Формирование релевантного множества слов в текстах бытового юмора методом Word2Vec[1]

В. А. Каладзе, email: wakaladze@yandex.ru[1]
А. А. Ворсунов, email: chuvooooo@gmail.com[2]

[1] Международный институт компьютерных технологий (МИКТ)
[2] Международный институт компьютерных технологий (МИКТ)

Аннотация. *В данной работе применён метод Word2Vec и его основные алгоритмы с использованием модели из библиотеки gensim. В работе проведён анализ полученных результатов, приводятся примеры прогноза релевантных слов методом Word2Vec*

Ключевые слова: *компьютерная лингвистика, обработка естественного языка, синтаксис, семантика, векторы слов, Word2Vec, косинусное сходство, CBOW, Skip-gram, gensim, jupyter-notebook*

Введение

Данное исследование относится к области компьютерной лингвистики, связанной с идентификацией алгоритмов для формального описания текстов естественных языков. Для компьютерной реализации подобных моделей исходные слова естественного языка проецируются в элементы числового векторного пространства, поскольку в компьютере происходит обработка только числовых данных, а непосредственно текстовую информацию компьютер не может воспринимать так же, как это делает человек. При этом вводимая информация должна отражать все особенности естественного языка.

При компьютерной реализации необходимо не только представлять текстовую информацию в числовой форме, но при этом сохранить уникальный синтаксис и семантику исходного текста. На всём протяжении развития компьютерной лингвистики в её арсенале появлялось множество способов обработки естественного языка (модель мешка слов one-hot-encoding, tf-idf и т.д.). Но, применяемая в исследовании модель Word2Vec примечательна тем, что впервые позволила полноценно сохранять конструкцию и семантику языка, обеспечив тем самым прорыв в обработке естественного языка. Именно на ней и основана данная работа, целью которой является использование модели Word2Vec и её алгоритмов для поиска контекстных слов.

1. Принцип Word2Vec

Модель Word2Vec, разработанная Томашем Миколовым [1], была представлена в 2013 году. Принцип модели основан на том, что схожие слова реализуются в схожие пространственные векторы, которые формируются с учётом встречаемости слов в контексте. Так, вектор слова «мужчина» будет иметь большее сходство с вектором слова «человек», а вектор слова «пупырчатый» будет ближе по сходству со словом «огурец». Например, обученная сеть (модель) связывает слово «мужчина» со словом «женщина» так же, как слова «король» и «королева».

$$\frac{\text{Мужчина}}{\text{Женщина}} \sim \frac{\text{Король}}{\text{Королева}}$$

Рис. 1. Релевантные отношения

Модель, построенная на основе схожести понятий, сохраняет смысловую близость слов в их векторных образах, обеспечивая релевантность генерируемого машиной текста исходному контексту.

На рис. 1 приведена классическая иллюстрация, как модель в векторной форме связывает слова (мужчина : женщина) и (король : королева). Т.е. модель позволяет оценить «отличие/сходство» между этими словами в семантической связи через родовую принадлежность.

Помимо построения слов, как элементов векторного пространства, модель Word2Vec прогнозирует контекстное слово через векторную проекцию. В таком случае релевантное слово определяется через вероятность появления этого слова в указанном контексте:

$$p(w_0 / w_i) = \frac{\exp(v_{w_i} v_{w_0}^T)}{\sum_{j=1}^{N} \exp(v_{w_i} v_{w_j}^T)} \quad (1)$$

где w_0 – вектор целевого слова, w_i – вектор контекстной окрестности целевого слова. $s(w_1, w_2)$ – косинусное сходство.

Данная оценка вероятности основана на функции Softmax [2], представляющей собой обобщение сигмоидной функции на многомерный случай. Функция Softmax преобразует вектор размерности в вектор той же размерности, где каждая компонента полученного

вектора представлена вещественным числом из отрезка [0, 1]. При этом сумма компонент p_i вектора p равна 1.

Кроме восприятия семантики языка моделью решается ещё и проблема размерности полученных векторов. До её появления пользовались преимущественно методом one-hot-encoding [3], векторы которого выходили слишком объёмными и при этом разреженными. Так, для корпуса из 100 тыс. уникальных слов текст кодировался 100 тысячами векторов, содержащих по 100 тыс. компонент каждый. В Word2Vec корпус текста такой же мощности можно представить набором из 100 тыс. векторов с размерностью в 300 компонент. Модель Word2Vec основана на двух алгоритмах – CBOW и Skip-gram, которые описаны ниже по тексту.

При этом следует отметить, что CBOW реализует семантику контекстного множества, а второй алгоритм Skip-gram отвечает за синтаксис формируемого множества.

2. CBOW

Алгоритм CBOW (Непрерывная модель мешка слов) заключается в нахождении целевого слова по входящему контексту. Для примера, возьмем следующий анекдот: «Работа программиста и шамана имеет много общего - оба бормочут непонятные слова, совершают непонятные действия и не могут объяснить, как оно работает». Из анекдота используем небольшой отрывок, с рабочим окном равным двум для удобства в описании. Например, имеем:

«работа программиста и шамана имеет много», следовательно, алгоритм по входящим словам [$w(t\text{-}1)$=«шамана»; $w(t\text{+}1)$=«много»] должен предложить слово $w(t)$=«имеет».

Архитектура CBOW представлена схемой:

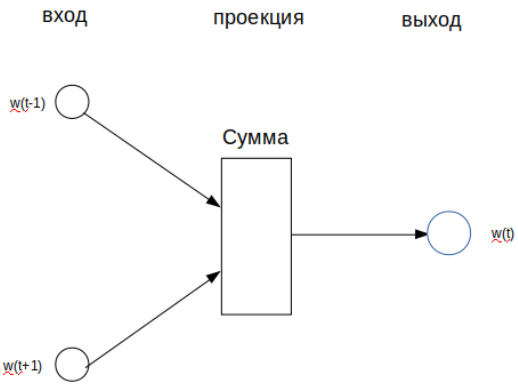

Рис. 2. Архитектура CBOW

Реализация нейронной сети состоит из трёх слоёв:

Входной слой. На данный слой поступают контекстные слова.

Слой Embedding. Данный слой-матрица имеет размерность $N*X$, где N – размер словаря, X – гиперпараметр, отражающий количество компонент в построенных векторах слов. Гиперпараметр подбирается под конкретный размер словаря, чаще всего эмпирически.

Выходной слой. Нейроны в данном слое используют вероятность того, что слово является целевым.

3. Skip-gram

Таким образом, CBOW подбирает целевое слово по окружающим его контекстным словам, в то время как Skip-gram формирует контекстные слова в окрестности целевого слова, т.е. действует в противоход CBOW.

В том же примере: «работа программиста и шамана имеет много» при входном слове [$w(t)$=«имеет»], алгоритм в ответ должен предложить два сопутствующих слова [$w(t\text{-}1)$=«шамана»; $w(t+1)$=«много»]. Архитектура Skip-gram представима в виде:

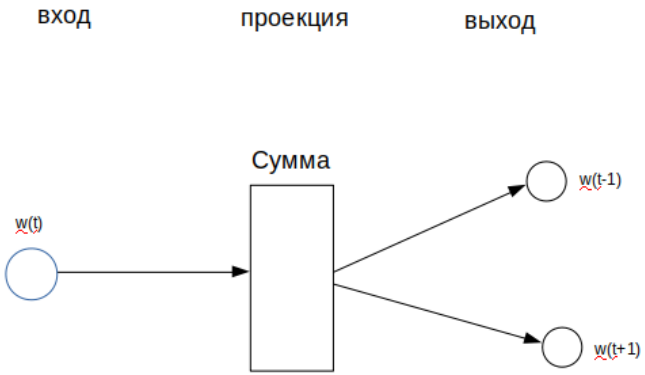

Рис. 3. Архитектура Skip-gram

В реализации Skip-gram в отличие от CBOW задачи входного и выходного слоёв поменяются местами: т.е. целевое слово будет подаваться на вход нейронной сети, а выходной слой будет прогнозировать контекстные слова из окрестности целевого слова.

4. Косинусное сходство

Определение косинусного сходства связано с оценкой расстояния между двумя элементами векторного пространства. При этом, косинусное сходство (КС) стремится к единице, когда угол между векторами стремится к нулю, т.е. они наиболее близки друг к другу. На этой основе определяется мера сходства между двумя словами, что так же позволяет определить наиболее вероятное продолжение фразы. Вычисляется КС векторов A и B на основе их скалярного произведения как косинус угла θ между ними

$$\cos\theta = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \qquad (2)$$

Данный метод имеет существенный недостаток. Для определения наиболее вероятного слова в предложении необходимо всякий раз рассчитывать КС для всех имеющихся векторов. Можно рассчитать заранее и хранить в памяти значения КС всех возможных слов. Но это тоже не эффективно, т.к. такие комбинации будут занимать много

памяти, если учесть, что, например, в Большом академическом словаре русского языка насчитывается около 145 000 слов современного русского литературного языка.

Решением данной проблемы может послужить реализация концепции вложений, при которой Word2Vec используется в качестве вспомогательного инструмента.

5. Концепция вложений

Концепция вложений заключается в том, что полученные векторы передаются на специальный слой другой нейронной сети, преобразующий целые числа (индексы слов) в вещественные числа. Вторая нейронная сеть уже непосредственно занимается решением поставленной задачи. К примеру, в задаче генерации текста нейронная сеть, реализующая модель Word2Vec строит векторы слов, которые затем подаются на слой embedding (keras.tensorflow). На этом слое целочисленный ввод будет сопоставляться с полученным векторным представлением. Все дальнейшие действия будут происходить именно с векторами, полученными из Word2Vec-модели. Затем используется вторая нейронная сеть (обычно рекуррентная), обеспечивающая генерацию текста.

6. Реализация в библиотеке gensim

Реализация проходит в gensim – библиотеке с открытым исходным кодом на python, предназначенной для обработки естественного языка. Данная библиотека представляет собой удобный инструмент для реализации множества моделей. В данном случае модель сформирована с использованием среды jupyter-notebook, которая удобна тем, что код можно выполнять поэтапно, при этом выполненные ранее блоки кода останутся в памяти. Следовательно, модель можно единожды обучить и с ней взаимодействовать, не обучая её заново каждый раз. Среда jupyter-notebook для языка python (версии 3.10.5), библиотеки gensim (версии 4.2.0), библиотеки pandas (версии 1.4.3). В других версиях используемые программные методы могут быть не реализованы/не актуальны.

Текст для обучения состоит из набора бытовых анекдотов Предложений в текстовом корпусе более 200 тыс.

Прежде чем использовать модель, необходимо подготовить текст, удалив регистр и лишние символы. К примеру, если не удалять регистр, слово «Я» и слово «я» будут разными словами, что сильно увеличит корпус, при этом никакого практического смысла оставлять регистр нет.

Текст хранится в файле excel с расширением xlsx. Для работы с данным форматом используется библиотека pandas.

```
# Открытие файла
```

```
excel_data = pd.read_excel("text.xlsx", header=None)
# Запись в переменную data текст, хранящийся в специальном
типе данных pandas
data = pd.DataFrame(excel_data)
texts_frame = []
# Запись из типа данных DataFrame в массив
for i in data.values.tolist():
# Проверка на пустые строки. Если строка пустая - в корпус она
не попадет.
if len(str(i[0])) > 0:
# Удаление всех символов, кроме русских букв с помощью
регулярного выражения
seq = re.sub(r'[^а-я]', ' ', str(i[0]).lower())
# Запись элемента в массив
texts_frame.append(seq)
# Удаление далее неиспользуемых переменных из памяти
del data, excel_data
# Функция для перемешивания текста
texts_frame = sorted(texts_frame, key=lambda A:
random.random())
len(texts_frame)
```

После обработки в корпусе остаётся 203396 предложений.

Затем следует текст привести к необходимому виду для модели согласно документации gensim:

```
max_sentence_len = 40
sentences = [[word for word in
text.lower().split()[:max_sentence_len]] for text in
texts_frame]
```

Таким образом, всё подготовлено для обучения модели. Следующий шаг – создание экземпляра класса Word2Vec gensim.

```
model = Word2Vec(sentences, vector_size=300, min_count=1,
window=2, epochs=150, sg=1)
```

Здесь sentences – ранее подготовленный текст, vector_size – размерность получаемых векторов, min_count – минимальное количество вхождений слова в текст. window – размер окна, epochs – количество эпох для обучения модели. sg – алгоритм построенной модели (1-skip-gram, 0-cbow).

Модель была успешно обучена. В следующем пункте приведён анализ уже обученной модели.

7. Анализ полученных результатов

После обучения модели можно использовать её в ранее описанных целях. К примеру, найдем ближайшие слова по косинусному сходству для слова «имеет», при обучении модели алгоритмом skip-gram:

```
[24]: model.wv.most_similar("имеет")

[24]: [('шамана', 0.9917073845863342),
       ('много', 0.9909415245056152),
       ('общего', 0.9863792061805725),
```

Рис. 4. Нахождение ближайших слов по косинусному сходству

Нахождение ближайших слов сильно зависит от конкретного корпуса текстов. Тем не менее, теоретический прогноз из пункта 1.2 подтвердился на практике.

С помощью метода get_vector можно получить итоговый вектор для определённого слова:

```
[36]: model.wv.get_vector("имеет")

[36]: array([ 0.16725959, -0.2949919 , -0.04586571, -0.02772975, -0.19318959,
             -0.41272947,  0.42536595,  0.05973024, -0.11448608, -0.08169582,
              0.40811756, -0.26969838, -0.11471198, -0.19049269,  0.23646633,
              0.1833827 ,  0.2932885 ,  0.18268205, -0.06373168,  0.20025976,
              0.22068545,  0.6403838 ,  0.35625103, -0.17701311,  0.12431911,
              0.01302469, -0.32920983,  0.4237583 ,  0.24096496, -0.1745416 ,
             -0.0183533 ,  0.01736533], dtype=float32)
```

Рис. 5. Вектор слова «имеет»

Кроме того, полученные векторы можно сохранить в отдельный файл для последующей реализации по концепции вложений, описанной выше.

Для визуализации векторов в пространстве используется метод библиотеки TSNE [6]:

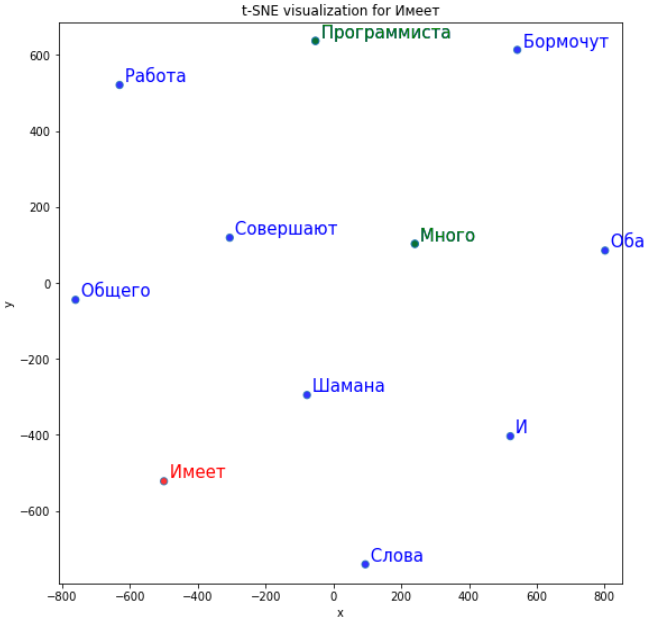

Рис. 6. Множество контекстных векторов, визуализированных в словесной форме

Как видно из рисунка, модель улавливает контекстные связи и строит семантически верное представление текста в пространстве.

Заключение

Word2Vec – эффективный метод представления текстовой информации, позволяющий сохранять семантику естественных языков в векторном пространстве. В работе описан принцип работы Word2Vec, исследованы основные алгоритмы Word2Vec, обучена исследовательская модель на основе модели библиотеки gensim. Метод Word2Vec на данный момент лежит в основе современных моделей типа трансформер [7], что определяет исследование этого метода, как важное направление в области компьютерной лингвистики и обработки естественного языка.

Список литературы

1. Efficient Estimation of Word Representations in Vector Space [Электронный ресурс] : науч. статья. – Режим доступа : https://arxiv.org/pdf/1301.3781.pdf

2. Softmax [Электронный ресурс] : науч. журн. – Режим доступа : https://congyuzhou.medium.com/softmax-3408fb42d55a

3. Survey on categorical data for neural networks [Электронный ресурс] : науч. журн. – Режим доступа : https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00305-w

4. tf-idf – статистическая мера оценки важности слова в контексте документа [Электронный ресурс] : науч. журн. – Режим доступа : https://habr.com/ru/company/skillfactory/blog/566414/

5. An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges [Электронный ресурс] : науч. статья. – Режим доступа : https://www.researchgate.net/publication/338511771_An_Overview_of_Bag_of_WordsImportance_Implementation_Applications_and_Challenges

6. sklearn.maniford.TSNE [Электронный ресурс] : документация – Режим доступа : https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

7. Attention Is All You Need [Электронный ресурс] : науч. статья, – Режим доступа : https://arxiv.org/pdf/1706.03762.pdf