# Практический подход к методам глубокого обучения для текстовых данных — Word2Vec: CBOW

# Введение

Работать с неструктурированными текстовыми данными сложно, особенно когда вы пытаетесь построить интеллектуальную систему, которая интерпретирует и понимает свободный естественный язык, как люди. Вам нужно уметь обрабатывать и преобразовывать зашумленные, неструктурированные текстовые данные в некоторые структурированные, векторизованные форматы, которые могут быть поняты любым алгоритмом машинного обучения.

# Недостатки традиционных моделей

Традиционные (на основе подсчета) стратегии разработки признаков для текстовых данных включают модели частоты терминов, TF-IDF (частота терминов, обратная частоте документа), N-граммы и так далее.

Хотя они являются эффективными методами извлечения признаков из текста, из-за того, что модель представляет собой просто набор неструктурированных слов, мы теряем дополнительную информацию, такую ​​как семантика, структура, последовательность и контекст рядом стоящих слов в каждом текстовом документе. Это является достаточной мотивацией для нас, чтобы исследовать более сложные модели, которые могут собирать эту информацию и давать нам функции, которые являются векторным представлением слов, обычно известными как вложения.

Давайте рассмотрим некоторые из этих передовых стратегий обработки текстовых данных и извлечения из них значимых функций, которые можно использовать в последующих системах машинного обучения.

# Модель Word2Vec

Эта модель была создана Google в 2013 году и представляет собой модель, основанную на прогнозирующем глубоком обучении, для вычисления и создания высококачественных, распределенных и непрерывных плотных векторных представлений слов, которые фиксируют контекстуальное и семантическое сходство.

Word2Vec может использовать две разные архитектуры модели для создания этих представлений встраивания слов:

* **Модель непрерывного мешка слов (CBOW)**
* **Модель Skip-gram**

# Модель непрерывного мешка слов (CBOW)

Архитектура модели CBOW пытается предсказать текущее целевое слово (центральное слово) на основе слов исходного контекста (окружающих слов).

Рассматривая простое предложение **«Эта быстрая коричневая лиса перепрыгивает через ленивую собаку»**, это могут быть пары **(context\_window, target\_word),** где, если мы рассмотрим контекстное окно размера 2, у нас будут такие примеры, как **([быстрая, лиса], коричневая), ([эта, коричневая], быстрая), ([через, собака], ленивый)** и так далее.

Таким образом, модель пытается предсказать на target\_word основе context\_window слов.

Теперь мы можем смоделировать эту архитектуру CBOW как модель классификации глубокого обучения, так что мы берем **контекстные слова в качестве входных данных X** и пытаемся предсказать **целевое слово Y** .

На самом деле построить эту архитектуру проще, чем модель пропуска грамм, где мы пытаемся предсказать целую кучу контекстных слов из исходного целевого слова.

Реализация будет состоять из четырех частей:

1. **Формирование корпусной лексики**
2. **Создание генератора CBOW (контекст, цель)**
3. **Создание архитектуры модели CBOW**
4. **Обучение модели**
5. **Получение вложения слов**

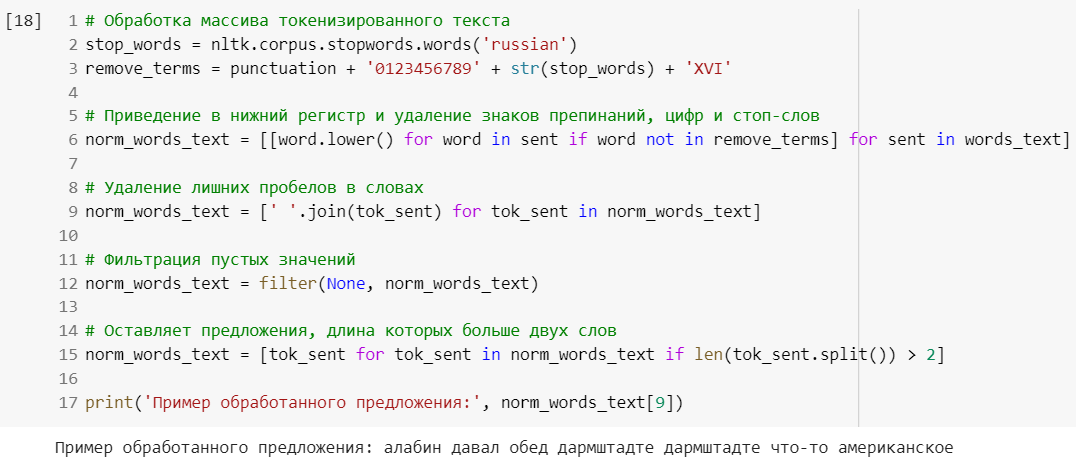
# Подготовительный этап

Для начала загрузим некоторые файлы. Первый с текстом произведения русского классика Льва Толстова «Анна Каренина», на котором мы будем формировать корпусную лексику и второй с текстом, на основе которого необходимо вывести реферат.



Следующий вывод показывает общее количество строк, полученных с текстового файла и то, как предварительная обработка работает с текстовым содержимым.



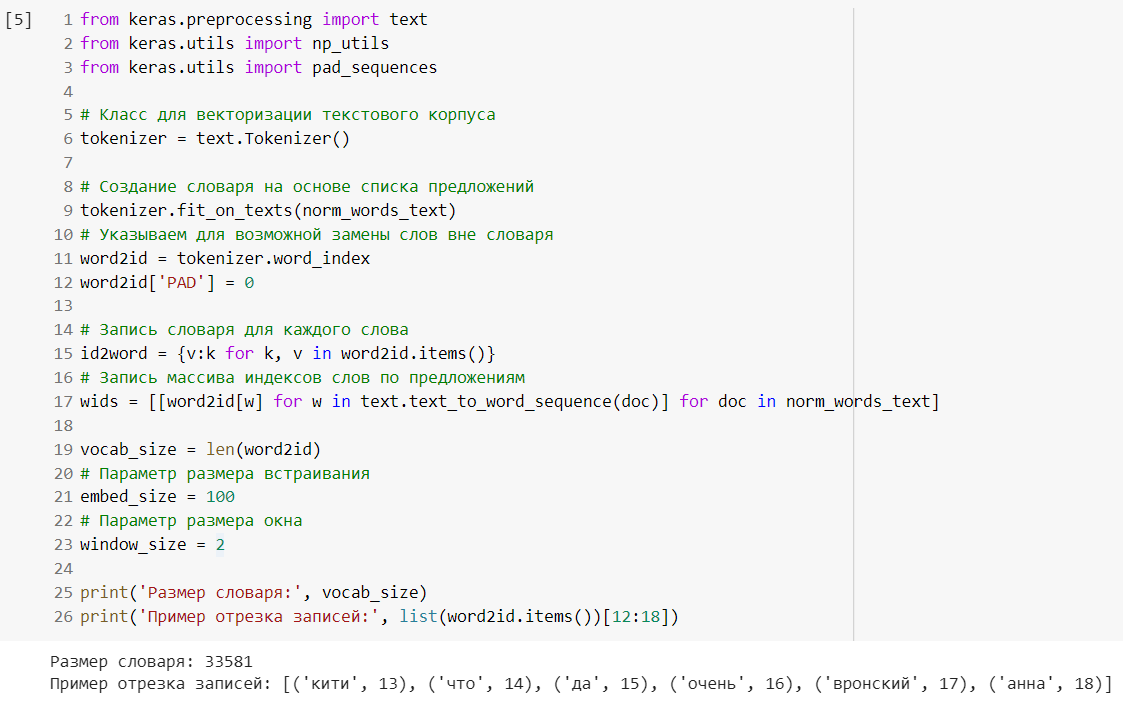


# Реализация модели непрерывного мешка слов (CBOW)

# ****Формирование корпусной лексики****

Для начала мы создадим наш словарь корпуса, где мы извлечем каждое уникальное слово из нашего словаря и сопоставим с ним уникальный числовой идентификатор.

Keras — открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями.



Embed\_size (параметр размера встраивания) - это количество измерений векторного пространства, в которое вложено слово

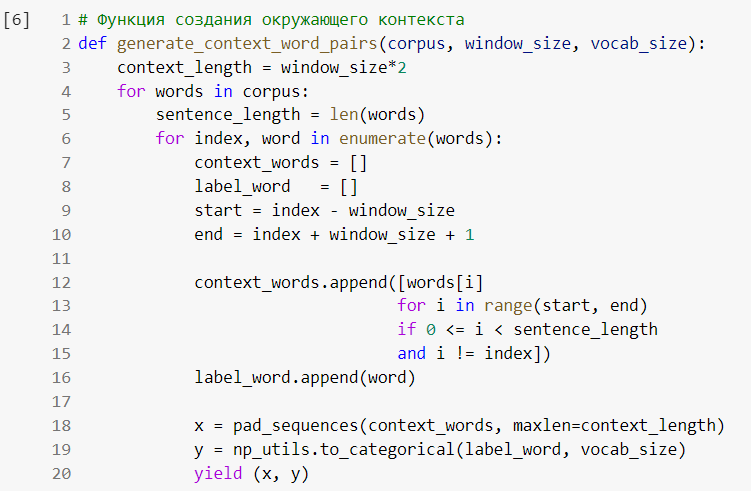
window\_size (размер окна) Gensim по умолчанию равен 5 (два слова до и два слова после входного слова, в дополнение к самому входному слову)

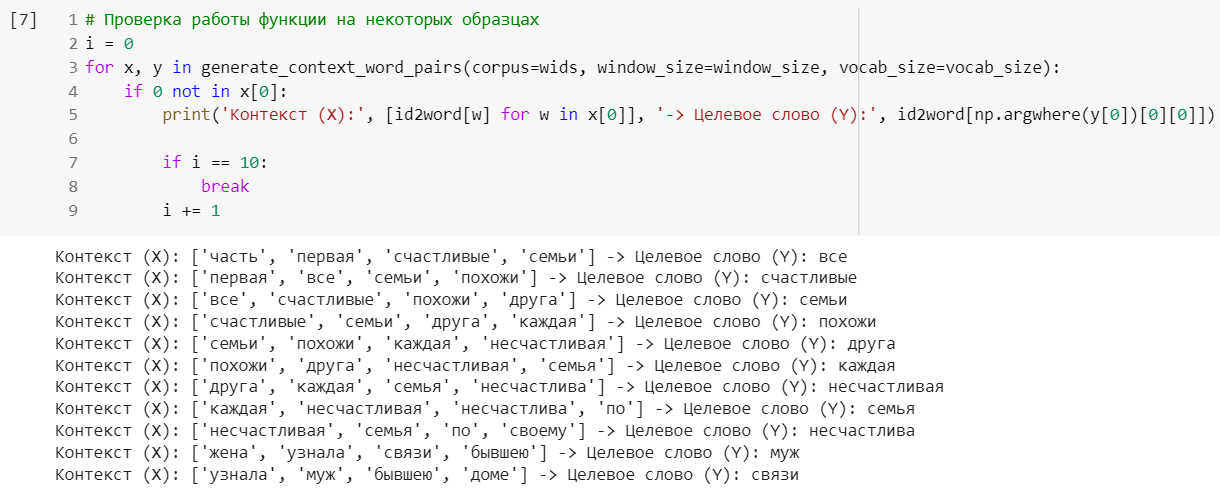
Таким образом, вы можете видеть, что мы создали словарь уникальных слов в нашем корпусе, а также способы сопоставления слова с его уникальным идентификатором и наоборот.

PAD обычно используется для дополнения слов контекста до фиксированной длины, если это необходимо.

# ****Создание генератора CBOW (контекст, цель)****

Нам нужны пары, состоящие из целевого центрального слова и слов окружающего контекста. В нашей реализации **целевое слово** имеет длину **1**, а **окружающий контекст** имеет длину **2 x window\_size**, когда мы берем **window\_size** слова до и после целевого слова в нашем корпусе.





Этот вывод должен дать некоторое представление о том, как **X** формирует наши контекстные слова, и мы пытаемся предсказать целевое центральное слово **Y** на основе этого контекста.

# ****Создание архитектуры модели CBOW****

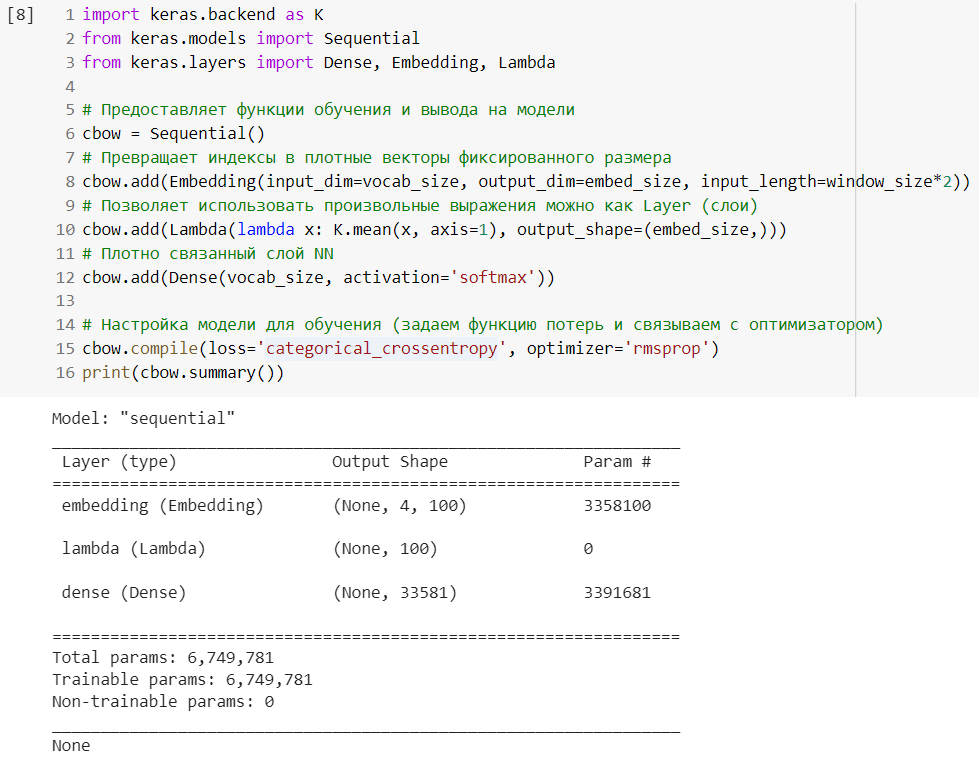
Теперь мы используем библиотеку tensorflow для создания нашей архитектуры глубокого обучения для модели CBOW.

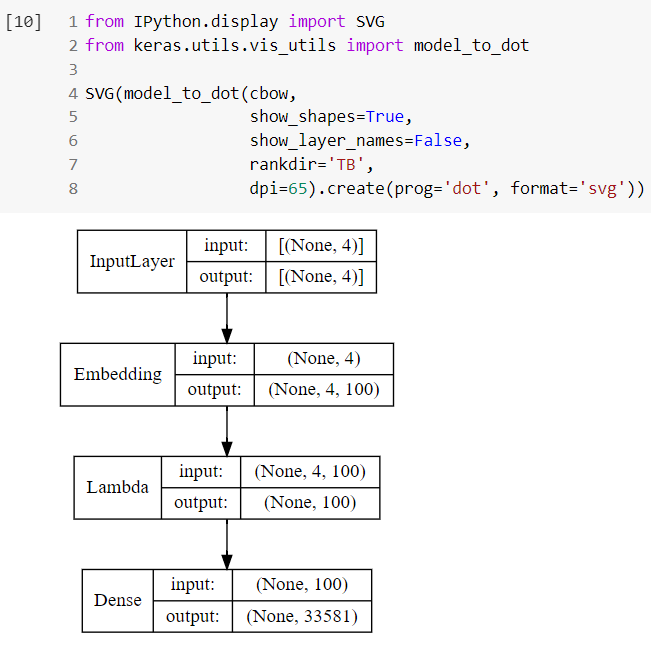
TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

Для этого нашими входными данными будут наши слова контекста, которые передаются на слой внедрения.

Вложения слов распространяются на лямбда-слой, где мы усредняем вложения слов, а затем мы передаем это усредненное вложение контекста в плотный softmax-слой, который предсказывает наше целевое слово.

Мы сопоставляем это с фактическим целевым словом, вычисляем потери, используя categorical\_crossentropy потери, и выполняем обратное распространение с каждой эпохой, чтобы обновить слой внедрения в процессе. Следующий код показывает нам архитектуру нашей модели.





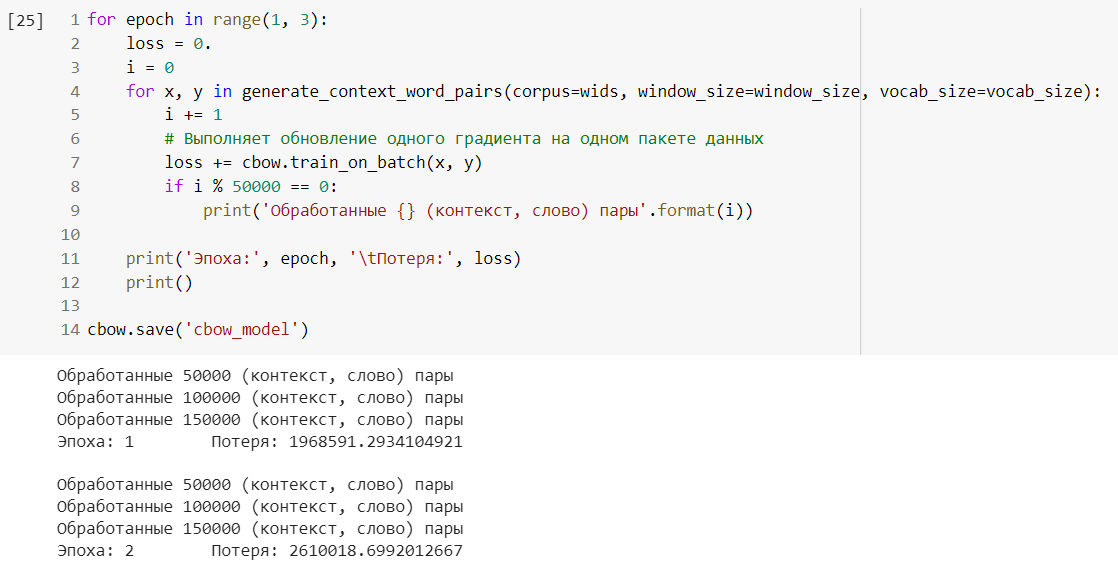
Суть этой архитектуры следующий: у нас есть **входные контекстные слова** размеров **(2 x window\_size)**, мы передадим их в **слой вложения** размера **(vocab\_size x embed\_size)**, который даст нам **плотные вложения слов** для каждого из этих контекстных слов **(1 x embed\_size for each word)**. Затем мы используем **лямбда-слой**, чтобы усреднить эти вложения и получить **среднее плотное вложение** **(1 x embed\_size)**, которое отправляется на **плотный слой softmax,** который выводит наиболее вероятное целевое слово.

Мы сравниваем это с фактическим целевым словом, вычисляем потери, распространяем ошибки обратно, чтобы скорректировать веса (в слое внедрения) и повторяем этот процесс для всех пар **(контекст, цель)** для нескольких эпох.

Теперь мы готовы обучить эту модель в нашем корпусе, используя наш генератор данных для ввода пар **(контекст, целевое\_слово)**.

# ****Обучение модели****

Запуск модели на нашем полном корпусе занимает довольно много времени, поэтому я просто запустила ее на 2 эпохи.



train\_on\_batch – запускает одно обновление градиента для одного пакета данных

Градиентный спуск – это алгоритм оптимизации, используемый для разнообразного обучения модели машинного обучения. Он лучше всего подходит для задач, в которых имеется большое количество функций и слишком много образцов, чтобы поместиться в памяти модели машинного обучения.

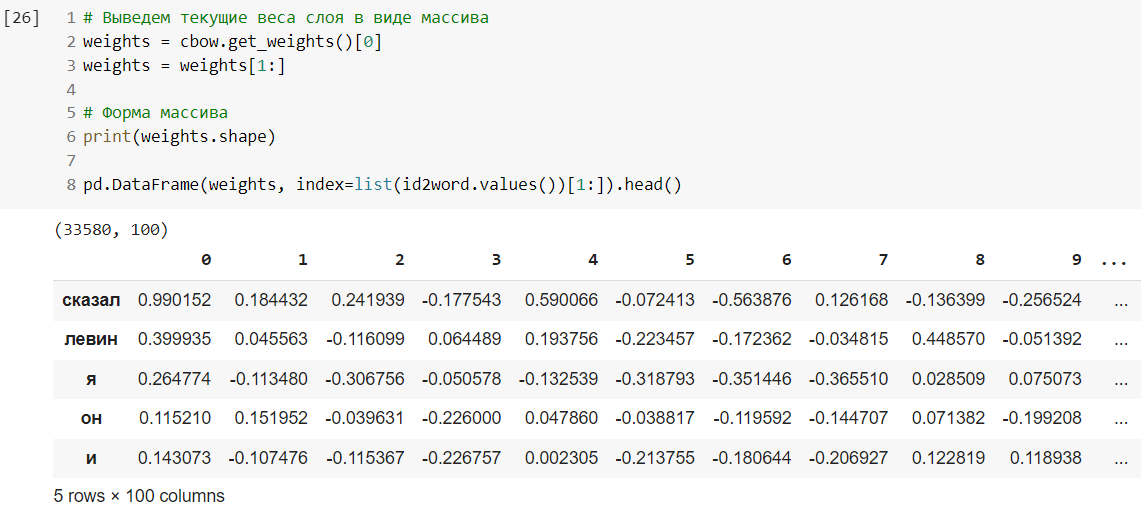
***Примечание.*** *Запуск этой модели требует больших вычислительных ресурсов и работает лучше при обучении с использованием графического процессора. Я тренировала всего лишь 2 эпохи целых 4 часа.*

После того, как эта модель обучена, похожие слова должны иметь одинаковые веса на основе слоя встраивания, и мы можем это проверить.

# ****Получение вложения слов****

Чтобы получить вложения слов для всего нашего словаря, мы можем извлечь их из нашего слоя встраивания, используя следующий код.

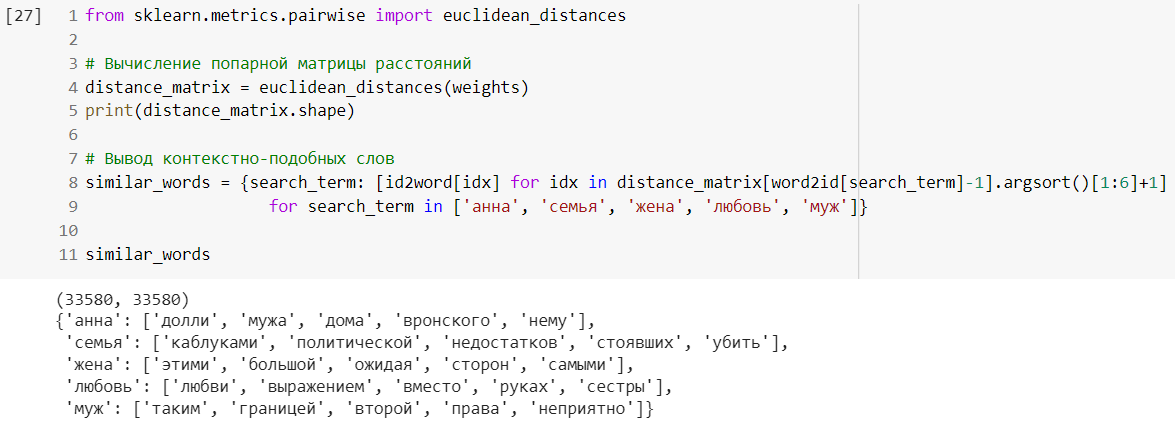
Мы не берем вложение в позиции 0, так как оно принадлежит (PAD) термину-заполнителю, который на самом деле не представляет интереса.



Таким образом, вы можете ясно видеть, что каждое слово имеет плотное вложение размера **(1x100)**, как показано в предыдущем выводе.

Давайте попробуем найти некоторые контекстуально похожие слова для конкретных интересующих слов на основе этих вложений.

Для этого мы строим матрицу попарных расстояний среди всех слов в нашем словаре на основе плотных векторов вложения, а затем находим n ближайших соседей каждого интересующего слова на основе кратчайшего расстояния.

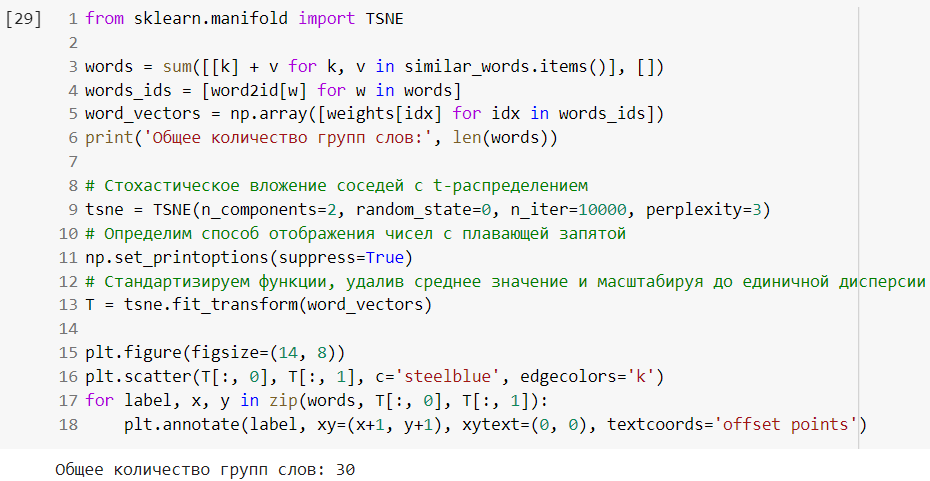


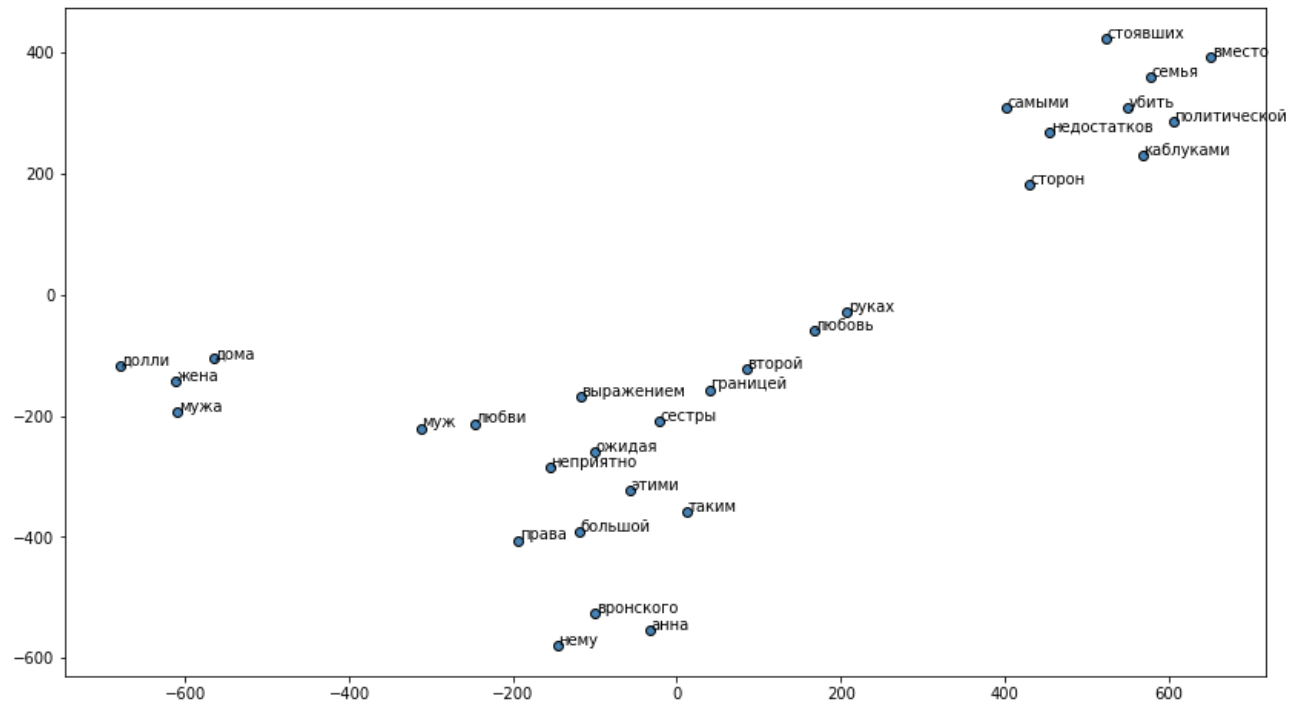
Вы можете ясно видеть, зная, о чем произведение, что некоторые пары имеют интересный контекстуальный смысл **(анна, долли)**, **(муж, второй)** и так далее, а некоторые могут и не иметь смысла.

Обучение большему количеству эпох обычно приводит к лучшим результатам.

# Визуализация результатов

Теперь давайте визуализируем вложения этих слов, используя **t-SNE**, который обозначает Стохастическое вложение соседей с t-распределением, внедряющегопопулярную технику уменьшения размерности для визуализации пространств более высоких измерений в более низких измерениях





На данном графике можно заметить группы, которые показывают разные слова контекстуального сходства, но расположенные рядом друг с другом в векторном пространстве.

# Надежные модели Word2Vec от Gensim

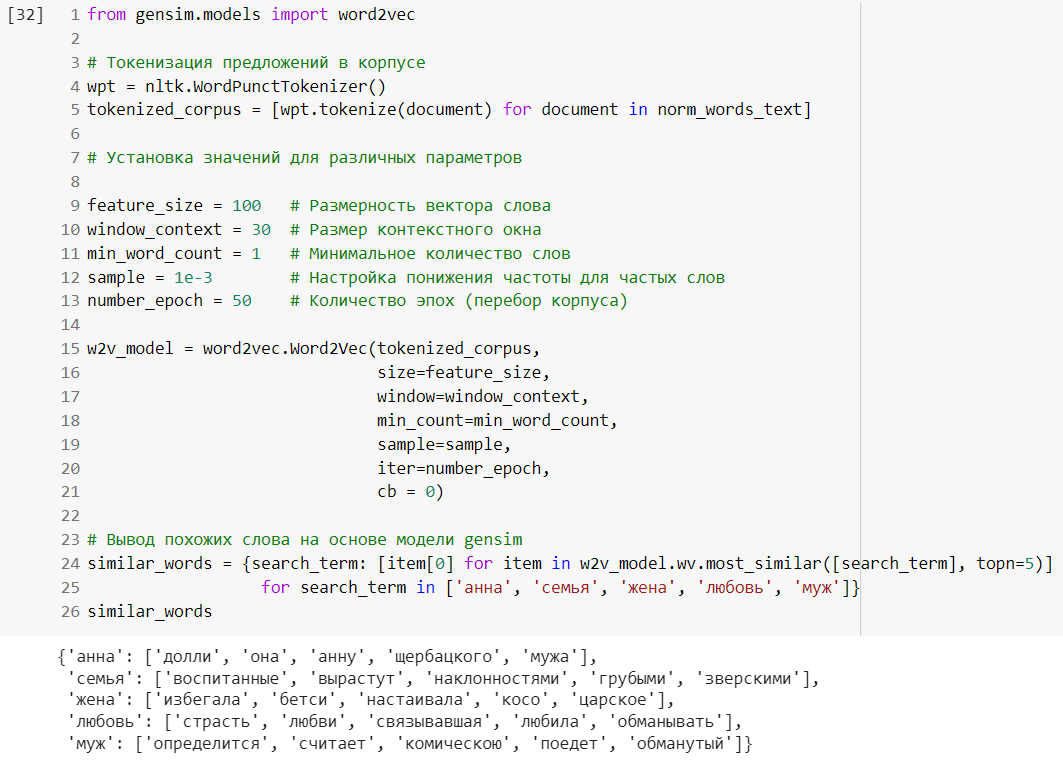
Хотя наши реализации достаточно приличные, они недостаточно оптимизированы для работы с большими корпусами.

Фреймворк gensim состоит из надежной, эффективной и масштабируемой реализации модели Word2Vec. Мы будем использовать тот же самый корпус «Анна Каренина».

Сосредоточимся на следующих четырех параметрах модели Word2Vec, чтобы построить его.

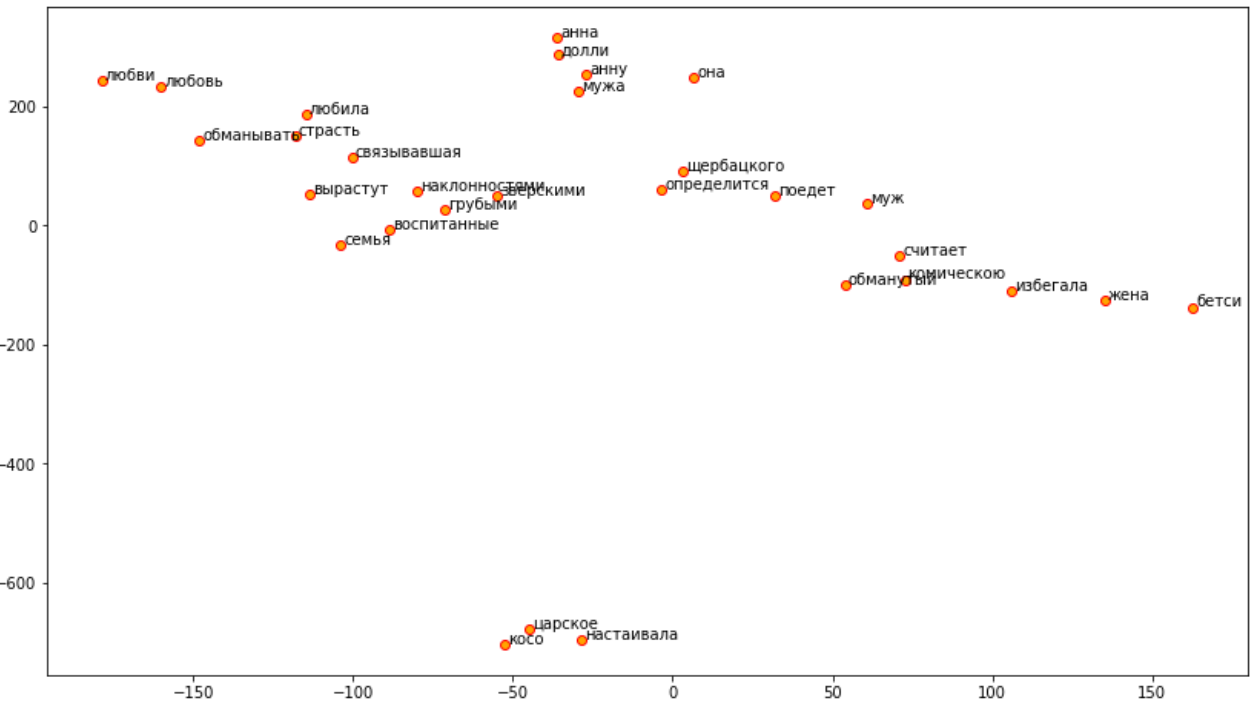
* **size:** Слово, включающее размерность
* **window:** размер окна контекста
* **min\_count:** минимальное количество слов
* **sample:** настройка понижения частоты для частых слов.
* **iter:** количество эпох
* **cb:** выбранная модель

После построения нашей модели мы будем использовать интересующие нас слова, чтобы увидеть наиболее похожие слова для каждого из них.



Подобные слова здесь определенно больше связаны с интересующими нас словами, и это ожидается, учитывая, что мы запускали эту модель для большего количества итераций, которые должны были давать более качественные и более контекстуальные вложения.

Давайте также визуализируем интересующие слова и похожие на них слова, используя их векторы встраивания после уменьшения их размеров до двумерного пространства с помощью t-SNE.



# Применение функций Word2Vec для обработки задач естественного языка

# Инициализация и обработка исходного текста

# C:\Users\1111\Desktop\Новая папка\17.png

# C:\Users\1111\Desktop\Новая папка\18.png

Определим векторизованную функцию, которая принимает вложенную последовательность объектов в качестве входных данных и возвращает один массив или кортеж массивов

# C:\Users\1111\Desktop\Новая папка\19.png

# Векторизация предложений

# Одной из стратегий, позволяющей векторизовать предложения, является усреднение вложений слов для каждого слова в этом предложении.

# C:\Users\1111\Desktop\Новая папка\21.png

# Это чрезвычайно полезная стратегия. Применим это к нашему корпусу, перед этим загрузив нашу обученную модель.

# C:\Users\1111\Desktop\Новая папка\20.png

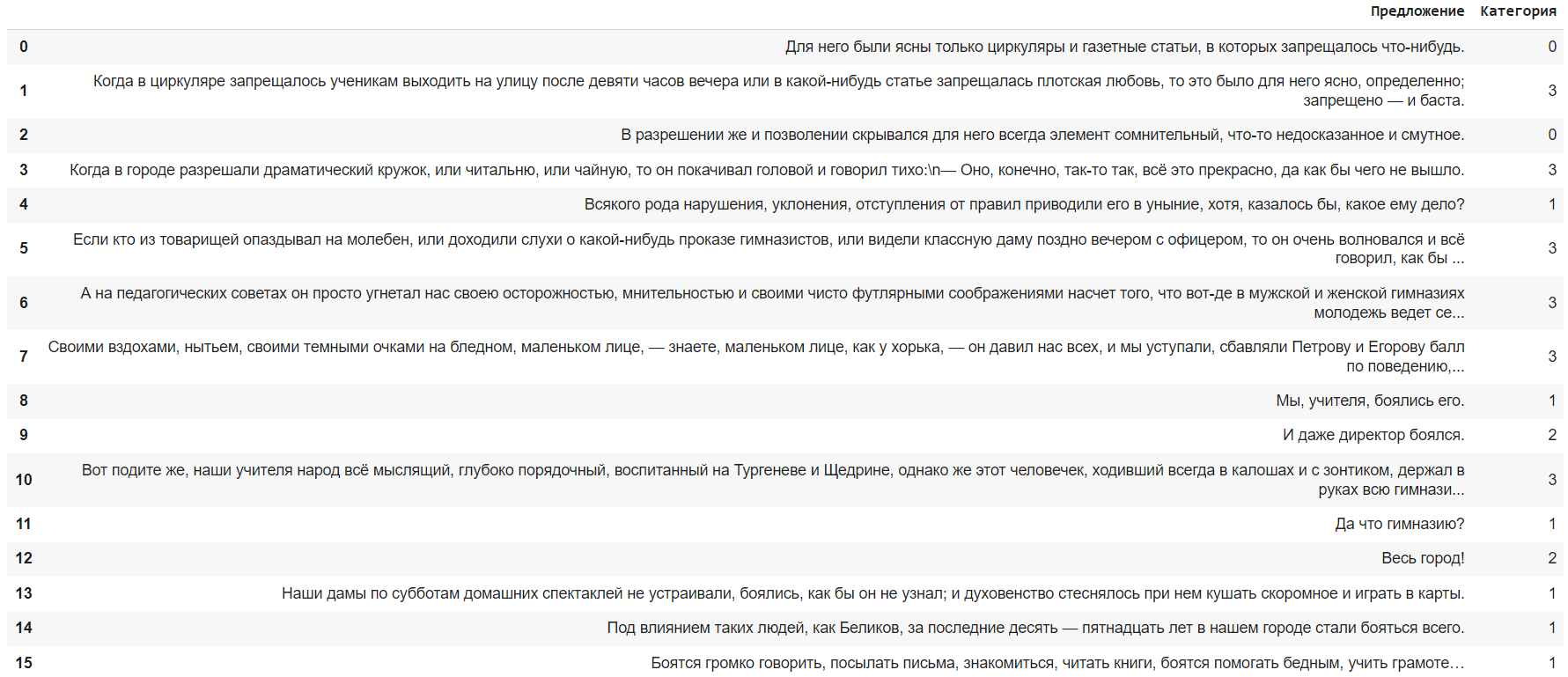
# C:\Users\1111\Desktop\Новая папка\28.png

# C:\Users\1111\Desktop\Новая папка\23.png

# Группировка предложений

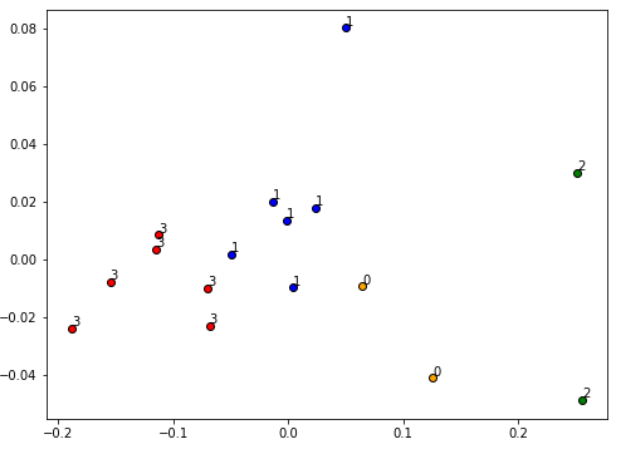
# Теперь, когда у нас есть функции для каждого документа, давайте сгруппируем наши предложения с помощью алгоритма [Affinity Propagation](https://en.wikipedia.org/wiki/Affinity_propagation), который представляет собой алгоритм кластеризации, основанный на концепции «передачи сообщений» между точками данных.

# C:\Users\1111\Desktop\Новая папка\24.png



Мы также можем визуализировать расположение каждого предложения в каждом кластере, используя **анализ основных компонентов (PCA)** , чтобы уменьшить размеры объекта до 2-D, а затем визуализировать то же самое (путем цветового кодирования каждого кластера).



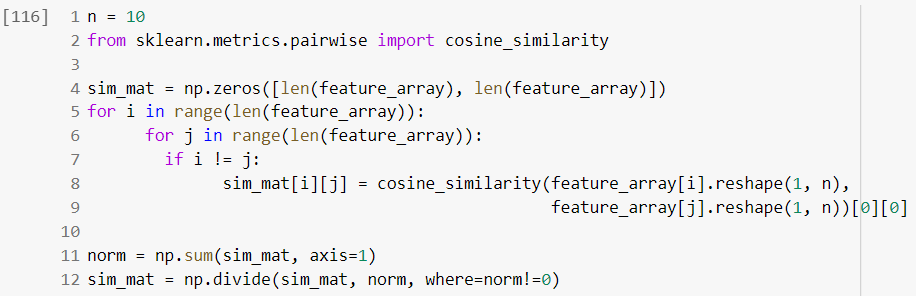


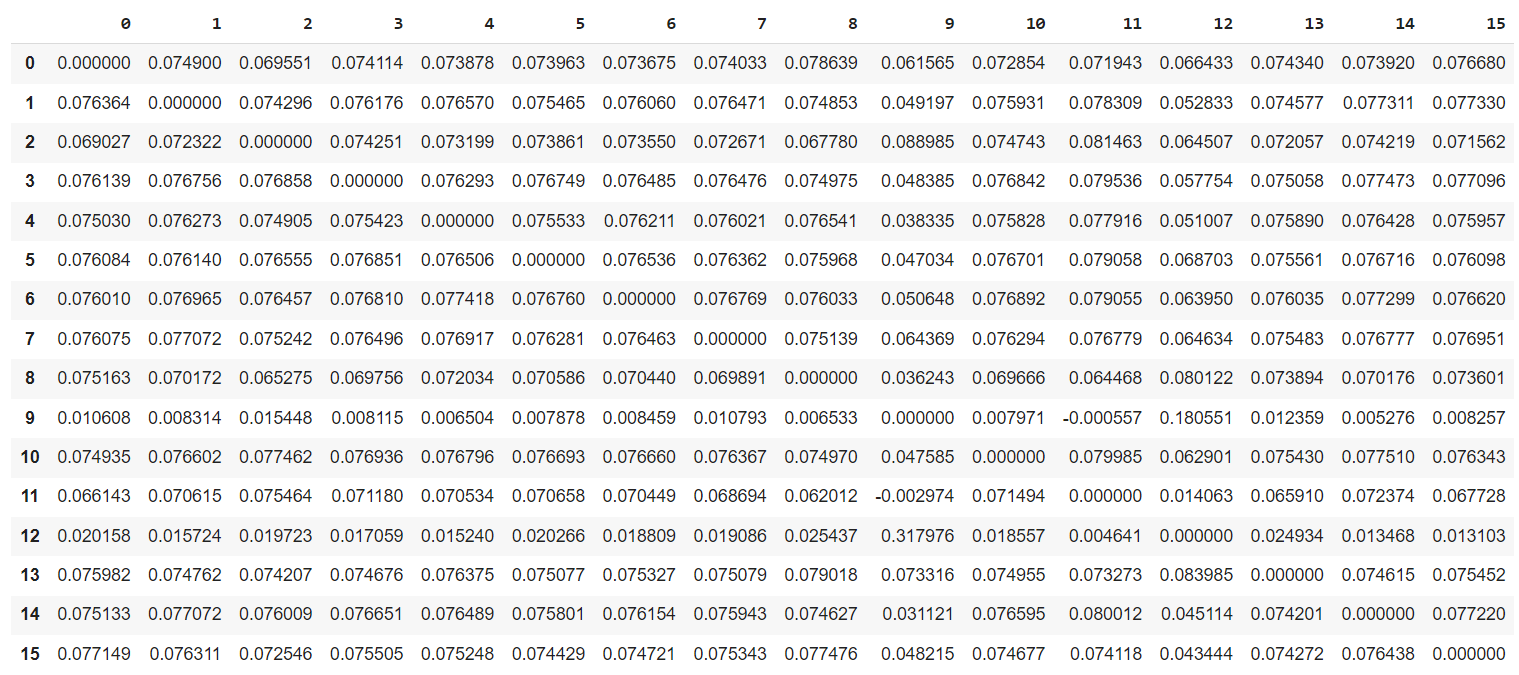
Вроде бы все в порядке, поскольку предложения в каждом кластере расположены ближе друг к другу и далеко друг от друга.

# Непосредственно реферирование

# Создание матрицы подобия косинусов

Сходство косинуса измеряет сходство между списками векторов путем вычисления угла косинуса между двумя списками векторов.





Каждый кластер вложений предложений можно рассматривать как группу идентичных предложений, которые более или менее несут одинаковую информацию.

Их значение можно представить одним предложением из кластера. Для этого используем PageRank из пакета NetworkX.

