# ClayRS

## *CBRS framework written in Python*

*Architecture and its three main modules*

*Slides made by:* Antonio Silletti

# Infographic timeline

## March 2020

### Initial idea

Alpha version made by Mattia Patruno, Roberto Barile, Francesco Benedetti, Carlo Parisi

## December 2020

### First Refactor

Refactor of classifiers and graphs, redo of the testing pipeline made by Antonio Silletti

## April 2021

### Complete Redo

Major Refactor and Overhaul of the whole framework made by Antonio Silletti, Elio Musacchio, Roberta Sallustio

## July 2021

### User Interface

Initial UI made by Raffaele Disabato (discontinued atm)

## October 2021

### Load tests

Testing of algorithms and metrics by Giuseppe Sancesario and Vincenzo Maria Giulio Martemucci

# Infographic timeline

## April 2022

### Moved to SWAP

Changed name to **ClayRS** and moved repo to SWAP organization

## July 2022

### v0.1.0 on pip 🎉

**Initial version** released on pypi with the complete documentation!

## September 2022

### v0.2.0 released

Implemented multiprocessing, added bootstrap partitioning, MAP and MAP@K metrics

## October 2022

### v0.3.0 released

Implemented Experiment class for easily compare different algorithms

## December 2022

### v0.4.0 released

Implemented different weighting schema for personalized page rank

2023

# Who's and what's next?

This is a project entirely made by students, with the constant and maximum supervision of the **SWAP** research group:

- *Prof. Pasquale Lops*
- *Ph.D. Marco Polignano*
- *Prof.  Giovanni Semeraro*
- *Prof. Cataldo Musto*
- *Prof. Pierpaolo Basile*

There's so much to do with so small time, that's why highly motivated people are always welcome to come on board ☺

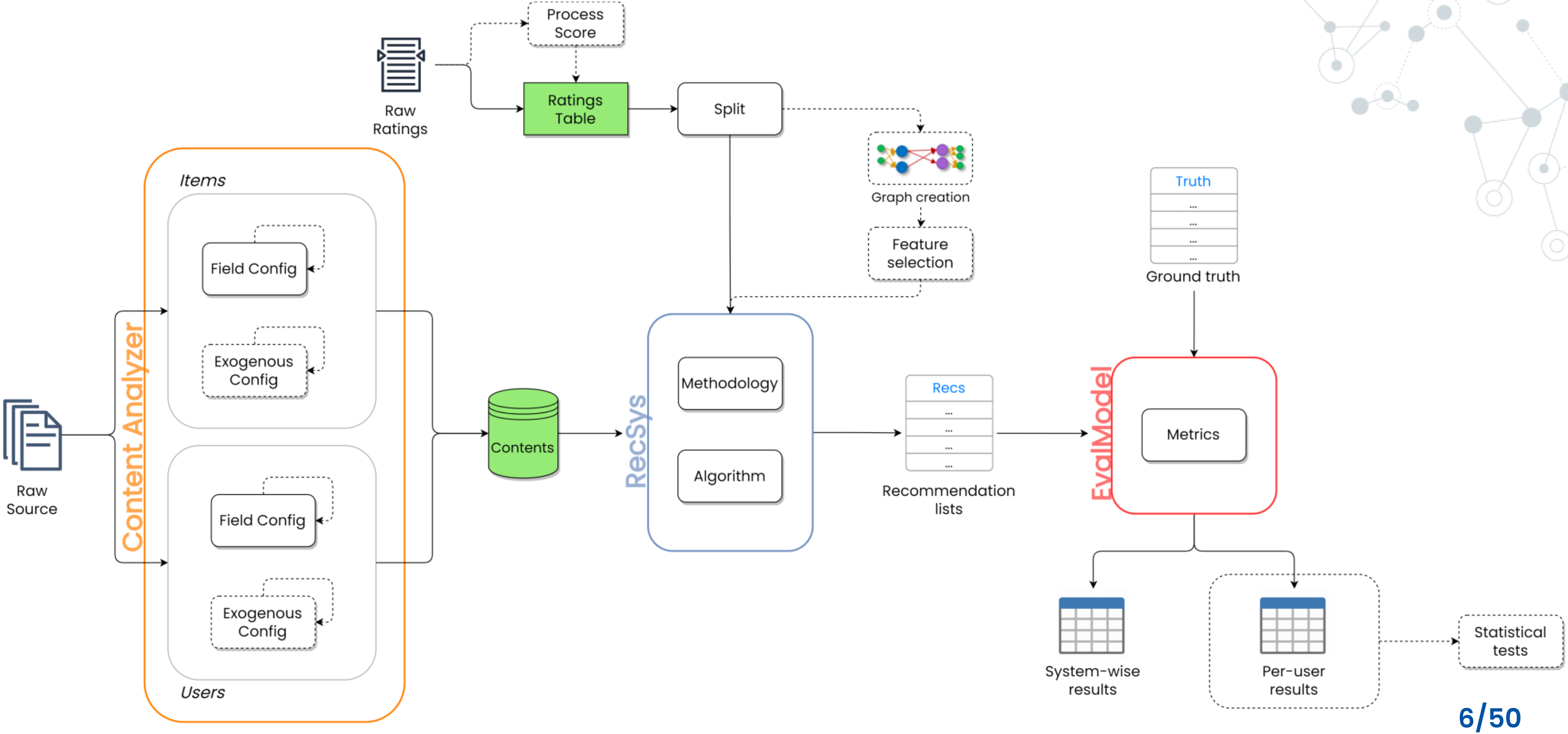
# Idea

ClayRS allows you to conduct a complete experiment, starting from a raw representation of users and items to building and evaluating a recommender system.

It does so with three main modules, which you can also use individually:

*Content Analyzer* | *RecSys* | *EvalModel*

# General pipeline

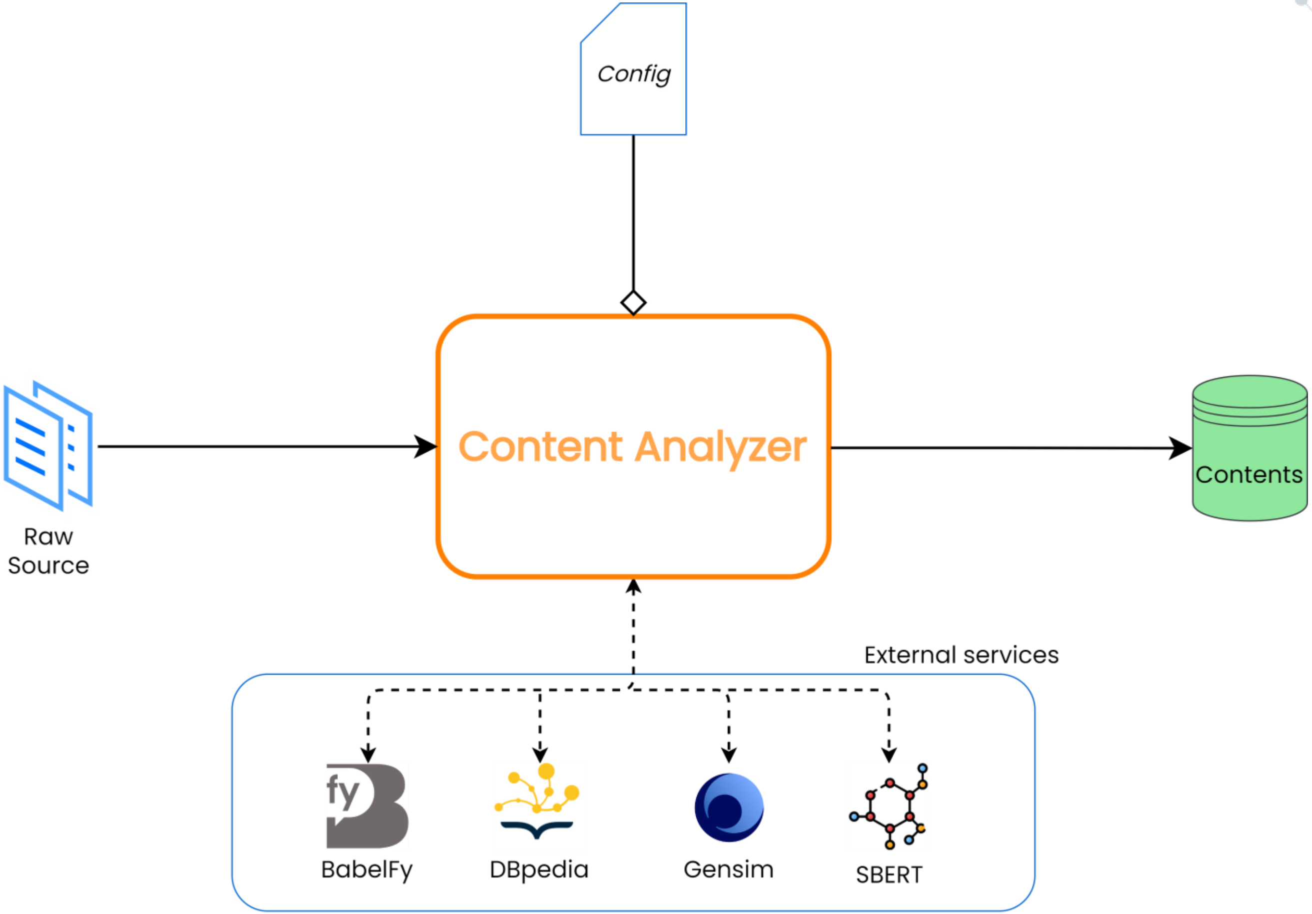


# *Content Analyzer*

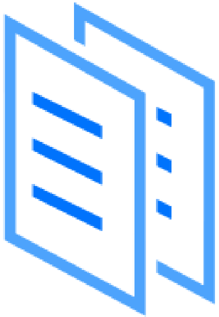
Given a raw source, the `Content Analyzer`:

- Creates and serializes content,
- Using the chosen configuration

# Content Analyzer Architecture



# What do we mean by Raw Source?



Raw Source

- A **Raw Source** is a file containing several information of the content to represent

  - Ex. Movies in a `JSON` file

```
{  "Title": "Jumanji",
   "IMDB_ID": "0113497"
   "Year": "1995",
   "Rated": "PG",
   "Genre": "Adventure, Family, Fantasy",
   "Released":"15 Dec 1995",
   "Runtime":"104 min",
   … }
```

  - Ex. Users in a `CSV` file (represented here as a table)

| User_ID | Name | Age | Occupation |
|---|---|---|---|
| 1 | Antonio | 24 | student |
| 2 | Mario | 44 | lawyer |

# How can we represent content?

*Config*

You can set via configuration which fields of every raw content must be represented and how to represent them

- *Multiple representations for a single field are allowed*

```
{ "Title": "Jumanji",
  "Year": "1995",
  "Rated": "PG",
  "Genre": "Adventure, Family, Fantasy",
  "Released": "15 Dec 1995",
  "Runtime": "104 min",
  … }
```

*tf-idf*

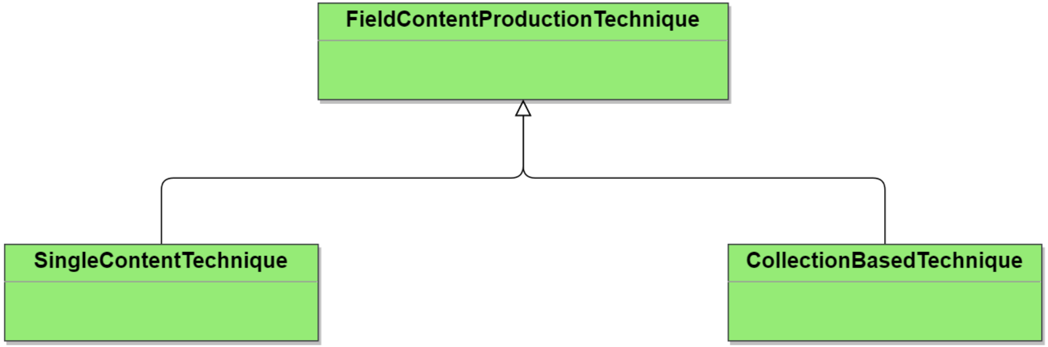
```
{ 'adventure': 0.49122938570380714,
  'family': 0.5631439252085217,
  'fantasy': 0.6645017758605305 }
```

*embedding with glove-twitter-25* model

```
array([-0.01160799, -0.28532598, 0.0177138,
       -0.128398,   -0.10475059, 0.37144921,
        1.08316798,  0.63348202, 0.226706,
        …,           …,          …])
```

# Which representation techniques are implemented?

Every representation technique is implemented as a `class`, in order to make use of polymorphism




There are two techniques implemented as `CollectionBased` techniques:

`tf-idf`

- `SkLearnTfIdf()`
- `WhooshTfIdf()`

`Synset Document frequency`

- PyWSDSynsetDocumentFrequency()

# Which representation techniques are implemented?

There are several `SingleContent` techniques available:

- techniques which can store the original representation of the `raw source` *(useful for* `IndexQuery` *algorithm):*
  - `OriginalData()`

- techniques which compute the embedding representation of the content (cont.)

# Embedding techniques

Embedding techniques implemented include:

- `WordEmbedding()`
- `SentenceEmbedding()`
- `DocumentEmbedding()`

They also exist in their «combined form», given a combiner (`Centroid()`, `Sum()`):

- `Word2SentenceEmbedding()`
- `Word2DocumentEmbedding()`

  …

For each *embedding* technique, a pre-trained model must be specified. It will be downloaded (optionally) from the following external sources:

- *Gensim*
- *SBERT*
- *Hugging Face models (BERT and T5 models)*
- *Local storage*

# Embedding techniques

It’s also possible to train via ClayRS the following Gensim models:

- `FastText`
- `Word2Vec`
- `RandomIndexing`
- `LatentSemanticAnalysis`
- `Doc2Vec`

The field of every content of the *raw source* will be used as corpus

- If preprocessing operations are specified, the preprocessed corpus will be used
- The trained model will then be used (and optionally saved) to represent contents

# Other available operations

## Reduce the content dimensionality

By performing preprocessing operations, such as *stemming*, *lemmatization*, *stopwords removal*, etc. via the *NLTK* library, *SPACY or Ekphrasis*

```
"This is beautiful too and entertaining"
                ↓
['This', 'beautiful', 'entertaining']
```

## Save representations in an index

Each representation can be saved in an index, both for exporting reason or to perform a specific recommendation algorithm *(IndexQuery)*

```
...
content_id: 0113497
    Plot#0#original: After being trapped…
    Plot#1#tfidf: {trapped: 0.0185185…, …}
...
```

# Content Analyzer | code example

Let's instantiate a config both for Items and Users:

```
import clayrs.content_analyzer as ca

movies_config = ca.ItemAnalyzerConfig(
        source=ca.JSONFile("items_info.json"),
        id="IMDB_ID",
        output_directory='movies_codified/'
)

users_config = ca.UserAnalyzerConfig(
        source=ca.CSVFile("users_info.csv"),
        id="User_ID",
        output_directory='users_codified/'
)
```

items_info.json

```
{  "Title": "Jumanji",
   "IMDB_ID": "0113497"
   "Year": "1995",
   "Rated": "PG",
   "Genre": "Adventure, Family, Fantasy",
   "Released":"15 Dec 1995",
   "Runtime":"104 min",
   … }
```

users_info.csv

| User_ID | Name | Age | Occupation |
|---|---|---|---|
| 1 | Antonio | 24 | student |
| 2 | Mario | 44 | lawyer |

Let's specify how to represent items:

```
movies_config.add_single_config(
    'Plot',
    ca.FieldConfig(ca.SkLearnTfIdf())
)

movies_config.add_single_config(
    'Plot',
    ca.FieldConfig(ca.WhooshTfIdf(),
                   preprocessing=ca.NLTK(stemming=True),
                   id="whooshtfidf")
)

movies_config.add_single_config(
    'Genre',
    ca.FieldConfig(
        ca.WordEmbeddingTechnique(ca.Gensim('glove-twitter-25')) )
)
```

Equivalent of using `add_multiple_config(…)` and passing a list of representation

# Exogenous techniques

You can also expand the content via:

- *`DBpedia`*
- *`BabelFy Entity Linking`*
- *`Local dataset`*

For the `DBpedia` mapping technique you can choose how to retrieve properties in four different ways:

- **`only_retrieved_evaluated`:** retrieve only properties in DBpedia which have a value
- **`original_retrieved`:** retrieve only local properties which have a value in DBpedia
- **`all_retrieved`:** retrieve all properties from DBpedia
- **`all`:** retrieve all properties from DBpedia + all properties in local

Let's add an exogenous representation both for items and users:

```
movies_config.add_single_exogenous(
    ca.ExogenousConfig(
        ca.DBPediaMappingTechnique('dbo:Film', 'dbpedia_label')
    )
)

users_config.add_single_exogenous(
    ca.ExogenousConfig(
        ca.PropertiesFromDataset(field_name_list=['gender', 'occupation'])
    )
)
```

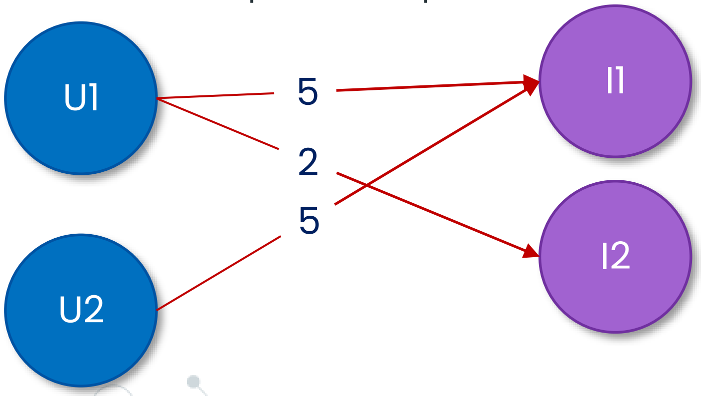
# Graphs

ClayRS can represent the User-Item rating matrix as a *graph*, which can be further manipulated by adding other kinds of nodes (property nodes), by removing some others, etc.

| User | Item | Rating |
|---|---|---|
| U1 | I1 | 5 |
| U1 | I2 | 2 |
| U2 | I1 | 5 |

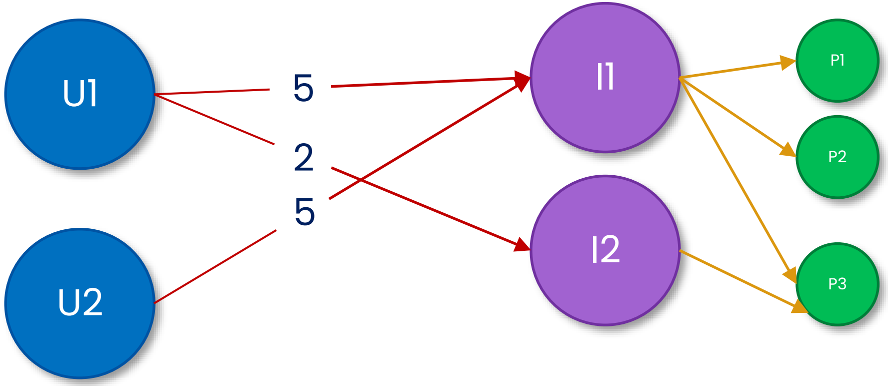
User-Item Rating Matrix

Bipartite Graph:



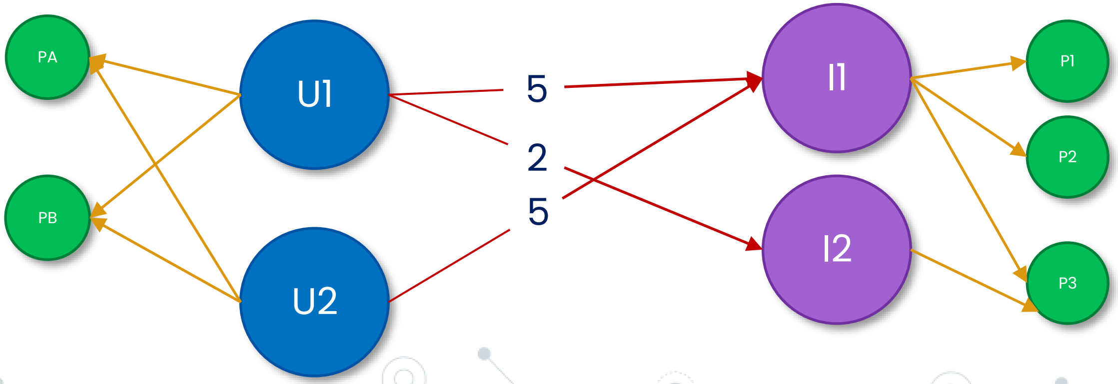
## Tripartite graph:

Graph with property nodes only linked to item nodes

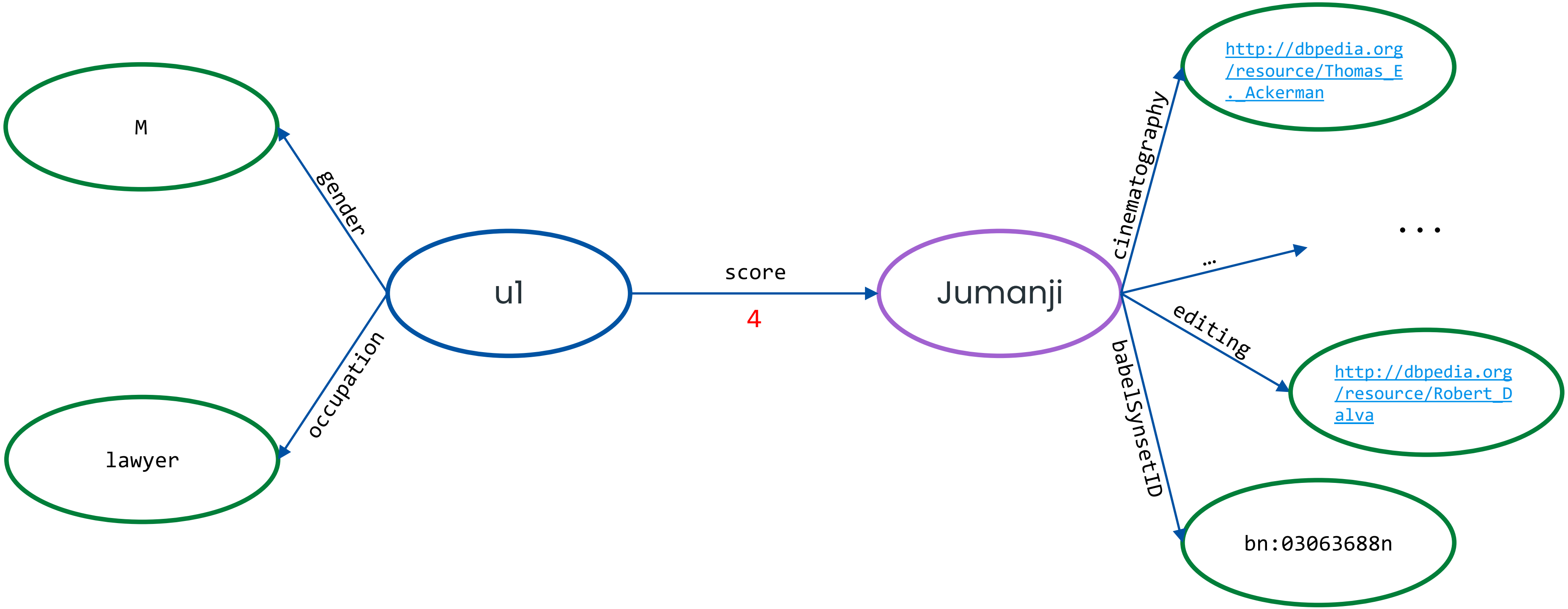



## Full Graph:

Graph with no restriction



# Full graph + exogenous techniques



# How to export content?

- In a format recognizable by the framework (`.xz`), so that contents could be used in the recommending phase

- In `JSON` format which is readable and allows complex representations to be used in other contexts



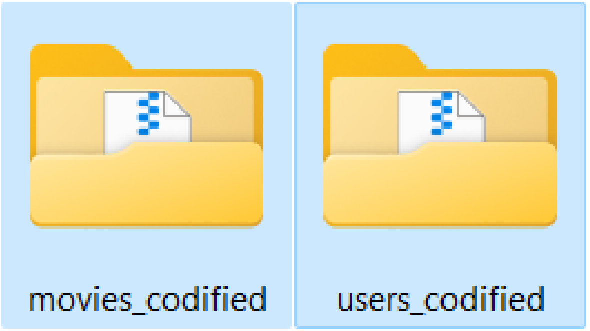
```
[
    {
        "content_id": "0113497",
        "Exo#0": "{'starring': ['http://dbpedia.org/resource/David_
        "Plot#0": "{'stop': 0.1901064866700191, 'city': 0.21627210C
        "Plot#1": "[[ 0.03125608  0.11693476  0.05183702 ... -0.109
        "Plot#2": "After being trapped in a jungle board game for 2
        "Plot#3": "trapped jungle board game 26 years , Man-Child w
        "Genre#0": "[[-4.71320003e-01  2.49479994e-01 -2.35080004e-
        "Genre#1": "{',': 0.028982257714459496, 'adventur': 0.45593
        "Genre#2": "Adventure, Family, Fantasy",
        "Genre#3": "Adventure , Family , Fantasy",
        "Year#0": "1995",
        "Year#1": "1995",
```

# Content Analyzer | output

Let’s serialize items and users with the representations we chose:

```
ca.ContentAnalyzer(movies_config).fit()

ca.ContentAnalyzer(users_config).fit()
```



We could also check if everything went smoothly:

```python
from clayrs.utils import load_content_instance

item = load_content_instance('movies_codified', '0113497')

print(item)
```

Exogenous representations

Field representations

```
Content:   0113497

Exogenous representations:
                                                   representation
internal_id external_id
0           NaN         {'cinematography': 'http://dbpedia.org/resourc...

Field: Plot
                                                   representation
internal_id external_id
0           NaN         {'them': 0.1144577150792816, 'stop': 0.1606690...
1           whooshtfidf {'26': 1.3010299956639813, 'After': 0.82390874...

Field: Genre
                                                   representation
internal_id external_id
0           NaN         [-0.58404666  0.21438999  0.06313567 -0.39509 ...
##############################
```

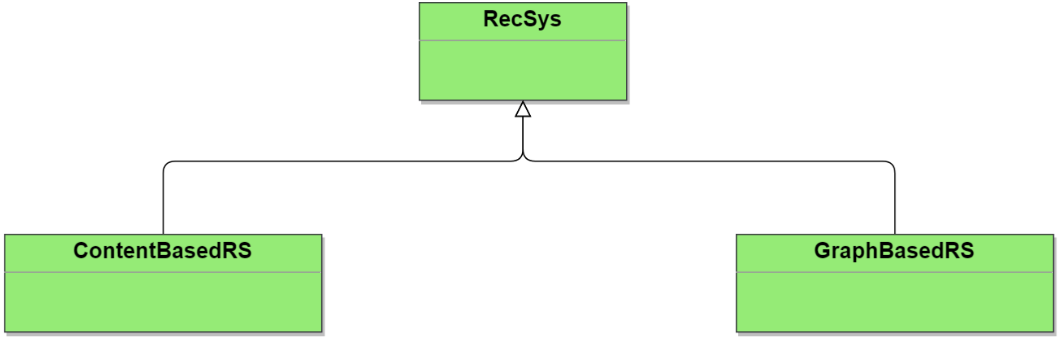
# *RecSys*

The `RecSys` module allows to:

- Instantiate a recommender system
  - Using items and users serialized by the *`Content Analyzer`*
- Make score prediction or recommend items for the active user

# Implemented Recommender Systems

Each recommender system is a specialization of the abstract class `RecSys`




There are two available implementations:

- *Content Based RecSys* (`ContentBasedRS`)
- *Graph Based RecSys* (`GraphBasedRS`)

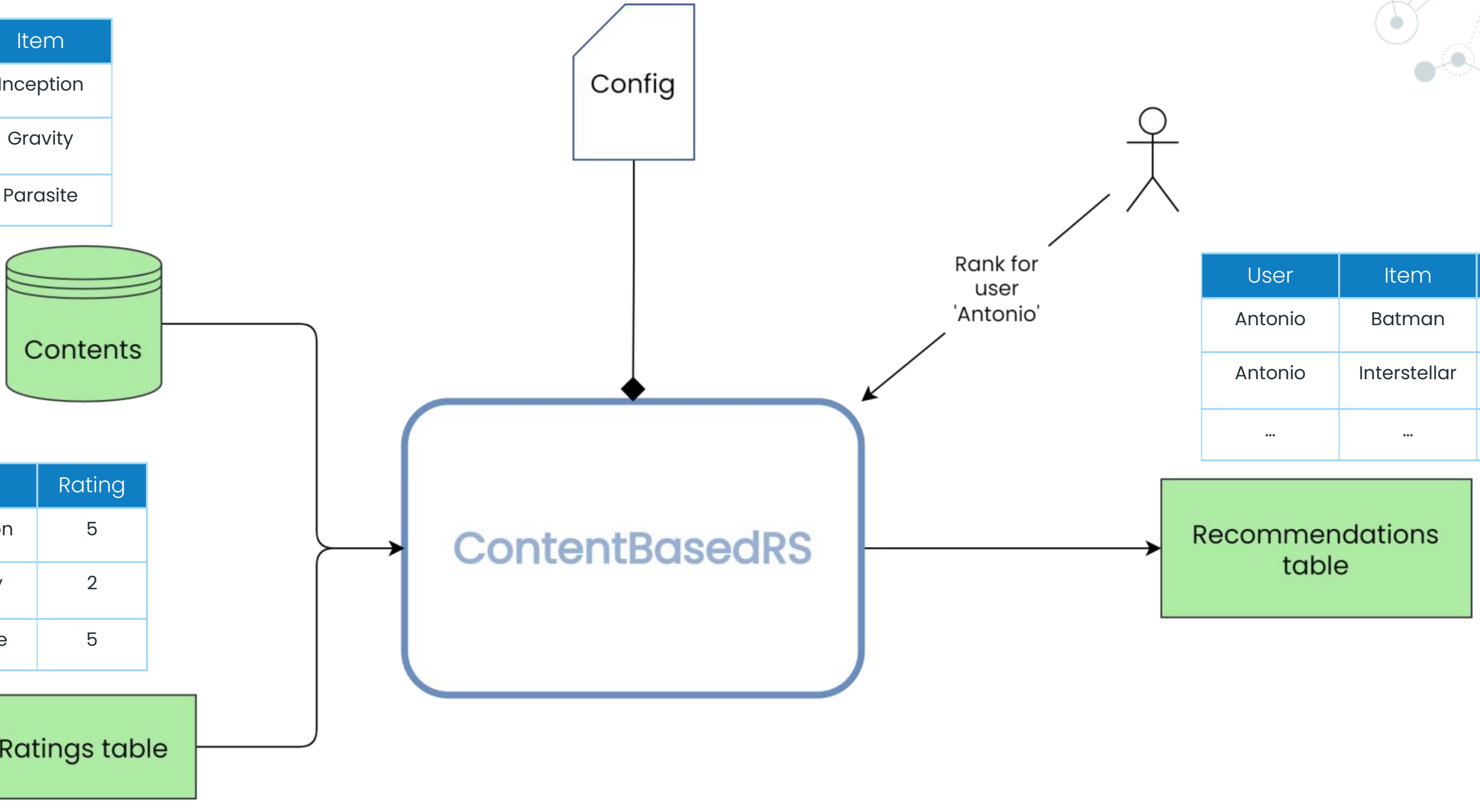
To implement other typologies, simply extend the `RecSys` class

# Content Based RS Architecture

| User |
|---|
| Antonio |
| Mattia |
| Gino |

| Item |
|---|
| Inception |
| Gravity |
| Parasite |

| User | Item | Rating |
|---|---|---|
| Antonio | Inception | 5 |
| Antonio | Gravity | 2 |
| Gino | Parasite | 5 |




| User | Item | Rank |
|---|---|---|
| Antonio | Batman | 0.9881 |
| Antonio | Interstellar | 0.9512 |
| ... | ... | ... |

# Ratings Table

In order to instantiate a RecSys, the User x Item matrix must be imported. The framework can also manipulate its ‘score’ field if a *processor* is specified

- `NumberNormalizer()`
- `TextBlobSentimentAnalysis()`

Ratings table

`NumberNormalizer` normalizes a numeric field in the $[-1, 1]$

`TextBlobSentimentAnalysis` returns the polarity of a text field in the $[-1, 1]$ range

# How to import Ratings into ClayRS

1. The columns are ordered:

```
ca.Ratings(ca.CSVFile('ratings_ordered.csv'))
```

2. The 'score' column isn't next to the 'item' column:

```
ca.Ratings(
    ca.CSVFile('ratings.csv')),
    score_column='points'
)
```

3. The 'score' column needs to be processed:

```
ca.Ratings(
    ca.CSVFile('ratings.csv')),
    score_column=2,
    score_processor=ca.TextBlobSentimentAnalysis()
)
```

`ratings.csv`

| user | item | review | points |
|---|---|---|---|
| Antonio | Inception | good | 4.5 |
| Mario | Gravity | bad | 1 |

# Splitting the dataset

```
n_split = 2
[train_split1, train_split2], [test_split1, test_split2] =
rs.KFoldPartitioning(n_split).split_all(original_ratings)
```

| from_id | to_id | rating |
|---|---|---|
| u1 | Tenet | 1 |
| u1 | Gravity | -0,5 |
| u2 | Her | 0,6 |
| u2 | Gravity | -0,2 |

1st split

*train_split1*

| from_id | to_id | rating |
|---|---|---|
| u1 | Tenet | 1 |
| u2 | Her | 0,6 |

*test_split1*

| from_id | to_id | rating |
|---|---|---|
| u1 | Gravity | -0,5 |
| u2 | Gravity | -0,2 |

2nd split

*train_split2*

| from_id | to_id | rating |
|---|---|---|
| u1 | Gravity | -0,5 |
| u2 | Gravity | -0,2 |

*test_split2*

| from_id | to_id | rating |
|---|---|---|
| u1 | Tenet | 1 |
| u2 | Her | 0,6 |

# Which partitioning techniques are available?

The following are the partitioning techniques available, all using the *sklearn* library:

- `KFoldPartitioning()`
- `HoldOutPartitioning()`
- `BootstrapPartitioning()`

# Choosing the CB algorithm

Representations codified

```
Field: Plot

internal_id external_id
0           NaN
1           whooshtfidf

Field: Genre

internal_id external_id
0           NaN

##########################
```

```python
import clayrs.recsys as rs

# Centroid Vector alg
alg = rs.CentroidVector(
    {
     'Plot': [0, 'whooshtfidf'],
     'Genre': 0
    }
    similarity=rs.CosineSimilarity()
    # threshold=2
)
# Classifier alg
alg = rs.ClassifierRecommender(
    {
     'Plot': [0, 'whooshtfidf'],
     'Genre': 0
    }
    classifier=rs.SkKNN(n_neighbors=4)
    # threshold=2
)
```

# Computing recommendations

Regardless of the algorithm chosen, simply instantiate the cbrs:

```
cbrs = rs.ContentBasedRS(alg, train_split1, items_path)
```

And then compute the rank, given the test set (which will only be used to compute which items are eligible for ranking):

```
cbrs.fit_rank(test_split1,
              recs_number=2,
              methodology=rs.TestRatingsMethodology())
```

| User | Item | Rank score |
|---|---|---|
| 3 | 0113497 | 0.98578 |
| 3 | 0114709 | 0.97324 |
| 8 | 0116367 | 0.99734 |
| 8 | 0117110 | 0.99711 |

# Which methodologies are available?

Methodologies available:
($L$ is the recommendation list, $u$ is the active user, $Te$ is the test set, $Tr$ is the training set)

- TestRatingsMethodology()

$$L_u = Te_u$$

- TestItemsMethodology()

$$L_u = \bigcup_v Te_v \setminus Tr_u$$

Where:
- $v$ is the generic user of the $Te$

- TrainingItemsMethodology()

$$L_u = \bigcup_{v \neq u} Tr_v$$

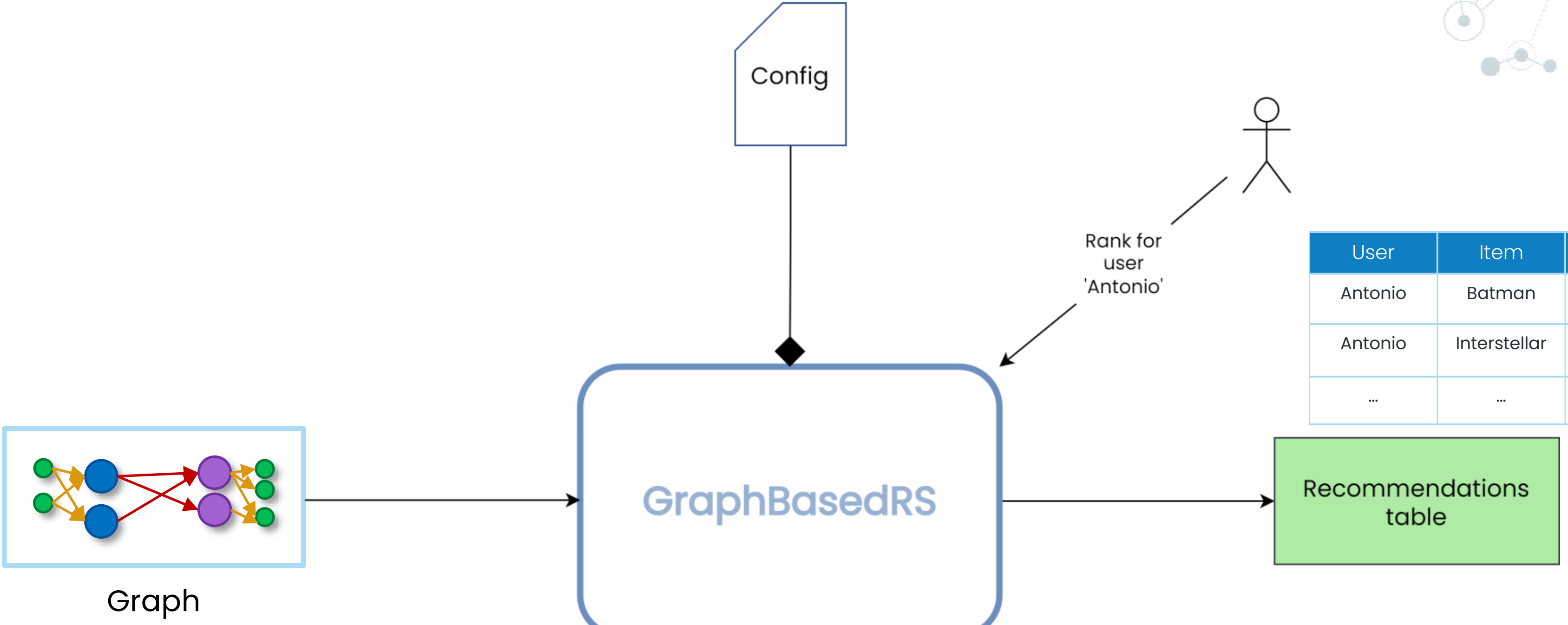
Where:
- $v$ is the generic user of the $Tr$

- AllItemsMethodology()

$$L_u = I \setminus Tr_u$$

Where:
- $I$ is the set which contains all items of the catalog

# Graph Based RS Architecture




| User | Item | Rank |
|---|---|---|
| Antonio | Batman | 0.9881 |
| Antonio | Interstellar | 0.9512 |
| ... | ... | ... |

# GBRS | code example

Graph creation:

```
full_graph = rs.NXFullGraph(train_set1,
                            user_contents_dir='users_codified/',
                            item_contents_dir='movies_codified/',
                            user_exo_representation=0,
                            item_exo_representation=0)
```

GBRS definition:

```
alg = rs.NXPageRank(personalized=True)

gbrs = rs.GraphBasedRS(alg, full_graph)
```

And then, just like for `ContentBasedRS`, you can compute the rank given the test set:

```
gbrs.rank(test_set1, …)
```

# Which recommendation algorithms are available?

GraphBasedRS:

- `PageRank()` *rank*
- `PageRank(personalized=True)` *rank*

---

ContentBasedRS:

- `CentroidVector()` *rank*
    - `CosineSimilarity()`
- `ClassifierRecommender()` *rank*
    - `SkSVC()`
    - `SkKNN()`
    - `SkLogisticRegression()`
    - `SkDecisionTree()`
    - `SkGaussianProcess()`
- `LinearPredictor()` *rank* *pred*
    - `SkLinearRegression()`
    - `SkRidge()`
    - `SkBayesianRidge()`
    - `SkSGDRegressor()`
    - `SkARDRegression()`
    - `SkHuberRegressor()`
    - `SkPassiveAggressiveRegressor()`
- `IndexQuery()` *rank*

# Flexibility of CB algorithms | a technical hint

Each content-based algorithm can work on multiple representations of a field, or even multiple representations of multiple fields

- e. g. You could train a classifier using a `tfidf` representation, a `WordEmbedding` representation and a `DocumentEmbedding` representation

Every representation specified will be vectorized (if necessary) and added as a column to the representation matrix that will be used in the *training phase*

- If vectors have different dimensions, they will be transformed into row vectors by using a chosen combiner from those implemented for the Content Analyzer (`Centroid()`, `Sum()`)

| tfidf (200 words) | word_embedding (357x50) | doc_embedding (1x300) | |
|---|---|---|---|
| tfidf_vectorized_sparse | word_embedding_centroid | doc_embedding | (1x550) |

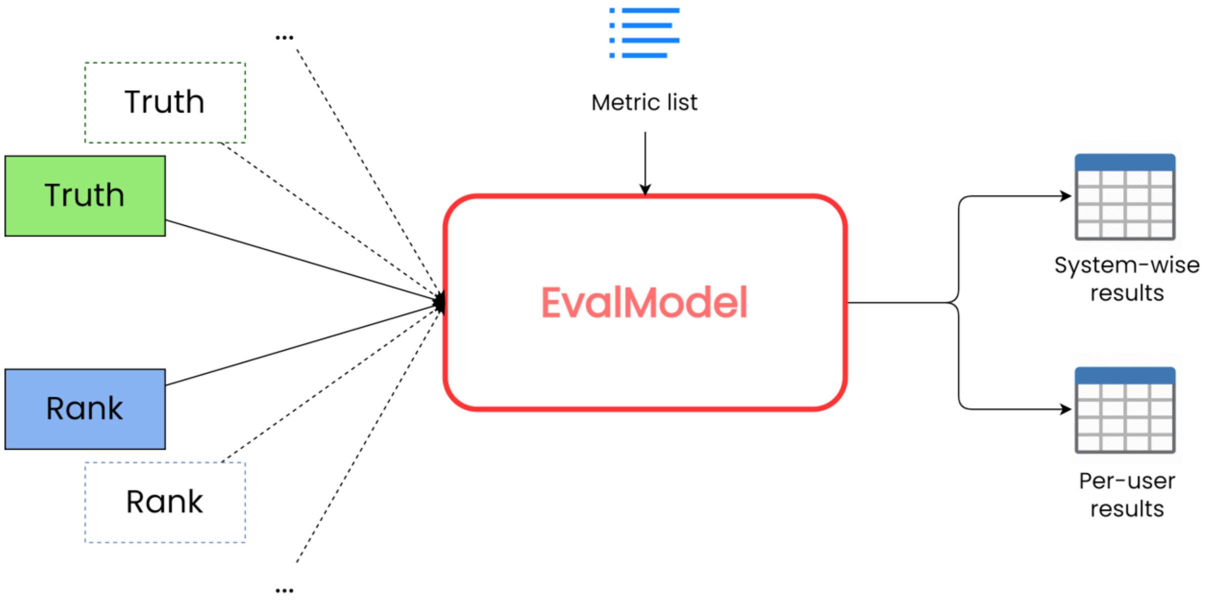
# *EvalModel*

The `EvalModel` has the task of evaluating a recommender system, using several state-of-the-art metrics

# EvalModel architecture

To evaluate a recommender, the `EvalModel` needs the rankings computed for each split, the relative ground truth, and a list of metrics

- It will return two pandas `DataFrame`, one containing metrics result for each user, the other containing metrics result of the whole system



# EvalModel | code example

```python
import clayrs.evaluation as eva

em = eva.EvalModel(
    pred_list=[rank_split0, rank_split1],
    truth_list=[test_split0, test_split1],
    metric_list=[
        eva.Precision(),
        eva.PrecisionAtK(k=1),
        eva.RPrecision(sys_average='micro'),
        eva.Recall(),
        eva.RecallAtK(k=3),
        eva.FMeasure(),
        eva.FMeasureAtK(k=2),
    ],
)

sys_result, users_result = em.fit()
```

## System wise results:

sys_result

| user_id | Precision - macro | Precision@1 | RPrecision – micro | … |
|---|---|---|---|---|
| sys - fold1 | 0.55662 | 0.55037 | 0.56369 | … |
| sys - fold2 | 0.54644 | 0.55885 | 0.55637 | … |
| sys - mean | 0.55153 | 0.55461 | 0.56003 | … |

## Results of each user

(avg of the two splits):

users_result

| user_id | Precision - macro | Precision@1 | RPrecision – micro | … |
|---|---|---|---|---|
| 1 | 0.6 | 1.0 | 0.6 | … |
| 10 | 0.25 | 0.5 | 0.25 | … |
| 100 | 0.5 | 0.5 | 0.5 | … |
| 101 | 0.75 | 1.0 | 0.75 | … |
| … | … | … | … | … |

# Which metrics are available?

**Classification** metrics (sys average computed as micro/macro):

- `Precision()`
- `PrecisionAtK()`
- `RPrecision()`
- `Recall()`
- `RecallAtK()`
- `FMeasure()`
- `FMeasureAtK()`

**Ranking** metrics:

- `NDCG()`
- `NDCGAtK()`
- `MRR()`
- `MRRAtK()`
- `Correlation()` → `'pearson'`, `'spearman'`, `'kendall'`
- `MAP()`
- `MAPAtK()`

# Which metrics are available?

**Error** metrics:

- `MSE()`
- `RMSE()`
- `MAE()`

**Fairness** metrics:

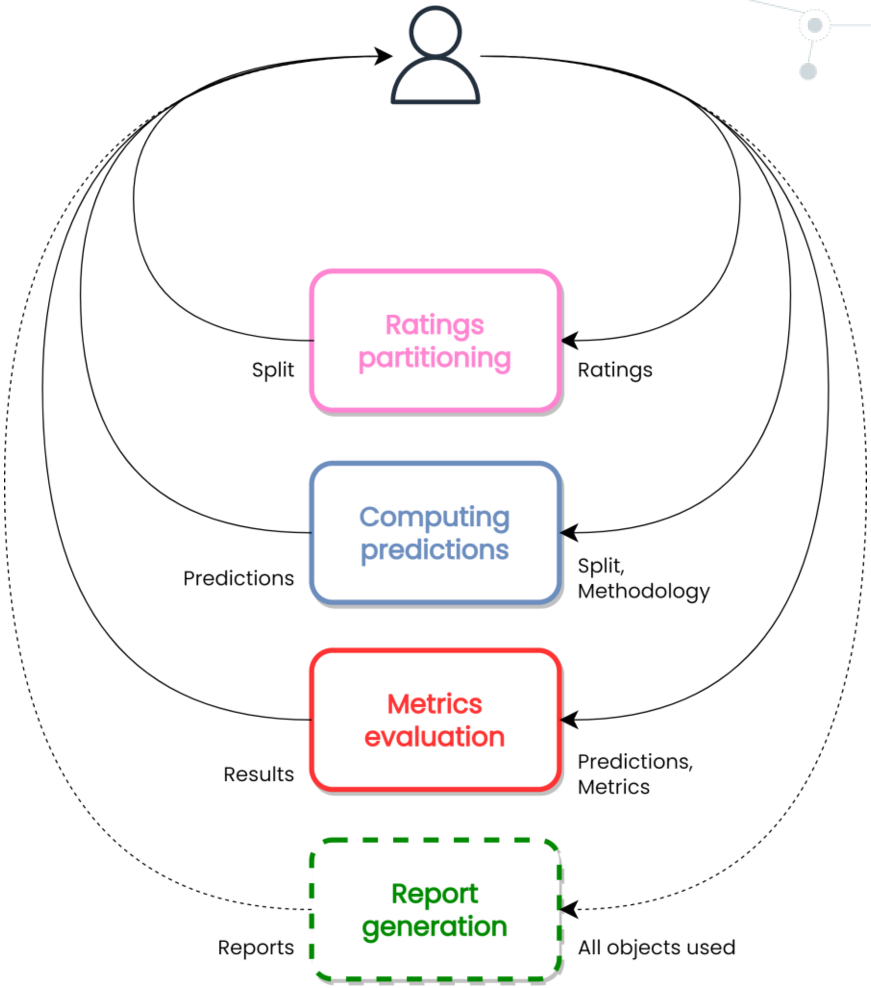
- `GiniIndex()`
- `PredictionCoverage()`
- `CatalogCoverage()`
- `DeltaGap()`

**Plot** metrics:

- `LongTailDistr()`
- `PopProfileVsRecs()`
- `PopRecsCorrelation()`

# Pipeline of recommending and evaluating

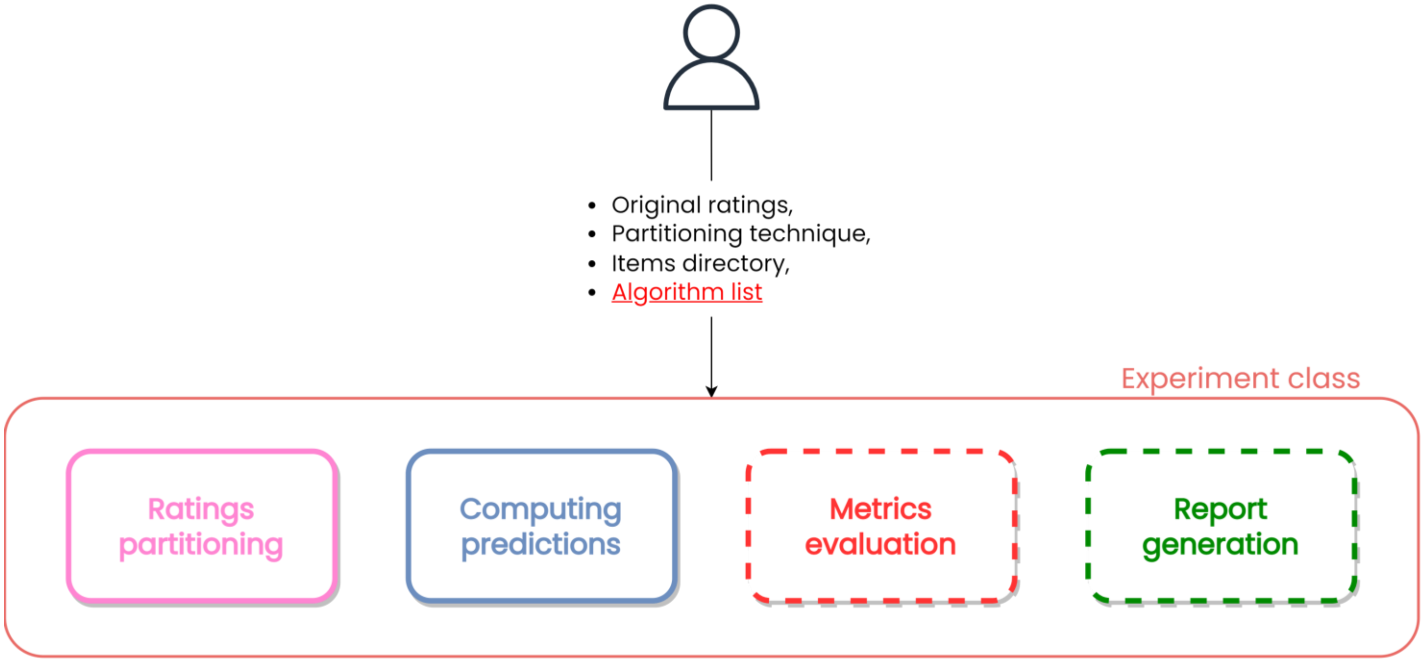
- Can this be automated?
- What about comparing different algorithms?



# Experiment class

With a simple interface, the `Experiment` class lets you cover a complete experiment!

- It also makes it easy to compare different algorithms



# Experiment class: code example

```
rat = ca.Ratings(ca.CSVFile('ratings.csv'))

alg1 = rs.CentroidVector({'plot': 0}, similarity=rs.CosineSimilarity())

alg2 = rs.ClassifierRecommender({'plot': 0}, classifier=rs.SkSVC())

metrics = [eva.RPrecision(sys_average='micro'), eva.RecallAtK(k=3)]

rs.ContentBasedExperiment(
        original_ratings=rat,
        partitioning_technique=rs.HoldOutPartitioning(),
        algorithm_list=[alg1, alg2],
        items_directory='movies_codified_plot',
        metric_list=metrics,
        report=True
).rank(n_recs=10)
```

# Experiment class: output

experiment_result

| | | | |
|---|---|---|---|
| CentroidVector_1 | 24/10/2022 19:35 | Cartella di file | |
| ClassifierRecommender_1 | 24/10/2022 19:35 | Cartella di file | |
| HoldOutPartitioning_test_split0.csv | 24/10/2022 19:35 | File con valori sep... | 255 KB |
| HoldOutPartitioning_train_split0.csv | 24/10/2022 19:35 | File con valori sep... | 995 KB |

| | | | |
|---|---|---|---|
| eva_report.yml | 24/10/2022 19:35 | File YML | 1 KB |
| eva_sys_results.csv | 24/10/2022 19:35 | File con valori sep... | 1 KB |
| eva_users_results.csv | 24/10/2022 19:35 | File con valori sep... | 23 KB |
| rs_rank_split0.csv | 24/10/2022 19:35 | File con valori sep... | 220 KB |
| rs_report.yml | 24/10/2022 19:35 | File YML | 1 KB |

ClayRS

# Thank you! 🤗

Antonio Silletti