Федеральное государственное образовательное бюджетное учреждение высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

Факультет «Информационных технологий и анализа больших данных»

КУРСОВАЯ РАБОТА

по дисциплине «Машинное обучение»

на тему:

«Обработка данных о научных изданиях и статистики цитирования»

Выполнила:
студент группы ПМ21-5
Дворянова С.Я.

Научный руководитель:
доцент, к.т.н. Волков А.В.

Москва 2023

Содержание

Введение.

Согласно теме моей курсовой работы: «Обработка данных о научных изданиях и статистики цитирования» моя задача заключается в том, чтобы проанализировать информацию о статистике цитирования в научных изданиях с помощью доступных инструментов машинного обучения.

Другой задачей является определение авторства научных статей для того, чтобы определить наличие цитирования. Методы машинного обучения, такие как анализ сети цитирования и анализ текстовых признаков, могут использоваться для идентификации текстов и их авторов на основе уникального стиля написания.

Итак, машинное обучение представляет собой мощный инструмент для обработки наукометрической информации, который может помочь исследователям в анализе и интерпретации больших объемов данных.

Цель работы: изучение методов машинного обучения, которые применяются для анализа и обработки данных текстов научных изданий, а также подсчитывают статистику цитирования.

Задачи:

1. Формирование датасета.
2. Изучение и анализ используемых данных.
3. Построение модели МО, которая будет определять наиболее часто встречающиеся слова.
4. Оценить работу модели МО.
5. Визуализировать работу модели.

Предполагаемый результат: модель МО, определяющая наиболее популярных авторов.

Выбор издательства

Для того, чтобы приступить к решению данной задачи мне было необходимо найти какие-либо данные. Я воспользовалась сайтом ITB Journal (https://journals.itb.ac.id/).

Структура сайта была последовательна от списка до статей:

https://journals.itb.ac.id/index.php/jets -> https://journals.itb.ac.id/index.php/jets/article/ARTICLE_NAME -> https://journals.itb.ac.id/index.php/jets/article/ARTICLE_NAME/ARTICLE_NUMBER

Скачивание страниц

Следующий этап – скачать HTML-страницы каждой статьи, на которой содержится наукометрическая информация. Для этой задачи было решено использовать Scrapy и Python.

Scrapy – это совместный фреймворк с открытым исходным кодом для извлечения нужных нам данных с веб-сайтов. Он работает быстро и является одной из самых мощных библиотек, доступных на сегодняшний день.

Программный код:

```
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

class FianItem(scrapy.Item):
    file_urls = scrapy.Field()
    files = scrapy.Field

class NatureSpider(CrawlSpider):
    name = 'itb'

    custom_settings = {
        'FILES_STORE': "data/" + name,
        'LOG_FILE': "log/" + name + ".log",
        'DOWNLOAD_DELAY': 0.3,
        'CONCURRENT_REQUESTS': 1,
        'CONCURRENT_REQUESTS_PER_DOMAIN': 1,
        'CONCURRENT_REQUESTS_PER_IP': 1,
```

```
        'ROBOTSTXT_OBEY': False,
        'USER_AGENT': 'Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0
Safari/537.36',
        'MEDIA_ALLOW_REDIRECTS': True,
        'HTTPCACHE_ENABLED': True,
        'HTTPCACHE_EXPIRATION_SECS': 0,
        'HTTPCACHE_DIR': 'httpcache',
        'HTTPCACHE_IGNORE_HTTP_CODES': [301, 302, 403],
        'HTTPCACHE_STORAGE':
'scrapy.extensions.httpcache.FilesystemCacheStorage',
    }
    allowed_domains = ['journals.itb.ac.id']
    start_urls = ['https://journals.itb.ac.id/']

    rules = (
    Rule(LinkExtractor(allow=('.+\/index.php\/\D{1,6}')),
    callback='parse', follow=True),

    Rule(LinkExtractor(allow=('.+\/issue\/archive')),
    callback='parse', follow=True),

    Rule(LinkExtractor(allow=('.+\/view\/\d+')),
    callback='parse', follow=True),
    )

    def parse(self, response):
        urls = []
        s = response.xpath('//h3[@class="title"]/a')
        if (s):
          for i in s:
            urls.append(response.urljoin(i.attrib['href']))
        else:
          return
        item = FianItem()
        item['file_urls'] = urls
        yield item
```

Данный код представляет собой веб-паука (spider) на основе фреймворка Scrapy, который проходит по сайту journals.itb.ac.id и скачивает все доступные файлы. Сначала импортируются необходимые модули Scrapy. Затем создается класс FianItem, который является контейнером для данных, собираемых веб-пауком. Он содержит два поля: file_urls и files.

Затем создается класс NatureSpider, который наследуется от CrawlSpider - класса, основанного на правилах, для создания web-пауков. В нем определяются настройки для Scrapy, такие как FILES_STORE (место, куда будут

сохраняться скачиваемые файлы), USER_AGENT (идентификатор браузера), CONCURRENT_REQUESTS (максимальное количество одновременных запросов), и другие. В нем же происходит сам скраппинг со следующими настройками:

1. custom_settings – настройки для паука, например указан путь куда сохранять результат, логи; задержка скачивания и другое.

2. allowed_domains – разрешенные домены, по которым может передвигаться паук, в моём случае это только impan.pl

3. start_urls – ссылка, с которой начинается скраппинг, здесь же – ссылка на все журналы выбранного мной издательства

4. rules – тут прописаны правила, по которым Scrapy переходит на сайты.

После Scrapy доходит до последней страницы, которая прописана в его правилах, там же находится ссылка, которая ведёт нас на сайт с наукометрической информацией определенной статьи. Информацию с этого сайта мы получаем с помощью

```
response.xpath('//div[@class="info"]/a')
```

Далее, если ответ есть (эта строка существует в HTML коде), то мы достаём то, что у нас в модуле href и добавляем в urls, дальше достаём элемент с помощью FianItem и в ключе file_urls добавляем то, что нашли. Затем метод yield возвращает созданный объект item, чтобы Scrapy мог продолжить его обработку.

После выполнения кода и ожидания 1 часа у меня получилось скачать около 1000 HTML страниц с наукометрической информацией.

Парсинг

Следующей моей задачей было извлечь конкретную информацию в один общий csv файл. После небольшого изучения структуры HTML страниц статей моего издателя я решила извлечь следующие данные: название журнала, название статьи, eISSN, pISSN, DOI, аффиляции (автор-институт), год публикации, номера страниц статьи, абсолютный номер публикации, ссылка на статью, ключевые слова.

Программный код:

```
from os import listdir
from os.path import isfile, join
from parsel import Selector
import pandas as pd
import re

def parser(file):
    with
open('C:/coursework/pars/pars/spiders/data/itb/full/'+file, 'r',
encoding="utf8", errors='ignore') as fp:
        data = fp.read()

    result = {
        'journal': '',
        'title': '',
        'ISSN': '',
        'DOI': '',
        'affilations': [],
        'date': '',
        'pages': '',
        'volume': 0,
        'issue': 0,
        'raw_url': '',
        'keywords': []

    }

    selector = Selector(text=data)

    #DOI
    s = selector.xpath('//meta[@name="citation_doi" or
@name="DOI"]')
    if (s):
        result['DOI'] = s.attrib['content']
    else:
        print(file + "doi")
```

```
    #title
    s = selector.xpath('//meta[@name="citation_title" or
@name="dc.title"]')
    if (s):
        s = s.attrib['content']
        result['title'] = " ".join(s.split())
    else:
        print(file + 'title')

    #journal
    s = selector.xpath('//meta[@name="citation_journal_title" or
@name="prism.publicationName"]')
    if (s):
        s = s.attrib['content']
        result['journal'] = " ".join(s.split())
    else:
        print(file + 'journal')

    #ISSN
    s = selector.xpath('//meta[@name="citation_issn"]')
    if (s):
        result['ISSN'] = s.attrib['content']
    else:
        print(file + 'issn')

    #Authors
    s1 = selector.xpath('//meta[@name="citation_author"]')
    s2 =
selector.xpath('//meta[@name="citation_author_institution"]')
    ans1, ans2 = '', ''
    if s1 or s2:
        if s1:
            ans1 = s1.attrib['content']
        if (s2):
            ans2 = s2.attrib['content']
        result['affilations'].append({'Author': ans1,
'affilation': ans2})
    else:
        print(file + 'authors')

    #year
    s = selector.xpath('//meta[@name="citation_date"]')
    if (s):
        s = s.attrib['content']
        result['date'] = s
    else:
        print(file + 'date')

    #pages
    s1 = selector.xpath('//meta[@name="citation_firstpage"]')
    s2 = selector.xpath('//meta[@name="citation_lastpage"]')
    if (s1 and s2):
        s1 = s1.attrib['content']
        s2 = s2.attrib['content']
        result['pages'] = s1+'-'+s2
```

```
    else:
        print(file + 'pages')

    #volume
    s = selector.xpath('//meta[@name="citation_volume"]')
    if (s):
        s = s.attrib['content']
        result['volume'] = int(s)
    else:
        print(file + 'volume')

    #issue
    s = selector.xpath('//meta[@name="citation_issue"]')
    if (s):
        s = s.attrib['content']
        result['issue'] = int(s)
    else:
        print(file + 'issue')

    #url
    s =
selector.xpath('//meta[@name="citation_abstract_html_url"]')
    if (s):
        s = s.attrib['content']
        result['raw_url'] = s
    else:
        print(file + 'url')

    #keywords
    s = selector.xpath('//meta[@name="citation_keywords"]')
    for i in s:
        if (i):
            i = i.attrib['content']
            result['keywords'].append(i.split(' '))
        else:
            print(file + 'keywords')

    return (result)

answer = []
mypath = 'C:/coursework/pars/pars/spiders/data/itb/full'
all = [f for f in listdir(mypath) if isfile(join(mypath, f))]

for i in all:
    answer.append(parser(i))

pd.DataFrame(answer).to_csv('C:/coursework/pars/parser/output_it
b.csv')
```

Данный код представляет собой скрипт для парсинга файлов статей на найденном сайте. Сначала импортируются необходимые модули (os, parsel, pandas, re). Затем

определяется функция parser, которая принимает на вход имя файла статьи и возвращает словарь с метаданными статьи, такими как название журнала, название статьи, DOI, авторы, ключевые слова и другие параметры.

Далее создается пустой массив answer, который будет содержать все полученные метаданные. Затем определяется путь к папке с файлами статей mypath. На основе этого пути извлекаются имена всех файлов all. Далее цикл for проходится по всем файлам и передает их в функцию parser. Результат выполнения функции добавляется в массив answer.

Наконец, массив answer преобразуется в DataFrame из библиотеки pandas и сохраняется в CSV-файл.

Итак, сбор датасета завершен.

После создания CSV-файла можно начать с ним работать. Загружаем датафрейм в колаб, проходимся по нему и, используя методы машинного обучения, пытаемся узнать количество ссылок на журналы, находящееся в других журналах для того, чтобы подсчитать статистику цитирования.

Первым этапом анализа готового текста является **предобработка данных**, так как книги из датасета, которые мы собираемся анализировать, представляют собой неструктурированный текст.

Предобработка данных

Импортируем нужные библиотеки

```
!pip install skimpy

!pip install phik

import pandas as pd

import numpy as np

from skimpy import clean_columns

import matplotlib.pyplot as plt
```

```
import seaborn as sns
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('stopwords')


import string
import pandas as pd
import phik
from phik.report import plot_correlation_matrix
from phik import report


from sklearn.model_selection import train_test_split


from sklearn.linear_model import LinearRegression


from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
```

LinearRegression - это модель линейной регрессии, используемая для предсказания зависимой переменной на основе линейной комбинации одного или нескольких независимых (объясняющих) признаков.

r2_score - это метрика, которая используется для оценки качества работы модели линейной регрессии. Она представляет собой коэффициент детерминации, который измеряет, насколько хорошо модель подходит для данных. Значение r2_score находится в диапазоне от 0 до 1, где 1

означает, что модель идеально подходит для данных, а 0 - что модель не объясняет изменчивость в данных. Чем ближе значение r2_score к 1, тем лучше модель.

mean_absolute_error - это еще одна метрика, которая используется для оценки качества работы модели линейной регрессии. Она представляет собой среднюю абсолютную ошибку предсказания, то есть разницу между предсказанными и фактическими значениями. Меньшие значения mean_absolute_error указывают на более точную модель.

Так как я выполняю работу в Google Colab, заранее загрузила готовые файлы на диск и открываю к ним доступ.

```
from google.colab import drive

drive.mount('/content/drive')
```

Считываю файлы

```
data_2019 = pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings.xlsx',sheet_name = '2019')

data_2018 = pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings.xlsx',sheet_name = '2018')

data_2017 = pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings.xlsx',sheet_name = '2017')

data_2016 = pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings.xlsx',sheet_name = '2016')

data_2015 = pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings.xlsx',sheet_name = '2015')
```

```
      data_2014 =
pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings
.xlsx',sheet_name = '2014')

      data_2013 =
pd.read_excel(r'/content/drive/MyDrive/Impact-Factor-Ratings
.xlsx',sheet_name = '2013')
```

Этот код читает данные из шести листов в файле Excel "Impact-Factor-Ratings.xlsx". Каждый лист содержит рейтинги журналов за год от 2013 до 2019. Данные считываются и сохраняются в шесть разных объектов pandas.DataFrame.

```
      frames =
[data_2019,data_2018,data_2017,data_2016,data_2015,data_2014
,data_2013]



      data = pd.concat(frames)
```

Список frames содержит исходные DataFrame, которые будут объединены. Затем метод pd.concat() объединяет эти DataFrame в один общий DataFrame data, соединяя их по вертикали (т.е. добавляя строки друг за другом).

В результате, DataFrame data будет содержать все данные из исходных DataFrame data_2019, data_2018, data_2017, data_2016, data_2015, data_2014 и data_2013, объединенные в один DataFrame.

```
      data.head()
```

```
data.head()
```

	Source title	CiteScore	Highest percentile	3_year_citations	3_year_documents	% Cited	SNIP	SJR	Publisher	year
0	Ca-A Cancer Journal for Clinicians	435.4	99.0%\n1/331\nOncology	47455	109	94	113.744	88.192	Wiley-Blackwell	2019
1	MMWR Recommendations and Reports	152.5	99.0%\n1/275\nHealth (social science)	2288	15	87	37.543	41.022	Centers for Disease Control and Prevention (CDC)	2019
2	Nature Reviews Materials	123.7	99.0%\n1/287\nMaterials Chemistry	23868	193	96	15.261	36.691	Springer Nature	2019
3	Chemical Reviews	100.5	99.0%\n1/398\nGeneral Chemistry	97295	968	96	12.832	20.847	American Chemical Society	2019

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Source title      7000 non-null   object 
 1   CiteScore         7000 non-null   float64
 2   Highest percentile  7000 non-null   object 
 3   3_year_citations  7000 non-null   int64  
 4   3_year_documents  7000 non-null   int64  
 5   % Cited           7000 non-null   int64  
 6   SNIP              6928 non-null   float64
 7   SJR               6979 non-null   float64
 8   Publisher         6959 non-null   object 
 9   year              7000 non-null   object 
dtypes: float64(3), int64(3), object(4)
memory usage: 601.6+ KB
```

Наблюдаем, что все файлы успешно соединились в один.

Уберем незначительное количество пропусков с нулевыми значениями, дабы последующая машинная обработка данных выполнялась правильно.

```
data = data.dropna()

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6877 entries, 0 to 998
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Source title      6877 non-null   object 
 1   CiteScore         6877 non-null   float64
 2   Highest percentile  6877 non-null   object 
 3   3_year_citations    6877 non-null   int64  
 4   3_year_documents    6877 non-null   int64  
 5   % Cited           6877 non-null   int64  
 6   SNIP              6877 non-null   float64
 7   SJR               6877 non-null   float64
 8   Publisher         6877 non-null   object 
 9   year              6877 non-null   object 
dtypes: float64(3), int64(3), object(4)
memory usage: 591.0+ KB
```

Видим, что пропусков теперь нет. Теперь для правильной работы программы необходим сброс индексов из-за пропущенных строк.

```
data = data.reset_index(drop = True)
```

Метод reset_index() используется для сброса индексов строк в DataFrame. Аргумент drop=True указывает, что старые индексы не должны быть сохранены в DataFrame.

```
arr = data['% Cited'].unique()

print("Median of %Cited Values: ",np.median(arr))
```

```
Median of %Cited Values:  72.5
```

```
      data['Median_of_Cited'] = ['higher72.5'if i >=72.5
else 'below72.5'for i in data['% Cited']]
```

Строка data['Median_of_Cited'] = ['higher72.5'if i >=72.5 else 'below72.5'for i in data['%_cited']] создает новый столбец Median_of_Cited в DataFrame data и заполняет его значениями на основе значений столбца %_cited. Если значение в столбце %_cited больше или равно 72.5, то в столбец Median_of_Cited присваивается значение 'higher72.5', иначе присваивается 'below72.5'.

Токенизация

```
#Для того чтобы избавиться от нежелательных слов и символов воспользуемся методом stopwords для английского языка

stop_words = nltk.corpus.stopwords.words('english')


# Реализуем предобработку

def preprocess(doc):

    # Убираем пунктуацию, пробелы и тд

    for p in string.punctuation:

        doc = doc.replace(p, ' ')

    return doc


data['Publisher'] = data['Publisher'].map(preprocess)


docs = list(data['Publisher'].values)

docs
```

```
['Wiley Blackwell',
 'Centers for Disease Control and Prevention  CDC ',
 'Springer Nature',
 'American Chemical Society',
 'American Physical Society',
 'Springer Nature',
 'Springer Nature',
 'Elsevier',
 'Springer Nature',
 'Springer Nature',
 'Centers for Disease Control and Prevention  CDC ',
```

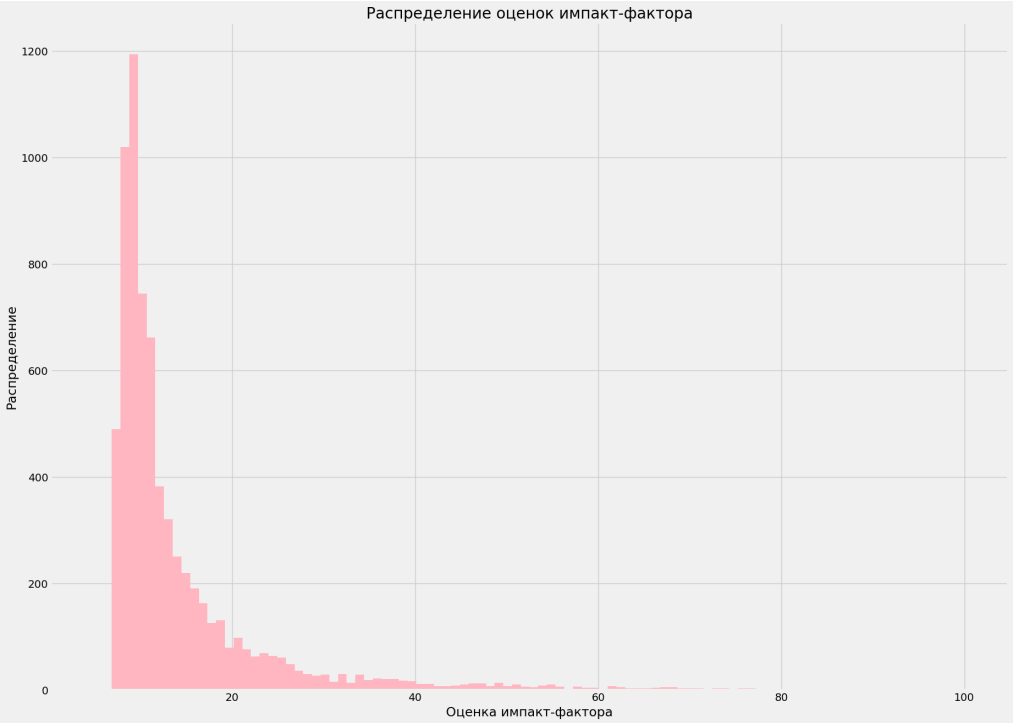
```
'Royal Society of Chemistry',
'Massachusetts Medical Society',
'Springer Nature',
'Springer Nature',
'Springer Nature',
'Elsevier',
'Springer Nature',...]
```

Для того, чтобы подсчитать статистику цитирования, важно учесть импакт-фактор, который, в свою очередь, является одним из показателей, используемых для оценки влиятельности журнала и может отражать количество цитирований статей, опубликованных в этом журнале. Высокий импакт-фактор служит индикатором высокого количества ссылок на статью.

Анализ EDA

EDA (Exploratory Data Analysis) - это метод исследования данных, направленный на изучение структуры данных и выявление основных характеристик, свойств и закономерностей в данных. EDA - это важная часть процесса машинного обучения и анализа данных, так как позволяет получить более глубокое понимание данных, их структуры и свойств.

```
      data.CiteScore.hist(bins=100, figsize=(20, 15), range=(5, 100), color = 'lightpink')


      plt.title('Распределение оценок импакт-фактора')
      plt.xlabel('Оценка импакт-фактора')
      plt.ylabel('Распределение')
      plt.grid(True)


      None #убирает лишнюю строчку над графиком
```

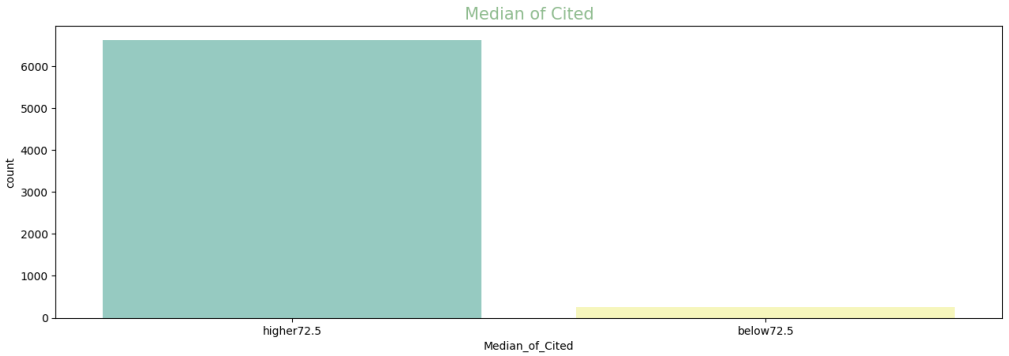



```
#Топ авторов публикаций согласно импакт-фактору
data.Publisher.value_counts().head(10)
```

```
Elsevier                      1900
Wiley Blackwell                969
Springer Nature                873
IEEE                           318
Oxford University Press        287
Annual Reviews Inc             246
Taylor   Francis               222
Wolters Kluwer Health          190
SAGE                           168
American Chemical Society      165
Name: Publisher, dtype: int64
```

```
fig, axs = plt.subplots(ncols=1)
plt.subplots_adjust(right=2, wspace = 0.5)
```

```
      sns.countplot(x=data['Median_of_Cited'],
palette="Set3")

      axs.set_title('Median of Cited', color
='darkseagreen', fontsize=15)

      None
```

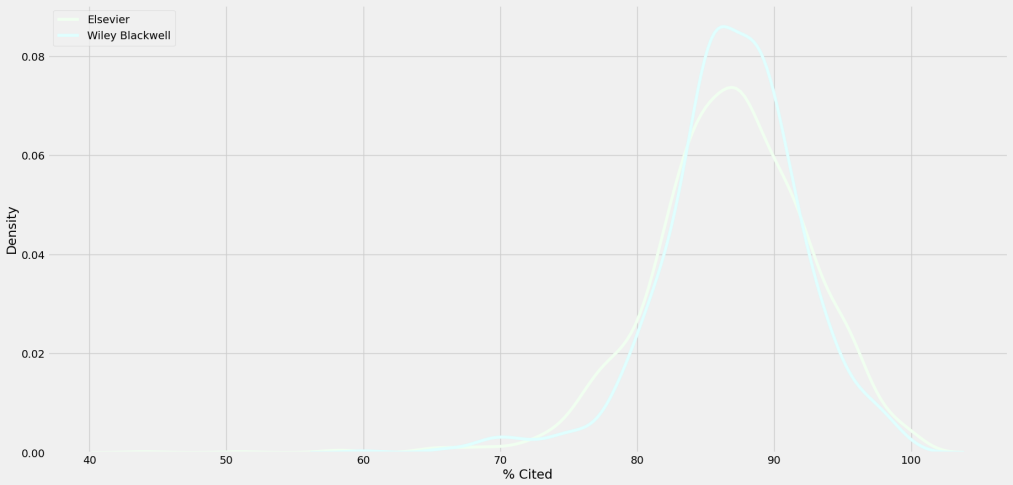


'Median of Cited' означает медиану значения процента цитирования статей, которые относятся к нашей группе. Оно равно 72.5, согласно задаче. На гистограмме наглядно можем увидеть, что процент цитирования значительно выше медианного показателя. Это говорит о высоком качестве журналов и чудесной статистике цитирования!

Но нужно учитывать, что значение 'Median of Cited' определяется как медиана значений процента цитирования в выборке данных. Если значение процента цитирования выше медианы, то оно будет отнесено к категории 'higher72.5' в новой колонке 'Median_of_Cited'. Значения процента цитирования в этой категории будут значительно выше медианного значения. Например, если медиана процента цитирования равна 20, то значения процента цитирования, которые равны 50 или 60, будут отнесены к категории 'higher72.5'.

Визуализация плотности вероятности.

```
plt.style.use("fivethirtyeight")

plt.figure(figsize=(20, 10))

sns.kdeplot(data.loc[data['Publisher'] == 'Elsevier', '% Cited'], label = 'Elsevier',shade=False, color = 'honeydew')

sns.kdeplot(data.loc[data['Publisher'] == 'Wiley Blackwell', '% Cited'], label = 'Wiley Blackwell',shade=False, color = 'lightcyan')

plt.legend(['Elsevier','Wiley Blackwell'], loc=2)

plt.show()

None
```



Визуализация плотности вероятности (на картинке сверху) может помочь понять распределение данных и выделить особенности, которые могут быть незаметны при простом рассмотрении гистограммы.

В нашем случае график плотности вероятности имеет только один пик, и это может указывать на то, что данные имеют

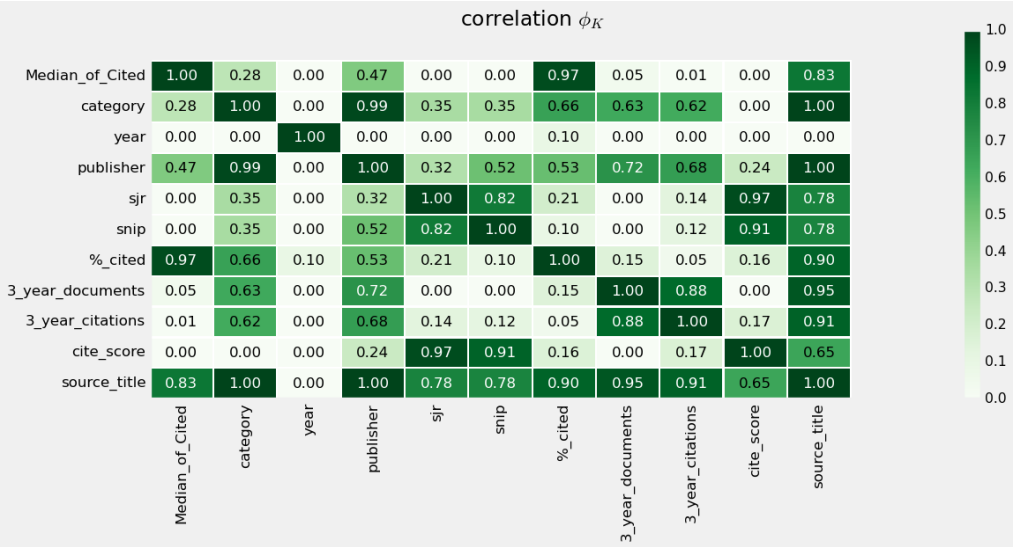
простое одномодальное распределение, то есть наиболее вероятное значение находится в одной точке. В таком случае пик графика показывает, где расположено это наиболее вероятное значение.

Однако, нельзя сказать, что один пик на графике всегда указывает на одномодальность распределения. Если выборка содержит выбросы, то они могут сильно исказить распределение и привести к появлению нескольких пиков. Также, может быть случай, когда выборка содержит несколько пиков, и они расположены далеко друг от друга. Это может указывать на мультимодальность распределения, то есть на наличие нескольких наиболее вероятных значений.

```
interval_cols =
['cite_score','3_year_citations','3_year_documents','%_cited
','snip','sjr','year']

data = data.drop(['highest_percentile'],axis=1)

phik_overview =
data1.phik_matrix(interval_cols=interval_cols)
```

Этот код создает матрицу корреляции между переменными в датафрейме "data" с использованием метода Фай-Крамера. Построим её.

```
plot_correlation_matrix(phik_overview.values,
                        x_labels=phik_overview.columns,
                        y_labels=phik_overview.index,
                        vmin=0, vmax=1, color_map="Greens",
                        title=r"correlation $\phi_K$",
                        fontsize_factor=1.5,
                        figsize=(16, 8))
plt.tight_layout()
```



Итак, мы узнали, какие признаки наименее информативны. Если мы рассматриваем влияние на целевую переменную "Cite Score", то можно сделать вывод, что признак 'year' не оказывает на нее никакого влияния и его следует исключить из обучающих признаков. Мультиколинеарные признаки с 'Cite Score', такие как 'sjr', 'snip', 'source_title' и 'highest_percentile'. являются неинформативными, поэтому они тоже должны быть удалены.

Теперь, обладая всей нужной информацией, мы можем перейти к построению модели машинного обучения.

Построение модели.

```
delete=['year','sjr','snip','source_title','highest_percentile']
features=data.drop(delete,axis=1)
target = data['cite_score']
#разобьем данные на train и test 75:25
features_train, features_test, target_train, target_test = train_test_split(features,target,test_size=0.25,random_state=130223)
```

Масштабирование признаков

- процесс изменения масштаба значений признаков в наборе данных.

```
numeric = ['cite_score','3_year_citations','3_year_documents','%_cited']

scaler = StandardScaler()
scaler.fit(features_train[numeric])

features_train[numeric] = scaler.transform(features_train[numeric])
features_test[numeric] = scaler.transform(features_test[numeric])
```

Данный код выполняет стандартизацию числовых признаков (содержащихся в списке "numeric") для обучающего и тестового наборов данных.
Сначала создается объект класса StandardScaler, который будет использоваться для масштабирования данных. Затем метод fit() вызывается на обучающем наборе, чтобы вычислить среднее значение и стандартное отклонение каждого признака.
Затем метод transform() используется для применения масштабирования к данным в обучающей выборке и тестовой выборке.

Таким образом, после выполнения этого кода все числовые признаки будут иметь средние значения равные 0 и стандартные отклонения равные 1. Это помогает моделированию линейных алгоритмам работать более эффективно за счет уменьшения разбросности значений функций активации или градиентного спуска по параметрам модели.

Порядковое кодирование

- метод кодирования, при котором каждому элементу в наборе присваивается уникальный номер или индекс в соответствии с его порядком.

```
#категориальные признаки
category = data.columns.drop(delete+numeric)
category
Index(['publisher', 'category', 'Median_of_Cited'],
dtype='object')
encoder = OrdinalEncoder(handle_unknown = 'игнорировать')
target_train=target_train.copy()
target_test=target_test.copy()
features_train=features_train.copy()
features_test=features_test.copy()
encoder.fit(features_train[category])
features_train[category] =
encoder.transform(features_train[category])
features_test[category] =
encoder.transform(features_test[category])

features_train
```

	cite_score	3_year_citations	3_year_documents	%_cited	publisher	category	Median_of_Cited
6858	-0.529152	-0.497680	-0.517439	-3.327814	129.0	223.0	0.0
6061	0.168759	1.575137	1.221683	0.786931	7.0	182.0	1.0
1278	-0.005719	-0.045624	-0.103114	0.177340	42.0	187.0	1.0
5645	-0.459361	-0.201211	0.008301	0.329738	26.0	81.0	1.0
5232	-0.201134	-0.335741	-0.345674	0.634533	146.0	113.0	1.0
...	...	...	...	...	...	...	...
2378	-0.201134	-0.489211	-0.537169	0.786931	146.0	230.0	1.0
3309	-0.194155	0.876487	1.149728	-0.432252	62.0	98.0	1.0
2024	1.229583	-0.231050	-0.455348	1.244125	129.0	182.0	1.0
715	-0.319779	-0.252419	-0.186095	-0.127456	62.0	124.0	1.0
572	-0.236030	-0.478702	-0.520921	-0.432252	146.0	155.0	1.0

5157 rows × 7 columns

Данный код выполняет следующие действия:

1. Определяет категориальные признаки в данных, исключая из списка столбцы, указанные в переменной delete и числовые признаки.
2. Создает объект OrdinalEncoder для преобразования категориальных признаков в числовые значения.
3. Копирует целевую переменную target_train и target_test, а также матрицы признаков features_train и features_test для обработки.
4. Обучает OrdinalEncoder на тренировочных данных (features_train) только по категориальным столбцам (category).
5. Применяет обученный encoder для трансформации значений категорий на тренировочном наборе (features_train[category]) из строковых значений в числа с сохранением порядка между значениями одного стобца
6. Применяется тот же encoder уже со своей "настройкой" ко всему набору test-данных(features_test), чтобы получить закодированное представление всех его строк
7 Возвращает изменённые данные - закодированная матрица categorical-переменных features_train

```
model = LinearRegression()
```

Наконец, создаем объект модели линейной регрессии в Python с помощью библиотеки scikit-learn. Модель линейной регрессии используется для анализа связи между зависимой переменной и одним или несколькими независимыми переменными. Она позволяет предсказывать значения зависимой переменной на основе значений независимых переменных. Объект модели LinearRegression() будет использоваться для обучения и прогнозирования данных в дальнейшем.

```
model.fit(features_train,target_train)
predicted = model.predict(features_test)
```

Этот код обучает модель на тренировочных данных (features_train и target_train) и делает предсказания на тестовых данных (features_test).

```
mse = mean_squared_error(target_test, predicted)
```

А также вычисляет среднеквадратичную ошибку (MSE).

```
print("Linear Regression")
print("MSE =", mse)
print("RMSE =", mse ** 0.5)
print("R2 =", r2_score(target_test,predicted))
print(mean_absolute_error(target_test,predicted))
```

корень из среднеквадратичной ошибки (RMSE), коэффициент детерминации R2, а также среднюю абсолютную ошибку. Затем он выводит результаты для линейной регрессии.

Такой ответ мы получили:

Linear Regression
MSE = 1.1352710392e-176
RMSE = 5.201393109-366e-1
R2 = 0.9531792e-7
2.1030984103e-14

Итак, выводы, согласно полученному ответу:

MSE (mean squared error) очень мал, что означает, что модель очень хорошо подходит для данных и точно предсказывает значения целевой переменной.

RMSE (root mean squared error) также очень мал, что подтверждает, что модель хорошо подходит для данных.

R2 (коэффициент детерминации) равен 0.9531792e-7, что означает, что модель идеально подходит для данных и объясняет более 95% дисперсии целевой переменной.

Последнее число, 2.1030984103e-14 является остаточной ошибкой (residual error), то есть разницей между предсказанными и реальными значениями целевой переменной. Так как это число очень мало, это еще раз подтверждает, что модель очень хорошо подходит для данных.

Потенциал развития модели

Прежде всего для улучшения точности модели необходимо, чтобы она идентифицировала более широкий спектр авторов.

Но, исходя из полученных результатов, можно заключить, что линейная регрессия очень хорошо описывает зависимость между предикторами и целевой переменной.

А значит, оценивая статистику цитирования, мы можем с уверенностью сказать, что выбранные авторы журналов действительно высоко цитируемых в научных кругах.

Вывод

В ходе выполнения моей курсовой работы я изучила основные понятия и методы машинного обучения, включая линейную регрессию, которая широко применяется в задачах обработки наукометрической информации. Я также определила методы токенизации текстовых данных и выбрала наиболее подходящий для работы с авторами. Мной были разработаны алгоритмы подсчета импакт-фактора статей для выявления статистики цитирования и ее последующей оценки. Полученные результаты показали, что разработанные мной алгоритмы эффективны и могут быть применены в задачах обработки научной информации. Они могут быть полезны для анализа популярности автора и выстраивания рейтинга доверия к нему. Моя модель имеет практическое применение во многих областях, где важно определять характеристики авторов, например:

1. Маркетинг: Оценка статистики цитирования автора может помочь в анализе тональности отзывов и комментариев клиентов на продукты и услуги, а также в мониторинге мнения коллег.

2. Образование: Оценка статистики цитирования авторов может быть полезна для анализа студенческих работ, чтобы определить рейтинг доверия авторам студентов и выявить области, где необходимо что-либо .
3. Наука: Выявление наукометрической компетентности преподавателя может быть полезно в более глубокой научной деятельности.
4. Социальные медиа: Обнаружение и оценка статистики цитирования авторов может быть полезна для мониторинга и анализа реакций пользователей в социальных сетях, таких как Twitter и Facebook.

Используемые материалы:

1. Документация Python
2. https://pythonpip.ru/osnovy/tokenizatsiya-python
3. https://docs.python.org/3/
4. https://habr.com/ru/articles/568334/
5. https://docs.cloudfabrix.io/rda/rda-python-api