Cluster Innovation Centre


# Analysis of Research in the Field of Artificial Intelligence

**By: Ridhwan Luthra**

# Introduction

Artificial Intelligence is a multidisciplinary field of study which has to do with the ability of machines to show human intelligence like capabilities. They are many components related to the field of artificial intelligence that are in constant research. A few major areas of research include the approach of study, tools of study, applications, ethics, etc.

There are many approaches to the study of artificial intelligence such as cybernetics, symbolic, statistical, etc. Some of the tools widely used in the field include neural networks, reinforcement learning, control theory, logic, etc. Some common applications include health care, self-driving cars, economics, business, etc.

In this paper we give an analysis of the various research going on in the field of artificial intelligence.

# Acknowledgement

I express my deep sense of gratitude to our respected and learned mentor Dr. Dorje Dawa who taught us History, culture & civilization, which also helped me in doing a lot of research and I came to know about so many new things. I am really thankful to him for showing me a way to work and learn simultaneously.
Secondly, I would also like to thank my parents and friends who helped me a lot in finishing the project within the specified time limit. Without their valuable suggestions and moral support this work would not have been in its present shape.

# Source of Data

## Scopus

Scopus is Elsevier's abstract and citation database. It contains all the peer reviewed journals that are of high quality, it covers three types of sources: Book Series, Journals and Trade Journals. All journals published in scopus are reviewed each year to ensure high-quality standards are maintained. There are a number of indexes that are published such as h-index, citescore, etc. Scopus features smart tools to analyse research and has an overview of research output in various fields like humanities, science, IT etc. You can keep a track of global, interdisciplinary and collaborative research works.

## DOI

A Digital Object Identifier or DOI is a persistent identifier or handle used to uniquely identify objects, standardized by the International Organization for Standardization ISO. DOIs are in wide use mainly to identify academic, professional, and government information, such as journal articles, research reports and data sets, and official publications though they also have been used to identify other types of information resources, such as commercial videos.The DOI system provides a technical and social infrastructure for the registration and use of persistent interoperable identifiers, called DOIs, for use on digital networks.

## Elsevier API

The website https://dev.elsevier.com provides an API using which it is straightforward to mine data from the scopus database. It allows us to search any keyword, get all the metadata related to each and every entry.

# Data Collection

Data Collection is the process of extracting data from a repository that can be used later for other analyses.

## Python SDK (elsapy)

A Python module for use with api.elsevier.com. Its aim is to make life easier for people who are not primarily programmers, but need to interact with publication and citation data from Elsevier products in a programmatic manner (e.g. academic researchers). The module consists of the following classes:

- ElsClient: represents a client interface to api.elsevier.com.
- ElsEntity: an abstract class representing an entity in the Elsevier (specifically, Scopus) data model. ElsEntities can be initialized with a URI, after which they can read their own data from api.elsevier.com through an ElsClient instance. ElsEntity has the following descendants:
    - elsProf: an abstract class representing a *profiled* entity in Scopus. This class has two descendants:
        - ElsAuthor: represent the author of one or more documents in Scopus.
        - ElsAffil: represents an affiliation (i.e. an institution authors are affiliated with) in Scopus
    - AbsDoc: represents a document in Scopus (i.e. abstract only). This document typically is the record of a scholarly article in any of the journals covered in Scopus.
    - FullDoc: represents a document in ScienceDirect (i.e. full text). This document is the full-text version of a scholarly article or book chapter from a journal published by Elsevier.
- Each ElsEntity (once read) has a .data attribute, which contains a JSON/dictionary representation of the object's data. Use the object's .data.keys() method to list the first-level keys in the dictionary; drill down from there to explore the data.
- ElsAuthor and ElsAffil objects also have a method, .readDocs(), that tells it to retrieve all the publications associated with that author/affiliation from Elsevier's API, and store it as a list attribute, .doc_list. Each entry in the list is a dictionary containing that document's metadata.
- ElsSearch: represents a search through one of Elsevier's indexes, which can be a document index (ScienceDirect or Scopus), an author index, or an

affiliation index. Once executed, each search object has a list attribute, .results, that contains the results retrieved from Elsevier's APIs for that search. Each entry in the list is a dictionary containing that result's metadata.(ElsevierDev, n.d.)

# Collection Procedure

## Generating the API Key

Visit http://dev.elsevier.com to generate your api key.

## Create a client object using the api key

```
## Load configuration
con_file = open("config.json")
config = json.load(con_file)
con_file.close()

## Initialize client
client = ElsClient(config['apikey'])
```

## Search for the keywords

```
## Initialize doc search object and execute search, retrieving all results in the scopus index
doc_srch = ElsSearch('artificial+intelligence','scopus')
doc_srch.execute(client, get_all = True)
## Initialize doc search object and execute search, retrieving all results in scidir
doc_srch = ElsSearch('artificial+intelligence','scidir')
doc_srch.execute(client, get_all = True)
```

NOTE: The actual implementation of execute function on elsapy allows only 5000 results. The implementation was modified to extract all the results.

# Analysis of Data

## Keyword Trend Analysis

In this section we analyse the number of papers published each year when we search for the keyword “Artificial intelligence”.

### Programming

This Information is extracted by scraping google scholar website using the script given below. It has been adapted from the work by (Pold, n.d.)

```
from bs4 import BeautifulSoup
import urllib.request, urllib.parse, urllib.error
from urllib.request import Request, build_opener, HTTPCookieProcessor
from http.cookiejar import MozillaCookieJar
import re
import time
import sys

def get_num_results(search_term, start_date, end_date):
    """
    Helper method, sends HTTP request and returns response payload.
    """

    # Open website and read html
    user_agent = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/48.0.2564.109 Safari/537.36'
    query_params = { 'q' : search_term, 'as_ylo' : start_date, 'as_yhi' :
end_date}
    url = "https://scholar.google.com/scholar?as_vis=1&hl=en&as_sdt=1,5&" +
urllib.parse.urlencode(query_params)
    opener = build_opener()
    request = Request(url=url, headers={'User-Agent': user_agent})
    handler = opener.open(request)
    html = handler.read()
```

```
    # Create soup for parsing HTML and extracting the relevant information
    soup = BeautifulSoup(html, 'html.parser')
    div_results = soup.find("div", {"id": "gs_ab_md"}) # find line 'About x
results (y sec)

    if div_results != None:
        res = re.findall(r'(\d+),?(\d+)?\s', div_results.text) # extract
number of search results
        num_results = ''.join(res[0]) # convert string to number
        success = True
    else:
        success = False
        num_results = 0

    return num_results, success


def get_range(search_term, start_date, end_date):

    fp = open("out.csv", 'w')
    fp.write("year,results\n")
    print("year,results")

    for date in range(start_date, end_date + 1):

        num_results, success = get_num_results(search_term, date, date)
        if not(success):
            print("Too many requests made")
            break
        year_results = "{0},{1}".format(date, num_results)
        print(year_results)
        fp.write(year_results + '\n')
        time.sleep(0.8)

    fp.close()
```

```
if __name__ == "__main__":

    if len(sys.argv) < 3:
        print("Error”)

    else:
        search_term = sys.argv[1]
        start_date = int(sys.argv[2])
        end_date = int(sys.argv[3])
        html = get_range(search_term, start_date, end_date)
```

## Graphical Representation

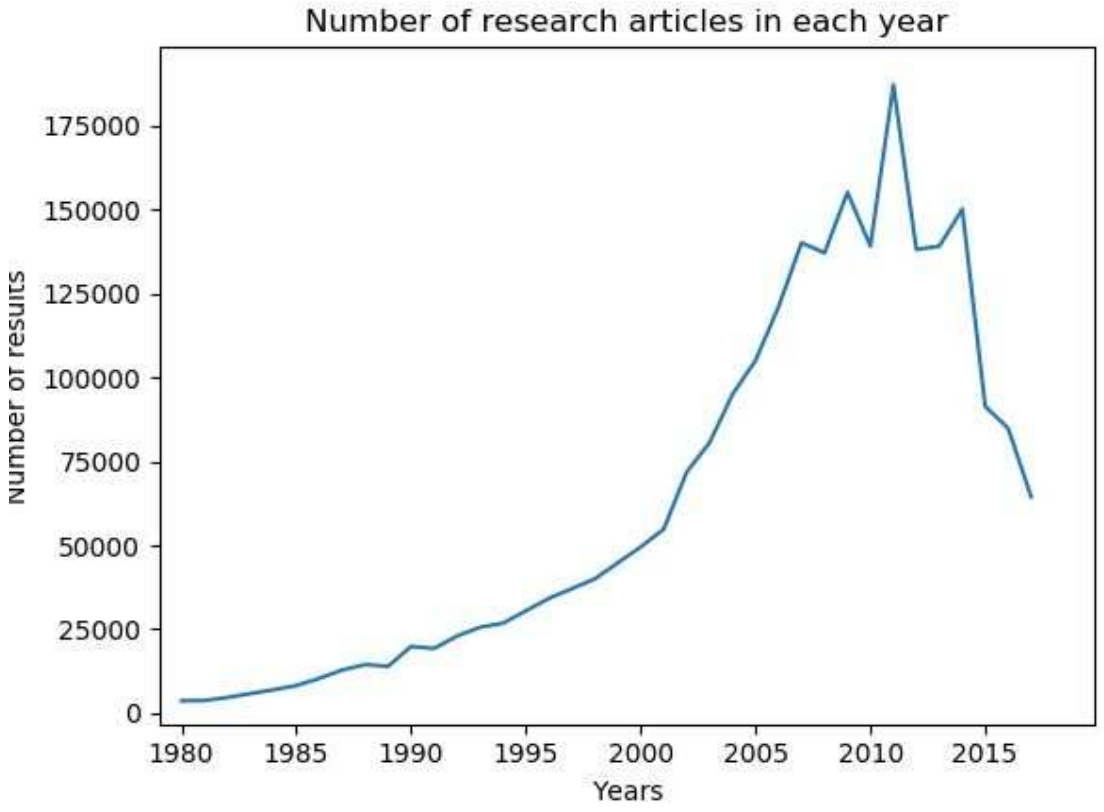
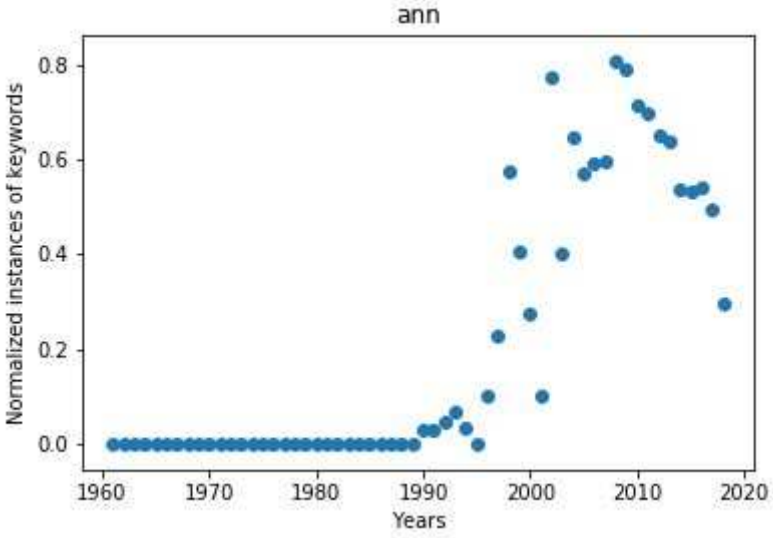


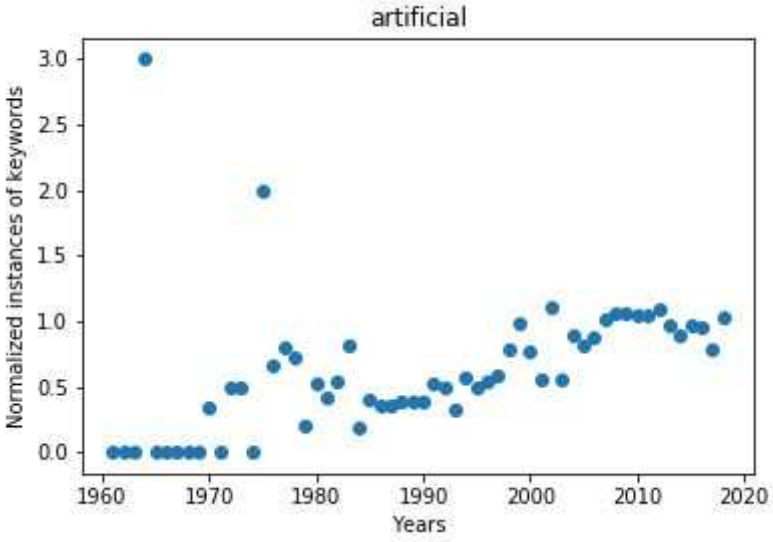

Figure1: This graph shows the number of articles published each year

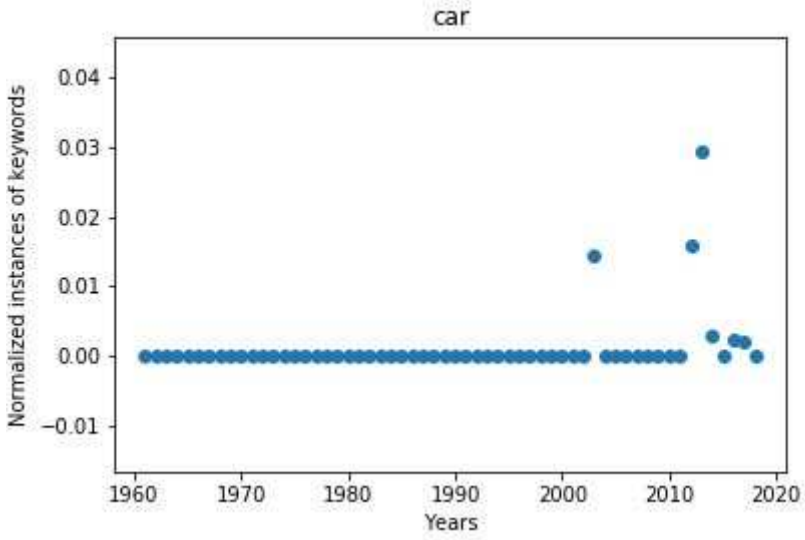
The graph shows an interesting trend. The general assumption would be that the number of research publications each year must be expanding but that is not the case.
Graph is generated using Matplotlib, (Droettboom et al., 2014).

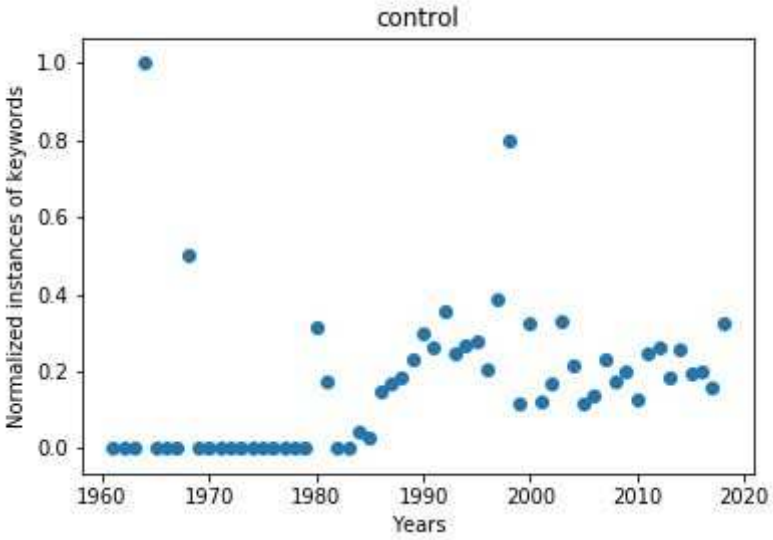
## Keyword Analysis

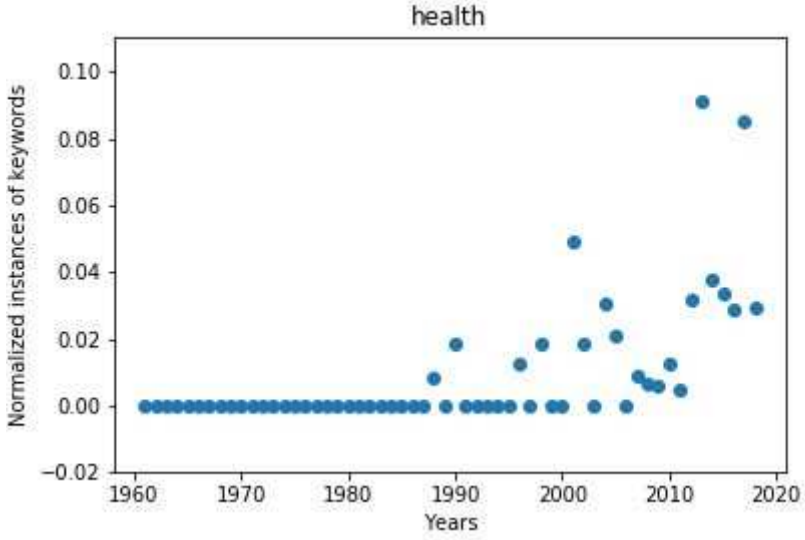
This includes the analysis of the various keywords used in the abstracts. Some keywords on which the analysis can be done is "artificial, intelligence, logic, health, car, identification, strategy, vision, computer, science, art, learning, neural, network"
A list of 300 keywords has been used for this analysis.
A few plots that show interesting trends have been presented here.

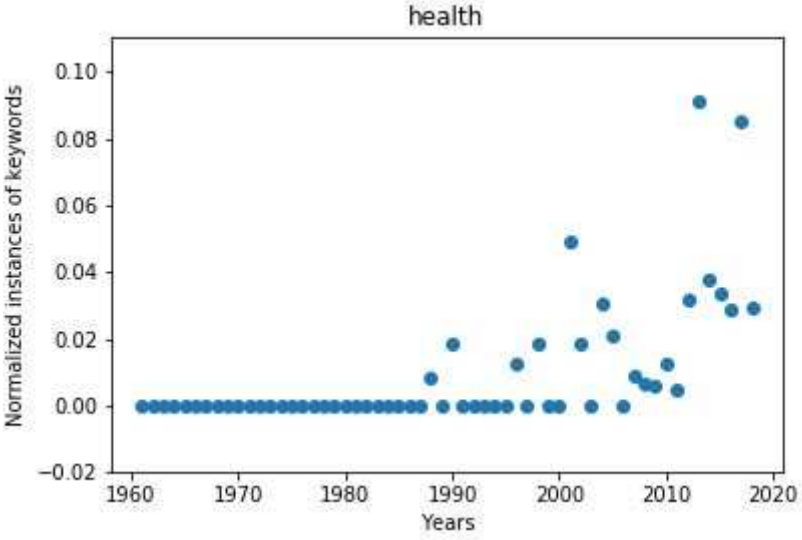


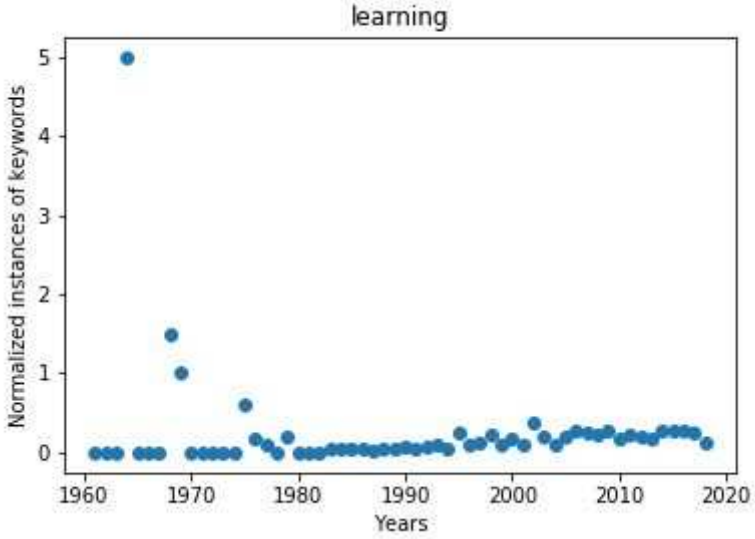



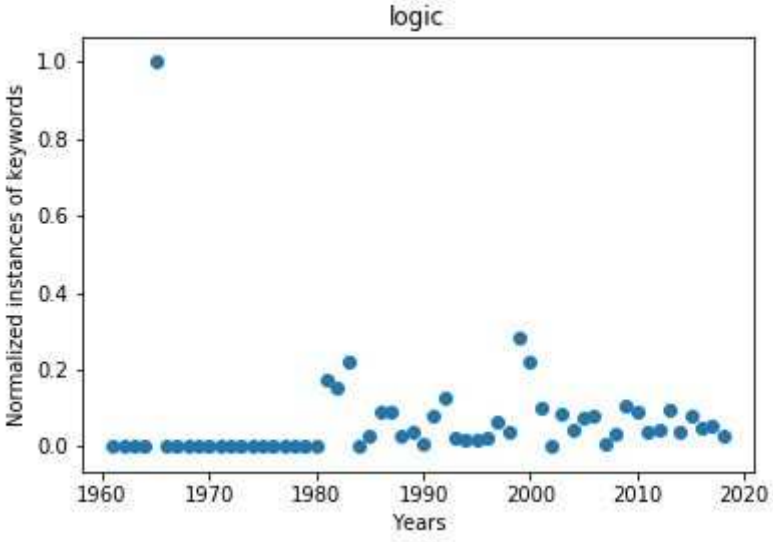

car
Normalized instances of keywords
0.04
0.03
0.02
0.01
0.00
−0.01
1960
1970
1980
1990
2000
2010
2020
Years

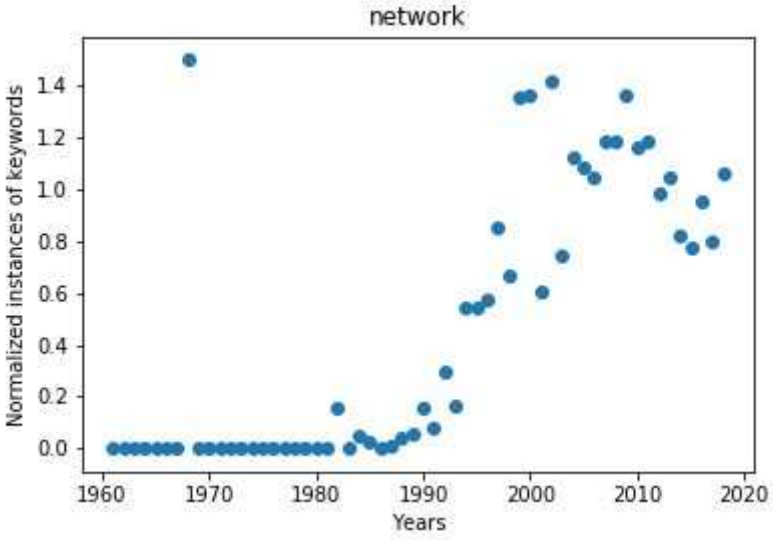

control
Normalized instances of keywords
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Years

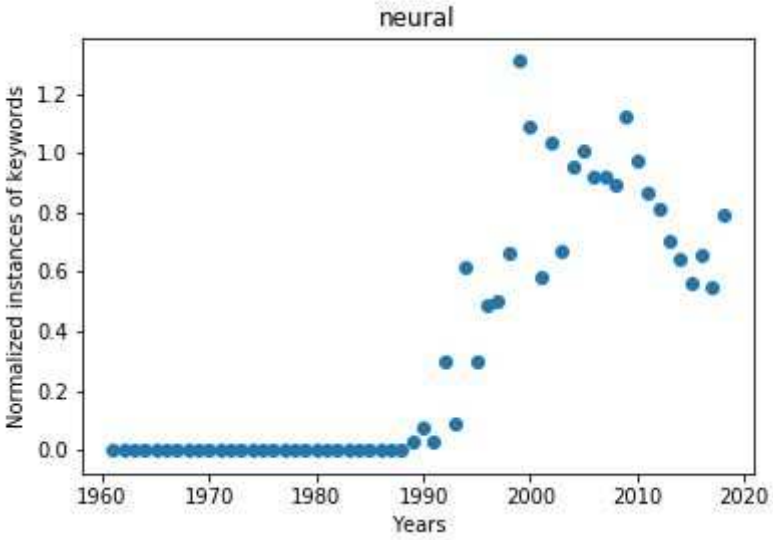

health
Normalized instances of keywords
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
1960
1970
1980
1990
2000
2010
2020
Years

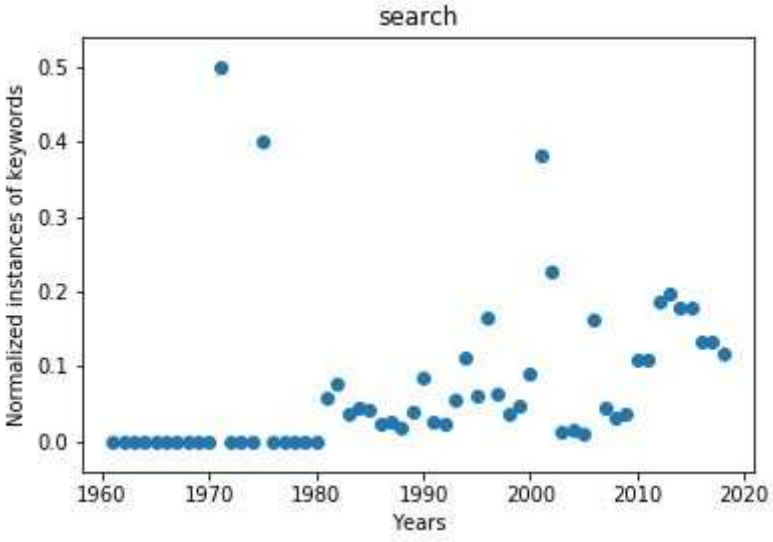

health
Normalized instances of keywords
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
1960
1970
1980
1990
2000
2010
2020
Years

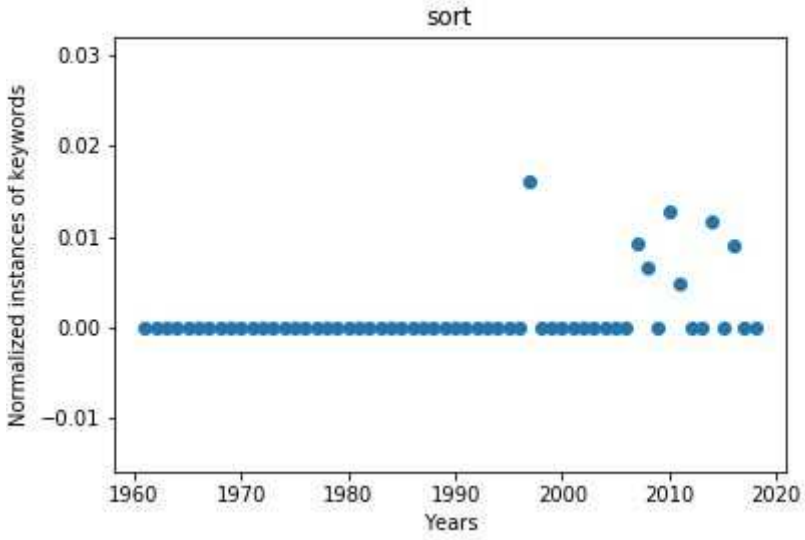

learning
Normalized instances of keywords
5
4
3
2
1
0
1960
1970
1980
1990
2000
2010
2020
Years

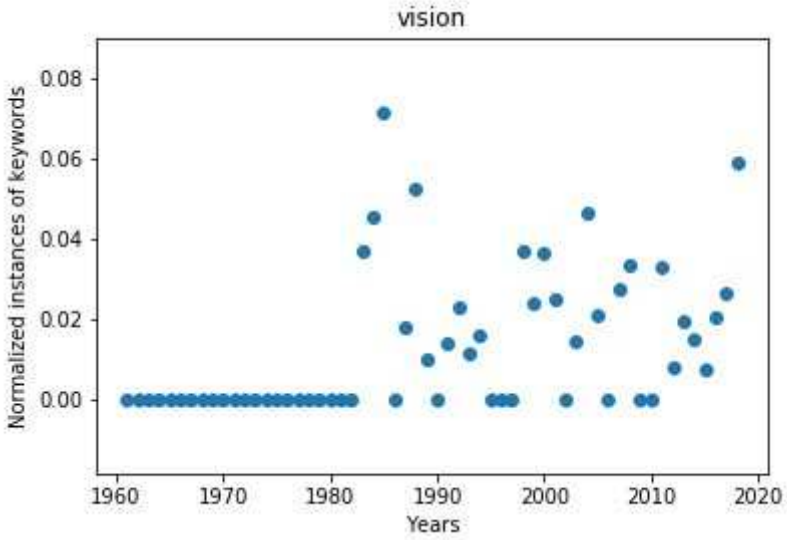

logic
Normalized instances of keywords
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Years


network
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1960
1970
1980
1990
2000
2010
2020
Years


neural
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1960
1970
1980
1990
2000
2010
2020
Years

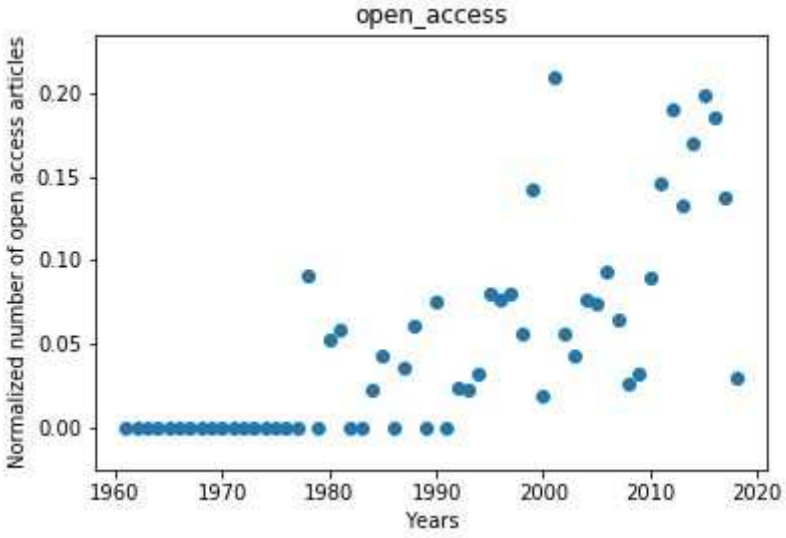

search
Normalized instances of keywords
0.0
0.1
0.2
0.3
0.4
0.5
1960
1970
1980
1990
2000
2010
2020
Years

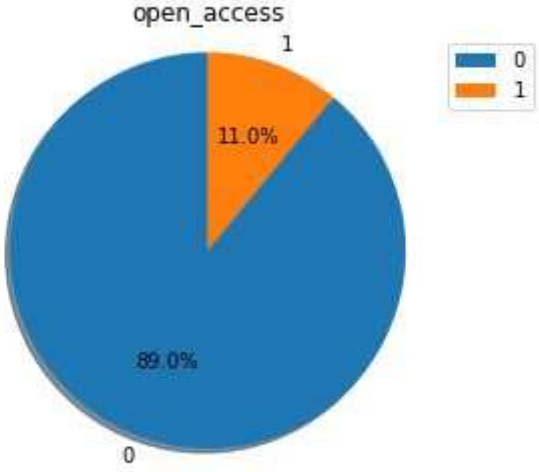


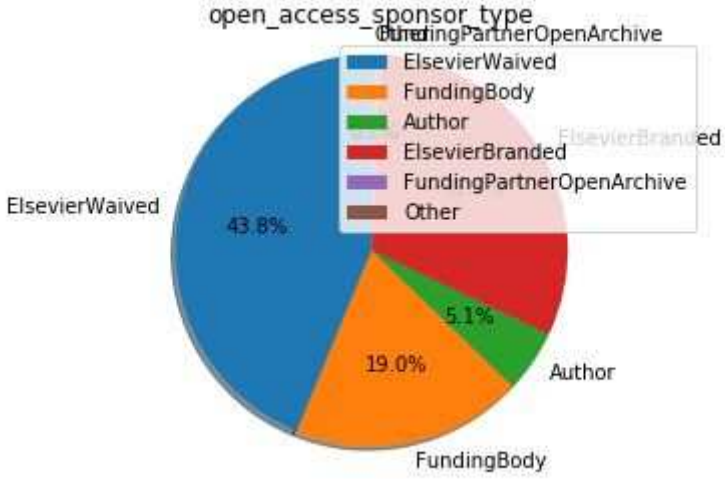



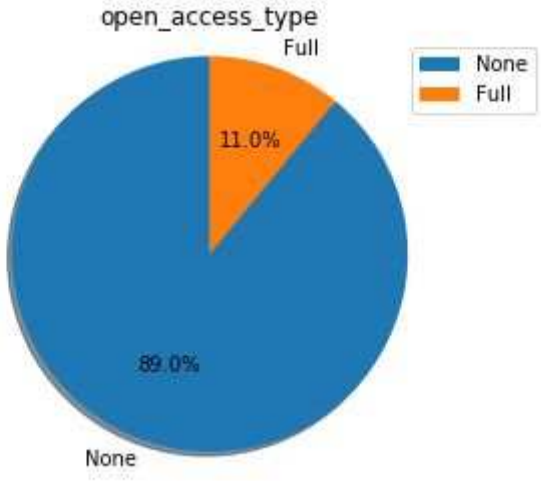

## Other Analysis

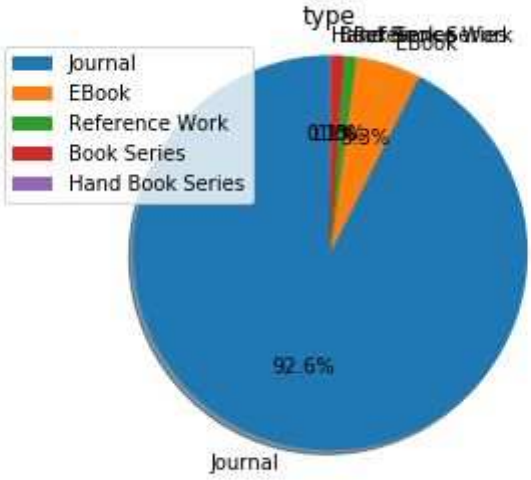
There are many other forms of analysis that are possible with the data on various grounds such as yearly, journal-wise, etc.
Over here an Analysis of open_access journals is given.
The distribution of sponsor_type, access_type, etc. is shown to see the distribution of the data.


open_access
Normalized number of open access articles
0.20
0.15
0.10
0.05
0.00
1960
1970
1980
1990
2000
2010
2020
Years


open_access
1
0
1
11.0%
89.0%
0


open_access_sponsor_type
ElsevierWaived
FundingBody
Author
ElsevierBranded
FundingPartnerOpenArchive
Other
43.8%
19.0%
5.1%
ElsevierWaived
Author
FundingBody




## Implementation

The Implementation is attached at the end of the report.

# References

Digital Object Identifier System. (n.d.). Retrieved November 7, 2017, from https://www.doi.org

Droettboom, M., Hunter, J., Firing, E., Caswell, T. A., Dale, D., Lee, J.-J., … Würtz, P. (2014). Matplotlib version 1.4.0. https://doi.org/10.5281/zenodo.11451

ElsevierDev. (n.d.). ElsevierDev/elsapy. Retrieved November 7, 2017, from https://github.com/ElsevierDev/elsapy

Pold. (n.d.). Pold87/academic-keyword-occurrence. Retrieved November 7, 2017, from https://github.com/Pold87/academic-keyword-occurrence

# analysis

November 9, 2017

In [2]:
```
import pickle
import numpy as np
import pandas as pd
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
%matplotlib inline
```

## 0.1 Load Unique Data

In [3]:
```
files = ["scidir_metadata_2000.p", "scidir_metadata_2000.p", "scidir_metadata_4950.p"]

rest = []
final = pickle.load(open("scidir_metadata/" + files[0], "rb"))[1]

for file in files[1:]:
    rest.extend(pickle.load(open("scidir_metadata/" + file, "rb"))[1])

print(len(final), len(rest))
```

```
1956 6904
```

In [4]:
```
prev_len_final = len(final) - 1

while prev_len_final != len(final):
    final_ids = [x.id for x in final]
    for each in rest:
        if each.id not in final_ids:
            final.append(each)
    prev_len_final = len(final)
    print(len(final))
```

```
4952
```

## 0.2 Extracting Useful Data

In [5]:
```
data = [x.data["coredata"] for x in final]
temp = data[62]
```

```
        temp.keys()
        # print(temp["prism:issueName"])
        # print(temp["prism:aggregationType"])
```

```
Out[5]: dict_keys(['pii', 'eid', 'pubType', 'prism:copyright', 'prism:pageRange', 'dc:title',
```

### 0.2.1 Testing cell

keep_list = ['pubType', 'dc:title', 'dc:description', 'prism:publisher', 'openaccess', 'dc:creator', 'prism:coverDate', 'prism:issueName', 'prism:doi', 'openaccessType', 'openaccessSponsorName', 'prism:aggregationType', 'openaccessSponsorType', 'prism:publicationName']

```
temp.keys()
for key in temp.keys():
    if key in keep_list:
        print(key, temp[key])
        break
```

```
In [6]: articles = list()

        for dat in data:
            try:
                article_dict = dict()
            #     article_dict["pub_type"] = dat["pubType"]
                article_dict["title"] = dat["dc:title"]
                article_dict["abstract"] = dat["dc:description"]
        #         article_dict["publisher"] = dat["prism:publisher"]
        #         article_dict["authors"] = [author["$"] for author in dat["dc:creator"]]
                article_dict["cover_date"] = dat["prism:coverDate"]
            #     article_dict["issue_name"] = dat["prism:issueName"]
                article_dict["doi"] = dat["prism:doi"]
                article_dict["open_access"] = dat["openaccess"]
                article_dict["open_access_type"] = dat["openaccessType"]
                article_dict["open_access_sponsor_name"] = dat["openaccessSponsorName"]
                article_dict["type"] = dat["prism:aggregationType"]
                article_dict["open_access_sponsor_type"] = dat["openaccessSponsorType"]
                article_dict["publication_name"] = dat["prism:publicationName"]
            except KeyError as key:
                print(key)

            articles.append(article_dict)
```

## 0.3 Analysis

```
In [7]: df = pd.DataFrame(articles)
```

```
In [8]: df.columns
```

```
Out[8]: Index(['abstract', 'cover_date', 'doi', 'open_access',
               'open_access_sponsor_name', 'open_access_sponsor_type',
               'open_access_type', 'publication_name', 'title', 'type'],
              dtype='object')

In [9]: abstracts = " ".join([abstract for abstract in list(df.abstract) if abstract is not No

In [10]: print(len(abstracts))

4714622


In [11]: from nltk.corpus import stopwords
         import nltk
         from nltk.stem import WordNetLemmatizer
         lemmatizer = WordNetLemmatizer()

In [12]: temp = word_tokenize(abstracts)
         word_tokens = []

         for tem in temp:
             if len(tem) > 2:
                 word_tokens.append(lemmatizer.lemmatize(tem.lower()))

In [13]: words_to_ignore = "wa,used,using,paper,based,study,approach,ha,method,result,problem,

         stop_words = stopwords.words("english")
         stop_words += words_to_ignore.split(",")

         words = [w for w in word_tokens if not w in stop_words]

In [15]: freq = nltk.FreqDist(words)

         freq.most_common(20)

Out[15]: [('artificial', 3991),
          ('network', 3838),
          ('algorithm', 3188),
          ('neural', 2920),
          ('data', 2896),
          ('intelligence', 2632),
          ('ann', 2161),
          ('analysis', 1355),
          ('prediction', 1129),
          ('optimization', 1100),
          ('control', 1038),
          ('learning', 933),
          ('function', 868),
          ('input', 865),
```

```
 ('accuracy', 857),
 ('test', 793),
 ('training', 792),
 ('number', 780),
 ('work', 770),
 ('tool', 756)]
```

```
In [16]: for i, date in enumerate(df.cover_date):
             df.cover_date[i] = int(date[:4])
         #     years.append(year)
         df.cover_date
```

```
Out[16]: 0       2018
         1       2017
         2       2017
         3       2017
         4       2017
         5       2017
         6       2017
         7       2017
         8       2017
         9       2017
         10      2017
         11      2018
         12      2017
         13      2017
         14      2017
         15      2017
         16      2018
         17      2017
         18      2017
         19      2017
         20      2018
         21      2017
         22      2017
         23      2017
         24      2017
         25      2017
         26      2017
         27      2017
         28      2017
         29      2017
                 ...
         4922    2008
         4923    2013
         4924    2008
         4925    2008
         4926    2008
```

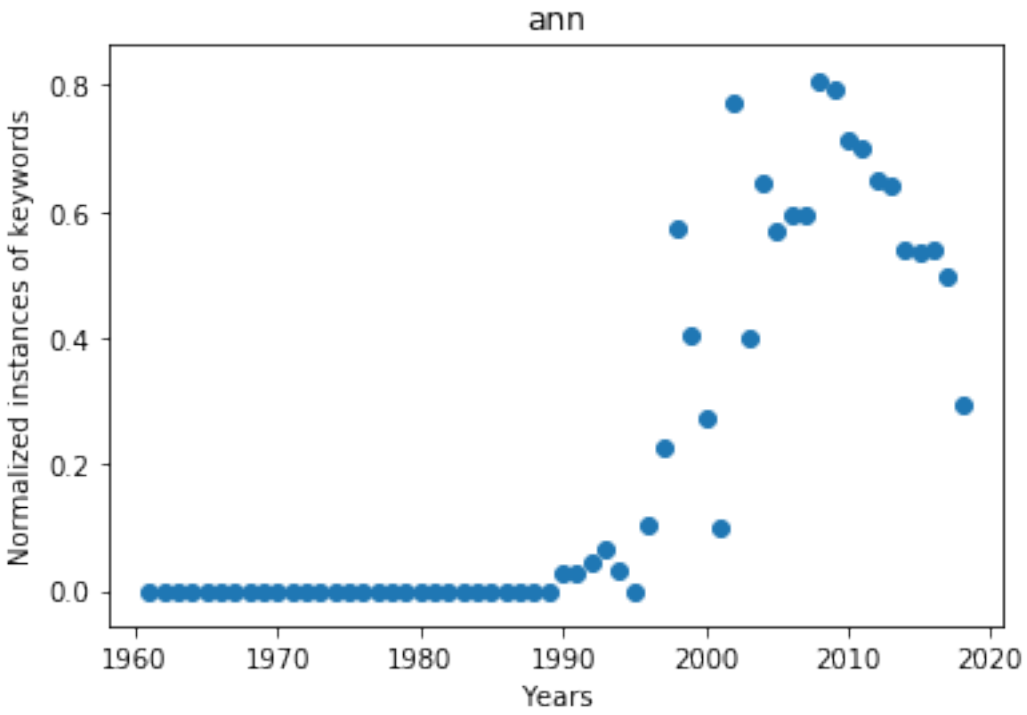
```
4927    2018
4928    2012
4929    2012
4930    2008
4931    2013
4932    2013
4933    1992
4934    1996
4935    2012
4936    2009
4937    2007
4938    2008
4939    1996
4940    2012
4941    2009
4942    1998
4943    1990
4944    1991
4945    2013
4946    2013
4947    2013
4948    1997
4949    1994
4950    2008
4951    2013
Name: cover_date, Length: 4952, dtype: object
```

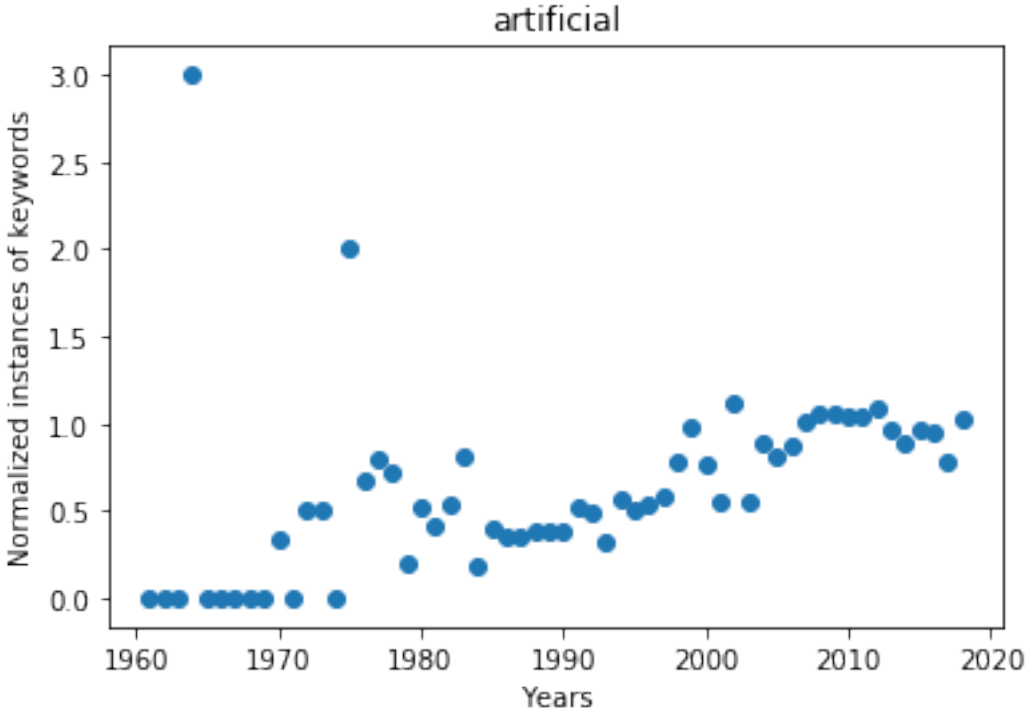
```
In [17]: def process_abstracts(temp_df):
             abstracts = " ".join([abstract for abstract in list(temp_df.abstract) if abstract
             temp = word_tokenize(abstracts)
             word_tokens = []

             for tem in temp:
                 if len(tem) > 2:
                     word_tokens.append(lemmatizer.lemmatize(tem.lower()))

             words_to_ignore = "wa,used,using,paper,based,study,approach,ha,method,result,probl

             stop_words = stopwords.words("english")
             stop_words += words_to_ignore.split(",")

             words = [w for w in word_tokens if not w in stop_words]

             return words
```

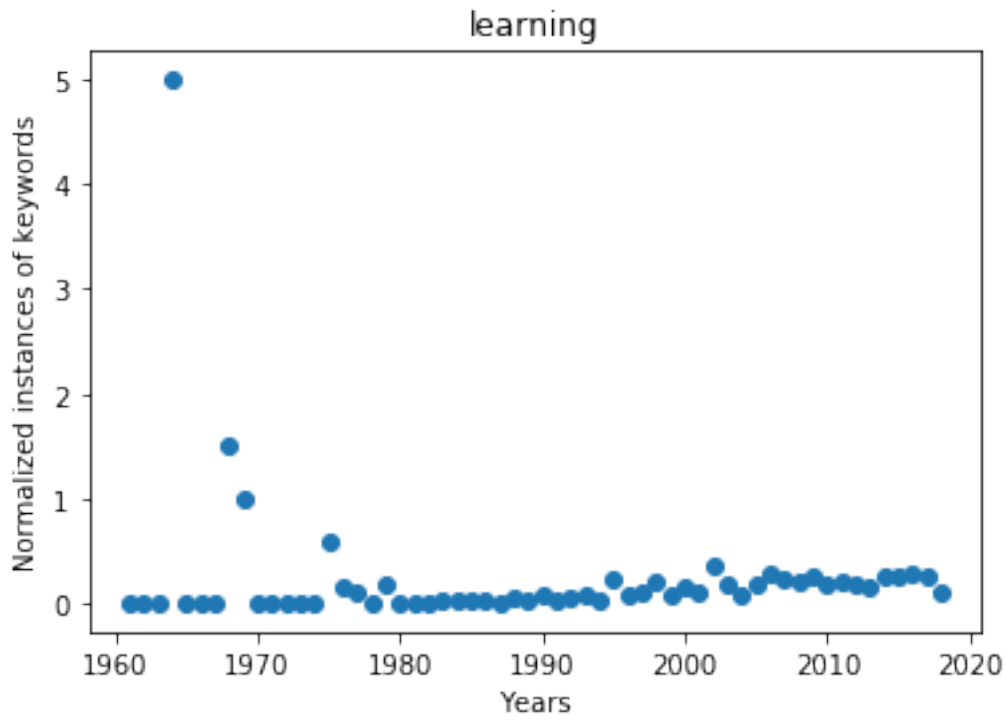
```
In [18]: def word_frequency(df, word):
             year = 1961
             x = []
```

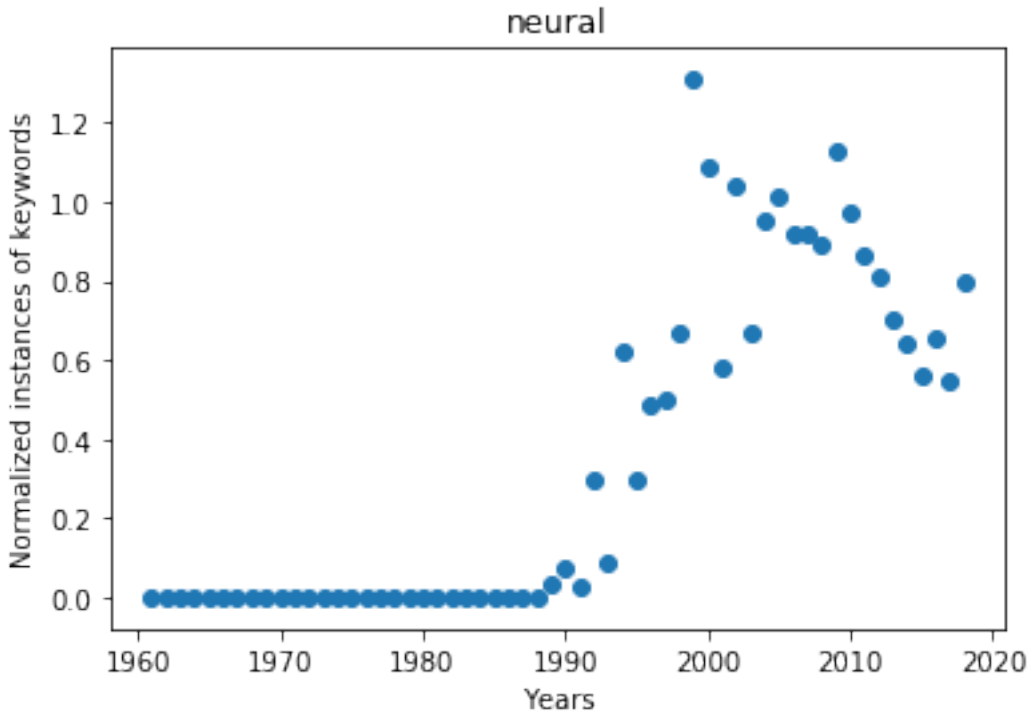
```
    y = []

    while year <= 2018:
        temp_df = df[df.cover_date == year]
        words = process_abstracts(temp_df)
        freq = nltk.FreqDist(words)
        try:
            y.append(freq[word] / len(temp_df))
        except ZeroDivisionError:
            y.append(freq[word])
        x.append(year)
        year += 1

    return x, y

def open_access(df):
    year = 1961
    x = []
    y = []

    while year <= 2018:
        temp_df = df[df.cover_date == year]
        try:
            y.append(sum(temp_df.open_access.astype("int64")) / len(temp_df))
        except ZeroDivisionError:
            y.append(sum(temp_df.open_access.astype("int64")))
        x.append(year)
        year += 1

    return x, y
```

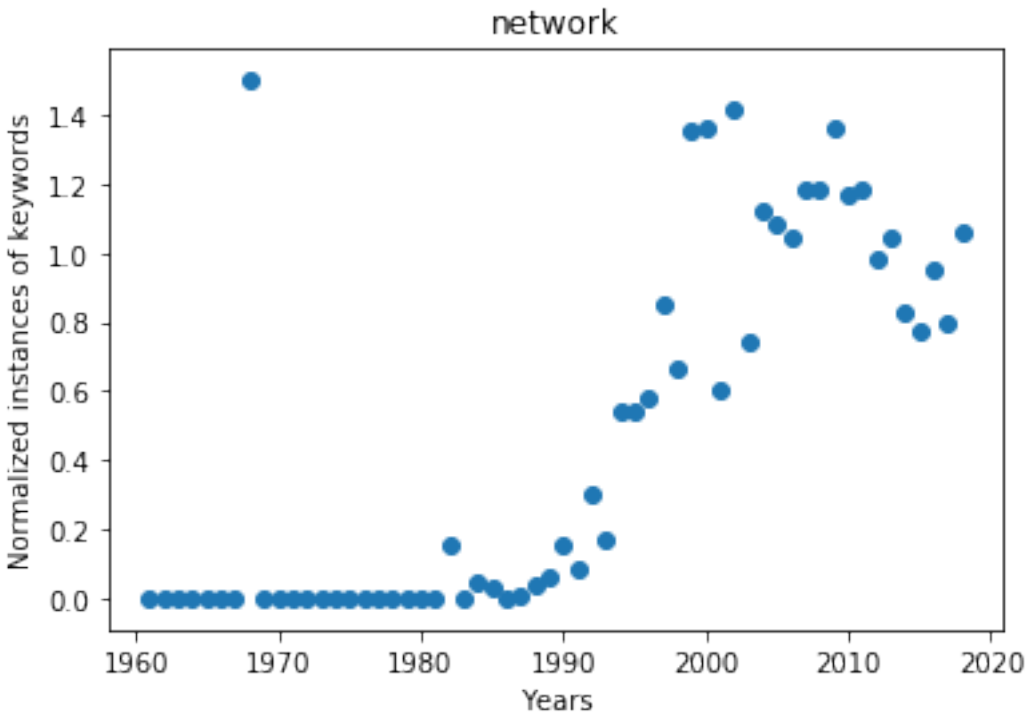
In [31]:
```
plot_list = "ann,artificial,learning,neural,network,control,logic,search,sort,health,
for each in plot_list.split(","):
    x, y = word_frequency(df, each)
    plt.figure()
    plt.scatter(x, y)
    plt.title(each)
    plt.xlabel("Years")
    plt.ylabel("Normalized instances of keywords")
    plt.savefig("plots/" + each + ".png")
    plt.show()
```

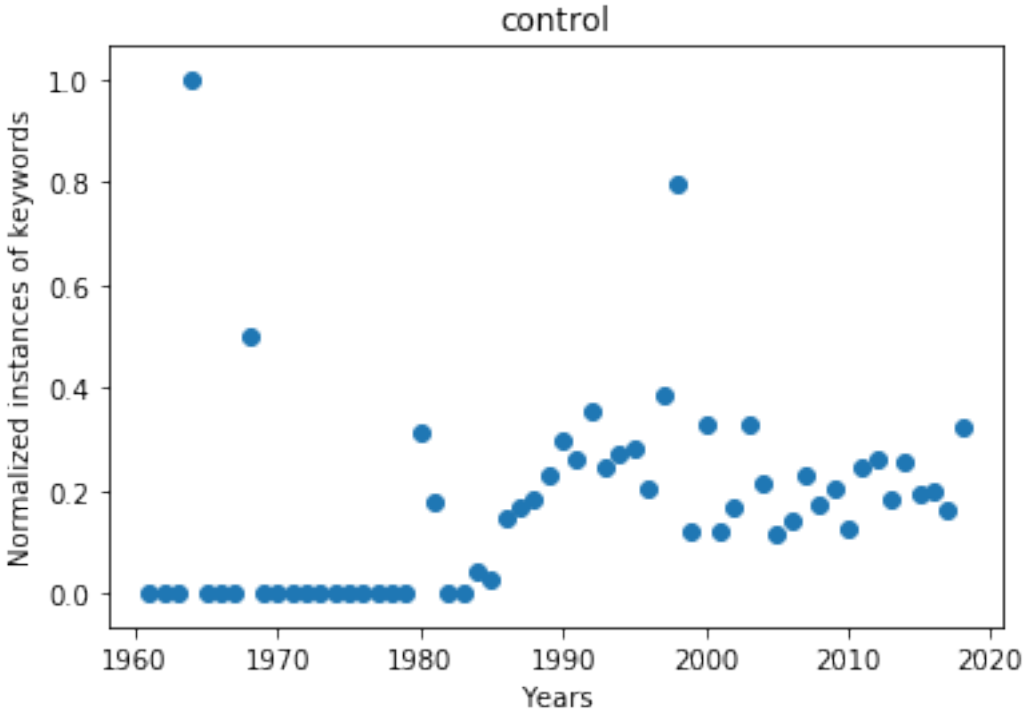

ann
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1960
1970
1980
1990
2000
2010
2020
Years

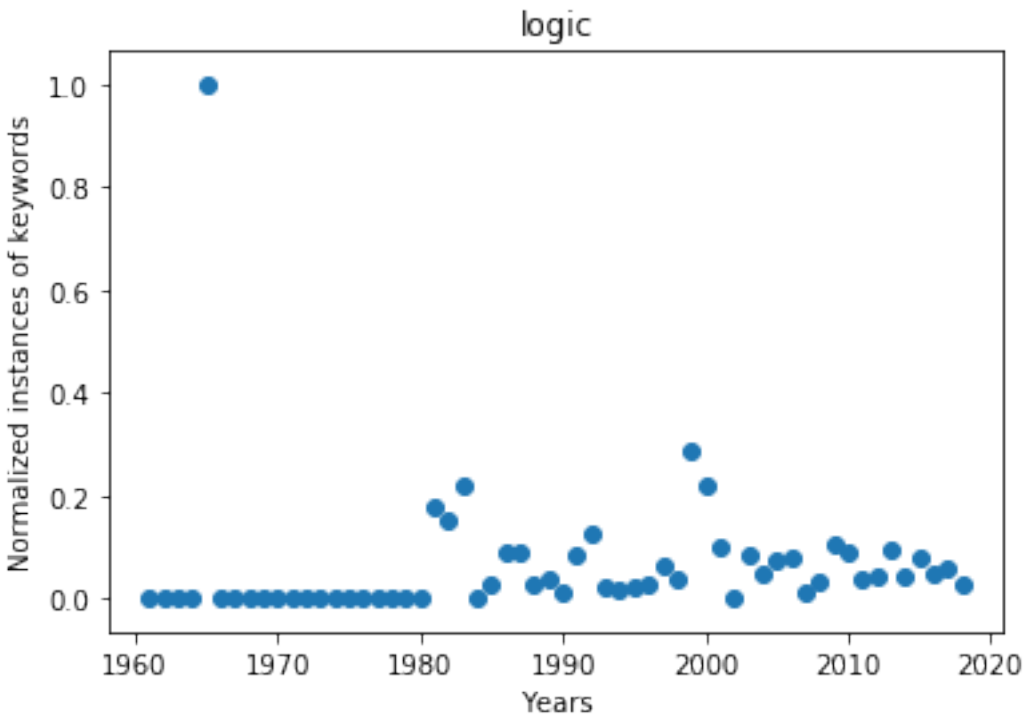

artificial
Normalized instances of keywords
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1960
1970
1980
1990
2000
2010
2020
Years

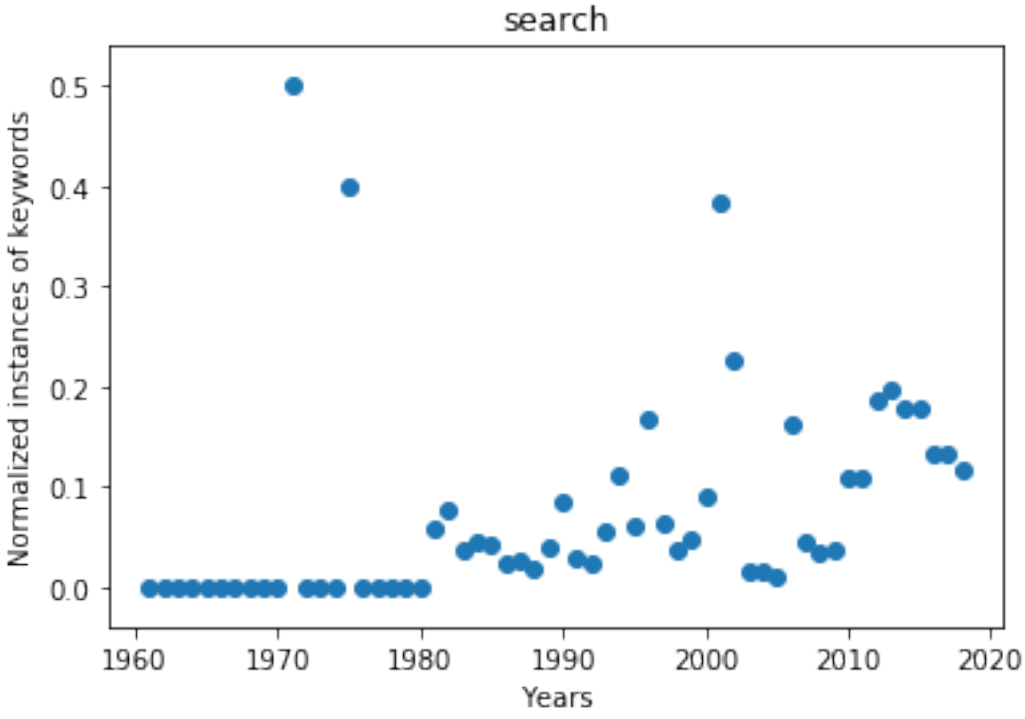

learning
Normalized instances of keywords
0
1
2
3
4
5
1960
1970
1980
1990
2000
2010
2020
Years

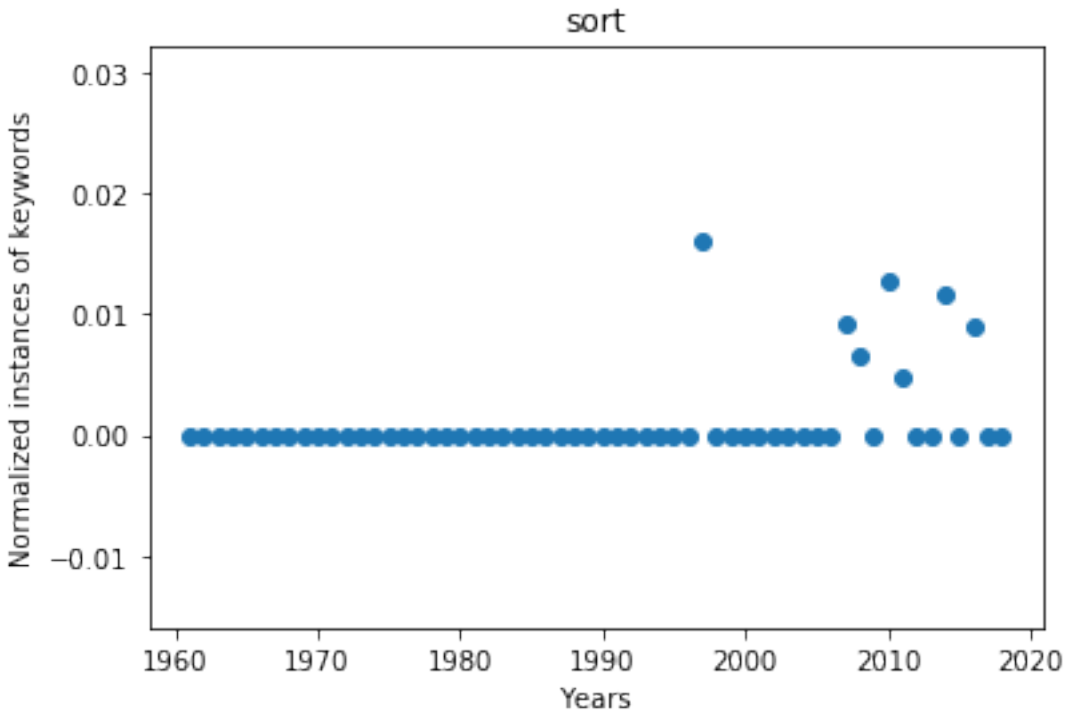

neural
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1960
1970
1980
1990
2000
2010
2020
Years

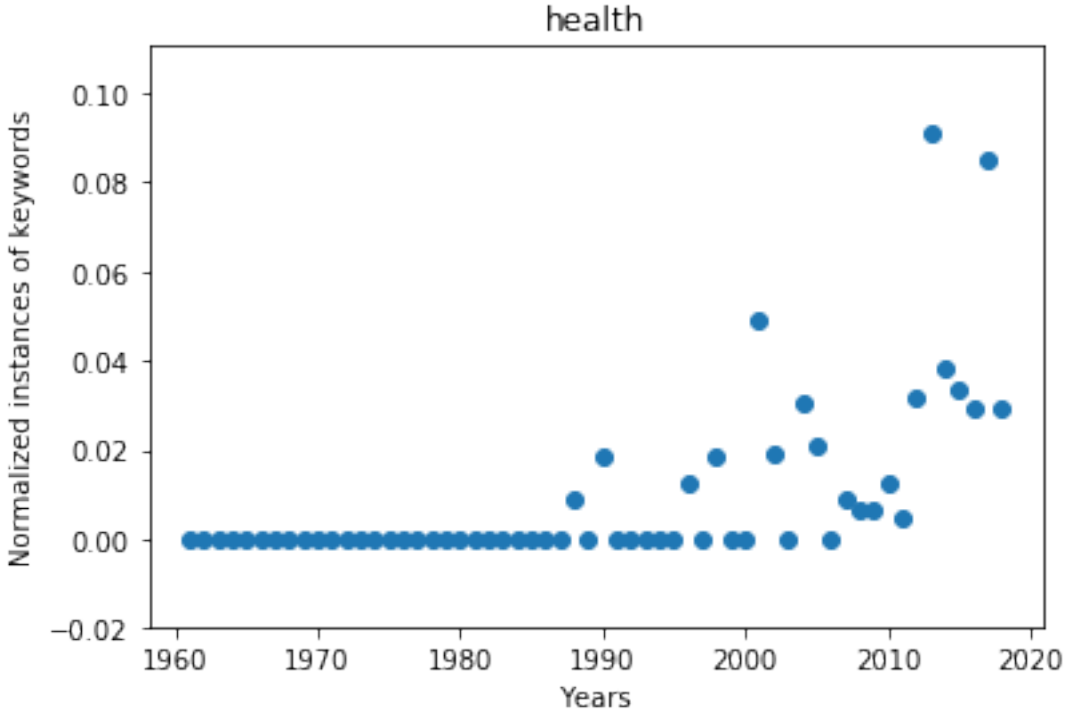

network
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1960
1970
1980
1990
2000
2010
2020
Years

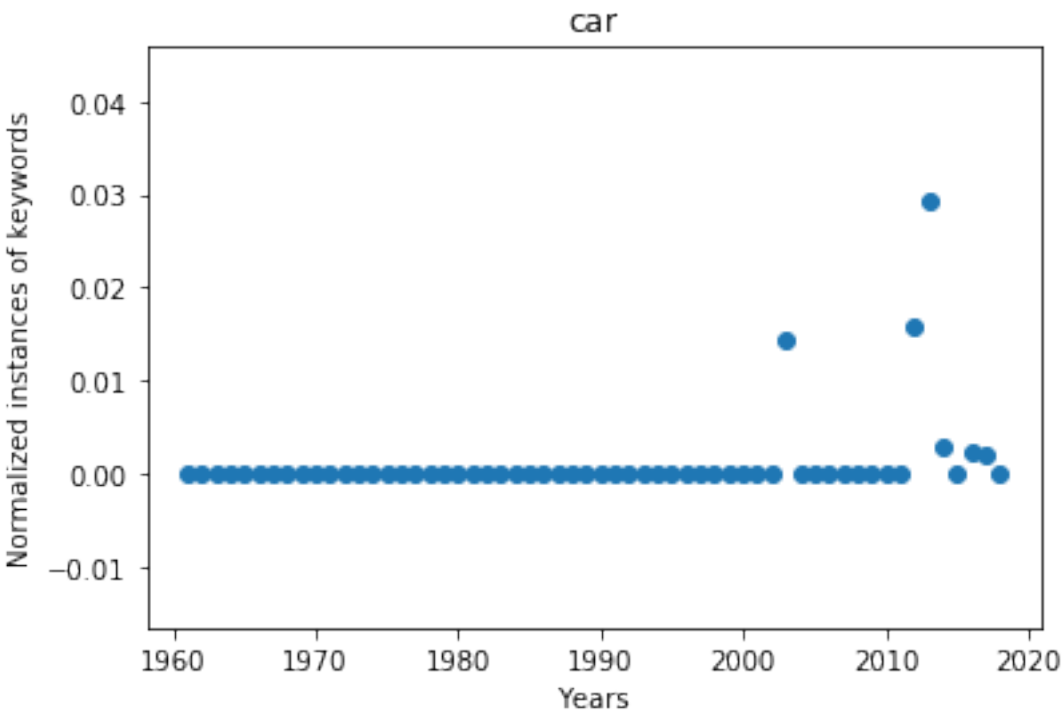

control
Normalized instances of keywords
0.0
0.2
0.4
0.6
0.8
1.0
1960
1970
1980
1990
2000
2010
2020
Years

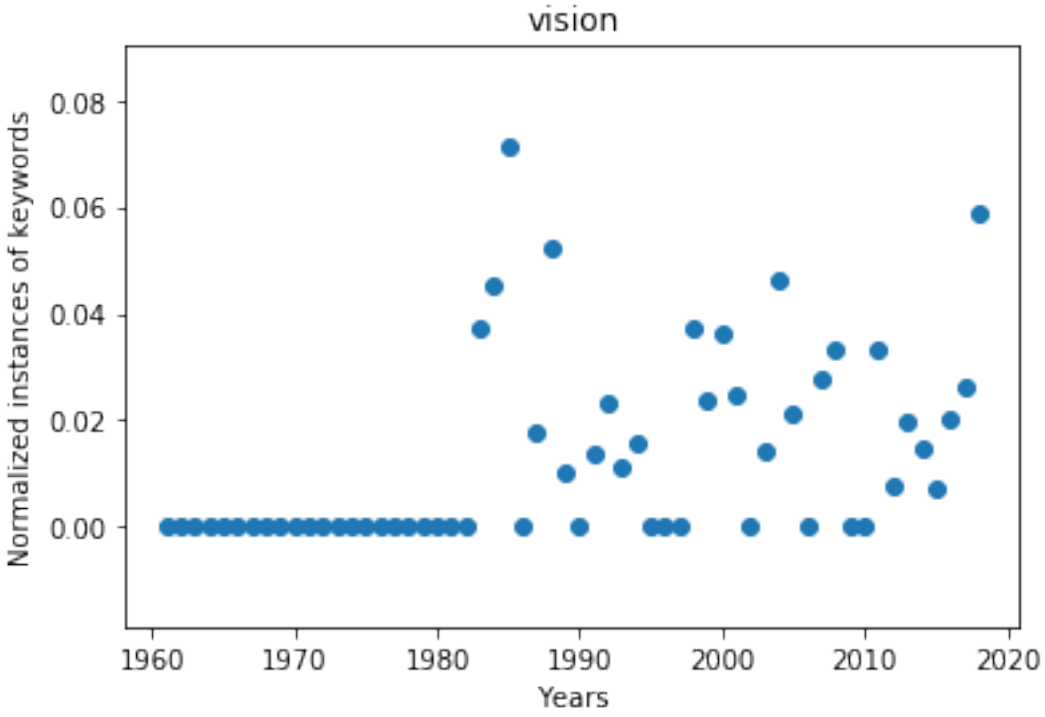

logic
Normalized instances of keywords
1.0
0.8
0.6
0.4
0.2
0.0
1960
1970
1980
1990
2000
2010
2020
Years


search
Normalized instances of keywords
0.5
0.4
0.3
0.2
0.1
0.0
1960
1970
1980
1990
2000
2010
2020
Years

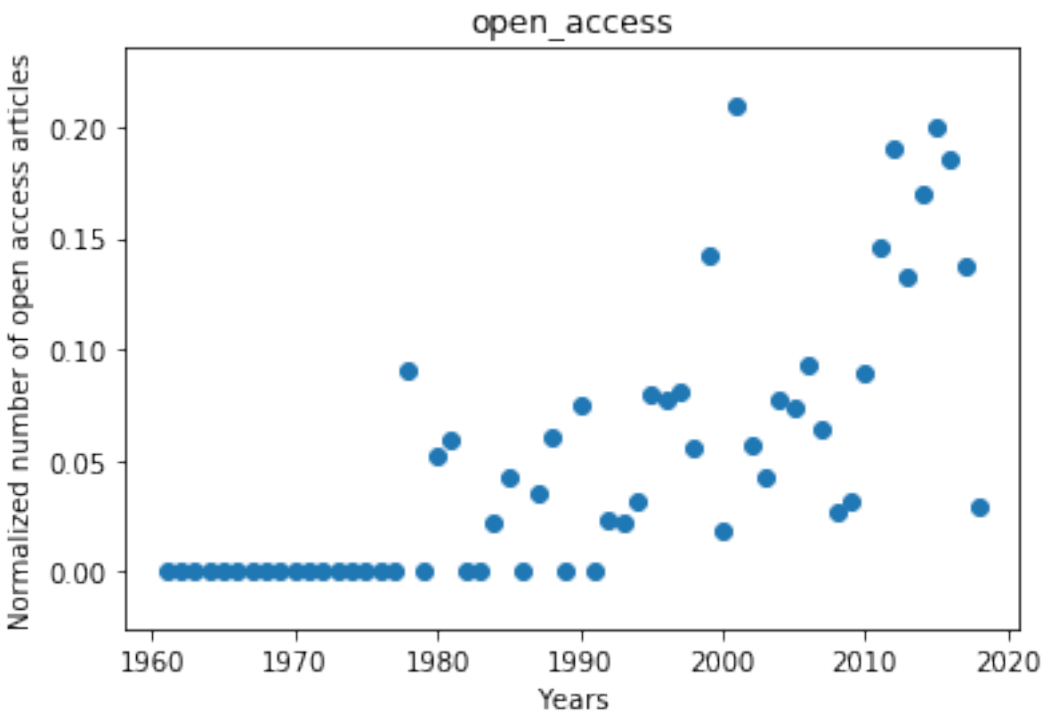

sort
Normalized instances of keywords
0.03
0.02
0.01
0.00
−0.01
1960
1970
1980
1990
2000
2010
2020
Years


health
Normalized instances of keywords
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
1960
1970
1980
1990
2000
2010
2020
Years


car
Normalized instances of keywords
0.04
0.03
0.02
0.01
0.00
−0.01
1960
1970
1980
1990
2000
2010
2020
Years


vision
Normalized instances of keywords
0.08
0.06
0.04
0.02
0.00
1960
1970
1980
1990
2000
2010
2020
Years

In [33]:
```
x, y = open_access(df)
plt.figure()
plt.scatter(x, y)
plt.title("open_access")
plt.xlabel("Years")
plt.ylabel("Normalized number of open access articles")
plt.savefig("plots/openaccess.png")
plt.show()
```

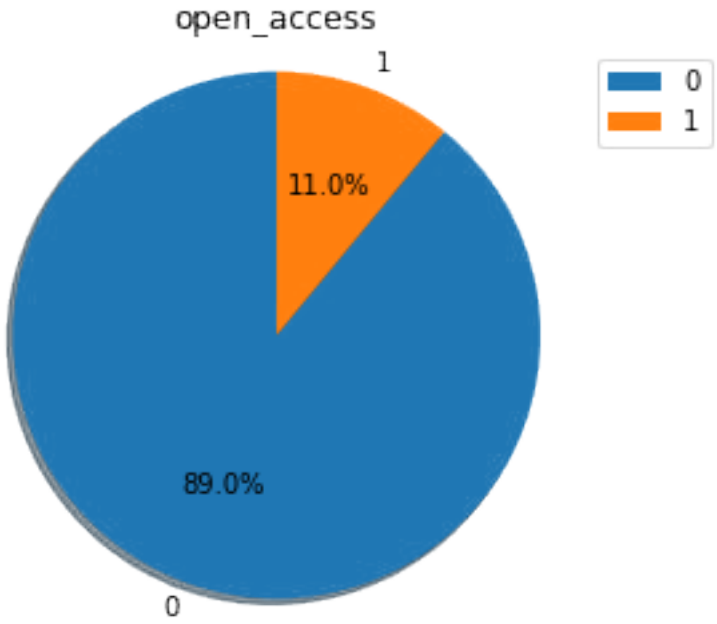


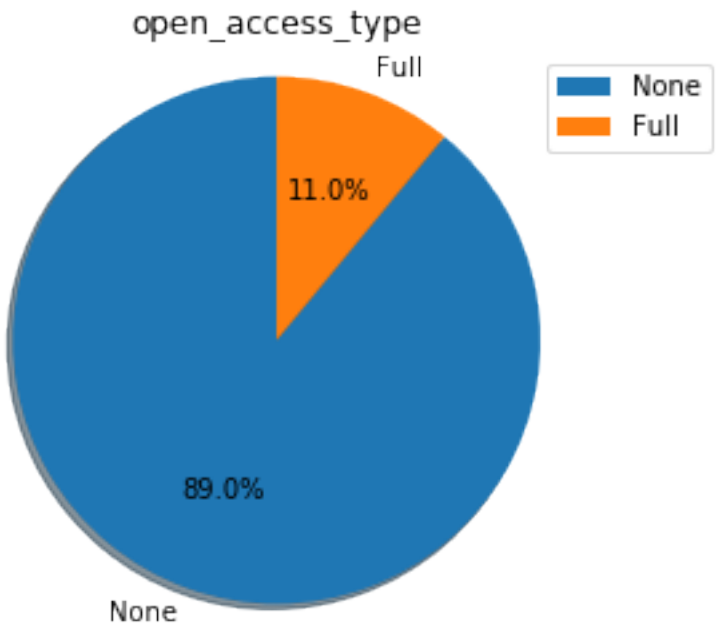

In [34]:
```
from collections import Counter as c
# sum(df.open_access.astype("int64"))
# len(df.open_access) - sum(df.open_access.astype("int64"))
a = c(df.open_access_sponsor_type)
print(a)
```

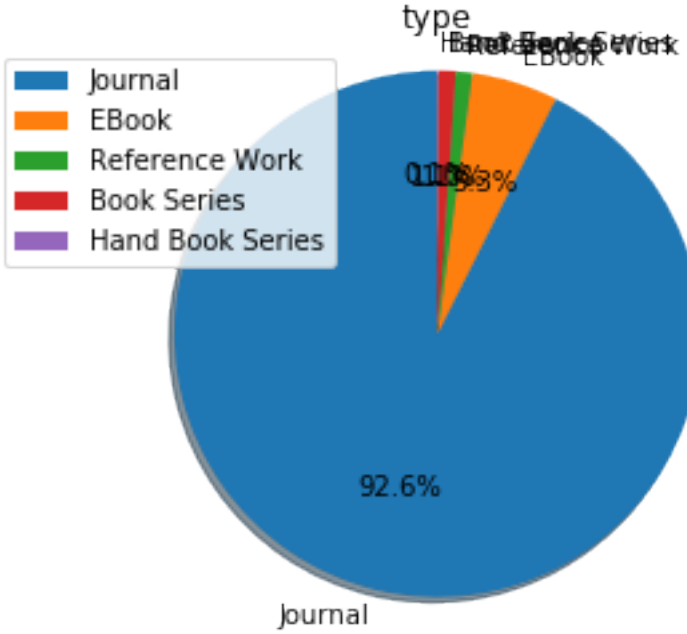
```
Counter({None: 4447, 'ElsevierWaived': 221, 'ElsevierBranded': 158, 'FundingBody': 96, 'Author
```

In [35]:
```
from collections import Counter as c

cols = "open_access,open_access_type,type"
for column in cols.split(","):
    labels = []
    sizes = []
```

```
temp = c(df[column])

for key in temp.keys():
    labels.append(key)
    sizes.append(temp[key])

plt.figure()
patches, texts, _ = plt.pie(sizes, labels=labels, autopct='%1.1f%%',shadow=True, s
plt.legend(patches, labels, loc="best")
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title(column)
plt.savefig("plots/" + column + ".png")
plt.show()
```

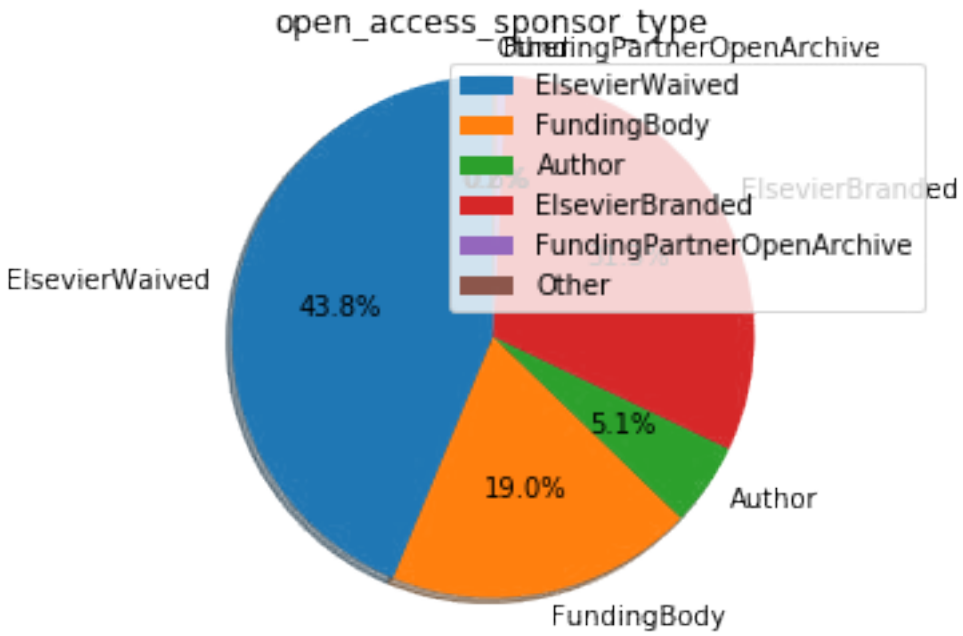





```
In [36]: labels = []
         sizes = []
```

```
temp = c(filter(None, list(df["open_access_sponsor_type"])))

for key in temp.keys():
    labels.append(key)
    sizes.append(temp[key])

plt.figure()
patches, texts, _ = plt.pie(sizes, labels=labels, autopct='%1.1f%%',shadow=True, star
plt.legend(patches, labels, loc=0)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("open_access_sponsor_type")
plt.savefig("plots/open_access_sponsor_type.png")
plt.show()
```



# keyword_analysis

November 9, 2017

In [1]:
```
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
dat = pickle.load(open("extracted_files/extracted_raw.p", "rb"))
dat = list(filter(None, dat))
df = pd.DataFrame(dat)
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['abstract', 'authors', 'cite_count', 'cover_date', 'doi', 'keywords',
       'publication_name', 'reference_count', 'subject_area', 'title', 'type',
       'volume'],
      dtype='object')
```

In [4]:
```
abstracts = " ".join(list(df.abstract))
```

In [5]:
```
abstracts = abstracts.lower()
```

In [6]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
```

In [7]:
```
words = word_tokenize(abstracts)
```

In [8]:
```
keywords = pickle.load(open("keywords.p", "rb"))
```

In [9]:
```
also_keywords = []
for key in keywords:
    also_keywords.append(key.split())

keywords = []
for sublist in also_keywords:
    for item in sublist:
        keywords.append(item.lower())
```

In [25]:
```
key_dict = {}
 for key in keywords:
     for word in words:
```

```
                if word.lower() == key.lower():
                    if key in key_dict.keys():
                        key_dict[key] += 1
                    else:
                        key_dict[key] = 1
```

In [27]:
```
import operator
# x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
sorted_x = sorted(key_dict.items(), key=operator.itemgetter(1))
```

In [28]:
```
for_plotting = sorted_x[-15:]
```

In [29]:
```
for_plotting
```

Out[29]:
```
[('problem', 1668),
 ('optimization', 1692),
 ('computational', 1988),
 ('model', 2052),
 ('for', 2084),
 ('intelligence', 2590),
 ('artificial', 2754),
 ('network', 2888),
 ('algorithm', 5418),
 ('learning', 6258),
 ('system', 8138),
 ('data', 8352),
 ('and', 11298),
 ('the', 12384),
 ('of', 38465)]
```

In [30]:
```
to_remove = "of,the,and,for,a,i,in,an"
to_remove = to_remove.split(",")
to_remove
```

Out[30]:
```
['of', 'the', 'and', 'for', 'a', 'i', 'in', 'an']
```

In [31]:
```
# for i, val in enumerate(for_plotting):
#     print(i,val)
#     if key in to_remove:
i = 0
while 1:
    if for_plotting[i][0] in to_remove:
        del for_plotting[i]
    else:
        i += 1
    if i >= len(for_plotting):
        break
```

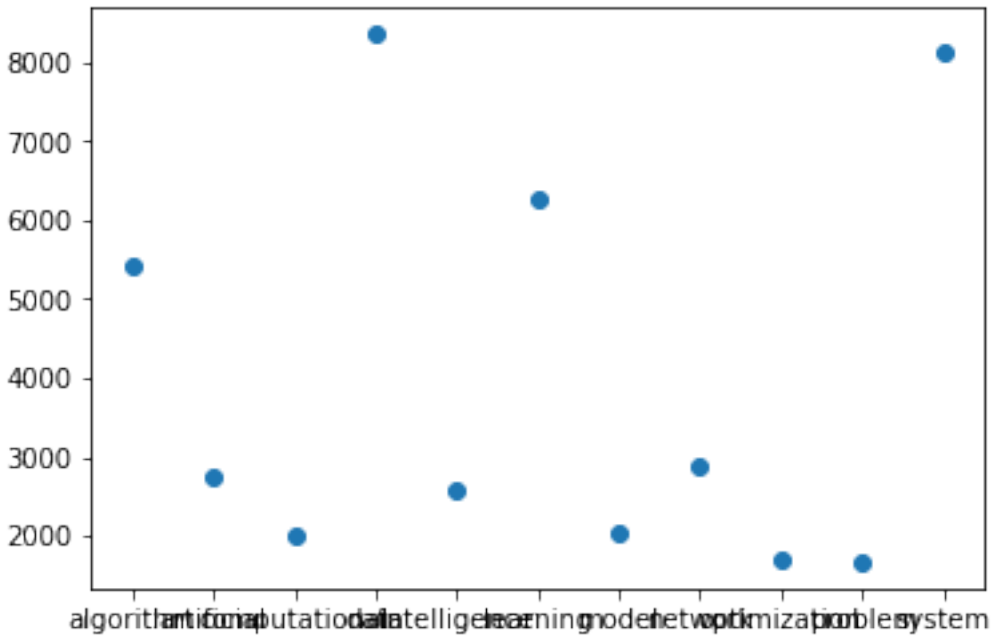
In [32]:
```
for_plotting
```

```
Out[32]: [('problem', 1668),
          ('optimization', 1692),
          ('computational', 1988),
          ('model', 2052),
          ('intelligence', 2590),
          ('artificial', 2754),
          ('network', 2888),
          ('algorithm', 5418),
          ('learning', 6258),
          ('system', 8138),
          ('data', 8352)]
```

```
In [33]: x = []
         y = []
         for val in for_plotting:
             x.append(val[0])
             y.append(val[1])
         # pickle.dump([x,y], open("for_plotting.p", "wb"))
```

```
In [34]: # x, y = pickle.load(open("for_plotting.p", "rb"))
```

```
In [35]: plt.scatter(x,y)
         plt.show()
```




```
In [37]: li = list(df.subject_area)
```

```
In [38]: flat_li = []
         for sublist in li:
             for item in sublist:
                 flat_li.append(item)

In [41]: from collections import Counter

In [42]: count = Counter(flat_li)

In [44]: sorted_count = sorted(count.items(), key=operator.itemgetter(1))

In [52]: temp = sorted_count[-8:]

         labels = []
         sizes = []

         for item in temp:
             labels.append(item[0])
             sizes.append(item[1])

         labels.append("others")
         sizes.append(1)

In [54]: # Data to plot
         # colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
         # explode = (0.1, 0, 0, 0)  # explode 1st slice

         # Plot
         plt.pie(sizes, labels=labels,
                 autopct='%1.1f%%', shadow=True, startangle=140)

         plt.axis('equal')
         plt.show()
```