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.

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 [12]:

```
# some utilitity 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")
            for lineid,docline in enumerate(doclines):
```

```python
                obj = {}
                obj['docid'] = docid;
                obj['title'] = doc['title']
                obj['lineid'] = lineid
                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):
                    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 [13]:

```python
# enable correct rendering
alt.renderers.enable('default')

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

Out[13]: 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.

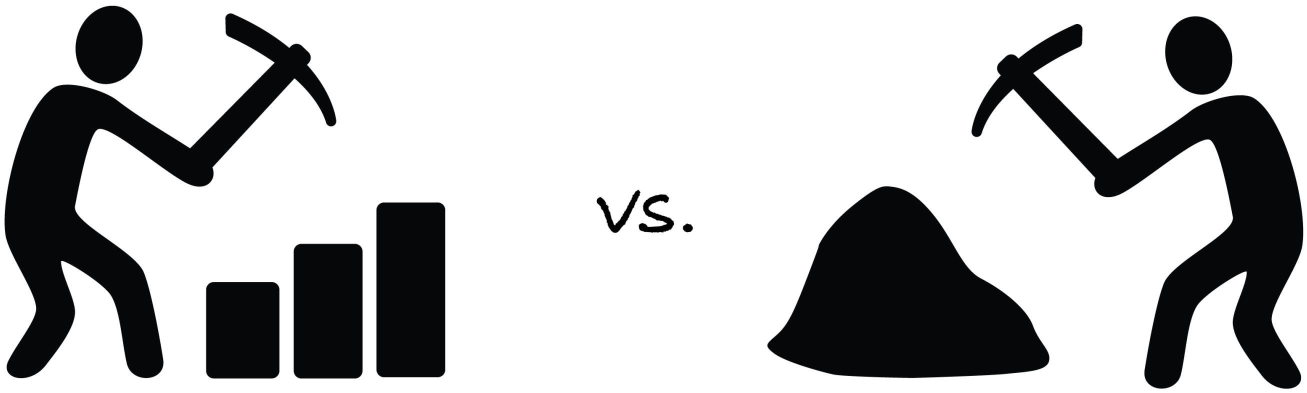
In [14]:

```python
# here's a few examples

query1 = "Data Mining"
print("The lemmatized version of "+query1+" is:", lemmatize(query1))

query1 = "executing awesome algorithms"
print("The lemmatized version of "+query1+" is:", lemmatize(query1))
```

```
The lemmatized version of Data Mining is: ['data', 'mining']
The lemmatized version of executing awesome algorithms is: ['execute', 'awesome', 'algorithm']
```

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 [15]:
```
# 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.jsonl')
# miningdf.to_csv("assets/miningvmining.csv",index=False)

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

In [16]:
```
# 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(5)
```

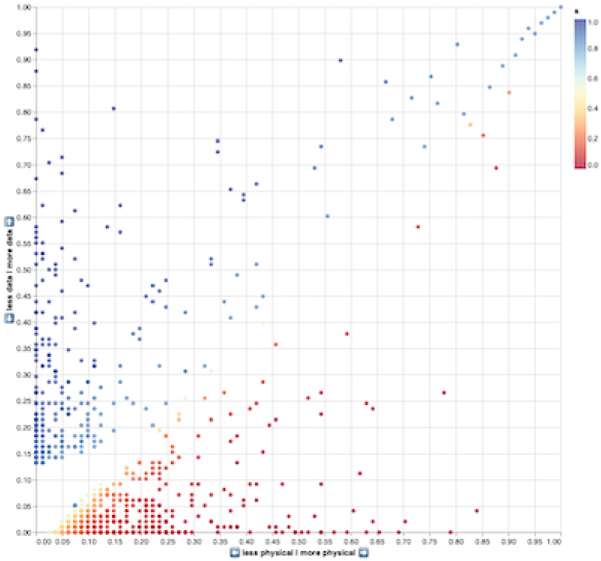
Out[16]:

	docid	title	text	label	category
154	154	Qualitative comparative analysis	in statistic , qualitative comparative analys...	data mining	Data analysis
129	129	General Architecture for Text Engineering	general architecture for text engineering or ...	data mining	Data mining and machine learning software
76	76	Sebastian Thrun	sebastian thrun (bear may 14 , 1967) be an ...	data mining	Machine learning researchers
400	198	Old Hundred Gold Mine	the old hundred gold mine be a gold mine in s...	real mining	NaN
322	120	Girdwood, Anchorage	girdwood be a resort town within the southern...	real mining	NaN

In [17]:
```
# you'll notice that the text is lemmatized. Here's the text for the first entry:
miningdf.head(1).text.values[0]
```

Out[17]:
```
' matlab ( matrix laboratory ) be a multi - paradigm numerical computing environment and proprietary programming language
develop by mathworks . matlab allow matrix manipulation , plot of function and datum , implementation of algorithm , creat
ion of user interface , and interfac with program write in other language , include c , c++ , c # , java , fortran and pyt
hon . \n\n although matlab be intend primarily for numerical computing , an optional toolbox use the mupad symbolic engine
, allow access to symbolic computing ability . an additional package , simulink , add graphical multi - domain simulation
and model - base design for dynamic and embed system . \n\n as of 2018 , matlab have more than 3 million user worldwide .
matlab user come from various background of engineering , science , and economic . '
```

We will be using the 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 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 [18]:

```
# 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 gets 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 an HTML page
    transform=st.Scalers.dense_rank # where to place identically ranked terms (on top of each other here)
)
```

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 [19]:

```
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', 'data', 'model', 'machi
ne', '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}
```

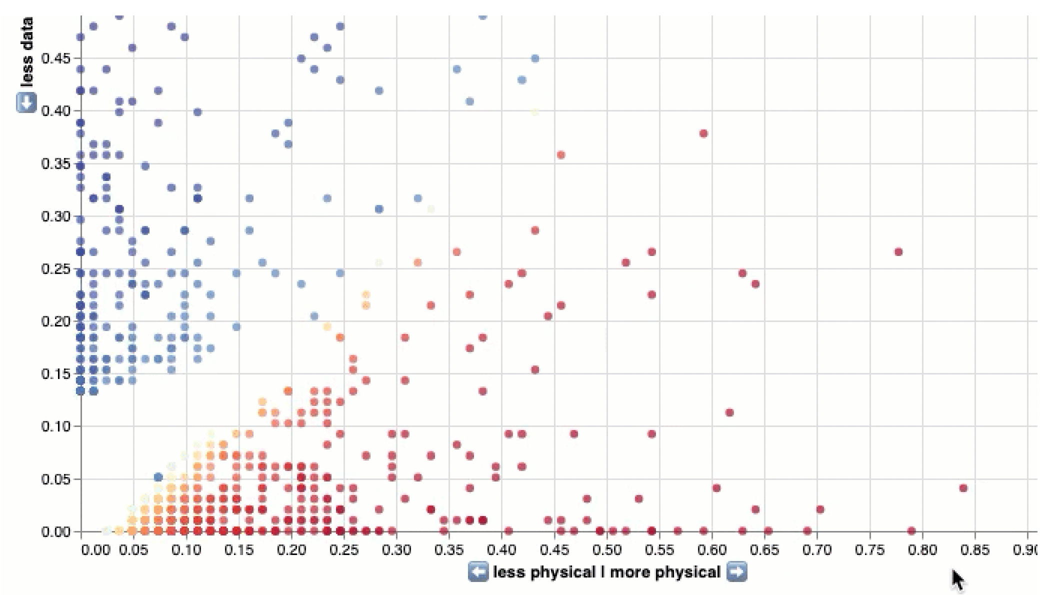
This is the first item in the data list. There are a number of fields here. You can look at the documentation for scattertext for the details. The only items we care about right now will be `x`, `y`, `term`, and `s`. These respectively tell us the x/y coordinate for the term, the term itself, and the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

In [20]:
```
scatterdata['data'][0]
```

Out[20]:

```
{'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}
```


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



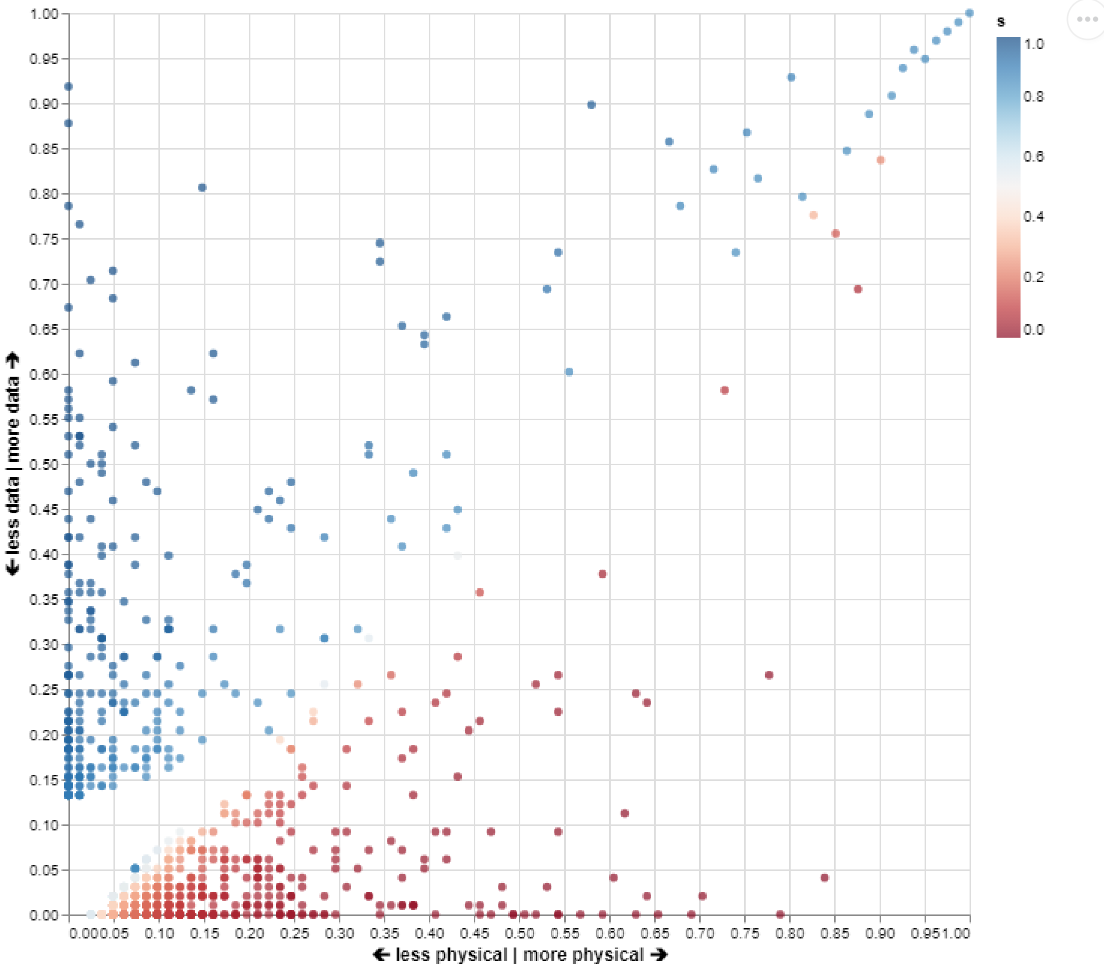
In [21]:
```
# add your solution here

df = pd.DataFrame(scatterdata['data'])

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

```
    toret = alt.Chart(df).mark_circle().encode(
            x=alt.X('x:Q', title='🡐 less physical | more physical 🡒'),
            y=alt.Y('y:Q', title='🡐 less data | more data 🡒'),
            color=alt.Color('s', scale=alt.Scale(scheme='redblue')),
            tooltip='term'
        ).configure_axis(
        titleFontSize=12, titleFontWeight="bold"
    ).properties(
        width=600, height=600
    )  # alt.Chart...
    # YOUR CODE HERE
    #raise NotImplementedError()
    return(toret)
```

In [22]:

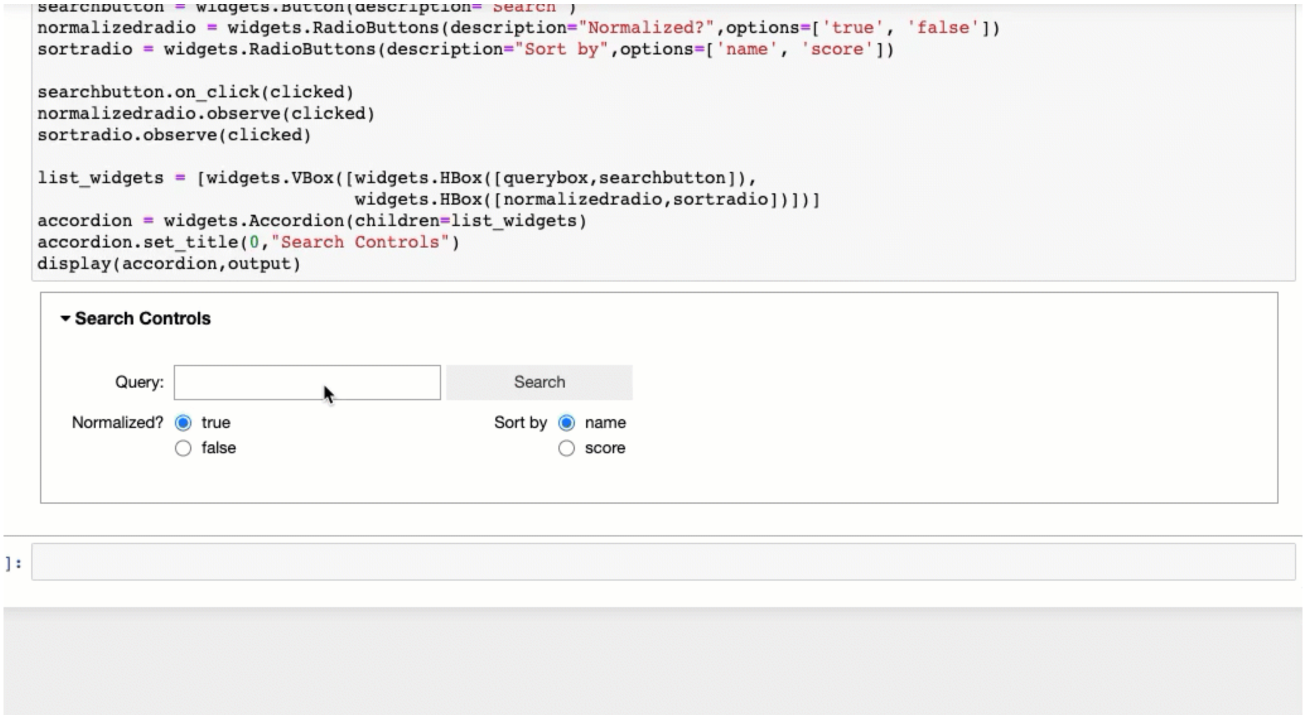
```
# if your code above works correctly, this should generate the plot
genScattertext()
```

Out[22]:



Problem 2

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"). If you build your interface correctly, you will hopefully get something like the following:



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.

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

In [23]:
```
# 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 [24]:
```
# let's look at the first few lines
dataminingdf.head(5)
```

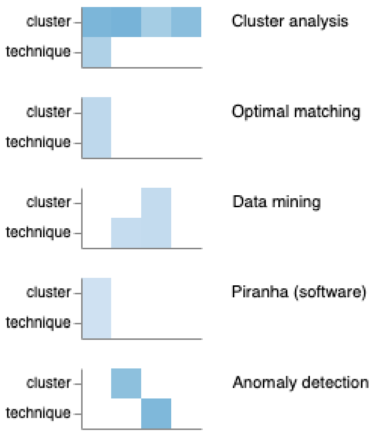
Out[24]:

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

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 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:



Again, you are not expected to reverse engineer our solution.

Some hints:

- 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.

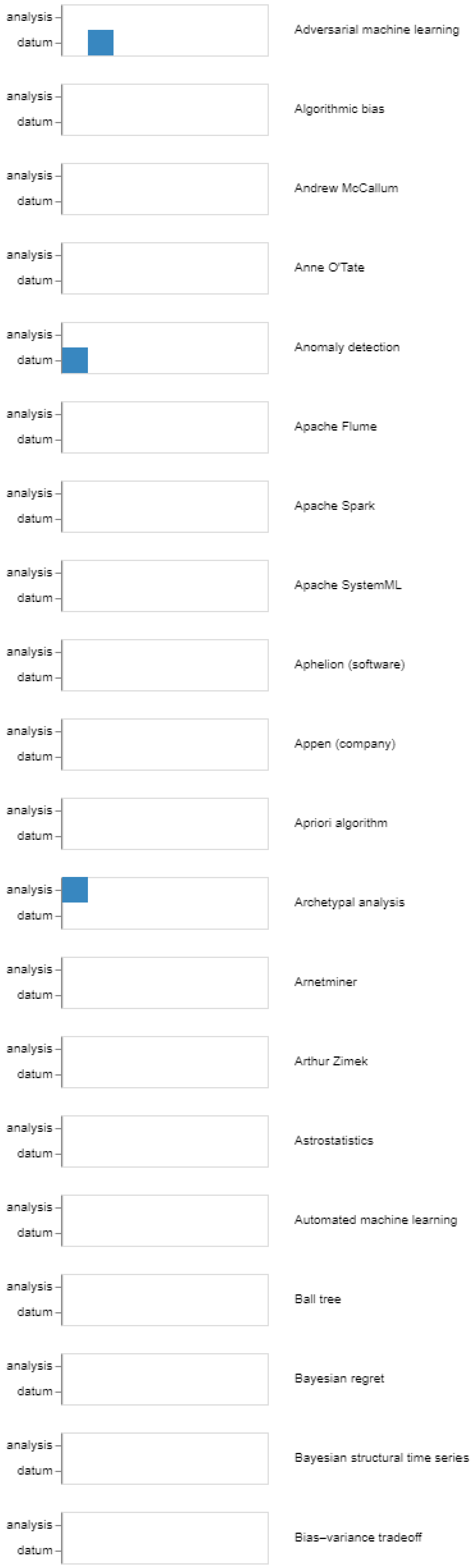
In [25]:

```
def drawTilebars(query,normalized=False,sortby='name'):
    # 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 in article?)
    # 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
    df = dataminingdf[['title','lineid','text','tokencount']]
    df['count'] = 0
    queries = lemmatize(query)
    df = df.assign(**{f'query_{i}': df.text.str.count(i) for i in queries})

    df = df.melt(
        id_vars=['title', 'lineid','tokencount'],
        value_vars=[f'query_{i}' for i in queries],
        var_name='lemma',
        value_name='score',
    ).replace(r'^query_', '', regex=True)
    df['norm_score'] = df['score'] / df['tokencount']
    df.rename(columns={'title':'name'}, inplace=True)

    df_group = df.groupby(['name','lemma','lineid']).score.min()[lambda x: x > 0]
    df_group=df_group.reset_index()
    # normalization dataframe
    df_norm = df.groupby(['name','lemma','lineid']).norm_score.min()[lambda x: x > 0]
    df_norm=df_group.reset_index()

    print("the lemmatized query terms are: ",lemmatize(query))
    print("nomalized is ",normalized)
    print("I will sort by", sortby)

    if normalized:
        if sortby == 'name':
            chart = alt.Chart(df_norm).mark_rect().encode(
                        alt.X('lineid:N', axis=alt.Axis(labels=False)),
                        alt.Y('lemma:O',title=None),
                        color=alt.Color('score:N',
                                        legend=None,
```

```
                                        scale=alt.Scale(
                                        domain=(0,2),
                                        scheme='blues')
                                        ),
                        tooltip='score:Q'
                    ).facet(
                        row=alt.Row('name',
                                    title=None,
                                    sort=alt.EncodingSortField('name'),
                                    header=alt.Header(
                                        labelOrient='right',
                                        labelPadding=20,
                                        labelAngle=0))
                    )
        else:
            chart = alt.Chart(df_norm).mark_rect().encode(
                        alt.X('lineid:N', axis=alt.Axis(labels=False)),
                        alt.Y('lemma:O',title=None),
                        color=alt.Color('score:N',
                                        legend=None,
                                        scale=alt.Scale(
                                        domain=(0,2),
                                        scheme='blues')
                                        ),
                        tooltip='score:Q'
                    ).facet(
                        row=alt.Row('name',
                                    title=None,
                                    sort=alt.EncodingSortField('score'),
                                    header=alt.Header(
                                        labelOrient='right',
                                        labelPadding=20,
                                        labelAngle=0))
                    )
    else:
        if sortby == 'name':
            chart = alt.Chart(df_group).mark_rect().encode(
                        alt.X('lineid:N', axis=alt.Axis(labels=False)),
                        alt.Y('lemma:O',title=None),
                        color=alt.Color('score:N',
                                        legend=None,
                                        scale=alt.Scale(
                                        domain=(0,2),
                                        scheme='blues')
                                        ),
                        tooltip='score:Q'
                    ).facet(
                        row=alt.Row('name',
                                    title=None,
                                    sort=alt.EncodingSortField('name'),
                                    header=alt.Header(
                                        labelOrient='right',
                                        labelPadding=20,
                                        labelAngle=0))
                    )
        else:
            chart = alt.Chart(df_group).mark_rect().encode(
                        alt.X('lineid:N', axis=alt.Axis(labels=False)),
                        alt.Y('lemma:O',title=None),
                        color=alt.Color('score:N',
                                        legend=None,
                                        scale=alt.Scale(
                                        domain=(0,2),
                                        scheme='blues')
                                        ),
                        tooltip='score:Q'
                    ).facet(
                        row=alt.Row('name',
                                    title=None,
                                    sort=alt.EncodingSortField('score'),
                                    header=alt.Header(
                                        labelOrient='right',
                                        labelPadding=20,
                                        labelAngle=0))
                    )

    return chart
# YOUR CODE HERE
# raise NotImplementedError()
```

```
In [26]: drawTilebars("data analysis",normalized=False,sortby='name').display()
```

```
the lemmatized query terms are:  ['datum', 'analysis']
nomalized is  False
I will sort by name
```



Bing Predicts

Biopac student lab

BisQue (Bioimage Analysis and Management Platform)

CANalyzer

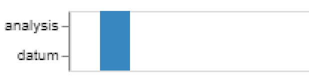
CN2 algorithm

COMPLEAT (Bioinformatics tool)

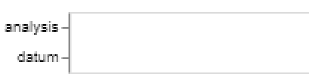
CellCognition

Cellebrite

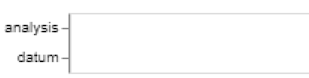
Chih-Jen Lin



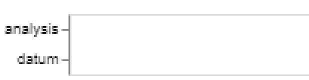
Cluster analysis



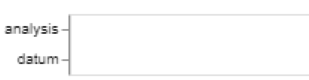
Combinatorial data analysis



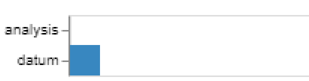
Computational X



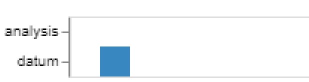
Computational learning theory



Connect (computer system)



Continuous analytics

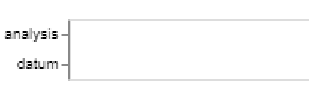


Cubes (OLAP server)

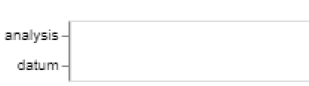
Curse of dimensionality

DADiSP

DIVA software



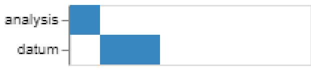
Dark data



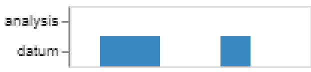
Data Mining and Knowledge Discovery

Data analysis

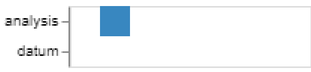
Data analysis expressions



Data dredging

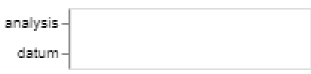


Data exploration



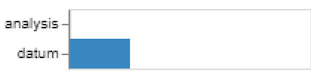
Data mining

Data profiling



Data stream mining

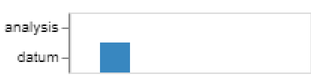
Data-informed decision-making



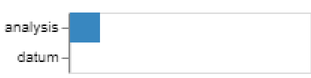
DataScene

Dataiku

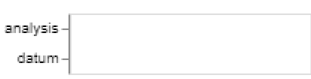
Distributed R



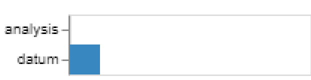
Dlib



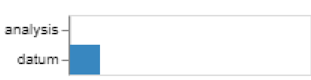
Dominance-based rough set approach



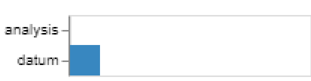
Double mass analysis



Dynamic time warping



ELKI



Eager learning

Early stopping

EgoNet

Elastic map

Encog

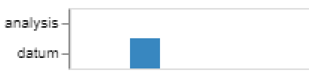
Epi Map

Error level analysis

FERET (facial recognition technology)

FERET database

Feature scaling



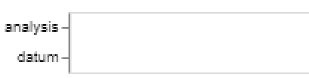
Federated learning

Fityk

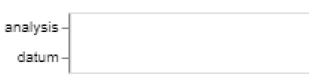
Fluentd

Formal concept analysis

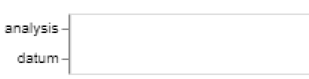
Generative model



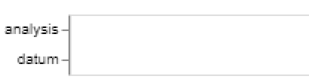
Geoffrey J. Gordon



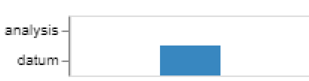
Google Flu Trends



Grace Wahba



H2O (software)



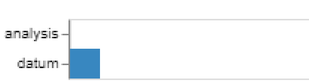
Hans-Peter Kriegel

Health care analytics

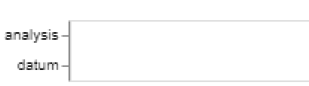
Heikki Mannila

HippoDraw

Hyperparameter (machine learning)

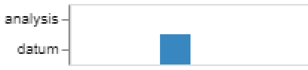


Hyperparameter optimization



I2 Limited

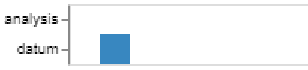
IcCube



Inauthentic text

Inductive probability

InfiniteGraph

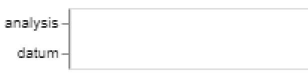


Information Harvesting

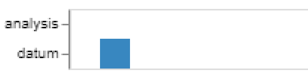
Instance selection

Instance-based learning

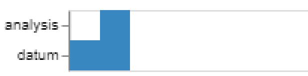
International Journal of Data Warehousing and Mining



Item tree analysis



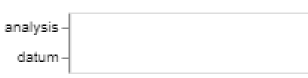
JMP (statistical software)



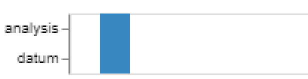
Java Analysis Studio

Jerome H. Friedman

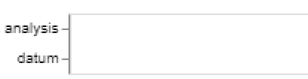
Julia (programming language)



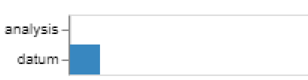
K-nearest neighbors algorithm



KNIME



Kernel density estimation



Kernel embedding of distributions

Kernel methods for vector output

Kernel principal component analysis

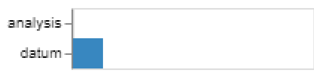
Kirix Strata

Kunihiko Fukushima

LIONsolver

LanguageWare

Leabra

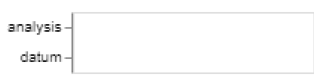


Learning curve (machine learning)

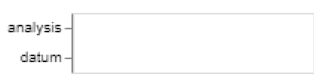
Learning to rank

Linde–Buzo–Gray algorithm

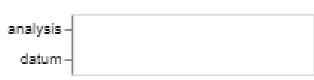
Lise Getoor



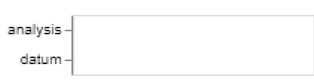
List of datasets for machine-learning research



Local outlier factor



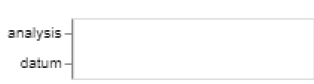
Logic learning machine



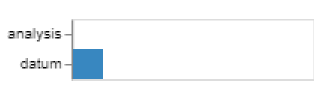
Léon Bottou

MATLAB

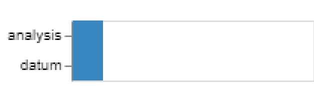
MEX file



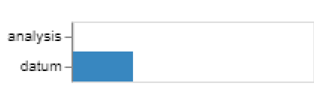
Machine learning



Machine learning in bioinformatics



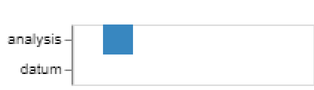
MagicPlot



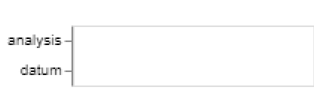
Maltego

Maple (software)

Massive Online Analysis



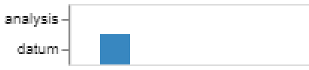
MeeMix



MetaboAnalyst

Microsoft Analysis Services

Minimum redundancy feature selection



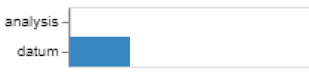
Mlpy

Moose (analysis)

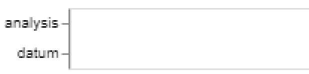
MountainsMap

Multifactor dimensionality reduction

Multilinear subspace learning



Multiple kernel learning



Natural Language Toolkit

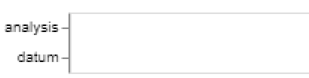
Negative testing

Neural Designer

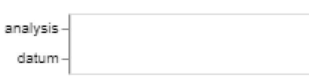
OPeNDAP

Ocean Data View

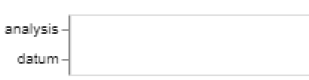
Open coding



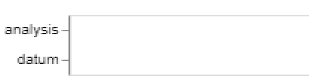
OpenScientist



Optimal matching



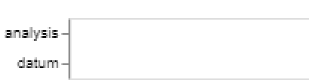
Orange (software)



Oren Etzioni

Origin (data analysis software)

Out-of-bag error



Outline of machine learning

PVLV

PatientsLikeMe

Pattern recognition

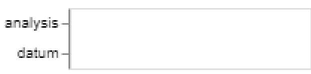
Paul Viola

Photoanalysis

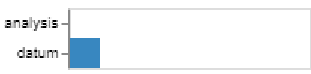
Pipeline Pilot

Piranha (software)

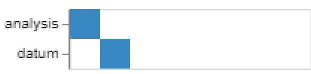
Pivot table



Poimapper

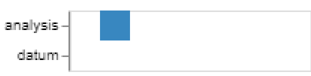


Post hoc analysis



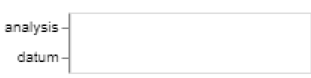
PowerLab

Prior knowledge for pattern recognition

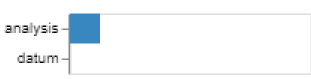


Programming with Big Data in R

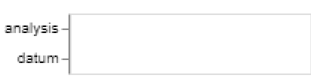
Qloo



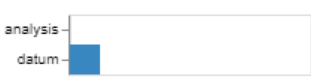
Qualitative comparative analysis



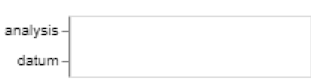
Quantum machine learning



R (programming language)



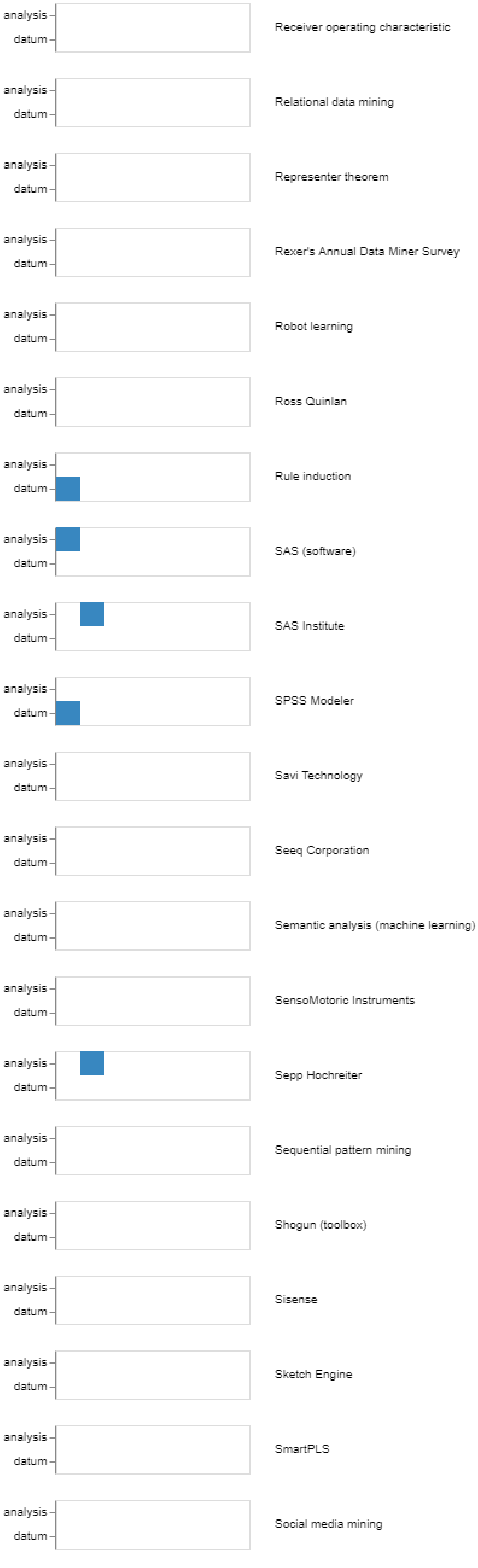
ROOT



Random indexing

Random mapping

RapidMiner


analysis
datum
Receiver operating characteristic
Relational data mining
Representer theorem
Rexer's Annual Data Miner Survey
Robot learning
Ross Quinlan
Rule induction
SAS (software)
SAS Institute
SPSS Modeler
Savi Technology
Seeq Corporation
Semantic analysis (machine learning)
SensoMotoric Instruments
Sepp Hochreiter
Sequential pattern mining
Shogun (toolbox)
Sisense
Sketch Engine
SmartPLS
Social media mining

Social profiling

Sparse dictionary learning

Speakeasy (computational environment)

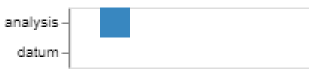
Stability (learning theory)

Statistical classification

Statistical learning theory

Stochastic block model

Stochastic gradient descent



Structured sparsity regularization

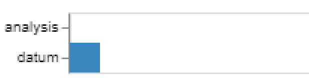
Symbolic data analysis

Tableau Software

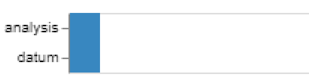
Tanagra (machine learning)

Technology mining

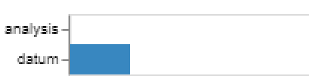
Teiresias algorithm



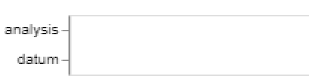
Tidyverse



Topological data analysis



Training, validation, and test sets

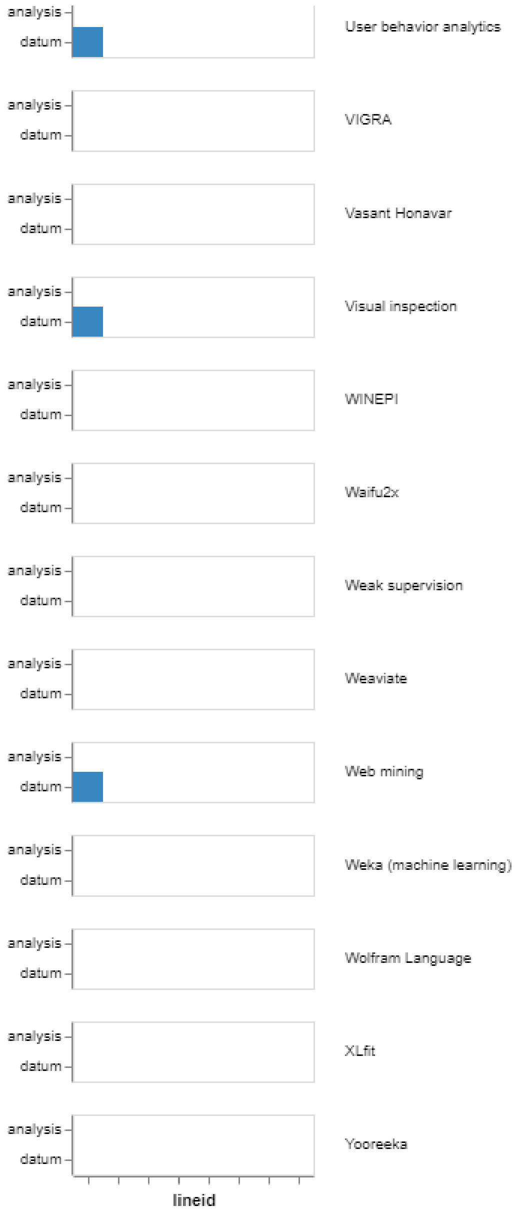


Trevor Hastie

UNISoN (Social Network Analysis Tool)

Uncertain data

Universal portfolio algorithm



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 [27]:

```
output = widgets.Output()
from IPython.display import display

def clicked(b):
    output.clear_output()
    with output:
        _norm = True
        _sortby = 'name'
        _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()

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=['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)
```

Additional comments

If you think we need to know anything about your solution or design choices, feel free to add details here.