|  |  |
| --- | --- |
| **Изображение выглядит как текст  Автоматически созданное описание** | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |
| Кафедра прикладной математики  **КУРСОВАЯ РАБОТА**  по дисциплине  Языки программирования для статистической обработки данных  **Тема курсовой работы**: Методы skip-gram на языке R   |  |  | | --- | --- | | Студент группы ИМБО-01-20 | Тихомиров Максим Алексеевич | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |  |  | | Руководитель курсовой работы | Митина Ольга Алексеевна | |  |  | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) | |  |  | | Работа представлена к защите | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. | |  |  | | Допущен к защите | « » \_\_\_\_\_\_\_\_\_\_\_\_2021 г. |   Москва 2021 | |

|  |  |
| --- | --- |
| **Изображение выглядит как текст  Автоматически созданное описание** | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |
| Институт информационных технологий | |

Кафедра прикладной математики

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

Языки программирования для статистической обработки данных

|  |  |
| --- | --- |
| Студент Тихомиров Максим Алексеевич | Группа ИМБО-01-20 |

**Тема работы:** Методы skip-gram на языке R

**Исходные данные:** среда разработки на языке R, стандарты языка R, методическое пособие, техническая документация.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Создание модели скип-грамма.
2. Масштабируемость обучаемости модели.

|  |  |  |  |
| --- | --- | --- | --- |
| **Срок представления к защите курсовой работы:** | | до «\_\_\_» \_\_\_\_\_\_\_\_\_\_2022 г. | |
|  | |  | |
| **Задание на курсовую работу выдал** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Митина О.А.) |
|  | | «\_\_\_» \_\_\_\_\_\_\_\_\_\_2022 г. | |
| **Задание на курсовую работу получил** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | (Тихомиров М.А.) |

Москва 2021

СОДЕРЖАНИЕ

[Введение 4](#_Toc73212128)

1 [Теоретическая часть 5](#_Toc73212129)

1.1 [Основные понятия и определения 5](#_Toc73212130)

1.2 Б[иблиотеки для реализации 11](#_Toc73212131)

2 [Практическая часть 12](#_Toc73212132)

2.1 [Реализация скип-грамма 12](#_Toc73212133)

2.2 [Визуализация 12](#_Toc73212133)

[Заключение 18](#_Toc73212135)

[Список используемой литературы 19](#_Toc73212136)

[Теоретическая часть 1](#_Toc73212133)9

Практическая часть [20](#_Toc73212133)

[Приложения 21](#_Toc73212137)

Приложение А [2](#_Toc73212133)1

# Введение

В последнее время в нашу жизнь вошло такое понятие как нейронные сети. Многие новостные программы, различные СМИ говорят об этом, заявляя стремительное развитие в этой сфере. Итак, что же такое нейронная сеть? В широком смысле – это одно из направлений научных исследований в области создания искусственного интеллекта. В узком же – совокупность нейронов(процессоров), соединенных и взаимодействующих друг с другом, главной целью которых является моделирование процессов головного мозга. Классифицировать нейроны можно по разным признакам, но обычно их классифицируют по характеру связей, самые широко используемые из них-это сети прямого распространения, рекуррентные и свёрточные. Касаемо применения нейросетей, то самыми распространенными задачами являются классификация, предсказание и распознавание:

В данной курсовой работе будет рассмотрена классификация текста. Такими задачами могут являться: соотнесение текстов темам, оценка отзывов на каком-либо информационном портале, определение рейтинга по тексту статьи и т.д.

Поэтому данная тема курсовой работы является актуальной для сферы продаж товаров и услуг.

Цель курсовой работы – построить и обучить нейросеть находить контекстные слова.

Задачи, решаемые в данной курсовой работе:

* определение необходимых библиотек;
* изучение технической документации используемых библиотек;
* поиск подходящих для обучения данных;
* использование знаний разработки с использованием языка программирования R;
* практика качественного оформления документации.

# Теоретическая часть

## Основные понятия и определения

Все тексты на естественном языке имеют большое количество слов, которые не несут информации о данном тексте. К примеру, в английском языке такими словами являются артикли, в русском к ним можно отнести предлоги, союзы, частицы. Данные слова называют шумовыми или стоп-словами. Для достижения лучшего качества классификации на первом этапе предобработки текстов обычно необходимо удалять такие слова. Второй этап предобработки текстов — приведение каждого слова к основе, одинаковой для всех его грамматических форм. Это необходимо, так как слова несущие один и тот же смысл могут быть записаны в разной форме. Например, одно и то же слово может встретиться в разных склонениях, иметь различные приставки и окончания.

Мы будем использовать набор данных Amazon Fine Foods Reviews . Этот набор данных состоит из обзоров изысканных продуктов с Amazon. Данные охватывают период более 10 лет, включая все ~500 000 обзоров до октября 2012 года. Обзоры включают информацию о продуктах и ​​пользователях, рейтинги и описательный текст.

Word embedding — это метод, используемый для сопоставления слов словаря с плотными векторами действительных чисел, где семантически похожие слова сопоставляются с соседними точками. Представление слов в этом векторном пространстве помогает алгоритмам повысить производительность в задачах обработки естественного языка, таких как синтаксический анализ и анализ тональности, путем группировки похожих слов. Например, мы ожидаем, что в пространстве вложения «кошки» и «собаки» отображаются в близлежащие точки, поскольку они оба являются животными, млекопитающими, домашними животными и т. д.

Концепция вложений (embeddings) — одна из самых замечательных идей в машинном обучении. Если вы когда-нибудь использовали Siri, Google Assistant, Alexa, Google Translate или даже клавиатуру смартфона с предсказанием следующего слова, то уже работали с моделью обработки естественного языка на основе вложений. За последние десятилетия произошло значительное развитие этой концепции для нейронных моделей (последние разработки включают контекстуализированные вложения слов в передовых моделях, таких как BERT и GPT2).

Word2vec — метод эффективного создания вложений, разработанный в 2013 году. Кроме работы со словами, некоторые его концепции оказались эффективны в разработке рекомендательных механизмов и придании смысла данным даже в коммерческих, неязыковых задачах. Эту технологию применили в своих движках рекомендаций такие компании, как Airbnb, Alibaba, Spotify и Anghami.

Модель Skip-gram — разновидность word2vec, класса вычислительно эффективных прогностических моделей для изучения word embedding из необработанного текста. Skip-gram лучше работает на больших наборах данных.

Целевое слово (Target Word): слово, которое предсказывается.

Контекстное слово (Context Word): каждое слово в предложении, кроме целевого слова.

Размерность вложения (Embedding Dimension): количество измерений пространства векторов, в которые мы хотим преобразовать слова.

Разберемся на примере:

Предложение: “The quick brown fox jumps over the lazy dog”(Перевод: “Шустрая бурая лиса прыгает через ленивого пса.”).

Допустим, мы пытаемся предсказать слово “fox”, тогда “fox” будет нашим целевым словом.

Остальные слова будут контекстными.

Входное слово унитарно кодируется (One-Hot) и отправляется в модель одно за другим. Скрытый слой пытается предсказать наиболее вероятное слово, основываясь на весах, собранных в слое.

Мы избрали размерность нашего вектора равной 300, поэтому каждое наше слово будет преобразовано в 300-мерный вектор.

Последний слой нашей модели w2v — это классификатор Softmax, который определяет наиболее вероятное выходное слово.

Входными данными сети является one-hot вектор, представляющий входное слово, и метка, которая также является one-hot вектором, представляющим целевое слово, однако выходные данные сети представляют собой вероятностное распределение целевых слов, но не обязательно one-hot вектор, как метки.

Строки матрицы весов скрытого слоя на самом деле являются векторами слов (встраивание слов)!

Скрытый слой работает как справочная таблица. Выходные данные скрытого слоя — это «вектор слов» для входного слова.

Количество нейронов на скрытом слое должно равняться размерности вложений, которую мы задаем.

Допустим, в нашем корпусе 5 слов:

Предложение 1: Have a Good Day

Предложение 2: Have a Great Day

Сколько уникальных слов в этих примерах? “have”, “a”, “good”, “great”, “day” = 5

Теперь предположим, что наше целевое слово для предложения 1 — “good”.

Когда я скармливаю остальную часть контекстных слов своей модели, она должна понять, что “good” - это именно то слово, которое она должна предсказать для предложения 1.

Предположим, что для предложения 2 наше целевое слово — “great”.

Когда я скармливаю своей модели остальную часть контекстных слов, она должна понять, что для предложения 2 корректное слово для предсказания — “great”.

Если вы внимательно посмотрите, то увидите, что когда мы уберем целевые слова, останутся одинаковые для двух предложений контекстные слова. Так каким образом модель точно предскажет, какое из них подойдет?

На самом деле это не имеет значения, потому что она предсказывает наиболее вероятное слово, и в нашем случае нет неправильного варианта — подойдут оба “good” и “great”. Таким образом, возвращаясь к фиксации синонимов, тут оба слова “good” и “great” будут по сути означать одно и тоже и являться очень вероятным выходным словом для этих конкретных контекстных слов. Именно так наша модель изучает семантическое значение слов.

Говоря простыми словами, если я могу использовать слова взаимозаменяемо, тогда изменяется только мое целевое слово, а не контекстные слова, т.е. любые два похожих слова, имеющих один синоним, будут весьма вероятным результатом для контекстных слов, скормленных модели — они будут иметь почти одинаковые числовые значения. Если у них почти одинаковые числовые значения, их векторы должны располагаться близко согласно их метрикам сходства. Таким образом мы фиксируем семантические значения.

Входными данными этих моделей является не что иное, как большой one-hot encoded вектор.

Давайте разберем на примерах:

Предположим, что в нашем текстовом корпусе 10000 уникальных слов, и мы хотим, чтобы наша размерность вложения соответствовала 300-мерному вектору.

Таким образом, входными данными будет вектор с размерностью 10000 с 0 для всех других слов, в то время как контекстное слово будет 1. Размерность нашего входного вектора составляет 1 x 10000.

Скрытый слой — это та самая матрица весов со всеми словами, которые в начале случайным образом преобразованы в по 300 измерениям. Затем мы обучаем нашу модель подбирать значения этих весов, чтобы на каждом этапе обучения модель лучше справлялась с предсказанием.

Выходной слой — это не что иное, как слой softmax, который использует значения вероятности для прогнозирования результатов среди 10000 векторов, которые у нас есть. Размер нашего выходного вектора равен 1 x 300.

Теперь давайте попробуем понять базовую математику, которая за этим стоит

Входные данные: 1 x 10000

Скрытый слой: 10000 x 300

Выходные данные = Softmax (Входные данные x Скрытый слой) (матричное умножение) = 1 x n \* n x dim = 1 x dim = 1 x 300

Выше я объяснил архитектуру CBOW, а чтобы понять Skip-gram, нам просто нужно изменить порядок ввода и вместо контекстного слова и определения целевых слов, вводить целевые слова и пытаться определить контекстные слова. CBOW — это несколько способов ввода и только один вывод, Skip-gram — это один ввод и несколько выводов.

В Skip-gram мы можем определять количество контекстных слов, которое по нашему желанию бы предсказывалось, и, соответственно, мы можем создать пары входных-выходных значений.

Например: ‘have a good day’

Если мы установим размер окна равным 2, то обучающие пары будут следующими:

Целевое слово: good

Контекстные слова: have, a, day

(good, have)

(good, a)

(good, day)

Таким образом, модель учится и пытается предсказать слова, близкие к целевому слову.

Сейчас синтаксическое и семантическое значения хорошо отражены в векторах слов, поэтому они могут идеально использоваться в качестве входных данных для любых других сложных программ NLP, где может появиться необходимость в понимании сложных разговоров людей или контекста языков.

W2V сама по себе является нейронной сетью, но она представляет из себя скорее вспомогательную функцию, которая облегчает работу другим NLP приложениям.

Обработка естественного языка Natural language processing (NLP) – это сфера искусственного интеллекта, которая занимается применением алгоритмов машинного обучения и лингвистики для анализа текстовых данных. Цель NLP – понимание и воспроизведение естественного человеческого языка.

Стохастическое вложение соседей с t-распределением (англ. t-Distributed Stochastic Neighbor Embedding, t-SNE) — метод визуализации данных высокой размерности с помощью представления каждой точки данных в двух или трехмерном пространстве, являющийся модификацией метода стохастического вложения соседей.

Косинусное сходство — это мера сходства между двумя векторами предгильбертового пространства, которая используется для измерения косинуса угла между ними.

Если даны два вектора признаков, A и B, то косинусное сходство, cos(θ), может быть представлено используя скалярное произведение и норму: (1)

Линейный классификатор — способ решения задач классификации, когда решение принимается на основании линейного оператора над входными данными. Класс задач, которые можно решать с помощью линейных классификаторов, обладают, соответственно, свойством линейной сепарабельности.

Пусть вектор {\displaystyle {\vec {x}}}{\vec x} из действительных чисел представляет собой входные данные, а на выходе классификатора вычисляется показатель y по формуле:

(2)

## Библиотеки для реализации

* readr — это пакет, который позволяет пользователю загружать табличные данные из самых разных источников. Например, csv или текстовые файлы. У данного пакета есть ряд неоспоримых преимуществ, которые выгодно выделяют его среди конкурентов (да, в языке R есть еще большое количество сторонних пакетов для такого же рода задач).
* stringr — пакет для работы со строками, который понадобится нам для очистки и подготовки данных.
* keras — открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями.
* reticulate — встраивает сеанс Python в наш сеанс R, обеспечивая бесшовную высокопроизводительную совместимость.
* purrr — позволяет работать со списками.
* dplyr — библиотека для удобной работы с базами данных.
* text2vec — предоставляет эффективную структуру с кратким API для анализа текста и обработки естественного языка (NLP).

# Практическая часть

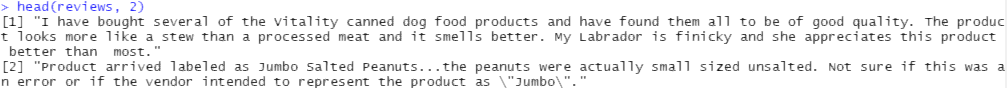
## Реализация скип-грамма

На первом этапе нам потребуется загрузить данные из файла, прочитать их и обработать



Рисунок 1 Загрузка данных из файла finewoods.txt.gz

Убедимся в корректности и читаемости данных

 Рисунок 2 Вывод первых двух рецензий из обработанного файла

Мы начнем с предварительной обработки текста с помощью keras text\_tokenizer(). Токенизатор будет отвечать за преобразование каждого отзыва в последовательность целочисленных токенов (которые впоследствии будут использоваться в качестве входных данных для модели skip-gram).



Рисунок 3 Инициализация tokenizer

Обратите внимание, что tokenizer изменяется на месте вызовом fit\_text\_tokenizer(). Каждому из 20 000 наиболее распространенных слов будет присвоен целочисленный токен (остальные слова будут присвоены токену 0).

В модели skip-gram мы будем использовать каждое слово в качестве входных данных для логарифмически-линейного классификатора с проекционным слоем, а затем предсказывать слова в определенном диапазоне до и после этого слова. Вывод распределения вероятностей по всему словарю для каждого целевого слова, которое мы вводим в модель, требует больших вычислительных ресурсов. Вместо этого мы собираемся использовать отрицательную выборку, что означает, что мы будем выбирать некоторые слова, которые не появляются в контексте, и обучать двоичный классификатор предсказывать, действительно ли переданное нами слово контекста взято из контекста или нет.

С более практической точки зрения, для модели пропуска граммы мы введем 1d целочисленный вектор токенов целевого слова и 1d целочисленный вектор выборочных токенов слова контекста. Мы сгенерируем прогноз 1, если выбранное слово действительно появилось в контексте, и 0, если нет.

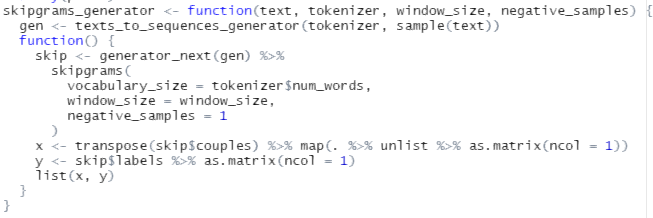


Рисунок 4 Определение функции генератора скип-граммов

Генераторная функция — это функция, которая при каждом вызове возвращает разное значение (генераторные функции часто используются для предоставления потоковых или динамических данных для обучающих моделей). Наша функция генератора получит вектор текстов, токенизатор и аргументы для скип-грамма (размер окна вокруг каждого исследуемого целевого слова и количество отрицательных образцов, которые мы хотим отобрать для каждого целевого слова).

Теперь давайте начнем определять модель keras. Мы будем использовать функциональный API Keras .



Рисунок 5 Определение модели keras

Сначала мы напишем заполнители для входных данных, используя layer\_input функцию.



Рисунок 6 Заполнение входных данных

Теперь давайте определим матрицу вложения (word embedding). Вложение представляет собой матрицу с размерами (словарь, размер\_встраивания), которая действует как таблица поиска для векторов слов.

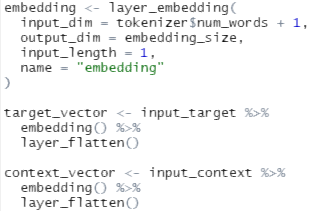


Рисунок 7 Определение матрицы вложения

Следующим шагом является определение того, как target\_vector будет связан с context\_vector, чтобы наша сеть выдавала 1, когда контекстное слово действительно появилось в контексте, и 0 в противном случае. Мы хотим чтобы target\_vector был похожим на context\_vector если бы они появились в том же контексте. Типичной мерой сходства является косинусное сходство для двух векторов A и B косинусное сходство определяется евклидовым скалярным произведением A и B по их величине. Поскольку нам не нужно нормализовать сходство внутри сети, мы вычислим только скалярное произведение, а затем выведем плотный слой с сигмовидной активацией.



Рисунок 8 Скалярное произведение

Теперь мы создадим модель и скомпилируем ее.



Рисунок 9 Создание модели

Мы можем увидеть полное определение модели, вызвав summary:

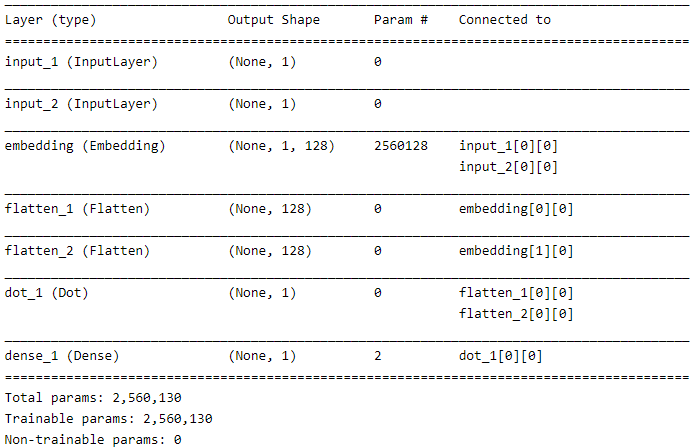
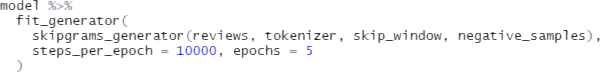


Рисунок 10 Параметры созданной модели

Перейдем к обучению модели. Мы приспособим модель с помощью fit\_generator() функции. Нам нужно указать количество шагов обучения, а также количество эпох, которые мы хотим обучить. Будем тренироваться на 100 000 шагов за 5 эпох. Обратим внимание, что мы также можем получить разумные результаты всего за одну эпоху обучения.

 Рисунок 11 Обучение модели

Оценим результаты обучения модели на графике, где ось абсцисс это номер эпохи, а ось ординат это посчитанная ошибка (где 1.0 — максимальная и 0 — минимальная соответственно).

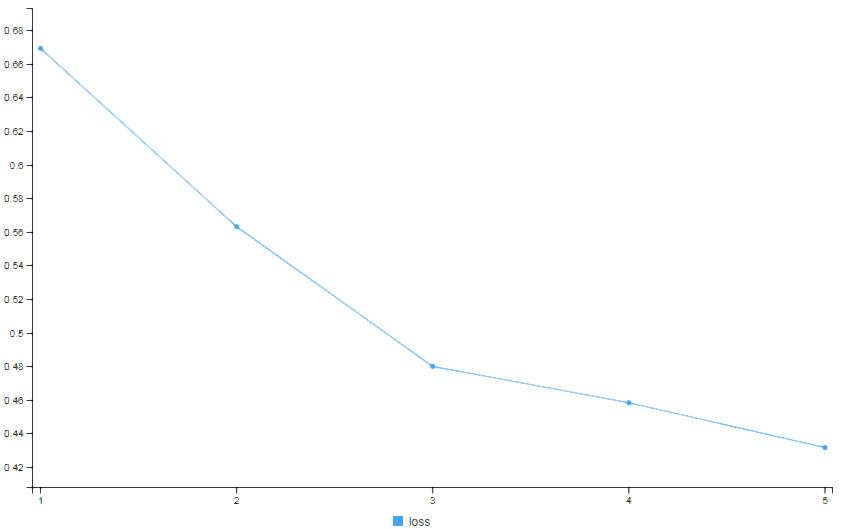


Рисунок 12 Результаты обучения модели

Теперь мы можем извлечь матрицу вложений(embeddings) из модели с помощью get\_weights() функции. Мы также добавили row.names в нашу матрицу встраивания, чтобы мы могли легко найти, где находится каждое слово.

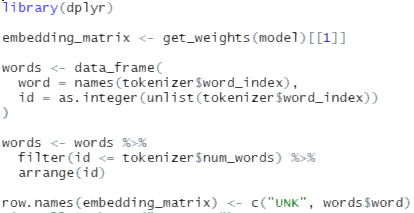


Рисунок 13 Извлечение матрицы вложений

Теперь мы можем найти слова, которые близки друг к другу во вложении. Мы будем использовать косинусное подобие, поскольку именно его мы обучили модели минимизировать.

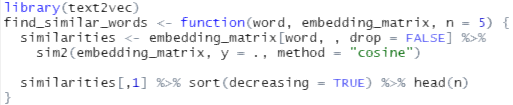


Рисунок 14 Определение функции поиска похожих слов

Попробуем найти слова, близкие друг к другу относительно слова thing:



Рисунок 15 Результат поиска схожих слов для thing

# 

# Визуализация

Для визуализации нам понадобится еще не сколько библиотек



Для визуализации мы будем использовать алгоритм t-SNE и будем визуализировать вложения слов (words embeddings). Для более читаемой иллюстрации мы будем использовать его только с первыми 500 словами.

Рисунок 15 Импорт библиотек для визуализации

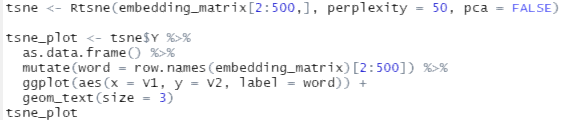


Рисунок 16 Построение t-SNE модели

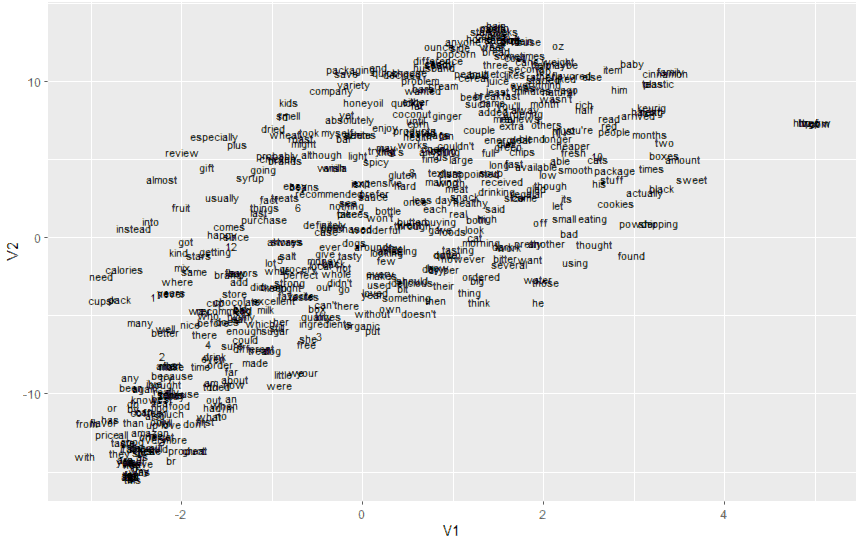


Рисунок 16 Изображение t-SNE модели

Приблизив одно из скоплений слов, мы можем увидеть что слова months, years, days, minutes сгруппированы очень близко. Нейросеть определила их все как сильно связанные по смыслу слова.

Это хороший показатель того, что нейросеть работает так, как задуманно.

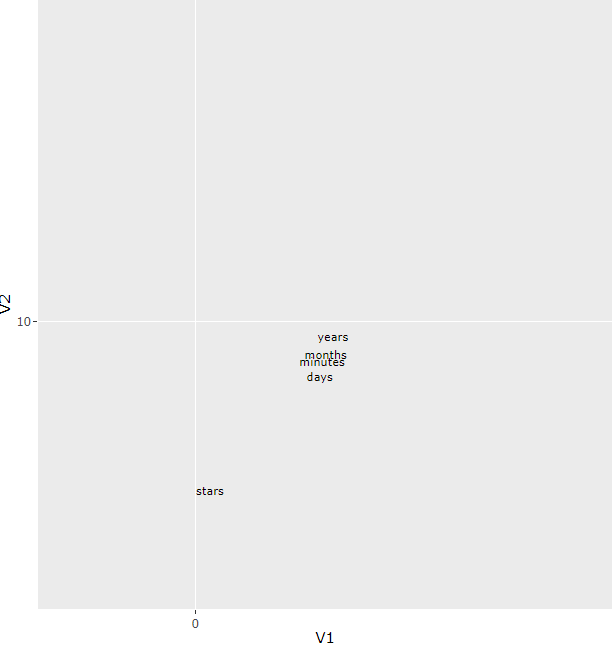


Рисунок 16 фрагмент t-SNE модели

# Заключение

Word2Vec - очень удобная технология работы с текстовыми данными. Она позволяет перевести слова в векторное пространство, сохраняя приэтом их семантические свойства, то есть смысловую и контекстную близость слов. А имея векторное представление слов, можно каким-ли образом сводить данные к стандартному формату обучающей выборки и применять разные методы машинного обучения. В курсовой работе были выполнены задачи классификации слов, а их векторное представление позволяет использовать такие методы как косинусное сходство, а также использовать большее количество ресурсов за счет подключения CUDA ядер.

Также в курсовой работе были рассмотрены основные этапы обработки естественного языка. В результате были освоены процессы предварительной обработки текста, его векторизации и классификации, изучены архитектура и принцип работы нейронных сетей.

Цель — построить и обучить нейросеть находить контекстные слова — достигнута.

В ходе данной курсовой работы выполнены следующие задачи:

* определены необходимые библиотеки;
* изучена техническая документация используемых библиотек;
* найдены подходящие для обучения данных;
* использованы знания разработки с использованием языка программирования R;
* отработано качественное оформление документации.

# Список используемой литературы

# Теоретическая часть

1 Raschka S. Python Machine Learning. Birmingham: Packt Publishing, 2015.

2 Jurafsky D., Martin J. Speech and Language Processing. New Jersey: Prentice Hall, 2000.

3 Daelemans W, van den Bosch A. Memory-Based Language Processing. Cambridge: Cambridge University Press, 2005.

4 Liddy E. Natural Language Processing. New York: Pergamon Press, 2001.

5 Hastie T. The Elements of Statistical Learning: Data Mining, Inference, and Prediction.

Stanford: Springer, 2007.

6 Anderson T.W. An Introduction to Multivariate Statistical Analysis. Boca Raton: CRC Press, 2009.

7 Anderson T.W. Greedy Function Approximation: A Gradient Boosting Machine. Stanford: John Wiley & Sons, 2003.

8 Geron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol: O’Reilly, 2017.

9 Müller A.C., Guido S. Introduction to Machine Learning with Python. New York: O’Reilly, 2016.

10 James Pustejovsky J., Stubbs A. Natural Language Annotation for Machine Learning. Boston: O’Reilly, 2012.

# Практическая часть

1. Наивный Байесовский классификатор в   25 строк кода URL - <https://habr.com/post/120194/> (04.12.2018)
2. Открытый курс машинного обучения. Тема 4. Линейные модели классификации  и  регрессии  URL   - <https://habr.com/company/ods/blog/323890/#2-logisticheskaya-regressiya> (04.12.2018)
3. Word2Vec: как работать с векторными представлениями слов URL - <https://neurohive.io/ru/osnovy-data-science/word2vec-vektornye-predstavlenija-slov-dlja-mashinnogo-obuchenija/> (04.12.2018)
4. Нейронные сети для начинающих. Часть 1 URL - <https://habr.com/post/312450/> (04.12.2018)
5. Нейронные сети для начинающих. Часть 2 URL - <https://habr.com/post/313216/> (04.12.2018)
6. Архитектура сверточной нейронной сети URL - <https://studfiles.net/preview/6871773/page:10/> (04.12.2018)
7. Классификация текстов с помощью сверточных нейронных сетей URL - <https://cyberleninka.ru/article/n/klassifikatsiya-tekstov-s-pomoschyu-svertochnyh-neyronnyh-setey> (04.12.2018)
8. NLTK 3.4 documentation URL -<http://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.punkt> (04.12.2018)
9. NLTK stop words URL - <https://pythonspot.com/nltk-stop-words/> (04.12.2018)
10. Морфологический анализатор pymorphy2 - <https://pymorphy2.readthedocs.io/en/latest/user/guide.html> (04.12.2018)
11. models.word2vec – Word2vec embeddings URL - <https://radimrehurek.com/gensim/models/word2vec> (04.12.2018)

# Приложение А

**Листинг main.R:**

# вводим основную и дополнительные библиотеки для считывания данных

library(readr)

library(stringr)

library(keras)

# считываем и обрабатываем данные

reviews <- read\_lines("finefoods.txt.gz")

reviews <- reviews[str\_sub(reviews, 1, 12) == "review/text:"]

reviews <- str\_sub(reviews, start = 14)

reviews <- iconv(reviews, to = "UTF-8")

# просматриваем полученный результат обработки данных

head(reviews, 2)

# заканчиваем предобработку данных

tokenizer <- text\_tokenizer(num\_words = 20000)

tokenizer %>% fit\_text\_tokenizer(reviews)

# подключаем библиотеки для работы с текстом

library(reticulate)

library(purrr)

# определяем генератор скип-граммов

skipgrams\_generator <- function(text, tokenizer, window\_size, negative\_samples) {

gen <- texts\_to\_sequences\_generator(tokenizer, sample(text))

function() {

skip <- generator\_next(gen) %>%

skipgrams(

vocabulary\_size = tokenizer$num\_words,

window\_size = window\_size,

negative\_samples = 1

)

x <- transpose(skip$couples) %>% map(. %>% unlist %>% as.matrix(ncol = 1))

y <- skip$labels %>% as.matrix(ncol = 1)

list(x, y)

}

}

# определяем модель keras

embedding\_size <- 128 # Dimension of the embedding vector.

skip\_window <- 5 # How many words to consider left and right.

num\_sampled <- 1 # Number of negative examples to sample for each word.

input\_target <- layer\_input(shape = 1)

input\_context <- layer\_input(shape = 1)

# определяем матрицу вложения word embedding

embedding <- layer\_embedding(

input\_dim = tokenizer$num\_words + 1,

output\_dim = embedding\_size,

input\_length = 1,

name = "embedding"

)

target\_vector <- input\_target %>%

embedding() %>%

layer\_flatten()

context\_vector <- input\_context %>%

embedding() %>%

layer\_flatten()

# вычисляем скалярное произведение

dot\_product <- layer\_dot(list(target\_vector, context\_vector), axes = 1)

output <- layer\_dense(dot\_product, units = 1, activation = "sigmoid")

# создаем и компилируем итоговую модель

model <- keras\_model(list(input\_target, input\_context), output)

model %>% compile(loss = "binary\_crossentropy", optimizer = "adam")

# смотрим на итоговую модель

summary(model)

#обучение модели

model %>%

fit\_generator(

skipgrams\_generator(reviews, tokenizer, skip\_window, negative\_samples),

steps\_per\_epoch = 100, epochs = 5

)

# извлекаем матрицу вложений word embedding и добавляем row.names

library(dplyr)

embedding\_matrix <- get\_weights(model)[[1]]

words <- data\_frame(

word = names(tokenizer$word\_index),

id = as.integer(unlist(tokenizer$word\_index))

)

words <- words %>%

filter(id <= tokenizer$num\_words) %>%

arrange(id)

row.names(embedding\_matrix) <- c("UNK", words$word)

# задаем метод для нахождения близких друг к другу по смыслу слов

library(text2vec)

find\_similar\_words <- function(word, embedding\_matrix, n = 5) {

similarities <- embedding\_matrix[word, , drop = FALSE] %>%

sim2(embedding\_matrix, y = ., method = "cosine")

similarities[,1] %>% sort(decreasing = TRUE) %>% head(n)

}

# ищем слова, похожие по смыслу на слово thing

find\_similar\_words("thing", embedding\_matrix)