Information Visualization II

School of Information, University of Michigan

Week 4:

- Text visualizations

Assignment Overview

The objectives for this week are for you to:

- Understand how to model a corpus using statistical and visual techniques
- Construct an interactive information visualization for search tasks

The total score of this assignment will be

- Problem 1 (20 points)
- Problem 2 (80 points)

Resources:

- We have created two textual datasets for you. One contains the text from Wikipedia pages related to data mining (algorithms, software, techniques, people, etc.). The second is related to *real* mining (equipment, companies, locations, etc.).

Important notes:

1) Grading for this assignment is entirely done by manual inspection. You will have lots of control over the look and feel of problem 2.

2) When turning in your PDF, please use the File -> Print -> Save as PDF option ***from your browser***. Do ***not*** use the File->Download as->PDF option. Complete instructions for this are under Resources in the Coursera page for this class.

If you're having trouble with printing, take a look at this video (https://youtu.be/PiO-K7AoWjk).

In [1]:
```
import pandas as pd
import altair as alt
import json
import ipywidgets as widgets
import spacy
import math
import numpy as np
import scattertext as st
from sklearn import manifold
from sklearn.metrics import euclidean_distances
sp = spacy.load('en_core_web_sm')
```

In [2]:
```
# enable correct rendering (unnecessary in later versions of Altair)
alt.renderers.enable('default')

# uses intermediate json files to speed things up
alt.data_transformers.enable('json')
```

Out[2]:
```
DataTransformerRegistry.enable('json')
```

In [3]:
```
# some utility classes that will help us load the data in
def lemmatize(instring,title="",lemmaCache = {}):
    parsed = None

    if ((title != "") & (title in lemmaCache)):
        parsed = lemmaCache[title]
    else:
        parsed = sp(instring)

    if (lemmaCache != None):
        lemmaCache[title] = parsed
    sent = [x.text if (x.lemma_ == "-PRON-") else x.lemma_ for x in parsed]
    return(sent)

def generateData(filepath,lemmaCache=None):
    articles = []
    with open(filepath) as fp:
        for docid, line in enumerate(fp):
            doc = json.loads(line)
            doclines = doc['text'].split("\n\n")
            ilineid = -1  # we're going to replace the original lineids so there are no gaps
            for lineid,docline in enumerate(doclines):
                obj = {}
                obj['docid'] = docid;
                obj['title'] = doc['title']
                paraterms = lemmatize(docline,doc['title']+str(lineid),lemmaCache)
                obj['text'] = ' ' + ' '.join(paraterms) + ' '
                obj['tokencount'] = len(paraterms)
                if ('category' in doc):
                    obj['category'] = doc['category']
                if (len(paraterms) > 10):
                    ilineid = ilineid + 1
                    obj['lineid'] = ilineid
                    articles.append(obj)
    return pd.DataFrame(articles)

def loadFile(classname,classpath,maxc=200,lemmaCache={}):
    articles = []
    with open(classpath) as fp:
        for docid, line in enumerate(fp):
            doc = json.loads(line)
            doclines = doc['text'].split("\n\n")
```

```
            obj = {}
            obj['docid'] = docid;
            obj['title'] = doc['title']
            paraterms = lemmatize(doc['text'],doc['title'],lemmaCache)
            obj['text'] = ' ' + ' '.join(paraterms) + ' '
            obj['label'] = classname
            if ('category' in doc):
                obj['category'] = doc['category']
            if (len(paraterms) > 10):
                articles.append(obj)
            if (docid > maxc):
                break
    return(articles)

def loadClasses(class1name,class1path,class2name,class2path,maxc=300):
    articles = loadFile(class1name,class1path) + loadFile(class2name,class2path)
    return pd.DataFrame(articles)
```

In [4]:
```
# enable correct rendering
alt.renderers.enable('default')

# uses intermediate json files to speed things up
alt.data_transformers.enable('json')
```

Out[4]:
```
DataTransformerRegistry.enable('json')
```

Before we start...

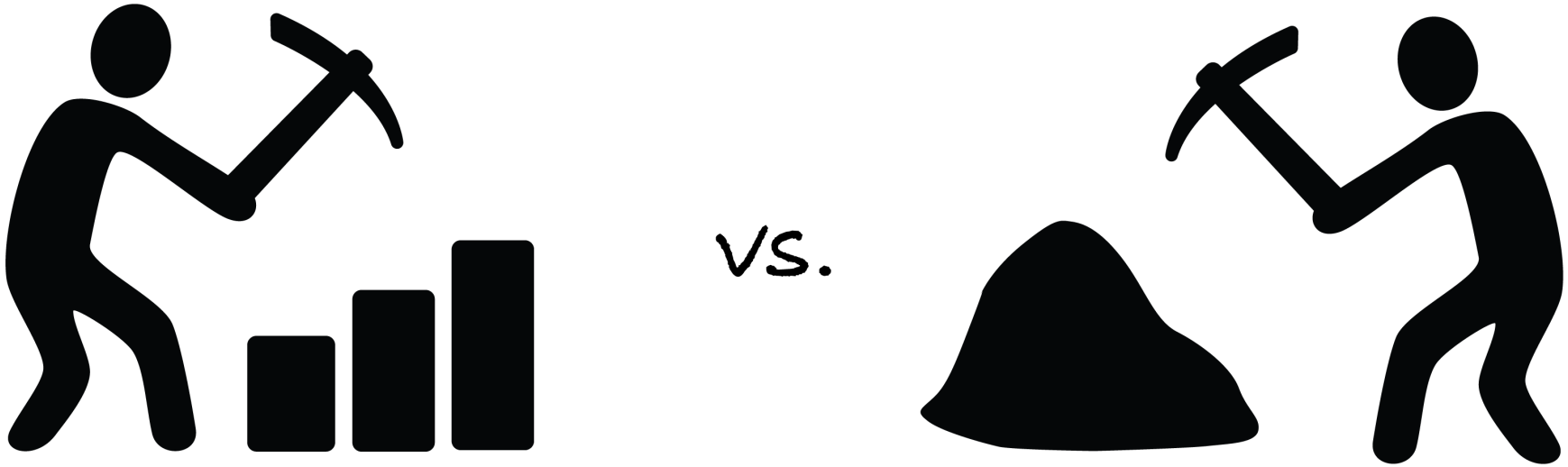
We have created a function for you called `lemmatize(...)`. It takes as input a string and assumes that spaces are token delimiters. For each token/word, the system will lowercase it, stem it (getting the root), and generally clean it up. The data we load from our files undergoes the same transformation. So it's important to lemmatize your terms if you are looking them up. For example, you won't find the word "data" in the DataFrame. All instances get transformed to "datum." Thus, it's important to remember to do this transformation. Note, however, that if you query for "Data Mining" (in capitals), the system assumes that this is a name, and will not change it.

In [5]:
```
# here's a few examples

query1 = "data mining"
print("The lemmatized version of '"+query1+"' is:", lemmatize(query1))

query2 = "Data Mining"  # proper name (like a business)
print("The lemmatized version of '"+query2+"' is:", lemmatize(query2))

query3 = "executing awesome algorithms"
print("The lemmatized version of '"+query3+"' is:", lemmatize(query3))

query4 = "Data mining algorithms are super exciting."
print("The lemmatized version of '"+query4+"' is:", lemmatize(query4))
```

```
The lemmatized version of 'data mining' is: ['datum', 'mining']
The lemmatized version of 'Data Mining' is: ['data', 'mining']
The lemmatized version of 'executing awesome algorithms' is: ['execute', 'awesome', 'algorithm']
The lemmatized version of 'Data mining algorithms are super exciting.' is: ['datum', 'mining', 'algori
thm', 'be', 'super', 'exciting', '.']
```

Problem 1 (20 points)

For this first problem, we will be comparing which terms most often appear in which of our two corpora: 'data' mining and 'real' mining.



Let's load the data in:

In [6]:
```
# this will load the two files and label them with one of the two class labels
# lemmatizing these files takes some time on Coursera so we've pre-calculated it for you.
# If you want to run this process, uncomment the next two lines of code
# miningdf = loadClasses('data mining','assets/mlarticles.jsonl','real mining','assets/miningarticles.js
# miningdf.to_csv("assets/miningvmining.csv",index=False)

# load from cached file
miningdf = pd.read_csv("assets/miningvmining.csv")
```

In [7]:
```
# let's look at what's inside.  We have a document id (docid), title (from Wikipedia)
# the text, the label (one of: 'real mining' or 'data mining'), and a category column
# which you can ignore for now (it has the Wikipedia category for just the data mining articles)

miningdf.sample(10)
```

Out[7]:

	docid	title	text	label	category
175	175	Instance-based learning	in machine learning , instance - base learnin...	data mining	Machine learning
133	133	Matthews correlation coefficient	the matthews correlation coefficient be use i...	data mining	Machine learning
82	82	Euler (software)	euler (now euler mathematical toolbox or eum...	data mining	Data analysis software
239	37	Pale Rider	pale rider be a 1985 american western film pr...	real mining	NaN
374	172	Lydenburg	lydenburg be a town in thaba chweu local muni...	real mining	NaN
223	21	Broken Hill	broken hill be an inland mining city in the f...	real mining	NaN
229	27	BHP	bhp , formerly know as bhp billiton , be the ...	real mining	NaN
140	140	Semantic analysis (machine learning)	in machine learning , semantic analysis of a ...	data mining	Machine learning
262	60	Kerr-McGee	the kerr - mcgee corporation , found in 1929 ...	real mining	NaN
142	142	Rademacher complexity	in computational learning theory (machine le...	data mining	Machine learning

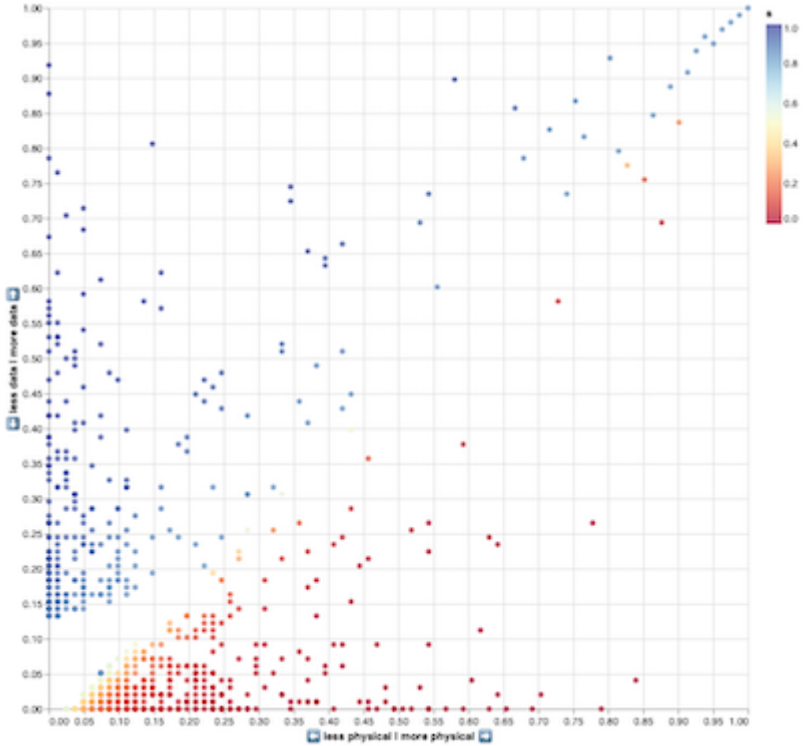
In [8]:
```
# note that all category labels for real mining are NaN
miningdf[miningdf['label']=='real mining']['category'].unique()

# note that the texts (all 404 rows) come from 202 unique documents
print(len(miningdf['docid']))
print(len(miningdf['docid'].unique()))
```

```
404
202
```

In [9]:
```
# you'll notice that the text is lemmatized. Here's the text for the first entry:
miningdf.head(1).text.values[0]
```

Out[9]:
```
' matlab ( matrix laboratory ) be a multi - paradigm numerical computing environment and proprietary p
rogramming language develop by mathworks . matlab allow matrix manipulation , plot of function and dat
um , implementation of algorithm , creation of user interface , and interfac with program write in oth
er language , include c , c++ , c # , java , fortran and python . \n\n although matlab be intend prima
rily for numerical computing , an optional toolbox use the mupad symbolic engine , allow access to sym
bolic computing ability . an additional package , simulink , add graphical multi - domain simulation a
nd model - base design for dynamic and embed system . \n\n as of 2018 , matlab have more than 3 millio
n user worldwide . matlab user come from various background of engineering , science , and economic .
'
```

We will be using the scattertext (https://github.com/JasonKessler/scattertext) library to create, analyze and position the terms. We encourage you to take a look at all the features of scattertext. It has a lot of "knobs" to control the analysis. It will also create a very fancy Web-based, interactive visualization for you if you want. For our initial experiment, we're going to start simple. We simply want to plots terms based on how common they are in 'data mining' and in 'real mining.' The lower-left corner will hold uncommon terms for both. The upper right will be terms that often appear in both domains. A way to think of this is that terms on the diagonal (slope 1) appear equally in both domains. The other two corners are the outliers--these are terms that are either more common for data or real mining. Here's a screenshot of what we'll get:



Click here (assets/scattertext.png) for a larger image.

We're going to run the analysis for you and have you generate the visualization. Once you get the first version working, you can play with the options to see how they impact the analysis/visualization. In particular, you might want to change the term frequency and PMI (pointwise mutual information) thresholds to see what they do.

In [10]:
```
# apply the scattertext analysis pipeline to the text, this will create a new column called parse
miningdf = miningdf.assign(
    parse=lambda df: df.text.apply(st.whitespace_nlp_with_sentences)
)

# create a "corpus" object
corpus = st.CorpusFromParsedDocuments(
    # use the miningdf as input. The category col is "label" and the parsed data is in "parse"
    miningdf, category_col='label', parsed_col='parse'
    # the unigram corpus means we want single words (there's another version that throws out stopwords)
    # the association compactor says we want the 2000 most label-associated terms
).build().get_unigram_corpus().compact(st.AssociationCompactor(2000))

# next, we build the actual visualization
scatterdata = st.produce_scattertext_explorer(
    corpus,                             # the corpus
    category='data mining',             # the "base" category
    category_name='data mining',        # the label for the category (same in this case)
    not_category_name='real mining',    # the label of the other category
    minimum_term_frequency=0,           # threshold frequency
    pmi_threshold_coefficient=0,        # the PMI threshold
    return_data=True,                   # this tells scattertext to return the data rather than saving a
    transform=st.Scalers.dense_rank     # where to place identically ranked terms (on top of each other
)
```

In [11]:
```
corpus.get_df()
```

Out[11]:

	index	docid	title	text	label	category	parse
0	0	0	MATLAB	matlab (matrix laboratory) be a multi - par...	data mining	Data mining and machine learning software	(matlab, (, matrix, laboratory,), be, a, mult...
1	1	1	Ray Kurzweil	raymond kurzweil (; bear february 12 , 1948 ...	data mining	Machine learning researchers	(raymond, kurzweil, (, ;, bear, february, 12, ...
2	2	2	Genetic algorithm	in computer science and operation research , ...	data mining	Machine learning	(in, computer, science, and, operation, resear...
3	3	3	Data mining	datum mining be the process of discover pat...	data mining	Data mining	(datum, mining, be, the, process, of, discover...
4	4	4	GNU Octave	gnu octave be software feature a high - level...	data mining	Data mining and machine learning software	(gnu, octave, be, software, feature, a, high, ...
...	...	...	...	...	...	...	...
399	399	197	NLC Workers Progressive Union	thozhilalar munnetra sangam , padaithurai uda...	real mining	NaN	(thozhilalar, munnetra, sangam, ,, padaithurai...
400	400	198	Old Hundred Gold Mine	the old hundred gold mine be a gold mine in s...	real mining	NaN	(the, old, hundred, gold, mine, be, a, gold, m...
401	401	199	Lonmin	lonmin plc , formerly the mining division of ...	real mining	NaN	(lonmin, plc, ,, formerly, the, mining, divisi...
402	402	200	Stilfontein	stilfontein (afrikaans for quiet spring) be...	real mining	NaN	(stilfontein, (, afrikaans, for, quiet, spring...
403	403	201	Wolmaransstad	wolmaransstad (afrikaans for " wolmarans cit...	real mining	NaN	(wolmaransstad, (, afrikaans, for, ", wolmaran...

404 rows × 7 columns

At this point, `scatterdata` will contain all kinds of information. For example, `scatterdata['info']['category_terms']` will give you the terms most related to the "category" (remember, this is data mining). In contrast, you can get the "real mining" terms using not_category_terms.

In [12]:
```
print("terms most associated with data mining ",scatterdata['info']['category_terms'],"\n")
print("terms most associated with real mining ",scatterdata['info']['not_category_terms'])
```

```
terms most associated with data mining  ['datum', 'learning', 'algorithm', 'analysis', 'computer', 'da
ta', 'model', 'machine', 'research', 'pattern']

terms most associated with real mining  ['mine', 'town', 'gold', 'south', 'coal', 'locate', 'miner',
'diamond', 'river', 'north']
```

The more important piece for our visualization purposes is the "data" part of scatterdata. This is a list of "objects," one for each term. For example:

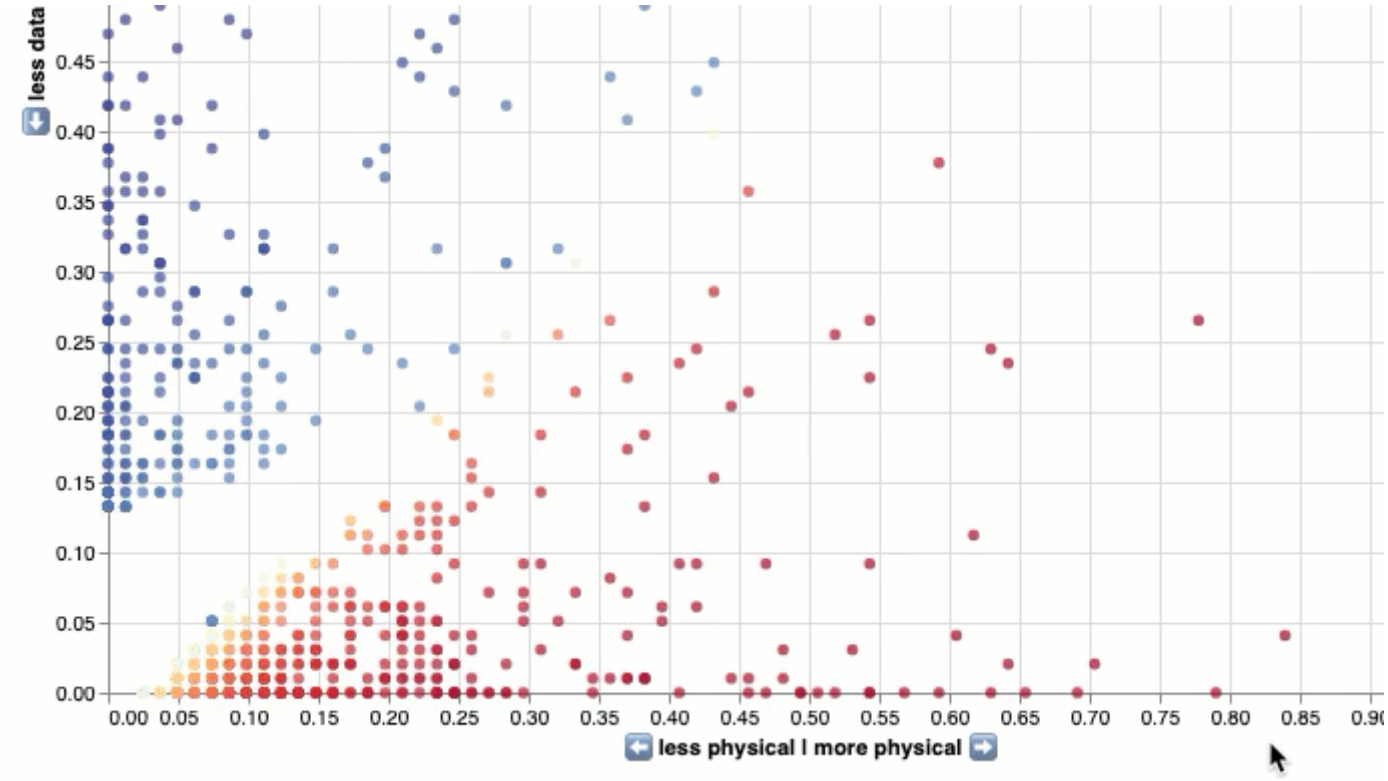
```
scatterdata['data'][0]
```

```
{'x': 0.0,
 'y': 0.13265306122448978,
 'ox': 0.0,
 'oy': 0.13265306122448978,
 'term': 'matlab',
 'cat25k': 15,
 'ncat25k': 0,
 'neut25k': 0,
 'neut': 0,
 'extra25k': 0,
 'extra': 0,
 'cat': 13,
 'ncat': 0,
 's': 0.8930722891566265,
 'os': 0.12921901946292189,
 'bg': 1.1352105640948616e-05}
```

This is the first item in the data list. There are a number of fields here. You can look at the documentation for scattertext for the details. The only items we care about right now will be `x`, `y`, `term`, and `s`. These respectively tell us the x/y coordinate for the term, the term itself, and the "distance" of the term from the central line (slope 1).

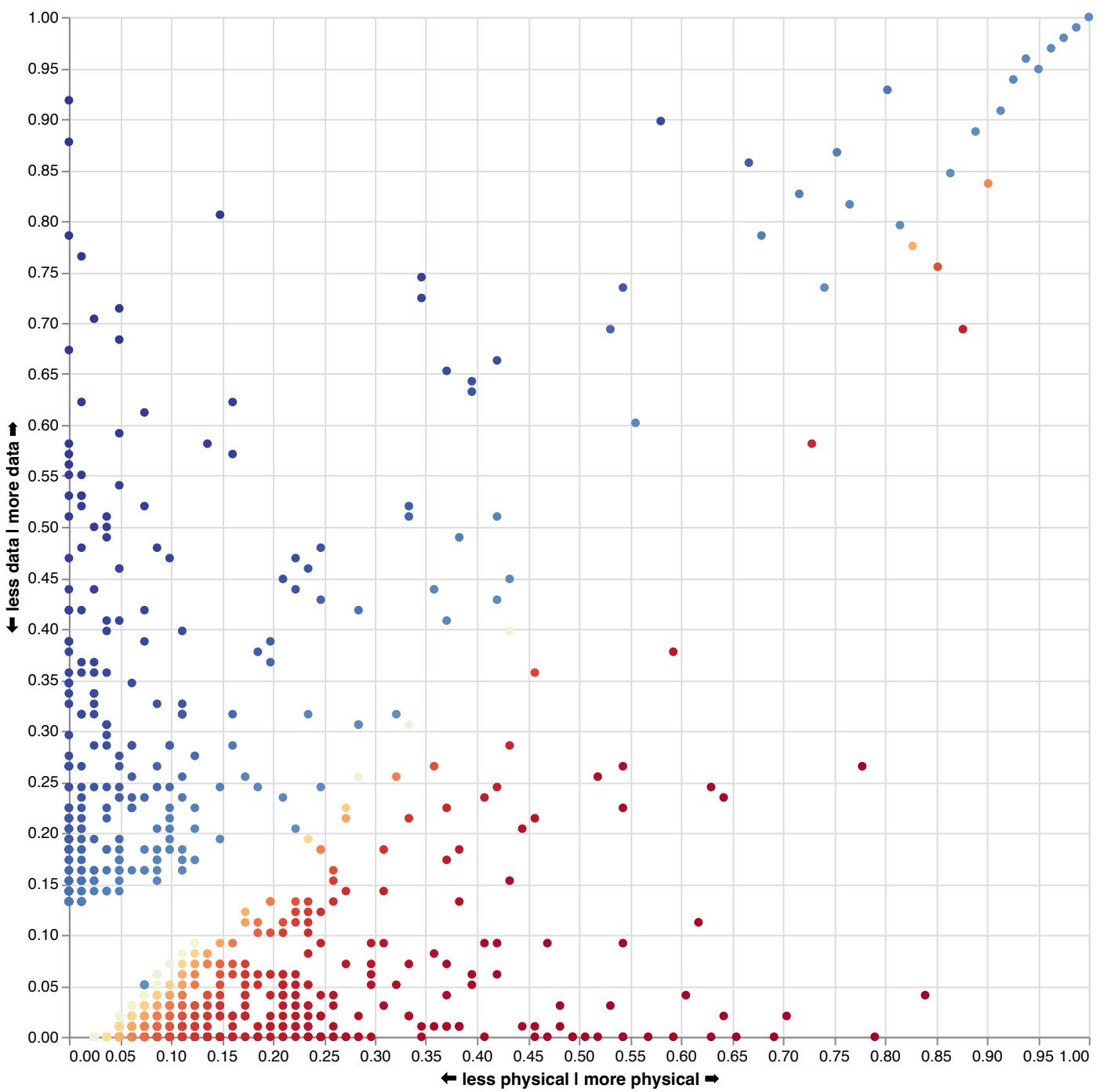
We would like for you to use this data to generate a visualization as in the example above. You're welcome to try to make it fancier, but consider this the minimum solution (notice the tooltips and colors).

Here's a *zoomed in* version just to show off the interactions:


less data
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85
less physical | more physical

In [13]:
```
def genScattertext():
    # this function should return an Altair chart as specified above

    coordData = pd.DataFrame.from_dict(scatterdata['data'])

    chart = alt.Chart(coordData).mark_circle(opacity=1).encode(
        x=alt.X(
            'x',
            scale=alt.Scale(
                domain=(0.0,1.0)
            ),
            axis=alt.Axis(tickCount=21.0),
            title=u'\u2B05' +' less physical | more physical ' + u'\u27A1'
        ),
        y=alt.Y(
            'y',
            scale=alt.Scale(
                domain=(0.0,1.0)
            ),
            axis=alt.Axis(tickCount=21.0),
            title=u'\u2B05' + ' less data | more data ' + u'\u27A1'
        ),
        color=alt.Color(
            's:Q',
            type='quantitative',
            scale=alt.Scale(
                scheme='redyellowblue',
                type='linear'
                            ),
            legend=None
        ),
        tooltip='term:N'
    ).properties(
        width=650,
        height=650
    ).display(renderer='svg')

    return chart
```

In [14]:
```
# if your code above works correctly, this should generate the plot
genScattertext()
```


1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
⬅ less data I more data ➡
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
⬅ less physical I more physical ➡

Problem 2

Problem 2.1 -- Implementation (70 points)

For this problem, we would like for you to build a visual query system using tilebars! You will need to build a function that returns an Altair visualization. It will take as input a query (text string), an option to normalize the data or not (A boolean True or False), and a string indicating the sort order ("name" or "score"). Here's a pretty bad implementation:

```
searchbutton = widgets.Button(description='Search')
normalizedradio = widgets.RadioButtons(description="Normalized?",options=['true', 'false'])
sortradio = widgets.RadioButtons(description="Sort by",options=['name', 'score'])

searchbutton.on_click(clicked)
normalizedradio.observe(clicked)
sortradio.observe(clicked)

list_widgets = [widgets.VBox([widgets.HBox([querybox,searchbutton]),
                              widgets.HBox([normalizedradio,sortradio])])]
accordion = widgets.Accordion(children=list_widgets)
accordion.set_title(0,"Search Controls")
display(accordion,output)
```


```
]:
```

Important: This example implementation isn't a great one. We expect you to use your skills to build something better. Simply replicating this example will not get you full credit.

You are welcome to come up with your own style, add interactivity, decide how to normalize and score, the data, etc. This problem is very open ended. If you don't remember how a tilebar is created, now is a good time to go back to the lecture and watch the video (or go to the source (https://people.ischool.berkeley.edu/~hearst/research/tilebars.html)).

Before we get started, let's load the corpus:

In [15]:
```
# we're only working with data mining here


# lemmatizing these files takes some time on Coursera so we've pre-calculated it for you.
# If you want to run this process, uncomment the next three lines of code
# We're going to "cache" lemmas to speed up some operations
# lemmaCache = {}
# dataminingdf = generateData('assets/mlarticles.jsonl',lemmaCache)
# dataminingdf.to_csv('assets/mlarticles.csv',index=False)

dataminingdf = pd.read_csv('assets/mlarticles.csv')
```

In [16]:
```
# let's look at the first few lines
dataminingdf.head(5)
```

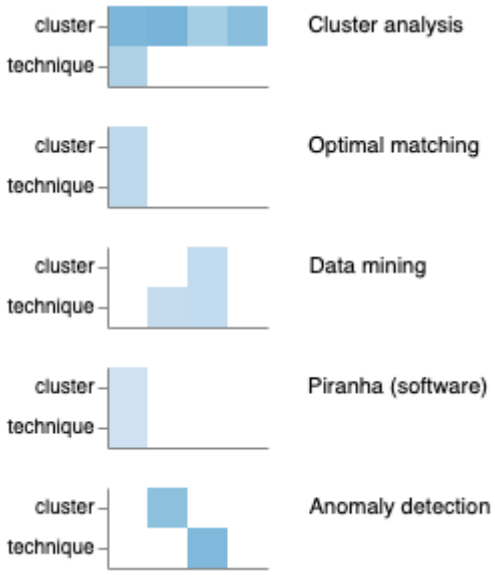
Out[16]:

	docid	title	text	tokencount	category	lineid
0	0	MATLAB	matlab (matrix laboratory) be a multi - par...	64	Data mining and machine learning software	0
1	0	MATLAB	although matlab be intend primarily for numer...	48	Data mining and machine learning software	1
2	0	MATLAB	as of 2018 , matlab have more than 3 million ...	27	Data mining and machine learning software	2
3	1	Ray Kurzweil	raymond kurzweil (; bear february 12 , 1948 ...	105	Machine learning researchers	0
4	1	Ray Kurzweil	kurzweil receive the 1999 national medal of t...	141	Machine learning researchers	1

What you see above is a row for every document and every "line." In pre-processing the top section of each Wikipedia article for you. We have taken each paragraph and made it into a new line. For example, the MATLAB (https://en.wikipedia.org/wiki/MATLAB) article has 3 lines. The frame has a document id (docid), title, line id (lineid -- based on the order the line appears), text (the text of the line), tokencount (the number of words in that line), and the category of the article (which you can use if you want).

At this point we're going to start implementing the `drawTilebars` function.

As we mentioned above, everything outside of the basic functions and tilebar encoding is fair game. You can decide how to rank documents (do you want to do it based on whether the matches are in the same line? whether there are many matches throughout the article?). You can also decide how you want to implement normalization on the tilebars themselves (by tokens in the line? by the maximum times the term appears in the document? the maximum time it appears in all documents?). Here's a static screenshot of our tilebars for "clustering techniques" normalized with score based ordering:



Some hints/ground rules (***read before you start***):

- ***Again, this particular version is NOT a good version, please don't simply replicate it.***
- We would like results to be *conjunctions* (i.e., we only want results to show those documents that match *all* search results: basically "ands). You can *optionally* implement disjunctions (i.e., "ors") and modify the interface to support that. You can extend the widget/function to support this if you like.
- Decide what comparisons need to be enabled by tilebars for them to work. Figure out how those can be *expressed* in the visualization and then made *effective*. There are some very simple things that will take our bad example and make it better. You should go back to the video/other materials. We will expect that *at least* satisfy the key tasks of the original tilebar implementation but you can add your own.
- We will consider your design choices in your grade. How it performs will matter, but so will how it looks.
- Make sure that sort by score does something reasonable. Think about what you would expect to be on top when you search for 'cluster analysis' (for example)
- Test, test, test - small queries, big queries, queries with no matches, etc
- Your solution should be performant. We should not be waiting more than a few seconds for the vis to show up.

- Take a look at some of the pandas features for text analysis. For example, for a row in our dataframe, you can get the count of the number of times a specific token appears by doing `row['text'].count(' ' + term + ' ')`
- You likely want to calculate two things--one is a dataframe describing your tilebar information, the other is some kind of document order. If you're clever, you can do it all at once. A less efficient solution might require two passes.
- Think about what you need to know in order to encode the "cell" of the tilebar. Your dataframe should contain that data.
- Consider the look and feel of your solution. We will be considering the aesthetic choices you are making.
- Think through how to build the "small multiples" here. You can use combinations of concatenation, faceting, and repeated charts. You'll likely need to play with a few solutions to get the look you're happy with.

In [43]:
```
def drawTilebars(query,normalized=False,sortby='title'):
    """
    This function takes:
        - query: a string query
        - normalized: an argument about whether to normalize the tilebar (True or False)
            - if false, the the color of the tile should map to the count
            - if true, you should decide how you want to normalize (by the max count overall? max count
        - sortby: a string of either "title" or "score"
            - if title, the tilebars should be returned based on alphabetical order of the articles
            - if score, you can decide how you want to rank the articles

    The function returns: an altair chart
    """
    master_df = pd.DataFrame()
    terms = lemmatize(query)
    drop_docs = []

    # loop through each document in the original DataFrame
    for doc in range(0,int(dataminingdf['docid'].max()+1)):

        # filter for the ith document
        doc_df = dataminingdf[dataminingdf['docid']==doc]

        # expand rows to represent each unique query term
        new_df = pd.DataFrame()
        for term in set(terms):
            doc_df['term']=term
            new_df = new_df.append(doc_df)
        new_df.reset_index(drop=True,inplace=True)

        # rename expanded DataFrame to avoid confusion
        doc_df = new_df
        new_df = None

        # iterate through each row (i.e., line x query term) and calculate term count
        for index,row in doc_df.iterrows():
            tfdf = row['text'].count(" " + row['term']+ " ")
            doc_df.at[index,'termcount']=tfdf

        # check for conjunction (i.e., whether each query term is in the document)
        for term in doc_df.term.unique():
            term_df = doc_df[doc_df.term==term]
```

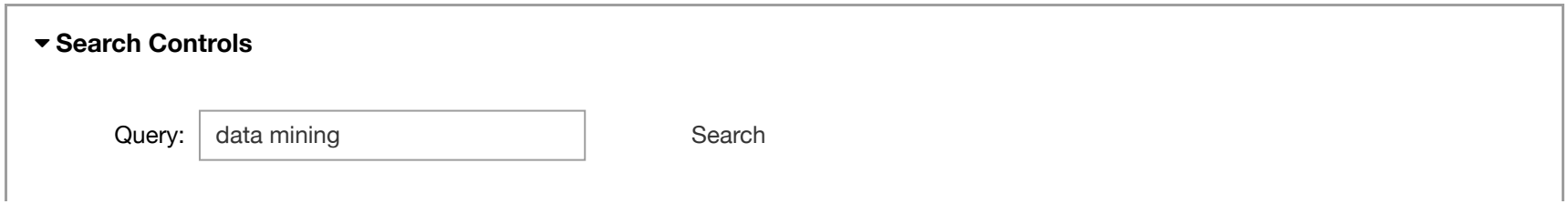
```
            if term_df.termcount.sum() == 0:
                [drop_docs.append(val) for val in term_df.docid.unique()]
                break

        master_df = master_df.append(doc_df).reset_index(drop=True)

    # drop rows where conjunction was not satisfied
    master_df = master_df[~master_df.docid.isin(drop_docs)]

    # if there are no matches, return text to inform user
    if len(master_df)==0:
        final_chart = alt.Chart(master_df).mark_text(text='No matches')
        return final_chart

    # if there is at least one conjunction match
    else:
        if normalized == True:

            # prep values for tfidf calculation
            text_df = dataminingdf.groupby('docid')['text'].apply(' '.join).reset_index()
            n_docs = len(dataminingdf['docid'].unique()) # number of unique documents

            # for each row in master_df calculate tfidf
            for index,row in master_df.iterrows():

                # count the number of documents in the entire set containing the term
                dft = len(text_data[text_data['text'].str.contains(row['term'])])

                try:
                    tfidf = np.log(1+row['termcount'])*np.log(n_docs/dft)
                except ZeroDivisionError: # TODO: I am not sure what to do here about tfidf when ZeroDiv
                    tfidf = 0

                master_df.at[index,'tfidf']=tfidf

        if sortby == 'title':
            master_df = master_df.sort_values(['title'],ascending=True)
        else:
            master_df = master_df.sort_values(['tfidf'],ascending=False)


    # create charts
```

```
charts = []
for doc in master_df['docid'].unique():

    doc_df = master_df[master_df['docid']==doc]
    doc_title = [val for val in doc_df.title.unique()][0]

    title_split = doc_title.split(' ')
    if len(title_split)>5:
        doc_title = [' '.join(title_split[0:5]),' '.join(title_split[5:])]

    # define tilebar for the current document
    tiles = alt.Chart(doc_df,title=alt.TitleParams(text=doc_title,anchor='start',align='left',fontSi
        alt.Y(
            'term:N',
            axis=alt.Axis(title=None)
        ),
        alt.X(
            'lineid:N',
            axis=None
        ),
        alt.Color(
            'termcount:Q',
        ),
        tooltip = ['lineid:N','termcount:Q','text:N']
    ).properties(
        width=200,
        height=100
    )
    charts.append(tiles)

# arrange charts in a matrix
rows = [charts[n:n+3] for n in range(0, len(charts), 3)]

rows_charts = []
for row in rows:
    curr_chart = None
    for chart in row:
        if curr_chart == None:
            curr_chart = chart
        else:
            curr_chart = alt.hconcat(curr_chart,chart,spacing=50)

    rows_charts.append(curr_chart)
```

```
    final_chart = None
    for row in rows_charts:
        if final_chart == None:
            final_chart = row
        else:
            final_chart = alt.vconcat(final_chart,row,spacing=50)

    return final_chart
```

If you built your solution correctly, you should be able to simply run the code below. Note that we don't use Altair interactivity because we don't know how you chose to implement your solution. The visualization will likely flicker as you recalculate it.

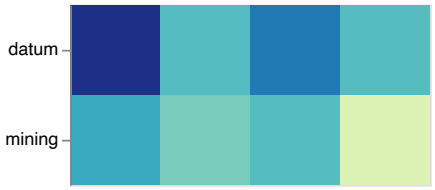
In [44]:
```
output = widgets.Output()
from IPython.display import display

def clicked(b):
    output.clear_output()
    with output:
        _norm = True
        _sortby = 'title'
        _query = querybox.value

        if (normalizedradio.value == "false"):
            _norm = False

        if (sortradio.value == 'score'):
            _sortby = 'score'

        if (_query == ""):
            print("please enter a query")
        else:
            drawTilebars(_query,normalized=_norm,sortby=_sortby).display(renderer='svg')

querybox = widgets.Text(description='Query:')
searchbutton = widgets.Button(description="Search")
normalizedradio = widgets.RadioButtons(description="Normalized?",options=['true', 'false'])
sortradio = widgets.RadioButtons(description="Sort by",options=['title', 'score'])

searchbutton.on_click(clicked)
normalizedradio.observe(clicked,names=['value'])
sortradio.observe(clicked,names=['value'])

list_widgets = [widgets.VBox([widgets.HBox([querybox,searchbutton]),
                              widgets.HBox([normalizedradio,sortradio])])]
accordion = widgets.Accordion(children=list_widgets)
accordion.set_title(0,"Search Controls")
display(accordion,output)
```



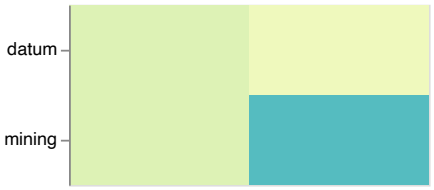
true
false

title
score

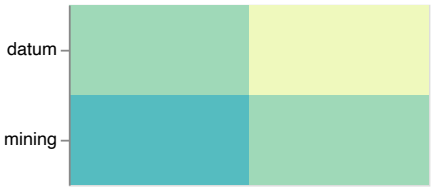
Data mining

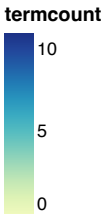


Tanagra (machine learning)



Sequential pattern mining

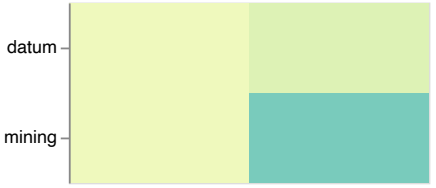




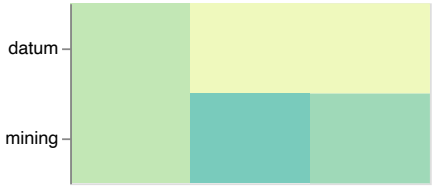
Relational data mining

Social media mining

Heikki Mannila

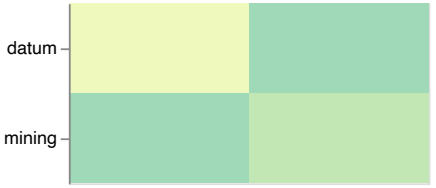


Web mining



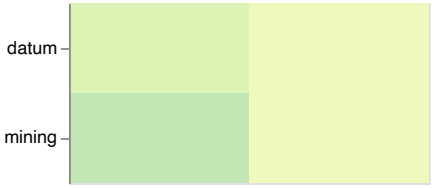
Rexer's Annual Data Miner Survey

Data stream mining



RapidMiner

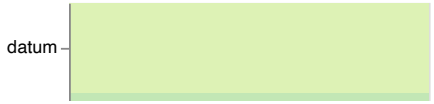
WINEPI



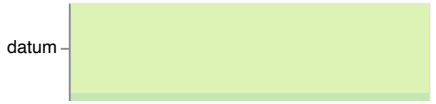
ELKI

Instance selection

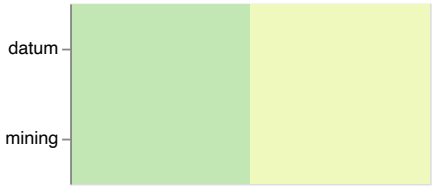
Data Mining and Knowledge Discovery



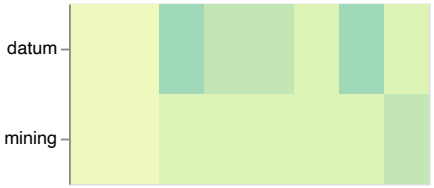
Trevor Hastie



SPSS Modeler



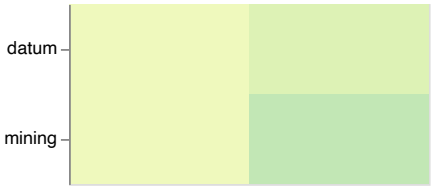
Hans-Peter Kriegel



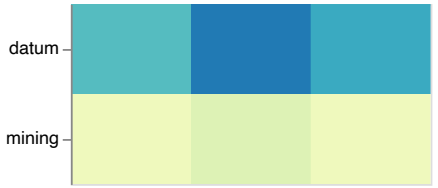
Microsoft Analysis Services

International Journal of Data Warehousing and Mining

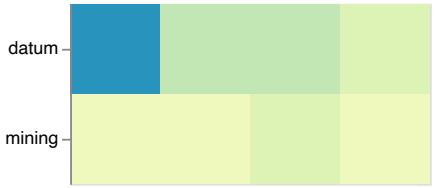
Formal concept analysis



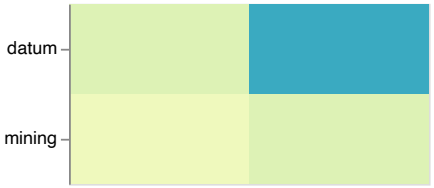
Data analysis



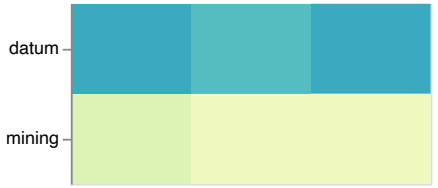
Data dredging



Curse of dimensionality

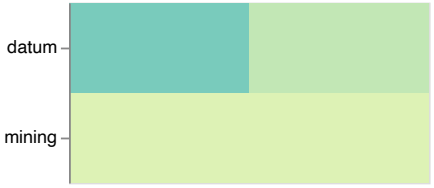


Seeq Corporation

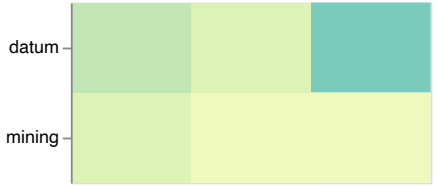


Machine learning

Information Harvesting



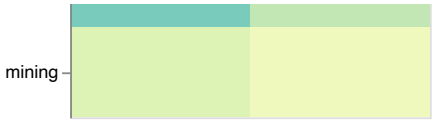
Anomaly detection

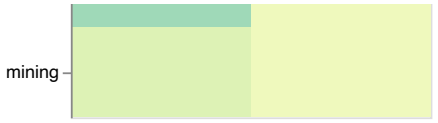


KNIME

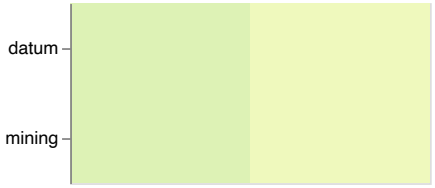
Orange (software)

R (programming language)

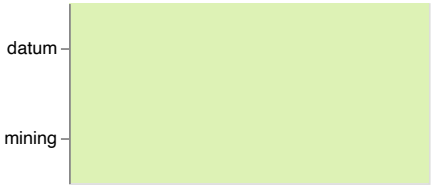




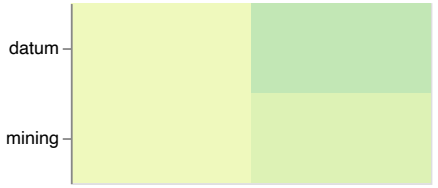
Piranha (software)



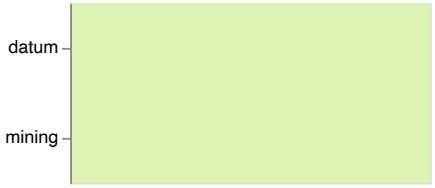
Oren Etzioni



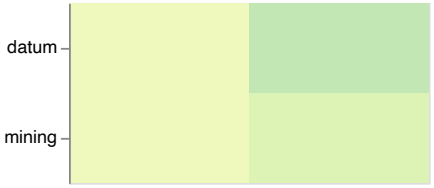
JMP (statistical software)



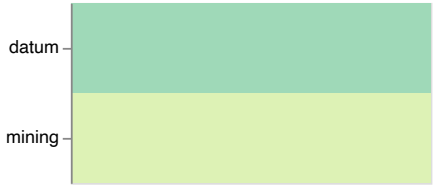
Jerome H. Friedman



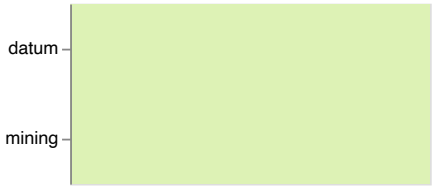
Dlib



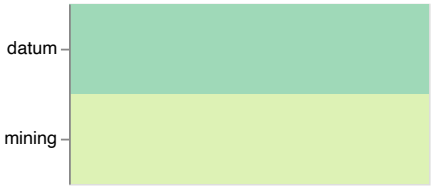
Astrostatistics



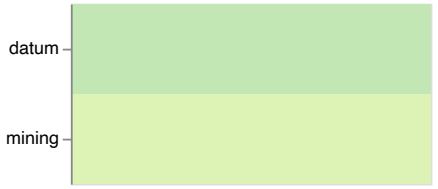
Ross Quinlan



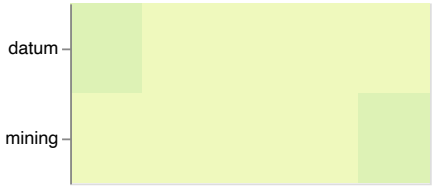
Random mapping



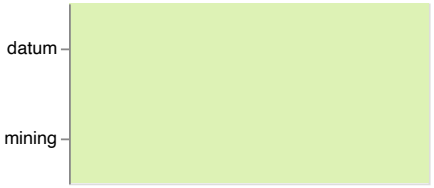
COMPLEAT (Bioinformatics tool)



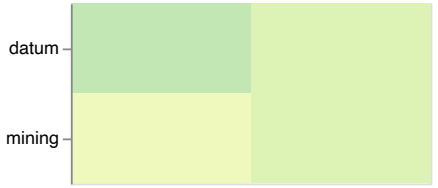
Minimum redundancy feature selection



Paul Viola



Machine learning in bioinformatics



Moose (analysis)

LIONsolver

Connect (computer system)

Problem 2.2 What did you do? (10 points)

Please detail why you made your design decisions. Again, we want you to improve on our less-than-optimal implementation of tilebars. You should reflect on how well you are meeting each of the objectives of tilebars.

I decided to create a small multiples chart instead of displaying all the tilebars in a vertical column. The ranking order is read from left to right across each row, then down to the next row. Having a small multiples chart helps the objective of tilebars as it minimizes the time spent scrolling. Additionally, when more charts are visible, it is easier to compare documents and choose which document is worth checking out further.

I decided to make all the tilebars the same size because while the length of the rectangle is a good visual indicator of the relative document lengths, when all the tilebars are scaled to the same size, it is much easier to read. Additionally, No information is lost as the length of the document is still apparent by the size of the tiles; the smaller the tiles, the longer the document. I made this compromise because if tilebars are laid out in a small multiples plot and are of different sizes, it is hard to read the figure as a whole and form comparisons across tilebars. It is much easier to read when tilebar sizes are uniform.

I also added a tooltip so that when the user hovers over a cell, a popup will display the lineid, the number of times the queried term appeared in the document and the text from the line in question. I thought this was a good way to display the big picture to users while still having the details available on demand. This way, if a user is curious about the contents of a document, they can briefly read a few lines and see if it is what they are looking for.

In [ ]: