Auton^n ML

Saswati Ray, Jessie Chen, Stefania La Vattiata, Artur Dubrawski

Feb 17, 2022

GETTING STARTED

1 Getting started **3**
- 1.1 Docker . . . 3
- 1.2 D3M dataset . . . 3
- 1.3 Starting script . . . 3
- 1.4 Search and predictions . . . 4

2 Convert raw dataset to D3M dataset **5**
- 2.1 Example . . . 5
- 2.2 Example of creating D3M dataset for image regression . . . 6
- 2.3 Valid task types(s) . . . 6
- 2.4 Valid data type(s) . . . 6
- 2.5 Valid metrics . . . 6

3 Auto^n ML core functionalities **9**

Auto^{n}ML

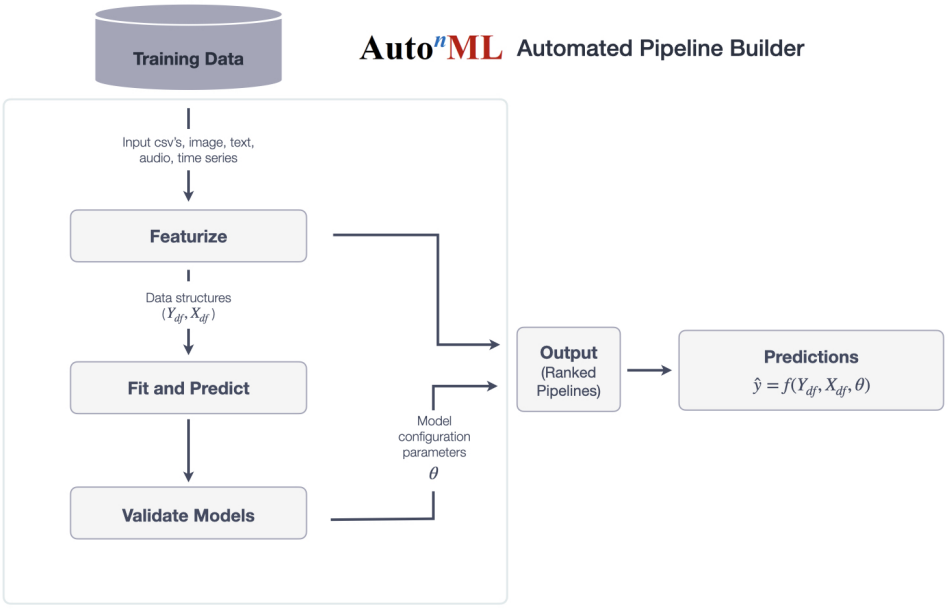
Welcome to the documentation site for CMU TA2 Auto^n ML.

Built using DARPA D3M ecosystem

Auto^{n}ML is an automated machine learning system developed by CMU Auton Lab to power data scientists with efficient model discovery and advanced data analytics Auton ML also powers the D3M Subject Matter Expert (SME) User Interfaces such as Two Ravens

Taking your machine learning capacity to the nth power

This is how the model works



CHAPTER

ONE

GETTING STARTED

1.1 Docker

Docker image available at

```
registry.gitlab.com/sray/cmu-ta2:latest
```

1.2 D3M dataset

The Auto^{n}ML can accept any type of D3M dataset. If there is no D3M dataset, this is how a raw dataset is converted to a D3M First install the d3m package

```
pip install d3m
python create_d3m_dataset.py <train_data.csv> <test_data.csv> <label> <metric> -t␣
↪classification <-t ...>
```

Currently d3m package needs Python 3.6 only

Detailed description of dataset type(s), task type(s) and metrics provided in the Converting raw data to d3m datasets section

1.3 Starting script

- Requires docker on your OS.
- Update location of your dataset for target "input" to the docker run.
- Run the following script

```
./scripts/start_container.sh
```

The above script has 4 mount points for the docker

1. input: Path of the input dataset
2. output: Directory where all outputs will be stored
3. static: Location of all static files (Use static directory of this repository)
4. scripts: Location of this repository's scripts.

1.4 Search and predictions

The above script will do the following

1. Pull docker image and run search for best pipelines for the specified dataset using TRAIN data
2. JSON pipelines (with ranks) will be output in JSON format at /output/<search_dir>/pipelines_ranked/
3. CSV prediction files of the pipelines trained on TRAIN data and predicted on TEST data will be available at /output/<search_dir>/predictions/
4. Training data predictions (cross-validated mostly) are produced in the current directory as /output/<search_dir>/training_predictions/<pipeline_id>_train_predictions.csv.
5. Python code equivalent of executing a JSON pipeline on a dataset produced at /output/<search_dir>/executables/

This code can be run as

```
python <generated_code.py> <path_to_dataset> <predictions_output_file>
```

An example

```
python /output/6b92f2f7-74d2-4e86-958d-4e62bbd89c51/executables/131542c6-ea71-4403-9c2d-
↪d899e990e7bd.json.code.py 185_baseball predictions.csv
```

- If feature_importances and intermediate outputs are desired, call scripts/run_outputs.sh instead of scripts/run.sh from scripts/start_container.sh

Features importances and intermediate step outputs will be produced in /output/<search_dir>/pipeline_runs/

CHAPTER

TWO

CONVERT RAW DATASET TO D3M DATASET

Currently d3m package needs Python 3.6 only.

```
pip install d3m
python create_d3m_dataset.py <train_data.csv> <test_data.csv> <label> <metric> -t␣
↪classification <-t ...>
```

2.1 Example

Some examples of valid commands are -

```
python create_d3m_dataset.py train_data.csv test_data.csv Label accuracy -t␣
↪classification
python create_d3m_dataset.py train_data.csv test_data.csv Value meanSquaredError -t␣
↪regression
```

-t option should be used to specify task types(s), data types(s). metrics. This script will create a directory structure "raw" for your dataset in D3M format. This dataset should be used as input to ./scripts/start_container.sh

This is the structure created for a generated D3M dataset:

```
raw$ tree
.
├── TEST
│   ├── dataset_TEST
│   │   ├── datasetDoc.json
│   │   ├── metadata.json
│   │   └── tables
│   │       └── learningData.csv
│   └── problem_TEST
│       └── problemDoc.json
└── TRAIN
    ├── dataset_TRAIN
    │   ├── datasetDoc.json
    │   ├── metadata.json
    │   └── tables
    │       └── learningData.csv
    └── problem_TRAIN
        └── problemDoc.json

8 directories, 8 files
```

2.2 Example of creating D3M dataset for image regression

```
python create_d3m_dataset.py train.csv test.csv WRISTBREADTH meanSquaredError -t␣
↪regression -t image
Namespace(dataFileName='train.csv', metric='meanSquaredError', target='WRISTBREADTH',␣
↪tasks=['regression', 'image'], testDataFileName='test.csv')
Going to create TRAIN files!
Going to create TEST files!
Please enter directory name for TRAIN media files: train_images
Please enter directory name for TEST media files: test_images
Please enter column name for media files: image_file
```

Note: Some task/data type(s) may not be entirely automated (Eg., object detection, graph problems). TRAIN, TEST hierarchies will be made available. However, datasetDoc.json might need to be customized for linking resources/tables for the specific task. For this purpose, example datasets are provided for reference purposes.

2.3 Valid task types(s)

linkPrediction, graphMatching, forecasting, classification, semiSupervised, clustering, collaborativeFiltering, regression, objectDetection, vertexNomination, communityDetection, vertexClassification

2.4 Valid data type(s)

Valid data type(s) to specify are- audio, image, video, text, timeSeries

2.5 Valid metrics

classification/linkPrediction/graphMatching/vertexNomination/vertexClassification: accuracy, f1Macro, f1Micro, rocAuc, rocAucMacro, rocAucMicro regression/forecasting/collaborativeFiltering: rSquared, meanSquaredError, meanSquaredError, meanAbsoluteError communityDetection/clustering: normalizedMutualInformation

Sample D3M dataset(s) for task type(s), data types(s):

- `classification`: 185_baseball_MIN_METADATA
- `regression`: 196_autoMpg_MIN_METADATA
- `forecasting`: LL1_736_stock_market_MIN_METADATA
- `audio`: 31_urbansound_MIN_METADATA
- `video`: LL1_VID_UCF11_MIN_METADATA
- `text`: LL1_TXT_CLS_airline_opinion_MIN_METADATA
- `timeseries`: 66_chlorineConcentration_MIN_METADATA
- `image`: 22_handgeometry_MIN_METADATA
- `collaborativeFiltering`: 60_jester_MIN_METADATA
- `communityDetection`: 6_70_com_amazon_MIN_METADATA
- `graphMatching`: 49_facebook_MIN_METADATA

- `linkPrediction`: 59_umls_MIN_METADATA
- `vertexClassification`: LL1_VTXC_1343_cora_MIN_METADATA

CHAPTER

THREE

AUTO^N ML CORE FUNCTIONALITIES

1. **Runs as server waiting to hear TA3 client GRPC requests on port 45042**

```
./src/main.py ta2ta3
```

2. **Runs in stand-alone TA2 mode without any client/TA2-3 API. This will search on TRAIN data for valid pipelines and output them**

```
./src/main.py search.
```

3. **For a given dataset-problem specification**
    - Search for valid pipelines (solutions)
    - Produce top pipelines with ranks. Pipelines are output as JSON files for evaluation purposes
    - Enable scoring, fit, produce on any pipeline. These calls are invoked from TA3

4. **TA2 pipelines can be evaluated independently using D3M's reference runtime framework on the TRAIN-TEST dataset splits. (See scripts/run.sh)**

"Pipeline" and "solution" are aliases and mean the same concept in TA2. A pipeline or solution is an end-to-end flowchart (DAG) composed of TA1 primitives, their hyperparameters and their connections to build model on the TRAIN data and produce predictions on the TEST data.