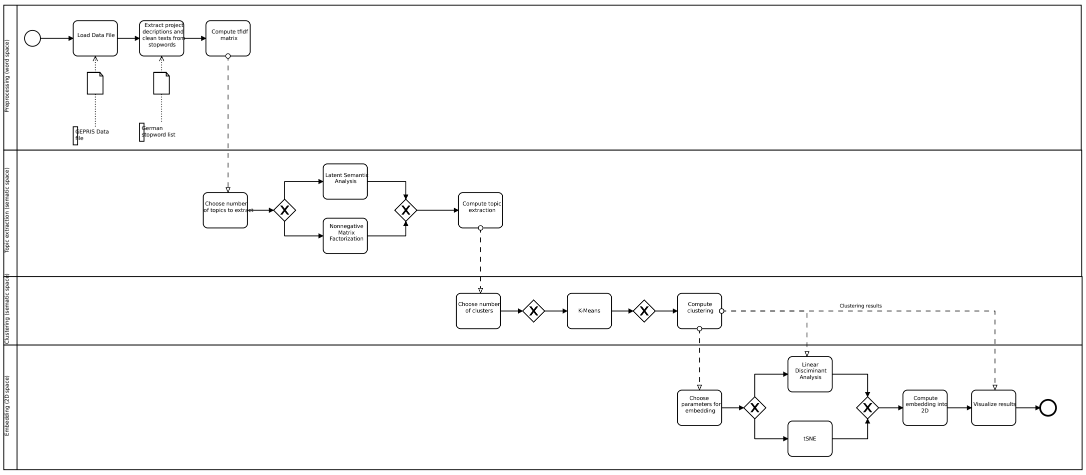
Topic extraction from the GEPRiS dataset and creation of an user-centric visualisation

Author: Tim Korjakow
Summer term 2018
Freie Universität Berlin
Fachgebiet Human-Centered Computing



In [1]:

```
import time
import json
import spacy
import regex as re
from langdetect import detect
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF as NonnegativeMatrixFactorization
from sklearn.cluster import KMeans
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.manifold import TSNE
import numpy as np

from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
import ipywidgets as widgets
from IPython.display import display

from bokeh.io import output_notebook, show
from bokeh.plotting import figure, ColumnDataSource
from bokeh.palettes import d3
output_notebook()
```

BokehJS 0.13.0 (https://bokeh.pydata.org) successfully loaded.

Loading and Cleaning

The first step in every NLP project which works with texts is always the preparation of the input data. In this example the Project dump from GEPRIS is loaded and the project descriptions are extracted. After that the texts get cleaned by removing all non-alphabetic chars and all stopwords from the texts. English texts are getting filtered in oder to make the analysis simpler and more comparable.

In [2]:
```
def loadProjects():
    with open('../../../assets/data/projects.json', 'r') as datafile:
        return json.load(datafile)

def loadGermanStopwords():
    with open('../../../assets/data/stopwords_de.json', 'r') as datafile:
        return json.load(datafile)

def loadEnglishStopwords():
    with open('../../../assets/data/stopwords_eng.json', 'r') as datafile:
        return json.load(datafile)

def cleanProjectTexts():
    cleanedProjectTexts = {}
    stopwordsDE = set(loadGermanStopwords())
    for key,project in loadProjects().items():
        if detect(project['beschreibung']) == 'de':
            letters_only = re.sub('[^\w]', ' ', project['beschreibung'])
            words = letters_only.lower().split()
            usefulWords = [x for x in words if not (x in stopwordsDE)]
            cleanedProjectTexts[key] = ' '.join(usefulWords)
    return cleanedProjectTexts
```

TF-IDF computation

Summary: This technique vectorizes a corpus, e.g. a collection of documents, by counting all appearences of words in the corpus and computing the tf-idf measure for each document, word pair.

In-depth explanation:

In [3]:
```
def lemmatize(text):
    nlp = spacy.load('de')
    return nlp(text)

def TfIdf(dict):
    start = time.time()
    tfidf_vectorizer = TfidfVectorizer(tokenizer=lemmatize)
    tfs = tfidf_vectorizer.fit_transform(list(dict.values()))
    print('TFIDF execution time: ', time.time() - start)
    return (tfidf_vectorizer, tfs)
```

In [4]:
```
tfidf_vectorizer, tfs = TfIdf(cleanProjectTexts())
```

```
TFIDF execution time:  87.91095805168152
```

Topic extraction

Latent Semantic Analysis

Summary: The LSA transforms an corpus from its word space given by the tf-idf matrice into its semantic space. In this semantic space the dimensions denote topics in the corpus and every document vector is a linear combination of all the implicitly extracted topics.

In-depth explanation:

In [5]:
```
def LSA(tfs,num_topics=40):
    start = time.time()
    lsa = TruncatedSVD(n_components=num_topics, random_state=0).fit(tfs)
    print('LSA execution time: ', time.time() - start)

    #tfidf_feature_names = [str(token) for token in tfidf_vectorizer.get_feature
    #print_top_words(lsa, tfidf_feature_names, 10)
    return lsa.transform(tfs), lsa
```

Non-negative matrix factorisation

Summary: **Coming soon**

In-depth explanation:

In [6]:
```
def NMF(tfs,num_topics=40):
    start = time.time()
    nmf = NonnegativeMatrixFactorization(n_components=num_topics, init='random',
    nmf.fit(tfs)
    print('NMF execution time: ', time.time() - start)

    #
    #print_top_words(nmf, tfidf_feature_names, 10)
    return nmf.transform(tfs), nmf
```

Get top words for each dimension

In [7]:
```
def get_top_words_dim(model, feature_names, n_top_words):
    dim_topics = {}
    for topic_idx, topic in enumerate(model.components_):
        dim_topics[topic_idx] = [feature_names[i]
                                for i in topic.argsort()[:-n_top_words - 1:-1]]
    return dim_topics
```

Clustering

K-Means

Summary: Given a clustering the LDA can be used to find a projection into a lower dimensional space which maximizes inter-class variance and minimizes intra-class variance. This leads to neater cluster, but is grounded in the hypotheses that the clusters have some real semantic meaning. Otherwise it may enforce preexisting biases.

In-depth explanation:

In [8]:
```
def clusterNumberHeuristic(tfs):
    return (tfs.shape[0]*tfs.shape[1])//tfs.count_nonzero()

def cluster(tfs_reduced, num_topics=10):
    start = time.time()
    km = KMeans(n_clusters=num_topics).fit(tfs_reduced)
    print('Clustering execution time: ', time.time() - start)
    return km
```

Get top words for each cluster

In [23]:
```
def get_top_words_cluster(model, clustering_centers, feature_names, n_top_words)
    cluster_topics = {}
    word_space_cluster_centers = model.inverse_transform(clustering_centers)
    for i, word_space_cluster in enumerate(word_space_cluster_centers):
        cluster_topics[i] = [feature_names[j]
                             for j in word_space_cluster.argsort()[:-n_top_words - 1:
    return cluster_topics
```

Embedding into 2D

Linear Discriminant Analysis

Summary: Given a clustering the LDA can be used to find a projection into a lower dimensional space which maximizes inter-class variance and minimizes intra-class variance. This leads to neater cluster, but is grounded in the hypotheses that the clusters have some real semantic meaning. Otherwise it may enforce preexisting biases.

In-depth explanation:

In [24]:
```
def dimReductionLDA(tfs_reduced, clusters):
    start = time.time()
    tfs_2d = LinearDiscriminantAnalysis(n_components=2).fit(tfs_reduced, cluster
    print('LDA execution time: ', time.time() - start)
    return tfs_2d
```

tSNE

Summary:

In-depth explanation:

In [25]:
```
def dimReductiontSNE(tfs_reduced, perplexity=30, learning_rate=100):
    start = time.time()
    tfs_2d = TSNE(n_components=2, perplexity=perplexity, learning_rate=learning_
    print('tSNE execution time: ', time.time() - start)
    return tfs_2d
```

Analysis

In [35]:
```
def visualize(tfs=None,dimreduction='LSA', clustering='KMEANS', embedding2d='LDA

    if dimreduction == 'LSA':
        tfs_reduced, model = LSA(tfs, num_topics=num_topics)
    elif dimreduction == 'NMF':
        tfs_reduced, model = NMF(tfs, num_topics=num_topics)
    else:
        return 'No dimensionality reduction technique was selected!'

    if clustering == 'KMEANS':
        clusters = cluster(tfs_reduced, num_topics=num_clusters)
    else:
        return 'No clustering technique was selected!'

    if embedding2d == 'LDA':
        tfs_2d = dimReductionLDA(tfs_reduced, clusters=clusters)
    elif embedding2d == 'tSNE':
        tfs_2d = dimReductiontSNE(tfs_reduced, perplexity=perplexity, learning_r
    else:
        return 'No dimensionality reduction technique was selected!'

    tfidf_feature_names = [str(token) for token in tfidf_vectorizer.get_feature_
    [print(i, words) for i, words in get_top_words_cluster(model, clusters.clust

    # configure bokeh plot
    source = ColumnDataSource(data=dict(
    x=tfs_2d[:, 0],
    y=tfs_2d[:, 1],
    ids=list(cleanProjectTexts().keys()),
    titles= [loadProjects()[key]['titel'] for key in cleanProjectTexts().keys()]
    colours=np.array(d3['Category20'][num_clusters])[clusters.labels_]
    ))

    TOOLTIPS = [
    ("index", "$index"),
    ("id", "@ids"),
    ("title", "@titles"),
    ]

    p = figure(plot_width=800, plot_height=800,
           title=None, toolbar_location="below", tooltips=TOOLTIPS)
    p.scatter('x', 'y', size=10,color='colours', source=source)
    show(p)
```

In [36]:
```python
def s(x,y):
    return IntSlider(min=x,max=y, value=(y-x)//2, continuous_update=False)

w = interactive(visualize,tfs=fixed(tfs), dimreduction=['LSA', 'NMF'], clusterir
output = w.children[-1]
output.layout.height = '1500px'
display(w)
```

```
/home/tim/.local/share/virtualenvs/rpcserver-R6dNBFY-/lib/python3.6/site-packag
es/scipy/sparse/compressed.py:226: SparseEfficiencyWarning: Comparing sparse ma
trices using == is inefficient, try using != instead.
  " != instead.", SparseEfficiencyWarning)
```

dimreduction LSA

clustering KMEANS

embedding2d LDA

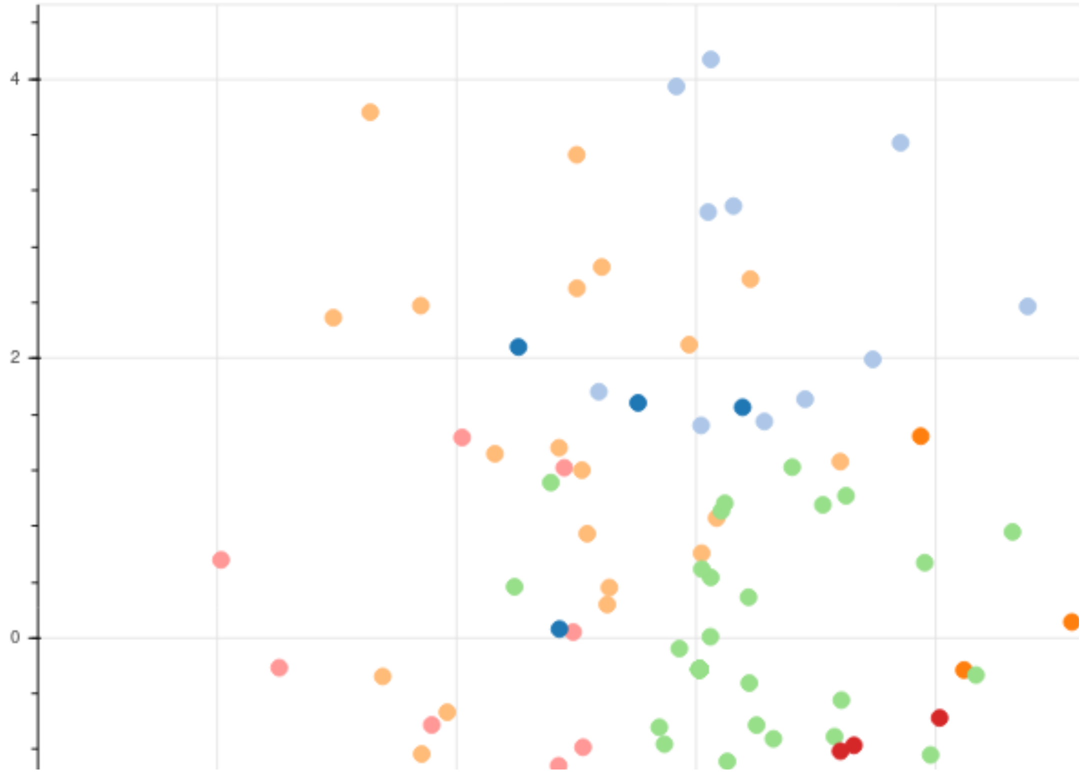
num_topics 19

num_clusters 8

perplexity 35

learning_rate 200

```
LSA execution time:  0.056895256042480047
Clustering execution time:  0.03601503372192383
LDA execution time:  0.0025475025177001953
0 ['fragestellungen', 'internationalen', 'erhaltung', 'thema', 'mischfauna']
1 ['monokotyledonen', 'morphoanatomie', 'clades', 'neues', 'ökophysiologische
n']
2 ['verhalten', 'direkt', 'körpergröße', 'fledermäusen', 'fledermäusen']
3 ['impaktors', 'absinken', 'kombinierte', 'fragmentierung', 'komponenten']
4 ['artensterben', 'erdgeschichte', 'einblicke', 'umschwunges', 'dynamik']
5 ['megaregolith', 'planeten', 'radiometrische', 'verteilung', 'datierung']
6 ['determinations', 'arten', 'flohkrebsen', 'bearbeitet', 'verbreitung']
7 ['übertragen', 'einheiten', 'capitan', 'evolutionsraten', 'stufe']
```



In [ ]:

In [ ]: