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

Федор Сергеевич Симонов

Разрешение лексической многозначности с помощью векторного представления слов

Курсовая работа

Научный руководитель
к.ф.-м.н. Н. В. Арефьев

Москва, 2016

В данной работе рассматриваются реализация алгоритмов разрешения лексической многозначности (WSD – Word Sense Disambiguation) на основе модели векторного представления слов (Word2vec).

Содержание

1 Введение

Системы, решающие задачи обрабоки текстов на естественном языке (NLP – Natural Language Processing), встречаются с множеством сложностей. Одна из таких проблем – это правильное понимание омонимов, т.е. слов, имеющих более одного значения, при этом, имеющих, например, одинаковое написание (или произношение). В этом и заключается задача разрешения лексической многозначности (WSD). Результатами WSD можно воспользоваться во многих областях компьютерной лингвистики, таких как информационный поиск, извлечение информации и машинный перевод. Было исследовано большое количество методов решения этой задачи: лингвистических (основанных на знаниях и правилах), методов обучения с учителем, методов обучения без учителя (кластеризующего слова на основе их значения) и также разных комбинированных методов. В данной работе изучаются методы, основанные на векторном представлении слов (Word embedding) и результатах разных алгоритмов решения задачи дискриминации значений слов (WSI).

Векторное представление слов – общее название для моделей семантики слов естественного языка, ставящих в соответствие словам n-мерные вектора вещественных чисел. В работе используется модель Word2Vec [1, 2].

2 Постановка задачи

Целью данной работы является реализация алгоритма на основе word2vec'a, который распознает, в каком из значений используется неоднозначное слово, подбор гиперпараметров этого алгоритма, а также, оценка результатов и сравнение их между собой.

3 Исследование предметной области

Представление слов в виде векторов способно значительно улучшить качество работы обучающихся методов обработки текстов на естественном языке за счет того, что оно помогает группировать схожие слова [1].

Как уже говорилось в § 1, в рассмотренной задаче будет использоваться модель векторного представления слов, разработанная Томасом Миколовым в 2013 году [1, 2].

3.1 Word2Vec

Word2Vec – это множество моделей, соотносящих слова с векторами из пространства R^n. Эти модели являются небольшими, нейронными сетями без

скрытого слоя, которые в процессе обучения строят вектора слов на основе контекстов этих слов. Word2Vec принимает на вход корпус текстов (например, вся англоязычная Википедия) и выдает векторное представление слов из корпуса. Размерность векторов задается пользователем в диапазоне от нескольких десятков до нескольких сотен. В полученном пространстве вектора расположены так, что близки друг к другу (более сонаправлены) вектора слов, у которых схожие контексты. Так, например, к вектору слова `понедельник` будут ближе всего вектора слов, обозначающие другие дни недели. Еще эта модель примечательна тем, что в ней можно выявить достаточно сложные семантические связи между словами. Например, вектор `Германия - Берлин + Испания` $\approx$ `Мадрид` (примерно равен в том смысле, что вектор `Мадрид` – ближайший вектор к вектору этой линейной комбинации).

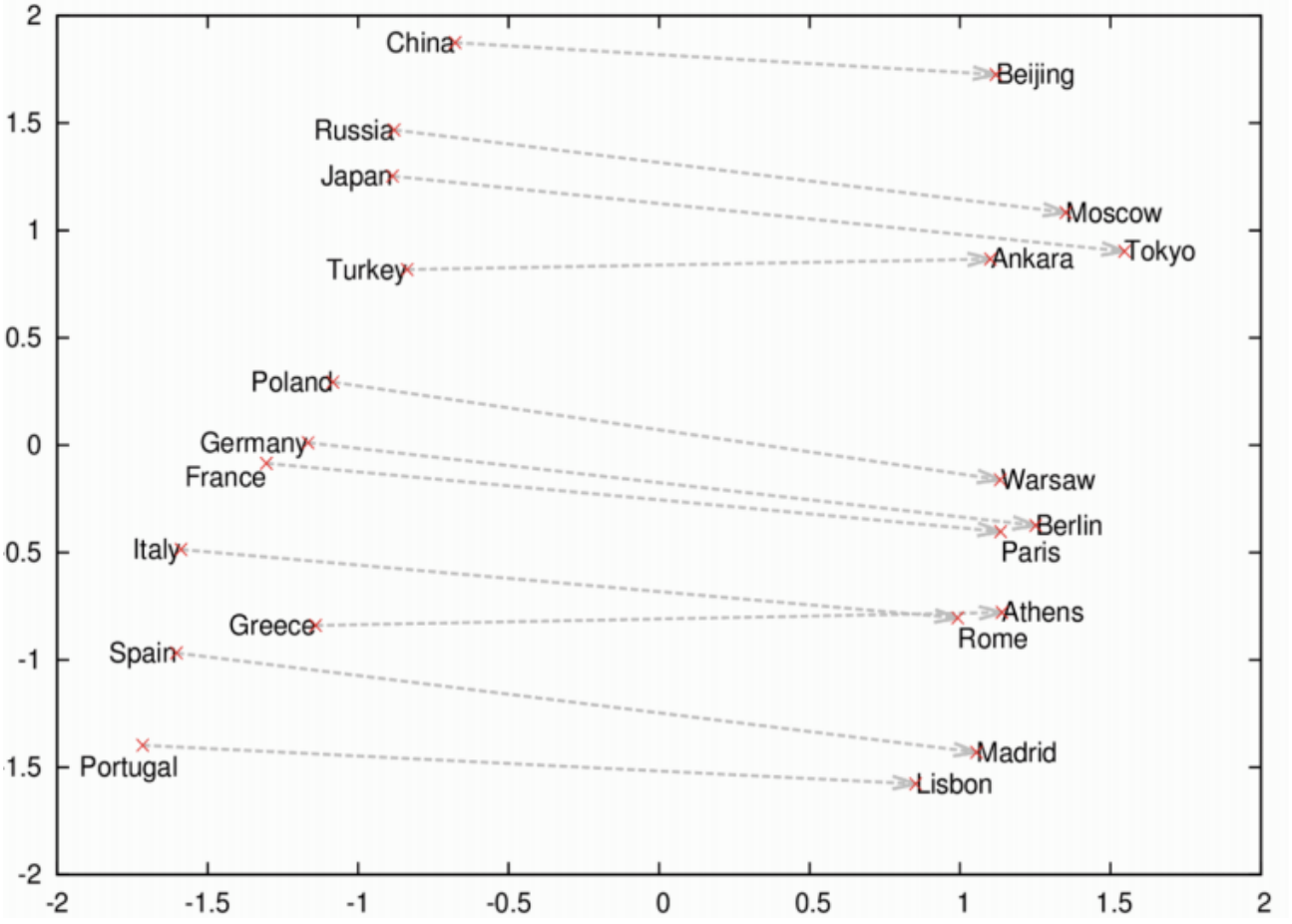

Рис. 1: Соотношение стран с их столицами после приведения N-мерного пространства к 2-ух мерному с помощью PCA (Principal Component Analysis). Взято из [1]

Основная идея модели – сближать вектора слов с векторами слов из контекста (контекстом слова называются C слов справа и слева от него в тексте, где C – некоторая константа, обычно берут $C = 5\text{-}10$). Основной результат –

близкие по значению слова получают близкие вектора. Есть несколько вариаций этой модели. Одна из них будет описана ниже. В работе используются вектора построенные этой модификацией word2vec'a.

3.1.1 Skip-gram

Для начала рассмотрим модель N-грамм. На вход этой модели подается корпус текстов $Text$, который состоит из слов w и их контекстов c. Также, мы рассматриваем условные вероятности $p(c|w)$ – вероятность встретить слово c в контексте слова w. Задача заключается в том, чтобы найти такие параметры θ, которые максимизируют вероятность по всему корпусу текстов:

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{w \in Text} \left(\prod_{c \in C(w)} p(c|w;\theta) \right) \tag{1}$$

в этом уравнении $C(w)$ – это множество контекстов слова w. Или по-другому:

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in} p(c|w;\theta) \tag{2}$$

здесь D - это множество всех пар слово-контекст, которые мы извлекаем из корпуса.

Можно рассмотреть простейший вариант в котором вероятность моделируется с помощью функции softmax:

$$p(c|w;\theta) = \frac{e^{v'_c \cdot v_w}}{\sum_{c' \in C} e^{v'_{c'} \cdot v_w}} \tag{3}$$

где v_w и $v'_c \in R^n$ – векторные представления слов w и c соответственно (важно, что векторное преставление слова и векторное представление этого же слова как контекста – это разные вектора, потому что необязательно слово встречается в контексте самого себя). C – множество всех контекстов. Параметры θ – это v_{c_i} и v_{x_i} для $w \in V$, $c \in C$ и $i \in 1, ..., n$ (всего $|V| \times |C| \times n$ параметров). Требуется найти такие параметры, чтобы максимизировать (2).

Теперь можно перейти от произведения к сумме логарифмов (так как логарифм – монотонно возрастающая функция):

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} p(c|w;\theta) = \tag{4}$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log p(c|w;\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \left(\log e^{v'_c \cdot v_w} - \sum_{c'} e^{v'^{c'} \cdot v^w} \right) \tag{5}$$

Но считать (5) вычислительно дорого, поэтому применяется другая модель, вычислительно менее затратная, но по эффективности она не уступает рассмотренной выше. называется модель *Negative Sampling*

3.1.2 Negative Sampling

В этой модели вместо того, чтобы рассматривать все возможные контексты, берется несколько отрицательных контекстов и максимизируется вероятность того, что они не близки со словом w. Отрицательные контексты – это случайные слова из корпуса. Делается это для того, чтобы избежать тривиального решения, когда все вектора слов будут одинаковые.

Пусть $p(D = 1|w, c)$ – вероятность, что (w, c) будет из корпуса, а $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ – вероятность, что (w, c) будет не из корпуса, D' – корпус случайных контекстов. А вероятность моделируется сигмоидой $\sigma(x) = \frac{1}{1+e^{-x}}$:

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v'_c \cdot v_w}} \tag{6}$$

Тогда задача заключается в том, чтобы подобрать такие параметры θ, что:

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \prod_{(w,c)\in D'} p(D = 0|w, c; \theta) \tag{7}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v'_c \cdot v_w}} + \sum_{(w,c)\in D'} \log 1 - \frac{1}{1 + e^{-v'_c \cdot v_w}} \tag{8}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v'_c \cdot v_w}} + \sum_{(w,c)\in D'} \log \frac{1}{1 + e^{v'_c \cdot v_w}} \tag{9}$$

после замены $\sigma(x) = \frac{1}{1+e^{-x}}$ получается:

$$\theta = \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \sigma\left(v'_c \cdot v_w\right) + \sum_{(w,c)\in D'} \log \sigma\left(v'_c \cdot v_w\right) \tag{10}$$

В модели Миколова [1, 2] D' строится таким образом, что каждой положительной паре слово-контекст $(w, c) \in D$ соответствует k отрицательный пар $(w, c_j) \in D'$.

3.2 Векторные представления многозначных слов

Представление каждого значения многозначного слова как отдельного вектора может значительно улучшить текущие векторные модели (такие как

word2vec). Как показывают эксперименты, такое разделение действительно повышает качество работы векторных моделей на некоторых задачах (например, определение части речи, определение семантических взаимосвязей), но на некоторых – нет (например, определение тональности текста) [3]. Эксперименты проводились с алгоритмом Chinese restaurant processes. Так как этот алгоритм показал улучшения в некоторых задачах, значит, имеет смысл рассматривать и пытаться улучшать модели векторного представления многозначных слов.

4 Выделение значений слов

WSI (Word Sense Induction) – задача выделения значений слов. На выходе необходимо получить инвентарь значений слов, в котором каждому значению соответствует набор близких по значению слов. Эта задача может решаться полностью машинным способом (обучение без учителя) с помощью кластеризации по значениям. Также, инвентарь значений можно получить с помощью разметки примеров (за счет крауд-сорсинга) [5]. В работе используются и те, и другие инвентари (см. таблицу 1).

Word	id	Neighbours
bank	1	financial institution:31, depository:16, thrift:9...
bank	2	side:4, border:2, shore:2, riverfront:1, embankment:1...
car	1	automobile:34, vehicle:27, auto:14, ride:12...
car	2	wagon:5, carriage:5, train car:4, bucket:4, unit:4...
language	1	dialect:20, speech:19, vocabulary:16, word:13...
language	2	computer language:10, programming language:7...
match	1	contest:37, game:26, competition:22, bout:13...
match	2	equal:16, duplicate:7, double:5, copy:5...
square	1	quadrate:10, four-side:10, equal-side:10, quadratic:7...
square	2	plaza:4, town square:2, center:2, market square:1...
square	3	squared:1, squared in area:1, quadratic:1...

Таблица 1: Примеры из инвентаря, полученного на основе размеченных данных. В левом столбце расположены слова, по середине – номер значения, в правом – слова, близкие по значению.

5 Разрешение лексической многозначности

На основе имеющегося инвентаря и векторов слов необходимо построить алгоритм, который будет разрешать неоднозначность. Собственно, эта часть – это и есть WSD.

5.1 Построение векторов значений

Вектора значений можно построить разными способами. В данной работе используется вариант, когда из инвентаря берутся векторные представления близких по значению слов (см. таблицу 1), и считается их среднее арифметическое. Итоговый вектор и есть вектор данного значения слова. Т.е.:

$$v_{sense_i} = \frac{\left(\sum_{v_{neigh} \in N} v_{neigh}\right)}{|N|} \tag{11}$$

где v_{word_i} – вектора i-го значения слова $word$, v_{neigh} – вектора близких по значению слов из инвентаря, N – множество этих векторов.

5.2 Выбор значения

На вход подается предложение с неоднозначным словом. Необходимо на основе контекста (в данном случае, контекст – это все слова из предложения, кроме самого неоднозначного слова) этого слова определить, какой из векторов значений больше всего подходить в данном случае. Есть несколько подходов к решению этой задачи:

1. Самый простой вариант – посчитать для каждого значения неоднозначного слова произведение условных вероятностей встречи с каждым контекстным словом, и выбрать то значение, для которого это произведение наибольшее

2. В предложении могут быть контекстные слова, которые могут быть недискриминативным (шумовыми), т.е. по ним нельзя определить значение слова (например, местоимения), но они могут дать суммарный прирост к неправильному значению больше, чем дискриминативные контексты, соответственно, необходимо некоторым образом отсекать шумовые контексты.

3. Придать больше веса дискриминативным контекстам и меньше – шумовым.

5.2.1 Сравнение по условным вероятностям

Рассмотрим первоначальный алгоритм. Входом для этого алгоритма будут вектора всех слов (word2vec), вектора значений неоднозначных слов, полученные в 5.1, и предложение с неоднозначным словом. Целью этого алгоритма является нахождение того значения слова, при котором произведение

вероятностей при условии каждого из соседних контекстов в предложении будет максимально. Т.е.:

$$i = \underset{i}{\operatorname{argmax}} \prod_{ctx \in S} p(word_i \mid ctx) = \tag{12}$$

$$= \underset{i}{\operatorname{argmax}} \sum_{ctx \in S} \log(p(word_i \mid ctx)) = \tag{13}$$

$$= \underset{i}{\operatorname{argmax}} \sum_{ctx \in S} \log \sigma(-v_{word_i} \cdot v_{ctx}) \tag{14}$$

где $\sigma(x) = \frac{1}{1+e^{-x}}$ – сигмоида, S – Множество слов в предложении, v_{word} – вектор слова $word$, $word_i$ – i-е значение слова $word$.

Идея заключается в том, что вычисляется $\log \sigma(v_{word_i} \cdot v_{ctx})$, где v_{word_i} – вектора i-го значения слова $word$, v_{ctx} – вектора контекстных слов, $(\cdot)$ – скалярное произведение, посчитать их сумму для каждого значения слова i и сравнить, в каком случае она будет больше. Это значение будет считаться правильным. $\log \sigma(v_{word_i} \cdot v_{ctx})$ – это мера близости двух слов. Результаты работы этого алгоритма есть в таблицах 2, 3 и 4 в строке $\gamma = 0$. Т.е в лучшем случае точность $< 55\%$.

Пример предложения: *By comparison, a disassembler translates an executable program into assembly language (an assembler could be used to assemble it back into an executable program).* Здесь $language_0$ – естественный язык, $language_1$ – язык программирования:

$P(By \mid language_0) = 0.4878$ $P(By \mid language_1) = 0.4995$
$P(comparison \mid language_0) = 0.5049$ $P(comparison \mid language_1) = 0.5064$
$P(disassembler \mid language_0) = 0.5281$ $P(disassembler \mid language_1) = 0.5960$
$P(translates \mid language_0) = 0.5569$ $P(translates \mid language_1) = 0.5272$
$P(an \mid language_0) = 0.4936$ $P(an \mid language_1) = 0.5383$
$P(executable \mid language_0) = 0.5156$ $P(executable \mid language_1) = 0.5863$
$P(program \mid language_0) = 0.5074$ $P(program \mid language_1) = 0.5485$

$P(into \mid language_0) = 0.5032$ $P(into \mid language_1) = 0.5143$
$P(assembly \mid language_0) = 0.5458$ $P(assembly \mid language_1) = 0.5396$
$P(an \mid language_0) = 0.4936$ $P(an \mid language_1) = 0.5383$
$P(assembler \mid language_0) = 0.5426$ $P(assembler \mid language_1) = 0.5532$
$P(could \mid language_0) = 0.5026$ $P(could \mid language_1) = 0.5170$
$P(be \mid language_0) = 0.4987$ $P(be \mid language_1) = 0.5251$
$P(used \mid language_0) = 0.5216$ $P(used \mid language_1) = 0.5298$
$P(assemble \mid language_0) = 0.5238$ $P(assemble \mid language_1) = 0.5106$
$P(it \mid language_0) = 0.5206$ $P(it \mid language_1) = 0.5409$
$P(back \mid language_0) = 0.4981$ $P(back \mid language_1) = 0.5052$
$P(into \mid language_0) = 0.5032$ $P(into \mid language_1) = 0.5143$
$P(an \mid language_0) = 0.4936$ $P(an \mid language_1) = 0.5383$
$P(executable \mid language_0) = 0.5156$ $P(executable \mid language_1) = 0.5863$
$P(program \mid language_0) = 0.5074$ $P(program \mid language_1) = 0.5485$

Сумма $\log \sigma(v_{word_i} \cdot v_{ctx})$ для $language_0 = -14.045819689817881$, а для $language_1$ = -13.10638118238573. Соотвественно, выбирается второй вариант, который и подразумевается в этом предложении.

5.2.2 Отсечение по константе

Как говорилось ранее, контексты могут быть недискриминативными, шумовыми. По ним нельзя определить, какое значение неоднозначного слова используется. Например: *"Я купил репчатый лук"* и *"Я стреляю из лука"*. В данном случае *"Я"* является недискриминативным контекстом, а *"репчатый"* и *"стреляю"* дискриминативны. По примеру из 5.2.1 видно, что таких контекстов, для которых $P(ctx|word_i) \approx P(ctx|word_j)$ достаточно много и они могут помешать алгоритму определить правильное значение многозначного слова. С такими контекстами можно попробовать бороться отсечение. Например, будут учитываться только те контексты, которые выдают отличающиеся больше, чем на какую-то константу γ, значения меры близости для этого контекста и хотя бы одной пары значений неоднозначного слова. Формально это выглядит так:

$$i = \underset{i}{\operatorname{argmax}} \prod_{ctx \in S_{discr}} p(word_i \mid ctx) = \tag{15}$$

$$= \underset{i}{\operatorname{argmax}} \sum_{ctx \in S_{discr}} \log(p(word_i \mid ctx)) = \tag{16}$$

$$= \operatorname{argmax}_i \sum_{ctx \in S_{discr}} \log \sigma(-v_{word_i} \cdot v_{ctx}) \quad (17)$$

$$S_{discr} = \{ctx \mid \exists\, i, j : \log \sigma(-v_{word_i} \cdot v_{ctx}) - \\ - \log \sigma(-v_{word_j} \cdot v_{ctx}) < \gamma\} \quad (18)$$

где γ – наперед заданный параметр, v_{word_i} и v_{word_j} – это вектора i-го и j-го значений слова $word$.

Этот метод работает крайне плохо. Из таблицы 2 и рис. 2 видно, что с линейным увеличением параметра γ линейно уменьшается точность, а при $\gamma = 0.1$ точность $\approx 38\%$, а такую точность можно получить, выбирая все время один и тот же вариант ответа (например, выбирая только первое значение слова). Соответственно, необходимо как-то модифицировать идею отсечения шумовых контекстов, и вместо константы динамически подбирать порог отсечения.

	Precision	Recall	F1
$\gamma = 0$	0,549	0,549	0,549
$\gamma = 0.01$	0,549	0,549	0,549
$\gamma = 0.02$	0,523	0,523	0,523
$\gamma = 0.03$	0,481	0,481	0,481
$\gamma = 0.04$	0,443	0,443	0,443
$\gamma = 0.05$	0,419	0,419	0,419
$\gamma = 0.06$	0,402	0,402	0,402
$\gamma = 0.07$	0,393	0,393	0,393
$\gamma = 0.08$	0,387	0,387	0,387
$\gamma = 0.09$	0,380	0,380	0,380
$\gamma = 0.1$	0,378	0,378	0,378

Таблица 2: Результаты работы программы с константным отсечением, 300-мерными векторами, обученными на новостях Google, и инвентарем, полученным на основе размеченных данных. В левом столбце параметр γ, дальше точность, полнота и F-мера. $\gamma = 0$ соответствует алгоритму без отсечения шумовых контекстов.

5.2.3 Динамическое отсечение

Вместо константы γ, возьмем некоторое значение $\gamma \cdot r(word, S)$, где S – множество контекстных слов, $word$ – множество значений неоднозначного слова, а $r()$ – некоторая функция от этих аргументов. Возьмем $r()$, такой, что:

$$r(word, S) = \max_{\forall\ word_i, word_j\ ctx \in S} \log \sigma(-v_{word_i} \cdot v_{ctx}) - \log \sigma(-v_{word_j} \cdot v_{ctx}) \quad (19)$$

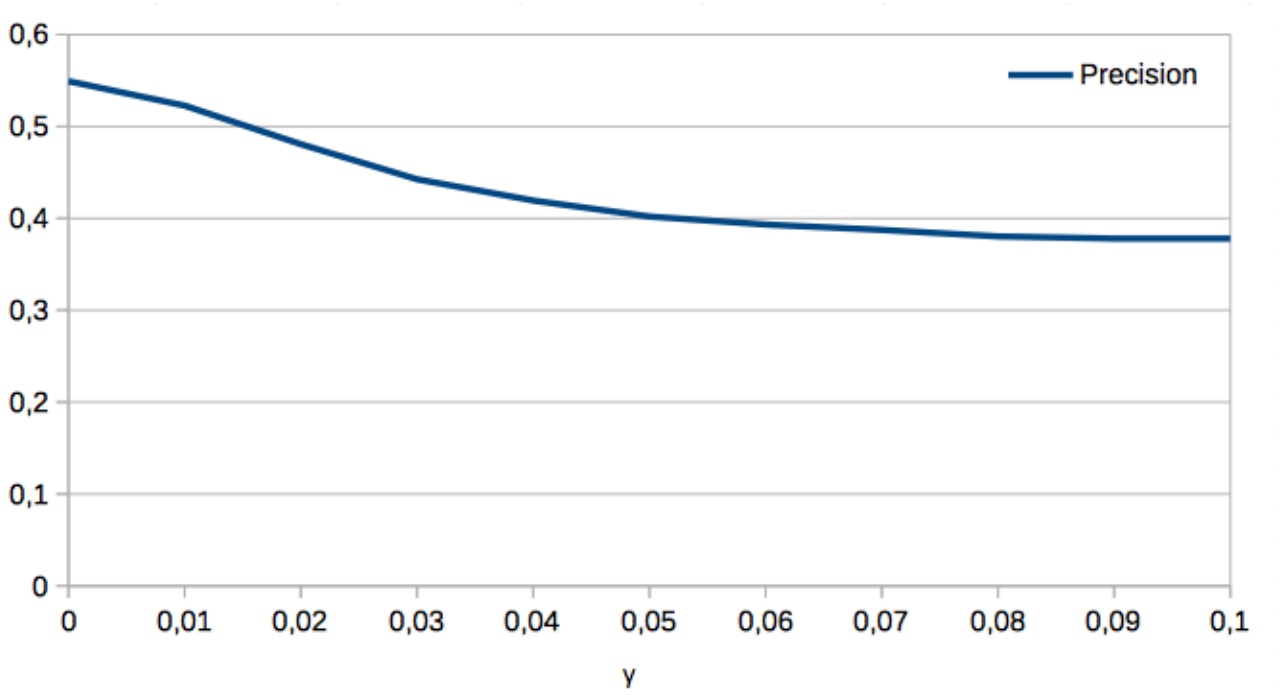

Рис. 2: График зависимости точности работы алгоритма разрешения лексической многозначности от параметра γ в случае константного отсечения.

Таким образом, задача заключается в поиске такого максимума:

$$i = \underset{i}{\operatorname{argmax}} \prod_{ctx \in S_{discr}} p(word_i \mid ctx) = \tag{20}$$

$$= \underset{i}{\operatorname{argmax}} \sum_{ctx \in S_{discr}} \log(p(word_i \mid ctx)) = \tag{21}$$

$$= \underset{i}{\operatorname{argmax}} \sum_{ctx \in S_{discr}} \log \sigma(-v_{word_i} \cdot v_{ctx}) \tag{22}$$

$$S_{discr} = \{ctx \mid \exists\ i, j : \log \sigma(-v_{word_i} \cdot v_{ctx}) - \\ - \log \sigma(-v_{word_j} \cdot v_{ctx}) < \gamma \cdot r(word, S)\} \tag{23}$$

где γ – наперед заданный параметр, v_{word_i} и v_{word_j} – это вектора i-го и j-го значений слова $word$.

Суть этого параметра γ заключается в том, что он показывает какую часть от максимальной разности логарифмов (см. (19)) надо брать как порог отсечения, т.е. если $\gamma = 0.2$, значит мы будет брать только те контексты которые хотя бы для одной пары значений неоднозначного слова дали разность большую чем $1/5$ от максимальной разности. Как видно из таблиц 3, 4 и рис. 3, при увеличении параметра γ точность работы алгоритма разрешения лексической многозначности растет на $1 - 2\%$, достигая максимума при $\gamma \approx 0.5$.

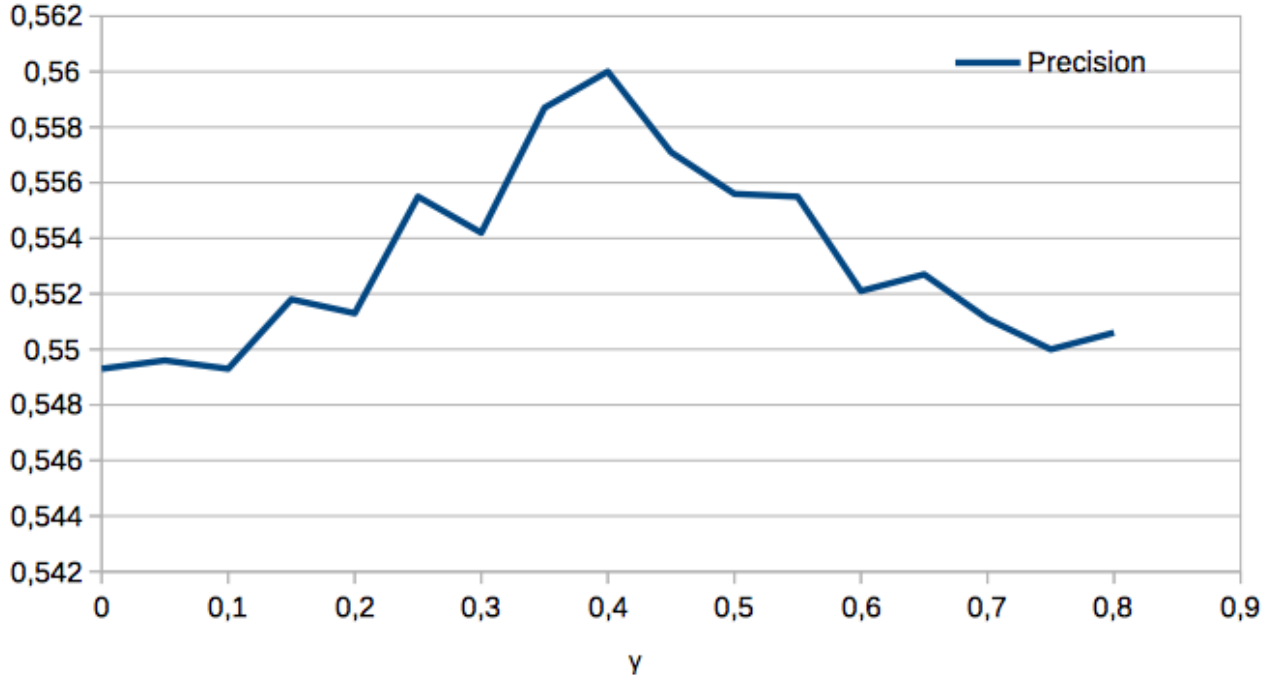

Рис. 3: График зависимости точности работы алгоритма разрешения лексической многозначности от параметра γ в случае динамического отсечения отсечения. Соответствует таблице 3.

	Precision	Recall	F1
$\gamma = 0$	0,549	0,549	0,549
$\gamma = 0.05$	0,550	0,550	0,550
$\gamma = 0.10$	0,549	0,549	0,549
$\gamma = 0.15$	0,552	0,552	0,552
$\gamma = 0.20$	0,551	0,551	0,551
$\gamma = 0.25$	0,556	0,555	0,555
$\gamma = 0.30$	0,554	0,554	0,554
$\gamma = 0.35$	0,559	0,559	0,559
$\gamma = 0.40$	0,560	0,560	0,560
$\gamma = 0.45$	0,557	0,557	0,557
$\gamma = 0.50$	0,556	0,556	0,556
$\gamma = 0.55$	0,556	0,555	0,555
$\gamma = 0.60$	0,552	0,552	0,552
$\gamma = 0.65$	0,553	0,553	0,553
$\gamma = 0.70$	0,551	0,551	0,551
$\gamma = 0.75$	0,550	0,550	0,550
$\gamma = 0.80$	0,551	0,551	0,551

Таблица 3: Результаты работы программы с динамическим отсечением, 300-мерными векторами, обученными на новостях Google, и инвентарем, полученного на основе размеченных данных. $\gamma = 0$ соответствует алгоритму без отсечения шумовых контекстов.

	Precision	Recall	F1
$\gamma = 0$	0,535	0,535	0,535
$\gamma = 0.05$	0,534	0,534	0,534
$\gamma = 0.10$	0,535	0,535	0,535
$\gamma = 0.15$	0,536	0,536	0,536
$\gamma = 0.20$	0,538	0,538	0,538
$\gamma = 0.25$	0,540	0,540	0,540
$\gamma = 0.30$	0,538	0,538	0,538
$\gamma = 0.35$	0,542	0,542	0,542
$\gamma = 0.40$	0,546	0,546	0,546
$\gamma = 0.45$	0,544	0,544	0,544
$\gamma = 0.50$	0,547	0,547	0,547
$\gamma = 0.55$	0,549	0,549	0,549
$\gamma = 0.60$	0,553	0,553	0,553
$\gamma = 0.65$	0,554	0,554	0,554
$\gamma = 0.70$	0,551	0,551	0,551
$\gamma = 0.75$	0,549	0,549	0,549
$\gamma = 0.80$	0,547	0,547	0,547

Таблица 4: Результаты работы программы с динамическим отсечением, 300-мерными векторами, обученными на англоязычной Википедии, и инвентарем, полученным на основе размеченных данных. $\gamma = 0$ соответствует алгоритму без отсечения шумовых контекстов.

6 Реализация

В работе используются готовые вектора: `GoogleNews`, обученные на новостях Google с помощью алгоритма Negative Sampling/Skip-gram (см. раздел 3.1) и вектора, обученные на английской Википедии `corpus-en.norm`. Еще используются инвентари: `Inventory-TWSI-2.csv`, полученный путем разметки данных, и инвентари `Inventory-59g-adgram-alpha-*.csv`, полученные путем кластеризации с помощью алгоритма Adgrad [4]. Для проверки используется таблица с примерами `TWSI-dev.csv`, по которой все измеряется. Результаты с инвентарем `Inventory-59g-adgram-alpha-15.csv` в таблице 5. Как видно по этой таблице, точность работы алгоритма с инвентарем из кластеризованных значений на 5-6% ниже, чем с инвентарем, полученным, на основе размеченных данных.

Часть алгоритма, ищущая максимум логарифмов на первом рис. с кодом. Часть алгоритма, отсекающая контексты по порогу, в зависимости от этого максимума (в варианте без отсечения *isRelevantCtx = True* всегда, а в случае константного отсечения вместо (*maxLp * gamma*) просто *gamma*), и выдающая ответ в виде номера значения из инвентаря многозначных слов на втором рис. с кодом.

```
def predict(self, cdf, gamma):
    ...
        for word in words: # words -- контекстные слова из предложения
            lp = [self.logprob(word, svecs[j]) for j in xrange(cntSenses)]

            for i in xrange(cntSenses):
                for j in xrange(i + 1, cntSenses):
                    diff = lp[i] - lp[j]  #lp -- логарифм от сигмоиды
                    if np.abs(diff) > maxLp:
                        maxLp = np.abs(diff)
    ...
```

Рис. 4: Начало описания функции `predict` на языке Python. которая принимает на вход параметр `gamma`, вектора слов, инвентарь и тестовый набор предложений. Во внешнем цикле перебираются все контекстные слова из текущего предложения, а затем во внутреннем двойном цикле ищется максимум разности логарифмов.

7 Заключение

В результате проведенных экспериментов с алгоритмом разрешения лексической многозначности, было выявлено, что базовая версия алгоритма дает точность примерно 53-55%. Также, стоит отметить, что бороться с недискриминативными контекстами путем отсечения их по константе, – плохой вариант. И напротив, если выбирать порог отсечения динамически, то точность работы алгоритма, пускай незначительно, но растет. Из проведенных экспериментов видно, что инвентарь, основанный на размеченных данных показывает себя лучше, чем инвентарь, полученный путем кластеризации.

В дальнейшем развивать базовое решение можно в нескольких направлениях: во-первых, имеет смысл попробовать другим способом строить вектор значений неоднозначного слова, нежели в 5.1, а во-вторых, можно попытаться использовать упомянутый в 5.2 в 3-м пункте, но не рассмотренный, вариант, в котором каким-то образом контекстам будут придаваться веса, чтобы больше выделялись дискриминативные.

В заключение стоит отметить, что разрешение лексической многозначности – это сложная задача, при изучении которой обнаруживается большое количество трудностей, начиная с проблемы составления словарей и определения частей речи (задачей, тесно связанной с задачей разрешения лексической многозначности), и заканчивая человеческим фактором.

	Precision	Recall	F1
$\gamma = 0$	0,484	0,483	0,483
$\gamma = 0.05$	0,484	0,484	0,484
$\gamma = 0.10$	0,483	0,483	0,483
$\gamma = 0.15$	0,483	0,483	0,483
$\gamma = 0.20$	0,485	0,485	0,485
$\gamma = 0.25$	0,489	0,489	0,489
$\gamma = 0.30$	0,489	0,489	0,489
$\gamma = 0.35$	0,494	0,494	0,494
$\gamma = 0.40$	0,493	0,493	0,493
$\gamma = 0.45$	0,494	0,494	0,494
$\gamma = 0.50$	0,494	0,494	0,494
$\gamma = 0.55$	0,495	0,495	0,495
$\gamma = 0.60$	0,496	0,495	0,495

Таблица 5: Результаты работы программы с динамическим отсечением, 300-мерными векторами, обученными на англоязычной Википедии, и инвентарем, полученным с помощью алгоритма Adgrad [4].

```
...
    for word in words:
        lp = [self.logprob(word, svecs[j]) for j in xrange(cntSenses)]

        vc = self.vc
        vctx = vc.syn0norm[vc.vocab[word].index]
        for i in xrange(cntSenses):
            for j in xrange(i + 1, cntSenses):
                diff = lp[i] - lp[j]
                if np.abs(diff) > (maxLp * gamma): #отсечение по порогу,
                                                    зависмому от максимума
                    isRelevantCtx = True
        if isRelevantCtx:
            for j in xrange(cntSenses):
                s[j] += dots[j]
    ind = np.argsort(s)
    if len(senses) != 0:
        sense = senses[ind[-1]]
        context = contexts[ind[-1]]
        y.append((sense, id))
...
return y
```

.

Рис. 5: Продолжение функции `predict`. В переменной `maxLp` хранится максимум разностей логарифмов. Теперь алгоритм проходит по тому тройному циклу и отсекает те контексты, для которых не нашлось ни одной пары значений, чтобы разность логарифмов для них была больше `(maxLp * gamma)`

Список литературы

[1] Mikolov, Tomas; et al. *Distributed Representations of Words and Phrases and their Compositionality.* 2013. PDF `http://arxiv.org/pdf/1310.4546v1.pdf`

[2] Yoav Goldberg and Omer Levy *Word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method.* 2014. PDF `arxiv.org/pdf/1402.3722v1.pdf`

[3] Jiwei Li, Dan Jurafsky *Do Multi-Sense Embeddings Improve Natural Language Understanding?.* 2014. PDF `nlp.stanford.edu/pubs/`

[4] Sergey Bartunov; et al. *Breaking Sticks and Ambiguities with Adaptive Skip-gram.* 2015. PDF `arxiv.org/pdf/1502.07257v2.pdf`

[5] Chris Biemann *Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems.* 2006. PDF