Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт
(государственный университет)»

Физтех-школа радиотехники и компьютерных технологий

Факультет радиотехники и кибернетики

Кафедра проблем передачи информации и анализа данных

АНАЛИЗ ПРОБЛЕМЫ ВЛОЖЕНИЯ ГРАФОВ И ПРИМЕНИМОСТИ К НЕЙ НОВОГО МЕТОДА – СТРУКТУРНОЙ ФУНКЦИИ ПОТЕРЬ

Выпускная квалификационная работа
(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:
студент 411а группы _____ Цепа Станислав Евгеньевич

Научный руководитель:
к.ф.-м.н. _____ Панов Максим Евгеньевич

Москва 2018

Содержание

1 Введение

Большинство алгоритмов машинного обучения основано на использовании признаков, являющихся числами (а также иногда категориями). Однако, далеко не все сущности, которые хотелось бы использовать в качестве признаков, можно легко перевести в числовой вид. Примерами таких объектов являются слова в тексте или вершины в графе. В таком случае возникает задача вложения[1] – как каждой отдельной сущности сопоставить точку вещественного d-мерного пространства, или, проще говоря, d вещественных чисел, чтобы дальше можно было использовать их как признаки в задачах машинного обучения.[2]

Полученные признаки позволяют использовать эти сущности при классификации (пример: определение эмоциональной окраски слова), кластеризации (пример: определение групп знакомых в социальной сети) или визуализации (пример: визуальное определение взаимосвязей между словами, см рис. 1)

Мы рассматриваем задачу вложения графов, то есть определения таких точек в d-мерном пространстве, чтобы взаимное расположение этих точек лучшим образом отображало структуру графа.

Существуют многие принципиально разные виды вложений графа. Например, нашей целью может быть построение вложения для ребер или вершин графа, или же для его подграфов. Наиболее распространенной задачей является, конечно, задача вложения вершин, поскольку они обычно означают отдельные объекты: людей в социальной сети или статьи в графе цитирования. В задаче вложения вершин существует также отдельное направление, в котором вершины располагаются вблизи друг от друга не тогда, когда они рядом в графе, а тогда, когда они имеют одинаковую структурную роль, например это могут быть «хабы» или «листовые» вершины.

В нашей работе мы сосредоточились на самом простом и естественном варианте вложения вершин графе – основанном на их локальном соседстве. От нашего алгоритма вложения мы ожидаем, что если две вершины имеют между собой ребро или же общего соседа, то они будут близко располагаться в пространстве вложения. Если же вершины не связаны между собой, то лучшим вариантом будет, если в пространстве вложения они будут линейно независимы.

Мы предлагаем принципиально новый метод вложения. Его основная идея в том, чтобы выделить статистическую характеристику, отражающую оптимальность вложения, и оптимизировать ее методом стохастического градиентного спуска. Преимущества нашего метода это простота и наглядность.

Идея нашего алгоритма была взята из [8], где похожим методом классифицируются изображения. После экспериментов было обнаружено что алгоритм из этой статьи не дает должного качества в случае работы с графами. Наша работа описывает улучшенную версию [8], адаптированную для работы с графами.

Мы проверяем качество вложения тремя способами, сравнивая результаты с результатами существующих алгоритмов.

2 Постановка задачи

В данном разделе мы формализуем задачу вложения графов и демонстрируем как можно разделить ее на подзадачи. Наша формализация основана на формализации, приведенной в [4].

[1] Термин «вложение» (*embedding*) означает что мы «вкладываем» сущность, не имеющую явных числовых признаков, в числовое пространство.

[2] Чаще всего предполагается, что $d \ll n$, где n – количество объектов.

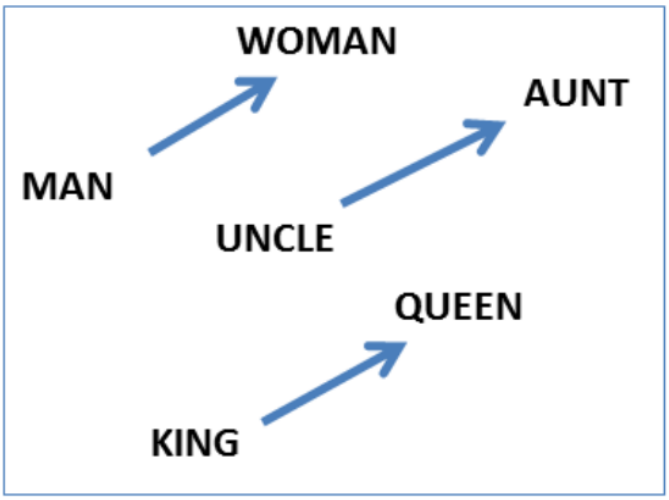


Рис. 1: Результат работы word2vec в двумерном пространстве, источник `https://blog.acolyer.org/word2vec-gender-relation/`

Пусть дан невзвешенный граф $G = (V, E)$, $n = |V|$. Его матрица смежности $A \in \mathbb{R}^{n \times n}$, $A_{ij} \in \{0, 1\}$.

Также вводится матрица сходства S, $S_{ij} \in \mathbb{R}_+$, каждый элемент которой оценивает некое "сходство"двух вершин. Обычно, матрицу сходства получают из матрицы смежности таким образом, чтобы учитывалось соседство второго и более высших порядков. Стоит заметить, что не во всех алгоритмах возможно использовать небинарную матрицу, например, в нашем алгоритме это возможно только при некоторых усложнениях, поэтому, хоть далее мы и будем для общности везде указывать матрицу S, в основном будет подразумеваться $S = A$.

Алгоритм, строящий вложение графа по этим данным, обычно состоит из следующих частей:

- функция кодирования (*encoder*) строит вложение графа,

$$\mathrm{enc} \colon V \to \mathbb{R}^{n \times d},$$

  далее будем обозначать вложение i вершины $\mathcal{E}_i = \mathrm{enc}(v_i)$;

- функция декодирования (*decoder*) оценивает сходство двух элементов вложения, в идеальном случае $\mathrm{dec}(\mathcal{E}_i, \mathcal{E}_j) = S_{ij}$,

$$\mathrm{dec} \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R};$$

- функция потерь (*loss*) $\mathcal{L}$ оценивает, насколько хорошо функции декодирования удалось восстановить элементы исходной матрицы сходства.

Таким образом, задача вложения графа сводится к поиску подходящего функционала кодирования и выбору правильной функции потерь. После этого решается задача оптимизации (при помощи градиентного спуска или аналитически). Далее мы рассмотрим каждую из составляющих алгоритма подробнее.

2.1 Функция кодирования

Определением функции кодирования можно считать

$$\text{enc}(v_i) = \mathcal{E}\mathbf{e}_i. \tag{1}$$

Здесь $\mathcal{E}$ – матрица вложения размера $n \times d$, а $\mathbf{e}_i$ – единичный вектор. Представление (1) является наиболее общим, в нем вложение каждой вершины может быть произвольным.

Однако, логично предположить, что для вершин, имеющих похожие вектора $\mathbf{s}_i$ в матрице сходства, и вложения должны быть близкими, из чего можно представить

$$\text{enc}(v_i) = f(\mathbf{s}_i), \tag{2}$$

где f – функция $\mathbb{R}^n \to \mathbb{R}^d$, а $\mathbf{s}_i$ – i столбец матрицы сходства.

В нашей работе мы принимаем, что f линейная функция, и (2) превращается в

$$\text{enc}(v_i) = \mathbf{W}\mathbf{s}_i + \mathbf{b} \tag{3}$$

или

$$\mathcal{E} = \mathbf{W}S + \mathbf{b}. \tag{4}$$

Здесь размер матриц $\mathbf{W}$ и $\mathbf{b}$ это $n \times d$. Стоит заметить, что если матрица S в выражении (4) имеет ранг хотя бы d, то матрица $\mathcal{E}$ может принимать любые значения.

2.2 Функция декодирования

Функция декодирования обычно имеет простой вид, поскольку она представляет из себя вычисление какой либо меры сходства между двумя d-мерными векторами. Вот примеры такой функции:

- евклидово расстояние
$$\text{dec}(\mathcal{E}_i, \mathcal{E}_j) = -\|\mathcal{E}_i - \mathcal{E}_j\|_2^2; \tag{5}$$
- скалярное произведение
$$\text{dec}(\mathcal{E}_i, \mathcal{E}_j) = \mathcal{E}_i^T \mathcal{E}_j; \tag{6}$$
- корреляция, нормированное скалярное произведение
$$\text{dec}(\mathcal{E}_i, \mathcal{E}_j) = \frac{\mathcal{E}_i^T \mathcal{E}_j}{|\mathcal{E}_i||\mathcal{E}_j|}; \tag{7}$$
- softmax
$$\text{dec}(\mathcal{E}_i, \mathcal{E}_j) = \frac{e^{\mathcal{E}_i^T \mathcal{E}_j}}{\sum_{k \in V} e^{\mathcal{E}_i^T \mathcal{E}_k}}.$$

2.3 Функция потерь

Существующие функции потерь могут быть условно разделены на два класса: попарная функция потерь и глобальная функция потерь.

2.3.1 Попарная функция потерь

Эта функция потерь представляет из себя сумму ошибок по каждому элементу S,

$$\mathcal{L} = \sum_{i,j \in V} \ell(\text{dec}(\text{enc}(v_i), \text{enc}(v_j)), S_{ij}). \tag{8}$$

Обычно функция ℓ представляет из себя квадратичную ошибку:

$$\ell(x, y) = \|x - y\|_2^2. \tag{9}$$

2.3.2 Глобальная функция потерь

В некоторых случаях, функция потерь имеет более сложный вид, и не может быть разложена в сумму. Каждый такой случай можно рассматривать отдельно. К такому виду относится и предлагаемая в работе структурная функция потерь – она пытается вычислять оптимальность вложения, основываясь на его статистических характеристиках.

3 Обзор алгоритмов

Среди уже существующих алгоритмов можно выделить несколько основных групп: методы на основе разложения матриц, методы на основе случайных блужданий, методы на основе нейронных сетей. Далее мы рассмотрим эти методы подробнее.

3.1 Методы на основе разложения матриц

3.1.1 Laplacian Eigenmaps [1]

В алгоритме Laplcian Eigenmaps оптимизируется следующая функция

$$\mathcal{L} = \frac{1}{2} \sum_{i,j \in V} \|\mathcal{E}_i - \mathcal{E}_j\|_2^2 S_{ij}. \tag{10}$$

Видно, что эта функция является (8) с функцией декодирования (5) и $\ell(x, y) = xy$.

3.1.2 HOPE [6]

В алгоритме HOPE вложение получается за счет минимизации

$$\mathcal{L} = \|S - \mathcal{E}^s \mathcal{E}^{tT}\|.$$

Этот алгоритм изначально создавался для ориентированных графов, поэтому в нем появляется два вложения: $\mathcal{E}^s$ и $\mathcal{E}^t$, source и target соответственно. Если ребро идет только из i в j, то $\mathcal{E}_i^s$ должно быть близко к $\mathcal{E}_j^t$, а $\mathcal{E}_i^t$ наоборот далеко от $\mathcal{E}_j^s$.

В алгоритме HOPE производится сингулярное разложение матрицы S,

$$S = \sum_{i=1}^{n} \sigma_i \mathbf{v}_i^s \mathbf{v}_j^{tT}.$$

Оптимальное вложение размерности d получается, если взять d самых больших сингулярных чисел.

$$\mathcal{E}^s = [\sqrt{\sigma_1}\mathbf{v}_1^s, \ldots, \sqrt{\sigma_d}\mathbf{v}_d^s],$$

$$\mathcal{E}^t = [\sqrt{\sigma_1}\mathbf{v}_1^t, \ldots, \sqrt{\sigma_d}\mathbf{v}_d^t].$$

Этот алгоритм примечателен тем, что в нем может использоваться любая матрица сходства, в том числе для взвешенного и ориентированного графа.

Как описано в [4], этот алгоритм можно привести к виду (8), где функция декодирования это скалярное произведение (6), а функция ℓ – квадратичная ошибка (9).

3.2 Методы на основе случайных блужданий

Метод случайных блужданий заключается в том, что граф переводится в набор признаков, описывающих локальное соседство вершин, и далее решается задача из области обработки текстовой информации (*word embedding*). В приведенных ниже примерах применяется широко распространенный алгоритм word2vec.

3.2.1 Deepwalk [7]

Для данного графа и стартовой точки на каждом шаге мы случайно выбираем соседнюю вершину и перемещаемся в нее, продолжая так определенное количество итераций. Запоминая результат, мы получаем последовательность вершин, которая называется случайным блужданием на графе. Далее каждая вершина интерпретируется как слово, а одно случайное блуждание как предложение. Используя механизм поиска скрытых представлений для слов word2vec, методы получают искомое вложение для вершин графа. В данном алгоритме переходы в любую соседнюю вершину равновероятны.

Подход алгоритмов из области вложения слов заключается в том, что мы подсчитываем вероятность встретить слово, если известен его контекст (несколько слов до и после него) или, наоборот, зная слово, определить вероятность слов в его контексте. Второй вариант и был реализован в deepwalk. Итоговая задача оптимизации формулируется как

$$\min_{\text{enc}} \; -\log P(\{v_{i-k}, \ldots v_{i-1}, v_{i+1}, \ldots, v_{i+k}\} \mid \text{enc}(v_i)),$$

где набор вершин $\{v_{i-k}, \ldots v_{i+k}\}$ является случайным блужданием.

Как описано в [4], алгоритм deepwalk можно свести к виду (8). При этом,

$$\text{dec}(\mathcal{E}_i, \mathcal{E}_j) = \frac{e^{\mathcal{E}_i^T \mathcal{E}_j}}{\sum_{k \in V} e^{\mathcal{E}_i^T \mathcal{E}_k}} \approx p_T(v_i | v_j),$$

где $p_T(v_i|v_j)$ – вероятность попасть из v_i в v_j за T шагов при случайном блуждании. Функция потерь определяется как

$$\mathcal{L} = \sum_{i,j \in D} -\log(\text{dec}(\mathcal{E}_i, \mathcal{E}_j)),$$

где подсчет ведется в множестве пар соседних вершин в случайных блужданиях.

3.2.2 Node2Vec [3]

Данный алгоритм является усовершенствованием алгоритма deepwalk. Основное отличие заключается в том, что процесс блуждания регулируется двумя параметрами p и q, контролирующими вероятность возвращения в вершину и, соответственно, удаления от вершины. Путем настройки этих параметров удается подобрать блуждания, лучше соответствующие структуре конкретного графа.

3.3 Методы на основе нейронных сетей

Стоит заметить, что задача вложения является задачей обучения без учителя, поэтому методы на основе нейронных сетей необходимо сперва адаптировать для использования в задачах без учителя. Одним из вариантов такой адаптации является использование автоэнкодера – сети в которой выходной слой равен по размеру входному, а посередине слой размером с искомое вложение. Таким образом, если алгоритму оптимизации удастся максимально приблизить выходной слой к входному то данные из срединного слоя можно будет считать вложением. Ниже приведен пример реализации алгоритма вложения при помощи автоэнкодера.

3.3.1 SDNE [9]

Этот метод представляет из себя автоэнкодер с количеством скрытых слоев $2K-1$. На вход подается $\mathbf{x}_i$ – i столбец S (здесь также $S=A$), слой $\mathbf{y}_i^{(K)}$ является вложением. Слои после K повторяют входные в обратном порядке, выход крайнего слоя - $\hat{\mathbf{x}}_i$.

Вводится две функции потерь:

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 s_{ij},$$

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b}_i\|_2^2 = \|(\hat{\mathbf{X}} - \mathbf{X}) \odot \mathbf{B}\|_F^2,$$

где $b_{ij} = 1$, если $s_{ij} = 0$, и $b_{ij} = \beta > 1$, если $s_{ij} > 0$.

Видно, что здесь $\mathcal{L}_{1st}$ является обычной попарной функцией потерь (8), а $\mathcal{L}_{2nd}$ это функция потерь, отвечающая за работу нейронной сети как автоэнкодера.

3.4 Методы получения матрицы сходства

Также стоит обсудить различные подходы к получению матрицы сходства из матрицы смежности. Чаще всего, это преобразование преследует цель добавить в матрицу смежности информацию о соседстве второго и более высоких порядков, чтобы алгоритмы вложения работали более эффективно.

3.4.1 Метод общих соседей

Это самый простой метод, в котором S_{ij} равняется количеству общих соседей у вершин i и j:

$$S = A^2.$$

3.4.2 Adamic-Adar

Этот метод можно назвать взвешенным методом общих соседей, здесь

$$S = ADA,$$

где D это диагональная матрица,

$$D_{ii} = \frac{1}{\sum_{j=1}^{n} A_{ij}}.$$

Обычно этот метод более удобен благодаря нормировке.

4 Структурная функция потерь

Основная идея нашего исследования заключалась в том, чтобы найти такую статистическую характеристику вложения, оптимизируя которую (при помощи градиентного спуска), мы бы получили вложение хорошего качества. Ниже приведена финальная реализация нашего алгоритма.

4.1 Описание алгоритма

Для каждой пары вершин u, v определим, соединены ли они ребром (положительная пара) или нет (отрицательная пара). Определим

$$a_{uv} = \begin{cases} 1, & \text{если } (u,v) \in E, \\ 0, & \text{если } (u,v) \notin E. \end{cases}$$

Легко заметить что a_{uv} есть не что иное как элементы матрицы смежности (в случае невзвешенного графа). То есть положительные пары для графа $G = (V, E)$ это все ребра $(u, v) \in E$, отрицательные – ребра $(u, v) \notin E$ соответственно.

Далее случайным образом инициализируем матрицы $\mathbf{W}$ и $\mathbf{b}$, это будут матрицы параметров для нашего градиентного спуска и вычислим

$$\mathcal{E} = \mathbf{W}S + \mathbf{b}.$$

Здесь в уравнение входит матрица сходства S, но пока считаем, что $S = A$.

После произведем попарное вычисление расстояния между для каждого элемента вложения

$$l_{uv} = \mathrm{dec}(\mathcal{E}_u, \mathcal{E}_v).$$

В качестве функции dec решено было выбрать корреляцию (7). В этом случае $l_{uv} \in [-1, 1]$, что будет удобно для дальнейшего использования.

И наконец, разобьем подсчитанные l_{uv} на два множества в соответствии с тем к какой паре, положительной или отрицательной, относится (u, v).

$$L^+ = \{l_{uv} = \mathrm{dec}(\mathcal{E}_u, \mathcal{E}_v) \mid a_{uv} = 1\},$$
$$L^- = \{l_{uv} = \mathrm{dec}(\mathcal{E}_u, \mathcal{E}_v) \mid a_{uv} = 0\}.$$

В итоге мы получили два множества значений попарных корреляций: L^+ и L^-.

Наш подход основан на эвристике, что вложение будет хорошим, если значения внутри L^+ будут максимально близки к 1, а внутри L^- максимально близки к 0[3].

Чтобы формализовать эту эвристику, мы прибегаем к следующему подходу: для множеств L^+ и L^- построим два плотностных распределения при помощи методов непараметрической оценки плотности: P^+, P^-. Также построим две гистограммы соответствующие этим распределениям с количеством бинов bin: H^+ и H^-.

Определим нашу функцию потерь как

$$\mathcal{L} = D(H^-, H^+),$$

где D – функция вычисления расстояния между распределениями (*divergence*). Одна из интересных особенностей нашего алгоритма (и отличие от [8]) в том, что в

[3] Мы выбираем не -1 (минимальное значение корреляции), а 0 потому что стремимся получить линейно независимые вложения для отрицательных пар.

качестве D мы используем усовершенствованное расстояние Вассерштайна [5] для одномерного случая.

Одномерное расстояние Вассерштайна (также называется *earth mover distance*, *EMD*) определяется как

$$EMD(H_1, H_2) = \sum_{i=1}^{bin} |\varphi_i|, \text{ где } \varphi_i = \sum_{j=1}^{i} \left(\frac{H_{1j}}{\|H_1\|_1} - \frac{H_{2j}}{\|H_2\|_1} \right). \tag{11}$$

Однако, оно обладает тем недостатком, что $EMD(H_1, H_2) = EMD(H_2, H_1)$. В нашем случае мы знаем что ситуация, когда H^+ левее H^- не симметрична, а наоборот, антисимметрична. Оказывается, если отказаться в (11) от модуля, то новая метрика как раз будет удовлетворять этому условию.

$$EMD_{asym}(H_1, H_2) = \sum_{i=1}^{bin} \varphi_i, \text{ где } \varphi_i = \sum_{j=1}^{i} \left(\frac{H_{1j}}{\|H_1\|_1} - \frac{H_{2j}}{\|H_2\|_1} \right), \tag{12}$$

$$EMD_{asym}(H_1, H_2) = -EMD_{asym}(H_2, H_1).$$

Расстояние Вассерштайна было выбрано потому, что его оптимизация не только уменьшает пересечение между двумя распределениями, но и стремится максимально «раздвинуть» их, поэтому, при применении (12), H^- слишком сильно «уезжала» влево к значению -1. Однако, в оптимальном вложении значения из H^- должны быть как можно ближе к 0 для того чтобы быть линейно независимыми. Чтобы обойти эту проблему мы решили отбрасывать из P^- все значения с $x < 0$. Таким образом к 0 смещались только те значения, которые были больше 0, а остальные оставались зафиксированы.

$$P^-_{cut}(x) = \begin{cases} P^-(x), & \text{если } x \in [0, 1], \\ 0, & \text{если } x \in [-1, 0). \end{cases}$$

Итак,

$$\mathcal{L} = D(H^-_{cut}, H^+) = EMD_{asym}(H^-_{cut}, H^+). \tag{13}$$

Запишем итоговую задачу оптимизации

$$\mathbf{W}_{opt}, \mathbf{b}_{opt} = \arg\min_{\mathbf{W},\mathbf{b}} \mathcal{L}, \tag{14}$$

$$\mathcal{E}_{opt} = \mathbf{W}_{opt} S + \mathbf{b}_{opt}.$$

5 Эксперименты

В данном разделе приведено экспериментальное исследование нашего алгоритма. Оптимизация (14) производилась при помощи библиотеки Tensorflow методом стохастического градиентного спуска. Матрицы $\mathbf{W}$ и $\mathbf{b}$ инициализировались случайными весами и были заданы как параметры оптимизации.

Также применялось несколько улучшений алгоритма. Например, на каждом шаге оптимизации вычисление градиентов производилось на случайном подмножестве данных, такой метод называется «батчинг» (*batching*). К сожалению, «батчинг» не подходит для графов размером $|V| > 10^5$, поскольку становится очень маловероятно,

Название	Количество вершин	Описание
American college football	115	реальный
Books about US politics	105	реальный
BlogCatalog	10312	реальный
Facebook	4039	реальный
Stochastic Block Model	900	модельный

Таблица 1: Список графов

что вершины в одном «батче» будут связаны. Также учитывался тот факт, что практически в любом графе $|L^-| \gg |L^+|$. В связи с этим на каждой итерации множество L^- сэмплировалось до размера L^+. Эксперименты показали, что применение этих оптимизаций не повлияло на качество итогового вложения, но позволило значительно ускорить работу алгоритма и уменьшить потребление оперативной памяти.

Код всех экспериментов доступен на `github.com/premolab/GraphEmbeddings`.

5.1 Данные для экспериментов

В качестве данных для экспериментов были использованы графы размером от 100 до 10000 вершин. Все графы являются либо модельными, либо реальными графами, часто используемыми для исследования алгоритмов. Данные о графах приведены в таблице 1. Отдельно стоит описать граф SBM (*Stochastic Block Model*). Он является модельным графом с 900 вершинами. Сперва вершины разбиваются на три группы, а затем ребра добавляются случайным образом: между вершинами одной группы – с вероятностью p_{in}, между вершинами разных групп – с вероятностью p_{out}. Значения p_{in} и p_{out} в наших экспериментах: $p_{in} \in \{0.08, 0.1\}$, $p_{out} \in \{0.01, 0.03\}$.

5.2 Описание экспериментов

5.2.1 Задача классификации

Эта задача формулируется таким образом: для какой-то части вершин V задана целевая переменная, например класс или бинарный вектор принадлежности к нескольким классам. Необходимо предсказать значение этой целевой переменной для остальных вершин. Подход к решению этой задачи, который мы рассматриваем здесь, это

1. построить вложение $\mathcal{E} = \mathcal{E}(V, E)$;
2. решить задачу обучения с учителем на числовых признаках $\mathcal{E}$.

Стоит отметить, что пункт 1 в данном алгоритме все также относится к задачам обучения без учителя, то есть целевая переменная в нем никак не используется.

5.2.2 Задача кластеризации

В этой задаче заранее известно разбиение вершин на кластеры (обычно непересекающиеся). Алгоритм почти такой же как и при классификации.

1. Построить вложение $\mathcal{E} = \mathcal{E}(V, E)$;
2. решить задачу кластеризации на числовых признаках $\mathcal{E}$.

Отличие заключается в том, что все пункты алгоритма относятся к обучению без учителя, а метки кластеров используются лишь при подсчете метрики качества.

5.2.3 Задача предсказания ребер

Задача предсказания ребер (*link prediction*) является более универсальным способом оценить качество вложения, поскольку не требует дополнительной целевой переменной. В ней вложение подсчитывается только на части ребер графа, а затем проверяется насколько хорошо по этому вложению можно предсказать пропущенные ребра.

1. Разбить случайным образом множество ребер E на две части E_{train} и E_{test};[4,5]
2. построить вложение $\mathcal{E} = \mathcal{E}(V, E_{train})$;
3. выбрать примеры ребер из множеств E_{train} и E_{test}, а также отрицательные примеры такие что $(u, v) \notin E$;
4. определить переменные

$$y(u,v) = \begin{cases} 1, & \text{если } (u,v) \in V, \\ 0, & \text{если } (u,v) \notin V, \end{cases}$$

$$X(u,v) = (\mathcal{E}_u, \mathcal{E}_v),$$

   где $(\cdot\,,\cdot)$ означает конкатенацию;
5. решить задачу обучения с учителем для данных X и целевой переменной y, подсчитанных на примерах из пункта 3.

5.3 Результаты экспериментов

Мы проводили сравнение работы нашего алгоритма с алгоритмами двух других семейств: алгоритмов на случайных блужданиях и алгоритмов с использованием матричной факторизации. Были выбраны алгоритмы показывающие лучшие результаты внутри своих групп: deepwalk и HOPE соответственно.

5.3.1 Задача классификации

Для данной задачи мы проверяли работу алгоритма на двух графах: BlogCatalog и SBM. Результаты представлены на рисунке 2. Результаты классификации нашего алгоритма на BlogCatalog (рис. 2a) оказались значительно ниже, чем у алгоритма deepwalk. Впрочем, как показывают обзорные статьи [2] и [4], все алгоритмы не относящиеся к семейству алгоритмов на случайных блужданиях дают такие же плохие результаты. На графе SBM наш алгоритм дал результаты сравнимые по качеству с остальными алгоритмами.

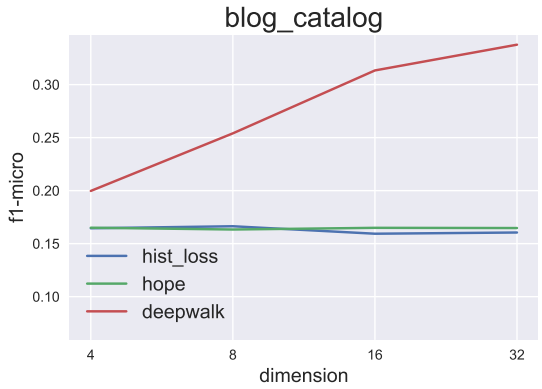
При оценке результатов, использовалась метрика f1-micro (15).

$$F1_{micro} = 2\frac{precision \cdot recall}{precision + recall}, \tag{15}$$

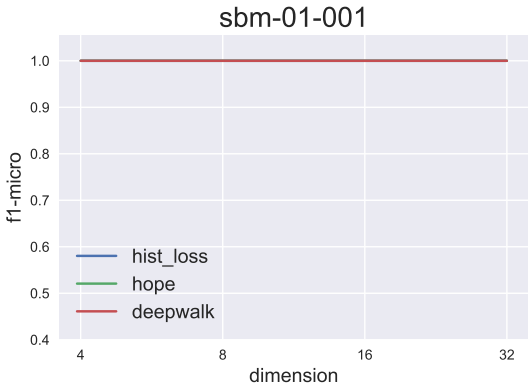
приставка *micro* означает, что *precision* и *recall* вычисляются глобально по всем классам.

[4]Чаще всего накладывается дополнительное требование чтобы $G_{train} = (V, E_{train})$ был связным графом.

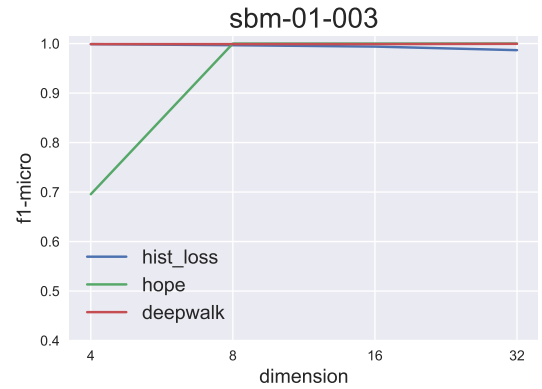
[5]В нашем случае был выбран такой коэффициент разбиения, что $|E_{train}| = |E_{test}| = \frac{|E|}{2}$



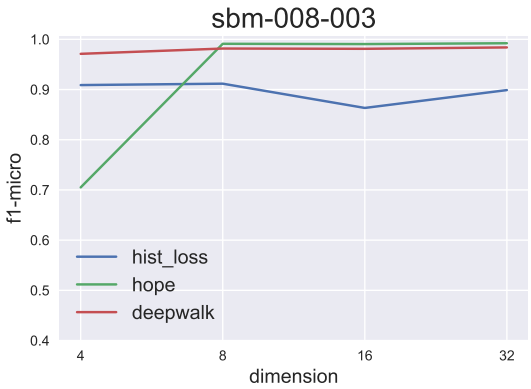
(a) BlogCatalog



(b) SBM $p_{in} = 0.1$, $p_{out} = 0.01$

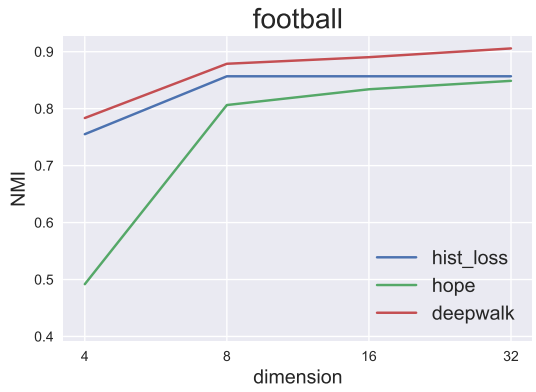


(c) SBM $p_{in} = 0.1$, $p_{out} = 0.03$

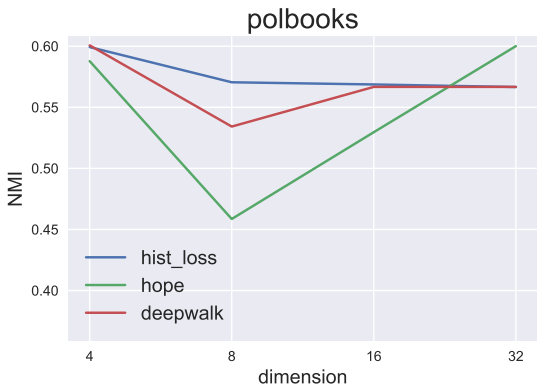


(d) SBM $p_{in} = 0.08$, $p_{out} = 0.03$

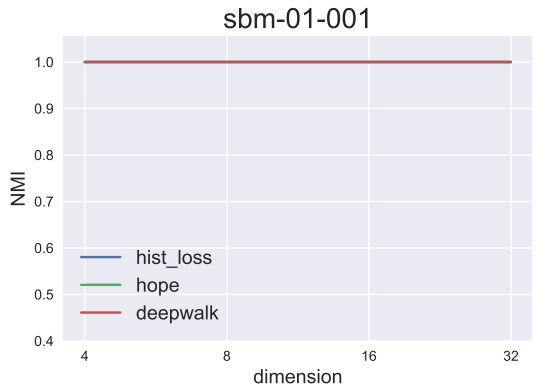
Рис. 2: Результаты классификации



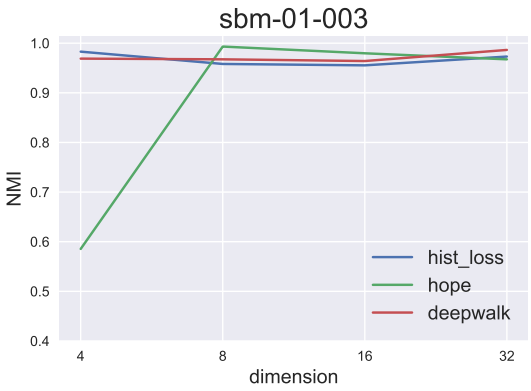
(a) Football



(b) Political Books



(c) SBM $p_{in} = 0.1$, $p_{out} = 0.01$



(d) SBM $p_{in} = 0.1$, $p_{out} = 0.03$

Рис. 3: Результаты кластеризации

5.3.2 Задача кластеризации

Для данной задачи мы проверяли работу алгоритма на графах Football, Political books и SBM, поскольку они имеют метки для кластеризации. Результаты представлены на рисунке 3. Можно заметить, что результаты нашего алгоритма в основном превосходят результаты алгоритма HOPE и приближаются к результатам deepwalk. Это ожидаемый результат, поскольку при оптимизации структурной функции потерь, $\text{dec}(\mathcal{E}_i, \mathcal{E}_j)$ для $(v_i, v_j) \in V$ стремится к 1, что в итоге дает очень плотные кластеры в пространстве вложения.

При подсчете результатов, исопльзовалась метрика NMI (*Normalized Mutual Info*).

$$\text{NMI}(U, V) = \frac{\sum_{i=1}^{|U|} \sum_{i=1}^{|U|} \frac{|U_i \cap V_j|}{N} \log\left(\frac{N|U_i \cap V_j|}{|U_i||V_j|}\right)}{\sqrt{H(U)H(V)}}, \tag{16}$$

где H – энтропия.

5.3.3 Задача предсказания ребер

Результаты представлены на рисунке 4. Видно, что на всех реальных графах наш алгоритм превосходит другие. Это может быть связано со спецификой работы метрики roc-auc, используемой при оценке качества предсказания ребер. Метрика roc-auc оценивает разделимость двух классов, в данном случае положительных и отрицательных пар вершин. В то же время, оптимизация структурной функции потерь пытается

«развести» положительные и отрицательные пары вершин как можно дальше, что и приводит к хорошей разделимости этих классов.

При подсчете результатов, использовалась метрика roc-auc, которая определяет разделимость двух классов. По одному из определений она равна вероятности того, что предсказанное значение для случайного элемента положительного класса окажется больше, чем предсказанное значение для случайного отрицательного элемента.

$$\text{ROC-AUC} = P(y_i^{pred} > y_j^{pred} \,|\, y_i = 1, y_j = 0).$$

6 Выводы

В данной работе был предложен метод вложения графов, который нельзя отнести ни к одному существующему семейству алгоритмов. Результаты экспериментов показали, что качество работы алгоритма сравнимо с качеством существующих методов, и что структурная функция потерь, при некоторой доработке, может считаться хорошим алгоритмом для вложения. Были выделены основные направления для дальнейшей работы: улучшение масштабируемости нашего алгоритма и улучшение работы с небинарными матрицами сходства.

Сейчас время работы алгоритма сильно увеличивается при увеличении размера графа, также построение гистограмм распределений может занимать большое количество оперативной памяти. Кроме того, наш алгоритм не рассчитан на небинарные значения s_{ij}, поскольку он строится на идее, что все ребра можно разделить на два класса, где s_{ij} принимает значение 0 или 1. Были осуществлены попытки бинаризации небинарной S, например считать $s_{ij} > threshold \approx 0.1$ положительными примерами, а остальные отрицательными, но это не дало прироста в качестве работы.

Список литературы

[1] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press, 2001.

[2] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *CoRR*, abs/1705.02801, 2017.

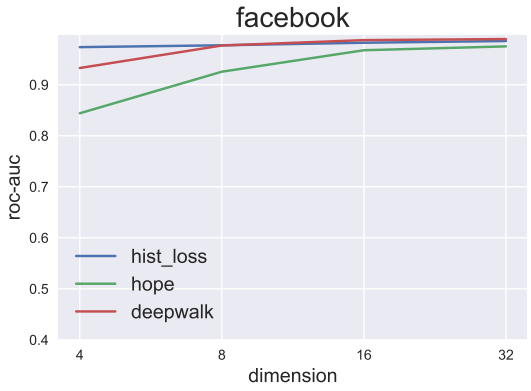
[3] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

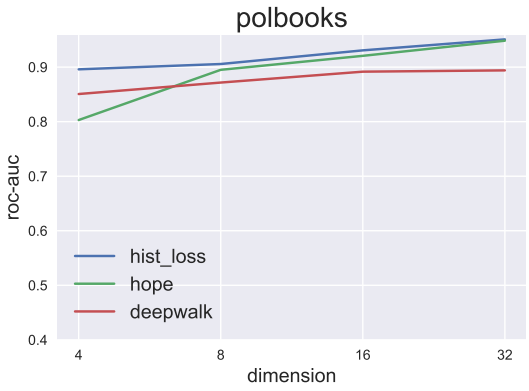
[5] Manuel Martinez, Makarand Tapaswi, and Rainer Stiefelhagen. A closed-form gradient for the 1d earth mover's distance for spectral deep learning on biological data.

[6] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1105–1114, New York, NY, USA, 2016. ACM.

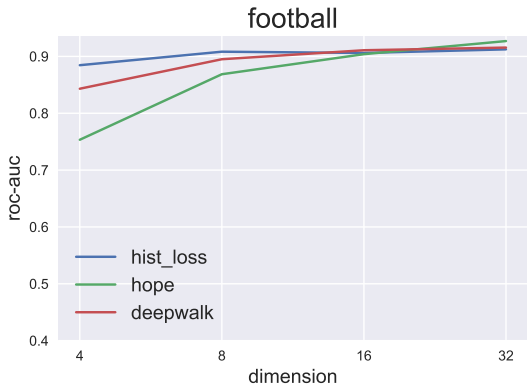
[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.



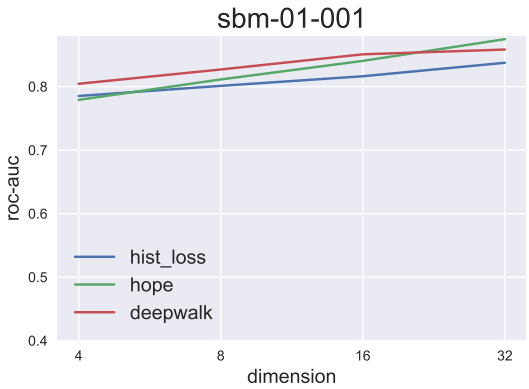
(a) Facebook



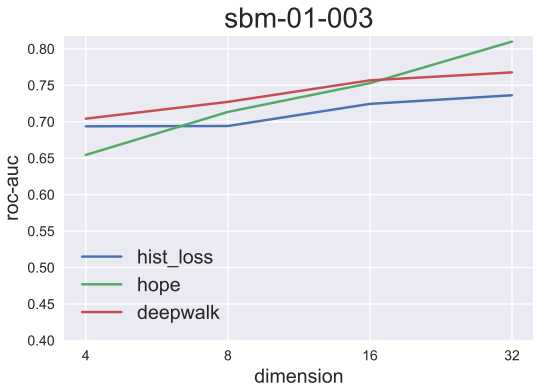
(b) Political Books



(c) Football



(d) SBM $p_{in} = 0.1$, $p_{out} = 0.01$



(e) SBM $p_{in} = 0.1$, $p_{out} = 0.03$

Рис. 4: Результаты предсказания ребер

[8] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In *Advances in Neural Information Processing Systems*, pages 4170–4178, 2016.

[9] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.