Projekt 3: Mendeley GruppeNr.1302

version 0.1

Yuankun LUO, Mathias Sabbagh, Wilhelm Reinhardt

December 16, 2013

Contents

Welcome to Super RSL's documentation!

Gruppe Info:

- GruppeNr: 1302
- Yuankun LUO
- Mathias Sabbagh
- Wilhelm Reinhardt

Contents:

Aufgabenstellung

Aufgabenstellung

In diesem Projekt geht es um die Analyse von Publikationen aus Mendeley1. Mendeley ist ein kollaboratives Literaturverwaltungsprogrammzum Organisieren, Teilen und Entdecken von Literatur. Mit Hilfe des Python-Clients2 für die Mendeley-API soll auf Publikationsdaten zugegriffen werden. Die abgerufenen Daten sollen in einem geeigneten Datenformat persistent abgelegt werden. Wie bereits inProjekt 2 müssen alle Auswertungen und Visualisierungen mit Python bzw. matplotlib erstellt werden. Die Dokumentation soll für eine Person verständlich sein, die kein Experte für Informetrie ist. Das bedeutet, die Vorgehensweise und die Interpretation der Ergebnissemüssen erklärt werden. Folgende Punkte müssen dabei berücksichtigt werden:

- Wie verteilen sich die in Mendeley abgelegten Publikationen auf die letzten 10 Jahre? (Dafür müssen nicht alle Publikationen heruntergeladen werden!)
- Was sind die Top20-Tags in der Kategorie „Computer and Information Science“?
- Was sind die 10 populärsten Publikationen die in der Zeitschrift „Nature“ erschienen sind?
- Auflistung aller Publikationen von Prof. Wolfgang G. Stock (Extrahiere diese Daten automatisch mit Hilfe von Python!)
- Erstelle ein Diagramm der Publikationsanzahl über die vorhandenen Jahre
- Erstelle ein Ranking aller Co-Autoren mit denen Prof. Stock zusammengearbeitet hat nach Anzahl der in Mendeley vorhandenen, gemeinsamen Publikationen.
- Suche nach dem Tag „ontology“ und bestimme die Häufigkeit für jede Kategorie in Mendeley für das Jahr 2011. Beachte dabei, dass es sich um eine Broad Folksonomy handelt. Stelle die als Diagramm dar.

HighLight

makeNameTuple()

This funktion extracts the letter of a given authorname, exp: From "Wolfgan. G. Stock" it returns "wolfgangstock".

We use this letter chain to compare with other authors.

```
def makeNamesTuple(inputList):
    """Get the (forename, surname) tuple for quickly computation

    .. warning:: Due to the confusion in authors name,
        EG: the confusion between 'Wolfgang G Stock' and 'Wolfgang G.Stock'
        Hier we use **re** package to extract the letters in authors name.
```

```
        If all the lettes of two names are the same, then they both was computed
        to one author.

    :param inputList: A list of authors
    :type inputList: List
    :returns:  a List of tuple of (forename, surname, lettersofname)
    """
    result = []
    for nameD in inputList:
        forename = nameD['forename']
        surename = nameD['surname']
        nameletters = str(forename +  surename).lower()
        letterList = re.findall(r'\w*', nameletters)
        nameletters = "".join(letterList)
        result.append((forename, surename,nameletters))
    return result
```

makeLineBar()

This figure was created by the funkction diagramm.makeLineBar(). The function gets a tuple as argument fsize.

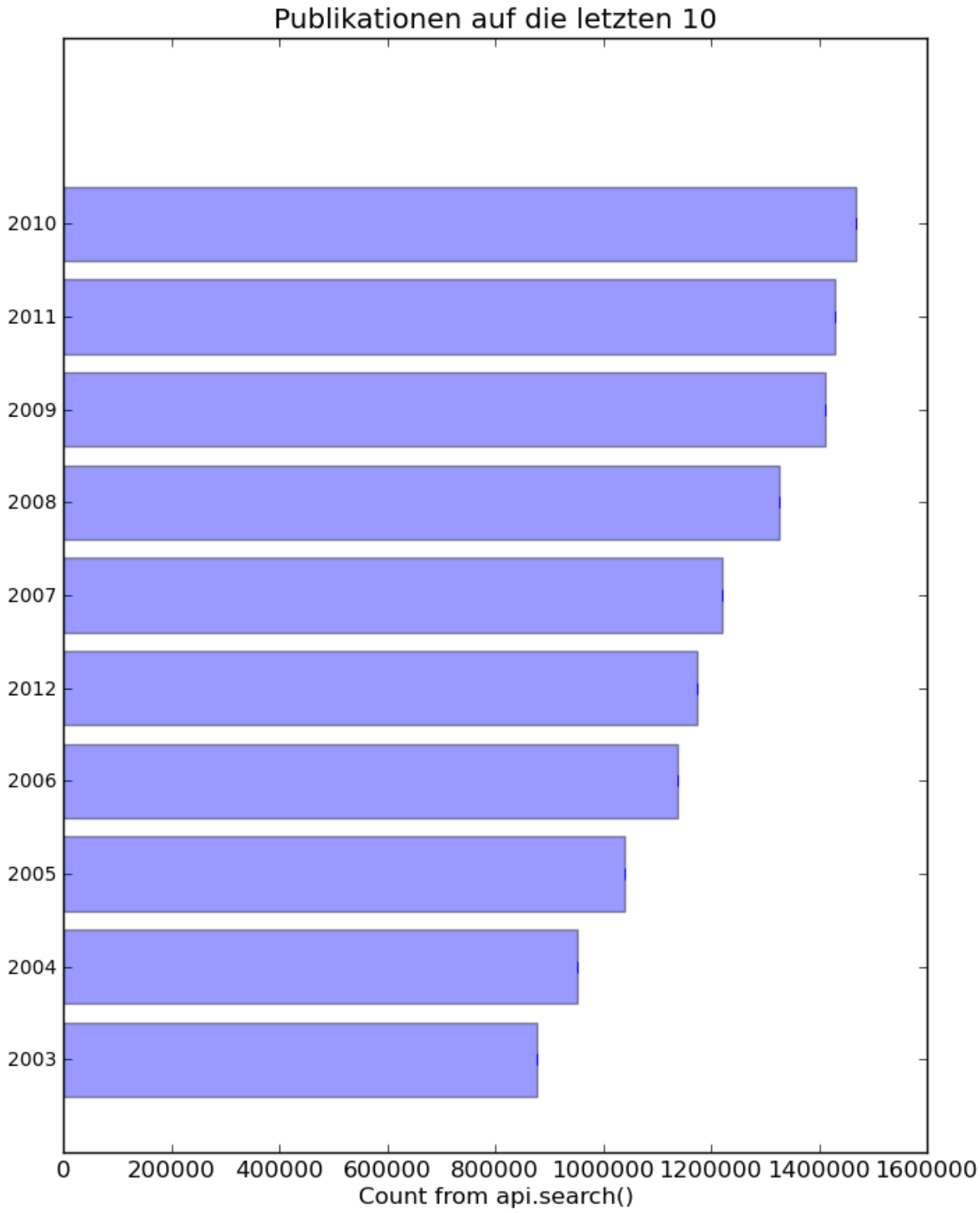
```
def makeLineBar(inputDict, filename, title, xlabel , output = \
"image/", fsize = (15,20,5,2)):
    """A function to make a line bar plot.

    :param inputDict: A inputData of {str:integer}
    :type inputDict: A Dict
    :param filename: The output picture's name.
    :type filename: String
    :param title: The title at top of this picture
    :type title: String
    :param xlabel: A short sentence at bottom of this pic.
    :type xlabel: String
    :param output: The folder path where our plot diagramms \
    were stored, **default** is the image folder.
    :type output: String
    :param fsize: The picture size
    :type fsize: A List of [width, length, xlimmax, xlimmin]
    :returns: None
    """
    import numpy as np
    import matplotlib.pyplot as plt
    data = sortDict(inputDict)
    data = [(k, v) for k, v in data.items()]
    y_keys = [k[0] for k in data ]
    y_pos = np.arange(len(y_keys))
    performance = [k[1] for k in data ]
    error = np.random.rand(len(y_keys))
    w, l, mi, ma = fsize[0], fsize[1], fsize[2],fsize[3]
    plt.figure(figsize=(w,l), dpi=300)
    plt.ylim((mi,ma))
    plt.autoscale()
    plt.barh(y_pos, performance, xerr=error, align='center', \
    alpha=0.4)
    plt.yticks(y_pos, y_keys, size='small')
    plt.xlabel(xlabel)
    plt.title(title)
    filename = filename.replace(" ", "_")
```

```
    plt.savefig(output + filename + ".png")
    plt.show()
```

Auswertung

Verteilung der Publikationen in Mendeley auf die letzten 10 Jahre

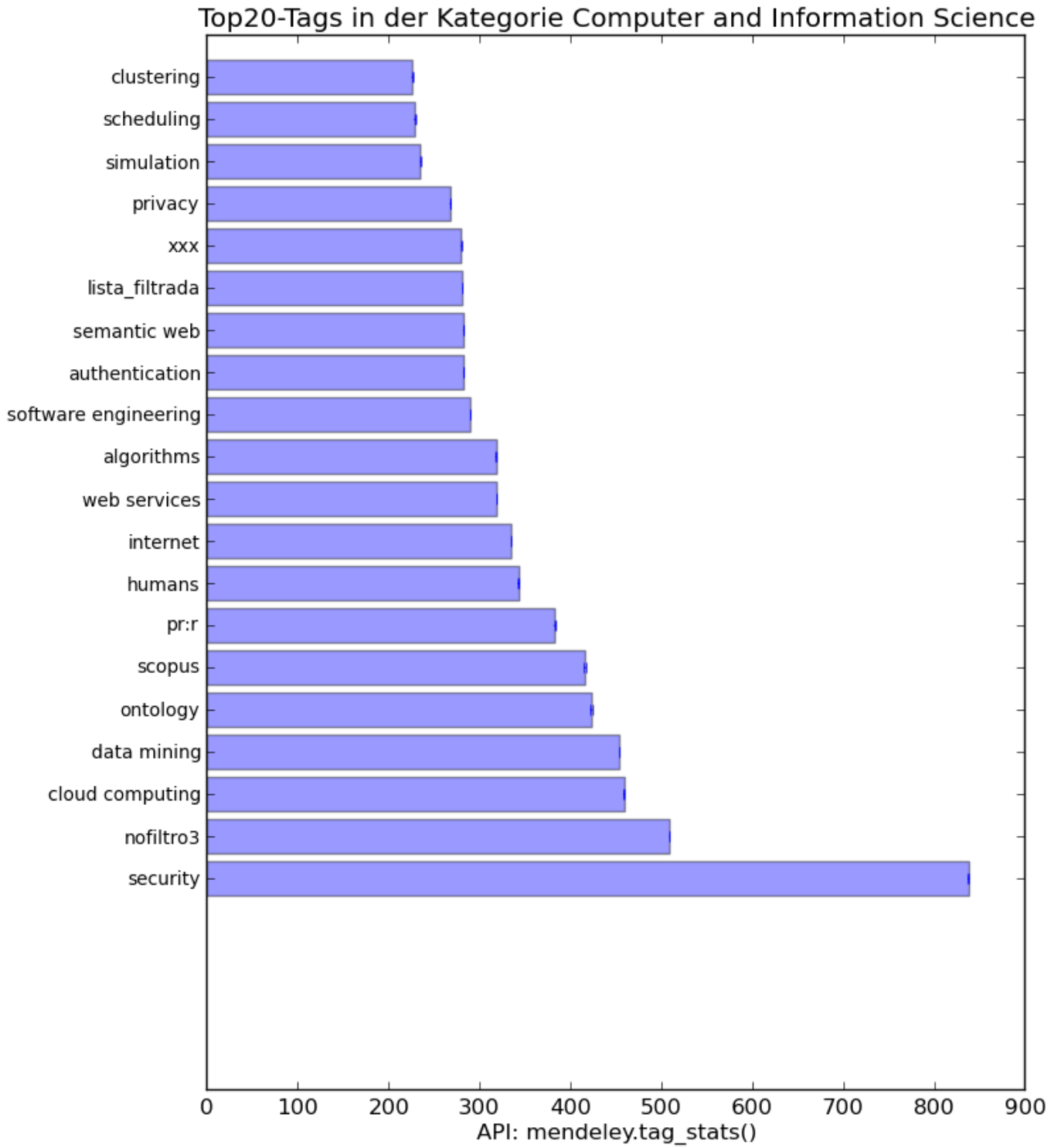
Grafik 1 zeigt die Verteilung der Publikationen in Mendeley für die letzten 10 Jahre. Die Grafik ist nach Anzahl, und nicht nach Jahren sortiert. Hierbei ist ersichtlich, dass im Zeitraum zwischen 2008 und 2011 die meisten Publikationen in Mendeley eingetragen wurden. 2003 bis 2006 weisen eine geringere Zahl auf, da dies erst die Startphase von Mendeley war.



Grafik 1 - Verteilung der Publikationen auf die letzten 10 Jahre

Top20-Tags in der Kategorie „Computer and Information Science“

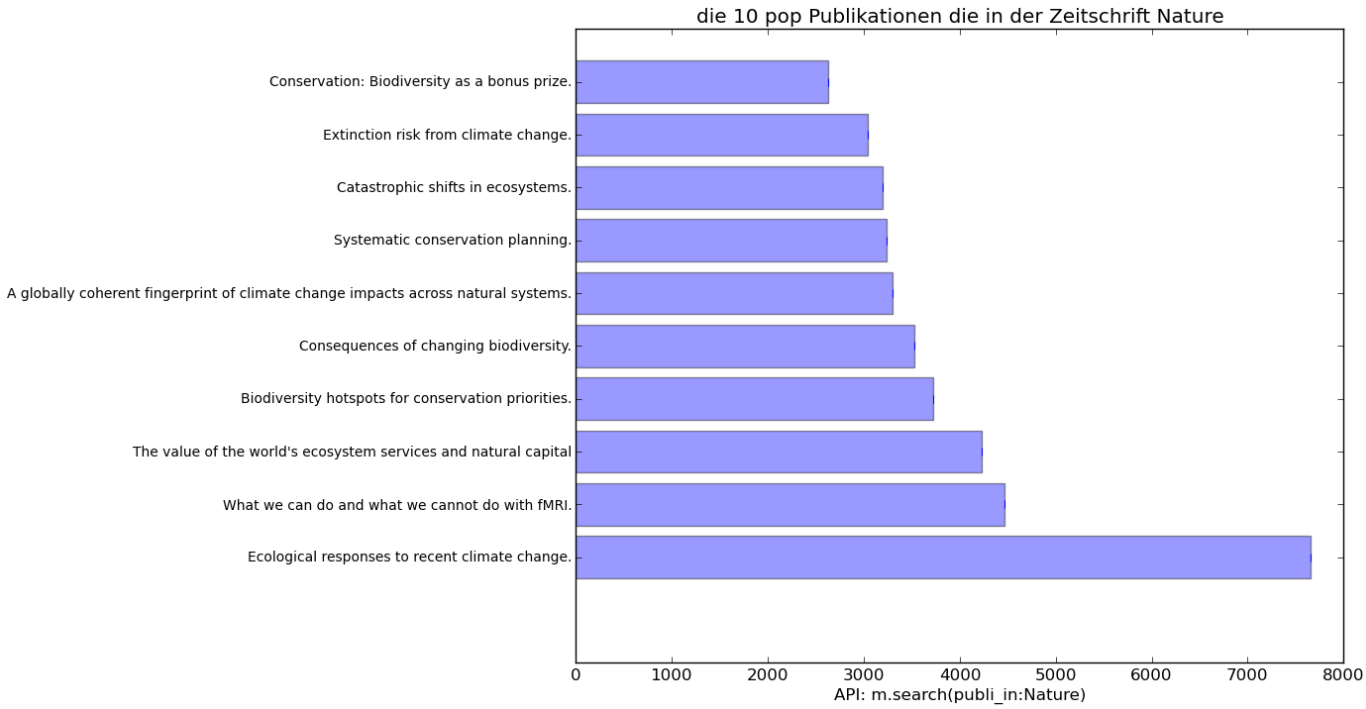
Grafik 2 stellt die 20 häufigsten Tags in der Kategorie „Computer and Information Science“ dar. "security" liegt hierbei mit einem Spitzenwert von etwa 850 ganz Vorne. Die anderen Tags verteilen sich zwischen 250 und 510.



Grafik 2 - Top20-Tags der Kategorie „Computer and Information Science“

Die 10 populärsten Publikationen die in der Zeitschrift „Nature“

Grafik 3 stellt die 10 populaersten Publikationen in der Zeitschrift "Nature" dar. Der klare Favorit ist hierbei die Publikation "Ecological responses to recent climate change" mit einem Wert von fast 8000. Die restlichen Publikationen haben lediglich Werte zwischen 2800 und 4500.



Grafik 3 - 10 populärsten Publikationen die in der Zeitschrift „Nature“

Auflistung aller Publikationen von Prof. Wolfgang G. Stock

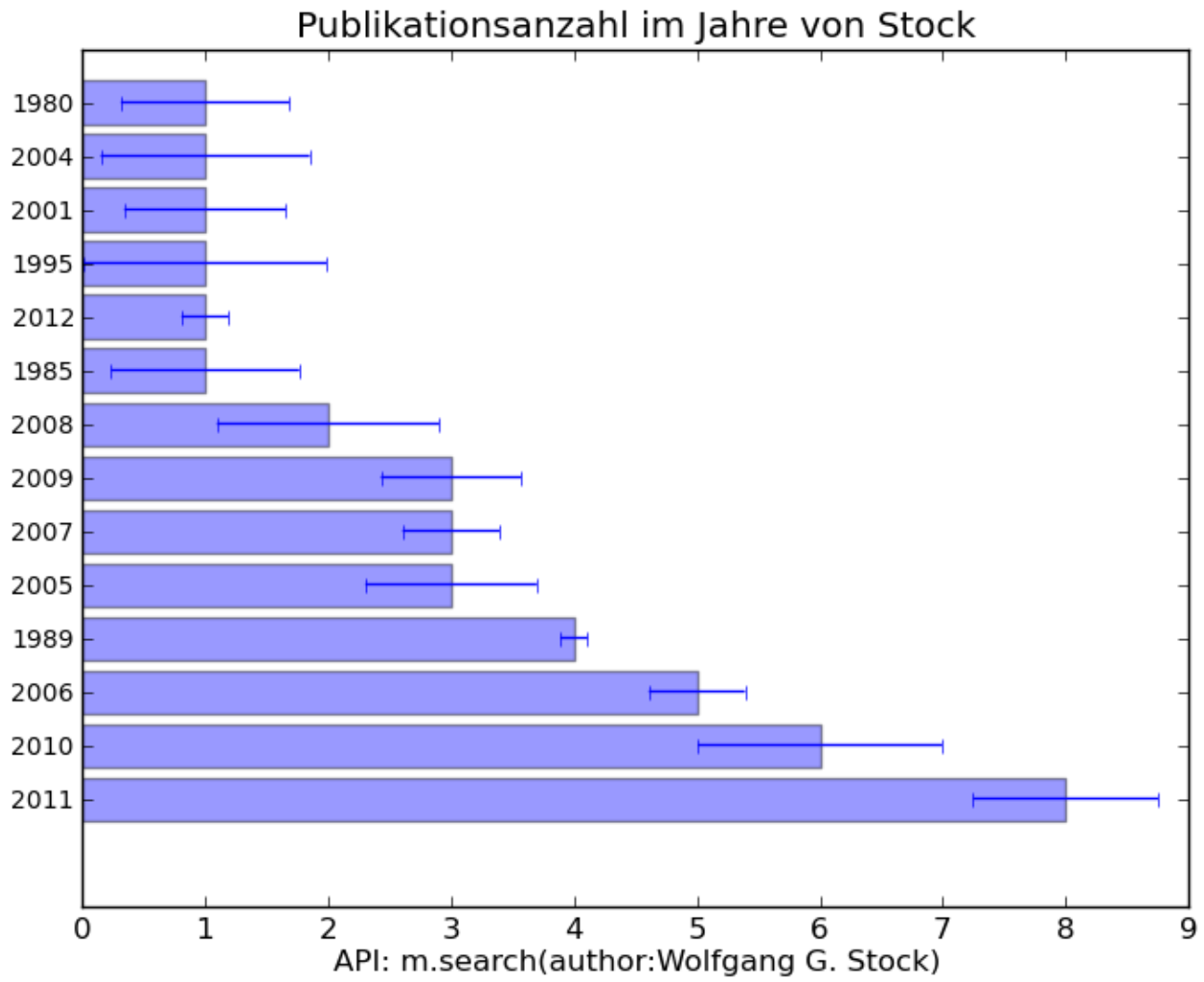
Folgende Titel konnten in der Mendeley Datenbank für den Namen Wolfgang G. Stock gefunden werden:

- Die Bedeutung der Zitatenanalyse fuer die Wissenschaftsforschung
- Datenbank "Grazer Schule"
- Die Genese der Theorie der Vorstellungsproduktion der Grazer Schule
- On relevance distributions
- Die Bedeutung Ludwig Flecks Fuer Die Theorie der Wissenschaftsgeschichte
- Publikation und Zitat: die problematische Basis empirischer Wissenschaftsforschung
- Georg Klaus ueber Kybernetik und Information
- Die Entstehung einer wissenschaftlichen Disziplin
- Information Retrieval: Informationen suchen und finden
- Die Bedeutung der Theorie der Vorstellungsproduktion der Grazer Schule fuer die kognitive Wissenschaft
- Folksonomies and science communication
- Informationelle Staedte im 21. Jahrhundert
- Evidenzbasierte Bibliotheks- und Informationspraxis : EBLIP5, Stockholm, 2009
- Informational cities: Analysis and construction of cities in the knowledge society
- Concepts and semantic relations in information science
- The inflation of impact factors of scientific journals
- Folksonomies in Wissensrepraesentation und Information Retrieval
- Online-Hosts fuer Wissenschaft, Technik und Medizin auf dem deutschenInformationsmarkt
- Gender-specific information search behavior
- Practitioners and academics as authors and readers: the case of LIS journals

- Impact and relevance of LIS journals: Ascientometric analysis of international and German-language LIS journals - Citation analysis versus reader survey
- "Power tags“ in information retrieval
- Deutsche Zeitschriften des Bibliotheks- und Informationswesens. Leser, Zitate und Redaktionen in szientometrischer Analyse
- Intellectual property information: A comparative analysis of main information providers
- Informationsmarkt. Informationen im I-Commerce anbieten und nachfragen
- Folksonomy and information retrieval
- Construction and Evaluation of a Blended Learning Platform for HigherEducation
- Collective Indexing of Emotions in Images. A Study in Emotional Information Retrieval
- Intellectual property information. A case study of Questel-Orbit
- Dimensions of Informational City Research
- Dimensionen der Zeitschriftenszientometrie am Beispiel von "Buch und Bibliothek"
- Testing Collaborative Filtering against Co-Citation Analysis and Bibliographic Coupling for Academic Author Recommendation
- Folksonomy: The Collaborative Knowledge Organization System
- Expert Recommendation for Knowledge Management in Academia
- MEMOSE. Search Engine for Emotions in Multimedia Documents
- Dimensions of the scientometrics of journals: "Buch und Bibliothek" as a concrete example
- Retrieval effectiveness of tagging systems
- Finding Emotional-Laden Resources on the World Wide Web
- Incentives for Emotional Multimedia Tagging
- Science and technology in the region: The output of regional science and technology, its strengths and its leading institutions

Publikationsanzahl von Prof. Stock über die vorhandenen Jahre

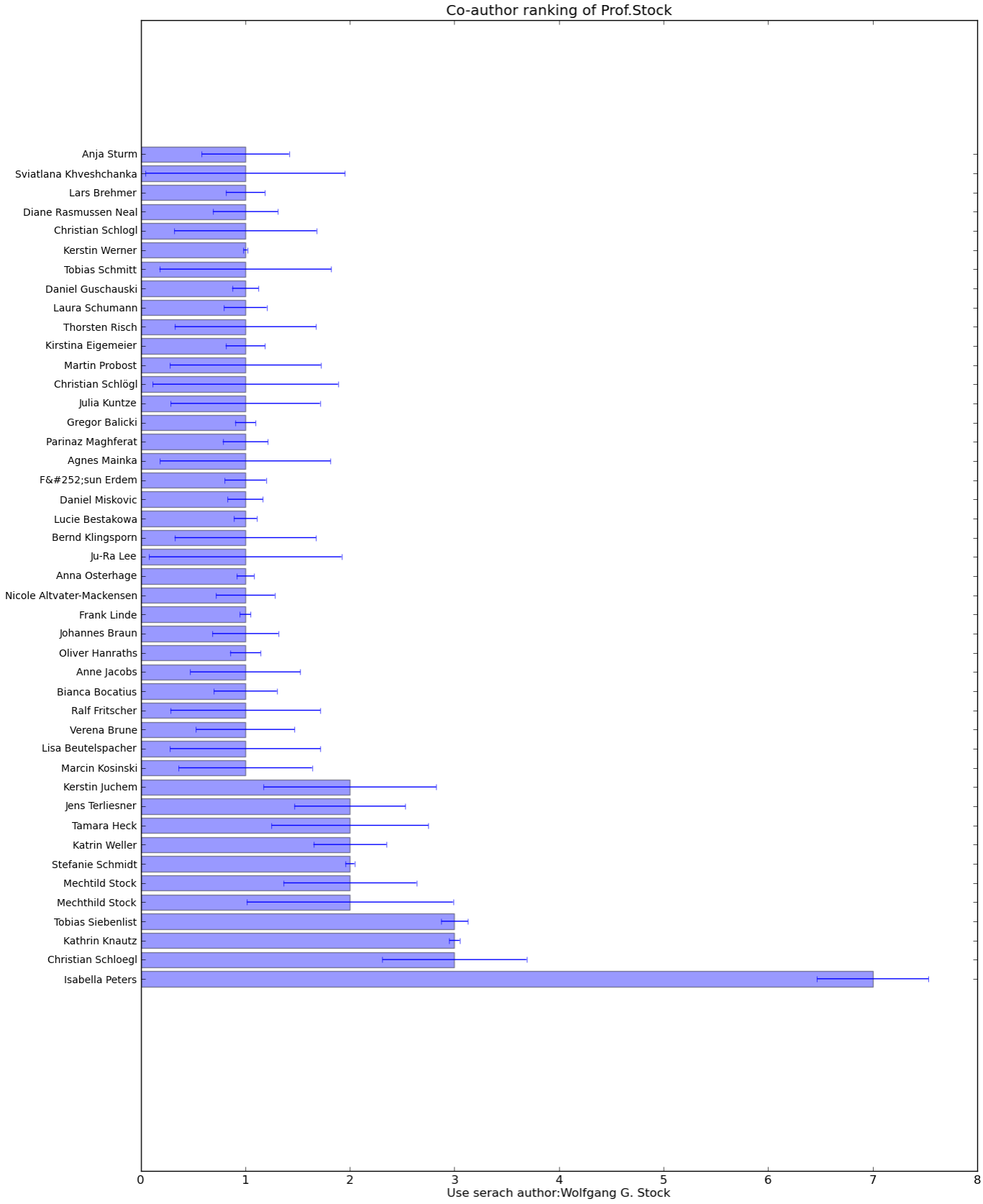
Grafik 4 zeigt die Publikationsanzahl von Prof. Stock pro Jahr. Hierbei sieht man, dass die Jahre 2010 und 2011 mit 6 und 8 Publikationen die erfolgreichsten waren.



Grafik 4 - Publikationsanzahl von Prof. Stock

Ranking aller Co-Autoren von Prof. Stock nach Mendeley

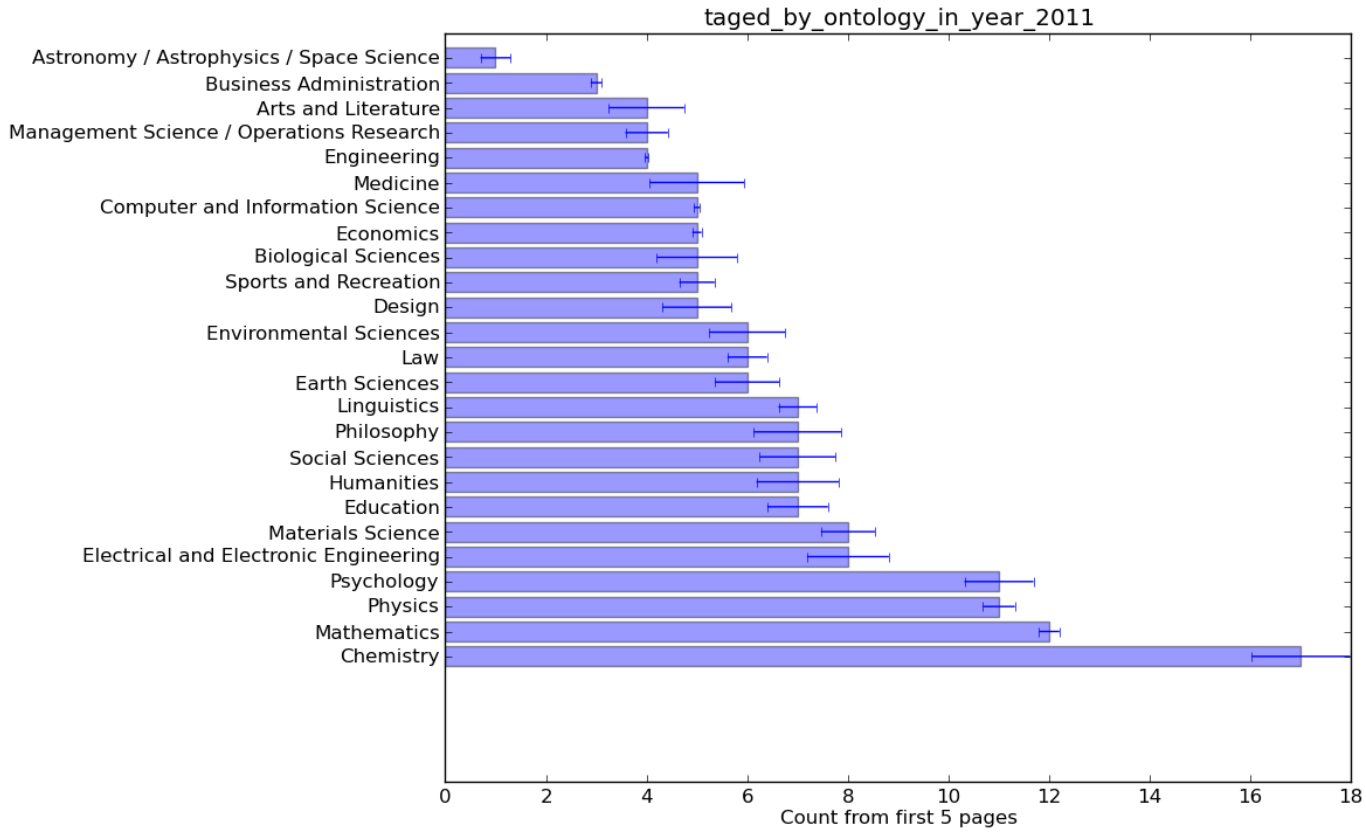
Grafik 5 zeigt die Co-Autoren von Prof. Stock die in Mendeley genannt werden. Hierbei war Isabella Peters 7 mal Co-Autor und nimmt damit den ersten Platz ein. Den zweiten Platz teilen sich Christian Schloegl, Kathrin Knautz und Tobias Siebenlist mit jeweils 3 Publikationen.



Grafik 5 - Co-Autoren von Prof. Stock

Häufigkeit von dem Tag "ontology" für jede Kategorie in Mendeley

Grafik 6 zeigt die Verteilung des Tags "ontology" in allen Kategorien von Mendeley. Artikel in der Kategorie Chemie werden mit am häufigsten mit diesem Wort getaggt. Den zweiten Platz belegt Mathematik mit einem Wert von 12. "Computer and Information Science" weist gerade mal einen Wert von 5 auf.



Grafik 6 - Häuifigkeit des Tags "ontology" für jede Kategorie

Documentation for the Code

All the source code are in the super_rsl package.

super_rsl.test module

super_rsl.datadownloader module

`super_rsl.datadownloader.`**`authorsPapers`** (*`mendeley`*, *`author='Wolfgang G. Stock'`*)
Search all papars for a given author's name

Parameters:
- **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance
- **author** (*String*) -- A author name

Returns: a List of objects

`super_rsl.datadownloader.`**`categoriesList`** (*`mendeley`*)
This function downloads the result that mendeley.categoreis() returns. Then store this result in order this categorie id into a list of tuple. Like [(catid, catname, catslug)]. We do this, because it is convinient to checkt the catid in anothers functions. :)

Parameters: **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance

Returns: A list of categories

`super_rsl.datadownloader.`**`computeFreq`** (*`resultDict`*, *`inputList`*)
A function to compute freq

Parameters:
- **resultDict** (*Dict*) -- A input Dict
- **inputList** (*A List*) -- A input List

Returns: A dict

`super_rsl.datadownloader.`**`papersInMag`** (*`mendeley`*, *`mag='Nature'`*)
Search for a given mag, then collect papers in this mag.

> ***Warning***
>
> Due the limit of this API max 500 / hour. This funktion collects infomations **only** in the fist 100 pages. Or onle collects max 500 results.

Parameters:
- **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance
- **topn** (*A Interger*) -- A Interger indicate ranking item
- **mag** (*A string*) -- A magzine name or a publication name

Returns: a list of tuple (uuid, title, year, url, publication_outlet)

`super_rsl.datadownloader.`**`papersInYears`** (*`mendeley`*, *`fromyear=2013`*, *`toyear=2013`*)
Download the year's account for a given timespace. Then store this information as a list of tuples.
Use mendeley.search("year:YEAR")

Parameters:
- **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance
- **fromyear** (*A four digit integer*) -- The year for start counting
- **toyear** (*A four digit integer*) -- The end year for counting

Returns: a List of tuple, (year:count)

`super_rsl.datadownloader.`**`popPaperInMag`** (*`mendeley`*, *`paperList`*)
After calling papersInMag() function we get all papers in the specifical magzine. Then we use this function to collect the reader nummer for ervery papaer in the given paperlist. This fuction also return a list of tuple(uuid, title, readnumber).

Parameters:
- **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance
- **paperList** (*When a str, then this str indicates the file path. Otherwise is a list object.*) -- a List of tuple resulted after papaerInMag() function's calling.

Returns: A list of tuples of (readers, title, mag, year, uuid, tags)

`super_rsl.datadownloader.`**`taggedInYear`** (*`mendeley`*, *`tag='ontology'`*, *`year=2011`*)
This function search all taged by the gaven tag **default** is ontology. Then filter these returned papers if it was publisched in 2011. Like all others functions, this one also stores the result locally

> ***Warning***
>
> Hier for every categorie we **only collect 5 returned pages**.

Parameters:
- **mendeley** (*mendeley_client.MendeleyClient*) -- A mendeley client instance
- **tag** (*String*) -- The tag, which was tagged a paper
- **year** (*Integer*) -- The specifics year

Returns: A Dict of {catid:(count, catname, catslug)}

`super_rsl.datadownloader.`**`top20TagsDownload`** (*`mendeley`*, *`catid=6`*)
Download the top20 tags count for the giving catagory id. Then store the result local use the catagorie name.

Parem mendeley: A mendeley client instance

Parameters: **cat** (*Integer*) -- The categroie id, **Default is 6**

Returns: a list of tuples, every tuple ist the (tagname, count) pair.

super_rsl.compute module

super_rsl.compute.**co_authorsAndYears** (*inputData*, *author=None*)
Compute the co-author freq, publications year's freq from a given data, that a downloaded data resulted from datadownloader.py module.

Parameters: **inputData** (*String*) -- The .db file name under data folder
Returns: A List of

super_rsl.compute.**computeFreq** (*inputList*, *resultDict=None*)
A function to compute normal freq

Parameters:
- **resultDict** (*Dict*) -- A input Dict
- **inputList** (*A List*) -- A input List

Returns: A dict

super_rsl.compute.**computeNameFreq** (*inputList*, *resultDict=None*)
A special function to compute authors' names frequence

Parameters:
- **resultDict** (*Dict*) -- A input Dict
- **inputList** (*A List*) -- A input List **in form (forename, surname, nameletter)**

Returns: A dict

super_rsl.compute.**computeTopPopPaperInMag** (*paperList*, *topn=10*)
This function computes the top n **default n = 10** papers in a given paperList, which is the result of calling datadownloader.popPapersInMag() function. It return the top n element as list of tuple of (title, number)

Parameters:
- **paperList** (*A List*) -- A list of tuples, which is a result of datadownloader.popPapersInMag()
- **topn** (*Integer*) -- A ranking number

Returns: A list of tuple of (title, readernumber)

super_rsl.compute.**listtodict** (*inputList*)
Covert a list to a dict.

Parameters: **inputList** (*A list.*) -- A list object of tuple (key,value)
Returns: a dict of {key, value}

super_rsl.compute.**makeNamesTuple** (*inputList*)
Get the (forename, surname) tuple for quickly computation

Warning

Due to the confusion in authors' names, EG: the confusion between 'Wolfgang G Stock' and 'Wolfgang G.Stock' Hier we use **re** package to extract the letters in authors name. If all the lettes of two names are the same, then they both were computed to one author.

Parameters: **inputList** (*List*) -- A list of authors
Returns: a List of tuples of (forename, surname, lettersofname)

super_rsl.compute.**makesureCoauthor** (*authorname*, *authorList*)
A function to make sure that the computed co-authors are true co-authors of a given author.

Parameters:
- **authorname** (*A String*) -- A given author name
- **authorList** (*A List of string*) -- A list of authors **nameletter**

Returns: True if ok, otherwise False

super_rsl.compute.**sortDict** (*inputDict*, *rank=None*, *n=1*)

Sort the inputDict.

Parameters:
- **inputDict** (*dict*) -- the Dict to sort
- **rank** (*Int*) -- A number to indicate how many items will be ranked

Returns: OrderedDict

super_rsl.diagramm module

`super_rsl.diagramm.`**`makeBarPlot`** (*`inputDict`*, *`filename`*, *`rank=10`*, *`title=None`*, *`ylabel='Count'`*, *`output='image/'`*, *`fsize=(15, 20, 5, 2)`*)

Plot the bar diagramm of the input dict. After the call of computation module's functions, we have a inputDict as a result. This function uses this result to make a diagramm (plot) then store the plot locally. ..Note: If the rank parameter is None, then output all data ranking.

Parameters:
- **inputDict** (*Dict, the key is String, the value is interger*) -- A dict of (key:value) paire.
- **rank** (*Integer*) -- Indicate the top N (ranking), **default** is None
- **output** (*String*) -- The folder path where our plot diagramms were stored, **default** is the image folder.
- **fsize** (*A tuple of (width, length, xlimmax, xlimmin)*) -- The picture size

Returns: None

`super_rsl.diagramm.`**`makeLineBar`** (*`inputDict`*, *`filename`*, *`title`*, *`xlabel`*, *`output='image/'`*, *`fsize=(15, 20, 5, 2)`*)

A function to make a line bar plot.

Parameters:
- **inputDict** (*A Dict*) -- A inputData of {str:integer}
- **filename** (*String*) -- The output picture's name.
- **title** (*String*) -- The title at top of this picture
- **xlabel** (*String*) -- A short sentence at bottom of this pic.
- **output** (*String*) -- The folder path where our plot diagramms were stored, **default** is the image folder.
- **fsize** (*A List of [width, length, xlimmax, xlimmin]*) -- The picture size

Returns: None

Indices and tables

- *genindex*
- *modindex*
- *search*

Index

Python Module Index

s