fuzzyops

Выпуск 1.0.4

Ilya, Nikita, Maxim

нояб. 21, 2024

Содержание:

1 Раздел для работы с нечеткими аналитическими сетями

`class fuzzyops.fan.fan.Edge(`*start_node:* Node, *end_node:* Node, *weight: float*`)`

Базовые классы: `object`

Представляет ребро в графе.

`start_node`

Начальный узел ребра.

Type
Node

`end_node`

Конечный узел ребра.

Type
Node

`weight`

Вес ребра, представляющий его степень осуществимости.

Type
float

Параметры

- `start_node` (`Node`) – Начальный узел ребра.
- `end_node` (`Node`) – Конечный узел ребра.
- `weight` (*float*) – Вес ребра, представляющий его степень осуществимости.

`__init__(start_node`

Node, end_node: Node, weight: float): Инициализирует ребро.

`class fuzzyops.fan.fan.Graph`

Базовые классы: `object`

Представляет направленный граф. Алгоритм реализован по статье https://cyberleninka.ru/article/n/nechetkaya-alternativnaya-setevaya-model-analiza-i-planirovaniya-proekta-v-usloviyah-neopredelennosti

`nodes`

Словарь узлов в графе.

Type
dict

`edges`

Список рёбер в графе.

Type
List[*Edge*]

`add_node(node_name`

str) -> Node: Добавляет узел в граф.

`add_edge(start_node_name`

str, end_node_name: str, weight: float) -> None: Добавляет ребро в граф.

`get_paths_from_to(start_node_name`

str, end_node_name: str) -> List[Node]: Возвращает все возможные пути от начального узла к конечному узлу.

`calculate_path_fuzziness(path`

List[Node]) -> float: Вычисляет значение нечеткости данного пути.

`find_most_feasible_path(start_node_name`

str, end_node_name: str) -> List[str]: Находит наиболее осуществимый путь между двумя узлами.

`macro_algorithm_for_best_alternative()` → List[str] | float

Выполняет макроалгоритм для определения наилучшей альтернативы.

`add_edge(`*start_node_name:* Node, *end_node_name:* Node, *weight: float*`)` → None

Добавляет ребро в граф.

Параметры

- `start_node_name` (Node) – Имя начального узла.
- `end_node_name` (Node) – Имя конечного узла.
- `weight` (*float*) – Вес

`add_node(`*node_name: str*`)` → *Node*

Добавляет узел в граф и возвращает его.

Параметры

`node_name` (*str*) – Имя начального узла.

`calculate_path_fuzziness(`*path: List[*Node*]*`)` → float

Вычисляет нечеткость заданного пути.

Параметры

`path` (*List [*Node*]*) – Путь, представленный в виде списка имён узлов.

Результат

Оценка нечеткости пути.

Тип результата

float

Исключение

`ValueError` – Если путь недействителен (т.е. между узлами нет рёбер).

`find_most_feasible_path(`*start_node_name:* Node, *end_node_name:* Node`)` → List[str]

Находит наиболее осуществимый путь между двумя узлами на основе нечеткости.

Параметры

- `start_node_name` (Node) – Имя начального узла.
- `end_node_name` (Node) – Имя конечного узла.

Результат

Наиболее осуществимый путь, представленный в виде списка имён узлов.

Тип результата

List[str]

`get_paths_from_to(`*start_node_name:* Node, *end_node_name:* Node`)` → List[*Node*]

Возвращает список всех возможных путей от начального узла до конечного узла.

Параметры

- `start_node_name` (Node) – Имя начального узла.
- `end_node_name` (Node) – Имя конечного узла.

Результат
Список путей, представляющих собой списки имён узлов.

Тип результата
List[*Node*]

`macro_algorithm_for_best_alternative()` → List[str] | float

Выполняет макроалгоритм для определения наилучшей альтернативы в сетевой модели.

Результат
Наилучший альтернативный путь и его оценка осуществимости.

Тип результата
Union[List[str], float]

`class fuzzyops.fan.fan.Node(`*name: str*`)`

Базовые классы: `object`

Представляет узел в графе.

`name`

Имя узла.

Type
str

`in_edges`

Список входящих рёбер для этого узла.

Type
List[*Edge*]

`out_edges`

Список исходящих рёбер из этого узла.

Type
List[*Edge*]

Параметры
`name` (*str*) – Имя узла.

`add_in_edge(edge`

Edge) -> None: Добавляет входящее ребро к узлу.

`add_out_edge(edge`

Edge) -> None: Добавляет исходящее ребро из узла.

`add_in_edge(`*edge: ['Edge']*`)` → None

Добавляет входящее ребро к узлу.

Параметры
`edge` (`Edge`) – Ребро

`add_out_edge(`*edge: ['Edge']*`)` → None

Добавляет исходящее ребро из узла.

Параметры
`edge` (`Edge`) – Ребро

fuzzyops.fan.fan.calc_final_scores(*f_ nums: List[FuzzyNumber]*) → float

Вычисляет итоговую оценку из списка нечетких чисел.

Параметры
f_nums (*List [FuzzyNumber]*) – Список нечетких чисел.

Результат
Результат дефаззификации нечетких чисел.

Тип результата
float

2 Раздел для работы с нечеткой логикой

class fuzzyops.fuzzy_logic.base_rules.BaseRule(*A: Domain*, *B: float | int | str*)

Базовые классы: object

Базовое правило для нечеткой логики, содержащее нечеткое множество и значение.

A

Нечеткое множество, которое используется в правиле.

Type
Domain

B

Значение, ассоциированное с правилом.

Type
Union[float, int, str]

A: Domain

B: float | int | str

class fuzzyops.fuzzy_logic.base_rules.FuzzyInference(*domain: Domain*, *ruleset: list[BaseRule]*, *defuzz_ by: str = 'cgrav'*)

Базовые классы: object

Класс для нечеткого вывода, использующий набор правил для формирования вывода.

domain

Нечеткая область, используемая для дискретизации выходных данных.

Type
Domain

ruleset

Набор правил, применяемых в процессе вывода.

Type
list[*BaseRule*]

defuzz_by

Метод дефаззификации, используемый для получения четкого значения. По умолчанию „cgrav“.

Type
str

__call__(X

Union[float, int]) -> float: Применяет нечеткую логику к входному значению X и возвращает четкое значение.

3 Раздел для работы нечеткими методами многокритериального анализа

fuzzyops.fuzzy_msa.msa_fuzzy_hierarchy.fuzzy_hierarchy_solver(*criteria_weights: List[List[FuzzyNumber]]*, *alternative_comparisons: List[List[List[FuzzyNumber]]]*) → List[FuzzyNumber]

Решатель нечеткой иерархии, использующий метод анализа иерархий для определения приоритетов альтернатив.

Параметры

- criteria_weights (*List[List[FuzzyNumber]]*) – Двумерный список нечетких чисел, представляющий веса критериев.
- alternative_comparisons (*List[List[List[FuzzyNumber]]]*) – Трехмерный список нечетких чисел, представляющий
- критерия. (*сравнительные оценки альтернатив для каждого*)

Результат

Список нечетких чисел, представляющий глобальные приоритеты альтернатив.

Тип результата

List[FuzzyNumber]

fuzzyops.fuzzy_msa.msa_fuzzy_pairwise.fuzzy_pairwise_solver(*alternatives: List[str]*, *criteria: List[str]*, *pairwise_matrices: List[List[List[FuzzyNumber]]]*) → List[Tuple[str, FuzzyNumber]]

Решатель сравнений парных нечетких альтернатив, использующий метод анализа для оценки и ранжирования.

Параметры

- alternatives (*List[str]*) – Список строк, представляющих названия альтернатив.
- criteria (*List[str]*) – Список строк, представляющих названия критериев.
- pairwise_matrices (*List[List[List[FuzzyNumber]]]*) – Список матриц парных сравнений для каждой альтернативы
- критерию (*по каждому*)
- числа. (*содержащих нечеткие*)

Результат

Список кортежей, каждый из которых содержит название альтернативы и соответствующий ей итоговую нечеткую оценку, отсортированные по убыванию.

Тип результата

List[Tuple[str, FuzzyNumber]]

Исключение
`ValueError` – Если количество матриц парных сравнений не совпадает с количеством критериев.

`fuzzyops.fuzzy_msa.msa_fuzzy_pareto.fuzzy_pareto_solver`(*solutions: List[List[FuzzyNumber]]*) → List[List[FuzzyNumber]]

Находит решения, не доминируемые другими решениями в многокритериальной задаче.

Параметры
`solutions` (*List[List[FuzzyNumber]]*) – Список решений, каждое из которых представлено списком нечетких чисел.

Результат
Список решений, которые не доминируются другими решениями (существует хотя бы одно решение, по которому другие решения лучше по всем критериям).

Тип результата
List[List[FuzzyNumber]]

`fuzzyops.fuzzy_msa.msa_fuzzy_sum.fuzzy_sum_solver`(*criteria_weights: List[FuzzyNumber]*, *alternatives_scores: List[List[FuzzyNumber]]*) → List[FuzzyNumber]

Вычисляет взвешенную сумму оценок для альтернатив на основе заданных весов критериев.

Параметры

- `criteria_weights` (*List[FuzzyNumber]*) – Список нечетких чисел, представляющий веса критериев.
- `alternatives_scores` (*List[List[FuzzyNumber]]*) – Двумерный список нечетких чисел, представляющий оценки для
- `критерию.` (*каждой альтернативы по каждому*)

Результат
Список нечетких чисел, представляющий итоговые взвешенные суммы для каждой альтернативы.

Тип результата
List[FuzzyNumber]

Исключение
`ValueError` – Если количество оценок для одной из альтернатив не соответствует количеству критериев.

4 Раздел для работы с нечеткой нейронной сетью (алгоритм 1)

`class fuzzyops.fuzzy_neural_net.layer.FuzzyNNLayer`(*ind: int*, *size: int*, *domain: Domain*, *neuronType: str*)

Базовые классы: `object`

Представляет слой нечеткой нейронной сети, состоящий из нескольких нечетких нейронов.

`_index`

Индекс слоя в сети.

Type
int

`_neurons`

Список нечетких нейронов в слое.

Type
List[*FuzzyNNNeuron*]

`_domain`

Объект домена для работы с нечеткими числами.

Type
Domain

Параметры

- `ind` (*int*) – Индекс слоя.
- `size` (*int*) – Количество нейронов в слое.
- `domain` (*Domain*) – Объект домена для работы с нечеткими числами.
- `neuronType` (*str*) – Тип нейронов в слое.

`add_into_synapse(toAddNeuronNumber`

int, Synapse: FuzzyNNSynapse) -> None: Добавляет входящее ребро к указанному нейрону в слое.

`add_out_synapse(toAddNeuronNumber`

int, Synapse: FuzzyNNSynapse) -> None: Добавляет исходящее ребро от указанного нейрона в слое.

`__len__()` → int

Возвращает количество нейронов в слое.

`forward()` → None

Проводит прямое распространение сигналов через все нейроны в слое.

`backward()` → None

Проводит обратное распространение ошибок через все нейроны в слое.

`add_into_synapse(`*toAddNeuronNumber: int*, *Synapse:* FuzzyNNSynapse`)` → None

Добавляет входящее ребро к указанному нейрону в слае.

Параметры

- `toAddNeuronNumber` (*int*) – Индекс нейрона, к которому добавляется входящее ребро.
- `Synapse` (FuzzyNNSynapse) – Синапс, который добавляется как входящее ребро.

`add_out_synapse(`*toAddNeuronNumber: int*, *Synapse:* FuzzyNNSynapse`)` → None

Добавляет исходящее ребро от указанного нейрона в слое.

Параметры

- `toAddNeuronNumber` (*int*) – Индекс нейрона, от которого добавляется исходящее ребро.
- `Synapse` (FuzzyNNSynapse) – Синапс, который добавляется как исходящее ребро.

`backward()` → None

Проводит обратное распространение ошибок через все нейроны в слое.

`forward()` → None

Проводит прямое распространение сигналов через все нейроны в слое.

`class fuzzyops.fuzzy_neural_net.network.FuzzyNNetwork(`*layersSizes: tuple | list*, *domainValues: Tuple = (0, 100)*, *method: str = 'minimax'*, *fuzzyType: str = 'triangular'*, *activationType: str = 'linear'*, *cuda: bool = False*, *verbose: bool = False*`)`

Базовые классы: `object`

Класс для создания нечеткой нейронной сети.

`_layers`

Список слоев нечеткой нейронной сети.

Type
List[*FuzzyNNLayer*]

`_verbose`

Функция для вывода отладочной информации.

Type
Callable

`_errors`

Список ошибок на каждой эпохе.

Type
List[float]

`_total_err`

Общая ошибка сети.

Type
float

`_domain`

Объект домена для работы с нечеткими числами.

Type
Domain

`_input_synapses`

Список входных синапсов.

Type
List[*FuzzyNNSynapse*]

`_output_synapses`

Список выходных синапсов.

Type
List[*FuzzyNNSynapse*]

`fit(x_train`

List[List[FuzzyNumber]], y_train: List[List[FuzzyNumber]], steps: int = 1) -> None: Обучает нечеткую нейронную сеть на заданных тренировочных данных.

predict(x_predict

List[FuzzyNumber]) -> List[float]: Делает предсказание на основе входных данных.

Параметры

- layersSizes (*Union[tuple, list]*) – Размеры слоев сети.
- domainValues (*Tuple, optional*) – Значения домена для нечетких чисел (по умолчанию (0, 100)).
- method (*str, optional*) – Метод работы с нечеткими числами (по умолчанию „minimax“).
- fuzzyType (*str, optional*) – Тип нечеткой числовой функции (по умолчанию „triangular“).
- activationType (*str, optional*) – Тип активационной функции (по умолчанию „linear“).
- cuda (*bool, optional*) – Использовать ли GPU (по умолчанию False).
- verbose (*bool, optional*) – Выводить ли отладочную информацию (по умолчанию False).

fit(*x_train: List[List[FuzzyNumber]]*, *y_train: List[List[FuzzyNumber]]*, *steps: int = 1*) → None

Обучает нечеткую нейронную сеть на заданных тренировочных данных.

Параметры

- x_train (*List[List[FuzzyNumber]]*) – Тренировочные данные входных значений.
- y_train (*List[List[FuzzyNumber]]*) – Целевые значения для тренировочных данных.
- steps (*int, optional*) – Число эпох обучения (по умолчанию 1).

Исключение

- AssertionError – Если размеры x_train и y_train отличаются, или если размеры x_train и y_train
- не соответствуют ожидаемым размерам входных и выходных синапсов соответственно. –

predict(*x_predict: List[FuzzyNumber]*) → List[float]

Делает предсказание на основе входных данных.

Параметры
x_predict (*List[FuzzyNumber]*) – Входные данные для предсказания.

Результат
Список значений предсказания.

Тип результата
List[float]

Исключение
AssertionError – Если размер x_predict не соответствует количеству входных синапсов.

class fuzzyops.fuzzy_neural_net.neuron.FuzzyNNNeuron(*neuronType: str = 'linear'*)

Базовые классы: object

Представляет нечеткий нейрон в нечеткой нейронной сети.

neuronType

Тип нейрона (например, „linear“, „relu“).

Type
str

intoSynapses

Список входящих синапсов.

Type
List[*FuzzyNNSynapse*]

outSynapses

Список исходящих синапсов.

Type
List[*FuzzyNNSynapse*]

Параметры
neuronType (*str*) – Тип нейрона (по умолчанию „linear“).

addInto(toAdd

FuzzyNNSynapse) -> None: Добавляет входящий синапс к нейрону.

addOut(toAdd

FuzzyNNSynapse) -> None: Добавляет исходящий синапс от нейрона.

doCalculateForward(value

Union[FuzzyNumber, float, int]) -> None: Выполняет вычисление прямого распространения для данного значения.

doCalculateBackward(value

Union[FuzzyNumber, float, int]) -> None: Выполняет вычисление обратного распространения для данного значения.

forward() → None

Проводит прямое распространение через нейрон.

backward() → None

Проводит обратное распространение через нейрон.

addInto(*toAdd:* FuzzyNNSynapse) → None

Добавляет входящий синапс к нейрону.

Параметры
toAdd (FuzzyNNSynapse) – Синапс, который будет добавлен в входящие синапсы.

addOut(*toAdd:* FuzzyNNSynapse) → None

Добавляет исходящий синапс от нейрона.

Параметры
toAdd (FuzzyNNSynapse) – Синапс, который будет добавлен в исходящие синапсы.

backward() → None

Проводит обратное распространение через нейрон. Накапливает ошибки выходящих синапсов и передает результат через входящие синапсы.

doCalculateBackward(*value: FuzzyNumber | float | int*) → None

Выполняет вычисление обратного распространения для данного значения.

Параметры
value (*Union[FuzzyNumber, float, int]*) – Ошибка для активации.

Результат
Производная активационной функции.

Тип результата
Union[FuzzyNumber, float, int]

doCalculateForward(*value: FuzzyNumber | float | int*) → None

Выполняет вычисление прямого распространения для данного значения.

Параметры
value (*Union[FuzzyNumber, float, int]*) – Входное значение для активации.

Результат
Результат активационной функции.

Тип результата
Union[FuzzyNumber, float, int]

forward() → None

Проводит прямое распространение через нейрон. Накапливает значения входящих синапсов и передает результат через выходящие синапсы.

class fuzzyops.fuzzy_neural_net.neuron.Linear

Базовые классы: object

Класс для линейной активационной функции.

forward(x

Union[FuzzyNumber, float, int]) -> Union[FuzzyNumber, float, int]: Применяет линейную активационную функцию к входному значению.

backward(x

Union[FuzzyNumber, float, int]) -> Union[FuzzyNumber, float, int]: Возвращает производную линейной функции.

static backward(*x: FuzzyNumber | float | int*)

static forward(*x: FuzzyNumber | float | int*)

class fuzzyops.fuzzy_neural_net.neuron.Relu

Базовые классы: object

Класс для активационной функции ReLU (Rectified Linear Unit).

forward(x

Union[FuzzyNumber, float, int]) -> Union[FuzzyNumber, float, int]: Применяет активационную функцию ReLU к входному значению.

backward(x

Union[FuzzyNumber, float, int]) -> Union[FuzzyNumber, float, int]: Возвращает производную функции ReLU.

static backward(*x: FuzzyNumber | float | int*)

static forward(*x: FuzzyNumber | float | int*)

class fuzzyops.fuzzy_neural_net.synapse.FuzzyNNSynapse(*weight: float | FuzzyNumber*)

Базовые классы: object

Представляет синапс в нечеткой нейронной сети, который соединяет два нейрона.

value

Значение синапса, передаваемое через него.

Type
Union[float, FuzzyNumber]

error

Ошибка, связанная с синапсом, для обновления веса.

Type
Union[float, FuzzyNumber]

weight

Вес синапса, влияющий на передаваемое значение.

Type
Union[float, FuzzyNumber]

Параметры
weight (*Union[float, FuzzyNumber]*) – Начальный вес синапса.

setValue(value

Union[float, FuzzyNumber]) -> None: Устанавливает значение синапса.

getValue() → FuzzyNumber

Возвращает значение синапса, умноженное на его вес.

setError(error

Union[float, FuzzyNumber]) -> None: Устанавливает ошибку синапса.

getError() → float | FuzzyNumber

Возвращает текущую ошибку синапса.

applyError() → None

Применяет ошибку к весу синапса для обновления его значения.

applyError() → None

Применяет ошибку к весу синапса, обновляя его значение.

Обновление веса происходит по формуле: новый вес = старый вес + ошибка * 0.1

getError() → None

Возвращает текущую ошибку синапса.

Результат
Ошибка синапса.

Тип результата
Union[float, FuzzyNumber]

`getValue()` → FuzzyNumber

Возвращает значение синапса.

Результат
Значение синапса, умноженное на его вес.

Тип результата
FuzzyNumber

`setError(`*error: float | FuzzyNumber*`)`

Устанавливает ошибку синапса.

Параметры
`error` (*Union[float, FuzzyNumber]*) – Ошибка, которая будет установлена для синапса.

`setValue(`*value: float | FuzzyNumber*`)`

Устанавливает значение синапса.

Параметры
`value` (*Union[float, FuzzyNumber]*) – Значение, которое будет установлено для синапса.

5 Раздел для работы с нечеткой нейронной сетью (алгоритм 2)

`class fuzzyops.fuzzy_nn.mf_funcs.BellMemberFunc(`*a: float*, *b: float*, *c: float*`)`

Базовые классы: `Module`

Представляет нечеткую функцию принадлежности обощенного колокола.

Параметры

- `a` (*float*) – Параметр, определяющий ширину функции.
- `b` (*float*) – Параметр, определяющий наклон функции.
- `c` (*float*) – Параметр центра функции.

`forward(x`

torch.Tensor) -> torch.Tensor: Вычисляет значение функции принадлежности колокола для входного тензора.

`forward(`*x: Tensor*`)` → Tensor

Вычисляет значение функции колокола для входного тензора.

Параметры
`x` (*torch.Tensor*) – Входной тензор, для которого необходимо вычислить значение.

Результат
Значение функции колокола.

Тип результата
torch.Tensor

class fuzzyops.fuzzy_nn.mf_funcs.GaussMemberFunc(*mu: float*, *sigma: float*)

Базовые классы: Module

Представляет нечеткую функцию принадлежности Гаусса.

Параметры

- mu (*float*) – Параметр центра (среднее значение) функции.
- sigma (*float*) – Параметр ширины (стандартное отклонение) функции.

forward(x

torch.Tensor) -> torch.Tensor: Вычисляет значение функции Гаусса для входного тензора.

forward(*x: Tensor*) → Tensor

Вычисляет значение функции принадлежности Гаусса для входного тензора.

Параметры
x (*torch.Tensor*) – Входной тензор, для которого необходимо вычислить значение.

Результат
Значение функции Гаусса.

Тип результата
torch.Tensor

fuzzyops.fuzzy_nn.mf_funcs.make_bell_mfs(*a: float*, *b: float*, *c_list: List[float]*) → List[*BellMemberFunc*]

Создает список функций колокола.

Параметры

- a (*float*) – Параметр ширины для всех создаваемых функций.
- b (*float*) – Параметр наклона для всех создаваемых функций.
- c_list (*List[float]*) – Список параметров центра для создания функций.

Результат
Список созданных функций колокола.

Тип результата
List[*BellMemberFunc*]

fuzzyops.fuzzy_nn.mf_funcs.make_gauss_mfs(*sigma: float*, *mu_list: List[float]*) → List[*GaussMemberFunc*]

Создает список функций Гаусса.

Параметры

- sigma (*float*) – Параметр ширины для всех создаваемых функций.
- mu_list (*List[float]*) – Список параметров центра для создания функций.

Результат
Список созданных функции Гаусса.

Тип результата
List[*GaussMemberFunc*]

class fuzzyops.fuzzy_nn.model.Model(*X: ndarray*, *Y: ndarray*, *n_terms: list[int]*, *n_out_vars: int*, *lr: float*, *task_type: str*, *batch_size: int*, *member_func_type: str*, *epochs: int*, *verbose: bool = False*, *device: device = device(type='cpu')*)

Базовые классы: `object`

Класс для создания и обучения модели нечеткой логики.

Этот класс предназначен для выполнения задач регрессии и классификации с использованием нечеткой логики. Он принимает входные данные, определяет параметры модели и осуществляет предварительную обработку данных.

`task_names`

Словарь, связывающий типы задач с их текстовыми представлениями.

Type
dict

`X`

Входные данные модели.

Type
np.ndarray

`Y`

Выходные данные модели.

Type
np.ndarray

`n_input_features`

Число входных признаков.

Type
int

`n_terms`

Список, содержащий число термов для каждой входной переменной.

Type
list[int]

`n_out_vars`

Количество выходных переменных.

Type
int

`lr`

Шаг обучения для оптимизации.

Type
float

`task_type`

Тип задачи («regression» или «classification»).

Type
str

`batch_size`

Размер подвыборки для обучения.

Type
int

`member_func_type`

Тип функции принадлежности.

Type
str

`device`

Устройство, на котором будет выполняться модель (например, «cpu» или «cuda»).

Type
torch.device

`epochs`

Количество эпох для обучения модели.

Type
int

`scores`

Список для сохранения показателей модели.

Type
list

`verbose`

Флаг «подробного» вывода информации о процессе обучения.

Type
bool

`model`

Модель для обучения, в настоящее время не определена.

Type
torch.nn.Module

Параметры

- `X` (*np.ndarray*) – Входные данные для модели.
- `Y` (*np.ndarray*) – Целевые значения для модели.
- `n_terms` (*list[int]*) – Число термов для каждой входной переменной.
- `n_out_vars` (*int*) – Количество выходных переменных.
- `lr` (*float*) – Шаг обучения.
- `task_type` (*str*) – Тип задачи: „regression“ или „classification“.
- `batch_size` (*int*) – Размер подвыборки для обучения.
- `member_func_type` (*str*) – Тип функции принадлежности.
- `epochs` (*int*) – Количество эпох для обучения модели.
- `verbose` (*bool*) – Уровень подробности вывода (по умолчанию False).

- `device` (*torch.device*) – Устройство для вычислений, по умолчанию «cpu».

`__str__()` → str

Строковое представление объекта модели.

`__repr__()` → str

Описание объекта модели для отладки.

`__preprocess_data()` → DataLoader

Предварительная обработка данных и создание DataLoader.

`__gauss_func(x`

torch.Tensor) -> Tuple[List]: Генерирует параметры для гауссовских функций принадлежности на основе входных данных.

`__bell_func(x`

torch.Tensor) -> tuple[list]: Генерирует параметры для колоколообразных функций принадлежности на основе входных данных.

`__compile(x`

torch.Tensor) -> _NN: Компилирует модель нечеткой нейронной сети на основе выбранного типа функции принадлежности.

`__class_criterion(inp`

torch.Tensor, target: torch.Tensor) -> float: Вычисляет значение функции потерь для задачи классификации.

`__reg_criterion(inp`

torch.Tensor, target: torch.Tensor) -> float: Вычисляет значение функции потерь для задачи регрессии.

`__calc_reg_score(preds`

torch.Tensor, y_actual: torch.Tensor) -> float: Вычисляет оценку модели для задачи регрессии.

`__calc_class_score(preds`

torch.Tensor, y_actual: torch.Tensor, x: torch.Tensor) -> float: Вычисляет точность модели для задачи классификации.

`__train_loop(data`

DataLoader, model: _NN, criterion: Callable, calc_score: Callable, optimizer: torch.optim.Adam) -> None: Основной цикл обучения модели.

`train()` → _NN

Запускает процесс обучения модели.

`save_model(path`

str) -> None: Сохраняет состояние обученной модели в файл.

`save_model(`*path: str*`)` → None

Сохраняет состояние обученной модели в файл. Сохраняет параметры модели с использованием указанного пути.

Параметры

`path` (*str*) – Путь к файлу, в который будет сохранена модель.

Исключение

`Exception` – Если модель не была обучена.

Результат
None

`task_names = {'classification': 'классификации', 'regression': 'регрессии'}`

`train()` → _NN

Запускает процесс обучения модели.

Выполняет предварительную обработку данных, компиляцию модели и выполнение цикла обучения с использованием заданных критериев и оптимизатора.

Результат
Обученная модель нечеткой нейронной сети.

Тип результата
_NN

6 Раздел для работы с нечеткими числами

7 Раздел для работы с нечеткими методами оптимизации

`class fuzzyops.fuzzy_optimization.fuzzy_linear_optimization.optimization.Response(`*interaction_coefs: numpy.ndarray*, *interactions: pandas.core.frame.DataFr* *alphas: numpy.ndarray*`)`

Базовые классы: `object`

`alphas: ndarray`

`interaction_coefs: ndarray`

`interactions: DataFrame`

`fuzzyops.fuzzy_optimization.fuzzy_linear_optimization.optimization.calc_root_value(`*square_num: ndarray*`)` → Tuple[ndarray, ndarray]

Вычисляет корневые значения для квадратного нечеткого числа.

Параметры
`square_num` (*np.ndarray*) – Входной массив нечеткого числа, чтобы вычислить корень.

Результат
Два массива с корнями.

Тип результата
Tuple[np.ndarray, np.ndarray]

`fuzzyops.fuzzy_optimization.fuzzy_linear_optimization.optimization.calc_scalar_value(`*c1: ndarray*, *c2: ndarray*`)` → ndarray

Вычисляет скалярное значение на основе двумерных массивов.

Параметры

- c1 (*np.ndarray*) – Первый массив поэлементных коэффициентов.
- c2 (*np.ndarray*) – Второй массив поэлементных коэффициентов.

Результат
Вычисленный результат как массив скалярных значений.

Тип результата
np.ndarray

fuzzyops.fuzzy_optimization.fuzzy_linear_optimization.optimization.get_interaction_matrix(*matrix: ndarray[FuzzyN type_of_all_nu triangular*)

fuzzyops.fuzzy_optimization.fuzzy_linear_optimization.optimization.transform_matrix(*func: Callable*) → Callable

Декоратор для проверки всех условий и трансформации матрицы нечетких чисел.

Параметры
func (*Callable*) – Функция, которая будет вызываться после проверки условий.

Результат
Обернутая функция с проверками.

Тип результата
Callable

class fuzzyops.fuzzy_optimization.fuzzy_metaevristic_optimization.optimization.AntOptimization(*data: ndarray, k: int, epsilon: float, q: float, n_iter: int, n_ant: int, ranges: list[Fuzzyl r: ndarray, R: int*)

Базовые классы: object

Алгоритм оптимизации муравьиных колоний для идентификации нечетких систем. Алгоритм реаоизован по статье: И.А. Ходашинский, П.А. Дудин. Идентификация нечетких систем на основе непрерывного алгоритма муравьиных колоний. Автометрия. 2012. Т. 48, № 1.

Параметры

- `data` (*np.ndarray*) – Входные данные для создания модели.
- `k` (*int*) – Количество начальных решений.
- `epsilon` (*float*) – Параметр для изменения веса.
- `q` (*float*) – Параметр для нормализации потерь.
- `n_iter` (*int*) – Количество итераций алгоритма.
- `n_ant` (*int*) – Общее количество муравьев (агентов).
- `ranges` (*list [FuzzyBounds]*) – Границы для нечетких значений.
- `r` (*np.ndarray*) – Входные данные для расчета потерь.
- `R` (*int*) – Количество повторений для расчета потерь.

`n`

Количество входных переменных.

Type
int

`p`

Количество наблюдений.

Type
int

`N`

Общее количество параметров.

Type
int

`R`

Количество повторений для расчета потерь.

Type
int

`X`

Датафрейм, содержащий входные переменные.

Type
pd.DataFrame

`t`

Целевые значения.

Type
np.ndarray

`r`

Входные данные для расчета потерь.

Type
np.ndarray

`ranges`

Границы для нечетких значений.

Type
list[*FuzzyBounds*]

`_low`

Минимальное значение границ.

Type
float

`_high`

Максимальное значение границ.

Type
float

`k`

Количество начальных решений.

Type
int

`theta`

Структура параметров.

Type
np.ndarray

`storage`

Хранилище для архива потерь и параметров.

Type
list[*Archive*]

`eps`

Параметр для управления изменениями.

Type
float

`q`

Параметр для нормализации потерь.

Type
float

`n_iter`

Количество итераций алгоритма.

Type
int

`n_ant`

Общее количество муравьев (агентов).

Type
int

`__f`(*theta*)

Вычисляет значение функции на основе параметров.

`_root_mean_squared_error(`*theta*`)`

Вычисляет среднеквадратическую ошибку между целевыми значениями и предсказанными значениями.

`_calc_weights(`*index*`)`

Вычисляет вес на основе позиции муравья.

`__init_solution()`

Инициализирует решения, рассчитывая потери для каждого из них.

`best_result()`

Возвращает лучшее решение на данный момент.

`continuous_ant_algorithm()`

Запускает непрерывный алгоритм муравьиных колоний.

`interp(num`

FuzzyNumber, value: Union[int, float]) -> float: Интерполяция нечеткого числа на заданном значении.

`__construct_fuzzy_num(theta`

np.ndarray) -> np.ndarray: Конструирует нечеткие числа из параметров.

`property best_result: Archive`

Возвращает лучшее решение на данный момент.

Результат
Лучшее решение с наименьшими потерями.

Тип результата
Archive

`continuous_ant_algorithm()` → ndarray

Запускает непрерывный алгоритм муравьиных колоний.

Результат
Итоговые параметры после выполнения оптимизации.

Тип результата
np.ndarray

`static interp(`*num: FuzzyNumber*, *value: int | float*`)`

`class fuzzyops.fuzzy_optimization.fuzzy_metaevristic_optimization.optimization.Archive(`*k: int*, *params: ndarray*, *loss: float*`)`

Базовые классы: `object`

Хранит параметры и результаты для одной итерации оптимизации.

`k`

Индекс архива.

Type
int

`params`

Параметры нечеткой модели.

Type
np.ndarray

`loss`

Потери (ошибка) для данной итерации.

Type
float

`k: int`

`loss: float`

`params: ndarray`

`class fuzzyops.fuzzy_optimization.fuzzy_metaevristic_optimization.optimization.FuzzyBounds(`*start: int | float*, *end: int | float*, *step: int | float*, *x: list[str]*`)`

Базовые классы: `object`

Определяет границы для нечетких значений.

`start`

Начальное значение границ.

Type
Union[int, float]

`end`

Конечное значение границ.

Type
Union[int, float]

`step`

Шаг для границ.

Type
Union[int, float]

`x`

Список меток или названий границ.

Type
list[str]

end: int | float

start: int | float

step: int | float

x: list[str]

fuzzyops.fuzzy_optimization.fuzzy_multy_opt.optimization.solve_problem(*A: ndarray*, *b: ndarray*, *C: ndarray*, *g: ndarray*, *t: ndarray*) → Tuple[float, ndarray]

Формулирует и решает задачу нечеткой оптимизации с заданными матрицами состояния и ограничениями.

Параметры

- A (*np.ndarray*) – Матрица коэффициентов для ограничений.
- b (*np.ndarray*) – Вектор правых частей для ограничений.
- C (*np.ndarray*) – Матрица коэффициентов для критических значений.
- g (*np.ndarray*) – Вектор критических значений.
- t (*np.ndarray*) – Вектор значений, определяющих степень допуска.

Результат

Кортеж, содержащий значение целевой функции (максимизированное значение) и
оптимальные значения переменных (вектор x).

Тип результата
Tuple[float, np.ndarray]

8 Раздел для работы с нечеткми графами

class fuzzyops.graphs.fuzzgraph.graph.FuzzyGraph(*node_type: str = 'simple'*, *node_number_type: str = 'triangle'*, *edge_type: str = 'undirected'*, *edge_number_type: str = 'triangle'*, *node_number_math_type: min | base | max = None*, *node_number_eq_type: min | base | max = None*, *edge_number_math_type: min | base | max = None*, *edge_number_eq_type: min | base | max = None*)

Базовые классы: object

add_edge(*from_ind: int*, *to_ind: int*, *value: List[int]*)

Добавляет ребро между двумя узлами.

Параметры

- from_ind (*int*) – Индекс исходного узла.
- to_ind (*int*) – Индекс целевого узла.
- value (*List[int]*) – Значение для ребра.

Исключение
`Exception` – Исключение возникает, если граф не допускает петель или если указаны несуществующие узлы.

`add_node`(*value=None*) → None

Добавляет узел в граф.

Параметры
`value` – Значение для узла (по умолчанию None). Если указано, создается нечеткое число для узла.

`check_directed_edge`(*from_ind: int*, *to_ind: int*) → bool

Проверяет, существует ли направленное ребро между двумя узлами.

Параметры

- `from_ind` (*int*) – Индекс исходного узла.
- `to_ind` (*int*) – Индекс целевого узла.

Результат
True, если направленное ребро существует, иначе False.

Тип результата
bool

Исключение
`Exception` – Исключение возникает, если узлы не существуют.

`check_node`(*index: int*) → bool

Проверяет, существует ли узел с заданным индексом.

Параметры
`index` (*int*) – Индекс узла.

Результат
True, если узел существует, иначе False.

Тип результата
bool

`check_nodes_full`(*nodes: List[int]*) → bool

Проверяет, охватывают ли указанные узлы все узлы в графе.

Параметры
`nodes` (*List[int]*) – Список индексов узлов для проверки.

Результат
True, если указанные узлы охватывают все узлы в графе, иначе False.

Тип результата
bool

`get_adjacency_matrix`() → List[List[int]]

Возвращает матрицу смежности графа.

Матрица смежности - это квадратная матрица, где элемент (i, j) представляет длину ребра между узлом i и узлом j. Если ребра нет, элемент будет равен None.

Результат
Матрица смежности графа.

Тип результата
List[List[int]]

Примечание

Если граф направленный, матрица будет отображать направление ребер. Если узлы не связаны, соответствующие элементы будут равны None.

`get_directly_connected`(*index: int*) → List[int]

Получает список узлов, которые напрямую связаны с заданным узлом.

Параметры
`index` (*int*) – Индекс узла.

Результат
Список индексов напрямую связанных узлов.

Тип результата
List[int]

Исключение
`Exception` – Исключение возникает, если узел не существует.

`get_edge_len`(*from_ind: int*, *to_ind: int*) → int

Получает длину ребра между двумя узлами.

Параметры

- `from_ind` (*int*) – Индекс исходного узла.
- `to_ind` (*int*) – Индекс целевого узла.

Результат
Длина ребра между узлами.

Тип результата
int

Исключение
`Exception` – Исключение возникает, если граф не допускает петель или если указаны несуществующие узлы.

`get_edges_amount`() → int

Возвращает количество ребер в графе.

Результат
Количество ребер.

Тип результата
int

`get_nodes_amount`() → int

Возвращает количество узлов в графе.

Результат
Количество узлов.

Тип результата
int

get_nodes_list() → List[int]

Получает список индексов всех узлов в графе.

Результат
Список индексов узлов.

Тип результата
List[int]

get_stronger_directly_connected(*index: int*, *value: List[int]*) → List[int]

Получает список узлов, которые напрямую связаны с заданным узлом и имеют более сильные ребра.

Параметры

- index (*int*) – Индекс узла.
- value (*List[int]*) – Список значений, по которым осуществляется фильтрация.

Результат
Список индексов более сильно связанных узлов.

Тип результата
List[int]

Исключение
Exception – Исключение возникает, если узел не существует.

9 Раздел для работы с алгоритмами на нечетких графах

fuzzyops.graphs.algorithms.dominating.dominating_set.dominating_set(*graph:* FuzzyGraph) → set

Находит любое доминирующее множество в заданном нечетком графе.

Доминирующее множество - это подмножество узлов графа, такое что каждый узел графа либо принадлежит этому подмножеству, либо смежен с хотя бы одним узлом из этого подмножества.

Параметры
graph (FuzzyGraph) – Экземпляр нечеткого графа.

Результат
Множество индексов узлов, входящих в доминирующее множество.

Тип результата
set

Исключение

- Exception – Исключение возникает, если переданный граф не является
- экземпляром класса FuzzyGraph. –

fuzzyops.graphs.algorithms.dominating.fuzzy_dominating_set.fuzzy_dominating_set(*graph:* FuzzyGraph, *number_value:* *GraphTriangleFuzzyNumber* → set

Находит доминирующее множество в заданном нечетком графе, где соединение между узлами должно быть сильнее заданного нечеткого числа.

Доминирующее множество - это подмножество узлов графа, такое что каждый узел графа либо принадлежит этому подмножеству, либо смежен с хотя бы одним узлом из этого подмножества. Однако, в отличие от обычного доминирующего множества, здесь учитываются только связи, которые сильнее заданного нечеткого числа.

Параметры

- `graph` (FuzzyGraph) – Экземпляр нечеткого графа.
- `number_value` (*GraphTriangleFuzzyNumber*) – Нечеткое число, задающее минимальную силу соединений для включения узлов в доминирующее множество.

Результат

Множество индексов узлов, входящих в доминирующее множество.

Тип результата

set

Исключение

- `Exception` – Исключение возникает, если переданный граф не является
- `экземпляром класса FuzzyGraph.` –

`fuzzyops.graphs.algorithms.dominating.is_dominating.is_dominating`(*graph:* FuzzyGraph, *nodes_set: set[int]*) → bool

Проверяет, является ли заданное множество узлов доминирующим в нечетком графе.

Доминирующее множество - это подмножество узлов графа, такое что каждый узел графа либо принадлежит этому подмножеству, либо смежен с хотя бы одним узлом из этого подмножества.

Параметры

- `graph` (FuzzyGraph) – Экземпляр нечеткого графа.
- `nodes_set` (*set[int]*) – Множество индексов узлов, которые необходимо проверить на доминирующесть.

Результат

Возвращает *True*, если *nodes_set* является доминирующим множеством в графе, иначе возвращает *False*.

Тип результата

bool

Исключение

- `Exception` – Исключение возникает, если переданный граф не является
- `экземпляром класса FuzzyGraph или если nodes_set не является множеством.` –
- `Также возникает исключение, если в nodes_set есть узлы, которых нет в графе.` –

`fuzzyops.graphs.algorithms.factoring.mle_clusterization_factors.mle_clusterization_factors`(*graph:* FuzzyGraph, *clusters_amou int*) → List[int]

Выполняет кластеризацию узлов в нечетком графе с использованием метода MLE.

Метод максимизации правдоподобия (MLE) позволяет выделить несколько кластеров узлов, основываясь на их взаимосвязях и значениях.

Параметры

- `graph` (FuzzyGraph) – Экземпляр нечеткого графа, содержащего узлы для кластеризации.
- `clusters_amount` (*int*) – Количество кластеров, на которое необходимо разбить узлы.

Результат
Список индексов кластеров, к которым принадлежит каждый узел графа.

Тип результата
List[int]

Исключение
`Exception` – Исключение возникает, если: - *graph* не является экземпляром класса *FuzzyGraph*. - *clusters_ amount* не является целым числом.

`fuzzyops.graphs.algorithms.transport.shortest_path.shortest_path(`*graph:* FuzzyGraph, *start_ node: int*, *end_ node: int*`)` → Dict

Находит кратчайший путь между двумя заданными узлами в нечетком графе.

Узлы считаются связанными, если существует путь между ними, и для этого пути определяется его длина. Если узлы не связаны, выбрасывается исключение.

Параметры

- `graph` (FuzzyGraph) – Экземпляр нечеткого графа, в котором необходимо найти путь.
- `start_node` (*int*) – Индекс начального узла.
- `end_node` (*int*) – Индекс конечного узла.

Результат

Словарь с двумя ключами:

- “path“: Список узлов, представляющих кратчайший путь от *start_ node* до *end_ node*.
- “len“: Длина кратчайшего пути.

Тип результата
Dict

Исключение
`Exception` – Исключение возникает, если: - Переданный граф не является экземпляром класса *FuzzyGraph*. - Начальный или конечный узел не существуют в графе. - Начальный и конечный узлы не соединены (т.е. не существует пути между ними).

10 Раздел для работы с алгоритмом нечеткой линейной регрессии

`class fuzzyops.prediction.linear.TriFNum(`*domain: Domain*, *a: Tensor*, *b: Tensor*, *c: Tensor*`)`

Базовые классы: `object`

Представляет треугольное нечеткое число (TriFNum) для метода нечеткой линейной регрессии.

`domain`

Область определения нечеткого числа.

Type
Domain

`a`

Левый конец треугольника.

Type
torch.Tensor

`b`

Пик треугольника.

Type
torch.Tensor

`c`

Правый конец треугольника.

Type
torch.Tensor

`__init__(domain`

Domain, a: torch.Tensor, b: torch.Tensor, c: torch.Tensor): Инициализирует треугольное нечеткое число.

`values()` → torch.Tensor

Вычисляет и возвращает значения нечеткого числа на заданной области определения.

`__add__(other`

TriFNum | int | float) -> TriFNum: Определяет операцию сложения для треугольных нечетких чисел.

`__sub__(other`

TriFNum) -> TriFNum: Определяет операцию вычитания для треугольных нечетких чисел.

`__mul__(other`

int | float) -> TriFNum: Определяет операцию умножения для треугольных нечетких чисел.

`integrate_left()` → torch.Tensor

Вычисляет интеграл для левой стороны треугольного нечеткого числа.

`integrate_right()` → torch.Tensor

Вычисляет интеграл для правой стороны треугольного нечеткого числа.

`integrate()` → torch.Tensor

Вычисляет интеграл (полную площадь) под кривой треугольного нечеткого числа.

`to_fuzzy_number()` → FuzzyNumber

Преобразует треугольное нечеткое число в его нечеткое представление.

`integrate()` → Tensor

Вычисляет интеграл (полную площадь) под кривой треугольного нечеткого числа.

Результат
Значение интеграла.

Тип результата
torch.Tensor

`integrate_left()` → Tensor

Вычисляет интеграл для левой стороны треугольного нечеткого числа.

Результат
Значение интеграла.

Тип результата
torch.Tensor

`integrate_right()` → Tensor

Вычисляет интеграл для правой стороны треугольного нечеткого числа.

Результат
Значение интеграла.

Тип результата
torch.Tensor

`to_fuzzy_number()` → FuzzyNumber

Преобразует треугольное нечеткое число в его нечеткое представление.

Результат
Нечеткое число, созданное из треугольного нечеткого числа.

Тип результата
FuzzyNumber

`values()` → Tensor

Вычисляет значения нечеткого числа на заданной области определения.

Результат
Значения нечеткого числа в области определения.

Тип результата
torch.Tensor

`fuzzyops.prediction.linear.convert_fuzzy_number_for_lreg`(*n: FuzzyNumber*) → *TriFNum*

Преобразует нечеткое число класса FuzzyNumber в треугольное нечеткое число TriNum.

Параметры
`n` (*FuzzyNumber*) – Нечеткое число для преобразования.

Результат
Преобразованное треугольное нечеткое число.

Тип результата
TriFNum

`fuzzyops.prediction.linear.fit_fuzzy_linear_regression`(*X: List[*TriFNum*]*, *Y: List[*TriFNum*]*) → Tuple[float, float]

Реализует нечеткую линейную регрессию с использованием треугольных нечетких чисел.

Эта функция находит коэффициенты a и b для линейной регрессии, которые минимизируют расстояние между предсказанными нечеткими значениями и фактическими нечеткими значениями. Реализовано на основе материалов https://ej.hse.ru/data/2014/09/03/1316474700/%D0%A8%D0%B2%D0%B5%D0%B4%D0%BE%D0%B2.pdf

Параметры

- `X` (*List [TriFNum]*) – Список треугольных нечетких чисел, представляющих независимые переменные (фичи).
- `(List[TriFNum]` (*Y*) – Список треугольных нечетких чисел, представляющих зависимую переменную (целевую переменную).

Результат

Коэффициенты a и b линейной регрессии,
где a - угловой коэффициент, a b - свободный член.

Тип результата
Tuple[float, float]

Исключение
`ValueError` – Если длины списков X и Y не совпадают.

Заметки

Для выполнения расчетов используется интегрирование и вычисление различных моментов на основе нечетких чисел.

`fuzzyops.prediction.linear.fuzzy_distance`(*fn0:* TriFNum, *fn1:* TriFNum) → float

Вычисляет расстояние между двумя треугольными нечеткими числами.

Параметры

- `fn0` (TriFNum) – Первое треугольное нечеткое число.
- `fn1` (TriFNum) – Второе треугольное нечеткое число.

Результат
Расстояние между двумя нечеткими числами.

Тип результата
float

`fuzzyops.prediction.linear.integral_of_product`(*a_0: Tensor*, *b_0: Tensor*, *a_1: Tensor*, *b_1: Tensor*) → Tensor

Вычисляет интеграл произведения двух интервалов.

Параметры

- `a_0` (*torch.Tensor*) – Начало первого интервала.
- `b_0` (*torch.Tensor*) – Конец первого интервала.
- `a_1` (*torch.Tensor*) – Начало второго интервала.
- `b_1` (*torch.Tensor*) – Конец второго интервала.

Результат
Результат интеграла произведения.

Тип результата
torch.Tensor

`fuzzyops.prediction.linear.integrate_sum_squares`(*a_0: Tensor*, *b_0: Tensor*, *a_1: Tensor*, *b_1: Tensor*) → Tensor

Вычисляет интеграл суммы квадратов двух интервалов.

Параметры

- `a_0` (*torch.Tensor*) – Начало первого интервала.

- b_0 (*torch.Tensor*) – Конец первого интервала.
- a_1 (*torch.Tensor*) – Начало второго интервала.
- b_1 (*torch.Tensor*) – Конец второго интервала.

Результат
Результат интеграла суммы квадратов.

Тип результата
torch.Tensor

11 Раздел для работы с алгоритмом задачи о назначении

class fuzzyops.sequencing_assignment.solver.FuzzySASolver

Базовые классы: object

Представляет решатель задачи о назначениях (SAS) с использованием нечеткого графа.

_graph

Нечеткий граф, содержащий информацию о работниках и задачах.

Type
FuzzyGraph

_workers

Список работников.

Type
List[str]

_tasks

Список задач.

Type
List[str]

load_graph(graph

FuzzyGraph) -> None: Загружает пустой граф с определённым нечетким математическим типом.

load_tasks_workers(tasks

List[str], workers: List[str]) -> None: Загружает списки задач и работников.

load_task_worker_pair_value(task

str, worker: str, value: List[int]) -> None: Загружает стоимость назначения для пары работника и задачи.

solve() → Dict

Основная функция решения, реализующая Венгерский алгоритм.

load_graph(*graph:* FuzzyGraph) → None

Загружает пустой граф с определённым нечетким математическим типом.

Параметры
graph (FuzzyGraph) – Нечеткий граф для загрузки.

Исключение
Exception – Если граф уже не пуст или не является экземпляром FuzzyGraph.

`load_task_worker_pair_value`(*task: str*, *worker: str*, *value: List[int]*) → None

Загружает стоимость назначения для пары работника и задачи.

Параметры

- `task` (*str*) – Задача для назначения.
- `worker` (*str*) – Работник, которому назначается задача.
- `value` (*List[int]*) – Стоимость назначения.

Исключение

`Exception` – Если список задач или работников не загружен,

`load_tasks_workers`(*tasks: List[str]*, *workers: List[str]*) → None

Загружает списки задач и работников.

Параметры

- `tasks` (*List[str]*) – Список задач для загрузки.
- `workers` (*List[str]*) – Список работников для загрузки.

Исключение

`Exception` – Если граф не загружен, или если tasks и workers не являются списками.

`solve()` → Dict

Основная функция решения, реализующая Венгерский алгоритм.

Результат

Словарь с назначениями работников на задачи и общей стоимостью.

Тип результата

Dict

Исключение

`Exception` – Если граф или списки работников/задач не загружены.

12 Indices and tables

- genindex
- modindex
- search

Содержание модулей Python

f

Алфавитный указатель

Символы

A

B

C

D

E

F

G

I

K

L

M

N

Y