测试文件命名为Test.##.csv，其中##为各个小组的编号，如Test.01.csv 表示 第一个小组的测试集。

11. 因此，各个小组最终需要提交的是**测试集预测结果**，提交文件的格式与Valid.gold.csv 相同。文件命名为Test.P##.csv，其中##为各个小组的编号，如Test.P01.csv 表示第 一个小组提交的测试集预测结果。

12. 数据集的统计

数据集	(作者-论文)对 个数
训练集（Train.csv）	11,263
验证集（Valid.csv）	2,347
测试集（Test.csv）	每个队伍的测试集不同, 约1,300

3. 提交格式

最终提交的文件是对“测试集”的预测结果。该预测结果文件的格式与训练集 **Train.csv** 的格式相同，包含AuthorId、ComfirmedPaperIds、DeletedPaperIds 字段。该预测结果文件的 命名为Test.P##.csv，其中##为各个小组的编号，如Test.P01.csv 表示第一个小组提交的测试 集预测结果。

4. 评估标准

使用在“测试集”上的结果的准确率（Accuracy）作为评估标准。

评估脚本位于model_trainer 文件夹下，名为 evalution.py，运行该脚本可以获得评估结果

```
python  evalution.py  gold_file_path  pred_file_path
```

其中，gold_file_path 为标准答案所在的路径，pred_file_path 为预测文件所在的路径。 例如，第一小组在验证集合上的预测结果与标准答案的评估：

```
python  evalution.py  valid_set/Valid.gold.csv  valid_set/predict.csv
```

KDD 基准系统的实现思路

这个基准系统的实现思路分为四步：

1. 首先，根据任务的目标，从给定的数据集合中构造出训练正负样本/测试样本；
2. 其次，从给定的数据集合中，对构造出的训练样本/测试样本进行特征的设计和抽取，并针对每个训练样本/测试样本生成相应的特征集合；
3. 然后，选择分类算法，在训练样本生成的特征集合上构造分类器；
4. 最后，使用分类器对测试样本的特征集合进行预测，并将分类器的预测结果转换为任务要求的提交格式。

具体地说，以上步骤的程序实现如下：

1、 构造训练 / 测试的正负样本

这部分代码位于 model_trainer/data_loader.py 中，其中 load_train_data(train_path) 和 load_test_data(test_path) 分别为加载训练样本和测试样本的方法。

1) 构建训练样本。 系统从 data/dataset/train_set/Train.csv 中构建训练集的正负样本。
   - 将 authorId 与 ConfirmedPaperIds 中的每个 paperId 组合，作为正样本（label 为 1）；
   - 将 authorId 与 DeletedPaperIds 中的每个 paperId 组合，作为负样本（label 为 0）。
2) 构建测试样本。系统从 data/dataset/valid_set/Valid.csv 或 `data/dataset/test_set/Test.csv 中构建测试样本。 由于测试集的类标是待预测的，这里直接将其赋值为 -1。

2、 构造特征

分别为每一个训练 / 测试样本设计并抽取特征。特征抽取函数位于 model_trainer/feature_functions.py 中。 目前基准系统已经实现的特征有：

1) 共作者特征（共作者的相似度特征）

一篇论文会存在多个作者，根据 PaperAuthor.csv 统计每一个作者的 top 10（也可以是 top 20 或者其他 top K）的共作者 coauthor（本系统已经从 PaperAuthor.csv 获取了每个作者 top 10 的共作者，保存在 **coauthor.json** 文件中。）。对于一个作者论文对（aid，pid），计算 PaperId 为 pid 的论文作者是否出现在 AuthorId 为 aid 的作者的 top 10 coauthor 中。有两种计算方式：

- 计算 PaperId 为 pid 的论文的作者在 AuthorId 为 aid 的作者的 top 10 coauthor 中出现的人（个）数，作为一个特征。对应于 model_trainer/feature_functions.py 代码中的 coauthor_1() 特征。
- 计算 PaperId 为 pid 的论文的作者，与在 AuthorId 为 aid 的作者的 top 10 coauthor 中的作者的合作次数进行累加，将累加后的次数作为一个特征。对应于 model_trainer/feature_functions.py 代码中的 coauthor_2() 特征。

  目前，这两个特征已经完成实现。

2) 字符串距离特征（计算作者名字和单位相似度特征）

首先, PaperAuthor.csv 里面是有噪音的，同一个（authorid,paperid）可能出现多次，把同一个（authorid,paperid）对的多个 name 和多个 affiliation 合并起来。例如,

```
aid,pid,name1,aff1
aid,pid,name2,aff2
aid,pid,name3,aff3
```

我们可以得到

aid,pid,name1##name2##name3,aff1##aff2##aff3 其中，“##”为分隔符。

本系统已经从 PaperAuthor.csv 中为每一个 (aid,pid) 对获取了 name1##name2##name3,aff1##aff2##aff3 信息，并保存于 **paperIdAuthorId_to_name_and_affiliation.json** 文件中。另一个方面，我们可以根据（authorid,paperid）对中的 authorid 到 Author.csv 表里找到对应的 name 和 affiliation。

假设当前的作者论文对是(aid,pid), 从 **paperIdAuthorId_to_name_and_affiliation.json** 里得到的 name 串和 affiliation 串分别为 name1##name2##name3, aff1##aff2##aff3, 根据 aid 从 Author.csv 表找到的 name 和 affliction 分别为 name-a，affliction-a，这样我们可以计算字符串的距离。

特征计算方式有两种：

- 计算 name-a 与,name1##name2##name3 的距离；类似地，计算 affliction-a 和,aff1##aff2##aff3 的距离。实现代码对应于 model_trainer/feature_functions.py 中的 stringDistance_1() 方法。
- 分别计算 name-a 与 name1，name2 和 name3 的各自距离，然后对这三个距离取平均；类似地，计算 affliction-a 和,aff1，aff2，aff3 的平均距离。实现代码对应于 model_trainer/feature_functions.py 中的 stringDistance_2() 方法。

距离（相似度）的度量实现以下三种, 代码位于 model_trainer/feature_functions.py 中：

- 编辑距离（levenshtein distance）
- 最长公共子序列（LCS）
- 最长公共子串（LSS）

这样, 我们就得到关于作者 name 和作者 affiliation 的字符串相似度的多个特征。

注意： 所有设计抽取的特征，可根据需要在特征向量列表中进行添加和删除，对应的代码在 model_trainer/traine.py 中的 main()中的 feature_function_list []。

3、分类器选择

分类器的实现代码在 classifier.py 中，每一种分类器，对应于一个类（class）。目前系统实现的分类器有：

- Decision Tree
- Naive Bayes
- KNN
- SVM
- Logister Regreation
- Random Forest
- AdaBoost
- VotingClassifier（ensemble）

附：系统实现的细节

1、特征的添加

每一个特征的抽取都对应一个特征函数，位于 model_trainer/feature_functions.py 中。因此，若需要添加一个特征，则在 model_trainer/feature_functions.py 中增加一个函数，但是必须保证函数的接口不变。

如下所示为共作者的特征函数，输入必须是 (AuthorIdPaperId, dict_coauthor, dict_paperIdAuthorId_to_name_aff, PaperAuthor, Author) 这些参数，返回值为特征对象。

```
def coauthor_1(AuthorIdPaperId, dict_coauthor, dict_paperIdAuthorId_to_name_aff, PaperAuthor, Author):
    authorId = AuthorIdPaperId.authorId
    paperId = AuthorIdPaperId.paperId

    # 从 PaperAuthor 中，根据 paperId 找 coauthor。
    curr_coauthors = list(map(str, list(PaperAuthor[PaperAuthor["PaperId"] == int(paperId)]["AuthorId"].values)))
    #
    top_coauthors = dict_coauthor[authorId].keys()

    # 简单计算 top 10 coauthor 出现的个数
    nums = len(set(curr_coauthors) & set(top_coauthors))

    return util.get_feature_by_list([nums])
```

添加完特征函数后，可直接在 model_trainer/trainer.py 中调用。添加到变量 feature_function_list 中即可。如下所示，表示使用 coauthor_1 和 coauthor_2 特征来训练模型：

```
''' 特征函数列表 '''
feature_function_list = [
    coauthor_1,
```

```
    coauthor_2,
    # stringDistance_1,
    # stringDistance_2,
]
```

2、分类器的添加

所有分类器的实现代码位于 classifier.py 中，每个分类器对应于一个类，并继承于策略（Strategy）类。每个分类器类都需要实现 train_model （训练模型） 和 test_model（测试模型）方法。

添加完特征后，可直接在 model_trainer/trainer.py 中通过改变 classifier 变量的值来调用，例如 Naive Bayes 分类器的调用：

```
classifier = Classifier(skLearn_NaiveBayes())
```

KDD 基准系统的改进思路

针对以上这个基准系统，大家可以考虑从下面几个方面进行系统的改进和提升

1. 添加 journal 和 conference 信息

可以考虑作者 aid 之前发表的论文的 journal 和 conference，与当前的论文 pid 的 journal 和 conference 之间的相似度，作为特征。

2. 论文的 keyword 信息

作者 A 写过的论文的 keyword 构成一个集合 X，一篇论文 B 的 keyword 构成一个集合 Y，这里的 keyword 指的是论文的 title 和 keyword 分词后得到的单词，对于一个作者论文对（A，B）计算他们的 keyword 的交集或者相似度：X∩Y。 每个单词可以计算类似于 tf-idf 的分数，最后把属于 X∩Y 的单词的分数累加起来作为一维新的特征。

3. 特征计算方式

考虑尝试不同的字符串相似度的计算方式。

4. 模型参数

考虑尝试调整模型的超参数来提升性能。

5. 模型算法的选择

考虑尝试使用不同的算法或 ensemble 的方法来提升性能。