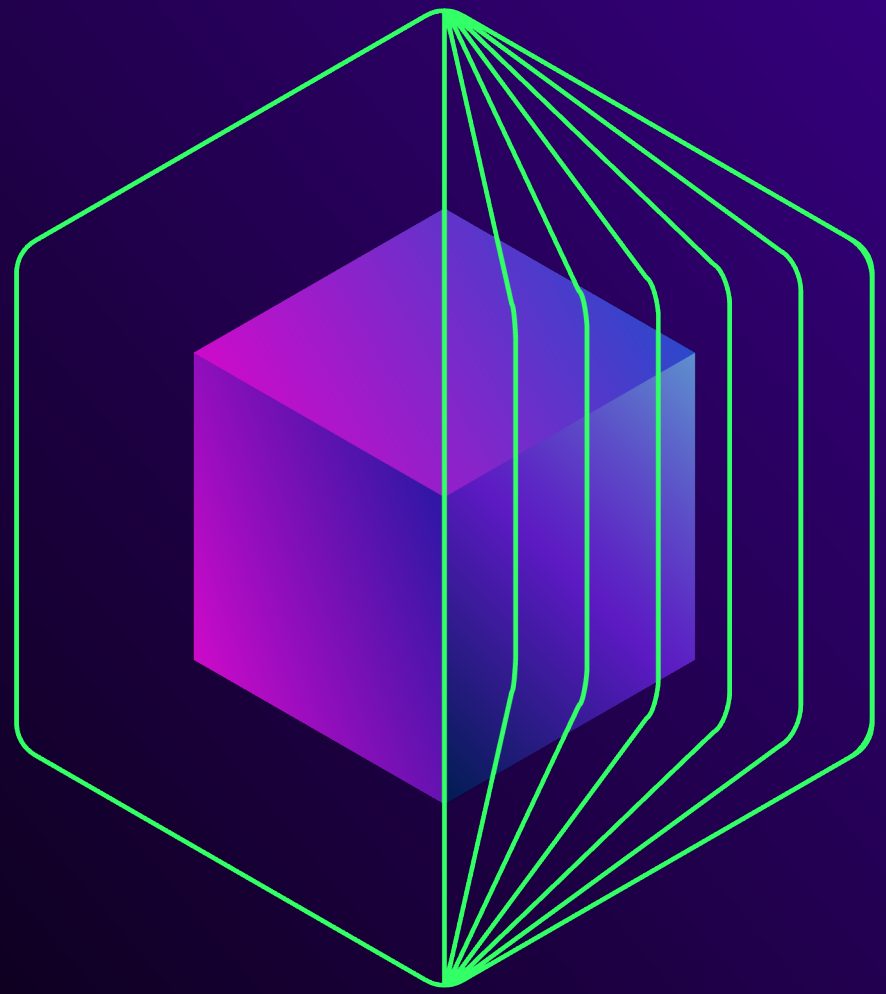
Predicting Properties of Nodes via Community-Aware Features

Sebastian Zając

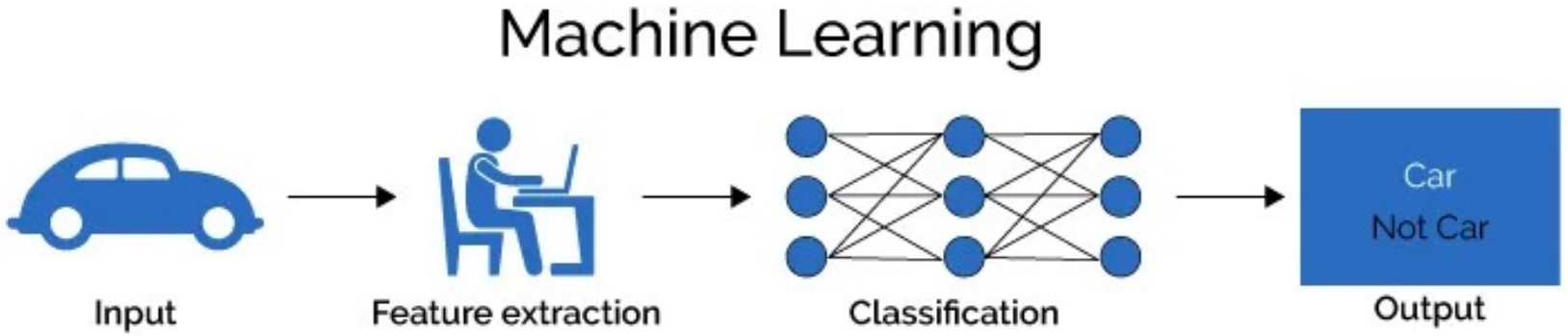


Toronto University - Online

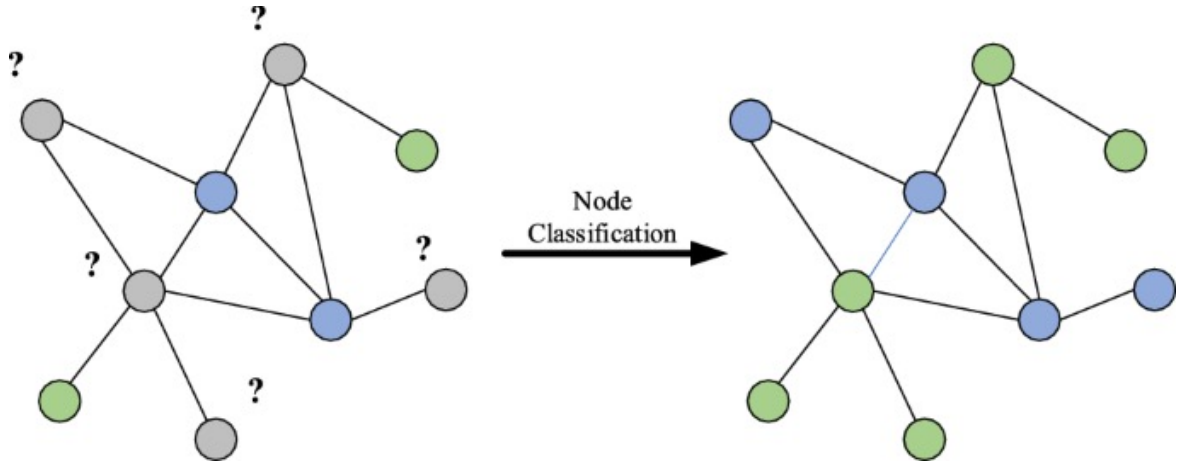
14.11.2023

www.sebastianzajac.pl

Motivation



Node Classification is an essential problem in which data is represented as a network, and the goal is to predict labels associated with its nodes.

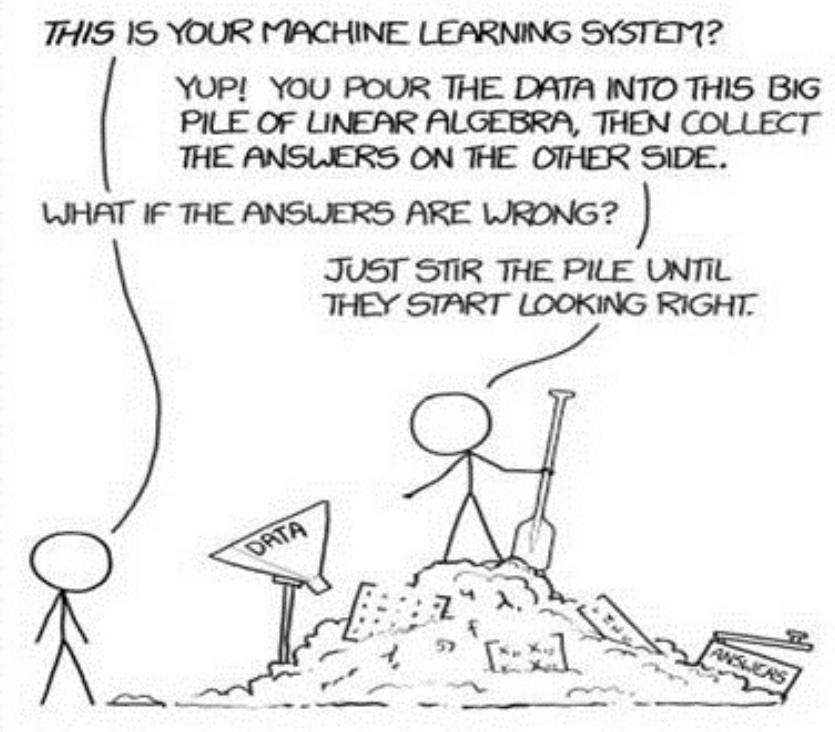


Practical applications:

1. Recommender systems
2. Social network analysis
3. Applied Chemistry
4. Many others

This work is done with **Bogumił Kamiński, Paweł Prałat**, and **Francois Theberge's** cooperation. arXiv:2311.04730

Motivation



No matter how sophisticated classifiers one builds, they will only perform well if they get informative input concerning the problem. We must have access to a set of highly informative node features that can discriminate representatives of different classes.

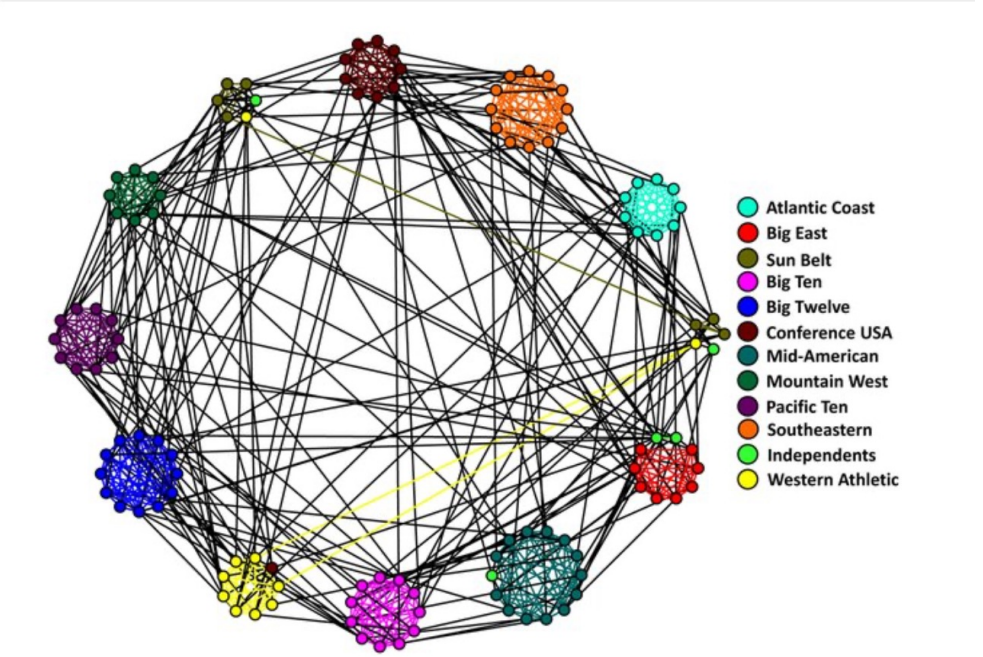
Community structure

We investigate a ***family of features that depend on the community structure*** often present in complex networks and play an essential role in their formation, affecting nodes' properties.

Such features are called ***community-aware features***.

The community structure of real-world networks often reveals the internal organization of nodes.
Identifying communities in a network can be done in an unsupervised way and is often the analysts' first step.

Such communities form groups of densely connected nodes with substantially fewer edges touching other parts of the graph.



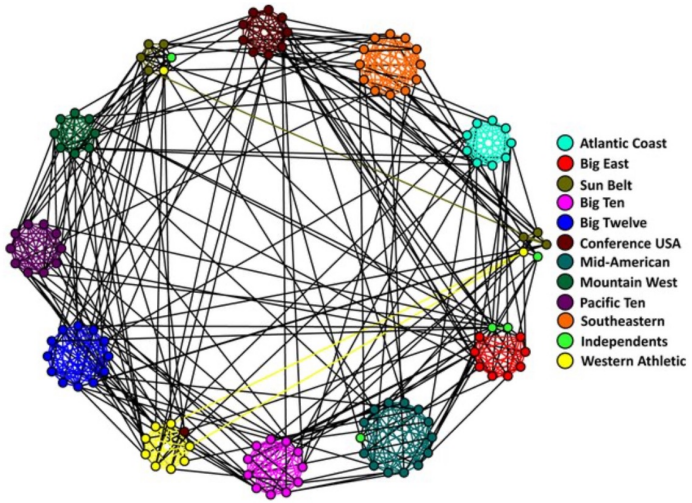
OUR goals:

1. Such features can be highly informative for many node classification tasks.
2. Community-aware features are not highly correlated to other features typically computed for networks.

Community structure

OUR goals:

1. Such features can be highly informative for many node classification tasks.
2. Community-aware features are not highly correlated to other features typically computed for networks.

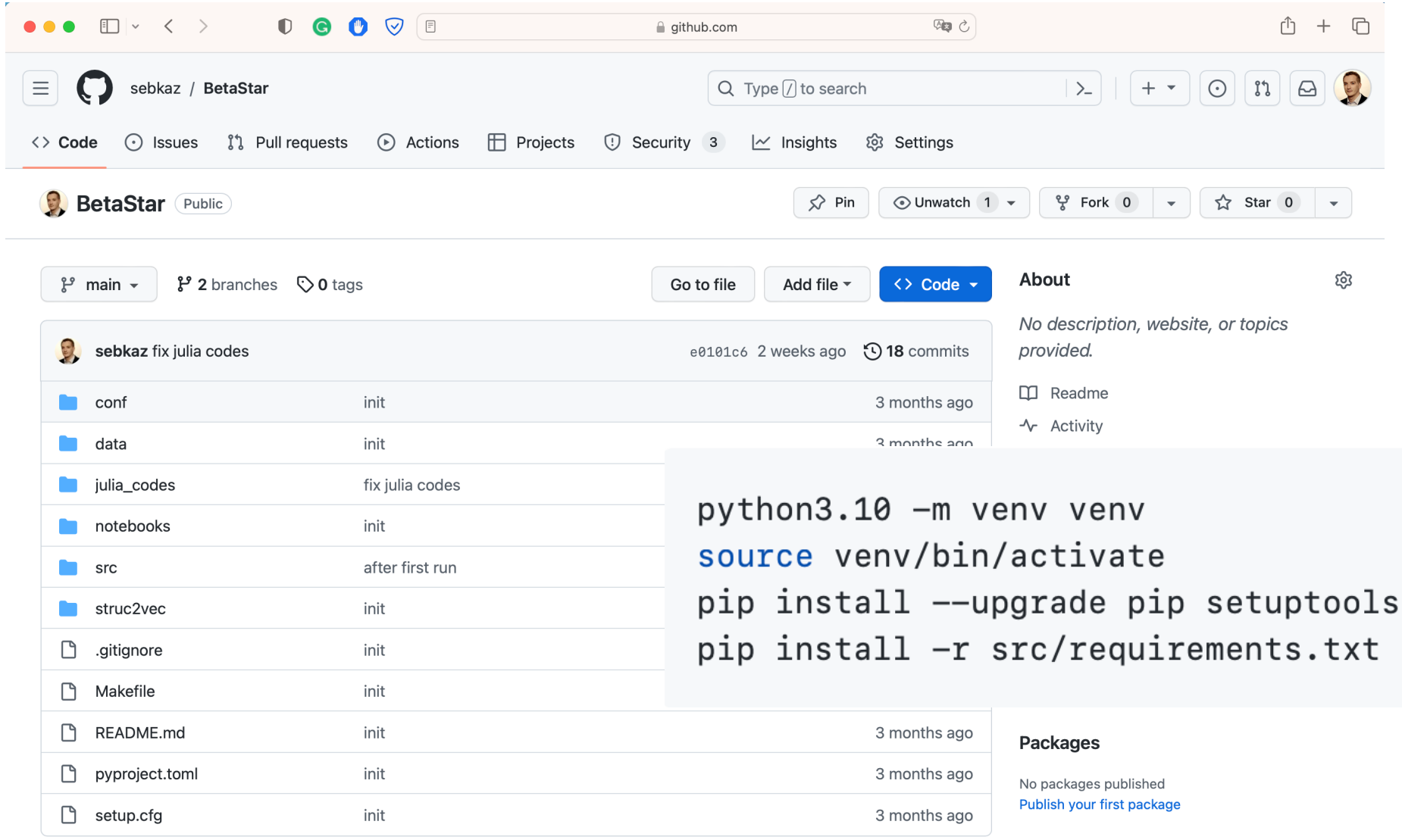


It might be essential whether a given node **is a strong community member** or it is loosely tied to many communities.

To compute community-aware features, we need **first to identify the community structure of a graph**.

This is a complicated non-linear transformation of the input graph, which **cannot be expected** to be quickly recovered **by supervised or unsupervised** ML or DL models.

Github repo: https://github.com/sebkaz/BetaStar

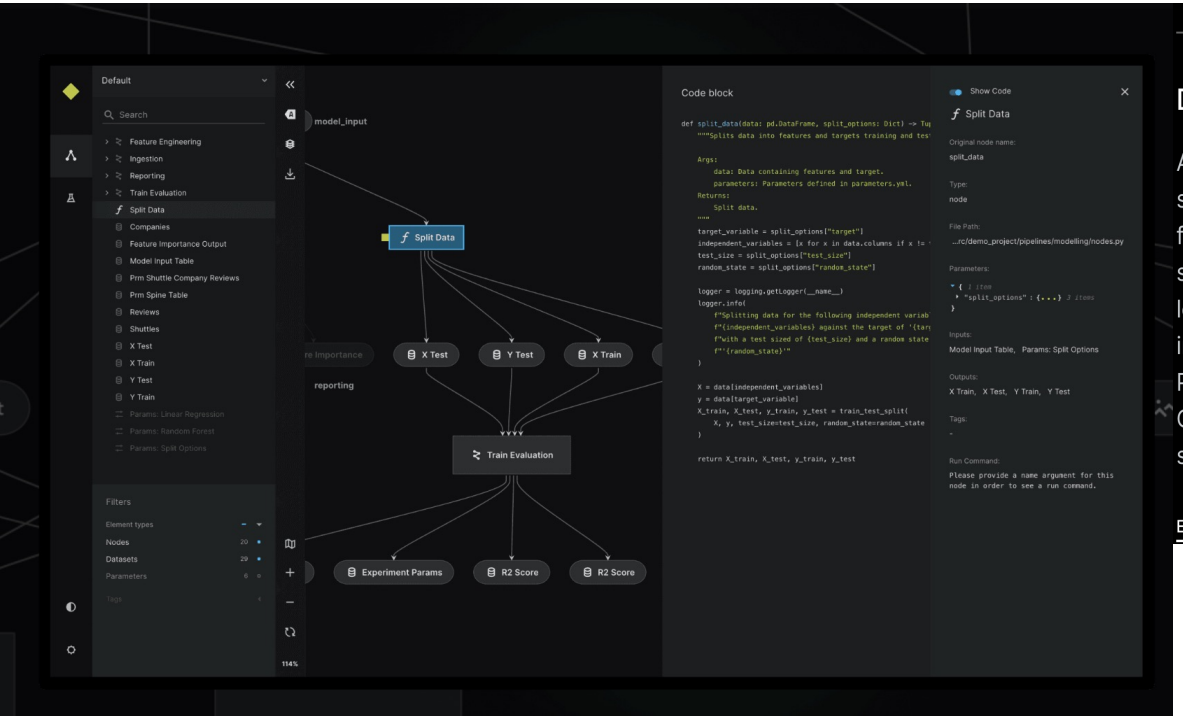



```
python3.10 -m venv venv
source venv/bin/activate
pip install --upgrade pip setuptools
pip install -r src/requirements.txt
```

Kedro Python

```
project-template     # Project folder
├── conf             # Configuration files
├── data             # Local project data
├── docs             # Documentation
├── logs             # Logs of pipeline runs
├── notebooks        # Exploratory Jupyter notebooks
├── pyproject.toml   # Project metadata and linter configuration
├── README.md        # README.md explaining your project
└── src              # Source code for pipelines
```

Code Standard – **Black, flake8, isort**
Automation Code Docs - **Sphinx**
Tests – **pytest**



— 01

Data Catalog

A series of lightweight data connectors used to save and load data across many different file formats and file systems. The Data Catalog supports S3, GCP, Azure, sFTP, DBFS, and local filesystems. Supported file formats include Pandas, Spark, Dask, NetworkX, Pickle, Plotly, Matplotlib, and many more. The Data Catalog also includes data and model snapshots for file-based systems.

Explore the data catalog

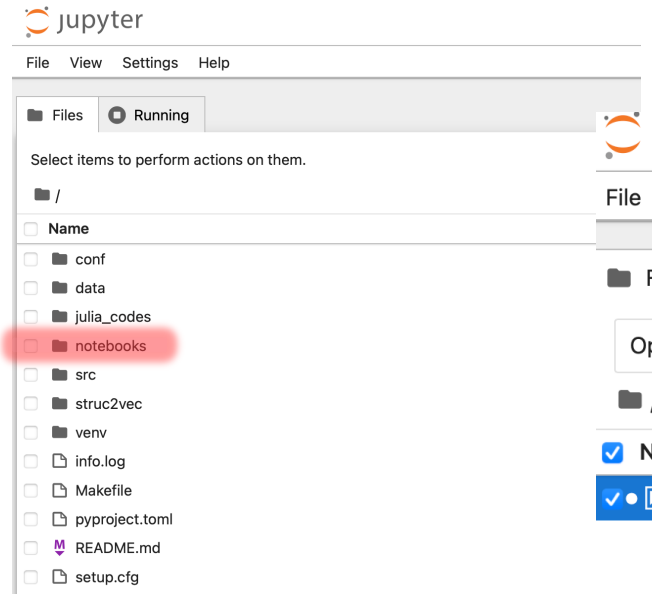
```yaml
# Load a Spark DataFrame on S3

flight_patterns:
  type: spark.SparkDataset
  filepath: s3a://your_bucket/data/01_raw/flight_patterns*
  credentials: dev_s3
  file_format: csv
```

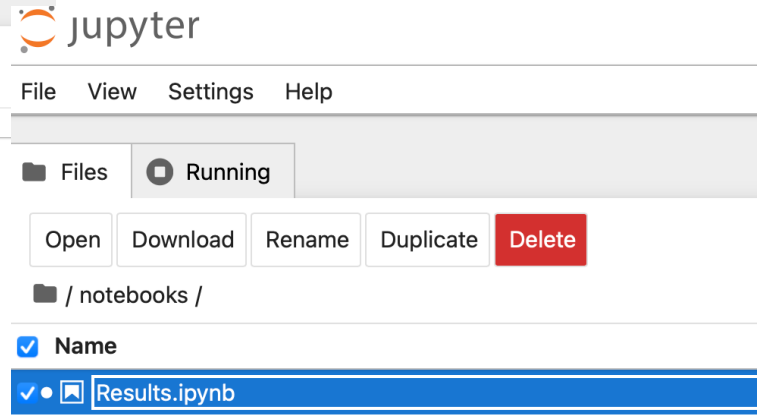
```yaml
# Save an image created with Matplotlib on Google Cloud Storage

results_plot:
  type: matplotlib.MatplotlibWriter
  filepath: gcs://your_bucket/data/08_results/plots/output_1.jpeg
  fs_args:
  project: my-project
  credentials: my_gcp_credentials
```

Jupyter notebooks
Easy integration with **Docker, Airflow** i **MLflow**

Work with Jupyter notebook

```
kedro jupyter notebook --env=x3_o1000 --port 8999

❯ cd Documents/GitHub/BetaStar
❯ ls
Makefile       data           pyproject.toml struc2vec
README.md      julia_codes    setup.cfg      venv
conf           notebooks      src
❯ source venv/bin/activate
❯ kedro info

 _            _
| | _____  __| |_ __ ___
| |/ / _ \/ _` | '__/ _ \
|   <  __/ (_| | | | (_) |
|_|\_\___|\__,_|_|  \___/
v0.18.11

Kedro is a Python framework for
creating reproducible, maintainable
and modular data science code.

No plugins installed
❯ kedro jupyter notebook --env=x3_o1000 --port 8999
/Users/air/Documents/GitHub/BetaStar/venv/bin/python3.10 -m jupyter notebook -
```





Kedro project

- you have access to `catalog`, `context` and `session` objects

```
[1]: session, catalog, context
```

```
[1]:
(
    <kedro.framework.session.session.KedroSession object at 0x107d00f10>,
    <kedro.io.data_catalog.DataCatalog object at 0x15ee83eb0>,
    <kedro.framework.context.context.KedroContext object at 0x10c402a40>
)
```

```
[1]: catalog.list()
```

```
[
    'graph',
    'data',
    'graph_pre',
    'graph_struct',
    'embedded_graph_node2vec',
    'embedded_graph_struc2vec',
    'model_data',
    'X_train_linear',
    'X_test_linear',
    'task1_predictions',
```

Fast data exploration and analysis

Kedro Data Catalog = Data in our project

We consider **undirected**, **connected**, and **simple graphs**. In each graph, we have some „ground-truth" labels.
We used two families of graphs:

Synthetic
Artificial Benchmark for Community Detection+o
Graphs

Empirical Graphs

We generated networks on:

N = 10,000 nodes with 1,000 of outliers.

The node's degree distr with a power-law with exponent $\gamma = 2.5$ and degrees between 5 and 500.

The community's distr with a power-law with exponent $\beta = 1.5$ and size range from 50 to 2,000.

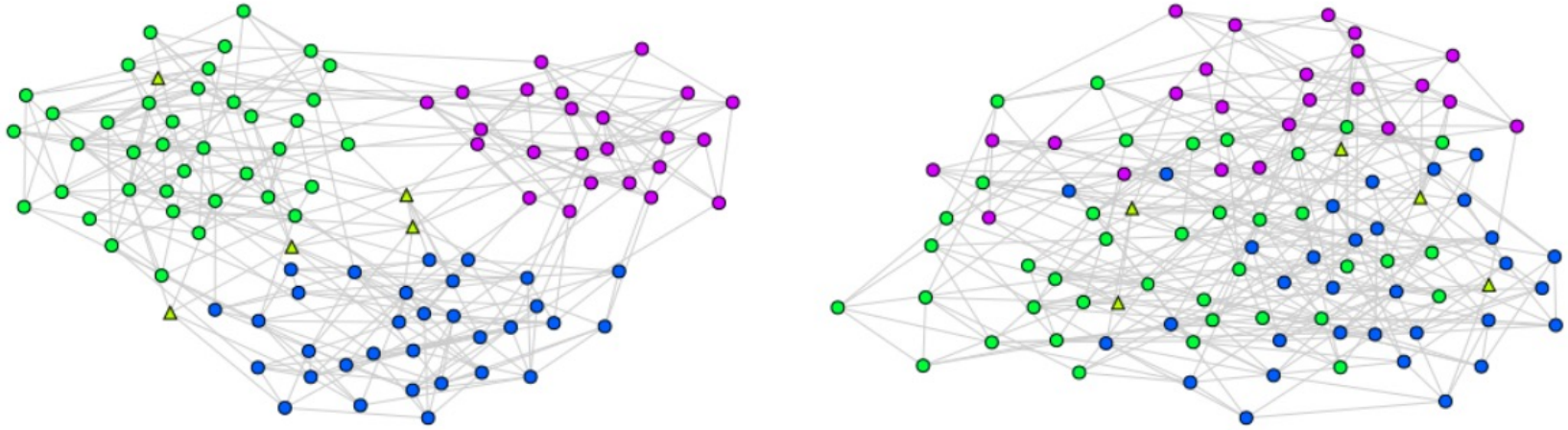
4 networks with different level of noise: $\xi \in \{0.3, 0.4, 0.5, 0.6\}$

Dataset	# nodes	Avg deg	# clusters	Target class prop
Reddit	10,980	14.3	12	3.66%
Grid	13,478	2.51	78	0.86%
LastFM	7,624	7.29	28	20.62%
Facebook	22,470	15.20	58	25.67%
Amazon	9,314	37.49	39	8.60%

Partition choosen as the best of 1,000 independent runs Leiden algo.

Kedro Data Catalog = Data in our project

The Artificial Benchmark for the Community Detection graph is a random model with community structure and power-law distribution for degrees and community sizes. It has been recently augmented to allow for the generation of outlier nodes (ABCD+o).



ABCD+o graphs with ($\xi = 0.2$, left) and ($\xi = 0.4$, right)

See also: ABCD graph generator in Julia programming language - `https://github.com/bkamins/ABCDGraphGenerator.jl`
B. Kamiński, P. Prałat, F. Théberge: „*Mining Complex Networks*", CRC Press (2022) or *Outliers in the ABCD Random Graph Model with Community Structure (ABCD+o)*.

Data preprocessing

Each graph is an independent **environment** in the Kedro project.

/ julia_codes /

Name
com_x3_o1000.dat
com_x4_o1000.dat
com_x5_o1000.dat
com_x6_o1000.dat
com_x7_o1000.dat
compute_stats.jl
edge_amazon.dat
edge_x3_o1000.dat
edge_x4_o1000.dat

Graph structure: data/01_raw/edges

All **features** with target: data/01_raw/data

Python Codes

1. Python/Julia graph - counter list
2. *Node2Vec*, *Struc2Vec* embeddings

Julia Codes

1. The Classical Node features
2. The Community-aware node features

/ ··· / 01_raw / edges /

Name
edge_amazon.dat
edge_facebook.dat
edge_grid.dat
edge_lastfm.dat
edge_reddit.dat
edge_x3.dat
edge_x4.dat
edge_x5.dat
edge_x6.dat

/ ··· / 01_raw / data /

Name
amazon.csv
facebook.csv
grid.csv
lastfm.csv
reddit.csv
x3.csv
x4.csv
x5.csv
x6.csv

jupyter catalog.yml Last Checkpoint: 22 days

File Edit View Settings Help

```yaml
# nodes and edges
graph:
  type: pandas.CSVDataSet
  filepath: data/01_raw/edges/edge_x3.dat
  load_args:
    engine: python
    sep: '\t'
    names: [in, out]

# community and non community features
data:
  type: pandas.CSVDataSet
  filepath: data/01_raw/data/x3.csv
```

Python Codes
1. Python/Julia graph - counter list
2. ***Node2Vec*, *Struc2Vec* embeddings**

Julia Codes
1. The Classical Node features
2. The Community-aware node features

Data preprocessing - Embeddings

Node2vec

Add languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

node2vec is an algorithm to generate vector representations of nodes on a graph. The *node2vec* framework learns low-dimensional representations for nodes in a graph through the use of random walks through a graph starting at a target node. It is useful for a variety of machine learning applications. Besides reducing the engineering effort, representations learned by the algorithm lead to greater predictive power.[1] *node2vec* follows the intuition that random walks through a graph can be treated like sentences in a corpus. Each node in a graph is treated like an individual word, and a random walk is treated as a sentence. By feeding these "sentences" into a skip-gram, or by using the continuous bag of words model paths found by random walks can be treated as sentences, and traditional data-mining techniques for documents can be used. The algorithm generalizes prior work which is based on rigid notions of network neighborhoods, and argues that the added flexibility in exploring neighborhoods is the key to learning richer representations of nodes in graphs.[2] The algorithm is considered one of the best graph classifiers.[3]

Struc2Vec

Params: dim=16; num-walks=10, walk-length=50, window-size=5, OPT1, OPT2, OPT3 set to true

jupyter parameters.yml

File Edit View Settings Help

```yaml
# parameters for node2vec
node2vec:
  dimensions: 16
  walk_length: 50
  num_walks: 10
  p: 1
  q: 1
  workers: 1
  seed: 123
```

Struc2vec

Add languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

struc2vec is a framework to generate node vector representations on a graph that preserve the structural identity.[1] In contrast to *node2vec*, that optimizes node embeddings so that nearby nodes in the graph have similar embedding, *struc2vec* captures the **roles** of nodes in a graph, even if structurally similar nodes are far apart in the graph. It learns low-dimensional representations for nodes in a graph, generating random walks through a constructed multi-layer graph starting at each graph node. It is useful for machine learning applications where the downstream application is more related with the structural equivalence of the nodes (e.g., it can be used to detect nodes in networks with similar functions, such as interns in the social network of a corporation). *struc2vec* identifies nodes that play a similar role based solely on the structure of the graph, for example computing the structural identity of individuals in social networks.[2] In particular, *struc2vec* employs a degree-based method to measure the pairwise structural role similarity, which is then adopted to build the multi-layer graph. Moreover, the distance between the latent representation of nodes is strongly correlated to their structural similarity. The framework contains three optimizations: reducing the length of degree sequences considered, reducing the number of pairwise similarity calculations, and reducing the number of layers in the generated graph.

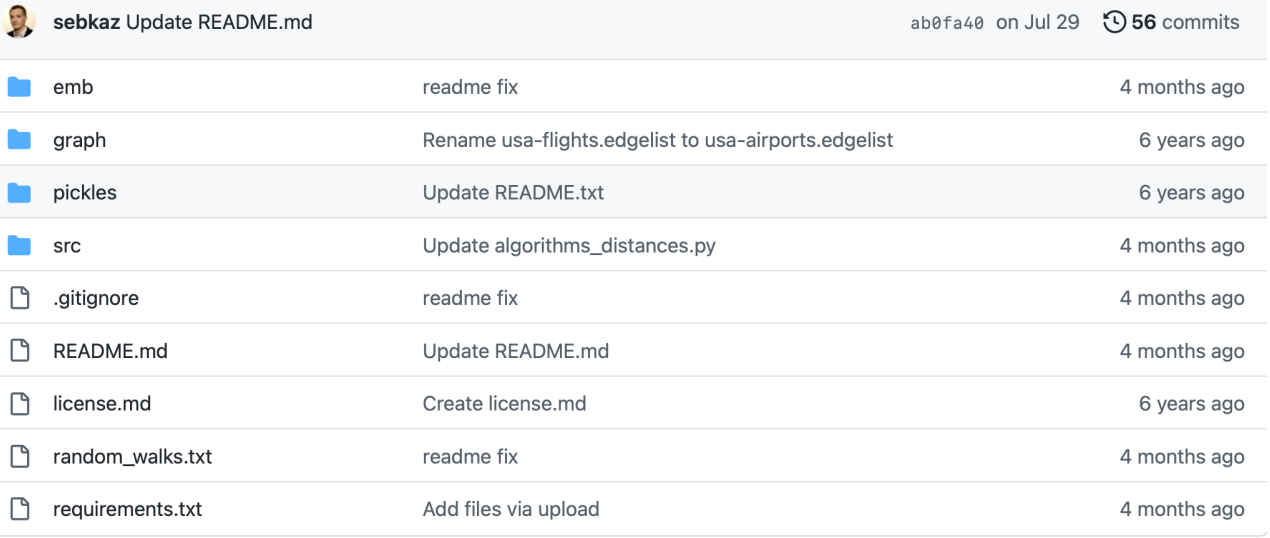
struc2vec follows the intuition that random walks through a graph can be treated as sentences in a corpus. Each node in a graph is treated as an individual word, and short random walk is treated as a sentence. In its final phase, the algorithm employs Gensim's *word2vec* algorithm to learn embeddings based on biased random walks.[3] Sequences of nodes are fed into a skip-gram or continuous bag of words model and traditional machine-learning techniques for classification can be used.[4] It is considered a useful framework to learn node embeddings based on structural equivalence.

Python Codes
1. Python/Julia graph - counter list
2. ***Node2Vec*, *Struc2Vec* embeddings**

Julia Codes
1. The Classical Node features
2. The Community-aware node features

Struc2vec - Embeddings

https://github.com/sebkaz/struc2vec/tree/master



sebkaz Update README.md — `ab0fa40` on Jul 29 — **56** commits

emb	readme fix	4 months ago
graph	Rename usa-flights.edgelist to usa-airports.edgelist	6 years ago
pickles	Update README.txt	6 years ago
src	Update algorithms_distances.py	4 months ago
.gitignore	readme fix	4 months ago
README.md	Update README.md	4 months ago
license.md	Create license.md	6 years ago
random_walks.txt	readme fix	4 months ago
requirements.txt	Add files via upload	4 months ago

README.md

struc2vec

This repository provides a reference implementation of *struc2vec* as described in the paper:

> struc2vec: Learning Node Representations from Structural Identity.
> Leonardo F. R. Ribeiro, Pedro H. P. Saverese, Daniel R. Figueiredo.
> Knowledge Discovery and Data Mining, SigKDD, 2017.

The *struc2vec* algorithm learns continuous representations for nodes in any graph. *struc2vec* captures structural equivalence between nodes.

Before executing *struc2vec*, it is necessary to install the packages from the `requirements.txt` file:

Python Codes
1. Python/Julia graph - counter list
2. *Node2Vec*, *Struc2Vec* embeddings

Julia Codes
1. The Classical Node features
2. The Community-aware node features

The Classical node features

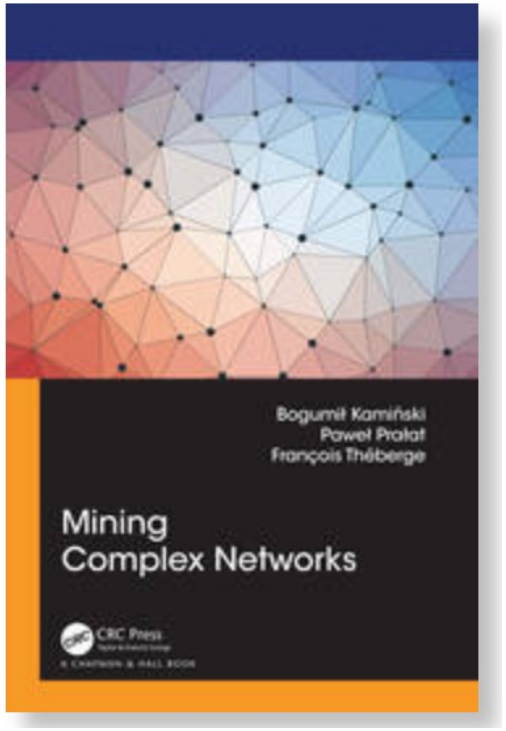


Table 3: Classical (non-community-aware) node features that are used in our experiments.

abbreviation	name	reference
`lcc`	local clustering coefficient	[50]
`bc`	betweenness centrality	[17]
`cc`	closeness centrality	[2]
`dc`	degree centrality	[30]
`ndc`	average degree centrality of neighbours	[1]
`ec`	eigenvector centrality	[6]
`eccen`	node eccentricity	[8]
`core`	node coreness	[30]
`n2v`	16-dimensional `node2vec` embedding	[19]
`s2v`	16-dimensional `struc2vec` embedding	[45]

Python Codes
1. Python/Julia graph - counter list
2. *Node2Vec*, *Struc2Vec* embeddings

Julia Codes
1. The Classical Node features
2. **The Community-aware node features**

Null Models to understand Community Structure

A null model is a random object that matches a property P observed in some dataset.

A degree distribution in a graph.

Using null models as a reference is a flexible approach for statistically testing the presence of properties.

As a result, **the null models can be used to test whether a given object exhibits some „strange”** property that is not expected based on chance alone.

A classical application of null models is frequentist hypothesis testing in statistics.

In network science – used to build various machine learning tools:

clustering algorithms or **unsupervised frameworks to evaluate node embeddings**

We consider null models for:

- extracting graph community structure
- quantify how tightly nodes are connected to the comms that surround them.

Null Models to understand Community Structure

To illustrate how it works, we define **the modularity function** that is key ingredient in ***Leiden***, the clustering algorithm we use to extract community structure.

Our community-aware features works also for other methods of community detection.

DEFINITIONS:

Let $G = (V, E)$ be **a simple unweighted graph**. V – set of nodes, E - set of edges.

Given a subset of nodes $A \subseteq V$ we define **the numer of edges in the graph induced by this set** as:

$$e(A) = |\{e \in E : e \subseteq A| \qquad\qquad e(V) = |E|$$

For each node $v \in V$ we define its **degree** as the numer of egdes that contain v : $\deg(v) = |e \in E : v \in e\}|$

Sum of degrees of nodes in A : $vol(A) = \sum_{v \in A} \deg(v)$

$A = \{A_1, A_2, \dots, A_l\}$ is **a partition of V** into l sets.

The number of neighbours of v in A_i : $\deg(A_i) = |N(v) \cap A_i|$.

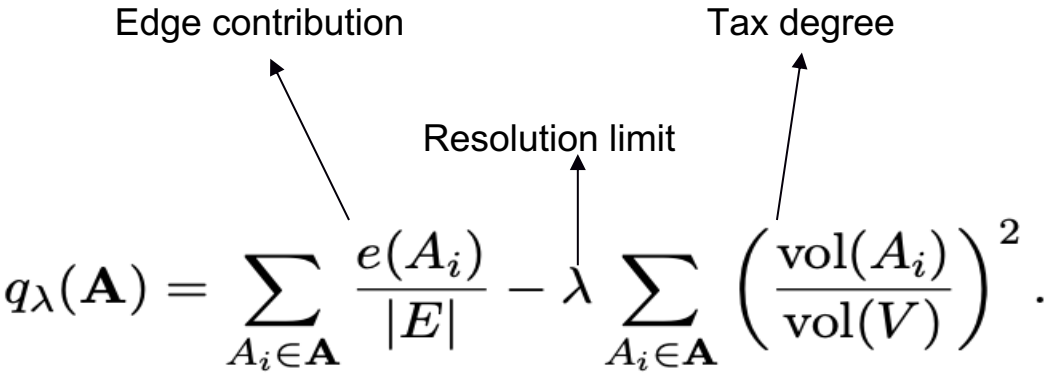
Python Codes
1. Python/Julia graph - counter list
2. *Node2Vec*, *Struc2Vec* embeddings

Julia Codes
1. The Classical Node features
2. **The Community-aware node features**

Null Models to understand Community Structure

MORE MORE DEFINITIONS:

Modularity function



Edge contribution:

Fraction of edges captured within communities in partition A.

Tax degree:

The expected fraction of edges that do the same in the corresponding Chung-Lu null-model.

Many popular clustering algorithms, such **as Louvain and Leiden**, use it as a quality function that performs well.

It also provides an easy way to **measure the presence of community structure** in a network.

If q(G) is close to 1 - a strong community structure;

if q(G) is close to 0 - there is no community structure.

Community - aware node features - literature

MORE MORE MORE DEFINITIONS:

ANOMALY SCORE (CADA)

$$\mathrm{cd}(v) = \frac{\deg(v)}{d_{\mathbf{A}}(v)}, \qquad \text{where} \qquad d_{\mathbf{A}}(v) = \max\left\{ \deg_{A_i}(v) : A_i \in \mathbf{A} \right\};$$

Represents the maximum number of neighboring nodes that belong to the same community.

If **all neighbors** of v belong to the same community, then $cd(v) = 1$.

If **no two neighbors** of v belong to the same community, then $cd(v) = \deg(v)$

NORMALIZED ANOMALY SCORE

$$\overline{\mathrm{cd}}(v) = \frac{\deg_{A_i}(v)}{\deg(v)}.$$

T. J. Helling et all. A community-aware approach for identifying node anomalies in complex networks. (2018)

Normalized Within-module Degree and Participation Coefficient

MORE MORE MORE MORE DEFINITIONS:

Normalized within – module degree

$$z(v) = \frac{\deg_{A_i}(v) - \mu(v)}{\sigma(v)},$$

Where $\mu(v)$ is the mean, and σ(v) is the standard deviation of $\deg_{A_i}(u)$.

Captures how strongly a particular node is connected to other nodes within its community, completely ignoring edges between communities.

Participation Coefficient:

$$p(v) = 1 - \sum_{i=1}^{\ell} \left(\frac{\deg_{A_i}(v)}{\deg(v)} \right)^2 .$$

R. Guimera, Luís A Nunes Amaral „ Functional cartography of complex metabolic networks. nature, 433(7028):895–900, 2005.

New community – aware node features

MORE MORE MORE MORE MORE DEFINITIONS:

We want to create features that consider the distribution of community size.

Suppose that we aim to adjust the (modified) modularity function **to detect nodes that are outliers**.

If the fraction of neighbors of node v in its (own) community is small relative to the corresponding expected fraction under the null model, then we will say that v is likely **to be an outlier.**

OUR GOAL – modified modularity function that nodes likely to be outliers are put into a single node community.

$$q_{\lambda,\beta}(\mathbf{A}) = \sum_{A_i \in \mathbf{A}} \frac{e(A_i) + \delta_{|A_i|=1}\beta \,\mathrm{vol}(A_i)/2}{|E| + Z/2} - \lambda \left(\sum_{A_i \in \mathbf{A}} \left(\frac{\mathrm{vol}(A_i)(1 + \delta_{|A_i|=1}\beta)}{\mathrm{vol}(V) + Z} \right)^2 \right),$$

For $\beta = 0$ we get the traditional modularity function.

New community – aware node features

MORE MORE MORE MORE DEFINITIONS:

It is easier to use the following approx :

$$q_{\lambda,\beta}(\mathbf{A}) \approx \sum_{A_i \in \mathbf{A}} \frac{e(A_i) + \delta_{|A_i|=1}\beta \mathrm{vol}(A_i)/2}{|E|} - \lambda \left(\sum_{A_i \in \mathbf{A}} \left(\frac{\mathrm{vol}(A_i)}{\mathrm{vol}(V)} \right)^2 \right).$$

Now we can ask:

What is the **threshold value** of $\beta^*(v)$ so that $\beta > \beta^*(v)$ then the approx of the regularized modularity function increases if v is moved from A_i to form its own (single node) community.

COMMUNITY ASSOCIATION STRENGTH:

$$\beta^*(v) = 2 \left(\frac{\deg_{A_i}(v)}{\deg(v)} - \lambda \frac{\mathrm{vol}(A_i) - \deg(v)}{\mathrm{vol}(V)} \right).$$

New community – aware node features

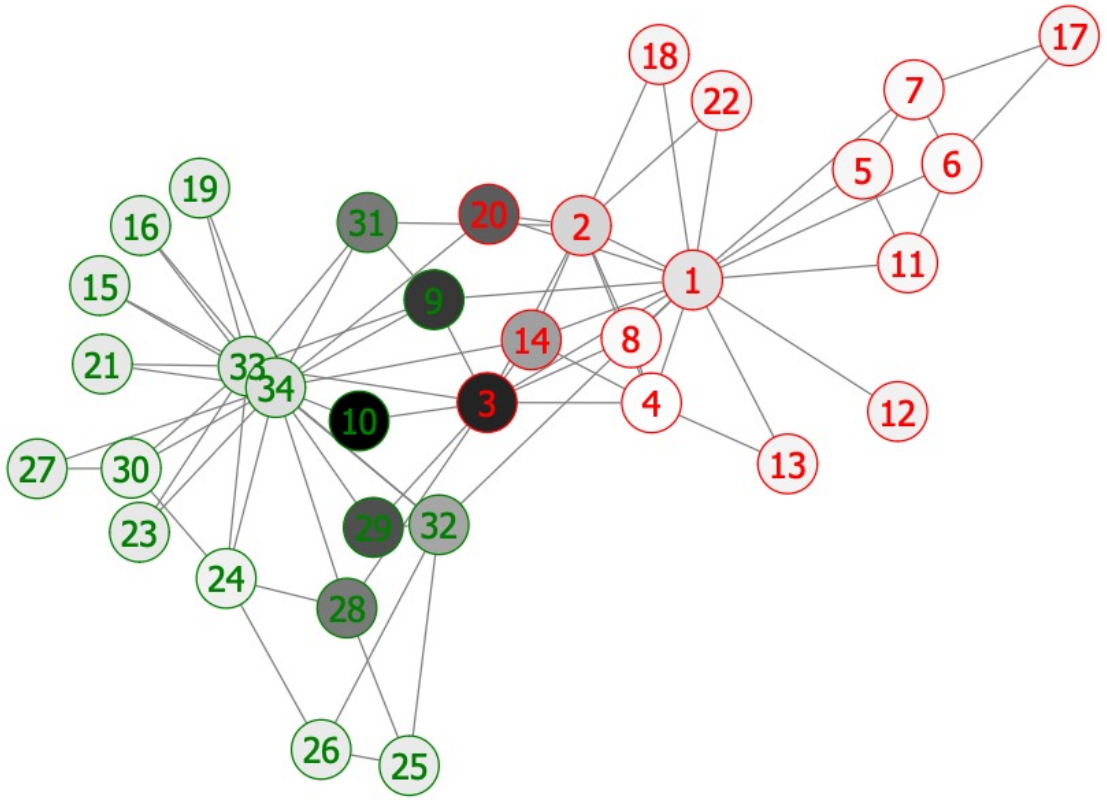


Figure 1: Communities (red and green colours) in the Karate graph. The shades of nodes correspond to their values of $\beta^*(v)$ (darker colours indicate lower values).

MORE New community – aware node features

Similarity:

Upgraded versions of the participation coefficient aim to measure how neighbors of node v are distributed between all parts of partition A.

Difference:

Pay attention to the sizes of parts of A and compare the distribution of neighbours to the corresponding predictions from the null model.

Let's vectorize our nodes and communities!

$$q_1(v) = \left(\frac{\deg_{A_1}(v)}{\deg(v)}, \frac{\deg_{A_2}(v)}{\deg(v)}, \ldots, \frac{\deg_{A_\ell}(v)}{\deg(v)} \right)$$

$$\hat{q}_1(v) = \left(\frac{\mathrm{vol}(A_1)}{\mathrm{vol}(V)}, \frac{\mathrm{vol}(A_2)}{\mathrm{vol}(V)}, \ldots, \frac{\mathrm{vol}(A_\ell)}{\mathrm{vol}(V)} \right) =: \hat{q}_1$$

How we can compare two vectors?

MORE New community – aware node features

- L^1 *norm*: $L_1^1(v) = \sum_{i=1}^{\ell} \left| \frac{\deg_{A_i}(v)}{\deg(v)} - \frac{\mathrm{vol}(A_i)}{\mathrm{vol}(V)} \right|$
- L^2 *norm*: $L_1^2(v) = \left(\sum_{i=1}^{\ell} \left(\frac{\deg_{A_i}(v)}{\deg(v)} - \frac{\mathrm{vol}(A_i)}{\mathrm{vol}(V)} \right)^2 \right)^{1/2}$
- *Kullback–Leibler divergence* [11]: $\mathrm{kl}_1(v) = \sum_{i=1}^{\ell} \frac{\deg_{A_i}(v)}{\deg(v)} \log \left(\frac{\deg_{A_i}(v)}{\deg(v)} \cdot \frac{\mathrm{vol}(V)}{\mathrm{vol}(A_i)} \right)$
- *Hellinger distance* [24]: $h_1(v) = \frac{1}{\sqrt{2}} \left(\sum_{i=1}^{\ell} \left(\left(\frac{\deg_{A_i}(v)}{\deg(v)} \right)^{1/2} - \left(\frac{\mathrm{vol}(A_i)}{\mathrm{vol}(V)} \right)^{1/2} \right)^2 \right)^{1/2}$

The Community-aware node features

Table 2: Community-aware node features used in our experiments. A combination of `WMD` and `CPC` is also used as a 2-dimensional embedding of a graph (`WMD+CPC`).

abbreviation	symbol	name	subsection
`CADA`	$\mathrm{cd}(v)$	anomaly score CADA	3.1
`CADA*`	$\overline{\mathrm{cd}}(v)$	normalized anomaly score	3.1
`WMD`	$z(v)$	normalized within-module degree	3.2
`CPC`	$p(v)$	participation coefficient	3.2
`CAS`	$\beta^*(v)$	community association strength	3.3
`CD_L11`	$L_1^1(v)$	L^1 norm for the 1st neighbourhood	3.4
`CD_L21`	$L_1^2(v)$	L^2 norm for the 1st neighbourhood	3.4
`CD_KL1`	$\mathrm{kl}_1(v)$	Kullback–Leibler divergence for the 1st neighbourhood	3.4
`CD_HD1`	$h_1(v)$	Hellinger distance for the 1st neighbourhood	3.4
`CD_L12`	$L_2^1(v)$	L^1 norm for the 2nd neighbourhood	3.4
`CD_L22`	$L_2^2(v)$	L^2 norm for the 2nd neighbourhood	3.4
`CD_KL2`	$\mathrm{kl}_2(v)$	Kullback–Leibler divergence for the 2nd neighbourhood	3.4
`CD_HD2`	$h_2(v)$	Hellinger distance for the 2nd neighbourhood	3.4

Experiment 1

https://github.com/sebkaz/BetaStar

This experiment aims to show that classical features cannot entirely explain community-aware features.

Models: Regression kind

Linear regression, Ridge regression, Random forest, XGBoost, Lightgbm.

Measures:

Kendall correlation, spearnman correlation, and R^2 score.

The conclusion is that it is worth including such features in predictive models as they could improve their predictive power. However, this additional information could be noise and not valuable for practice.

Experiment 1. Results

ABCD+o
max Kendall correlation

target	$\xi = 0.3$	$\xi = 0.4$	$\xi = 0.5$	$\xi = 0.6$
CADA	0.3305	0.2541	0.2292	0.1766
CADA*	0.3613	0.2877	0.2772	0.1713
CPC	0.3540	0.3568	0.3231	0.3106
CAS	0.4205	0.3584	0.3138	0.2167
CD_L21	0.4539	0.4043	0.3823	0.3313
CD_L22	0.6265	0.5589	0.5009	0.4492
CD_L11	0.5935	0.5571	0.5834	0.5648
CD_L12	0.6503	0.5799	0.5464	0.5188
CD_KL1	0.6991	0.6411	0.5918	0.4929
CD_HD1	0.6809	0.6334	0.6170	0.5584
CD_KL2	0.7453	0.6602	0.6090	0.5471
CD_HD2	0.7546	0.7119	0.6815	0.6352
WMD	0.7670	0.7288	0.6915	0.6387

Empirical Graphs
max Kendall correlation

target	Amazon	Facebook	Grid	LastFM	Reddit
CADA	0.5830	0.5666	0.2156	0.4815	0.6826
CADA*	0.6058	0.5828	0.2174	0.5058	0.6867
CPC	0.6338	0.5992	0.2193	0.5175	0.7193
CAS	0.6538	0.6257	0.2999	0.5594	0.7306
CD_L21	0.7052	0.6464	0.3496	0.5698	0.7574
CD_L22	0.7554	0.7355	0.3557	0.6295	0.7941
CD_L11	0.7251	0.7041	0.6978	0.6220	0.7735
CD_L12	0.7794	0.7785	0.6447	0.6884	0.7810
CD_KL1	0.7176	0.7516	0.7394	0.6289	0.7755
CD_HD1	0.7383	0.7482	0.7168	0.6459	0.7853
CD_KL2	0.7706	0.7826	0.7292	0.6853	0.8097
CD_HD2	0.8212	0.8173	0.6930	0.7369	0.8221
WMD	0.8447	0.8456	0.8488	0.8531	0.7638

Experiment 2

Target: Verify the usefulness of the community-aware features for the node classification task. For each graph, we build a single model predicting the target variable.

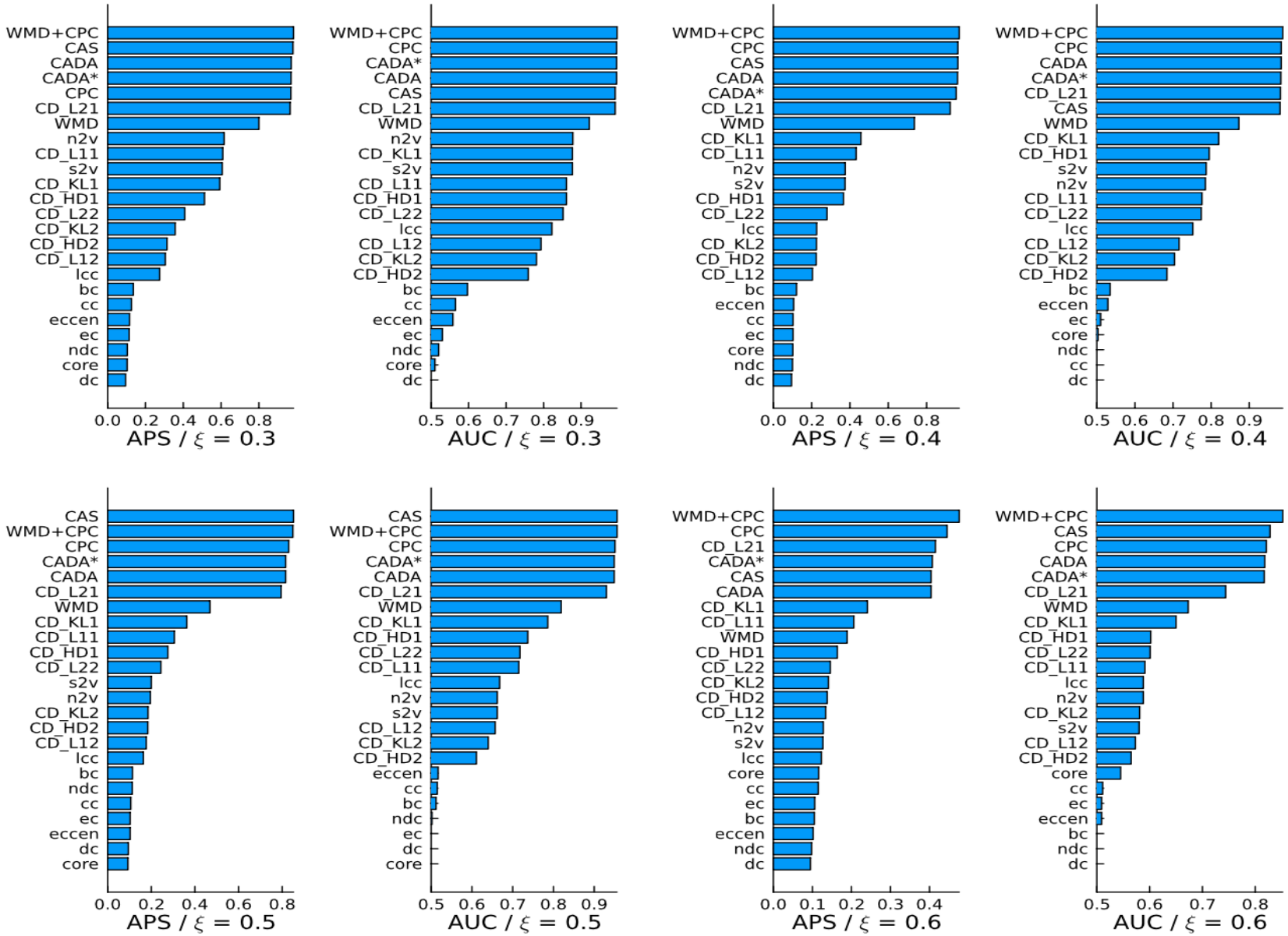
Models: Classification kind

Logistic regression, Random forest, XGBoost, Lightgbm.

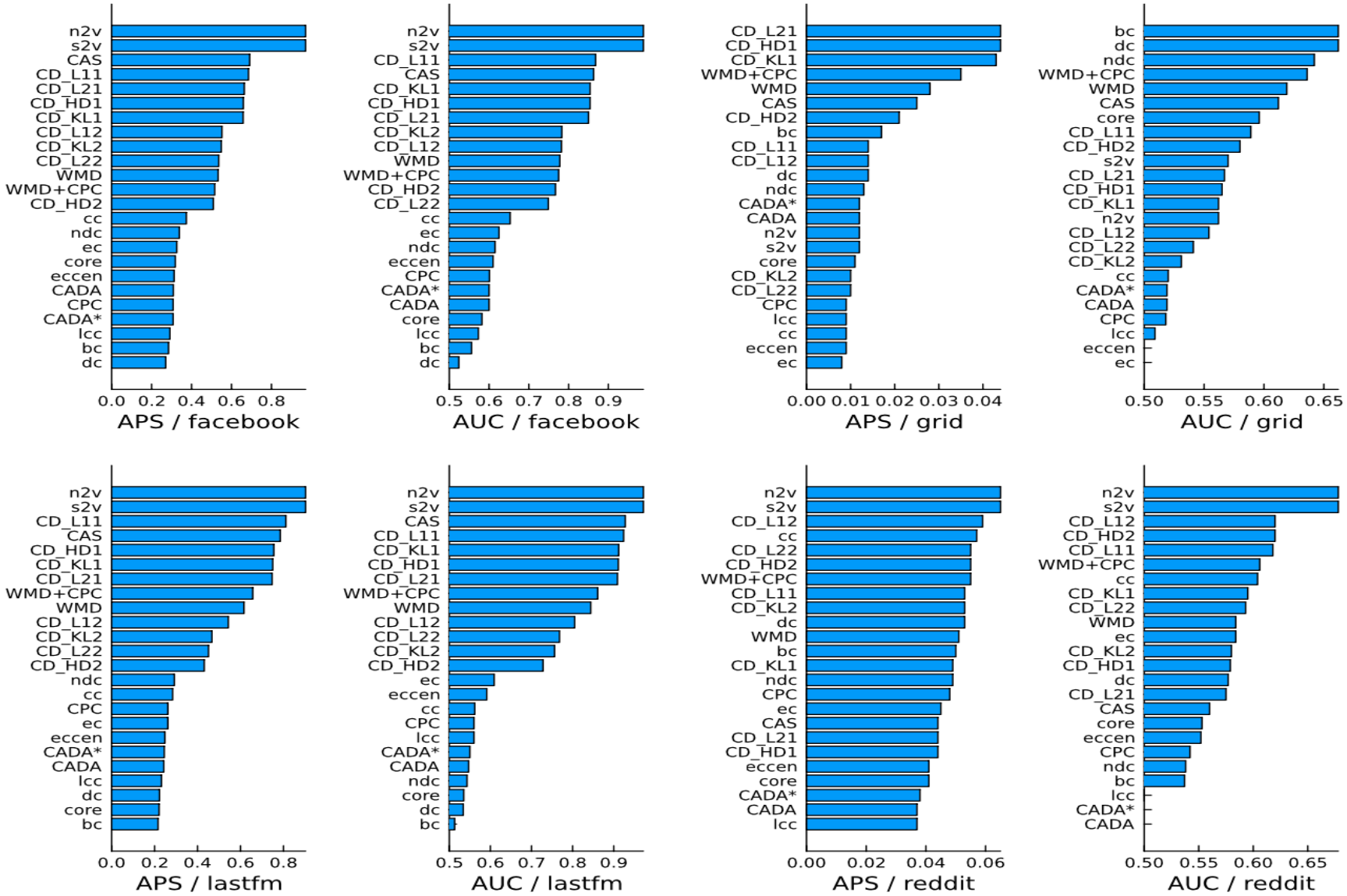
Measures:

ROC AUC score, Average Precision Score.

Experiment 2. Results ABCD+o



Experiment 2. Results Empirical Graphs



The Summary

1. Community-aware features are helpful for the prediction of labels of nodes.

   - We confirmed this hypothesis on synthetic graphs in which community-aware features outperformed other classical node features.
   - on empirical graphs, community-aware features were only sometimes the most important. But still, it is recommended to include them in predictive models.

2. Community-aware features have a relatively low computational complexity compared to many classical features or node embeddings.

3. They are easily interpretable.

4. The new measures proposed in this paper (CAS and distribution-based ones) generally perform better than community-aware features offered earlier in the literature.