explore-topic-models

February 6, 2018

0.0.1 Visualize and Explore a LDA Topic Model

Some examples on how LDA topic models can be evaluated and interpreted. This notebook uses digitized text from *the Daedalus* - a yearly book from the Swedish National Museum of Science and Technology.

0.0.2 Introduction

TODO: Add purpose and scope of this notebook. Why is it hard to evaluate topic models? What metrics can be used?

References: Topic Model Diagnostics: Assessing Domain Relevance via Topical Alignment Jason Chuang, Sonal Gupta, Christopher D. Manning, Jeffrey Heer International Conference on Machine Learning (ICML), 2013 PDF Sup

PyData Berlin 2017 (Matti Lyra) NB

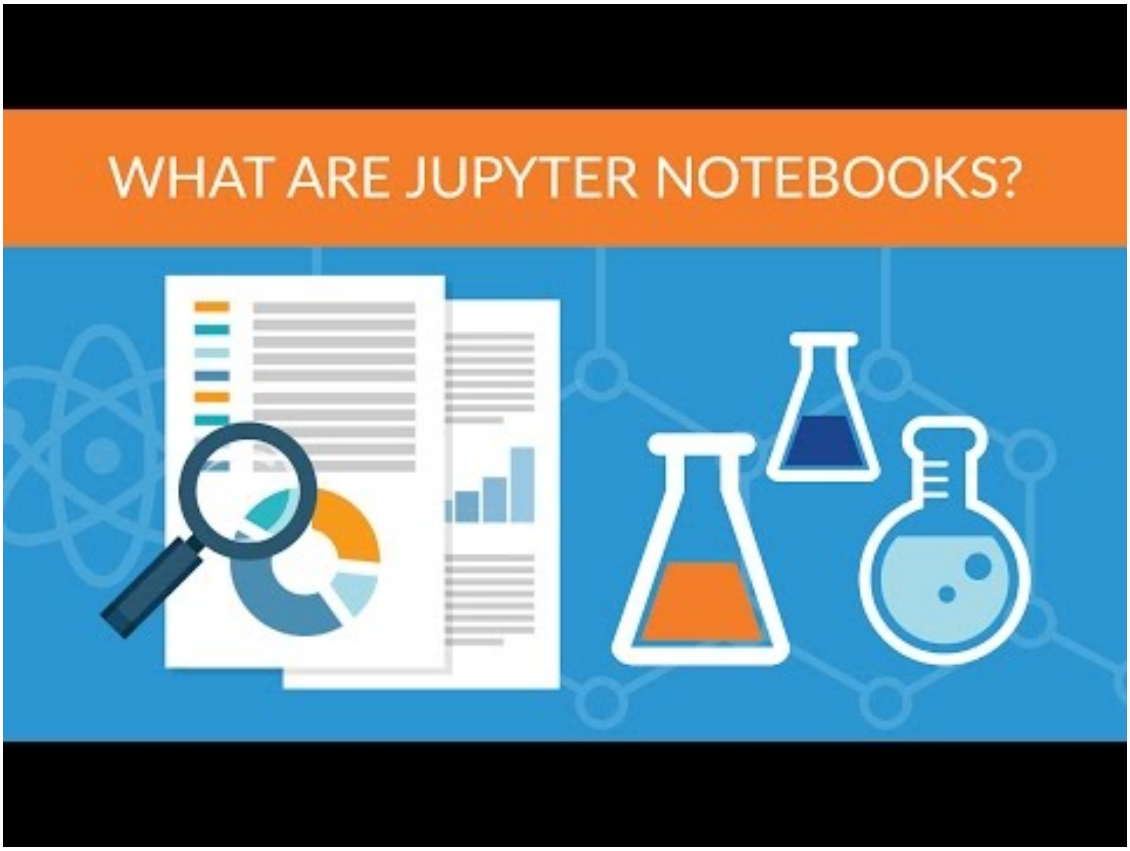
0.0.3 Brief Instructions on Jupyter Notebooks

Please see [add link] for an introduction on what Jupyter notebooksare and how to use them. In short, a notebook is a document with embedded executable code presented in a simple and easy to use web interface. Most important things to note are: - Click on the menu Help -> User Interface Tour for an overview of the Jupyter Notebook App user interface. - The **code cells** contains the script code (Python in this case, but can be other languages are also suported) and are the sections marked by **In [x]** in the left margin. It is marked as **In []** if it hasn't been executed, and as **In [n]** when it has been executed(n is an integer). A cell marked as **In [*]** is either executing, or waiting to be executed (i.e. other cells are executing). - The **current cell** is highlighted with a blue (or green if in "edit" mode) border. You make a cell current by clicking on it, - Code cells aren't executed automatically. Instead you execute the current cell by either pressing **shift+enter** or the **play** button in the toolbar. The output (or result) of a cell's execution is presented directly below the cell prefixed by **Out[n]**. - The next cell will automatically be selected (made current) after a cell has been executed. Repeatadly pressing **shift+enter** or the play button hence executes the cells in sequence. - You can run the entire notebook in a single step by clicking on the menu Cell -> Run All. Note that this can take some time to finish. You can see how cells are executed in sequence via the indicator in the margin (i.e. "In [*]" changes to "In [n]" where n is an integer). - The cells can be edited if they are double-clicked, in which case the cell border turns green. Use the ESC key to escape edit mode (or click on any other cell).

To restart the kernel (i.e. the computational engine assigned to your session), click on the menu Kernel -> Restart.

```
In [1]: from IPython.lib.display import YouTubeVideo
        YouTubeVideo("h9S4kN4l5Is", width=400, height=300)
```

Out[1]:



0.0.4 Setup and Initialize the Notebook

Import Python libraries and frameworks, and initialize the notebook.

```
In [2]: # MANDATORY! Import dependencies and setup notebook
        %run ./common/utility
        %run ./common/model_utility
        %run ./common/plot_utility
        %run ./common/widgets_utility
        %run ./common/network_utility
        %run ./common/vectorspace_utility

        import warnings
        warnings.filterwarnings("ignore", category=DeprecationWarning)
        warnings.filterwarnings("ignore", category=FutureWarning)
```

```
        import os
        import glob
        import math
        import types
        import ipywidgets as widgets
        import logging

        logger = logging.getLogger('explore-topic-models')

        from pivottablejs import pivot_ui
        from IPython.display import display, HTML, clear_output, IFrame
        from itertools import product
        # from IPython.core.interactiveshell import InteractiveShell
        # InteractiveShell.ast_node_interactivity = "all"

        %config IPCompleter.greedy=True
        %autosave 120

        import bokeh.models as bm
        import bokeh.palettes

        from bokeh.io import output_file, push_notebook
        from bokeh.core.properties import value, expr
        from bokeh.transform import transform, jitter
        from bokeh.layouts import row, column, widgetbox
        from bokeh.plotting import figure, show, output_notebook, output_file
        from bokeh.models.widgets import DataTable, DateFormatter, TableColumn

        import pandas as pd

        pd.set_option('display.height', 1000)
        pd.set_option('display.max_rows', 500)
        pd.set_option('display.max_columns', 500)
        pd.set_option('display.width', 1000)

        TOOLS = "pan,wheel_zoom,box_zoom,reset,previewsave"
        AGGREGATES = { 'mean': np.mean, 'sum': np.sum, 'max': np.max, 'std': np.std }

        output_notebook()

        pd.set_option('precision', 10)
```

```
Autosaving every 120 seconds
```

0.0.5 Select LDA Model to be Used in Subsequent Plots

Select one of the avaliable topic models stored in the ./data directory. New models are made avaliable by uploading them into seperate folders in ./data (for instance using Jupyter Lab). Note that the data must have been prepares with the **compute_lda_model.py** script, and all resulting files must be uploaded. Note that it can take some time (20-30 seconds) to load a model for the first due to large file sizes. Subsequent load is much faster since the system extracts data to CSV-files which gives faster loads.

Note! Subsequent cells are NOT updated automatically when a new model is selected. Instead you must use the **play** button, or press **Shift-Enter** to execute the current cell.

```
In [3]: # Current model state
        class ModelState:

            def __init__(self, data_folder):

                self.data_folder = data_folder
                self.basenames = ModelUtility.get_model_names(data_folder)
                self.basename = self.basenames[0]

            def set_model(self, basename=None):

                basename = basename or self.basename

                self.basename = basename
                self.topic_keys = ModelUtility.get_topic_keys(self.data_folder, basename)
                state.max_alpha = self.topic_keys.alpha.max()
                self.topic_overview = ModelUtility\
                    .get_result_model_sheet(self.data_folder, basename, 'topic_tokens')
                self.document_topic_weights = ModelUtility\
                    .get_result_model_sheet(self.data_folder, basename, 'doc_topic_weights')\
                    .drop('Unnamed: 0', axis=1)
                self.topic_token_weights = ModelUtility\
                    .get_result_model_sheet(self.data_folder, basename, 'topic_token_weights')\
                    .drop('Unnamed: 0', axis=1)
                self._years = list(range(
                    self.document_topic_weights.year.min(), self.document_topic_weights.year.ma
                self.min_year = min(self._years)
                self.max_year = max(self._years)
                self.years = [None] + self._years
                self.n_topics = self.document_topic_weights.topic_id.max() + 1
                # https://stackoverflow.com/questions/44561609/how-does-mallet-set-its-default
                self.initial_alpha = 5.0 / self.n_topics if 'mallet' in state.basename else 1.0
                self.initial_beta = 0.01 if 'mallet' in basename else 1.0 / self.n_topics
                filename = os.path.join(self.data_folder, basename, 'gensim_model_{}.gensim.gz
                self._lda = None #LdaModel.load(filename)
                self.topic_tokens_as_text = None
                self.corpus_documents = None
```

```
        print("Current model: " + self.basename.upper())
        # _fix_topictokens()
        return self

    def get_document_topic_weights(self, year=None, topic_id=None):
        df = self.document_topic_weights
        if year is None and topic_id is None:
            return df
        if topic_id is None:
            return df[(df.year == year)]
        if year is None:
            return df[(df.topic_id == topic_id)]
        return df[(df.year == year)&(df.topic_id == topic_id)]

    def get_unique_topic_ids(self):
        return self.document_topic_weights['topic_id'].unique()

    def get_topic_weight_by_year_or_document(self, key='mean', year=None):
        pivot_column = 'year' if year is None else 'document_id'
        df = self.get_document_topic_weights(year) \
            .groupby([pivot_column,'topic_id']) \
            .agg(AGGREGATES[key])[['weight']].reset_index()
        return df, pivot_column

    def get_lda(self):
        if self._lda is None:
            filename = 'gensim_model_{}.gensim.gz'.format(self.basename)
            self._lda = LdaModel.load(os.path.join(self.data_folder, self.basename, fi
            print('LDA model loaded...')
        return self._lda

    def get_topics_tokens_as_text(self, n_words=100, cache=True):
        if cache and self.topic_tokens_as_text is not None:
            return self.topic_tokens_as_text
        topic_tokens_as_text = ModelUtility.get_topics_tokens_as_text(state.topic_toke
        if cache:
            self.topic_tokens_as_text = topic_tokens_as_text
        return topic_tokens_as_text

    def get_topic_tokens(self, topic_id, max_n_words=500):
        tokens = state.topic_token_weights\
            .loc[lambda x: x.topic_id == topic_id]\
            .sort_values('weight',ascending=False)[:max_n_words]
        return tokens

    def get_topic_alphas(self):
        tokens = state.topic_token_weights\
            .loc[lambda x: x.topic_id == topic_id]\
```

```
                    .sort_values('weight',ascending=False)[:max_n_words]
                alpas = ModelUtility.get_topic_alphas
                return tokens

            def get_topic_year_aggregate_weights(self, fn, threshold):
                df = self.document_topic_weights
                #df = df[(df.weight>=threshold)]
                df = df.groupby(['year', 'topic_id']).agg(fn)['weight'].reset_index()
                df = df[(df.weight>=threshold)]
                return df

            def get_topic_proportions(self):
                corpus_documents = self.get_corpus_documents()
                document_topic_weights = self.get_document_topic_weights()
                topic_proportion = ModelUtility.compute_topic_proportions(document_topic_weight
                return topic_proportion

            def get_corpus_documents(self):
                if self.corpus_documents is None:
                    self.corpus_documents = ModelUtility.get_corpus_documents(self.data_folder
                return self.corpus_documents

        state = ModelState('./tm-data')

        wdg_basename = widgets.Dropdown(
            options=state.basenames,
            value=state.basename,
            description='Topic model',
            disabled=False,
            layout=widgets.Layout(width='75%')
        )
        wdg_model = widgets.interactive(state.set_model, basename=wdg_basename)
        display(widgets.VBox((wdg_basename,) + (wdg_model.children[-1],)))
        wdg_model.update()
```

```
VBox(children=(Dropdown(description='Topic model', layout=Layout(width='75%'), options=('topics
```

0.0.6 Topic Distribution Hyperparameter Alpha

See http://psiexp.ss.uci.edu/research/papers/SteyversGriffithsLSABookFormatted.pdf for a description of LDA hyperparamers. The **alpha** hyperparameter affects the sparsity of the document-topic distribution. The LDA model is said to be *symmetric* if the same alpha value is used for all topics, and *assymetric* if it can vary per topic. In the case of assymetric topic modelloing, high alphas can indicate topics that contains stopwords (common words), and low values can indicate bogus topics. This chart is of no value for symmetric models. Topics that have ab alpha value lower than the initial alpha value (default 1 / number of topics) are marked in red (not significant topics).

In [4]:
```
# Alpha / Lambda Plot
_topic_keys = ModelUtility.get_topic_keys(state.data_folder, state.basename)

def plot_alpha(df):

    source = ColumnDataSource(df)
    p = figure(x_range=df.topic.values,plot_width=900, plot_height=400, title='',
               tools=TOOLS, toolbar_location="above")
    p.xaxis[0].axis_label = 'Topic'
    p.yaxis[0].axis_label = 'Alpha'
    p.xaxis.major_label_orientation = 1.0
    p.y_range.start = 0.0
    x_axis_type = 'enum'
    p.xgrid.visible = False

    glyph = bm.glyphs.VBar(x='topic', top='alpha', bottom=0, width=0.5, fill_color='co
    cr = p.add_glyph(source, glyph)

    titles = ModelUtility.get_topic_titles(state.topic_token_weights, n_words=100)
    p.add_tools(bm.HoverTool(tooltips=None, callback=WidgetUtility.glyph_hover_callbac
        source, 'topic_id', titles.index, titles, 'alpha_plot'), renderers=[cr]))

    return p

def display_alpha(output_format, sort_by, window):
    global state
    palette = bokeh.palettes.PiYG[4]
    topic_keys = ModelUtility.get_topic_keys(state.data_folder, state.basename).reset_
    topic_keys = topic_keys[((topic_keys.alpha >= window[0]) & (topic_keys.alpha <= wi
    topic_keys['topic'] = topic_keys.topic_id.apply(lambda x: str(x))
    topic_keys['color'] = topic_keys.alpha.apply(lambda x: palette[1] if x >= state.in
    if sort_by.lower() == 'alpha':
        topic_keys = topic_keys.sort_values('alpha', axis=0)
    if output_format == 'Chart':
        p = plot_alpha(topic_keys)
        show(p)
    else:
        source = bm.ColumnDataSource(topic_keys)
        columns = [
            TableColumn(field="topic_id", title="ID"),
            TableColumn(field="alpha", title="Alpha"),
            TableColumn(field="tokens", title="Tokens"),
        ]
        data_table = DataTable(source=source, columns=columns, width=950, height=600)
        show(widgetbox(data_table))

za = BaseWidgetUtility(
    text_id='alpha_plot',
```

```
            text=wf.create_text_widget('alpha_plot',default_value='Hover topics to display word
            output_format=wf.create_select_widget('Format', ['Chart', 'Table'], default='Chart
            sort_by=wf.create_select_widget('Sort by', ['Topic', 'Alpha'], default='Alpha'),
            window=widgets.FloatRangeSlider(
                description='Window',
                min=0, max=state.max_alpha + 0.1,
                step=0.01,
                value=(state.initial_alpha, state.max_alpha + 0.1),
                continuous_update=False
            )
        )
        za.next_topic_id = za.create_next_id_button('topic_id', state.n_topics)

        wa = widgets.interactive(
            display_alpha,
            output_format=za.output_format,
            sort_by=za.sort_by,
            window=za.window
        )
        za.text.layout = widgets.Layout(width='95%', height='120px')
        wa.children[-1].layout = widgets.Layout(width='98%')

        display(widgets.VBox([
            za.text,
            widgets.HBox([za.output_format, za.window, za.sort_by]),
            widgets.HBox([wa.children[-1]])
        ]))
        wa.update()
```

```
VBox(children=(HTML(value="<span class='alpha_plot'>Hover topics to display words!</span>", lay
```

0.0.7 Documents' Topic-Weight Distribution

This graph displays **the distribution of document topic-weights** for the selected model. The X-axis percentage value between 0 and 100 and the Y-axis is the number of document topic-weights for each (integer) percentage. Not surprisingly is that the vast majority (97-98)% of the weights are zero, or close to zero.

```
In [5]: # Topic Weight Distribution
        topic_weights_distribution = state.get_document_topic_weights()
        topic_weights_distribution['weight%'] = (topic_weights_distribution.weight * 100).asty
        topic_weights_distribution = topic_weights_distribution.groupby('weight%').size()
        topic_weights_count = topic_weights_distribution.sum()

        def display_topic_weights_distribution(p_range):
            global topic_weights_distribution
            selection = topic_weights_distribution[p_range[0]:p_range[1]+1]
```

```
        title = '{0:.2f}% of all document-topic weights are within selected interval'\
              .format(100 * (selection.sum() / topic_weights_count))
        selection.plot(figsize=(12,6), title=title, kind='line', xlim=(0,100))

    p_range = widgets.SelectionRangeSlider(
        options=range(0,100), index=(0,99), description='Interval', continues_update=False
    )

    w = widgets.interactive(display_topic_weights_distribution, p_range=p_range)
    display(widgets.VBox(
        (p_range,) +
        (w.children[-1],)))
    w.update()
```

```
VBox(children=(SelectionRangeSlider(description='Interval', index=(0, 99), options=(0, 1, 2, 3
```

0.0.8 Generate Topic Wordclouds

In [6]:
```
# Display LDA topic's token wordcloud
opts = { 'max_font_size': 100, 'background_color': 'white', 'width': 900, 'height': 800

zwc = BaseWidgetUtility(
    n_topics=state.n_topics,
    text_id='wc01',
    text=wf.create_text_widget('wc01'),
    topic_id=wf.topic_id_slider(state.n_topics),
    word_count=wf.word_count_slider(1, 500),
    output_format=wf.create_select_widget('Format', ['Wordcloud', 'List', 'Pivot'], de
    progress = wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widg
)
zwc.prev_topic_id = zwc.create_prev_id_button('topic_id', state.n_topics)
zwc.next_topic_id = zwc.create_next_id_button('topic_id', state.n_topics)

def display_wordcloud(topic_id=0, n_words=100, output_format='Wordcloud'):
    global state, zwc
    zwc.progress.value = 1
    df_temp = state.topic_token_weights
    df_temp = df_temp.loc[(df_temp.topic_id == topic_id)]
    tokens = state.get_topics_tokens_as_text(n_words=n_words, cache=True).iloc[topic_id
    zwc.progress.value = 2
    zwc.text.value = 'ID {}: {}'.format(topic_id, tokens)
    if output_format == 'Wordcloud':
        WordcloudUtility.plot_wordcloud(df_temp, 'token', 'weight', max_words=n_words,
    elif output_format == 'List':
        zwc.progress.value = 3
        df_temp = state.get_topic_tokens(topic_id, n_words)
        zwc.progress.value = 4
```

```
            display(HTML(df_temp.to_html()))
        else:
            display(pivot_ui(state.get_topic_tokens(topic_id, n_words)))
        zwc.progress.value = 0

    iw = widgets.interactive(
        display_wordcloud,
        topic_id=zwc.topic_id,
        n_words=zwc.word_count,
        output_format=zwc.output_format)

    zwc.text.layout = widgets.Layout(width='95%', height='120px')

    display(widgets.VBox(
        (zwc.text,) +
        (widgets.HBox((zwc.prev_topic_id,) + (zwc.next_topic_id,) +
                      (zwc.topic_id,) + (zwc.word_count,) + (zwc.output_format,)),) +
        (zwc.progress,) +
        (iw.children[-1],)))
    iw.update()
```

```
VBox(children=(HTML(value="<span class='wc01'></span>", layout=Layout(height='120px', width='95
```

0.0.9 Display Topic's Word Distribution

The following chart shows the word distribution for each selected topic. You can zoom in on the left chart. Hugg av (eller markera) var/när orden faller under **initialvärdet** för beta.

In [7]:
```
# Display topic's word distribution

    def plot_tokens(tokens, **args):

        source = ColumnDataSource(tokens)

        p = figure(toolbar_location="right", **args)

        cr = p.circle(x='xs', y='ys', source=source)

        label_style = dict(level='overlay', text_font_size='8pt', angle=np.pi/6.0)

        text_aligns = ['left', 'right']
        for i in [0, 1]:
            label_source = ColumnDataSource(tokens.iloc[i::2])
            labels = bm.LabelSet(x='xs', y='ys', text_align=text_aligns[i], text='token',
                                 y_offset=5*(1 if i == 0 else -1),
                                 x_offset=5*(1 if i == 0 else -1),
                                 source=label_source, **label_style)
```

```
        p.add_layout(labels)

    p.xaxis[0].axis_label = 'Token #'
    p.yaxis[0].axis_label = 'Probability%'
    p.ygrid.grid_line_color = None
    p.xgrid.grid_line_color = None
    p.axis.axis_line_color = None
    p.axis.major_tick_line_color = None
    p.axis.major_label_text_font_size = "6pt"
    p.axis.major_label_standoff = 0
    return p


def plot_topic_tokens_charts(tokens, flag=True):

    if flag:
        left = plot_tokens(tokens, plot_width=900, plot_height=600, title='', tools='bo
        show(left)
        return

    left = plot_tokens(tokens, plot_width=450, plot_height=500, title='', tools='box_z
    right = plot_tokens(tokens, plot_width=450, plot_height=500, title='', tools='pan'

    source = ColumnDataSource({'x':[], 'y':[], 'width':[], 'height':[]})
    left.x_range.callback = create_js_callback('x', 'width', source)
    left.y_range.callback = create_js_callback('y', 'height', source)

    rect = bm.Rect(x='x', y='y', width='width', height='height', fill_alpha=0.0, line_
    right.add_glyph(source, rect)

    show(row(left, right))

def display_topic_tokens(topic_id=0, n_words=100, output_format='Wordcloud'):
    global state, g
    g.progress.value = 1
    tokens = state.get_topic_tokens(topic_id=topic_id).\
        copy()\
        .drop('topic_id', axis=1)\
        .assign(weight=lambda x: 100.0 * x.weight)\
        .sort_values('weight', axis=0, ascending=False)\
        .reset_index()\
        .head(n_words)
    if output_format == 'Wordcloud':
        g.progress.value = 3
        tokens = tokens.assign(xs=tokens.index, ys=tokens.weight)
        plot_topic_tokens_charts(tokens)
        g.progress.value = 4
    elif output_format == 'List':
```

```
            #display(tokens)
            display(HTML(tokens.to_html()))
        else:
            display(pivot_ui(tokens))
        g.progress.value = 0

    g = BaseWidgetUtility(
        n_topics=state.n_topics,
        text_id='wc01',
        text=wf.create_text_widget('wc01'),
        topic_id=wf.create_int_slider(description='Topic ID', min=0, max=state.n_topics -
        word_count=wf.create_int_slider(description='Word count', min=1, max=500, step=1,
        output_format=wf.create_select_widget('Format', ['Wordcloud', 'List', 'Pivot'], de
        progress = wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widg

    )
    g.prev_topic_id = g.create_prev_id_button('topic_id', state.n_topics)
    g.next_topic_id = g.create_next_id_button('topic_id', state.n_topics)

    w = widgets.interactive(
        display_topic_tokens, topic_id=g.topic_id, n_words=g.word_count, output_format=g.ou
    )

    display(widgets.VBox(
        (g.text,) +
        (widgets.HBox((g.prev_topic_id,) + (g.next_topic_id,) +
            (g.topic_id,) + (g.word_count,) + (g.output_format,)),) +
        (g.progress, ) +
        (w.children[-1],)))

    w.update()
```

```
VBox(children=(HTML(value="<span class='wc01'></span>", placeholder=''), HBox(children=(Button
```

0.0.10 Plot Topic's Weight Over Time

Display a specific topics share over time as well as listing topic terms in descending order (based on yearly mean weight over all documents). The *whisker* displays max and mean topic weight for given year.

In [8]:
```
# Plot a topic's yearly weight over time in selected LDA topic model

    def plot_topic_over_time(df, pivot_column, value_column, topic_id=0, year=None, whiske

        source = ColumnDataSource(df)
        p = figure(plot_width=900, plot_height=600, title='', tools=TOOLS, toolbar_locatio
        p.xaxis[0].axis_label = pivot_column.title()
```

```
    p.yaxis[0].axis_label = value_column.title() + ('weight' if value_column != 'weight
    p.y_range.start = 0.0
    p.y_range.end = 1.0

    day_width = 60*60*24*1000
    glyph = bm.glyphs.VBar(x=pivot_column, top=value_column, bottom=0, width=1, fill_co
    p.add_glyph(source, glyph)
    if whisker and year is None:
        p.add_layout(
            bm.Whisker(source=source, base=pivot_column, upper="max", lower=value_colum
        )
    #if not year is None: print(df_temp[['index', 'document', 'topic_id', 'weight']])
    return p

def display_topic_over_time(topic_id, year, value_column):
    global state, zj

    tokens = state.get_topics_tokens_as_text(n_words=200, cache=True).iloc[topic_id]
    zj.text.value = 'ID {}: {}'.format(topic_id, tokens)

    pivot_column = 'year' if year is None else 'document_id'
    value_column = value_column if year is None else 'weight'

    df = state.document_topic_weights[(state.document_topic_weights.topic_id==topic_id)

    if year is None:
        df = df.groupby([pivot_column, 'topic_id']).agg([np.mean, np.max, np.std])['wei
        df.columns = ['year', 'topic_id', 'mean', 'max', 'std']
    else:
        df = df[(df.year==year)]

    p = plot_topic_over_time(df, pivot_column, value_column, topic_id, year,  False)
    show(p)

zj = BaseWidgetUtility(
    n_topics=state.n_topics,
    text_id='topic_share_plot',
    text=wf.create_text_widget('topic_share_plot'),
    year=wf.create_select_widget('Year', options=state.years),
    topic_id=wf.create_int_slider(description='Topic ID', min=0, max=state.n_topics - 1
    output_format=wf.create_select_widget('Format', ['Wordcloud', 'List', 'Pivot'], def
    progress=wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widget
    aggregate=wf.create_select_widget('Aggregate', list(AGGREGATES.keys()), 'max')

)
zj.prev_topic_id = zj.create_prev_id_button('topic_id', state.n_topics)
zj.next_topic_id = zj.create_next_id_button('topic_id', state.n_topics)
```

```
        wj = widgets.interactive(
            display_topic_over_time,
            topic_id=zj.topic_id,
            year=zj.year,
            value_column=zj.aggregate
        )

        display(widgets.VBox(
            (zj.text,) +
            (widgets.HBox((zj.prev_topic_id,) + (zj.next_topic_id,) + (zj.topic_id,) + (zj.yea
            (zj.progress,) +
            (wj.children[-1],)))
        wj.update()
```

```
VBox(children=(HTML(value="<span class='topic_share_plot'></span>", placeholder=''), HBox(chil
```

0.0.11 Plot Stacked Bar of Most Relevant Topics

Display topic shares in descending order as a stacked bar chart. Order is based on selected aggregate function.

In [9]:
```
# Plot topic shares (year aggregate or per document for selected year)

        def prepare_stacked_topic_share_data(key, n_topics, year):
            global state
            pivot_column = 'year' if year is None else 'document_id'

            df_data = state.get_document_topic_weights(year)

            df = ModelUtility.get_document_topic_weights_pivot(df_data, AGGREGATES[key], pivot_
            df.set_index(pivot_column, inplace=True)

            n_topics = min(len(df.columns), n_topics)
            topic_toplist = df[df.columns].sum().sort_values(axis=0, ascending=False)
            df_top = df[topic_toplist[:n_topics].index].copy()

            df = df_top.reset_index()
            df.columns = [ str(x) for x in  df.columns ]

            return df, pivot_column, n_topics

        def generate_category_colors(n_items, palette=bokeh.palettes.Category20[20]):
            ''' Repeat palette to get n_items colors '''
            colors = (((n_items // len(palette)) + 1) * palette)[:n_items]
            return colors

        def plot_stacked_bar_of_topic_over_time(df, pivot_column, key='mean', n_topics=3, year=
```

```
    categories = list(df.columns[1:])
    colors = generate_category_colors(n_topics)
    source = ColumnDataSource(df)

    p = figure(plot_width=900, plot_height=800, title=state.basename, tools=TOOLS, tool

    p.xaxis[0].axis_label = key.title() + ' weight'
    p.yaxis[0].axis_label = pivot_column.title()

    #legend = [ value(x) for x in categories ]
    #p.hbar_stack(categories, y=pivot_column, source=source, color=colors, height=0.5,

    bottoms, tops = [], []
    for i, category in enumerate(categories):
        tops = tops + [category]
        cr = p.hbar(y=pivot_column,
                    left=expr(bm.expressions.Stack(fields=bottoms)),
                    right=expr(bm.expressions.Stack(fields=tops)),
                    color=colors[i],
                    height=0.5,
                    source=source,
                    legend='Topic ' + str(category))
        topic_id = int(category)
        tooltip = 'ID {}: {}'.format(topic_id, state.get_topics_tokens_as_text(n_words=
        p.add_tools(bm.HoverTool(tooltips=tooltip, renderers=[cr]))
        bottoms = bottoms + [category]

    return p

def display_stacked_bar_of_topic_over_time(key='max', n_topics=3, year=None, output_fo

    global state

    ''' Prepare the plot data '''

    df, pivot_column, n_topics = prepare_stacked_topic_share_data(key, n_topics, year)

    if output_format == 'Wordcloud':
        p = plot_stacked_bar_of_topic_over_time(df, pivot_column, key, n_topics, year)
        show(p)
    elif output_format == 'List':
        #display(tokens)
        display(HTML(df.to_html()))
    else:
        display(pivot_ui(df))

zh = BaseWidgetUtility(
```

```
            n_topics=state.n_topics,
            text_id='topic_share_plot',
            text=wf.create_text_widget('topic_share_plot'),
            year=wf.create_select_widget('Year', options=state.years),
            topics_count=wf.create_int_slider(description='Topic count', min=1, max=state.n_to
            output_format=wf.create_select_widget('Format', ['Wordcloud', 'List', 'Pivot'], de
            progress=wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widge
            aggregate=wf.create_select_widget('Aggregate', list(AGGREGATES.keys()), 'max')
        )
        wh = widgets.interactive(
            display_stacked_bar_of_topic_over_time, n_topics=zh.topics_count,
            key=zh.aggregate, year=zh.year, output_format=zh.output_format
        )

        display(widgets.VBox(
            (zh.text,) +
            (widgets.HBox((zh.aggregate,) + (zh.topics_count,) + (zh.year,) + (zh.output_forma
            (zh.progress,) +
            (wh.children[-1],)))

        wh.update()
```

```
VBox(children=(HTML(value="<span class='topic_share_plot'></span>", placeholder=''), HBox(chil
```

0.0.12 Display Document-Topic Weights

List aggregated topic weights.

```
In [10]: # Folded code
         import IPython.display # import display, HTML
         pd.set_option('display.height', 1000)
         pd.set_option('display.max_rows', 500)
         pd.set_option('display.max_columns', 500)
         pd.set_option('display.width', 1000)

         def plot_stacked_bar_of_topic_over_time(key='mean', year=None, output_format=None):
             global state
             pivot_column = 'year' if year is None else 'document_id'
             df_data = state.get_document_topic_weights(year)
             df_temp = ModelUtility.get_document_topic_weights_pivot(df_data, AGGREGATES[key],
             df_temp.set_index(pivot_column, inplace=True)
             df_temp.columns = [ str(x) for x in df_temp.columns ]
             if output_format == 'List':
                 # print(df_temp.columns)
                 display(HTML(df_temp.to_html()))
             else:
                 display(pivot_ui(df_temp, rows=['year']+list(df_temp.columns)))
```

```
        zk = BaseWidgetUtility(
            n_topics=state.n_topics,
            text_id='topic_share_plot',
            text=wf.create_text_widget('topic_share_plot'),
            year=wf.create_select_widget('Year', options=state.years),
            output_format=wf.create_select_widget('Format', ['List', 'Pivot'], default='List')
            aggregate=wf.create_select_widget('Aggregate', list(AGGREGATES.keys()), 'max')
        )

        wk = widgets.interactive(
            plot_stacked_bar_of_topic_over_time, key=zk.aggregate, year=zk.year, output_format
        )

        display(widgets.VBox((zk.text,) +
               (widgets.HBox((zk.aggregate,) + (zk.year,) + (zk.output_format,)),) +
               (wk.children[-1],)))

        wk.update()
```

```
VBox(children=(HTML(value="<span class='topic_share_plot'></span>", placeholder=''), HBox(child
```

0.0.13 Scatter plot (or heatmap) of topic shares per year or document

Display topic shares as a scatter plot using gradient color for topic's weight.

```
In [11]: # plot_topic_relevance_by_year

        def setup_glyph_coloring(df):
            max_weight = df.weight.max()
            #colors = list(reversed(bokeh.palettes.Greens[9]))
            colors = ["#efefef", "#75968f", "#a5bab7", "#c9d9d3", "#e2e2e2", "#dfccce", "#ddb
                      "#933b41", "#550b1d"]
            mapper = bm.LinearColorMapper(palette=colors, low=df.weight.min(), high=max_weight
            color_transform = transform('weight', mapper)
            color_bar = bm.ColorBar(color_mapper=mapper, location=(0, 0),
                                    ticker=bm.BasicTicker(desired_num_ticks=len(colors)),
                                    formatter=bm.PrintfTickFormatter(format=" %5.2f"))
            return color_transform, color_bar

        def plot_topic_relevance_by_year(df, xs, ys, glyph, titles, text_id):

            ''' Setup axis categories '''
            x_range = list(map(str, df[xs].unique()))
            y_range = list(map(str, df[ys].unique()))

            ''' Setup coloring and color bar '''
```

```
    color_transform, color_bar = setup_glyph_coloring(df)

    source = ColumnDataSource(df)

    plot_height = max(len(y_range) * 6, 300)
    p = figure(title="Topic heatmap", toolbar_location=None, tools="", x_range=x_rang
           y_range=y_range, x_axis_location="above", plot_width=900, plot_height=plot_

    args = dict(x=xs, y=ys, source=source, alpha=1.0, hover_color='red')

    if glyph == 'Circle':
        cr = p.circle(color=color_transform, **args)
    else:
        cr = p.rect(width=1, height=1, line_color=None, fill_color=color_transform, *

    p.x_range.range_padding = 0
    p.ygrid.grid_line_color = None
    p.xgrid.grid_line_color = None
    p.axis.axis_line_color = None
    p.axis.major_tick_line_color = None
    p.axis.major_label_text_font_size = "5pt"
    p.axis.major_label_standoff = 0
    p.xaxis.major_label_orientation = 1.0
    p.add_layout(color_bar, 'right')

    p.add_tools(bm.HoverTool(tooltips=None, callback=WidgetUtility.glyph_hover_callba
        source, 'topic_id', titles.index, titles, text_id), renderers=[cr]))

    return p

def display_topic_relevance_by_year(key='max', year=None, glyph='Circle'):
    global state, zo
    zo.progress.value = 1
    titles = ModelUtility.get_topic_titles(state.topic_token_weights, n_words=100)
    zo.progress.value = 2
    df, pivot_column = state.get_topic_weight_by_year_or_document(key=key, year=year)
    zo.progress.value = 3
    df[pivot_column] = df[pivot_column].astype(str)
    df['topic_id'] = df.topic_id.astype(str)
    zo.progress.value = 4
    p = plot_topic_relevance_by_year(df, xs=pivot_column, ys='topic_id', glyph=glyph,
                                     titles=titles, text_id='topic_relevance')
    show(p)
    zo.progress.value = 0

#u = TopTopicWidgets(0, state.years, aggregates=list(AGGREGATES.keys()), text_id='top

zo = BaseWidgetUtility(
```

```
            text_id='topic_relevance',
            text=wf.create_text_widget('topic_relevance'),
            year=wf.create_select_widget('Year', options=state.years),
            # output_format=wf.create_select_widget('Format', ['List', 'Pivot'], default='Lis
            aggregate=wf.create_select_widget('Aggregate', list(AGGREGATES.keys()), 'max'),
            progress=wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widg
        )

        zo.glyph = widgets.Dropdown(options=['Circle', 'Square'], value='Circle', description=

        wo = widgets.interactive(display_topic_relevance_by_year, key=zo.aggregate, year=zo.y

        display(widgets.VBox(
            (widgets.HBox((zo.aggregate,) + (zo.glyph,) + (zo.year,)),) +
            (zo.progress,) +
            (zo.text,) +
            (wo.children[-1],)))

        wo.update()
```

```
VBox(children=(HBox(children=(Dropdown(description='Aggregate', index=2, options=('sum', 'std'
```

0.0.14 Plot Topic-to-Document Associations as a Network Visualization

In [12]:
```
        # Visualize year-to-topic correlations by means of topic-document-weight dimensional

        def plot_topic_year_network(network, layout, scale=1.0, titles=None):

            year_nodes, topic_nodes = NetworkUtility.get_bipartite_node_set(network, bipartit

            year_source = NetworkUtility.get_node_subset_source(network, layout, year_nodes)
            topic_source = NetworkUtility.get_node_subset_source(network, layout, topic_nodes
            lines_source = NetworkUtility.get_edges_source(network, layout, scale=6.0, normal

            edges_alphas = NetworkMetricHelper.compute_alpha_vector(lines_source.data['weight

            lines_source.add(edges_alphas, 'alphas')

            p = figure(plot_width=900, plot_height=900, x_axis_type=None, y_axis_type=None, t

            r_lines = p.multi_line(
                'xs', 'ys', line_width='weights', alpha='alphas', color='black', source=lines
            )
            r_years = p.circle(
                'x','y', size=40, source=year_source, color='lightgreen', level='overlay', li
            )
```

```
        r_topics = p.circle('x','y', size=25, source=topic_source, color='skyblue', level=

        p.add_tools(bm.HoverTool(renderers=[r_topics], tooltips=None, callback=WidgetUtili
            glyph_hover_callback(topic_source, 'node_id', text_ids=titles.index, text=titl
        )

        text_opts = dict(
            x='x', y='y', text='name', level='overlay',
            x_offset=0, y_offset=0, text_font_size='8pt'
        )

        p.add_layout(
            bm.LabelSet(
                source=year_source, text_color='black', text_align='center', text_baseline
            )
        )
        p.add_layout(
            bm.LabelSet(
                source=topic_source, text_color='black', text_align='center', text_baseli
            )
        )

        return p

    def display_topic_year_network(
        layout_algorithm, threshold=0.10, scale=1.0, year=None, output_format='Network'
    ):
        global state, zn
        zn.progress.value = 1
        titles = state.get_topics_tokens_as_text()
        df = state.get_document_topic_weights(year=year, topic_id=None)
        df = df[(df.weight >= threshold)]
        zn.progress.value = 2

        network = NetworkUtility.create_bipartite_network(df, 'document', 'topic_id')
        zn.progress.value = 3

        if output_format == 'Network':
            args = PlotNetworkUtility.layout_args(layout_algorithm, network, scale)
            layout = (layout_algorithms[layout_algorithm])(network, **args)
            zn.progress.value = 4
            p = plot_topic_year_network(network, layout, scale=scale, titles=titles)
            show(p)

        elif output_format == 'List':
            display(HTML(df.to_html()))
        else:
            display(pivot_ui(df))
```

```
            zn.progress.value = 0

        zn = BaseWidgetUtility(
            n_topics=state.n_topics,
            text_id='nx_id1',
            text=wf.create_text_widget('nx_id1'),
            year=wf.create_int_slider(
                description='Year', min=state.min_year, max=state.max_year, step=1, value=stat
            ),
            scale=wf.create_float_slider('Scale', min=0.0, max=1.0, step=0.01, value=0.1),
            threshold=wf.create_float_slider('Threshold', min=0.0, max=1.0, step=0.01, value=0
            output_format=wf.create_select_widget('Format', ['Network', 'List', 'Pivot'], defa
            layout=wf.create_select_widget('Layout', list(layout_algorithms.keys()), default=
            progress=wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widge
        )
        zn.previous = zn.create_prev_id_button('year', 10000)
        zn.next = zn.create_next_id_button('year', 10000)

        wn = widgets.interactive(
            display_topic_year_network, layout_algorithm=zn.layout,
            threshold=zn.threshold, scale=zn.scale,
            year=zn.year, output_format=zn.output_format
        )

        display(widgets.VBox(
            (zn.text, ) +
            (widgets.HBox((zn.layout, ) + (zn.year,) + (zn.previous,) + (zn.next,)),) +
            (widgets.HBox((zn.threshold,) + (zn.scale,) + (zn.output_format,)),) +
            (zn.progress, ) +
            (wn.children[-1],)))

        wn.update()
```

```
VBox(children=(HTML(value="<span class='nx_id1'></span>", placeholder=''), HBox(children=(Dropd
```

0.0.15 Topic Co-Occurence

```
In [13]: # Visualize topic co-occurrence
         %run ./common/plot_utility
         G = None
         def display_topic_co_occurrence_network(layout, threshold, scale, output_format):

             global state, zn

             metric = 'Threshold'
             titles = state.get_topics_tokens_as_text()
```

```
    if metric == 'Threshold':
        df = state.get_document_topic_weights()
        df = df.loc[(df.weight >= threshold)]
        df = pd.merge(df, df, how='inner', left_on='document_id', right_on='document_
        df = df.loc[(df.topic_id_x < df.topic_id_y)]
        df = df.groupby([df.topic_id_x, df.topic_id_y]).size().reset_index()
        df.columns = ['source', 'target', 'weight']

    if output_format == 'Network':
        network = NetworkUtility.create_network(df, source_field='source', target_fiel
        p = PlotNetworkUtility.plot_network(
            network=network,
            layout_algorithm=layout,
            scale=scale,
            threshold=0.0,
            node_description=state.get_topics_tokens_as_text(),
            node_proportions=state.get_topic_proportions(),
            weight_scale=10.0,
            normalize_weights=True,
            element_id='cooc_id'
        )
        show(p)
    elif output_format == 'List':
        display(HTML(df.to_html()))
    else:
        display(pivot_ui(df))

zn = BaseWidgetUtility(
    n_topics=state.n_topics,
    text_id='cooc_id',
    text=wf.create_text_widget('cooc_id'),
    scale=wf.create_float_slider('Scale', min=0.0, max=1.0, step=0.01, value=0.1),
    threshold=wf.create_float_slider('Threshold', min=0.0, max=1.0, step=0.01, value=
    output_format=wf.create_select_widget('Format', ['Network', 'List', 'Pivot'], defa
    layout=wf.create_select_widget('Layout', list(layout_algorithms.keys()), default=
    progress=wf.create_int_progress_widget(min=0, max=4, step=1, value=0, layout=widg
)

wn = widgets.interactive(
    display_topic_co_occurrence_network,
    layout=zn.layout,
    threshold=zn.threshold,
    scale=zn.scale,
    output_format=zn.output_format
)

display(widgets.VBox(
```

```
            (zn.text, ) +
            (widgets.HBox((zn.layout, )),) +
            (widgets.HBox((zn.threshold,) + (zn.scale,) + (zn.output_format,)),) +
            (zn.progress, ) +
            (wn.children[-1],)))

        wn.update()
```

```
VBox(children=(HTML(value="<span class='cooc_id'></span>", placeholder=''), HBox(children=(Dro
```

0.0.16 Topic Similarity Network

This plot displays topic similarity based on **euclidean or cosine distances** between the **topic-to-word vectors**. Please note that the computations can take some time to exceute, especially for larger LDA models.

1. Compute a multi dimensional topic vector space based on the top n words for each topic. Since the subset of words differs, and their positions differs between topics they need to be aligned in common space so that 1) each vector has the same dimension (i.e. number of unique top n tokens over all topics) and 2) each token has the same position within that space. (using sklearn DictVectorizer). The vector space will have as many dimensions as the number of unique top n words over all topics.
2. Reduce the topic vector space into a 2D space (using sklearn PCA)
3. Normalize the 2D space (sklearn Normalizer)

- TODO: Save network to file (either via pandas or networkx)
- TODO: Should partition/community be computed before or after network is filtered?

Note: Steps 1 to 3 above (the most time consuming) are executed whenever an option marked with an asterix is changed.

In [14]:
```
# Visualization
if 'zy_data' not in globals():
    zy_data = types.SimpleNamespace(
        basename=None,
        network=None,
        X_n_space=None,
        X_pca_norm=None,
        X_tsne_norm=None,
        distance_matrix=None,
        metric=None,
        reducer=None,
        topic_proportions=None,
        n_words = 0
    )

def plot_clustering_dendogram(clustering):
    plt.figure(figsize=(16,6))
```

```
    # https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hiera
    R = dendrogram(clustering)
    plt.show()
    plt.close()

def display_correlation_network(
    layout_algorithm,
    threshold=0.10,
    scale=1.0,
    metric='Euclidean',
    reducer='tsne',
    n_words=200,
    output_format='Network'
):
    global state, zy_data, zy

    try:

        zy.progress.value = 1
        metric = DISTANCE_METRICS[metric]
        n_components = 3 if reducer == 'tsne' else 20
        perplexity=30

        node_description = state.get_topics_tokens_as_text()
        node_proportions = state.get_topic_proportions()

        zy.progress.value = 2
        if zy_data.network is None or state.basename != zy_data.basename or \
                zy_data.metric != metric or zy_data.reducer != reducer or zy_data.n_w

            zy_data.basename = state.basename
            zy_data.n_words = n_words
            zy_data.metric, zy_data.reducer = metric, reducer
            zy_data.pca_norm = None
            zy_data.X_n_space, _ = ModelUtility.compute_topic_terms_vector_space(state
            zy.progress.value = 3
            zy_data.X_m_space = VectorSpaceHelper.\
                reduce_dimensions(
                    zy_data.X_n_space, method=reducer, n_components=n_components, per
                )

            zy.progress.value = 5
            zy_data.distance_matrix = VectorSpaceHelper.compute_distance_matrix(zy_da
            zy.progress.value = 7

            zy_data.network = NetworkUtility.create_network_from_correlation_matrix(z

        zy.progress.value = 8
```

```
            if output_format == 'List':
                x = NetworkUtility.matrix_weight_iterator(matrix, threshold)
            else:
                p = PlotNetworkUtility.plot_network(
                    network=zy_data.network,
                    layout_algorithm=layout_algorithm,
                    scale=scale, threshold=threshold,
                    node_description=node_description,
                    node_proportions=node_proportions,
                    element_id='nx_id3'
                )

            zy.progress.value = 10
            show(p)
            zy.progress.value = 0
        except Exception as ex:
            logger.exception(ex)
            print('Error: {}'.format(ex))

    zy = BaseWidgetUtility(
        n_topics=state.n_topics,
        text_id='nx_id3',
        text=wf.create_text_widget('nx_id3'),
        scale=wf.create_float_slider('Scale', min=0.0, max=1.0, step=0.01, value=0.1),
        year=wf.create_int_slider(
            description='Year', min=state.min_year, max=state.max_year, step=1, value=sta
        ),
        reducer=wf.create_select_widget(
            label='Reducer*', values=['passthrough','pca','pca_norm','tsne'], default='pc
        ),
        n_words=wf.create_int_slider(description='#words*', min=10, max=500, step=1, valu
        metric=wf.create_select_widget(label='Metric*', values=list(DISTANCE_METRICS.keys
        threshold=wf.create_float_slider('Threshold', min=0.0, max=1.0, step=0.01, value=
        output_format=wf.create_select_widget('Format', ['Network', 'List', 'Pivot'], def
        layout=wf.create_select_widget('Layout', list(layout_algorithms.keys()), default=
        progress=wf.create_int_progress_widget(min=0, max=10, step=2, value=0, layout=wid
    )

    wy = widgets.interactive(
        display_correlation_network,
        layout_algorithm=zy.layout,
        threshold=zy.threshold,
        scale=zy.scale,
        metric=zy.metric,
        reducer=zy.reducer,
        n_words=zy.n_words,
        output_format=zy.output_format
    )
```

```
        display(widgets.VBox(
            (zy.text, ) +
            (widgets.HBox((zy.threshold,) + (zy.reducer,) + (zy.metric,) + (zy.output_format,)
            (widgets.HBox((zy.n_words,) + (zy.layout,) + (zy.scale,)),) +
            (zy.progress,) +
            (wy.children[-1],)))

        wy.update()
```

```
VBox(children=(HTML(value="<span class='nx_id3'></span>", placeholder=''), HBox(children=(Float
```

0.0.17 Visualization of Topic Similarity using 2D T-SNE Dimensionality Reduction

FIXME: var title = circle.data.words[index]; ska vara var title = circle.data.words[topic_id];???

In [15]:
```
        #
        import types
        tr_data = types.SimpleNamespace(
            X_n_space=None,
            X_m_space=None,
            n_words=None,
            method=None,
            perplexity=None,
            corpus_documents=state.get_corpus_documents(),
            topic_proportions=state.get_topic_proportions(),
            tokens=state.get_topics_tokens_as_text(n_words=200)
        )
```

In [16]:
```
        # Plot 2d utility function
        def plot_2d_vector_space(
            X_2_space, proportions=None, size=(20, 60),
            description=None, dom_id='id99', glyph_style=None, label_style=None
        ):
            global tr_data
            xs, ys = zip(*X_2_space)
            n_dim = len(xs)
            item_ids = description.index if not description is None else range(0, n_dim)

            if proportions is not None:
                proportions = PlotNetworkUtility.project_series_to_range(proportions, size[0]

            source = ColumnDataSource(
                dict(xs=list(xs),
                     ys=list(ys),
                     size=proportions if not proportions is None else [size[0]] * n_dim,
                     text=description if not description is None else item_ids,
```

```
                item_id=item_ids
            )
        )
        p = figure(plot_width=800, plot_height=800, title='', tools=TOOLS)

        glyph_style = extend(dict(color='green', alpha=0.2, hover_color='red') , glyph_sty
        cr = p.circle(x='xs', y='ys', size='size', source=source, **glyph_style)

        label_style = extend(dict(level='overlay', text_align='center', text_baseline='mi
                                  text_font_size='8pt') , label_style or {})
        labels = bm.LabelSet(x='xs', y='ys', text='item_id', source=source, **label_style)

        p.add_layout(labels)

        p.add_tools(bm.HoverTool(renderers=[cr], tooltips=None, callback=WidgetUtility.\
            glyph_hover_callback(source, 'item_id', text_ids=description.index, text=desc
        )

        return p
```

In [17]:
```
#
def reduce_and_plot_vector_space(n_words, method='tsne', perplexity=30):
    global state, zc

    pp = zc.progress

    pp.value = 1

    if tr_data.X_n_space is None or tr_data.n_words != n_words:
        tr_data.X_n_space, _ = ModelUtility.compute_topic_terms_vector_space(state.get
        tr_data.X_m_space = None

    pp.value = 2

    if  tr_data.X_m_space is None or tr_data.method != method or tr_data.perplexity !=
        tr_data.X_m_space = VectorSpaceHelper.reduce_dimensions(
            tr_data.X_n_space, method=method, n_components=2, perplexity=perplexity
        )

    tr_data.n_words = n_words
    tr_data.method = method
    tr_data.perplexity = perplexity

    pp.value = 4

    p = plot_2d_vector_space(
        tr_data.X_m_space, proportions=tr_data.topic_proportions, size=(20,40),
        description=tr_data.tokens, dom_id='text99'
```

```
            )
            pp.value = 5
            show(p)
            pp.value = 0

        zc = BaseWidgetUtility(
            n_words=wf.create_int_slider(description='Word count', min=10, max=500, step=10, v
            progress=wf.create_int_progress_widget(min=0, max=5, step=1, value=0, layout=widge
            perplexity=wf.create_int_slider(description='Perplexity', min=1, max=100, step=1,
            reducer=wf.create_select_widget(
                label='Reducer*', values=['pca','pca_norm','tsne'], default='tsne'
            ),
            text=wf.create_text_widget(element_id='text99')
        )
        wc = widgets.interactive(
            reduce_and_plot_vector_space, n_words=zc.n_words, method=zc.reducer, perplexity=zc
        )

        display(widgets.VBox(
            (zc.text, ) +
            (widgets.HBox((zc.n_words,) + (zc.reducer, ) + (zc.perplexity,)),) +
            (zc.progress, ) +
            (wc.children[-1],)))

        wc.update()
```

```
VBox(children=(HTML(value="<span class='text99'></span>", placeholder=''), HBox(children=(IntSl
```

0.0.18 Document Similarity using LDA topic weights

A similarity metric between documents is computed using a distance metric between the of document-topic vectors obtained from applying the trained LDA model. The vector space consists of n coordinates (i.e. 1036 segmented Daedalus articles) in m dimensions where m equals the number of topics. If all coordinates were to be used....

Problems:

- It is desirable to exclude uninteresting topics from the computation and/or the plot. Documents with a close to even distriibution of topic weights are (clear?) candidates for exclusion.
- Is there an established method of identifying the most (topically) interesting documents?
- Use a goodness of fit to test against uniform discrete density distribution? Wasserstein distance? Chi-square? KS-test

First attempt using T-SNE

- It is hard to ses clusters.

In [18]:
```
def plot_similarity_distribution():
    df = state.get_document_topic_weights()
    X_m_n_sparse = compute_document_topic_vector_space(df)
    matrix = VectorSpaceHelper.compute_distance_matrix(X_m_n_sparse, metric='cosine')
    x_dim, y_dim = matrix.shape
    items = ((i, j, matrix[i,j]) for i, j in product(range(0,x_dim), range(0,y_dim)) i
    ns, nm, ws = list(zip(*items))
    df = pd.DataFrame(dict(n=ns,m=nm,w=ws))
    df['similarity'] = (df.w*1000).astype('int')
    p = df.groupby('similarity').size().iloc[0:970].plot()
```

In [19]:
```
#https://stackoverflow.com/questions/22433884/python-gensim-how-to-calculate-document

def compute_document_topic_vector_space(df):

    #''' Filter out topics below given threshold '''
    #df = df[df.weight][['document_id', 'topic_id', 'weight']]

    ''' Create a dict (pair) for each topic-weight row '''
    df['weight_dict'] = df.apply(lambda x: { int(x.topic_id): x.weight}, axis=1)

    ''' Create a list of all dicts for each documents'''
    df = df.groupby('document_id')['weight_dict'].apply(list)

    ''' Merge the list of pair dicts into a single dict '''
    df = df.apply(lambda L: { k: v for d in L for k, v in d.items() } )

    ''' Fit the topic weighs into a sparse matrix (dimensions m_documents X n_topics)
    v = DictVectorizer()
    X_m_n_sparse = v.fit_transform(df)

    return X_m_n_sparse
```

In [20]:
```
# T-SNE 2D Visualization
if 'ds_data' not in globals():
    ds_data = types.SimpleNamespace(
        X_m_n_sparse=None,
        X_2_space=None,
        threshold=None,
        reducer=None,
        perplexity=None,
        G=None,
        description=state.get_corpus_documents().rename(columns={'document': 'text'})
    )

def plot_document_similarity_by_topics_tsne(threshold=0.001, reducer='tsne', perplexi
    global u, ds_data
```

```
    df = state.get_document_topic_weights()

    u.progress.value = 1
    if ds_data.X_m_n_sparse is None:
        ds_data.X_m_n_sparse = compute_document_topic_vector_space(df)
        ds_data.threshold = threshold
        ds_data.X_2_space = None

    u.progress.value = 2
    if ds_data.X_2_space is None or ds_data.reducer != reducer\
            or ds_data.perplexity != perplexity or ds_data.threshold != threshold:
        ds_data.X_2_space = VectorSpaceHelper.reduce_dimensions(
            ds_data.X_m_n_sparse, method=reducer,
            n_components=2, perplexity=perplexity)
        ds_data.reduce = reducer
        ds_data.perplexity = perplexity

    u.progress.value = 3

    description = state.get_corpus_documents().rename(columns={'document': 'text'})['t

    u.progress.value = 4
    p = plot_2d_vector_space(ds_data.X_2_space, proportions=None,
            size=(20,60), description=ds_data.description, dom_id='nx_id4', glyph_styl

    u.progress.value = 5
    show(p)
    u.progress.value = 0

u = BaseWidgetUtility()
u.threshold = u.create_float_slider('Threshold', min=0.0, max=0.10, step=0.01, value=0
u.reducer = u.create_select_widget(label='Reducer*', values=['pca', 'pca_norm', 'tsne
u.progress = u.create_int_progress_widget(min=0, max=5, step=1)
u.perplexity = u.create_int_slider(description='Perplexity', min=1, max=100, step=1, v
u.text = u.create_text_widget(element_id='nx_id4')

w = widgets.interactive(plot_document_similarity_by_topics_tsne,
                threshold=u.threshold,
                reducer=u.reducer,
                perplexity=u.perplexity)

display(widgets.VBox(
    (u.text, ) +
    (widgets.HBox((u.threshold,) + (u.reducer,) + (u.perplexity,) + (u.progress,)),) +
    (w.children[-1],)))

w.update()
```

```
VBox(children=(HTML(value="<span class='nx_id4'></span>", placeholder=''), HBox(children=(Float
```

0.0.19 The Same Data Visualized as a Network

```
In [ ]: # Code
        if True or 'zu_data' not in globals():
            zu_data = types.SimpleNamespace(
                X_m_n_sparse=None,
                top=None,
                metric=None,
                reducer=None,
                document_topic_weights=state.get_document_topic_weights(),
                corpus_documents=state.get_corpus_documents(),
                topic_proportions=state.get_topic_proportions(),
                G=None
            )

        def plot_document_similarity_by_topics_network(
            layout_algorithm, top, metric, reducer
        ):
            global zu
            scale = 1.0
            threshold = 0.0
            zu.progress.value = 1
            df = zu_data.document_topic_weights

            zu.progress.value = 2
            if zu_data.X_m_n_sparse is None:
                zu_data.X_m_n_sparse = compute_document_topic_vector_space(df)
                zu_data.metric = None
                zu.progress.value = 3
            metric = DISTANCE_METRICS[metric]
            if zu_data.metric != metric or zu_data.top != top:
                zu_data.top = top
                zu_data.metric = metric
                matrix = VectorSpaceHelper.compute_distance_matrix(zu_data.X_m_n_sparse, metri
                #edges = NetworkUtility.matrix_weight_iterator(matrix, threshold)
                edges = NetworkUtility.df_stack_correlation_matrix(matrix, threshold=0.0, n_to
                zu.progress.value = 4
                G = nx.Graph()
                G.add_weighted_edges_from(edges)
                zu_data.G = G
                print(nx.info(zu_data.G))

            node_ids, degrees = list(zip(*list(zu_data.G.degree(zu_data.G.nodes()))))
            node_proportions = pd.DataFrame(dict(node_id=node_ids, size=degrees)).set_index('n
            node_proportions['size'] *= 1000
```

```
        zu.progress.value = 5
        p = PlotNetworkUtility.plot_network(
            network=zu_data.G,
            layout_algorithm=layout_algorithm,
            scale=scale,
            threshold=0.0,
            node_description=state.get_corpus_documents(),
            node_proportions=node_proportions,
            weight_scale=1.0,
            normalize_weights=True,
            element_id='nx_id_5'
        )
        zu.progress.value = 6
        show(p)
        zu.progress.value = 0

    zu = BaseWidgetUtility(
        text = wf.create_text_widget(element_id='nx_id_5'),
        #scale = wf.create_float_slider('Scale', min=0.0, max=1.0, step=0.1, value=1.0),
        reducer = wf.create_select_widget(label='Reducer*', values=['none','pca','pca_norm
        progress = wf.create_int_progress_widget(min=0, max=6, step=1, value=0, layout=widg
        threshold = wf.create_float_slider('Threshold', min=0.01, max=1.0, step=0.01, value
        top = wf.create_int_slider('Top', min=100, max=1000, step=100, value=100),
        metric = wf.create_select_widget(label='Metric*', values=list(DISTANCE_METRICS.keys
        layout_algorithm = wf.layout_algorithm_widget(list(layout_algorithms.keys()), defau
    )

    wu = widgets.interactive(plot_document_similarity_by_topics_network,
                    layout_algorithm=zu.layout_algorithm,
                    #threshold=u.threshold,
                    top=zu.top,
                    metric=zu.metric,
                    reducer=zu.reducer)

    display(widgets.VBox(
        (zu.text, ) +
        #(zu.threshold,) +
        (widgets.HBox((zu.reducer,) + (zu.metric,)),) +
        (widgets.HBox((zu.layout_algorithm,) + (zu.top,)),) +
        (zu.progress, ) +
        (wu.children[-1],)))

    wu.update()
```

0.0.20 pyLDAvis

This visualization uses pyLDAvis which is http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf

In [22]:
```
# Code
# from IPython.display import IFrame, display
# IFrame('./data/{}/pyldavis.html'.format(state.basename), width=900, height=900)
%run ./common/model_utility
import pyLDAvis.gensim as gensimvis
import pyLDAvis

lda = state.get_lda()
dictionary = ModelUtility.load_dictionary(state.data_folder, state.basename)
corpus = ModelUtility.load_corpus(state.data_folder, state.basename)

pyLDAvis.enable_notebook()
vis_data = gensimvis.prepare(lda, corpus, dictionary)
pyLDAvis.display(vis_data)
```

```
/usr/local/lib/python3.5/dist-packages/pyLDAvis/_prepare.py:387: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  topic_term_dists = topic_term_dists.ix[topic_order]
```

Out[22]: <IPython.core.display.HTML object>

0.0.21 Compute and Plot Document Similarity using TF-IDF and T-SNE

FIXME Fill in real TF-IDF values (from model) for tokens not in top-list (instead of zero)
FIXME Simple (to simlple) document similarity metric, use text2vec instead!

In [23]:
```
# Code
from gensim.models.tfidfmodel import TfidfModel
from gensim import corpora
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

class TfidfReducer:

    def __init__(self):
        self.corpus = corpora.MmCorpus(os.path.join(state.data_folder, state.basename
        self.dictionary = corpora.Dictionary.load(os.path.join(state.data_folder, sta
        self.data_folder = data_folder
        self.basename = basename

    def tfidf_vectors(self, tfidf, corpus, n_tokens):
        for document in corpus:
            yield tfidf[document][:n_tokens]
```

```
    def tfidf_vectors_as_dicts(self, tfidf, corpus, n_tokens):
        ''' Create a dict(token_1: weight, ..., token_n: weight } for each document '
        for tfidf_vector in self.tfidf_vectors(tfidf, corpus, n_tokens):
            yield { x[0]: x[1] for x in tfidf_vector }

    def fit_transform(self, tfidf, corpus, n_tokens, perplexity=30):

        ''' Align vectors... '''
        v = DictVectorizer()
        dict_vectors = self.tfidf_vectors_as_dicts(tfidf, corpus, n_tokens)
        X = v.fit_transform(dict_vectors)
        feature_names = v.get_feature_names()

        print('Shape: ', X.shape)
        reducer = TSNE(n_components=2, init='pca', random_state=2019, perplexity=perpl
        X_reduced = reducer.fit_transform(X.toarray())

        return X, feature_names, X_reduced

class TfidfDocumentWidgets():

    def __init__(self, years):
        self.text_id = 'document_text'
        self.text = widgets.HTML(value="<span class='{}'/>".format(self.text_id), pla
        self.perplexity = widgets.IntSlider(
            min=1, max=200, step=1, value=30, description='Perplexity', continuous_up
        )
        self.word_count = widgets.IntSlider(
            min=50, max=250, step=1, value=200, description='Word count', continuous_
        )
        #self.dropdown = widgets.Dropdown(options=[], value='None', description='Drop
        self.year = widgets.Dropdown(
            options=state.years, value=state.years[0], description='Year', disabled=Fa
        )

    def setup_hover_callback_tool(self, cr):
        code = """
        var indices = cb_data.index['1d'].indices;
        if (indices.length > 0) {
            var index = indices[0];
            var topic_id = circle.data.topic_id[index];
            var title = circle.data.words[index];
            //var share = (100.0 * circle.data.topic_proportion[index]).toFixed(1).toS
            $('."""" + self.text_id + """').html('DOC ' + topic_id.toString() + ': ' +
        }
        """
        callback = CustomJS(args={'document_glyph': cr.data_source}, code=code)
```

```
        p.add_tools(HoverTool(tooltips=None, callback=callback, renderers=[cr]))
        return HoverTool(tooltips=None, callback=callback, renderers=[cr])

def plot_tf_idf_document_vector_space(X_reduced, document_index):

    xs, ys = zip(*X_reduced)
    source = ColumnDataSource(
        dict(xs=list(xs),
             ys=list(ys),
             #size=5,
             #words=titles,
             #topic_id=titles.index
        )
    )
    p = figure(plot_width=800, plot_height=800, title='', tools=TOOLS)
    cr = p.circle(x='xs', y='ys', size=5, source=source, alpha=0.2, hover_color='red')
    show(p)

if 'corpus' is not in globals():
    corpus = corpora.MmCorpus(os.path.join(state.data_folder, state.basename, 'corpus
    dictionary = corpora.Dictionary.load(os.path.join(state.data_folder, state.basenam
    id2document = ModelUtility.get_corpus_documents(data_folder, basename)
    tfidf_corpus = TfidfCorpus(state.data_folder, state.basename, tfidf, corpus, n_to
    tfidf = TfidfModel(corpus)

if 'X_reduced' is not in globals():
    ''' This takes some time to compute...'''
    document_tfidf_vectors = tfidf_vectors_as_dicts(tfidf, corpus)
    X, feature_names, X_reduced = compute_document_pca(document_tfidf_vectors)

def display_tf_idf_document_vector_space(perplexity, word_count, year):
    global X_reduced
    plot_tf_idf_document_vector_space(X_reduced, perplexity)

u = TfidfDocumentWidgets(state.years)
w = interactive(display_tf_idf_document_vector_space,
                perplexity=u.perplexity, word_count=u.word_count, year=u.year)

display(widgets.VBox(
    (u.text,) + (widgets.HBox((u.year,) + (u.perplexity,) + (u.word_count,)),)
    + (w.children[-1],)))

# w.update()


  File "<ipython-input-23-57f36eb75466>", line 84
if 'corpus' is not in globals():
                  ^
```

```
SyntaxError: invalid syntax
```

0.0.22 TODO: Document similarity using BOW document vectorization:

https://de.dariah.eu/tatom/working_with_text.html

Goodness of Fit using **Kolmogorov-Smirnov** (alternatives are **chi square** and **maximum likelihood**)

https://stats.stackexchange.com/questions/113464/understanding-scipy-kolmogorov-smirnov-test *"For the KS test the p-value is itself distributed uniformly in [0,1] if the H0 is true (which it is if you test whether it your sample is from U(0,1)U(0,1) and the random number generation works okay). It therefore must "vary wildly" between 0 and 1, in fact its standard deviation is 1/121/12 which is roughly 0.3."*

https://en.m.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test *"The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples. The null distribution of this statistic is calculated under the null hypothesis that the sample is drawn from the reference distribution (in the one-sample case) or that the samples are drawn from the same distribution (in the two-sample case). In each case, the distributions considered under the null hypothesis are continuous distributions but are otherwise unrestricted....The Kolmogorov–Smirnov test can be modified to serve as a goodness of fit test. "*

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html

scipy.stats.wasserstein_distance

0.0.23 TODO Add use of HDP model (Hierarchical Dirichlet Process)

Hierarchical Dirichlet process

Teh, Y. W.; Jordan, M. I.; Beal, M. J.; Blei, D. M. (2006). "Hierarchical Dirichlet Processes" (PDF). Journal of the American Statistical Association. 101: pp. 1566–1581.

hdp = models.hdpmodel.HdpModel(corpus, dictionary, T=50)

hdp.save('basename.model')

HDP is an extension of LDA. HDP is non-parametric method, it will fit as many topics as it can find.

0.0.24 Citing

To cite NetworkX please use the following publication:

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

@inproceedings{rehurek_lrec, title = {{Software Framework for Topic Modelling with Large Corpora}}, author = {Radim {Ř}eh{ů}{ř}ek and Petr Sojka}, booktitle = {{Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks}}, pages = {45--50}, year = 2010, month = May, day = 22, publisher = {ELRA}, address = {Valletta, Malta}, note={http://is.muni.cz/publication/884893/en}, language={English} }