Федеральное государственное образовательное бюджетное

учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ**

**ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(Финансовый университет)**

Факультет информационных технологий и анализа больших данных

**Департамент анализа данных и машинного обучения**

**Пояснительная записка к курсовой работе по дисциплине   
“Технологии анализа данных и машинного обучения”**

на тему:

«Предварительный анализ данных и построение признаков в задачах классификации текстов»

**Выполнила:**

Студентка группы ПИ19-1В

Пылева Татьяна Евгеньевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Научный руководитель:**

Попов Владимир Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Москва, 2022 г.**

**Оглавление**

[Введение 4](#_Toc102428497)

[1 Машинное обучение 6](#_Toc102428498)

[1.1 Python в машинном обучении. 6](#_Toc102428499)

[1.2 Текстовые данные в машинном обучении. 6](#_Toc102428500)

[2 Анализ текста с помощью языка Python. 8](#_Toc102428501)

[2.1 Набор текстовых данных для анализа 8](#_Toc102428502)

[2.2 Подготовка данных для обработки текста 8](#_Toc102428503)

[2.2.1 Импорт текстовых файлов. 8](#_Toc102428504)

[**2.2.2** **Предварительная обработка набора данных.** 9](#_Toc102428505)

[3 Анализ набора данных 12](#_Toc102428506)

[3.1 Модель суммирование слов 12](#_Toc102428507)

[3.2 Трансформирование слов в векторы 12](#_Toc102428508)

[3.3 Очистка важности слов при помощи проема tf-idf 14](#_Toc102428509)

[4 Очистка текстовых данных 17](#_Toc102428510)

[5 Переработка документов в лексемы 19](#_Toc102428511)

[6 Обучение логистической регрессионной модели для классификации документов 20](#_Toc102428512)

[7 Работа с более крупными данными – внешнее обучение 24](#_Toc102428513)

[8 Тематическое моделирование с помощью латентного размещения Дирихле 27](#_Toc102428514)

[9 Реализация LDA в библиотеке scikit-learn 28](#_Toc102428515)

[Заключение 31](#_Toc102428516)

[Литература 32](#_Toc102428517)

[Приложение 33](#_Toc102428518)

# Введение

Машинное обучение (МО) на данный момент можно уверенно назвать наукой, наукой об алгоритмах, которые понимают смысл вводимых и выводимых данных. Также в настоящем мы живем в изобилии возможностей Интернета и социальных сетей, в которых рецензии, отзывы, мнения и рекомендации стали значимыми в таких областях как: политология, судебная экспертиза и бизнес.

При помощи современных технологий сегодня каждый способен находить и анализировать большие данные более рациональным путем. В повседневной жизни одно из направлений, в котором используется машинное обучение, — это классификация текстов. Он включает в себя определенные методы по маркеровке текстов на естественном языке предопределенного набора данных с разбиением по категории. К примеру, первый яркий образ классификации – это сегментация новостных статей по темам, актуальности, годам, здесь может быть любой признак. Но на этом прогресс не останавливается, применив современные технологии искусственного интеллекта, мы можем получить простой процесс автоматизированной системы и минимальной ручной работой. Такая концепция применяется по всему миру, представлена перед лицом людей и наблюдают за ним каждый день, это обычная фильтрация писем в почте. На основании прикрепленных материалов, шаблонных слов сообщения (типичных признаков) обученная модель распределяет всю входящую информацию на действительно важную и нежелательную.

В данной курсовой работе будут рассмотрены общие понятия машинного обучения, несколько методов обучения, строительные блоки для успешного проектирования систем в будущем.

Целью работы является анализ данных и предварительная настройка признаков для будущего машинного обучения в задачах классификации текстов. Чтобы достичь желаемых результатов необходимо будет выполнить следующие пункты:

* Нужно провести очистку и подготовку определенных текстовых данных;
* Требуется построить векторы признаков из блоков текста;
* Необходимо обучить модели машинного обучения для типизации рецензий фильма на положительные и отрицательные.
* Отработать крупные наборы текстовых данных, применяя внешее обучение (out-of-core learning);
* Рассмотреть выведение жанров из совокупности набора данных для категоризации;
* Оценить результаты моделирования.

Объектом исследования являемся набор рецензий на фильмы, состоящий из 50000 отзывов критиков на фильмы.

К используемым методам относятся модель суммирования слов, оценка важности слов с помощью tf-idf, создание лексем, алгоритм стемминга, обучение логистической регрессионной модели, внешнее обучение. Для обработки данных и обучения моделей используется язык Python. Среда разработка рекомендуемая - Jupyter Notebook.

# Машинное обучение

## Python в машинном обучении.

Python это один из самых популярных языков программирования для машинного обучения и искусственного интеллекта. Он предлагает гибкие возможности и функции, обширные библиотеки, которые позволяют создавать высокопроизводительные системы обучения.

Наиболее популярными библиотеками являются:

NumPy – используется для обработки больших многомерных массивов и матриц.

Pandas – библиотека для анализа данных.

Scikit-learn – библиотека, которая содержит большинство из распространенных алгоритмов обучения.

NLTK (Natural Language Toolkit) – платформа для создания обработки естественного языка на Python

## Текстовые данные в машинном обучении.

Машинное обучение и анализ данных о тексте – это растущая область с огромным потенциалом в науке о данных. В повседневной жизни текстовые данные могут принимать множество форм. Неудивительно, что существует огромное количество сценариев, требующих обработку и анализ текстовой информации.

Здесь стоит отметить понятие единицы анализа письменного текста, чаще всего под ними используются лексемы и словоформы. На практике человек сам решает какую единицу использовать в своем исследовании. Например для звучащего текста чаще используются словоформы так как для первичного анализа употребляются фонетические слова.

Во время анализа текстов в МО обычно употребляются методы классификации, кластеризации, суммирования и регрессии. Самой сложной задачей данного блока является задача научить машину «читать». Для человека текст это набор слов имеющих смысл, определенную последовательность, орфография тоже имеет важную роль, в это же время вводные данные для машины это обычный набор битовых данных, ничем не связанных между собой.

Первым шагом в обработке будет представление документа, на этом этапе мы преобразовываем текст из набора в вектор документа. Сам вектор обычно представляет из себя векторы весов терминов. Далее уже идет извлечение признаков и их подбор.

Во время извлечения признаков при помощи удаления стоп-слов, применения стеммизации и т. д., текстовый документ становится пригодным для дальнейшей обработки. Токенизация – лексический анализ текста, с ее помощью находится минимальная единица в тексте. Токен – единица. В лексическом анализе за токен мы можем взять слово, предложение или целый абзац.

Стеммитизация – это процесс сокращения морфологических форм слова до его корня. Существуют два условия, которые следует учитывать, используя алгоритм стеммера:

- морфологическая форма слова должна совпадать с одном и тем же корнем и иметь одно и то же базовое значение.

- слова, не имеющие одинакового значения должны хранится отдельно.

Также не будет забывать, что все языки отличаются по сложности морфологии, поэтому и влияние разное.

Стоп-листинг – применяется для удаления стоп-слов, к ним относятся и символы. Обычно частота стоп-слов в тексте превышает количество ключевых слов. Поэтому данный шаг нельзя пропускать для получения четкого результата в МО.

# Анализ текста с помощью языка Python.

## Набор текстовых данных для анализа

Смысловой анализ, иногда также называемый глубинным анализом мнений (option mining), представляет собой популярную дисциплину из более широкой области NLP, он занимается автоматизацией обработки документов. Популярной задачей при смысловом анализе является классификация документов на основе уже выраженных мнений или эмоций авторов по отношению к определенной теме.

В этой работе мы будет работать с рецензиями на фильм, крупный набор данных из базы IMDb, он был собран Эндрю Маасом и другими для изучения словарных векторов в задачах смыслового анализа. Сам набор данных состоит из 50000 рецензий на фильмы, которые заранее промаркерованы на положительные и отрицательные, в данном контексте положительный отзыв имеет фильм с рейтингом IMDb выше шести звезд, а отрицательный – меньше пяти звезд.

## Подготовка данных для обработки текста

### Импорт текстовых файлов.

Сжатый архив в формате gzip мы загружаем по ссылке [Sentiment Analysis (stanford.edu)](https://ai.stanford.edu/~amaas/data/sentiment/). По ссылке находится подготовленный набор данных для двоичной классификации рецензий. В архиве предоставлены 25000 популярных отзывов на фильмах, которые используются для обучения модели, и 25000 для тестирования. Также нам предоставляются необработанные текста.

На Windows, проще всего для распаковки данных будет использовать 7-Zip, это бесплатная программа архивации.

Для пользователей среды разработки Linux или macOS распаковку можно совершить при помощи терминала.

В данной работе будет использоваться альтернативный – универсальный вариант – распаковка сжатого массива данных при помощи модуля tarfile в Python:



Рисунок 1

### **Предварительная обработка набора данных.**

Целью данного параграфа является приведение набора данных в более удобный формат. Так, после успешной разархивации, мы постараемся собрать текстовые документы в один большой файл CSV формата. Для этого мы используем объект DataFrame из pandas, сам процесс обработки может занять около 15 минут. Чтобы наблюдать за процессом обработки набора данных подключим пакет реализации индикатора выполнения на Python – Pyprind. Данный пакет был разработан в таких целях. Но для начала его нужно установить:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2

В приведенном ниже коде в basepath указывается каталог, где находится разархивированный архив. Но так как файл с программой и архив находятся в одной папке, нам не приходится прописывать полный путь:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3

Для начала мы инициализируем новый объект индикатора выполнения pbar, и указываем 50000 итерация, это количество соответствует количеству документов, с которыми мы будем работать дальше. При помощи цикла for проходим по обучающемуся и тестовому подкаталогам train и test соответственно. Далее считывает файлы из подкаталогов положительные и отрицательные рецензии pos и neg соответственно, которые добавляем к pandas-объекту DataFrame по имени df вместе с целочисленными метками классов, 1 – для положительных рецензий и 0 – для отрицательных.

Изначально метки классов в собранном нами наборе отсортированы, и для дальнейшего анализа нам требуется перетасовать объект, для этого мы используем библиотеку NumPy и функцию permutation. Для собственного удобства сохраним перетасованный набор данных с отзывами в файле CSV формата:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4

После успешного выполнения стоит проверить, что данные были успешно сохранены в правильный формат, для теста прочтем первые три образца:

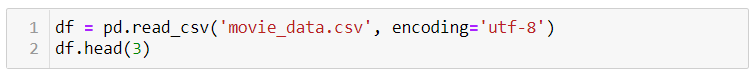


Рисунок 5

Если все правильно отработалось в Jupyter Notebook должна отобразиться таблица с первыми тремя образцами из набора данных, точнее показана ниже:

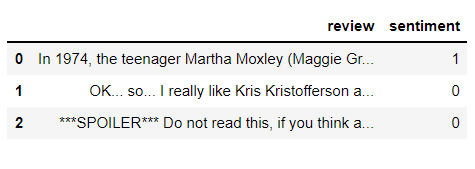


Рисунок 6

Для проверки работоспособности перед тем как перейти к следующему этапу следует удостовериться, что объект DataFrame имеет в себе все 50000 строк:

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 7

# Анализ набора данных

Работая с набором данных впервые полезно сначала исследовать его, чтобы получить знания, которые можно будет применить в дальнейшей работе.

## Модель суммирование слов

В этой части мы введем модель суммирования слов (bad-of-words model), которая позволит нам представить текстовые данные в виде векторов числовых признаков. Сама идея суммирования слов описана довольно просто и описана ниже:

- Требуется создать словарь уникальных лексем – слов из полного набора документов.

- После из каждого документа создается вектор признаков, данный набор состоит из счетчиков частоты появления каждого слова в уникальном документе.

Так как уникальные слова в каждом документе появляются в крайне небольшом количестве от всех слов в глоссарии модели суммирования слов, то сами векторы будут главным образов состоять из нулей, поэтому мы называем из разреженными.

## Трансформирование слов в векторы

Для того чтобы построить модель суммирования слов, беря в основу счетчики слов из соответствующих документов, будем использовать класс CountVectorizer, который реализован в библиотеке scikit-learn. Ниже продемонстрирован кусочек кода, где CountVectorizer принимает массив текстовых данных и конструирует модель суммирования слов на примере наименования Финансового Университета:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8

Вызывая метод fit\_transform через CountVectorizer, мы создаем глоссарий модели суммирования слов. Для более наглядного понимая концепции выведем содержимое глоссария:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9

Только что полученный глоссарий находится в словаре Python, в нем отражены уникальные слова на целочисленные индексы. Попробуем теперь вывести векторе признаков, которые только что создали:



Рисунок 10

Индексы позиции в предоставленных векторах соответствуют целочисленным значениям, которые хранятся как части словаря в глоссарии созданном при помощи CountVectorizer.

Этот метод может показать нам так называемые серые частоты термов (raw frequency): tf (t, d) – сколько раз терм t встречается в документе d. Также необходимо сказать, что порядок, котором находятся слова в предложении или документе, не играют роли и никак не влияют на частоты термов. А порядок, который определяем расположение термов в векторе признаков, определяется индексами в словаре, обычно они назначаются по алфавиту.

Класс CountVectorizer в scikit-learn дает нам возможность применять модели основываясь на разные n-граммы при помощи своего параметра ngram\_range. В это же время по умолчанию используется предоставление в виде 1-граммы, также при дальнейшей работе мы могли бы переключиться на 2-граммы, при этом нам потребуется инициализировать новый объект CountVectorizer и указать ngram\_range = [2,2].

## Оценка важности слов при помощи приема tf-idf

Во время анализа текстовых документов мы часто сталкиваемся со словами, которые уже встречались нам во множестве других документов из разных классов. К сожалению, эти слова не содержат для нас никакой полезной или различительной информации. Но была создана удобная мера, которая называется, - обратная частота документа (term frequency-inverse document frequency – tf-idf), обычно она применяется для снижения веса частот встречающихся слов в векторах признаков. Для расчета меры tf-idf требуется умножить частоту терма на обратную частоту документа:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11

В формуле tf (t, d) – это частота терма, которая была введена ранее, а idf (t, d) – обратная частота документа, она вычисляется по формуле ниже:

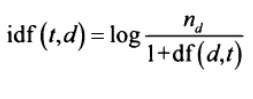


Рисунок 12

В приведенной выше формуле nd представляет собой общее количество документов, в частота df (t, d) – количество документов d, содержащих терм t. Остается только разобрать, почему к знаменателю добавляется единица. Этот шаг является не обязательным и несет цель присвоить ненулевое значение термам, которые встречаются во всех обучающих образцах; логарифм используется для гарантии, что документам с низкими частотами не назначится слишком большой вес.

В библиотеке scikit-learn реализован еще один преобразователь, класс TfifTransformer, которые принимает в качестве входа сырые частоты термов от класса CountVectorizer и трансформирует их в меры tf-idf:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13

Если бы изначально вручную вычисляли меры tf-idf индивидуальных термов в векторах признаков, то заметили бы, что TfifTransformer вычисляет меры tf-idf немного иначе по сравнению со стандартными формулами. Уравнение представленное в библиотеке scikit-learn выглядит следующим образом:

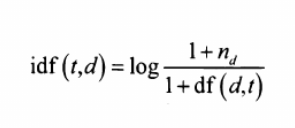


Рисунок 14

По похожей формуле вычисляются и меры tf-idf в scikit-learn, оно совсем немного отличается того, что было представлено ранее:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15

Добавление единицы в формуле выше связано с установкой smooth\_idf=True из примера кода выше, она полезна для назначения нулевых весов термам, которые встречаются во всех документах.

Наряду с тем, что обычно проводится нормализация сырах частот термов перед вычислением мер tf-idf, класс TfifTransformer нормализуем меры tf-idf напрямую. По умолчанию выставляется norm = ‘l2’ класс TfifTransformer из scikit-learn применяет нормализация L2, которая возвращает вектор длиной 1 путем деления ненормализованного вектора признаков v на его норму L2:

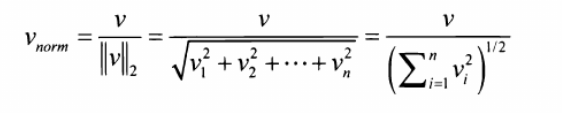


Рисунок 16

# Очистка текстовых данных

Несмотря на то, что мы уже ознакомились с моделью суммирования слов, узнали о частотах и термах в документах, не стоит пропускать такой важный шаг, как очистка данных, которая должна быть всегда. В нашем случае очистка данных предусматривает очистку данных путем удаления нежелательной символики. Для важности этого шага мы отобразим последние 50 символов одного из документов в перетасованном наборе данных с отзывами о фильмах:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17

Здесь трудно не заметить HTML-разметку, знаки препинания и другие небуквенные символы. Вся эта символика не несет в себе никакой полезной семантики, хотя для NLP знаки препинания могут предоставить важную дополнительную информацию. Однако мы постараемся удалить все знаки препинания кроме эмотиконов по типу :), так как они полезны для смыслового анализа и определения эмоционального состояния рецензента. Для решения данной задачи мы будем использовать библиотеку для работы с регулярными выражения (re):

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 18

С помощью регулярного выражения - '<[^>] \*>' – мы постараемся удалить все ненужную для смыслового анализа HTML-разметку. После удаления мы задействуем чуть более уже сложное выражение для нахождения эмотиконов и временно сохраним их в emoticons. Далее удалим из текста не словарные символы посредством регулярного выражения - [\W] + - и одновременно с этом приведем все буквы к нижнему регистру.

После проделанной работы заполняем строки документа эмотиконами, которые временно хранятся в emoticons. Также для согласованности уберем из эмотиконов символ «носа», то есть «-».

Возможно эстетически некрасиво, что в конце строки добавляется эмотикон, но стоит отметить, что используя модель суммирование слов их порядок не имеет значения, так как глоссарий состоит из однословных лексем.

Перед тем как перейти к следующему этапу, проверим правильно ли работает preprocessor:

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 19

После успешной проверки preprocessor`а мы будем использовать очищенные текстовые документы, поэтому применим данную функцию на весь DataFrame с рецензиями.

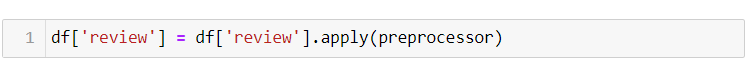


Рисунок 20

# Переработка документов в лексемы

Можно уверено сказать, что мы подготовили набор данных и теперь нам стоит задуматься о разделении самих текстов на элементы. Одним из таких способов является разбиение очищенных документов на отдельные слова основываясь на пробельные символы:

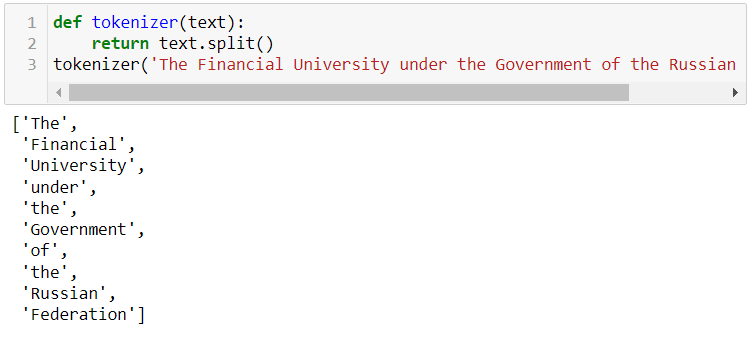


Рисунок 21

В случае разбивки на лексемы еще одним удобным способом является стемминг слов (word stemming), при помощи него мы находим основу слова, этот процесс предполагает трансформирование слова в его корневую форму. Также данный прием помогает отображать односвязные слова на одну и ту же основу.

Далее мы будем работать с пакетом библиотек NLTK, поэтому не забудем про установку:



Рисунок 22

Следующим шагом демонстрируется алгоритм стемминга Портера. При использовании класса PorterStemmer из пакета nltk, модифицируем функцию tokenizer, таким образом сокращаем слова из текста до корневой формы:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 23

Перед переходом к следующему этапу требуется учесть, что наши тексты перезаполнены стоп-словами. Стоп-слова – слова, которые крайне часто преобладают во всех видах текстов, а также не несут в себе много полезной информации, а при анализе, как в нашем случае, могут только мешать. Для удаления стоп-слов мы будем использовать набор из 127 слов английского языка, его можно загрузить, используя библиотеку NLTK:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 24

После успешной загрузки пакета стоп-слов применяем его:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 25

# Обучение логистической регрессионной модели для классификации документов

На данном этапе мы обучим логистическую регрессионную модель, основанную на модели суммирования слов, которая будет классифицировать рецензии на позитивные и негативные. Для начала разобьем объект с очищенными документами DataFrame на 25000 документов для обучения и 25000 для испытания, как показано ниже:

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 26

Чтобы найти оптимальный набор параметров для логистической регрессии воспользуемся объектом GridSearchCV и применим стратифицированную перекрестную проверку сразу по пяти блокам:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 27

Чтобы ускорить поиск изначально выставим n\_jobs=-1, это нужно для задействования всех свободных ядер процессора.

При инициализации объекта GridSearchCV и его структуры параметров (param\_grid), используя показанный выше код, так мы ограничиваемся лимитированным числом комбинаций параметров. Несмотря на установки выполненные заранее процесс решетчатого поиска может занять около полутора часов на ноутбуке средней мощности.

В приведенном выше кусочке кода мы заменили объекты Count Vectorizer и TfidfTransformer объектом TfidfVectorizer, он комбинирует Count Vectorizer с TfidfTransformer. Наш param\_grig состоит из двух словарей параметров. В первом применяется TfidfVectorizer со стандартными настройками для вычисления мер tf-idf. Во втором словаре устанавливаются параметры 'vect\_\_use\_idf’: [False] и 'vect\_\_norm’: [None], чтобы обучить модель на сырых частотах термов.

После завершения решетчатого поиска мы может вывести наилучший набор параметров:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 28

По результатам вывода видно, что наилучшие результаты поиска мы получили, применяя обычное разбиение на лексемы без стеммера Портера, при отсутствии библиотек стоп-слов, с использованием мер tf-idf в сочетании с классификатором на основе логистической регрессии, при этом применяя регуляризацию L2 с обратным параметром регуляризации С, равным 10.0.

Так как мы уже имеем наилучшую модель, найденную при помощи решетчатого поиска, давайте выведем среднюю меру правильности при перекрестной проверке по пяти блокам на обучающем наборе и правильность классификации на испытательном наборе данных:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 29

По результатам можно отметить, что наша модель способна спрогнозировать, является ли рецензия на фильм позитивной или негативной примерно с 90% правильностью.

# Работа с более крупными данными – внешнее обучение

Сейчас мы попробуем воспользоваться функцией partial\_fit класса SGDClassifier из scikit-learn для потоковой передачи текстов, которые будут идти напрямую из локального диска небольшими мини-пакетами и обучим логистическую регрессионную модель.

Для начала требуется определить функцию tokenizer, она помогает очистить необработанные данные из файла, который мы создавали в самом начале, далее разобьем его на лексемы и удалит стоп-слова:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 30

Следующим ходом определим генераторную функцию, которая читает и возвращаем один документ за раз:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 31

Для того, чтобы проверить корректность работы нашей программы попробуем прочитать первый документ из файла, в результате мы должны получить кортеж с текстом рецензии и соответствующую метку класса:

Изображение выглядит как текст, внутренний, снимок экрана

Автоматически созданное описание

Рисунок 32

Затем определим новую функцию, которая будет принимать поток документов от предыдущей функции и возвращать количество документов, указанное в параметре size:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 33

При дальнейшей работе нам придется отказаться от классов CountVectorizer и TfidfVectorizer, так как они занимают слишком большой объем в памяти. Однако мы будем использовать HashingVectorizer, который так же доступен в библиотеке scikit-learn:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 34

В представленном выше коде идет инициализация объекта HashingVectorizer с функцией tokenizer и устанавливается количество признаков, на нашем случае 2\*\*21. Кроме того, устранавливаем классификатор на основе логистической регрессии добавляя параметр loss объекта SGDClassifier в “log”.

Создав все дополнительные функции, можно начать внешнее обучение, для этого запустим приведенный ниже код:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 35

Для оценки эффективности модели возьмем последние 5000 документов:

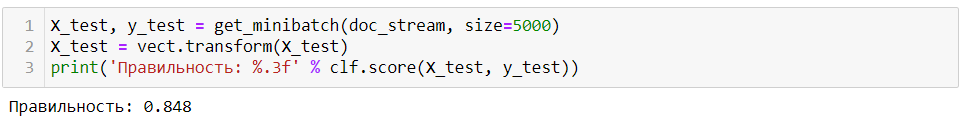


Рисунок 36

По результатам видим, что правильность составляет всего 85%, это ниже чем правильность, которая была у нас ранее, когда мы использовали решетчатый поиск. Но несмотря на это, внешнее обучение невероятно рационально по отношению к расходу памяти и времени затраченного на процесс. Теперь применяем последние 5000 документов для обновления модели:

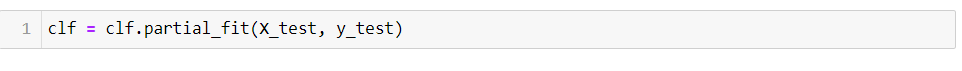


Рисунок 37

# Тематическое моделирование с помощью латентного размещения Дирихле

Тематическое моделирование (topic modeling) описывает значительную часть задач по темам классификации текстовой информации. Грубо говоря, данный раздел можно назвать задачей кластеризации.

Здесь будет применяться прием латентное размещение Дирихле (LDA). LDA – порождающая вероятность модель, она старается отыскать группы слов, часто встречающихся вместе в различных документах. Именно эти слова представляют наши темы. На входе LDA получает модель суммирования слов, которую мы уже создали и подготовили, и разделяет на две новые матрицы:

- матрица, отображающая документы по темам;

- матрица, отображающая слова на темы;

Далее, если эти две матрицы перемножить, то будет состояние чтобы произвести вход, то есть матрицу суммирования слов, с самой низкой возможной ошибкой.

# Реализация LDA в библиотеке scikit-learn

Здесь мы будем применять класс LatentDirichlet Allocation для разбивки набора данных с рецензиями на разные темы. В примере ниже ограничивается анализ тем десятью темами.

Сначала загрузим данные в объект DataFrame из pandas, которые мы собрали ранее:

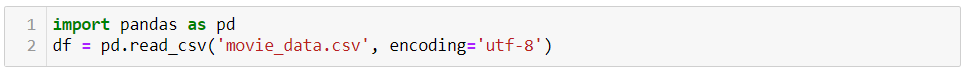


Рисунок 38

Следующим шагом создаем матрицу суммирования слов, которая является входом для LDA, и ради удобства применим встроенную библиотеку со стоп-словами, которая упоминалась ранее:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 39

Сразу устанавливаем максимальную частоту слов, чтобы исключить часто встречающиеся слова. А также ограничимся количеством уникальных слов, чтобы улучшить выведение, выполняемое LDA.

В приведенном ниже коде демонстрируется подгонка LatetntDirichletAllocation к матрице суммирования слов и выводятся десять тем документов. Весть процесс может занять примерно 10 минут:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 40

Когда мы устанавливаем в learning\_method ‘batch’, идет расчет оценки на основе всех доступных обучающих данных.

После подгонки LDA для нас открывается доступ к атрибуту, который хранит матрицу, содержащую значения важности слов в порядке возрастания:

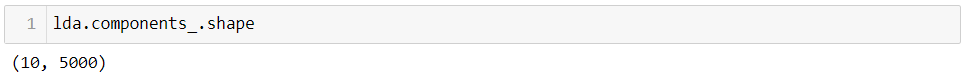


Рисунок 41

Для анализа результатов попробуем вывести по пять самых важных слов для каждой темы:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 42

На основании результатов можно выдвинуть предположения, как идентифицированы эти фильмы:

1. Фильмы с низким рейтингом, плохие
2. Семейные фильмы
3. Военные фильмы
4. Фильмы об искусстве
5. Криминальные фильмы
6. Фильмы ужасов
7. Комедии
8. Возможно обзоры телевизионных шоу
9. Фильмы по книгам
10. Боевики

Для подтверждения нашей теории попробуем вывести три фильма жанра ужасы - 6 категория, в индексной позиции 5:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 43

на основании выведенных данных мы можем утвержадать, что модель работает, но по 2-му фильму все еще остаются вопросы, хотя на основании всего 300та символом можно было бы отнести его просто к фильмам с низким рейтингом.

# Заключение

В ходе выполнения курсовой работы были изучен теоретический материал по текстовым данных и машинному обучению. Были рассмотрены особенности и различные способы построения моделей машинного обучения.

Мы научились учитывать факт, что при работе с текстовыми данными довольно высокие затраты памяти и вычислительные затраты из-за больших векторов признаков.

Все полученные знания можно применять для разных видов текстовой информации.

# Литература

1. С. Рашка, В. Мирджали Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2, 3-е издание: Диалектика, 2020 г. - 846 с.
2. Вандерплас Д. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с.
3. Х.Брикс, Дж.Ричардс, М.Феверолф Машинное обучение: Питер, 2017 г. – 336 с.
4. А. Мюллер, С. Гвидо Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными: Москва, 2017г. – 393 с.

# Приложение

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание