Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Шавалиева Ралина Забировна

Автоматическое извлечение гиперонимов из больших текстовых корпусов.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:

Лукашевич Наталья Валентиновна

Москва, 2018

Содержание

Введение

В наше время компьютерная обработка естественного языка является одной из самых востребованных областей искусственного интеллекта. Для того, чтобы правильно распознать смысл фразы, написанной человеком, необходимо иметь дополнительные знания о каждом слове этого предложения и существующих связей между ними. К такой вспомогательной информации относится отношение *is-a*, что означает отношение обобщения.

Способность к обобщению лежит в основе человеческого познания. Люди без труда могут заменить частное на общее. Например, слово «кошка» на слово «животное», «автомобиль» на «транспорт», «красный» на «цвет». Для каждой пары общее слово имеет термин *гипероним*, а частное - *гипоним*. Для каждого гипонима может существовать несколько гиперонимов, и наоборот.

Рис. 1: Пример связи гипонимов с гиперонимами.

Способность успешного распознавания таких лингвистических отношений приносит вклад в такие области, как вопросно-ответная система, системы семантического поиска, управление записями (система хранения и отслеживания документов), а также информационный поиск и навигация по сайтам.

Например, наличие информации, что слова «гепард» и «животное» связаны отношением *is-a*, может помочь при ответе на вопрос «какое самое быстрое животное?». А система информационного поиска сможет подобрать соответствующие сайты, при получении такого же запроса.

В добавок, отношение гипоним-гипероним — это основа почти любой семантической сети и таксономии (иерархической системы отношений сущностей определенной области знаний). Наличие таких построенных структур, помогает при реферировании текста, т.е. извлечении из него основного содержания или заданной информации с целью письменного изложения.

Таким образом, проблема определения отношений обобщения весьма актуальна как в области лингвистики, так и в области искусственного интеллекта.

На данный момент большинство ресурсов, позволяющих определять отношение обобщения, написаны вручную, что очень дорого в плане их создания и поддержки в актуальном состоянии. Ручные словари зачастую имеют недостаточно большую область покрытия. К тому же, в большинстве случаев нет возможности выставления вероятности наличия связи непрерывной случайной величиной от 0 до 1, а лишь только дискретная степень допустимости существования отношения. Например, 0 - нет связи, 1-слабая, 2 - средняя, 3 - сильная. Таким образом, нет возможности определить, какое из слов в качестве гиперонима подходит больше, если есть несколько кандидатов имеющих одинаковую степень. Для современных задач, появляется необходимость создания автоматической системы поиска связи гипоним-гипероним.

Существует два основных вида постановки задачи. Первая, наиболее распространенная, это сопоставление каждой паре слов 1 или 0, в зави-

симости от того, есть связь или нет. Такого рода задача бинарной классификации неоднократно подвергалась критике за её чрезмерную простоту, а также из-за невозможности сравнения кандидатов, как и в случае ручных словарей. Второй способ заключается в поиске гиперонимов, т. е. предоставлении ранжированного списка слов, которые наиболее вероятно являются гиперонимами заранее заданному гипониму. Поиск таких слов происходит из конкретного словаря, приближенному к словарю всех слов и устойчивых выражений для определенного языка.

В данной работе исследуется второй вид постановки задачи - поиск гиперонимов. Рассматриваются различные алгоритмы, как с применением машинного обучения, так и основанные на текстовых шаблонах. И производится сравнение полученных результатов по различиям метрикам качества.

1 Постановка задачи

Целью данной работы является реализация различных алгоритмов извлечения гиперонимов из текстовых корпусов. Исследовать алгоритм, основанный на анализе текстов средством рукописных регулярных выражений, а также класс алгоритмов, использующих разные виды нейронных сетей, с последующим обучением на размеченных данных.

Алгоритм должен для заданного гипонима составлять упорядоченный по вероятности список гиперонимов.

ЗАДАЧИ

1. Исследование существующих подходов к решению поставленной задачи
2. Выбор набора данных. Разделение его на обучающую и тестовую части.
3. Выбор метрик оценки качества модели. Метрики должны учитывать порядок выбранных гиперонимов, так как основная задача – ранжирование списка.
4. Построение и тестирование модели, основанной на рукописных шаблонах
5. Исследование первой модели векторного представления слов, основанной на гипотезе дистрибутивной семантики. *PPMI + SVM.*
6. Построение моделей, использующих различные комбинации векторов, полученных алгоритмом Word2Vec. Дообучение алгоритмом *GBR* и LambdaRank.

7. Реализация нейронной сети, разделяющей каждое слово на 2 вектора: слово в качестве гипонима и слово в качестве гиперонима. Распараллеливание алгоритма средствами Hadoop MapReduce.

8. Сравнение результатов, полученных всеми построенными моделями. Определение лучшей модели.

9. Тестирование выбранной модели на полных данных.

2 Обзор существующих алгоритмов

2.1 Модель, основанная на лексико-синтаксических шаблонах

Данный метод был разработан профессором Марти Херст в 1992 году [?]. Алгоритм является одним из первых в области автоматического определения отношения обобщения между словами. Считается, что пара слов в предложении связана отношением гипоним-гипероним, если она удовлетворяет одному из шаблонов, например, таких, как:

- $[A]$ for example $[B]$ - например
- $[A]$ such as $[B]$ - такие как
- $[A]$ include $[B]$ - включая
- $[A]$ especially $[B]$ - особенно

Здесь $[A]$ обозначает *гипероним*, а $[B]$ - список *гипонимов*. Например, «I like flowers, such as roses or peonies» - «мне нравятся цветы, такие как розы или пионы». В данном случае в качестве гиперонима выступает слово «цветы», а гипонимы - «розы» и «пионы».

Список таких словосочетаний не имеет фиксированного размера. Можно добавлять свои примеры, подходящие конкретной области или исключать неподходящие.

ПРЕИМУЩЕСТВА

- Высокая точность

НЕДОСТАТКИ

- Главный недостаток такого подхода - низкая полнота определения связей. Необходим очень большой текстовый корпус, чтобы выделить хотя бы базовые пары отношений.

- Не каждый пример связи можно описать шаблоном.

- Не улавливаются цепочки связей. То есть, если определены пары «ласточка - птица» и «птица - животное», то может не быть пары «ласточка - животное».

- Несмотря на высокую точность, встречаются случаи, когда пара слов неверно отмечена, как имеющая связь is-a. Один из таких примеров пара предложений:

  «...**cities** in Asian countries such as **Tokyo** ...» - «... города в странах Азии, такие как Токио ...»

  «...cities in Asian **countries** such as **Japan** ...» - «... города в странах Азии, таких как Япония ...»

  В первом случае пара «countries - Tokyo» была бы отмечена неправильно

- Для ранжирования гиперонимов недостаточно иметь только число, означающее сколько раз встретилась конкретная пара, так как это зависит от конкретного текстового корпуса.

2.2 Векторное представление слов

2.2.1 Представление слов векторами, полученными при SVD разложении матрицы PPMI

Данный подход основывается на предположении, которое называется «дистрибутивная гипотеза»: лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения. Для такого подхода необходимо иметь достаточно большой текстовый корпус.

Существует несколько способов получения контекстов для слова. Наиболее популярный называется «window-based» метод. Задается величина ширины окна k. Далее просматриваются все предложения, содержащие конкретное слово. К примеру, интересующее нас слово W_i находится в позиции i в рассматриваемом предложении. Контекстом данного предложения к W_i будет набор слов $(W_{i-k}, \dots W_{i-1}, W_{i+1}, \dots W_{i+k})$ т.е k слов, стоящих до W_i и k слов после. Величина окна произвольная, но чаще всего выбирается от 2 до 5. Таким образом, для каждого слова строится набор его контекстов.

Исследовав схожесть наборов контекстов двух разных слов, можно судить об отношении этих слов между собой. Например, в случае синонимов, наборы контекстов будут близки. Задание же функции вычисления схожести является главным параметром таких моделей.

PPMI (positive pointwise mutual information) - положительная поточечная взаимная информация. Функция *PPMI* является одной из оценок схожести двух лингвистических единиц, например слов, контекстов, абзацев и т.п., на основе заданного текстового корпуса. Для случая вычисления взаимной информации между словом и контекстом данная функция задается следующей формулой:

$PPMI(w, c) = max(PMI(w, c), 0)$

$PMI(w, c) = \log \frac{p(w,c)}{p(w) \times p(c)}$, где

- w - слово из текстового корпуса;
- c - контекст;
- $p(w)$ - вероятность встречи слова w в корпусе = частота появления слова в корпусе, деленная на общее число слов
- $p(c)$ - вероятность встречи данного контекста c = частота появления контекста в корпусе, деленная на общее число контекстов
- $p(w, c)$ - вероятность встречи пары «слово - контекст» = частота появления данной пары в корпусе, деленная на общее число пар

Таким образом, *PPMI* вычисляется напрямую из текстового корпуса. Если слово и контекст не связаны между собой, то

$p(w, c) = p(w) \times p(c)$

и *PPMI* для этой пары будет равно 0. Чем выше данный показатель, тем чаще слово w появляется в сопровождении контекста c.

Строится таблица M, строки которой соответствуют словам, а столбцы – контекстам. Таблица заполняется соответствующими значениями *PPMI*. Вектор слова определяется величинами, расположенными в его строке. Длина вектора - количество найденных в корпусе контекстов. Таким образом, схожесть наборов контекстов для двух слов определяется близостью их векторов [**?**].

Как можно заметить, величина встречаемости слова намного меньше размера множества всех контекстов. Значит, матрица M будет иметь большой размер, и при этом она будет сильно разреженной. Возникает проблема

«проклятия размерности». Измерение расстояния между такими векторами будет неинформативным, т.к согласно Закону Больших Чисел, сумма разности каждого признака n слагаемых стремится к некоторому фиксированному пределу при $n \to \infty$, следовательно, все точки выборки становятся почти одинаково далеки друг от друга.

Одним из решений данной проблемы является снижение размерности алгоритмом *SVD* (Singular Value Decomposition). Данный алгоритм может приблизить матрицу M размера $n \times m$, некоторой другой матрицей M_k с заданным рангом k, такой, что её можно разложить в произведение трех других матриц.

$M \approx M_k = U_k \times \Sigma_k \times V_k^T$,

где U_k и V_k - две унитарные матрицы, состоящие из левых и правых сингулярных векторов соответственно

V_k^T - это сопряжённо-транспонированная матрица к V_k;

Σ_k — матрица размера $k \times k$ с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми)

Полученная матрица U_k будет иметь размерность $n \times k$. Строки такой матрицы будут соответствовать строкам исходной матрицы M, только размерность векторов изменится с m на k. Часть информации потеряется, но существенная её часть остается. Аналогично, матрица V_k^T несет информацию о столбцах матрицы M, только имея при этом меньший размер.

Применительно к нашей задачи, разложение матрицы *PPMI* приведет к получению компактного векторного представления слов, сохраняющего информацию о контекстах.

Дальнейшая работа с полученными векторами будет описана в пункте

2.4.

2.2.2 WORD2VEC

Еще одним инструментом, реализующим модель векторного представления слов, является Word2Vec. Этот инструмент был разработан группой исследователей Google в 2013 году, под руководством Томаша Миколова.

Word2Vec реализует две архитектуры - *Continuous Bag-of-Words* (*CBOW*) и *Skip-gram*. Обе архитектуры построены на нейронных сетях и основаны на дистрибутивной гипотезе.

Принцип работы *CBOW* — предсказание слова при данном контексте, а *Skip-gram* наоборот — предсказывает контекст при заданном слове. Независимо от архитектуры, модель принимает в качестве входных параметров текстовый корпус, и формирует векторное представление каждого слова, входящего в корпус и имеющего частотность в заданном диапазоне.

Применяется искусственная нейронная сеть прямого распространения (Feedforward Neural Network [**?**]) с функцией активации иерархический софтмакс (Hierarchical Softmax [**?**]) и/или негативное сэмплирование (Negative Sampling [**?**]). Метрика близости векторов – косинусное расстояние.

Таким образом, данная модель позволяет получить еще одно векторное представление каждого слова.

2.2.3 Dynamic distance-margin model

Две предыдущие модели позволили получить векторные представления слов, основываясь на наборах их контекстов. При этом каждое слово учитывалось только один раз, не было различия: слово выступает в качестве гипонима или в качестве гиперонима.

Отношение is-a не является симметричным. Если пара слов $A-B$ связана отношением гипоним-гипероним, то пара $B-A$ таким отношением уже не связана. Более того, чем ближе вектор A к вектору B, тем больше вероятность, что слова A и B являются синонимами. Необходимо подбирать границы близости: не слишком далекие, чтобы слова имели связь друг с другом, и не слишком близкие, чтобы не получать синонимичные пары. С точки зрения построения моделей, задача поиска максимума или минимума решается проще и качественнее, чем определение таких границ. Поэтому возникает предположение о разделение одного вектора слова на два - вектор гипоним и вектор гипероним [**?**].

Вектор слова w, используемый, когда данное слово будет выступать в качестве гипЕронима, обозначим за $E(w)$. Например, слово *птица* в отношении (*ласточка*, *птица*). Когда в качестве гипОнима - за $O(w)$. Пример: в паре (*птица*, *животное*).

Предполагается, что построенная новая модель должна удовлетворять следующим трем правилам:

1. Если пара слов $u-v$ связана отношением гипоним-гипероним, то вектора $O(u)$ и $E(v)$ находятся близко друг к другу. $O(u) \approx E(v)$.

2. Если пара слов $u-v$ являются ко-гипонимами, т.е. существует слово w, являющееся гиперонимом, как для слова u, так и для слова v, то должно выполняться соотношение $O(u) \approx O(v)$.

3. Аналогично, если пара слов является ко-гиперонимами, то должно быть верно: $E(u) \approx E(v)$.

Таким образом задача сводится к минимизации расстояний $O(u) \approx E(v)$, $O(u) \approx O(v)$ и $E(u) \approx E(v)$.

В качестве обучающего множества берется набор триплетов (u, v, q), где u - гипоним, v - гипероним, а q - сколько раз пара гипоним-гипероним (u, v) встретилась в текстовом корпусе. Такой набор данных может быть собран вручную или, например, методом шаблонов, описанным в пункте 2.1. Необходимо учитывать, что для получения хороших результатов, размер текстового корпуса должен быть очень большим.

Для каждой пары $x = (u, v, q)$ из выбранного набора данных, вычисляется функция расстояния между векторами $O(u)$ и $E(v)$. В качестве функции расстояния может, например, выступать 1-норма разности векторов. Так

$f(x) = ||O(u) - E(v)||_1$

Для того, чтобы оценить величину ошибки модели на примере x, расстояние между векторами слов u и v сравнивается с расстоянием от u до произвольного слова v'. Т.е выбирается негативный пример гиперонима для гипонима u. По предположению, построенные вектора $O(u)$ и $E(v)$ должны быть ближе, чем вектора $O(u)$ и $E(v')$. Насколько ближе они должны быть, определяется величинами q и q'. Величина q' означает, сколько раз пара гипоним-гипероним (u, v') встретилась в текстовом корпусе. Так как слово v выбирается случайным из всего корпуса, то наиболее вероятно q равняется нулю.

Величины q и q' показывают насколько вероятным может быть существование отношения is-a между соответствующими им парами слов. Тогда, чем больше разница этих вероятностей, тем расстояния между величинами $f(x)$ и $f(x')$ должно быть больше. Учитывая проведенный анализ, вводятся следующие соотношения:

- $x' = триплет\ (u, v', q')$

- $f(x) < f(x') - m(q, q')$
- $m(q, q') = log(q + 1) - log(q' + 1) = log\frac{q+1}{q'+1}$

В таком случае, ошибку модели на примере x можно вычислить как

$$max(0, f(x) - f(x') + m(q, q'))$$

И тогда модель будет обучаться уже на парах (x, x'). Аналогично, в качестве негативного примера x' может выступать не только триплет (u, v', q'), но и (u', v, q').

Для того, чтобы обучать модель чаще на парах, которые с большей вероятностью имеют отношение is-a, используется искусственное увеличение обучающего множества пар (x, x') с учетом этой вероятности. Для каждого триплета x подбирается столько негативных примеров x', сколько раз (u, v) встретилась в текстовом корпусе, т.е создается q различных пар (x, x'). Итоговая функция потерь построенной модели вычисляется как:

$$\sum_{x=(u,v,q)} \sum_{j=1}^{q} max(0, f(x) - f(x'_j) + m(x, x'_j))$$

При успешном обучении данной модели, она будет удовлетворять всем трем условиям, описанным выше:

1. $O(u) \approx E(v)$ достигается за счет непосредственной минимизации расстояния $f(x)$
2. Если для пары (u, v) существует слово w, являющееся гиперонимом, как для слова u, так и для слова v, то из первого пункта следует:

$O(u) \approx E(w)$ и $O(v) \approx E(w)$, а значит, и $O(u) \approx O(v)$. Другими словами, оба слова u и v подтягиваются к слову w, тем самым становясь ближе друг к другу.

3. Аналогично достижима цель $E(u) \approx E(v)$

Выполняя все эти 3 условия, становится возможным определение пары (*ласточка, животное*) имея в изначальном наборе данных только пары (*ласточка, птица*) и (*птица, животное*). Устранился один из важных недостатков чисто шаблонных моделей.

Обучение такой модели сводится к обучению нейронной сети, имеющей следующую архитектуру:

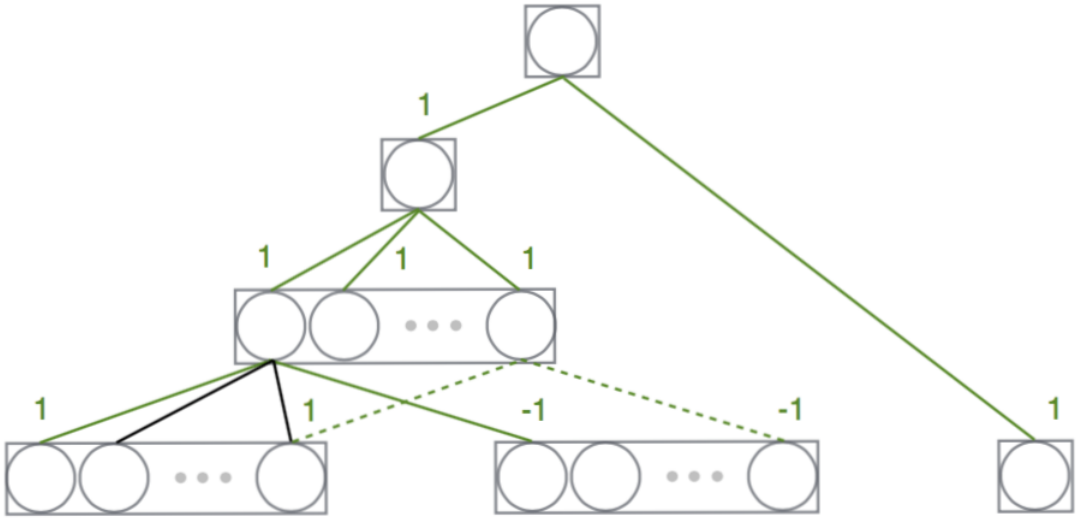

Рис. 2: $p(x)$ - вычисление расстояния между векторами $O(u)$ и $E(v)$ для $x = (u, v, q)$.

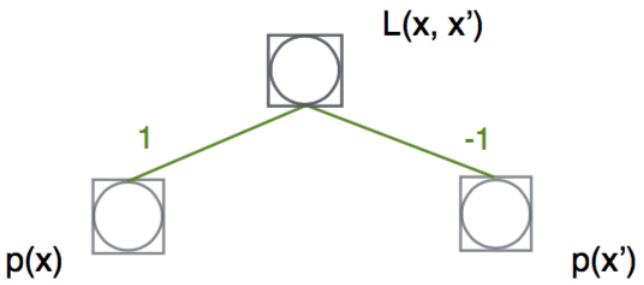

Рис. 3: $L(x, x')$ - вычисление ошибки для триплета x и его негативного примера x'.

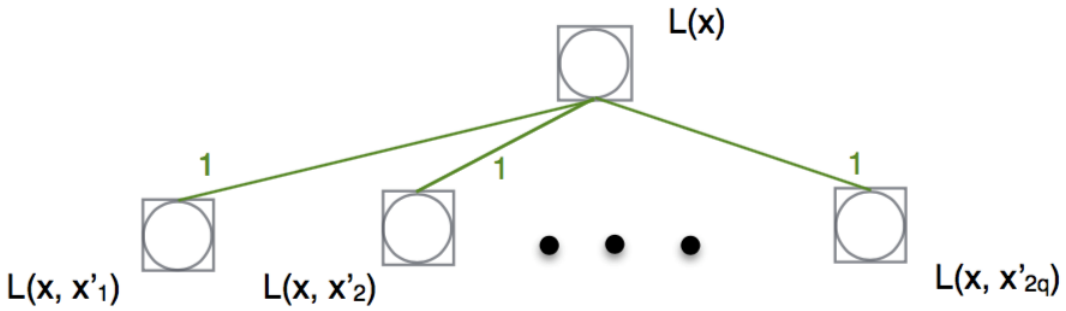

Рис. 4: $L(x)$ - вычисление суммарной ошибки для триплета x.

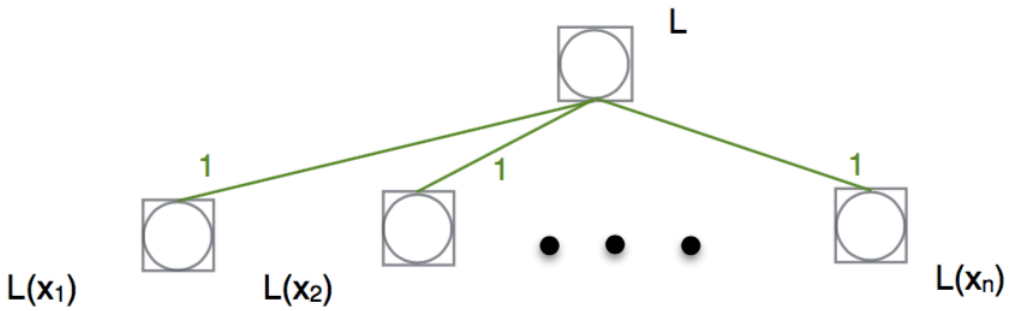

Рис. 5: L - вычисление функции потерь модели.

Модель оптимизируется алгоритмом SGD.

Таким образом, разобраны три различные модели получений векторов слов. Осталось рассмотреть способы вычисления итоговых предсказаний

существования отношений *is*-а между словами, используя их векторное представление.

2.2.4 Обучение модели, предсказывающей существование связи IS-A, на основе векторов слов.

Как было сказано, большинство задач, связанных с темой определения существования связи *is*-а, были сведены к выставлению каждой паре слов либо 0 (связи нет), либо 1 (при её наличии). Поэтому чаще всего для последующего обучения векторов был выбран алгоритм SVM.

Для обучения брался готовый размеченный корпус пар слов, имеющих между собой отношение гипоним-гипероним, и подмешивался к нему набор пар, не имеющих такой связи. Все, что оставалось, это скомбинировать вектора слов из пары, в качестве входного вектора для обучения.

Существует множество различных комбинаций. Наиболее используемые из них следующие:

- Разность векторов $< u - v >$
- Соединение векторов (конкатинация) $< u, v >$
- Вычисление косинусного расстояния $cos(u, v)$
- Евклидово расстояние $||u - v||_2$
- И их комбинации, например:
  - конкатенация с добавлением евклидового расстояния $< u, v, ||u - v||_2 >$
  - разность векторов с добавлением косинусного расстояния $< u - v, cos(u, v) >$

Таким образом для каждой из трех моделей получения векторного представления слов, выбирался какой-то из способов комбинирования векторов и происходило обучение на таких входных данных алгоритма SVM.

Как показали эксперименты, такие виды моделей давали показатели метрик классификации лучше, по сравнению с чисто шаблонными методами.

3 Описание тестовых данных и метрик оценки качества моделей

3.1 Данные

Для обучения и сравнения моделей, необходимо выбрать размеченный набор данных, словарь, из которого будут выбираться подходящие гиперонимы, а также большой текстовый корпус.

Все необходимые для данной задачи данные были предоставлены участникам конкурса SemEval-2018 [ссылка].

1. Размеченный набор данных.

   Для каждого из 1500 гипонимов, составлен эталонный ранжированный список гиперонимов. Длина списка варьируется от 3 до 15 слов.

2. Словарь слов.

   Предоставлен словарь, состоящий из 218755 слов. Только слова, включенные в этот список могут быть взяты в качестве гиперонима

3. Текстовый корпус.

   Для обучения своих моделей предоставлен UMBC корпус, содержащий в себе 3 миллиарда слов. Этот корпус составлен из отрывков веб страниц и является частью Stanford WebBase Project. UMBC содержит в себе информацию многих различных областей.

Примеры данных находятся в приложении А.

3.2 Метрики качества

Так как главной задачей является извлечение и ранжирование гиперонимов, то были выбраны метрики качества, учитывающие порядок слов в списке:

- Mean Reciprocal Rank (MRR) - является показателем средней позиции первого правильно определенного гиперонима

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

где Q - множество гипонимов в тестовой выборке, а $rank_i$ - позиция первого предсказанного гиперонима, который есть в эталонном списке

- Precision@k (P@k) - оценивает долю верно извлеченных гиперонимов в списке слов, усеченном до k-й позициии.

$$P@k = \frac{1}{k} \sum_{i=1}^{k} isTrue(i)$$

где $isTrue(i) = 0$, если гиперонима, стоящего на i-й позиции в предсказанном списке, нет в эталонном списке, и 1, если есть. Для задачи данной работы, оценивается средняя величина $P@k$ ($ap@k$) по всем гипонимам, где $k = 1, 3, 5$ и 15

- Mean Average Precision @15 (MAP) - усредненная оценка $P@k$ по всем k от 1 до 15

$$map@K = \frac{1}{N} \sum_{j=1}^{N} ap@K_j$$

4 Исследование и построение решений задачи

4.1 Шаблонный метод

4.1.1 Извлечение из корпуса UMBC

Для тестирования первого метода решения поставленной задачи, необходимо было составить шаблоны поиска гипонимов и гиперонимов, аналогичных шаблонам Марти Херст.

Были найдены ручные примеры шаблонов как в научных статьях, так и в готовых скриптах, написанных на языке программирования. Наиболее широким списком разнообразных шаблонов обладал класс *hearstPatterns*, написанный на языке Python. Всего в нем содержалось 48 примеров, написанных регулярными выражениями.

Для удобной и более быстрой работы с данным классом, необходимо было выгрузить исходный код и подправить его под текущую задачу. Производилась внутренняя лемматизация слов и приведение их к нижнему регистру, а также удаление пунктуации предложения. В итоге, была получена функция, на вход которой передавалось необработанное предложение, а на выходе составлялся список найденных пар гипоним-гипероним.

Целью проверки данного метода шаблонов было извлечение всех пар, связанных отношением *is*-a, из текстового корпуса UMBC средствами класса *hearstPatterns*, и последующая оценка выбранными метриками.

Как оказалось на практике, написанных шаблонов было недостаточно для качественного обнаружения искомых пар. В качестве примера: из 1,5 млн предложений было обнаружено всего 8003 пары. Более того, на обработку одно предложения всеми регулярными выражениями требовалось очень много времени. Просмотр всего корпуса требовал более 5-ти дней.

Поэтому добавление новых рукописных шаблонов не представлялось возможным.

Результаты:

- MRR: 0.005
- MAP: 0.0017
- P@1: 0.0033
- P@3: 0.0023
- P@5: 0.0018
- P@15: 0.0012

Не имея в наличии мощной техники, достаточного времени и богатого списка шаблонов, протестировать данной метод в полном объеме не удалось.

4.1.2 PROBASE

Алгоритм, основанный на шаблонах, имеет высокую точность и не требует обучения. Поэтому некоторые компании, обладающие большими ресурсами, составляют свои наборы таких шаблонов и обрабатывают с их помощью огромное количество текстовых корпусов. Одним из таких полученных наборов слов является ProBase компании Microsoft.

Probase представляет собой набор триплетов вида (u, v, q), где u — гипоним, v — гипероним, q — количество раз, сколько пара $u - v$ встречалась в текстовых корпусах. Microsoft обработал более 1.5 миллиарда веб страниц, содержащих текстовую информацию различных областей, из которых удалось извлечь 33 миллиона различных триплетов.

На основе этих данных была построена модель, сопоставляющая каждому из 1500 гипонимов список гиперонимов, упорядоченных по уменьшению значения q, для соответствующей пары гипоним-гипероним. Например, если для слова *море* в ProBase существуют триплеты: (*море, отдых, 80*), (*море, водоем, 100*) и (*море, путешествие, 50*), и других нет, то для этого слова составится список гиперонимов: [*водоем, отдых, путешествие*].

Протестировав данный алгоритм, были получены следующие результаты:

- MRR: 0.02649
- MAP: 0.01239
- P@1: 0.01343
- P@3: 0.01119
- P@5: 0.01166
- P@15: 0.01305

В связи с просмотром большой базой текстовых корпусов, результаты этой модели ProBase оказались лучше, но все равно остаются совсем невысокими.

4.2 PPMI + SVD

Для исследования первого метода, основанного на дистрибутивной гипотезе, каждое предложение текстового корпуса было разбито на контексты с шириной окна в 5 слов. Составлена разреженная матрица частоты

встречаемости пары (слово, контекст). Затем полученная матрица была пересчитана в *PPMI* матрицу по формулам, описанным в пункте 2.1.

Размер матрицы оказался слишком большим, чтобы было возможно применить алгоритм снижения размерности *SVD*. Были опробованы готовые реализации модели на языках Python (классы numpy.linalg.svd и scipy.sparse.linalg.svds) и Matlab (svd), но ни одна из них не смогла преобразовать *PPMI* матрицу.

Таким образом, метод *PPMI* + *SVD*, получения векторного представления слов, протестировать на существующий данных не удалось.

4.3 WORD2VEC

4.3.1 Модель GoogleNews

Следующим исследуемым методом получения векторного представления слов был Word2Vec.

Существует множество готовых обученных моделей, опубликованных в открытом доступе. Для тестирования решения данной задачи была применена модель, имеющая архитектуру CBOW и обученная на корпусе GoogleNews, размером в 3 миллиона слов. Основные параметры: размерность вектора – 300, ширина окна – 5.

Алгоритм Word2Vec основывается на информации о контекстах, в которых употреблялось слово в текстовом корпусе. Поэтому получение вектора возможно только для тех слов, которые встречались в корпусе хотя бы 1 раз (в общем случае, не менее N раз, где порог N является гиперпараметром). Значит, если гипоним, для которого необходимо найти все гиперонимы, не встречался, в обучающем корпусе, то построенная модель не сможет подобрать для него требуемый список.

Из 1500 гипонимов, находящихся в выбранном корпусе данных, построенная модель, смогла предоставить вектора только для 77%. Для оставшихся слов, необходимо было применять другую модель.

Для получения итогового списка гиперонимов к каждому гипониму, было применено 2 различных алгоритма: *GBR* (Gradient Boosting for regression) и *LambdaRank*.

Алгоритм *GBR* можно применить как pointwise алгоритм ранжирования. В то время как LambdaRank является представителем pairwise подхода. Нельзя заранее сказать, какой из двух подходов будет работать лучше, поэтому для тестирования были использованы оба.

Подготовка обучающего множества

Так как необходимо было исследовать множество алгоритмов с различными параметрами, для экономии скорости и минимальной потери точности, извлечение гиперонимов происходило не из полного словаря, содержащего ≈ 220 тыс слов. Для каждого гипонима составлялся свой словарь по следующему алгоритму:

1. Добавлялись все слова из эталонного списка гиперонимов. Среднее количество гиперонимов для каждого гипонима составляло 5 слов

2. Каждому слову из списка присваивалось значение, отражающее его позицию. Эти величины служат целевым признаком для предсказания. Для алгоритма LambdaRank чаще всего используются значения 0,1,2 и 3, поэтому для эталонных гиперонимов были выбраны значения 1,2 и 3. Список делился на три части, если слово находилось в первой из них, ему присваивалась величина 3, если во второй, то 2, оставшейся последней части - 1.

3. Далее в случайном порядке добавлялись к полученному списку дополнительно 500 слов, играющих роль негативных примеров. Каждое такое слово имело целевое значение 0.

В качестве признаков для обучения моделей были протестированы следующие комбинации векторов гипонима и гиперонима:

1. Разность векторов:
   $Diff :< u - v >$

2. Конкатенация векторов + евклидово расстояние степени 1:
   $||u - v||_1 = \sum_{i=1}^{d} |u_i - v_i|$
   $Dist1 :< u, v, ||u - v||_1 >$

3. Конкатенация векторов + евклидово расстояние степени 2:
   $||u - v||_2 = \sqrt{\sum_{i=1}^{d} |(u_i - v_i)^2|}$
   $Dist2 :< u, v, ||u - v||_2 >$

4. Конкатенация векторов + косинусное расстояние между ними:
   $Cos :< u, v, cos(u, v) >$

Полученный новый набор данных разделялся на обучающую и тестовую выборку в отношении 2 : 1. Так как не для всех гипонимов построены вектора, то тестирование и обучение происходило только на 77% данных.

Тестирование

Получены следующие результаты:

Лучший результат среди всех опробованных моделей с векторами *Word2Vec* - GoogleNews, показал алгоритм *LambdaRank* с основными параметрами: шаг обучения = 0.1, кол-во деревьев 100, доля выборки на каждом шаге

	MRR	**MAP**
Diff	0.914	0.504
Dist1	0.942	0.487
Dist2	0.914	0.787
Cos	0.039	0.044

Таблица 1: LambdaRank

	MRR	**MAP**
Diff	0.893	0.442
Dist1	0.912	0.463
Dist2	0.918	0.659
Cos	0.078	0.062

Таблица 2: GBR

обучения (subsample) = 0.8, максимальная глубина 5. Для метрики *MRR*, более успешными было представление векторов комбинацией Dist1, а для *MAP* - Dist2.

4.3.2 Обучение собственное модели WORD2VEC

Далее, был обучен алгоритм *Word2Vec* на текстовом корпусе UBMC. Целью такого исследования было увеличение доли гипонимов, для которых возможно построить вектор, и возможность настроить главные гиперпараметры.

Из опробованных комбинаций параметров, лучший результат показала модель Skip-gramm, с шириной окна 7 и размером вектора 300.

Так как среди гипонимов встречались не только слова, но и словосочетания из 2-3 слов, не удалось построить вектора для них всех. Доля с 77% увеличилась до 82%.

Тестирование

Были применены все те подходы, которые использовались для эксперимента с моделью *Word2Vec* - GoogleNews.

Лучшей также оказалась модель *LambdaRank* с теми же параметрами.

Получены следующие результаты:

	MRR	MAP
Diff	0.945	0.489
Dist1	0.931	0.705
Dist2	0.924	0.793
Cos	0.035	0.020

Таблица 3: LambdaRank

	MRR	MAP
Diff	0.914	0.459
Dist1	0.921	0.588
Dist2	0.927	0.655
Cos	0.081	0.073

Таблица 4: GBR

4.4 DYNAMIC DISTANCE-MARGIN MODEL

Для применения данного алгоритма, необходимо иметь набор триплетов (u, v, q), где u - гипоним, v - гипероним, а q - сколько раз пара гипоним-гипероним (u, v) встретилась в текстовом корпусе. В качестве такого набора данных был взят ProBase, описанный в главе шаблонных методов.

ProBase был получен на основе очень больших наборов текстовых корпусов, поэтому значения q могли достигать величины в 35000. Алгоритм *DDM* (Dynamic distance-margin), учитывает и обучает каждую пару столько раз, сколько она встречалась. Таким образом, некоторые пары могли учитываться 35 тысяч раз за одну эпоху, в то время, как большинство других имели значение $q < 20$. В таком случае модель могла практически не обучить большинство векторов. Чтобы устранить такой большой разрыв, значение q было изменено по формуле $\sqrt[1.85]{q}$. Степень корня подбиралась так, чтобы максимальное значение не было слишком большим или слишком маленьким.

Для каждого триплета x подбирался негативный триплет x', где был заменен либо гипероним, либо гипоним на случайный. Чтобы детерминировать данный выбор и не вносить различия в обучение, было решено для каждой пары подбирать сразу 2 негативных триплета - негативный гипо-

ним и негативный гипероним.

Нейронная сеть, используемая в данной модели, обучает входные вектора. Поэтому, было решено вручную рассчитать градиенты для изменения этих векторов (метод обратного распространения ошибки).

После расчетов, получился следующий алгоритм изменения векторов на каждой эпохе:

for $x = (u, v, q)$ in X **do** ▷ для каждой пары из ProBase
  for i in $[1..q]$ **do** ▷ для каждой пары негативных примеров
    $x'_i = (\mathrm{u},\ v'_i,\ q'_i)$ ▷ негативный пример гиперонима
    if $f(x) + log(q) < f(x'_i) + log(q'_i)$ **then**
      $u+ = (u - v)\ /\ ||u - v||_2$
      $v- = (u - v)\ /\ ||u - v||_2$
      $u- = (u - v')\ /\ ||u - v'||_2$
      $v'+ = (u - v')\ /\ ||u - v'||_2$
    end if

    $x'_i = (u'_i,\ \mathrm{v},\ q'_i)$ ▷ негативный пример гипонима
    if $f(x) + log(q) < f(x'_i) + log(q'_i)$ **then**
      $u+ = (u - v)\ /\ ||u - v||_2$
      $v- = (u - v)\ /\ ||u - v||_2$
      $u'- = (u' - v)\ /\ ||u' - v||_2$
      $v+ = (u' - v)\ /\ ||u' - v||_2$
    end if
  end for
end for

Для одной эпохи необходимо было рассчитать расстояния и градиент

для более 170 миллионов пар слов.

Был реализован алгоритм на языке Python, но как и следовало ожидать, для его обучения не было достаточно памяти и разумного времени.

После исследования данных ProBase, было установлено, что не со всеми словами из этого набора, связаны необходимые для нашей задачи гипонимы. Была выделена отдельная компонента связности, составляющая 1 / 15 часть всего набора пар ProBase. То есть обучение 14 / 15 всех пар векторов никак не влияло на улучшение результата поставленной задачи.

После уменьшение данных в 15 раз была проведена еще одна попытка запуска алгоритма, написанного на языке Python. Уменьшение данных все равно не позволило обучить алгоритм, что привело к поиску других решений реализации данной модели.

4.4.1 Распараллеливание алгоритма средствами HADOOP

Одним из используемых современных решений работы с большим объемом данных является парадигма параллельных вычислений Hadoop MapReduce. Хорошо иллюстрирует данный подход схема, расположенная ниже.

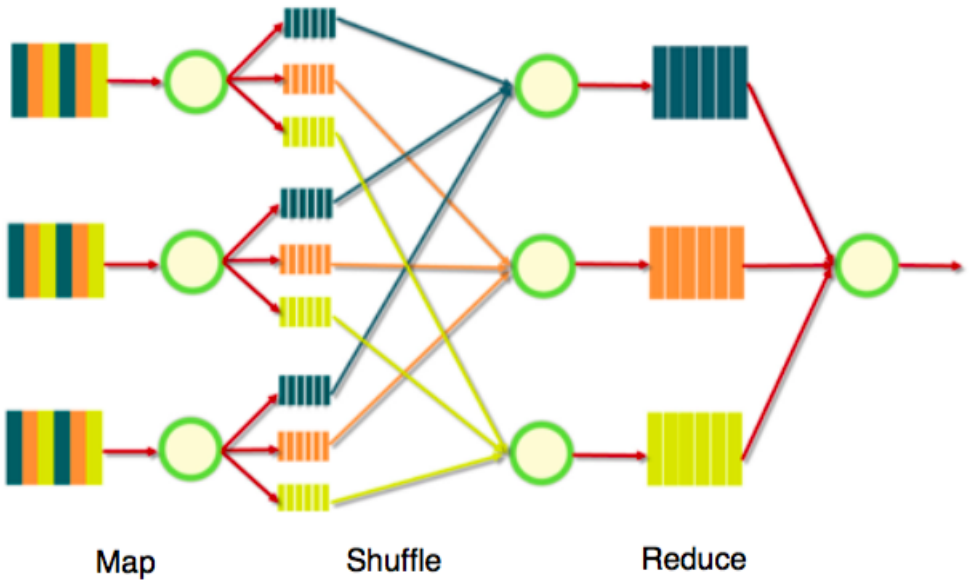

Рис. 6: Схема распараллеливание.

На стадии Map параллельно считываются данные из независимых блоков, на которые разбит исходный входной файл. Происходит их преобразование в пары (Ключ, Значение). Затем идет стадия Shuffle: вывод функции Map распределяется по «корзинам», в зависимости от значения ключа. Все пары (Ключ, Значение), имеющие одинаковый ключ, попадают в одну корзину и отдаются определенному редьюсеру. Таким образом, данные из разных частей входного файла могут собраться вместе при обработке на стадии Reduce. Такая архитектура позволяет быстрое распараллеливание и не загружает оперативную память, что необходимо при решении поставленной задачи.

Краткая реализация алгоритма *DDM* в парадигме *MapReduce*:

1. Считываются значения векторов для каждого слова. Для первой эпохи значения заполняются случайными величинами в диапазоне $[-0.1, 0.1]$.

2. На стадии Map происходит параллельное считывание пар (u, v, q).

3. Формируется пара (Ключ, Значение) для её обработки на стадии Reduce. Ключ $= u$, Значение $= v$. Отправляются на Reduce.

4. Для каждой такой пары генерируется q негативных примеров гипонимов u' и q негативных примеров гиперонимов v'.

5. Для каждой пары (u, v') (аналогично для (u', v)) рассчитывается разница расстояния между u и v' и расстояния между u, v. Если $f(x) + \log(q) < f(x') + \log(q')$, то необходимо изменить вектора u, v, v' - переход к пункту 4. Иначе, просматривается следующая пара.

6. Рассчитываются антиградиенты du, dv, dv' и умножаются на шаг обучения.

7. Формируются пары (Ключ, Значение) для их обработки на стадии Reduce. Ключ соответствует самому вектору, а значение - изменению. Получаются пары (u, du), (v, dv), (v', dv'). Отправляются на Reduce.

8. На стадии Reduce для каждого вектора u (аналогично v), собирается список всех его изменений, полученных из пункта 6 и изначальное значение вектора из пункта 1.

9. Высчитывается новое значение вектора и сохраняется в файле, для его считывания следующей эпохой.

4.4.2 Тестирование

При распараллеливании данного алгоритма на 24 кластера, получилось, что средняя продолжительность вычисления одной эпохи составляет 50 секунд.

Параметры сети были установлены следующими: длина вектора 100, шаг обучения 0.1, количество эпох 100.

Были получены вектора для 91.5% гипонимов, что показало наибольшее покрытие среди всех реализованных в данной работе методов.

Также, как и при исследовании метода *Word2Vec* были протестированы алгоритмы *GBR* и *LambdaRank* с видами представления векторов: $Diff$, $Dist1, Dist2, Cos$.

Полученные результаты:

	MRR	**MAP**
Diff	0.922	0.739
Dist1	0.943	0.847
Dist2	0.954	0.874
Cos	0.101	0.083

Таблица 5: LambdaRank

	MRR	**MAP**
Diff	0.922	0.647
Dist1	0.937	0.758
Dist2	0.915	0.744
Cos	0.078	0.015

Таблица 6: GBR

Данная модель показала самый лучший результат среди всех реализованных в данной работе.

5 Тестирование на полных данных

5.1 Постановка задачи

После реализации всех моделей и сравнения полученных результатов, был выбран алгоритм DDM + LambdaRank для тестирования на полных данных. Ранее, для сравнения моделей строился частичный набор данных, состоящий из правильно подобранных гиперонимов и произвольных 500 слов (отрицательные примеры) для каждого гипонима. Полные данные подразумевают ранжирование всего словаря (более 200 тысяч слов) для каждого отдельно взятого гипонима. Такой подход позволяет в точности оценить качество модели, но требует значительного времени.

Первая попытка показала низкие результаты:

- MRR: 0.09649
- MAP: 0.08239
- P@1: 0.09343
- P@3: 0.08119
- P@5: 0.08166
- P@15: 0.08305

Связано это с тем, что модель LambdaRank обучалась только на явных положительных и отрицательных примерах. Такая модель смогла выучить только тривиальные закономерности, аналогичные тому, сколько раз встретилась пара слов в ProBase. Тем не менее результат в несколько раз превосходит использования шаблонных методов, построенных на том же словаре ProBase.

Чтобы лучше отделять пограничные примеры (похожие на положительные, но тем не менее являющиеся отрицательными), необходимо при составлении набора данных для алгоритма LambdaRank добавить такие случаи. Проблема состоит в том, как найти такие примеры. Составление ручной разметки требует пересмотра всего большого словаря для каждого слова. Кроме того, необходимо создать критерий, определяющий, является ли слово пограничным. Для каждого алгоритма, каждого его состояния в процессе обучения, такой критерий может быть своим: схожесть слова с правильными гиперонимами зависит от текущих скрытых параметров модели.

5.2 Реализация алгоритма подбора отрицательных примеров

С похожей проблемой сталкивались разработчики компании Яндекс при обучении нейронной сети DSSM, составляющей ранжированный список сайтов по запросу пользователя. Необходимо было правильно составить список "неправильных"сайтов для каждого запроса. После ряда различных экспериментов они предложили алгоритм отбора таких примеров на каждой эпохе обучения.

Алгоритм основывался на идеи обучать модель на собственных ошибках. На первой эпохе в качестве отрицательных примеров выбирались N случайных сайтов. На таких данных обучалась модель, а затем предсказывала сайты для тех же запросов. Далее выбирались N ошибочных сайтов, которые оказались выше всех в списке. Они и выбирались в качестве отрицательных примеров для следующей эпохи обучения модели. Такой алгоритм склонен к переобучению, поэтому необходимо проверять результаты каждой эпохи на валидационной выборке.

Эта идея была реализована для задачи подбора отрицательных приме-

ров гиперонимов. Для каждого гипонима на каждой эпохе отбирались 500 слов, схожих с эталонными гиперонимами, но являющихся ошибочными.

5.3 Результаты

Алгоритм, описанный выше, позволяет избавиться от ручной разметки, но тем не менее остается необходимость на каждой эпохе для каждого гипонима предсказывать вероятность каждого слова из словаря и переобучать модель. Из-за такого большого количества операций, время обучения одной эпохи не позволяло проводить множество экспериментов.

Было проведено 7 эпох. Лучший результат показала модель на пятой эпохе:

- MRR: 0.09649
- MAP: 0.08239
- P@1: 0.09343
- P@3: 0.08119
- P@5: 0.08166
- P@15: 0.08305

Видно, что результаты значительно улучшились.

При просмотре гиперонимов, которых извлекала модель, оказалось, что большинство слов и словосочетаний, находящихся в начале списка, действительно являются правильными гиперонимами. Но при сравнении с эталонным списком, многие из них отмечались ошибочными. Связано это с тем, что эталонный список составлялся по текстовому корпусу UMBC. И

найденная моделью DDM пара гипоним-гипероним не встречалась в предложениях этого корпуса, как слова, связанные отношением is-a. Например, пара: *tropical storm - natural disturbance* отмечена ошибочной.

Заключение

В данной работе были исследованы и реализованы различные методы извлечения гиперонимов из текстовых корпусов. Рассмотрен алгоритм, основанный на анализе текстов средством рукописных регулярных выражений. Протестированы шаблоны из библиотеки *hearstPatterns (Python)* на текстовом корпусе UMBC и заранее извлеченные пары ProBase. Также реализованы и проанализированы алгоритмы получения векторного представления слов: Word2Vec и DDM, с последующим их дообучением алгоритмами GBR и LambdaRank. Лучшая модель протестирована на полном наборе данных с итеративным дообучением посредством выбора определенных отрицательных примеров (NegativeExample).

Наиболее удачной оказалась связка ProBase + DDM + LambdaRank + NegativeExample. ProBase обеспечивает высокой точностью, DDM увеличивает полноту, LambdaRank и NegativeExample подстраивают полученные вектора под конкретные эталонные данные.

Полученная модель извлекает достаточно правильные гиперонимы, относительно действительных значений слов. Увеличения точности результатов на конкретных эталонных данных, позволит составление качественных шаблонов, аналогичных ProBase, на текстовом корпусе UMBC.

Приложение А

ДАННЫЕ

5.3.1 Гипонимы

- *pollution*
- *song*
- *software program*
- *tropical storm*
- ...

Всего 1500

5.3.2 Словарь

- *abas*
- *abased*
- *abasement*
- *abash*
- ...

Всего 218755

5.3.3 Эталон

- *pollution* -
  dirtiness,
  dirtying,
  environmental condition,
  impurity,
  sanitary condition,
  uncleanness

- *song* -
  sound,
  work of art,
  musical composition,
  possession

- ...

Всего 1500