

Введение в глубинное обучение в анализе графовых данных

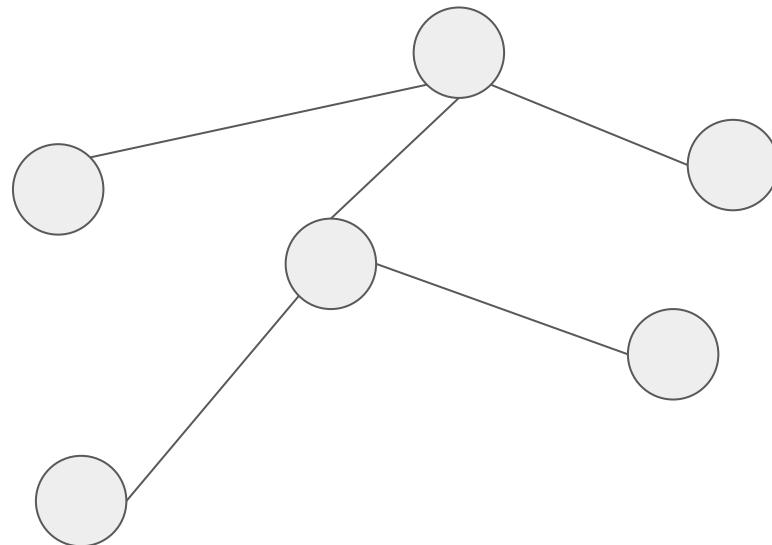
Вебинар

О курсе. Преподаватель

- Валитов Эльдар (ex Yandex, ex AliExpress Russia, HSE)

Что такое графы

$G = (V, E)$, где V - множество вершин, E - множество ребер



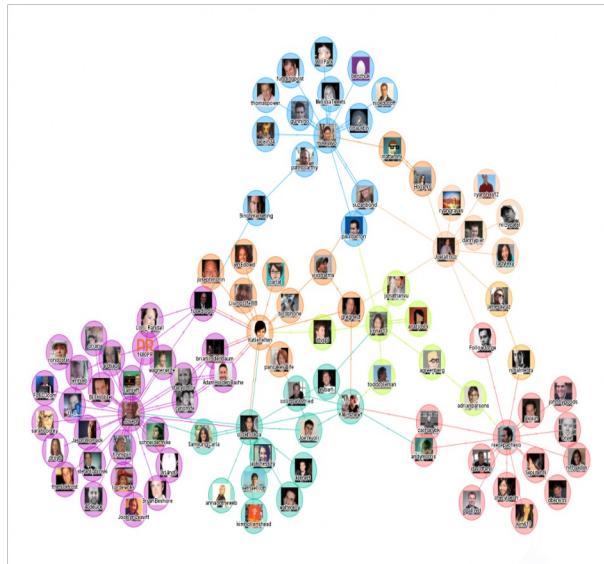
Зачем нужны графы в контексте данных

- Графы являются обобщенным способом представления любой информации связанной с наличием взаимодействия между объектами
- Существует большое количество различных методов анализа графов, поэтому удобно приводить данные из различных областей к одному формату

Примеры графовых данных

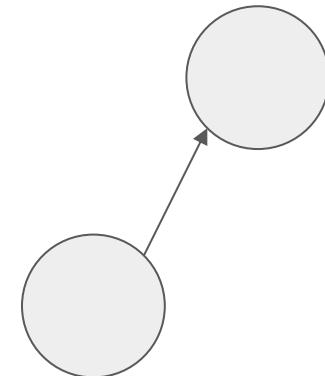


[Схема метрополитена](#)



[Социальный график](#)

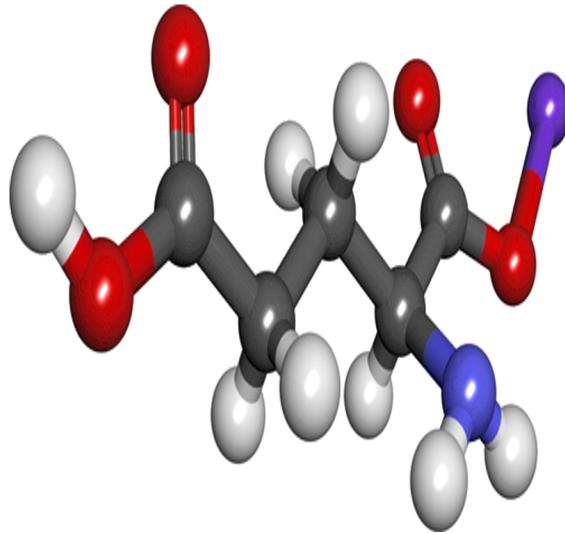
Санкт-Петербург



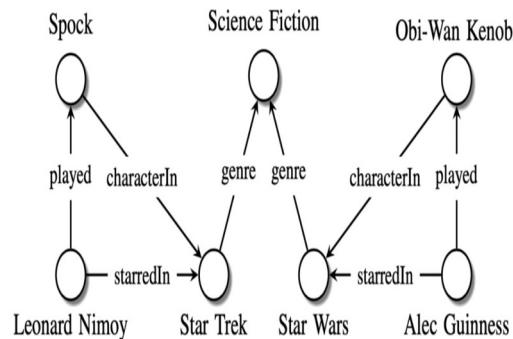
Москва

схема трафика

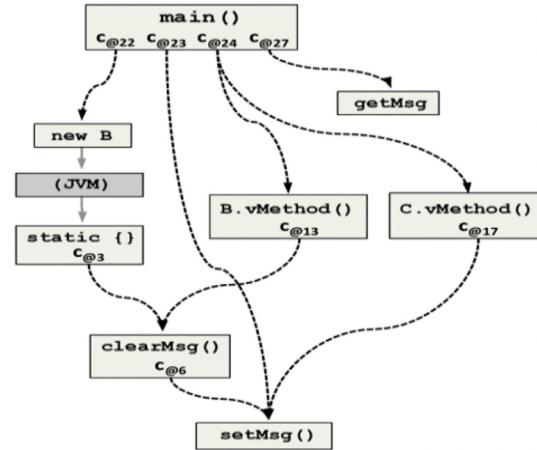
Примеры графовых данных



молекулы



графы знаний



КОД

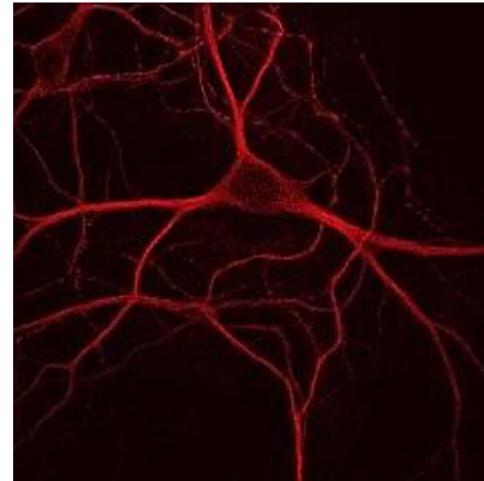
Примеры графовых данных



компьютерная сеть



интернет



сети нейронов

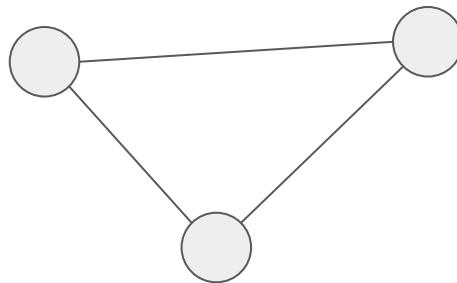
Мотивация

Основные применения глубинного обучения сегодня - последовательные данные:

- NLP : “askfnkksnf” - последовательность текста
- CV : структурированная из пикселей картинка
- Audio : звуковые волны

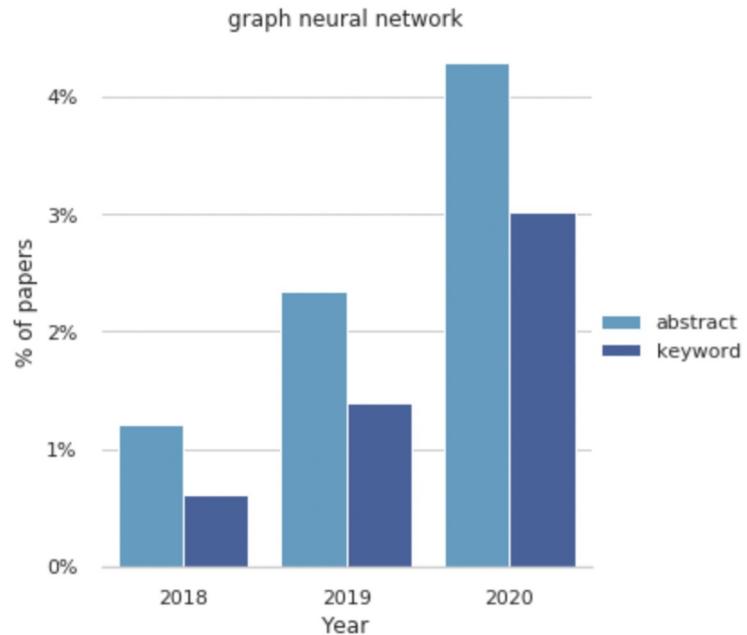
Мотивация

Отличительная особенность графов - связи между объектами, следовательно информацию можно обрабатывать шире и иначе



Где начало и конец?

Мотивация

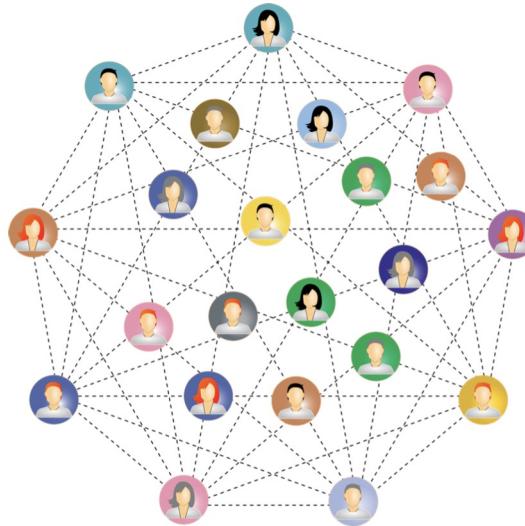


[Тренд GNN на ICLR](#)

Типы задач на графах

1. Edge-level - задачи связанные с ребрами (классификация связей, регрессия, дополнение графа)
2. Node-level - задачи связанные с вершинами (классификация вершин, регрессия, кластеризация)
3. Graph-level - специфичный для задач на графах тип задач (также классификация, регрессия, генерация, эволюция)

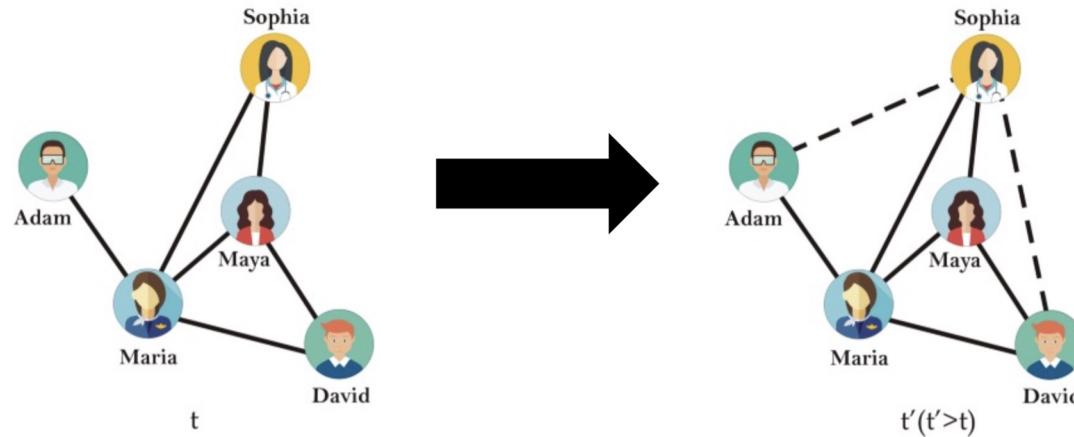
Node-level задачи



Определение интересов человека по его окружению

Хотим выучить $f: (V, E) \rightarrow \mathbb{R}^{|V|}$

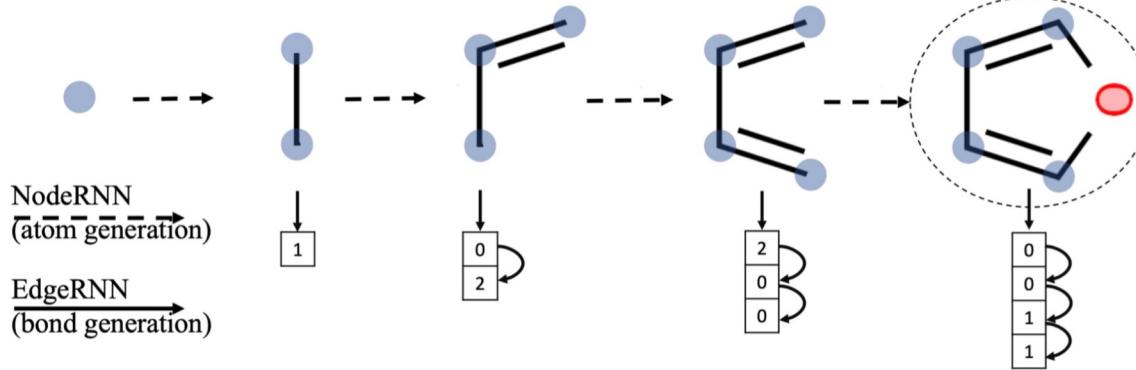
Edge-level задачи



Сделать рекомендацию на друзей в социальной сети

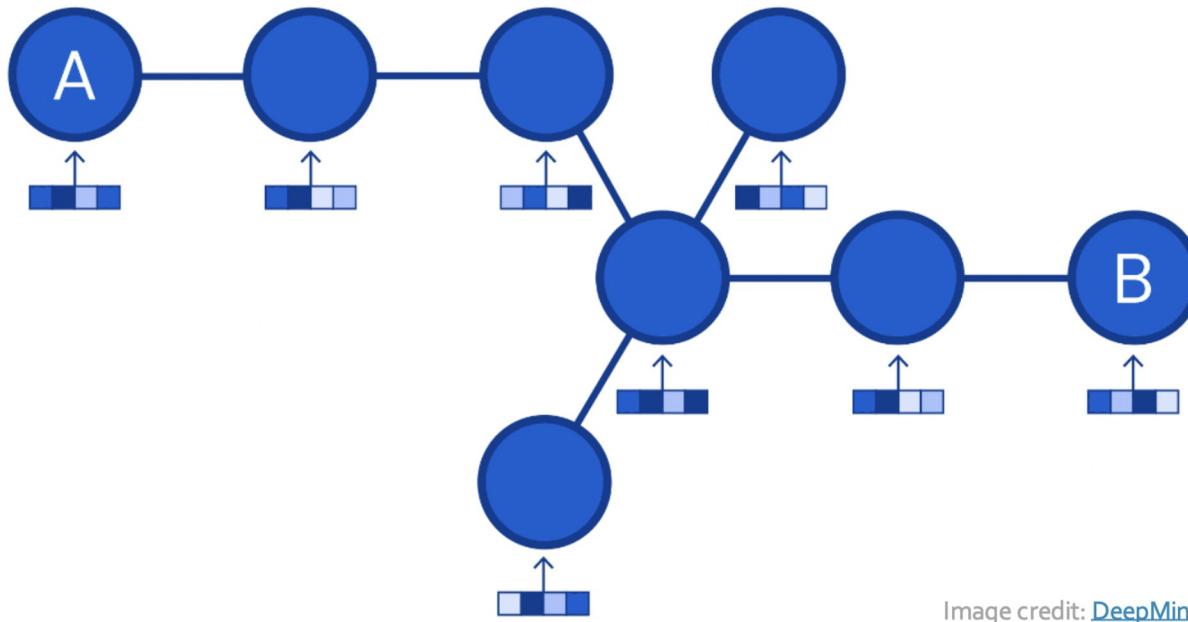
Для всех пар v_i, v_j хотим сделать предсказание $f(v_i, v_j)$

Graph-level задачи



Хотим научиться создавать графы с определенными характеристиками

И subgraph-level задачи



Предсказание времени прибытия машины (разбиение дороги на сегменты (вершины) и доступность из одного сегмента в другой (ребра))

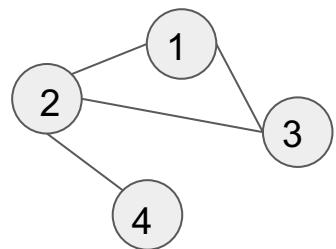
Особенности алгоритмов на графах

Для работы со сложными системами возникает определенная проблема - выбор представления для использования методов анализа

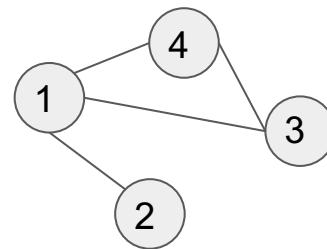
Существуют некоторые базовые требования к алгоритмов на графах

Особенности алгоритмов на графах

Алгоритмы на графах должны работать независимо от перестановки индексов вершин

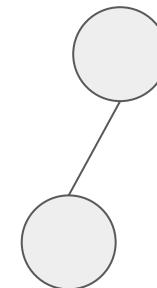
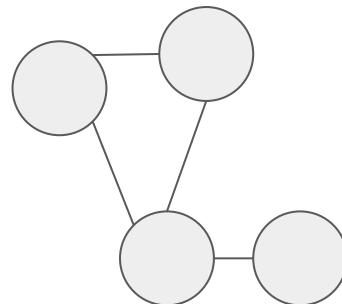
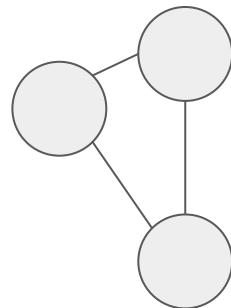


=



Особенности алгоритмов на графах

Алгоритмы на графах должны уметь принимать на вход графы разных размеров



Выбор представления данных

Важным этапом перед применением глубинного обучения является выбор представления

Нужно разобраться какими свойствами должен обладать граф для представления данных

До глубинного обучения

Прежде чем переходить к основным методам глубинного обучения следует ознакомиться с существующими подходами для создания так называемых hand-crafted признаков

По аналогии с типами существующими задач для анализа графовых данных, будем разбирать признаки для вершин, ребер и графов

ВАЖНО: Предполагаем, что все графы будут неориентированными

Постановка задачи машинного обучения на графах

Задача: сделать предсказание для ряда объектов

Признаки:

Объекты:

Функция потерь:

Постановка задачи

Задача: сделать предсказание для ряда объектов

Признаки: n -мерные векторы

Объекты: вершины, ребра, графы

Функция потерь: Зависит от задачи

Node-level statistics

- степень вершины
- центральность
- коэффициент кластеризации
- графлеты

Степень вершины

Определение: степенью вершины называется число ребер, инцидентных этой вершине

В случае неориентированных графов без петлей можно считать что степенью вершины называется количество соседей этой вершины

$$d_u = \sum_{v \in V} \mathbf{A}[u, v]$$

Центральность

Степень вершины не позволяет понять **значимость** вершины в контексте графа

Для более точного понимания значения вершины в графе можно посмотреть на **меру центральности**

Существуют различные способы посмотреть на эту статистику:

- с точки зрения собственных векторов
- с точки зрения соседства
- с точки зрения близости

Центральность с помощью собственных значений

Определение: центральностью называется сумма центральностей всех соседей

Определение рекурсивное - не очень удобное, нужно аналитическое

$$e_u = \frac{1}{\lambda} \sum_{v \in V} \mathbf{A}[u, v] e_v \quad \forall u \in \mathcal{V},$$

Центральностью с помощью собственных значений

Если определить **e** как вектор центральностей вершин, то можно перейти к стандартному уравнению для собственных значений матрицы смежности

Получается **e** - собственный вектор матрицы смежности **A**, для

$$\lambda \mathbf{e} = \mathbf{A}\mathbf{e}$$

Центральность с помощью соседства

Можно сделать предположение, что вершина v важная если она лежит на большом количестве кратчайших путей между другими вершинами, которые содержат эту вершину

$$C_v = \frac{\sum_{s \neq v \neq t} \text{количество кратчайших путей между } s \text{ и } t \text{ содержащих } v}{\text{количество кратчайших путей между } s \text{ и } t}$$

Центральность с помощью близости

Можно предположить что вершина важная, если она находится очень близко ко всем вершинам

$$c_v = \frac{1}{\sum_{u \neq v} \text{длина кратчайшего пути между } u \text{ и } v}$$

Коэффициент кластеризации

Для определения сообществ можно посчитать статистику для вершины, показывающую соединенность для всех соседей вершины

$$c_u = \frac{|(v_1, v_2) \in \mathcal{E} : v_1, v_2 \in \mathcal{N}(u)|}{\binom{d_u}{2}}.$$

От коэффициента кластеризации к графлётам

Если заметить, то коэффициент кластеризации считает число
“треугольников”

Возникает закономерный вопрос - а что будет если считать не только
треугольники?

Графлеты

Графлетьми называется подграфы, описывающие структуру сети соседей вершины

По сути, можно сказать что это обобщение для предыдущих способов описания признаков:

- степень графа это количество ребер, которые связаны с вершиной
- коэффициент кластеризации это количество треугольников, которые связаны с вершиной

Графлеты

Различных графлетов размера 2 - 5 существует 73 (в то время как неизоморфных графов 30), соответственно вектор графлетов размером это 73 числа показывающее, сколько графлектом под определенным номером касается наша вершина (размер, естественно, не ограничен пятью)

Смысл графлетной степени вершины - это дополнительная информация о локальной топологии

Графлеты

Определение: порожденный подграф - подмножество вершин и ребер графа, где вершины соединены если они соединены в оригинальном графе

Определение: изоморфные графы - 2 графа называются изоморфными если у них одинаковое число вершин соединенных одинаково

Определение: графлеты - соединенные неизоморфные порожденные подграфы

Предсказание пропущенных связей

Дано: набор ребер

Выход: новые ребра

Как придумать признаки для пары вершин?

Предсказание пропущенных связей. 2 типа задачи

- Случайное удаление ребер :

Часть вершин удаляются из графа и ставится цель предсказать их появления

- Ребра с течением времени

Имея момент времени t_0 и t'_0 для графа вывести отранжированный список вершин для моментов времени t_1 и t'_1

Предсказание пропущенных связей

Основная идея состоит в следующем подходе

- считаем для каждой вершин в графе пары score
- ранжируем
- лучшие n предсказываем как новые связи
- проверяем, какие на самом деле появились в итоге состояний графа в t_1 и t'_2

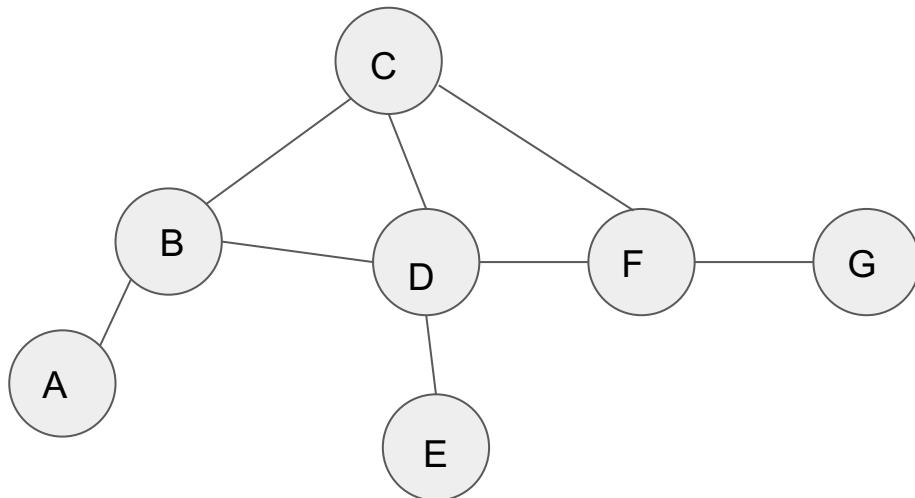
Предсказание пропущенных связей : признаки

Рассмотрим следующие виды признаков

- дистанция
- локальное пересечение
- глобальное пересечение

Дистанция

Дистанцией между парой вершин можно считать длину кратчайшего пути между ними



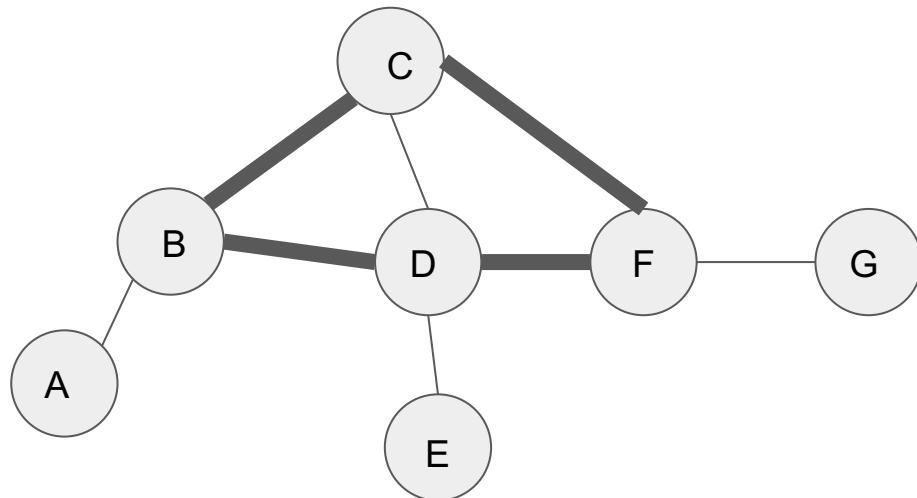
$$AC = CE = 2$$

$$BF = 2$$

$$BG = 3$$

Дистанция

Кратчайшие пути BE и BF равны, но из B в F можно существует два кратчайших пути. Из этого можно извлечь больше информации.



Локальное пересечение

Имеет смысл смотреть на множества соседей двух вершин.

- **Общие соседи**

$$\mathbf{S}[u, v] = |\mathcal{N}(u) \cap \mathcal{N}(v)|$$

- **Коэффициент Жаккара**

$$\mathbf{S}_{\text{Jaccard}}[u, v] = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}$$

- **Индекс Адамика-Адара**

$$\mathbf{S}_{\text{AA}}[v_1, v_2] = \sum_{u \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)} \frac{1}{\log(d_u)}$$

Проблемы локального пересечения

В случае если вершины не пересекаются, то значения метрики выбранной для локального пересечения будут равны нулю.

Это имеет смысл с точки зрения текущего состояния, но в случае изменения графа вершины могут иметь связь через некоторое время. Для большего количества информации следует смотреть на граф целиком.

Глобальное пересечение

На что имеет смысл смотреть для произвольного графа, если мы хотим посмотреть на связь двух далеких друг от друга вершин?

Напрашивается - посчитать количество всех путей между двумя вершинами.
(Индекс Каца)

$$\mathbf{S}_{\text{Katz}}[u, v] = \sum_{i=1}^{\infty} \beta^i \mathbf{A}^i[u, v],$$

На уровне графа

Отдельно разобрав признаки, которые характеризуют вершины и связи, пришло время разобраться, как можно охарактеризовать граф

Один из вариантов - взять просто агрегировать статистики вершин (в теории неплохо для старта)

На помощь приходят ядра. После его задания можно вернуться к классическому машинному обучению (kernel SVM).

- Графлетовые ядра
- Weisfeiler - Lehman kernel

Графлетовые ядра

Основная идея - позаимствовать идею мешка слов, заменив его на мешок вершин (*bag of nodes*).

Для создания графлетового ядра примем следующие уточнения:

- вершины в графлете не должны быть соединены
- нет корневой вершины

Графлетовые ядра

Вектор количества графлетов - вектор где указаны количества графлетов в сети

$$K(G, G') = \mathbf{f}_G^T \mathbf{f}_{G'}$$

Графлетовые ядра

При разных G, G' - проблема с результатом

Нужна нормализация

$$\mathbf{h}_G = \frac{\mathbf{f}_G}{\text{Sum}(\mathbf{f}_G)} \quad K(G, G') = \mathbf{h}_G^T \mathbf{h}_{G'}$$

Графлетовые ядра. Проблемы

Очень дорого.

Для размера k графа размера n цена будет n^k

Почему? По причине того что изоморфизм подграфов это NP-hard

Если степень вершины сверху ограничена d то nd^k

Weisfeiler-Lehman Kernel

Идея алгоритма - итеративная агрегация по соседям

Алгоритм:

1. Назначим каждой вершине метку

$$l^{(i)}(v) = \text{HASH}(\{\{l^{(i-1)}(u) \mid u \in \mathcal{N}(v)\}\})$$

2. после k шагов этот вектор суммирует структуру о k-hop соседстве

WL Kernel

Хороший способ - вычислительно эффективен

Вычисляя ядро требуется следить только за цветами, появляющимися в двух графах

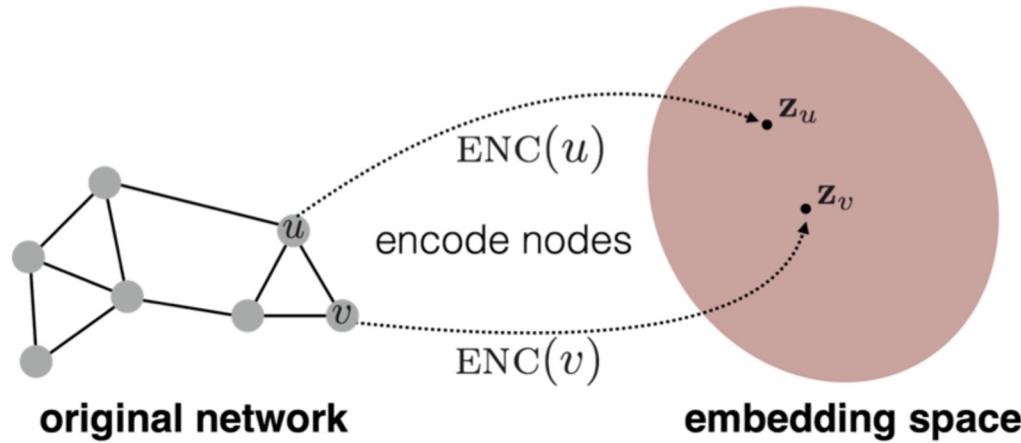
Считать цвета - линия

В общем получается линия по ребрам

Эмбеддинги

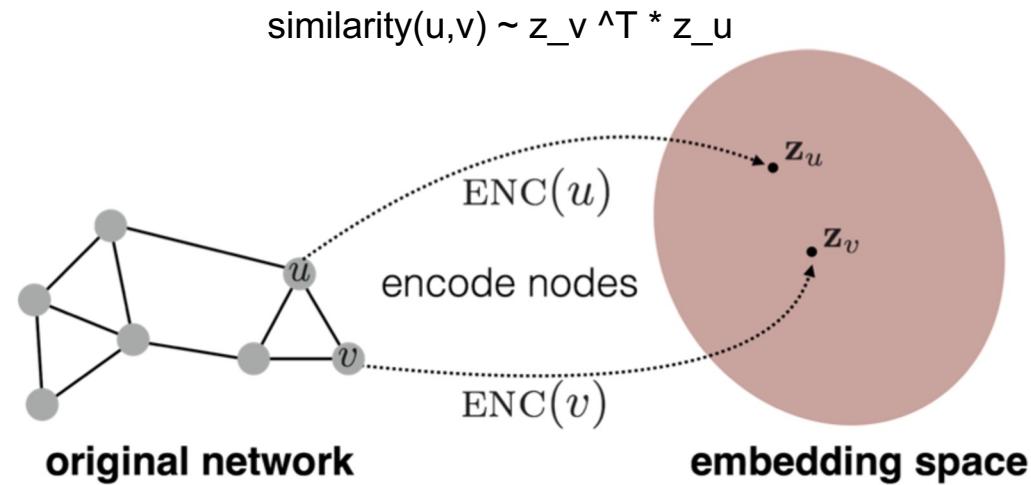
Эмбеддинги вершин - векторы, суммирующие позицию вершины в графе и локальное соседство

По сути хотим получить проекцию в пространство, где геометрические соотношения векторов будут отображать связи в оригинальном графе



Эмбеддинги

Перед нами стоит задача как-то закодировать вершины, чтобы похожесть в эмбеддинговом пространстве позволяла аппроксимировать похожесть в графе



Encoder

Encoder - функция, которая проецирует вершины графа в эмбеддинговое пространство

$$v \rightarrow z_v \quad \text{ENC} : \mathcal{V} \rightarrow \mathbb{R}^d$$

$$\text{ENC}(v) = \mathbf{Z}[v]$$

Decoder

Decoder - функция, по представлению реконструирующая оригинал и связи

В простом случае парный декодировщик позволяет спрогнозировать связаны ли две вершины в графе

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$$

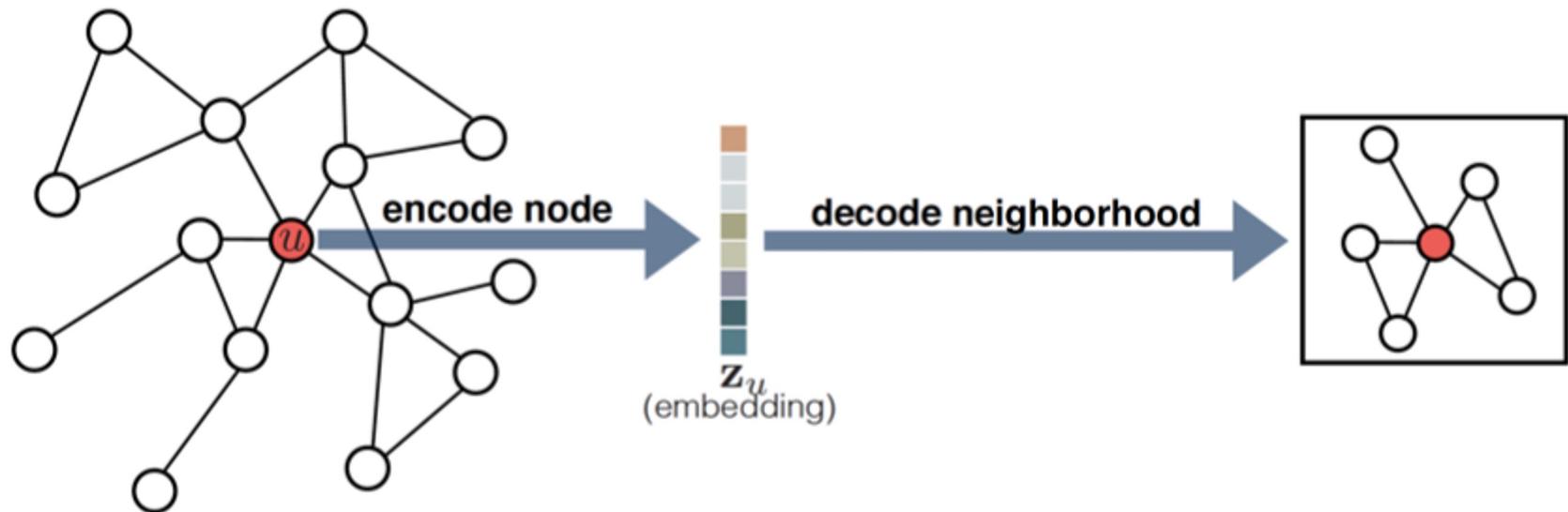
$$\text{DEC}(\text{ENC}(u), \text{ENC}(v)) = \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u, v].$$

Парадигма

1. encoder переводит вершины в пространство эмбеддингов
2. определяем каким-то образом похожесть вершин в графе
3. decoder переводит эмбеддинги в степень похожести
4. оптимизируем параметры encoder'а для достижения следующей цели

$$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

Парадигма



Оптимизация

Задачу обучения декодировщика можно описать как задачу минимизации следующего функционала:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \ell (\text{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v])$$

Обзор

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\ \mathbf{z}_u - \mathbf{z}_v\ _2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], \dots, \mathbf{A}^k[u, v]$	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v u)$	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v u)$ (biased)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

Случайные блуждания

Обсужденные ранее методы как правило определяют S как некую полиномиальную функцию, и оптимизируется похожесть произведения двух эмбеддингов и $S[u,v]$

В последнее время успешными начинают быть методы, которые связаны с вероятностным подходом к похожести эмбеддингов, основанными на том, как часто вершины взаимно встречаются на случайных проходах

Альтернативные подходы

- Можно совершать “прыжки” в блужданиях
- struct2vec

Проблемы

- каждая вершина в таком подходе рассматривается индивидуально
- не используют признаки вершин
- не способны работать с новыми вершинами

решим потом) (on the way to GNN)

Графы знаний (или multi-relational graphs)

Граф - $G = (V, E)$, V - вершины, E - ребра

В обычном графе $e = (u, v)$

В графе знаний $e = (u, t, v)$

В общем случае на графах знаний решается задача предсказания пропущенных связей, но бывают и задачи классификации вершин

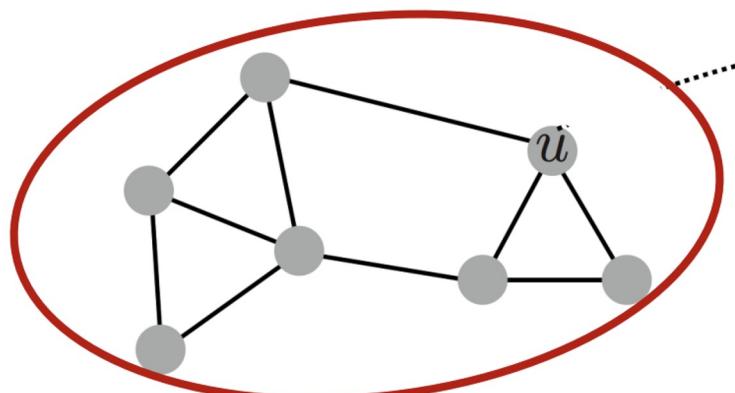
Обзор

Name	Decoder	Relation Parameters
RESCAL	$\mathbf{z}_u^\top \mathbf{R}_\tau \mathbf{z}_v$	$\mathbf{R}_\tau \in \mathbb{R}^{d \times d}$
TransE	$-\ \mathbf{z}_u + \mathbf{r}_\tau - \mathbf{z}_v\ $	$\mathbf{r}_\tau \in \mathbb{R}^d$
TransX	$-\ g_{1,\tau}(\mathbf{z}_u) + \mathbf{r}_\tau - g_{2,\tau}(\mathbf{z}_v)\ $	$\mathbf{r}_\tau \in \mathbb{R}^d, g_{1,\tau}, g_{2,\tau} \in \mathbb{R}^d \rightarrow \mathbb{R}^d$
DistMult	$\langle \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v \rangle$	$\mathbf{r}_\tau \in \mathbb{R}^d$
ComplEx	$\text{Re}(\langle \mathbf{z}_u, \mathbf{r}_\tau, \bar{\mathbf{z}}_v \rangle)$	$\mathbf{r}_\tau \in \mathbb{C}^d$
RotateE	$-\ \mathbf{z}_u \circ \mathbf{r}_\tau - \mathbf{z}_v\ $	$\mathbf{r}_\tau \in \mathbb{C}^d$

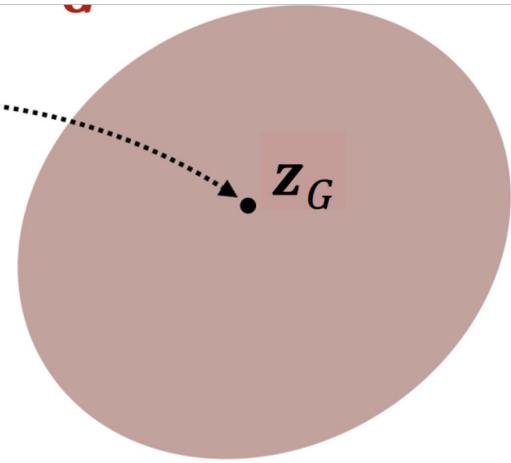
Свойства

- симметрия (асимметрия)
- инверсионность
- композитность

Эмбеддинги графов



original network

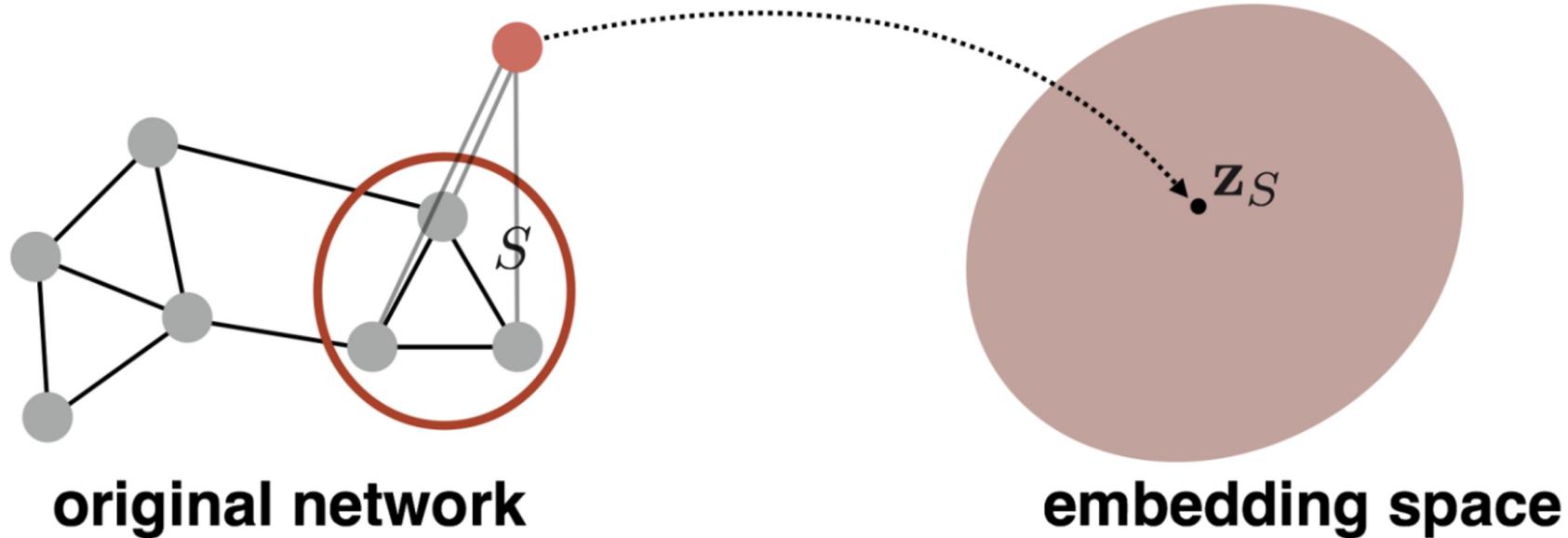


embedding space

Среднее

$$z_G = \sum_{v \in G} z_v$$

Виртуальная вершина



Анонимные случайные блуждания

1. Сэмплируем анонимные случайные блуждания и отображаем граф как частоты появления случайных блужданий
2. Обучаем эмбеддинги графа вместе с анонимными случайными блужданиями

PageRank

- это алгоритм, используемый для ранжирования страниц в интернете

Для сетевого анализа это наглядный пример использования всех полученных ранее знаний для создания готового способа решения насущной проблемы поиска

Что есть страницы в интернете

Будем рассматривать интернет в простом приближении (как это было на заре)

В интернете есть страницы (pages)

На страницах есть ссылки на другие страницы

Стоит **задача** - понять, какие страницы самые важные

Связь с графиками

В таком простом приближении можно удобно представить интернет как огромный граф.

Вершины - страницы

Ребра - ссылки

Соответственно граф наш будет ориентированный - ребро между двумя вершинами будет направленным, оно будет означать что на странице А есть ссылка на страницу Б

Особенности такого графа

- Граф будет громадный
- Могут быть петли (на странице есть ссылка на саму себя)
- Ребра могут быть направлены из А в Б и из Б в А в одно и то же время
(пример - главная страница ведет на подстраницу, на которой есть опция прыгнуть на главную страницу)

Интуиция

Самая очевидная идея - страница важна, если у нее очень много ссылок

Сразу возникает вопрос - какие ссылки важнее, **исходящие или входящие?**

Более того, возникает еще вопрос - а все ли ссылки **одинаково** важны?

Интуиция. Модель

Ссылка от важной страницы должна сигнализировать о важности страницы

Опишем модель:

- Вклад каждой ссылки должен быть пропорционален важности страницы, от которой она исходит
- Если страница i с важностью r_i имеет d_i ссылок, то каждая исходящая ссылка будет давать вклад r_i/d_i
- Страница j будет иметь важность, равную сумме входящих ссылок

рекурсивненько как-то...

Ранг

Как решать? Гауссов метод в случае многих миллиардов страниц будет не очень хорош

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

Матричная форма

Стochasticная матрица смежности:

Если есть ссылка из i в j , то $M_{ij} = 1/d_i$

Тогда переписать уравнения можно в матричном виде

$$r = M r$$

Интуиция за матричной формой

Попробуем понять, в чем смысл $r = M r$

Представим какого-то случайного пользователя в сети Интернет:

1. В момент времени t он попадает на страницу i
2. В момент времени $t+1$ он следует по случайной ссылке со страницы i
3. Он оказывается на странице j , попав на нее из i (**важно: j может быть страницей i**)
4. Процесс продолжается бесконечно

Пусть $p(t)$ - вектор, у которого на i -том месте будет вероятность того, что юзер окажется на странице i в момент времени t

Тогда $p(t)$ - вероятностное распределение на всех страницах

Интуиция за матричной формой

Движение по ссылкам случайным равновероятностным образом можно описать как $p(t+1) = M p(t)$

Представим, что в какой-то момент $p(t+1) = M p(t) = p(t)$

Тогда $p(t)$ - стационарное распределение случайного блуждания

Вспоминая $r = Mr$ получим, что r - стационарное распределение

Более того...

Вспомним про центральность через собственное значение

$$\lambda c = Ac$$

λ - собственное значение, c - собственный вектор

У нас $r = Mr$

Тогда подставим $\lambda = 1$, получим $1 r = M r$

Вывод

Объединив все три идеи получим вывод -

r - собственный вектор стохастической матрицы смежности M с собственным значением 1

Начиная с любого вектора и $M(M(\dots(Mu)))$ - долгосрочное распределение блуждающих юзеров

PageRank = Ограничивающее распределение = Главный собственный вектор M

Теперь можно решить задачу.

Решение PageRank

1. В начале назначим каждой вершине начальный ранг
2. Продолжать до сходимости (норма разницы между рангами в моменты $t+1$ и t меньше некого эпсилон)

Power Iteration

- алгоритм для решения задачи
- 1. инициализация - $r^0 = [1/N, \dots, 1/N]$
- 2. шаг $r^{t+1} = M r^t$
- 3. повторять, пока не $|r^{t+1} - r^t|_1 < \varepsilon$
 - a. $r := r^{t+1}$

Какие вопросы возникают к PageRank?

- тупики (ломают)
- циклы (сходимость будет, но результаты не совсем те)

Уход от циклов

С помощью вероятностей можно научиться выбираться из цикла

- На каждом шаге с вероятностью β юзер выбирает одну из d_i ссылок на странице
- С вероятностью $1 - \beta$ юзер телепортируется

За конечное число шагов из цикла юзер выпрыгнет

Уход от тупиков

- заранее договориться, что в тупике сработает случайный телепорт

PageRank

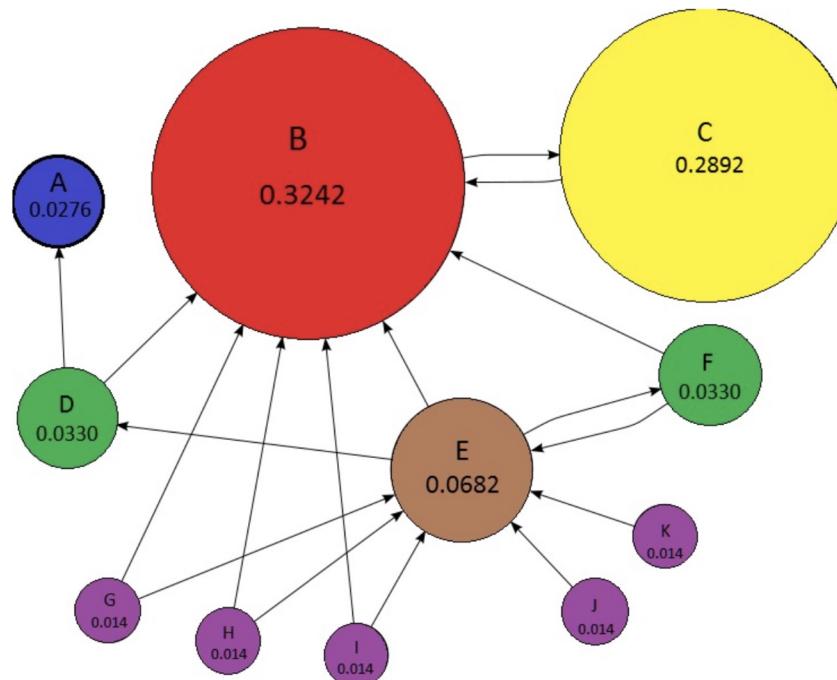
$$r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

Новая матрица

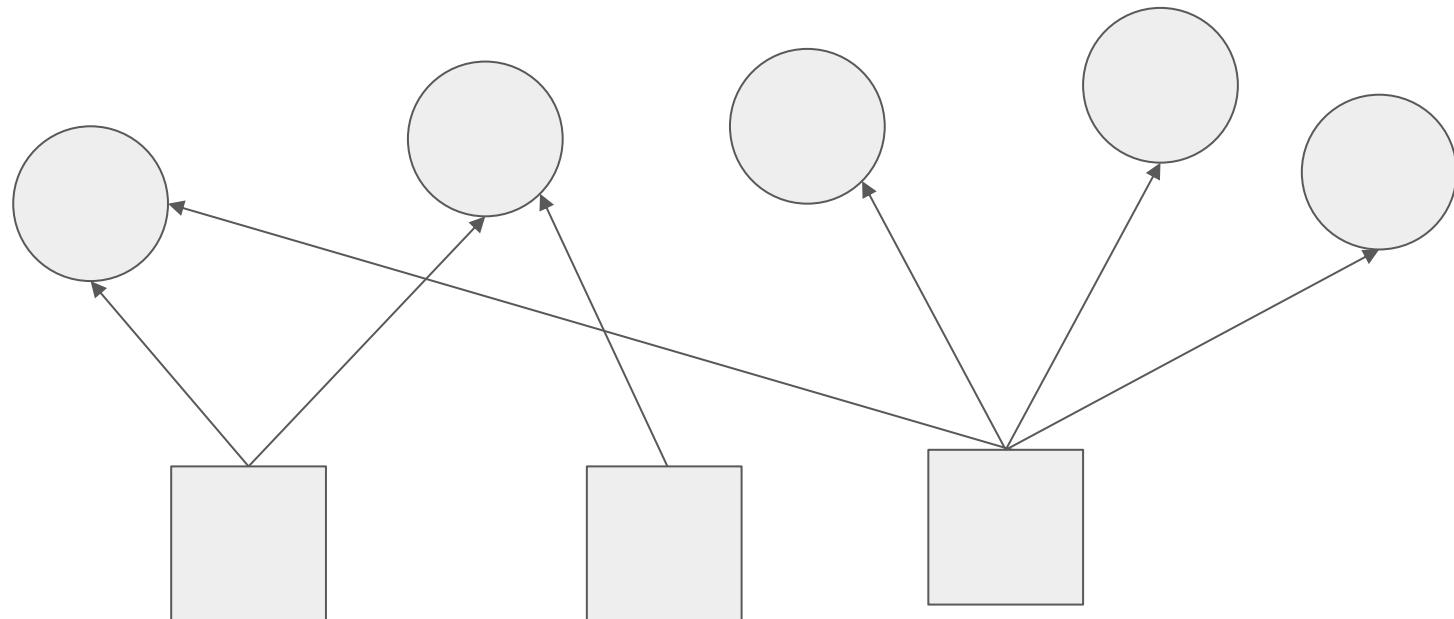
$$G = \beta M + (1 - \beta) \begin{bmatrix} 1 \\ N \end{bmatrix}_{N \times N} \quad \dots$$

И получаем задачу $r = Gr$

PageRank пример



Рекомендации



Приложение к рекомендациям

- Меняем формат телепортации, ограничивая множество вершин, в которые будет совершен обратный прыжок
- Максимально похожий на какой-то элемент можно найти, ограничив множество для телепортации до одной конкретной вершины