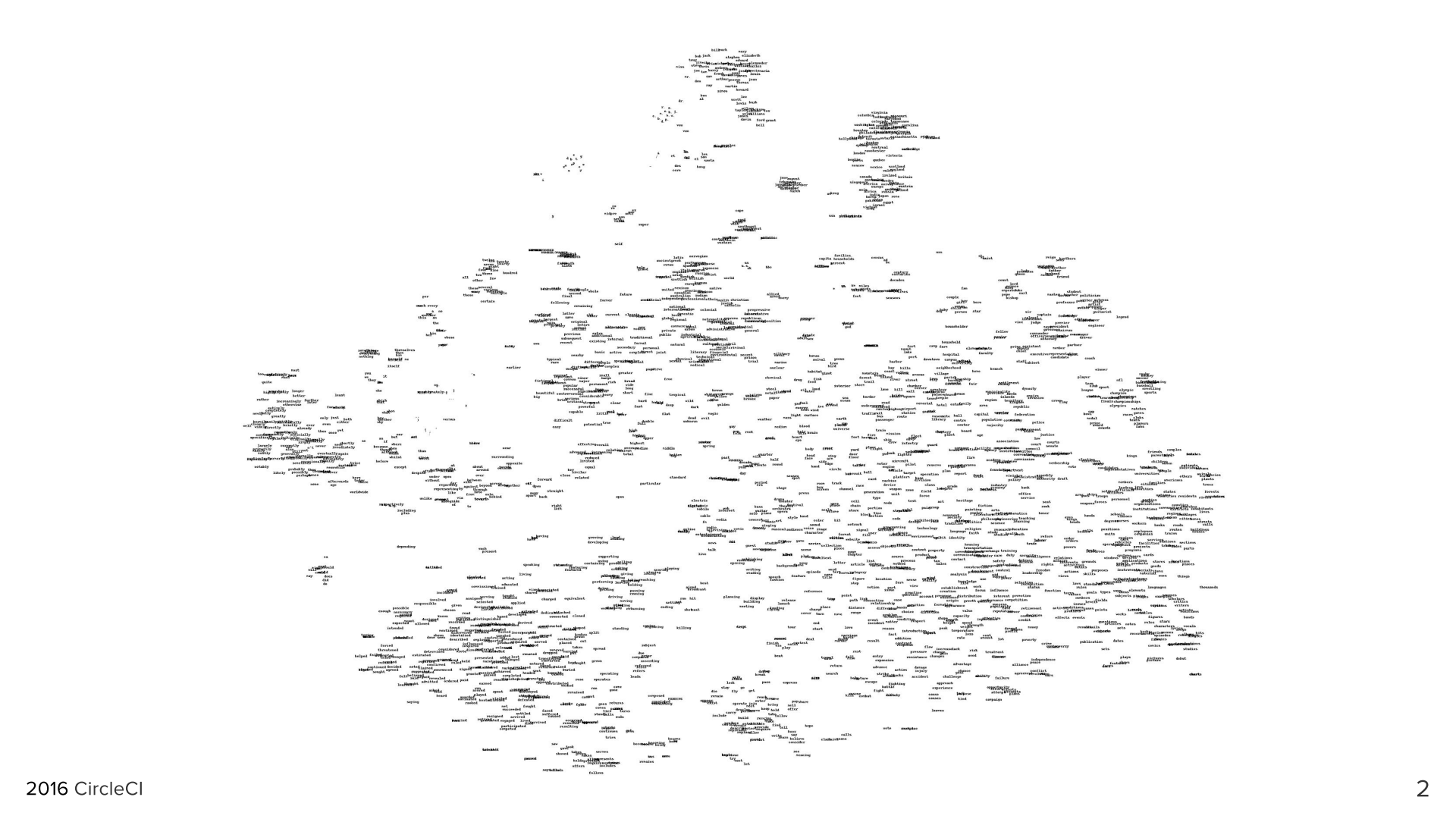
Charting the English Language

...In Pure Clojure



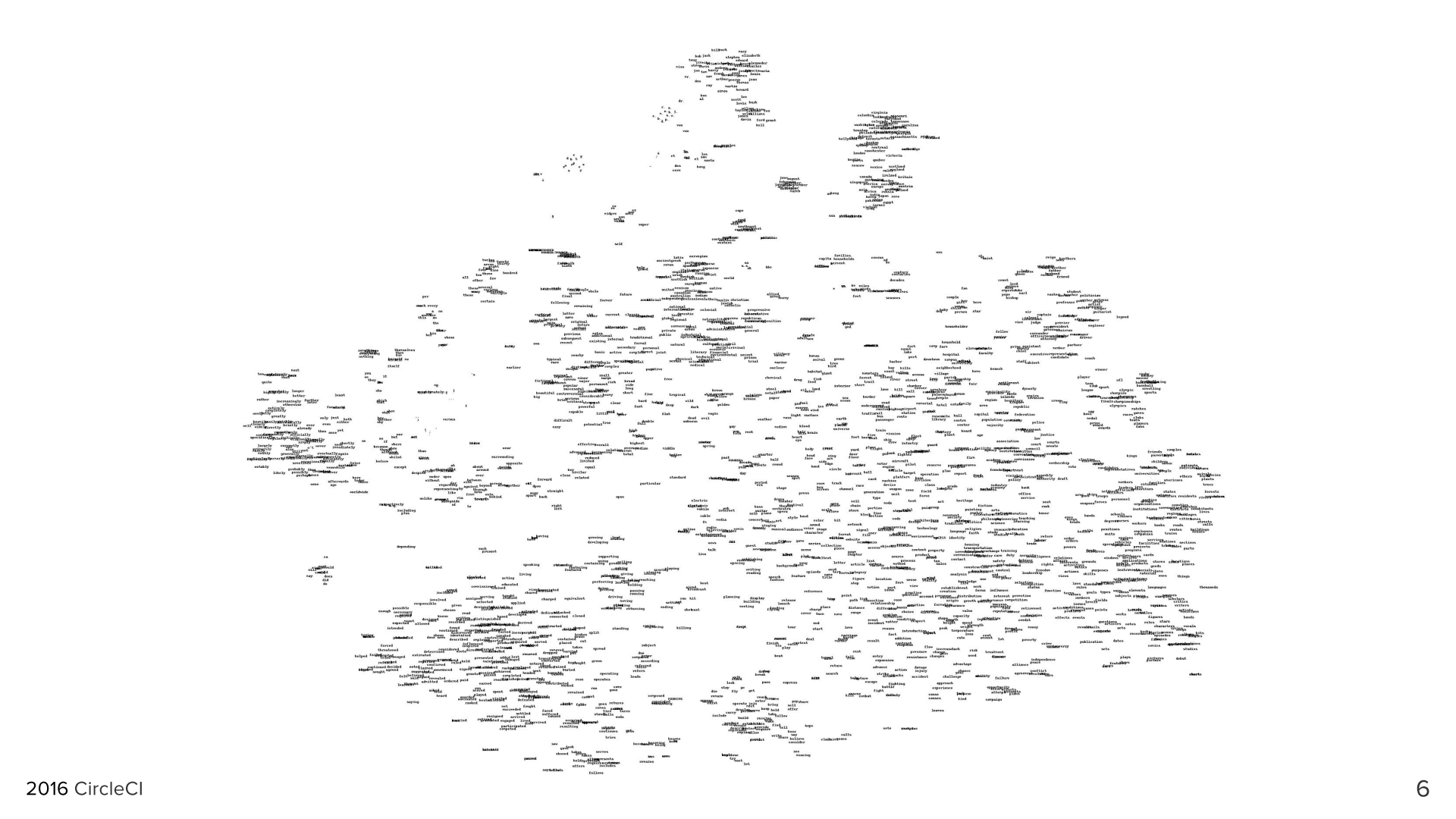
Goals for me

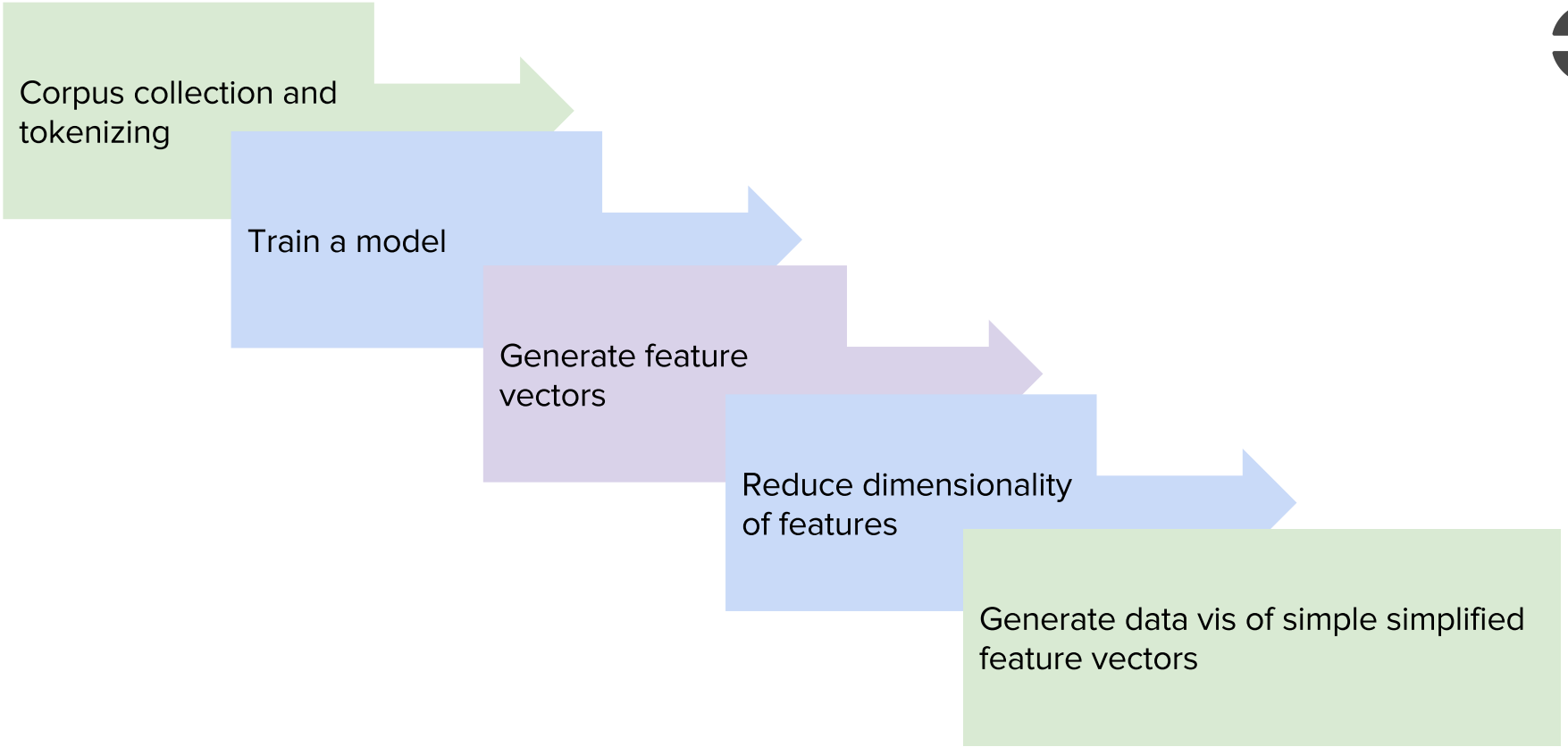
- Yak shaving
- Deep dive into the JVM
- Deep dive into Clojure
- Learning about the state of ML in Clojure
- Making some cool images to use as art in my flat

Goals for those attending talk

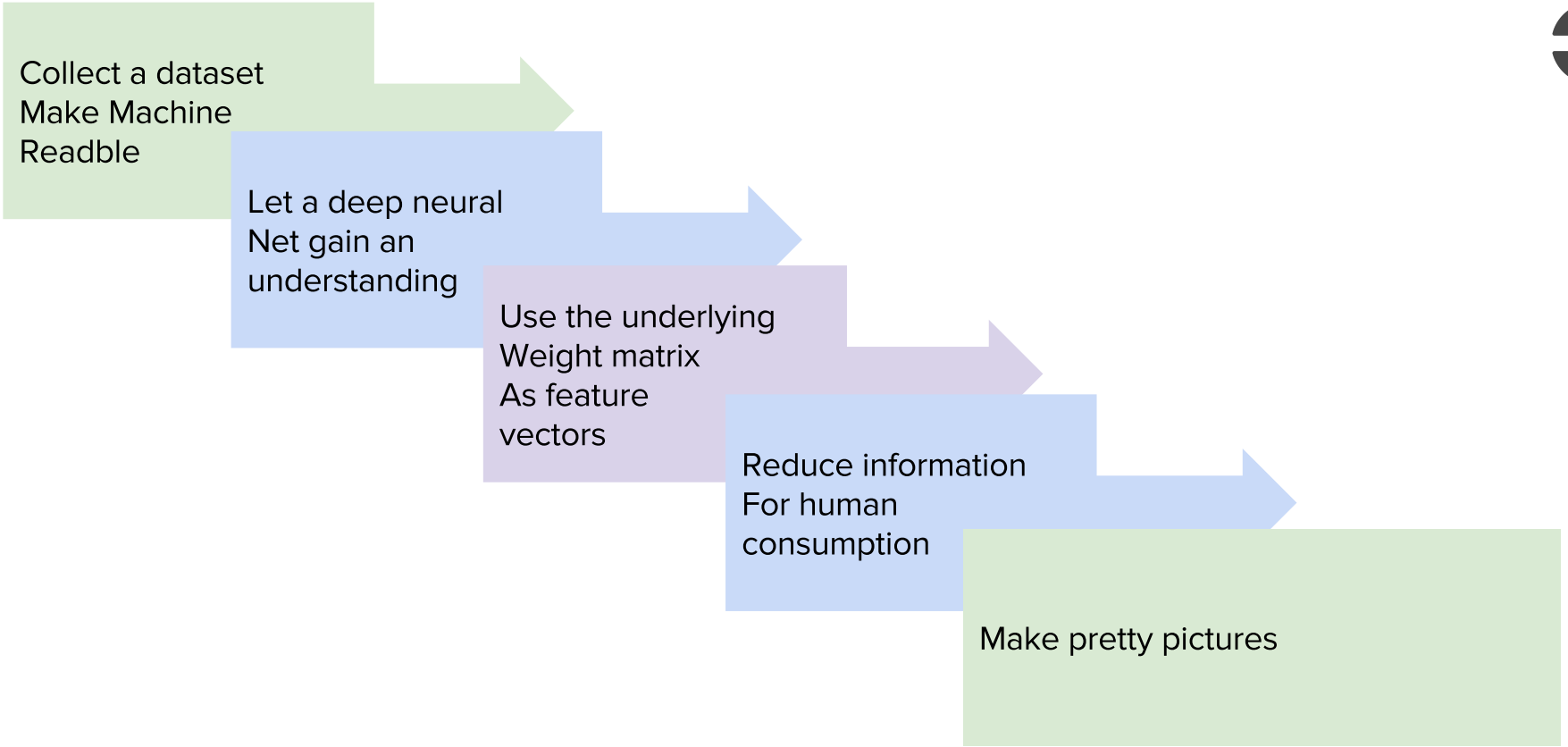
- History on this entire process
- Checklist for how to proceed in actually charting the english language in pure Clojure
- Highlighting some cool features/things I learned in my code
- Clojure for ML
- Highlighting some images and resources which are frankly astounding

How It’s Built: ML Edition

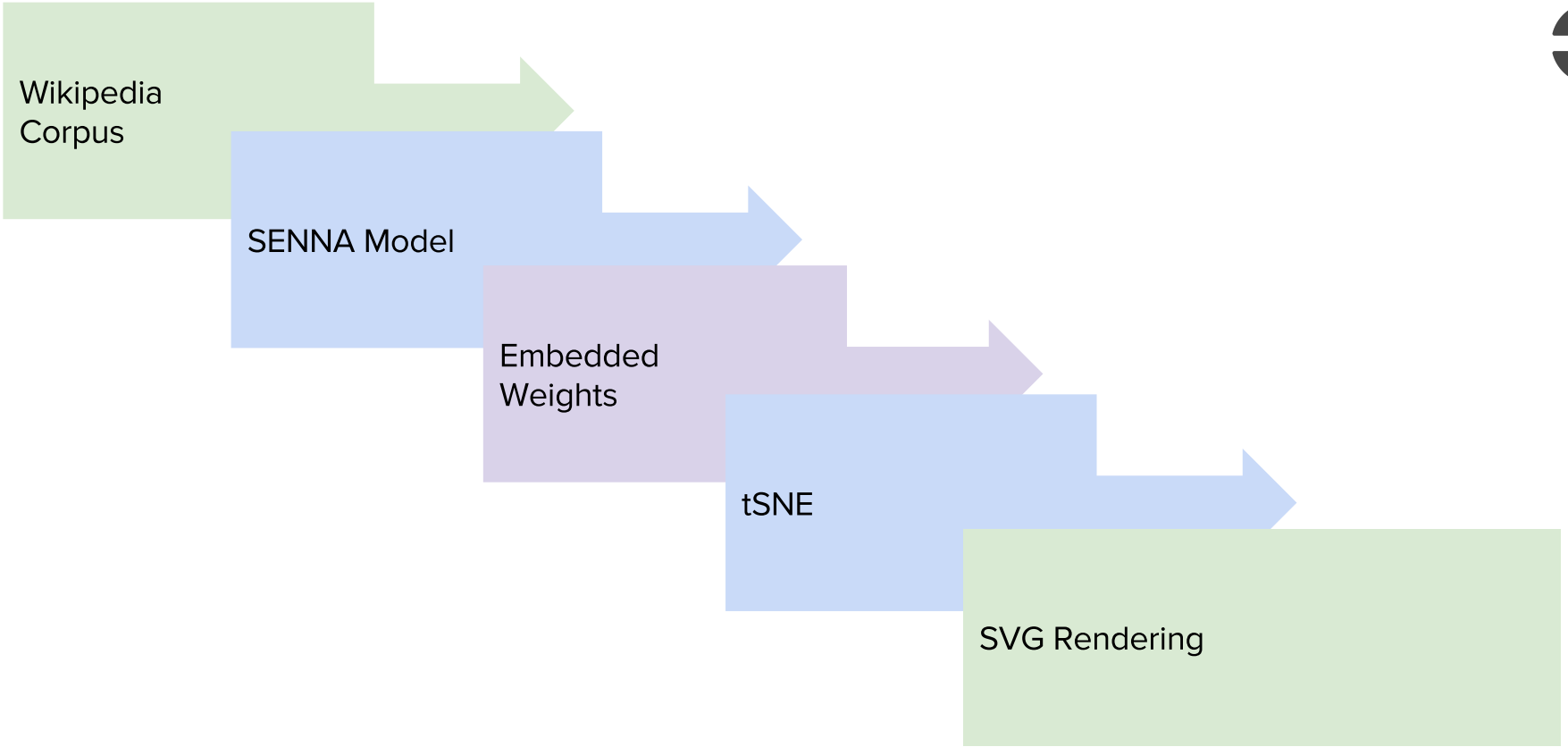



Corpus collection and tokenizing
Train a model
Generate feature vectors
Reduce dimensionality of features
Generate data vis of simple simplified feature vectors

The path



Simplified explanation



The technology/resources at each point

SENNA

Developed in large part due to Collobert and Weston in ~2011

Deep neural net approach to machine learning in language processing

This means unsupervised, the machine just figures things out on its own

Uses a multilayered architecture to turn sentences into parts of speech, semantic meanings, etc.

tSNE

t-Distributed Stochastic Neighbor Embedding

Metric space: defined distance from one point to another

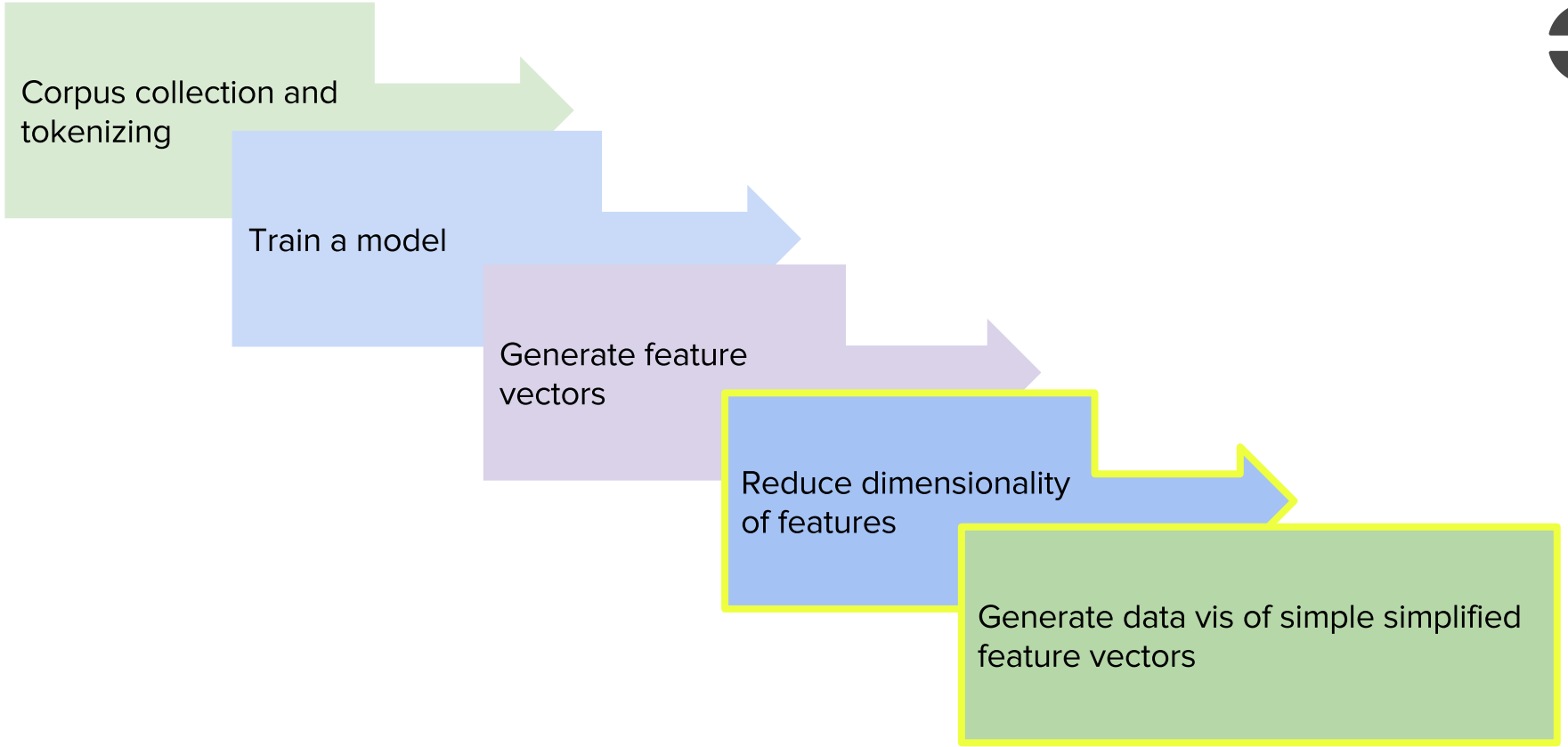
tSNE arose from a desire to better redefine metric spaces

ex: consider the following words

`conj`, `conference`, `parenthese`

when you have 3 dimensions, you can imagine `conference` and `parenthese` both being close to `conj` but far away from each other

with only 2 dimensions you want this same relationship to hold but it's easy to see that it might not


Corpus collection and tokenizing
Train a model
Generate feature vectors
Reduce dimensionality of features
Generate data vis of simple simplified feature vectors

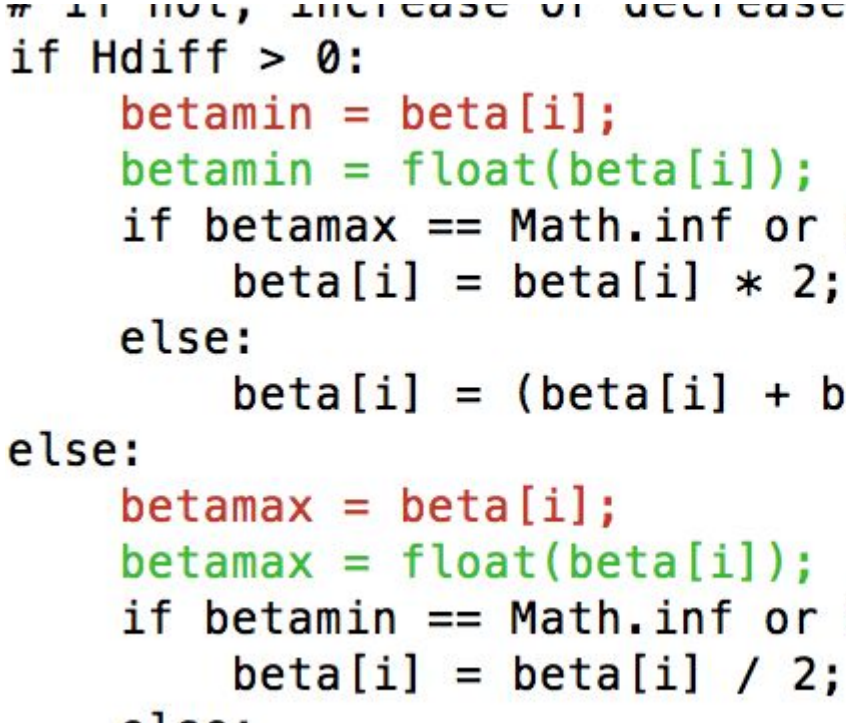
Where I got ^

What went wrong

What went wrong:

- I hitched my code's correctness to an implementation of tSNE which was buggy
- When failures would happen, it took a while to assess whether it was my fault or another bug
- Hitches in 3rd party software
- Unsurprisingly, ML from scratch isn’t the simplest thing in the world...

Buggy Code Implementations



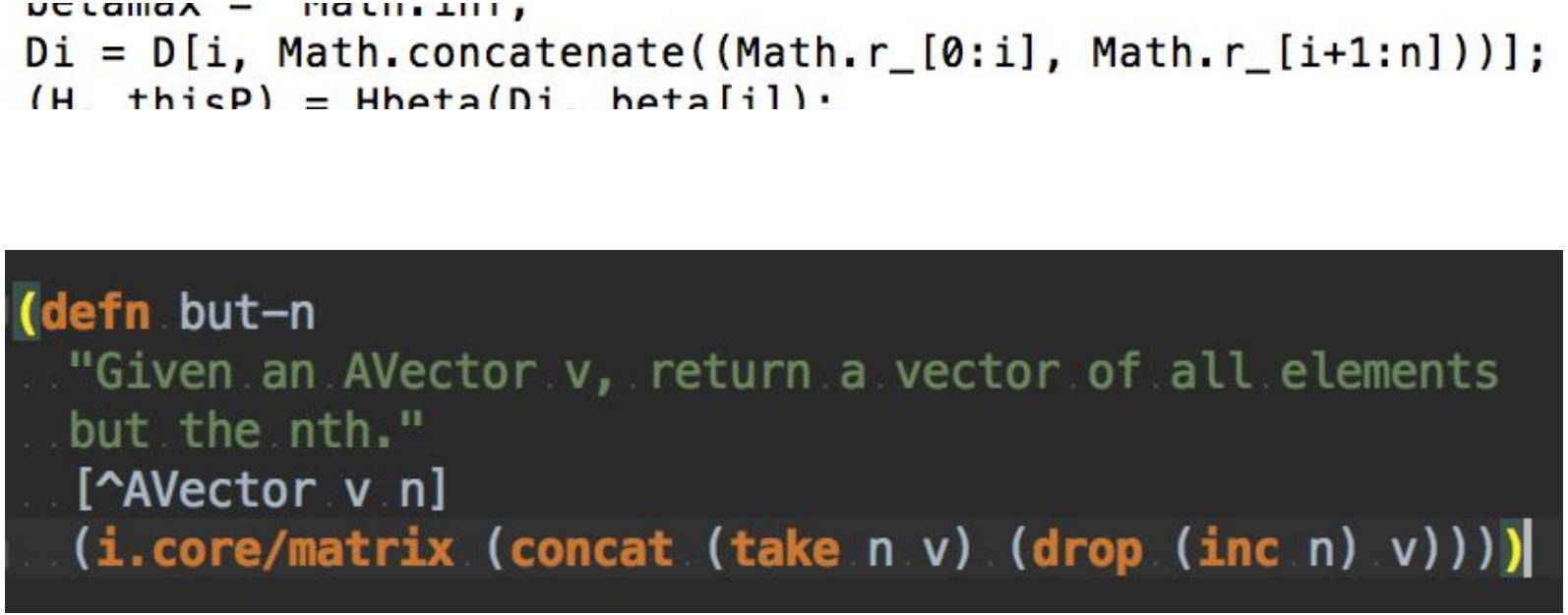
Should have used linear algebra definitions of code to start

Started using an academic Python implementation...which is not production Python code...written by programmers

Stable implementation shelled out to cpp

```
# if not, increase or decrease
if Hdiff > 0:
    betamin = beta[i];
    betamin = float(beta[i]);
    if betamax == Math.inf or
        beta[i] = beta[i] * 2;
    else:
        beta[i] = (beta[i] + b
else:
    betamax = beta[i];
    betamax = float(beta[i]);
    if betamin == Math.inf or
        beta[i] = beta[i] / 2;
```

~~Buggy~~ Obtuse Code Implementations



```
betaMax = Math.Inf;
Di = D[i, Math.concatenate((Math.r_[0:i], Math.r_[i+1:n]))];
(H, thisP) = Hbeta(Di, beta[i]);
```

```
(defn but-n
  "Given an AVector v, return a vector of all elements
  but the nth."
  [^AVector v n]
  (i.core/matrix (concat (take n v) (drop (inc n) v))))
```

Things like equivalence took a while to get used to understanding

(Also, it's probably mostly obtuse to someone who is a parentheses zealot)

Hiccups in Using 3rd Party Software

Precision in Clojure.core/Matrix

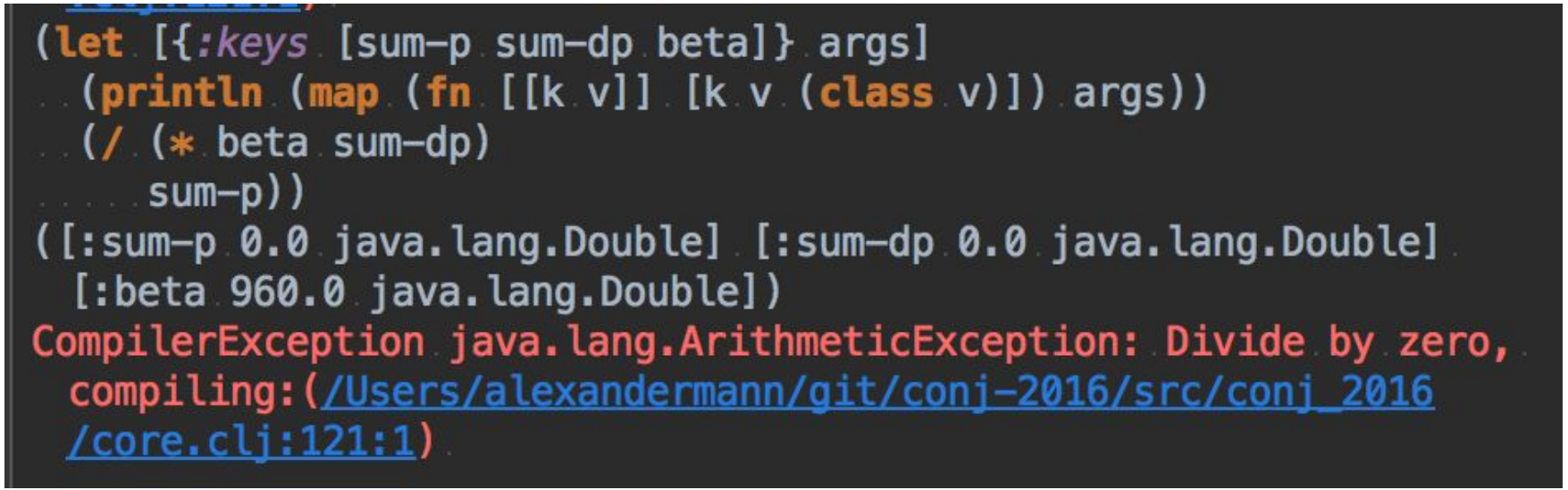
Apache Commons Matrix doesn't even load on modern Clojure

Dali requires that you require the namespaces of the features you want to use due to defmethod defining

http://dev.clojure.org/jira/browse/CLJ-1142

```
Clojure 1.9.0-alpha13
nREPL server started on port 64464 on hos
  127.0.0.1 - nrepl://127.0.0.1:64464
Loading src/conj_2016/core.clj...
CompilerException java.lang.RuntimeExcep
  many arguments to throw, throw expects
  Throwable instance, compiling:
  (apache_commons_matrix/core.clj:19:13)
```

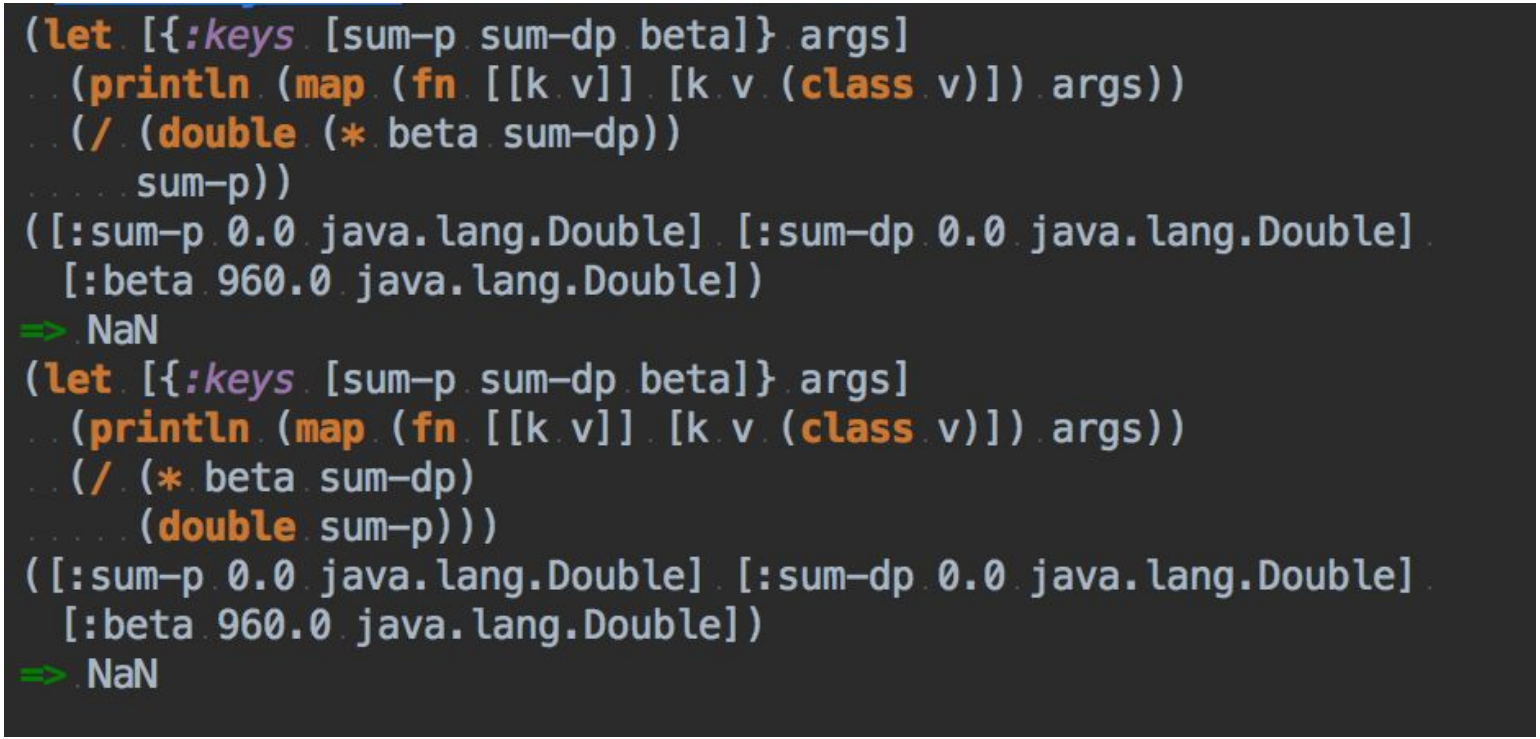
Jira Ticket CLJ-1142



```
(let [{:keys [sum-p sum-dp beta]} args]
  (println (map (fn [[k v]] [k v (class v)]) args))
  (/ (* beta sum-dp)
     sum-p))
([:sum-p 0.0 java.lang.Double] [:sum-dp 0.0 java.lang.Double]
 [:beta 960.0 java.lang.Double])
CompilerException java.lang.ArithmeticException: Divide by zero,
 compiling:(/Users/alexandermann/git/conj-2016/src/conj_2016
 /core.clj:121:1)
```

Clojure is not always innocent either

Jira Ticket CLJ-1142



```
(let [{:keys [sum-p sum-dp beta]} args]
  (println (map (fn [[k v]] [k v (class v)]) args))
  (/ (double (* beta sum-dp))
     sum-p))
([:sum-p 0.0 java.lang.Double] [:sum-dp 0.0 java.lang.Double]
 [:beta 960.0 java.lang.Double])
=> NaN
(let [{:keys [sum-p sum-dp beta]} args]
  (println (map (fn [[k v]] [k v (class v)]) args))
  (/ (* beta sum-dp)
     (double sum-p)))
([:sum-p 0.0 java.lang.Double] [:sum-dp 0.0 java.lang.Double]
 [:beta 960.0 java.lang.Double])
=> NaN
```

Clojure is not always innocent either

Consequences of “wrong”

What I ended up building:

- Took a Python implementation and made it into Clojure
- Made a generative testing suite which compares Clojure to Python
- Made a `fuzzy=` implementation that allows objects to be compared in many different modes
- Set of “parsers” which convert datasets to Clojure data

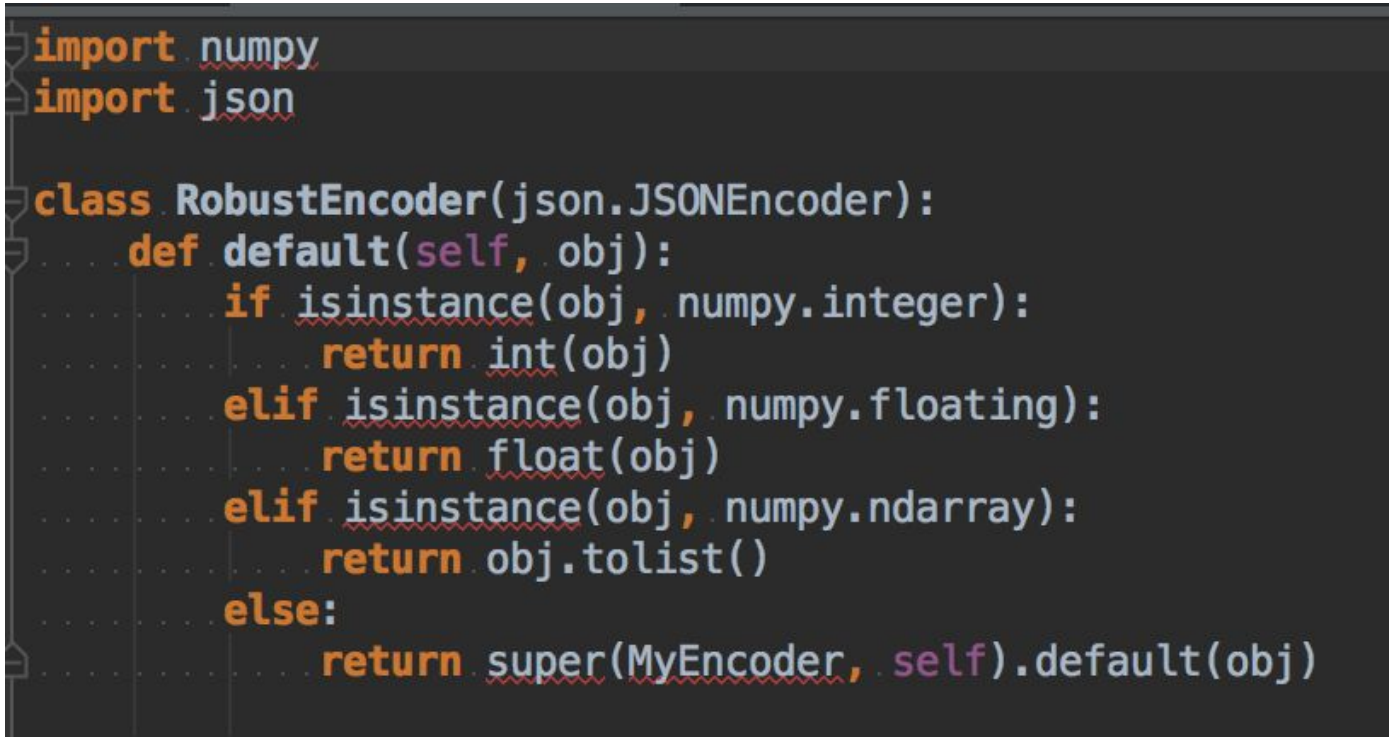
Generative Python Testing

Wrote generators for various types of Matrices

Python Interop capability (sort of)

```
{:repo   "tsne_cpp"
 :module "calc_tsne"
 :fname  "tsne"}
```

Generative Python Testing



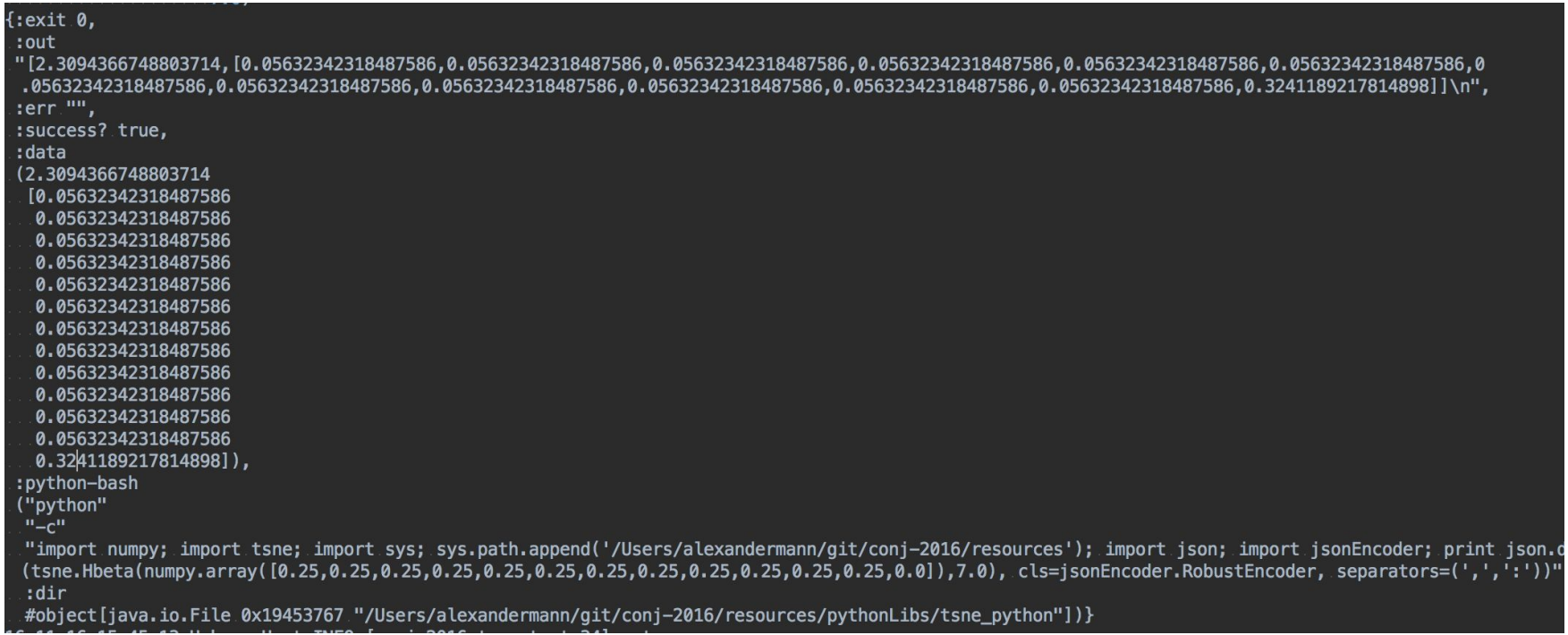
```
import numpy
import json

class RobustEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, numpy.integer):
            return int(obj)
        elif isinstance(obj, numpy.floating):
            return float(obj)
        elif isinstance(obj, numpy.ndarray):
            return obj.tolist()
        else:
            return super(MyEncoder, self).default(obj)
```

JSON Encoder so that Python can handle Numpy values:

https://github.com/AlexanderMann/conj-2016/blob/master/resources/jsonEncoder.py

Generative Python Testing



```
{:exit 0,
 :out
 "[2.3094366748803714,[0.05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0
  .05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0.05632342318487586,0.3241189217814898]]\n",
 :err "",
 :success? true,
 :data
 (2.3094366748803714
  [0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.05632342318487586
   0.3241189217814898]),
 :python-bash
 ("python"
  "-c"
  "import numpy; import tsne; import sys; sys.path.append('/Users/alexandermann/git/conj-2016/resources'); import json; import jsonEncoder; print json.d
  (tsne.Hbeta(numpy.array([0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.0]),7.0), cls=jsonEncoder.RobustEncoder, separators=(',',':'))"
  :dir
  #object[java.io.File 0x19453767 "/Users/alexandermann/git/conj-2016/resources/pythonLibs/tsne_python"])}
```

Example Python return value

Generative Python Testing

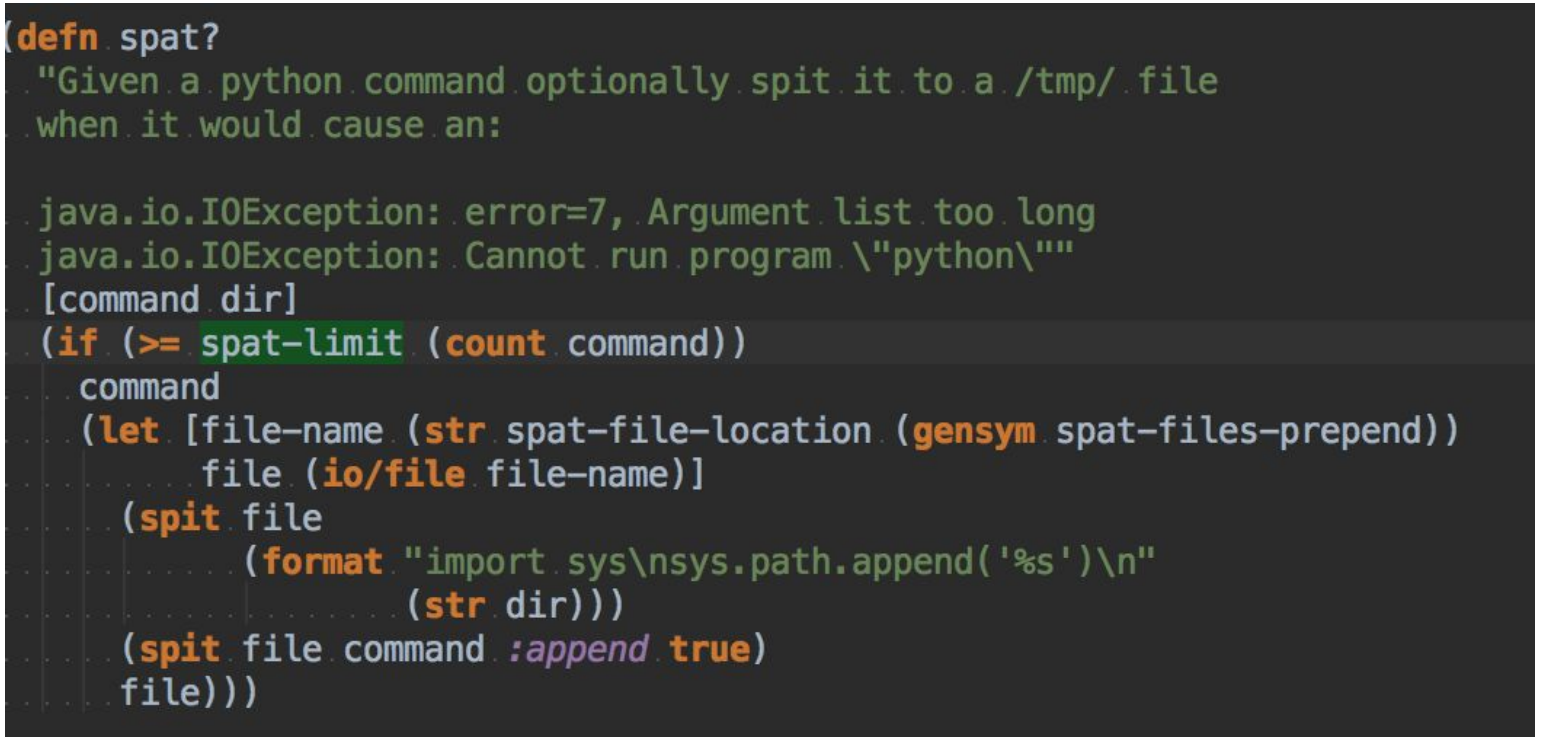
```clojure
(def spat-limit (-> (shell/sh "getconf" "ARG_MAX")
                    :out
                    string/trim
                    Integer/parseInt))
```

`getconf ARG_MAX`

(http://unix.stackexchange.com/questions/120642/what-defines-the-maximum-size-for-a-command-single-argument):

https://github.com/AlexanderMann/conj-2016/blob/master/src/conj_2016/util/python.clj

Generative Python Testing



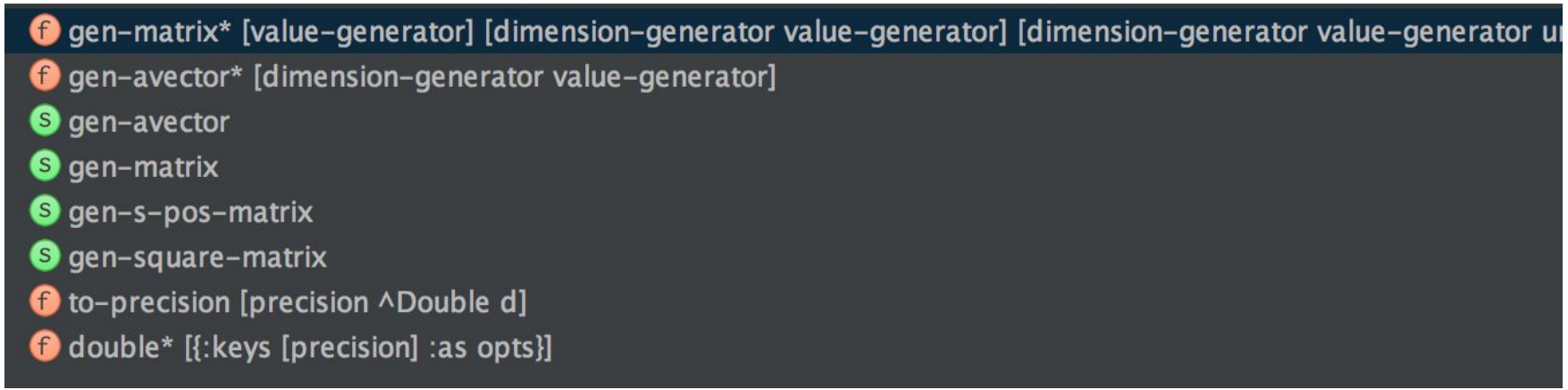
```clojure
(defn spat?
  "Given a python command optionally spit it to a /tmp/ file
  when it would cause an:

  java.io.IOException: error=7, Argument list too long
  java.io.IOException: Cannot run program \"python\""
  [command dir]
  (if (>= spat-limit (count command))
    command
    (let [file-name (str spat-file-location (gensym spat-files-prepend))
          file (io/file file-name)]
      (spit file
            (format "import sys\nsys.path.append('%s')\n"
                    (str dir)))
      (spit file command :append true)
      file)))
```

`spat?` (yes, the name is the worst):

https://github.com/AlexanderMann/conj-2016/blob/master/src/conj_2016/util/python.clj

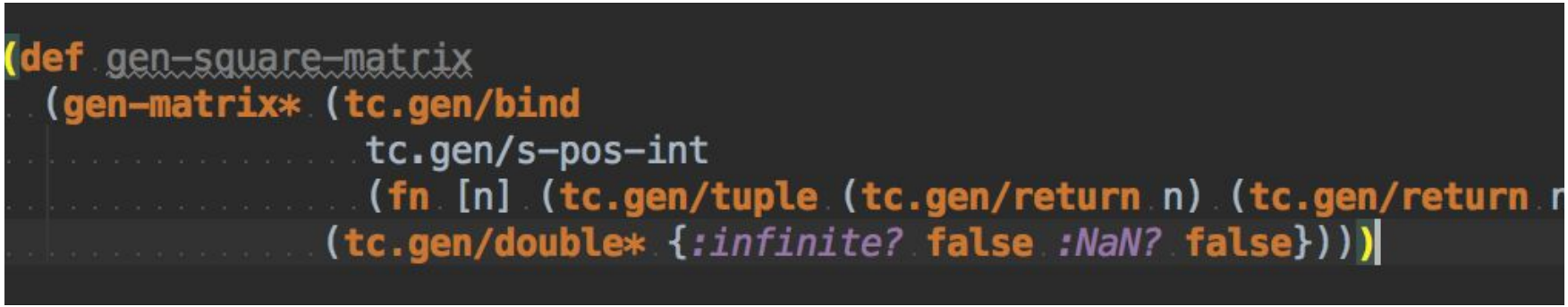
Matrix Generators



Matrix Generators of all shapes and sizes:

https://github.com/AlexanderMann/conj-2016/blob/master/test/conj_2016/util/generator.clj

Matrix Generators

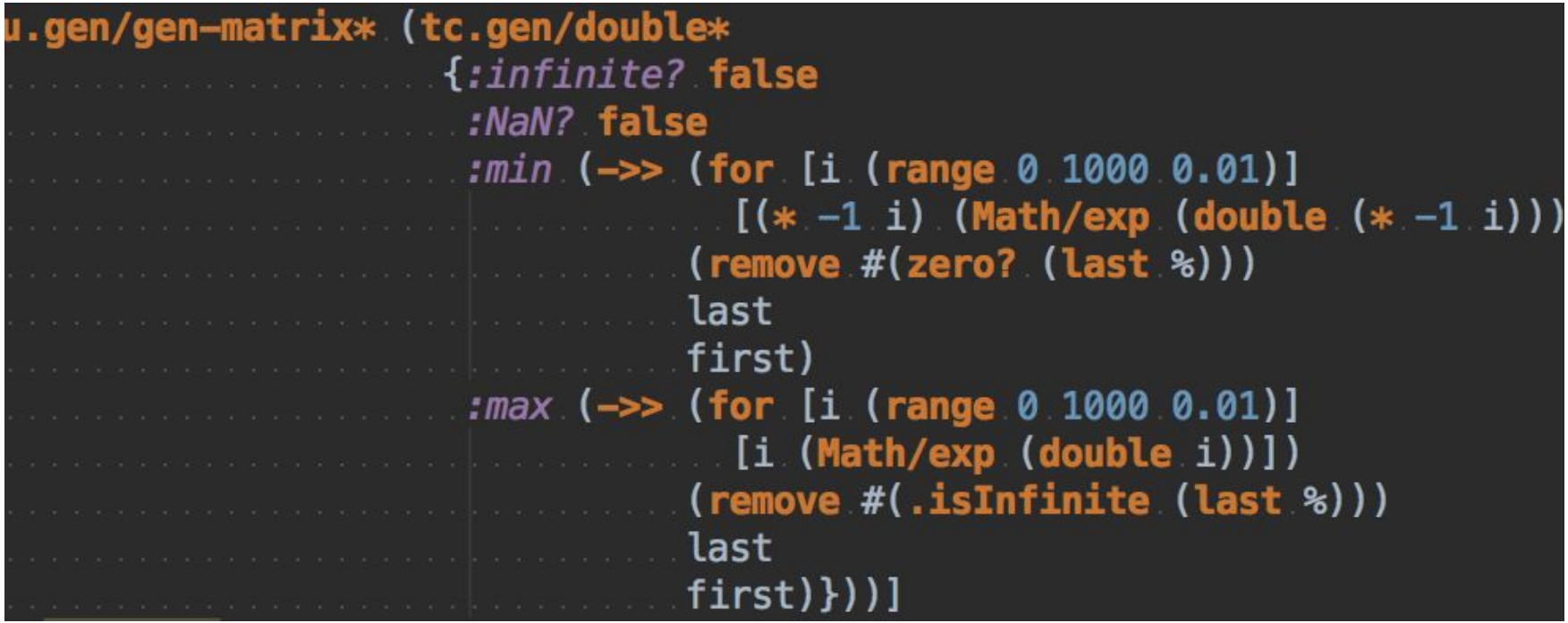


```
(def gen-square-matrix
  (gen-matrix* (tc.gen/bind
                 tc.gen/s-pos-int
                 (fn [n] (tc.gen/tuple (tc.gen/return n) (tc.gen/return n
                (tc.gen/double* {:infinite? false :NaN? false}))))
```

Matrix Generators of all shapes and sizes:

https://github.com/AlexanderMann/conj-2016/blob/master/test/conj_2016/util/generator.clj

Matrix Generators



```
u.gen/gen-matrix* (tc.gen/double*
                   {:infinite? false
                    :NaN? false
                    :min (->> (for [i (range 0 1000 0.01)]
                                [(* -1 i) (Math/exp (double (* -1 i)))])
                              (remove #(zero? (last %)))
                              last
                              first)
                    :max (->> (for [i (range 0 1000 0.01)]
                                [i (Math/exp (double i))])
                              (remove #(.isInfinite (last %)))
                              last
                              first)})))]
```

Matrix Generators of all shapes and sizes:

https://github.com/AlexanderMann/conj-2016/blob/master/test/conj_2016/util/matrix_test.clj

Fuzzy=

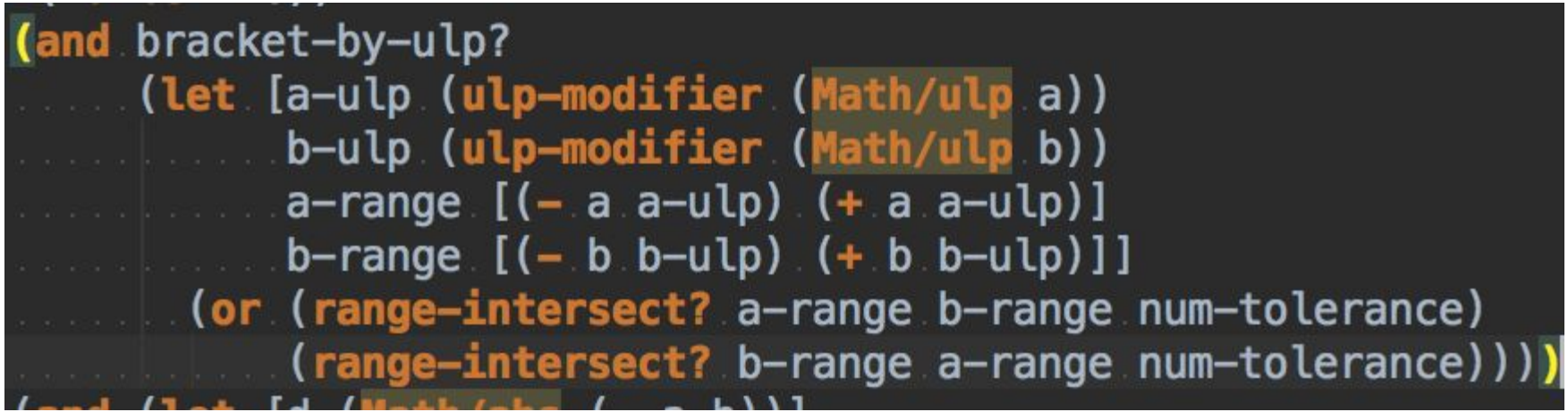
Necessary for comparing Python's math internals and number representation and json write/parse without losing your mind

Obvious implementations:

- Threshold
- Relative
- Common type mapping (`list == vector`, etc.)
- `Infinite == NaN == nil`

```
(defn non-numeric-number?
  "Returns true when nil, or i
  [obj]
  (when (or (nil? obj)
            (double? obj))
    (or (nil? obj)
        (.isNaN obj)
        (.isInfinite obj))))
```

Non obvious fuzzy=



```
(and bracket-by-ulp?
     (let [a-ulp (ulp-modifier (Math/ulp a))
           b-ulp (ulp-modifier (Math/ulp b))
           a-range [(- a a-ulp) (+ a a-ulp)]
           b-range [(- b b-ulp) (+ b b-ulp)]]
       (or (range-intersect? a-range b-range num-tolerance)
           (range-intersect? b-range a-range num-tolerance))))
```

Ulp range intersects (https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html#ulp(double)):

https://github.com/AlexanderMann/conj-2016/blob/master/src/conj_2016/util/comparator.clj

Clojure For ML

As a language

Having to opt *out* of immutability is a subtle, but huge win

S-forms in LISP are another subtle but massive pro

It's not Scheme, but the culture around naming and what the language allows are positive

REPL is, no surprise, a nice way to incrementally understand complex code paths

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}, \qquad q_{ii} = 0.$$

$$p_{j|i} = \begin{cases} \frac{\exp(-d(\mathbf{x}_i, \mathbf{x}_j)^2 / 2\sigma_i^2)}{\sum_{k \in \mathcal{N}_i} \exp(-d(\mathbf{x}_i, \mathbf{x}_k)^2 / 2\sigma_i^2)}, & \text{if } j \in \mathcal{N} \\ 0, & \text{otherwis} \end{cases}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}.$$

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4(F_{attr} + F_{rep}) = 4 \left(\sum_{j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j} \right.$$

Opting *into* mutability

Writing shallow code means you'll have potentially many helper functions (which when well named makes for healthy documentation, thanks Uncle Bob)

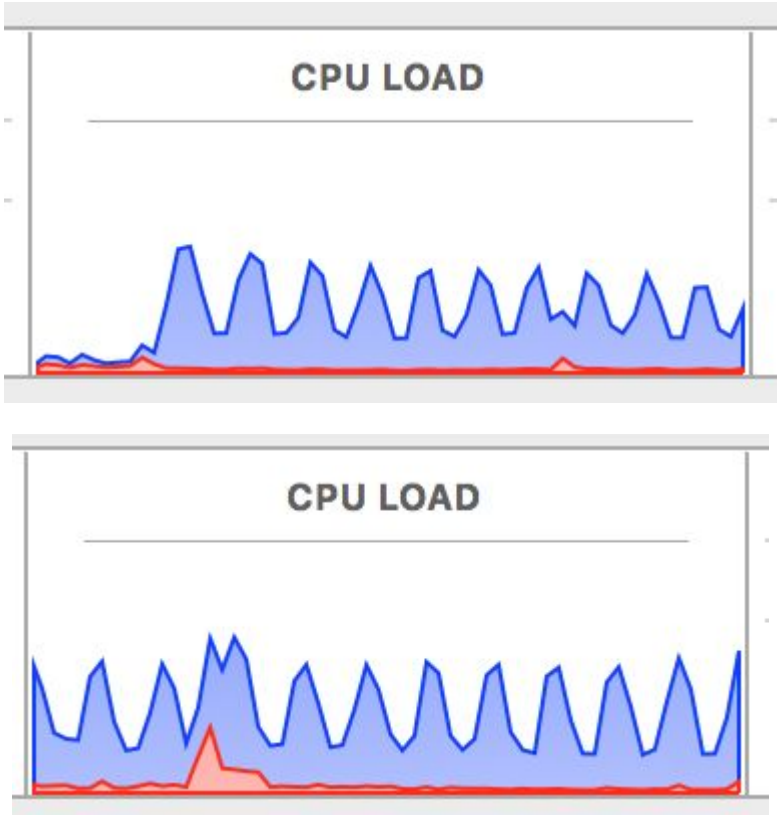
These helpers are great places to quickly speed up execution

The mystic power of !

```
(doto (m/clone num_n_n)
  u.m/normalize!
  (u.m/clamp! 1E-12)
  m/negate!
  (m/add! pvalues_n_n)
  ;157 ... PQ[:,i] * num[:,i]
  (m/mul! num_n_n)))]
```

```
(defn update!-gains
  [^Matrix gains_n_m ^Matrix g
  ;164 gains = (gains + 0.2) *
```

Biggest drawback?



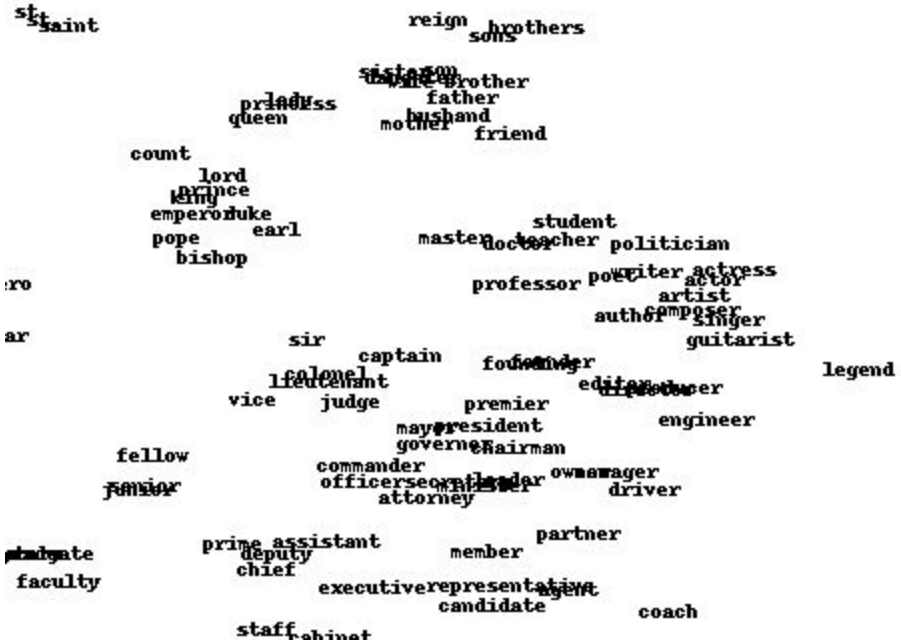
Speed (shocking right?)

Used https://github.com/ptaoussanis/tufte to get Clojure performant

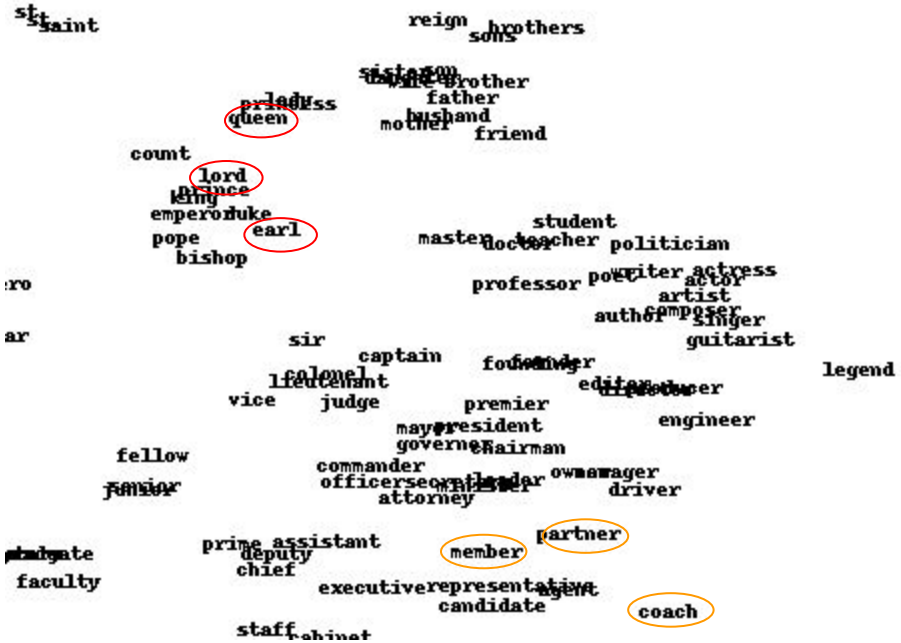
...buttttttttt the JVM is not cpp, which is what I was comparing alongside

Art/“Data-Vis”

http://www.cs.toronto.edu/~hinton/turian.png

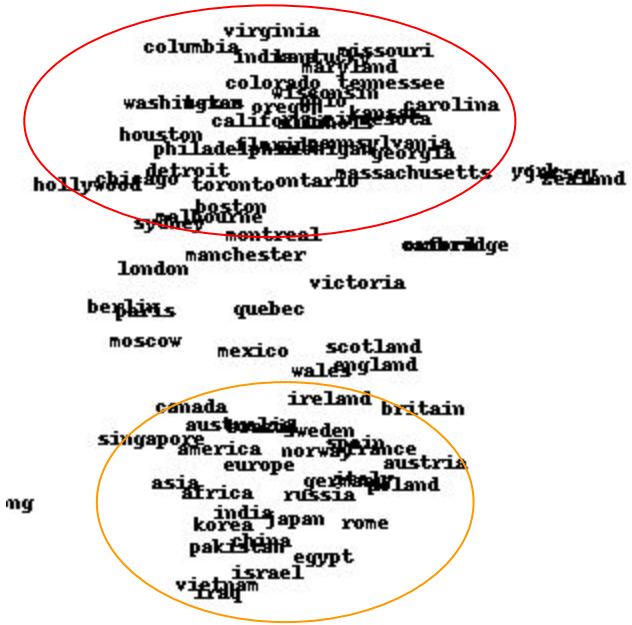


http://www.cs.toronto.edu/~hinton/turian.png

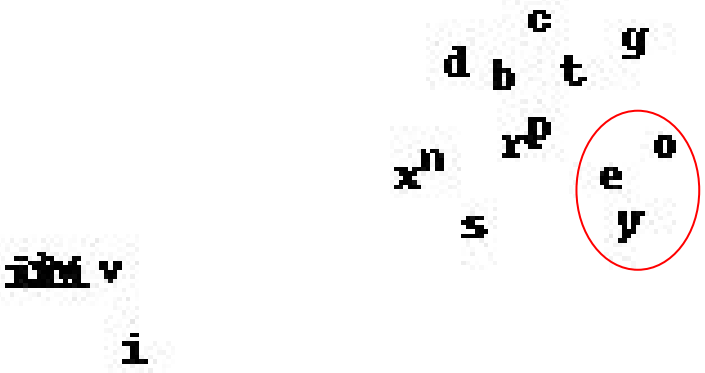


http://www.cs.toronto.edu/~hinton/turian.png

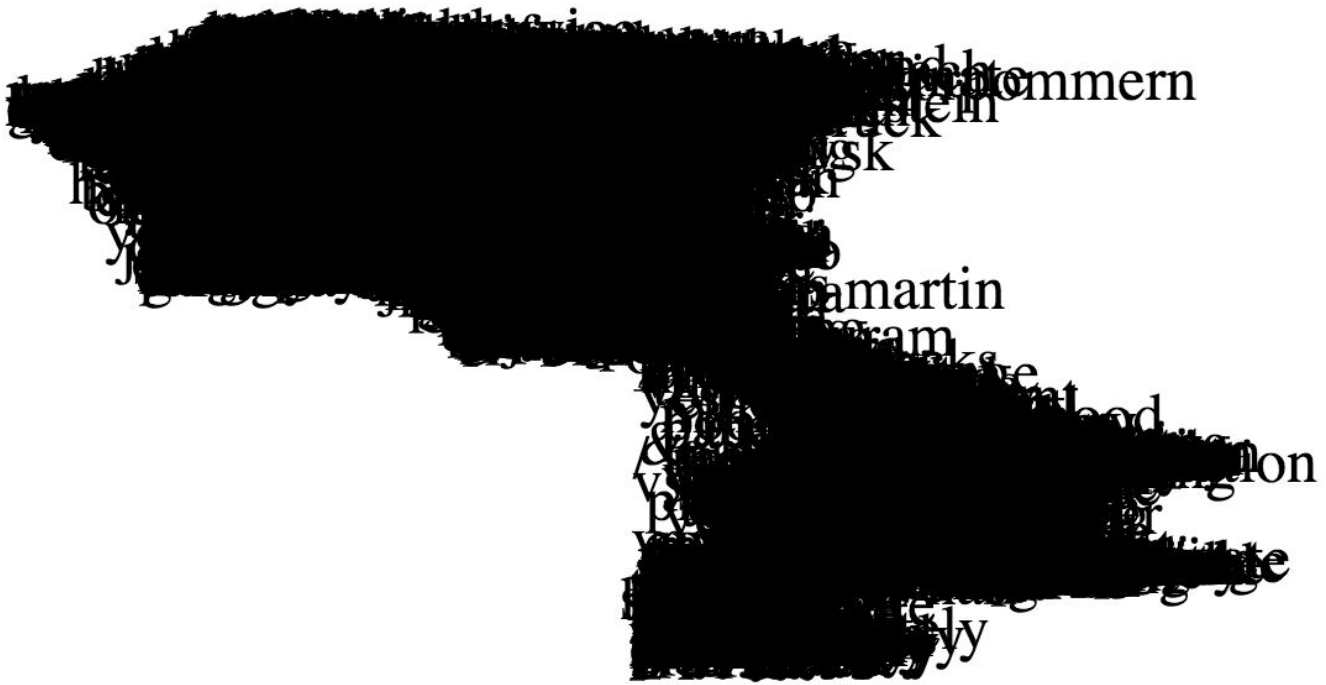
http://www.cs.toronto.edu/~hinton/turian.png



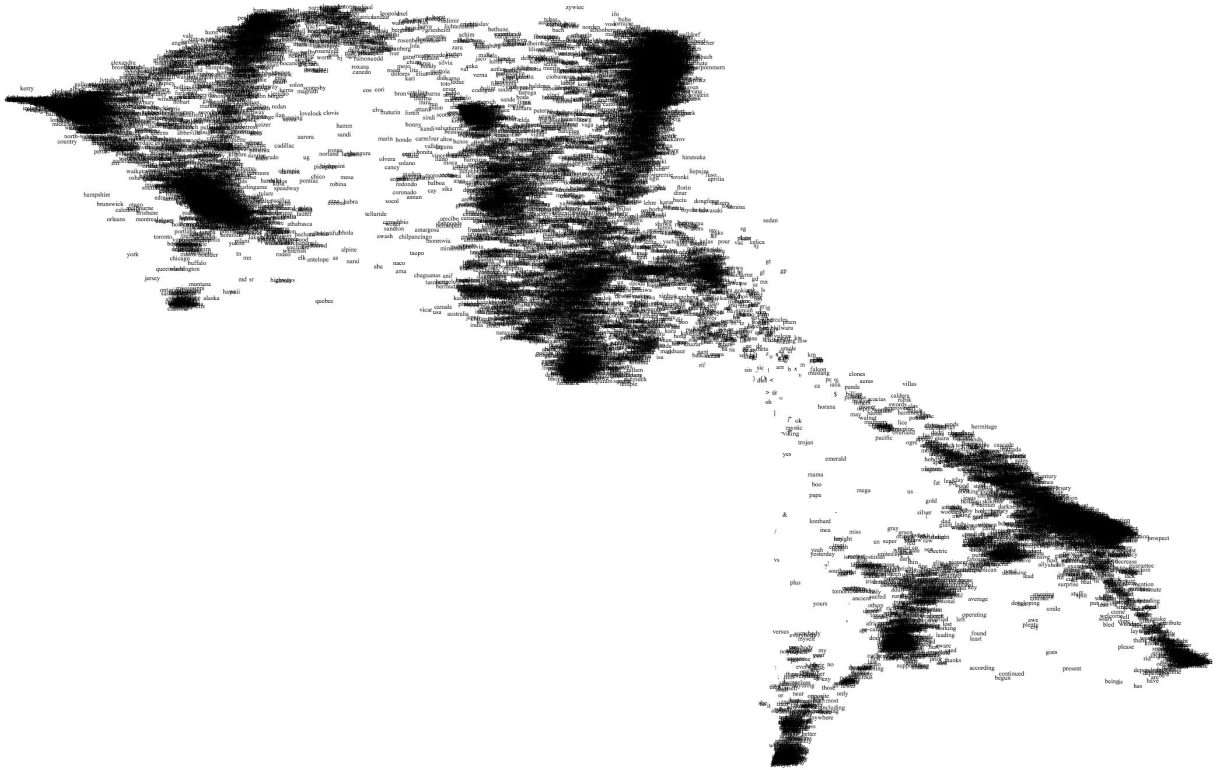
http://www.cs.toronto.edu/~hinton/turian.png



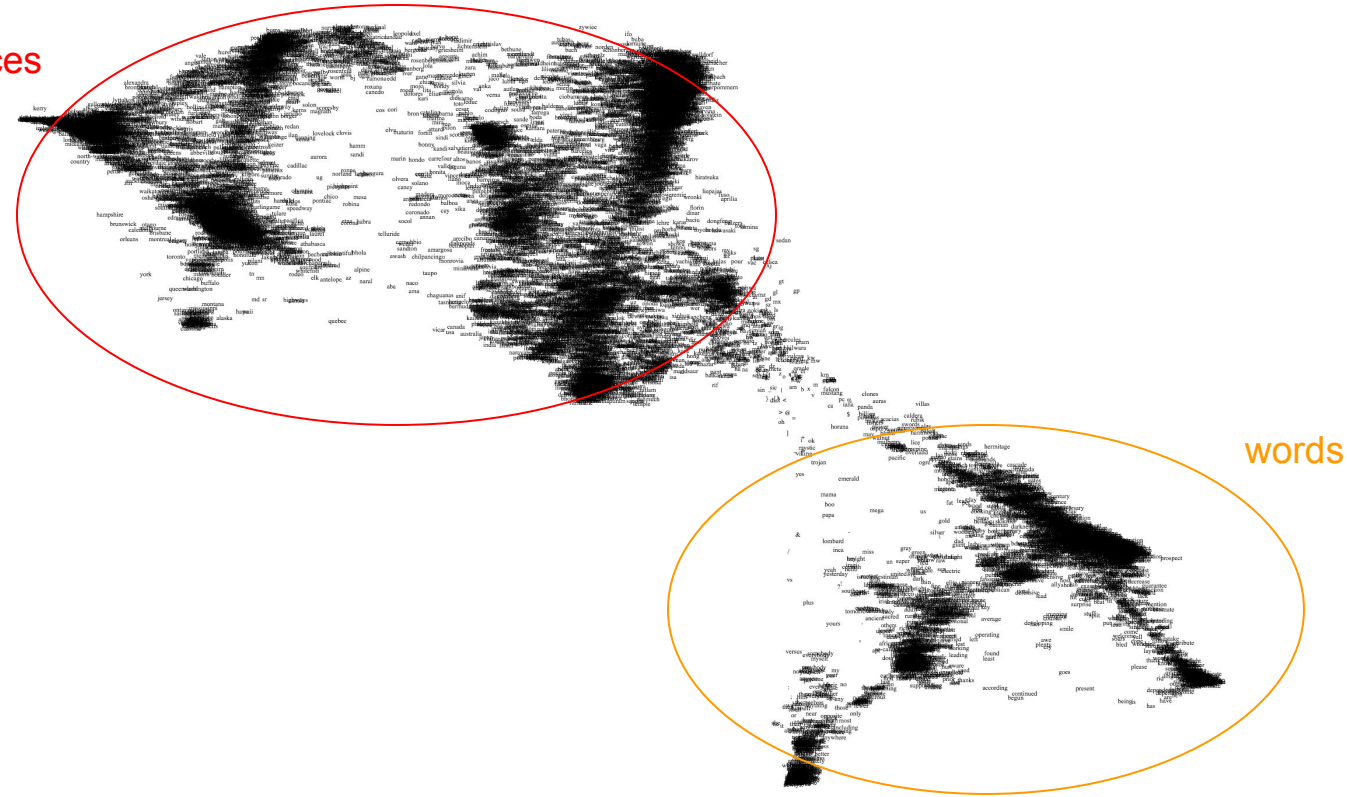
http://www.cs.toronto.edu/~hinton/turian.png



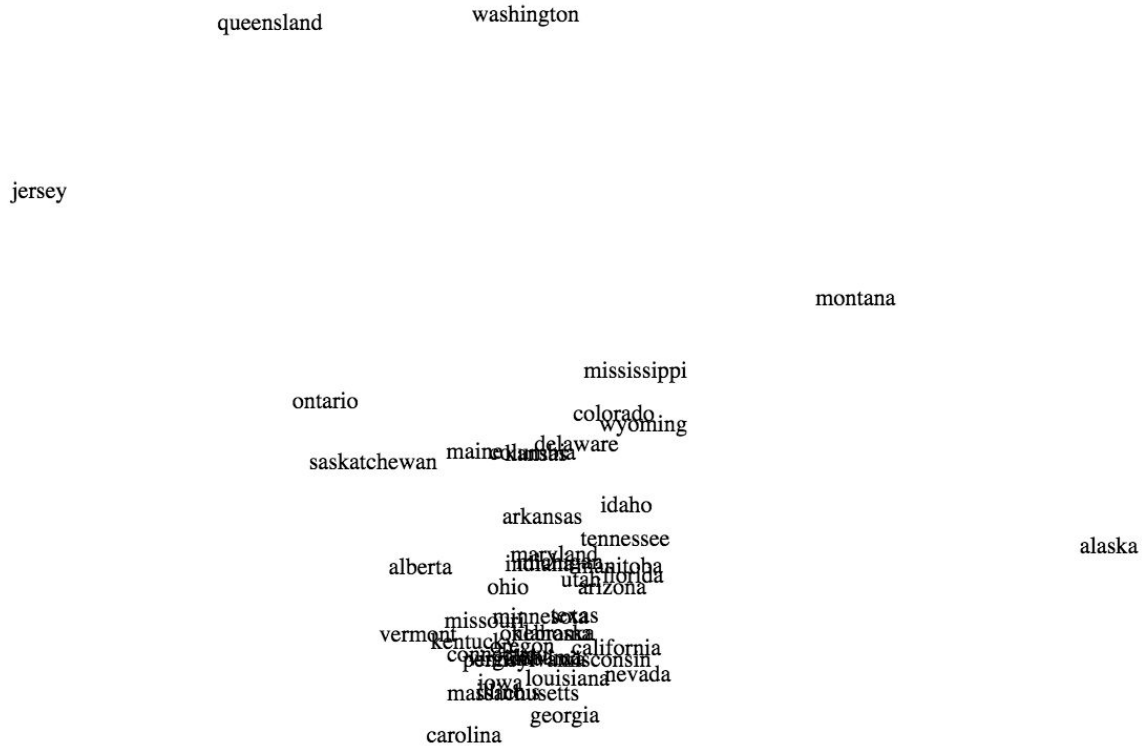
Personal rendering of Collobert and Weston Word Embeddings - Take 1



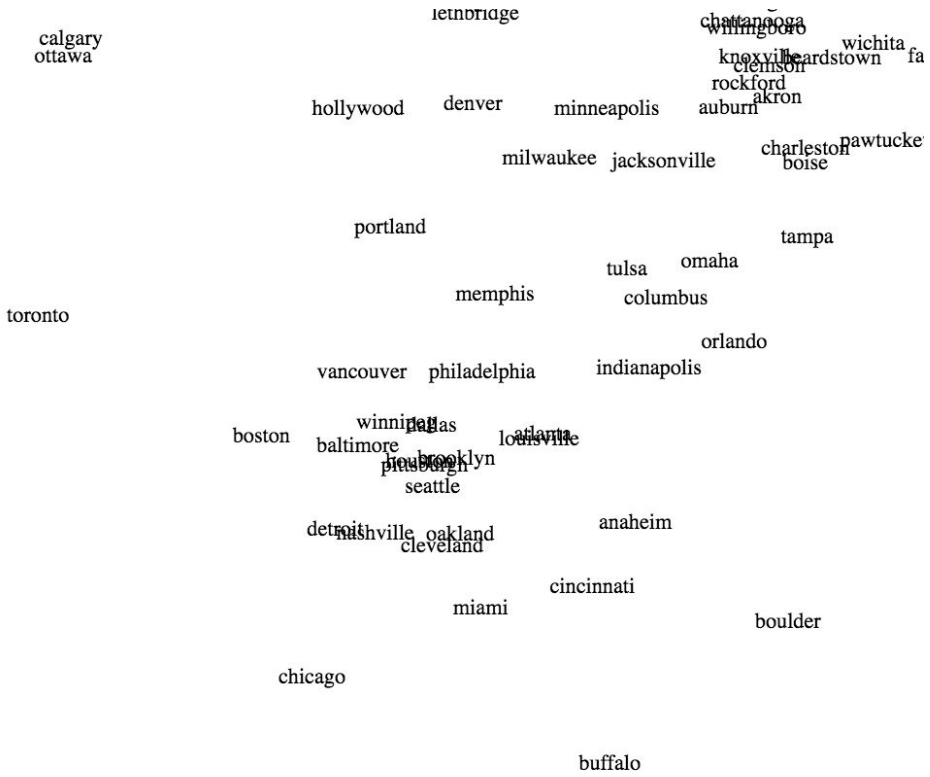
Personal rendering of Collobert and Weston Word Embeddings - Take 2



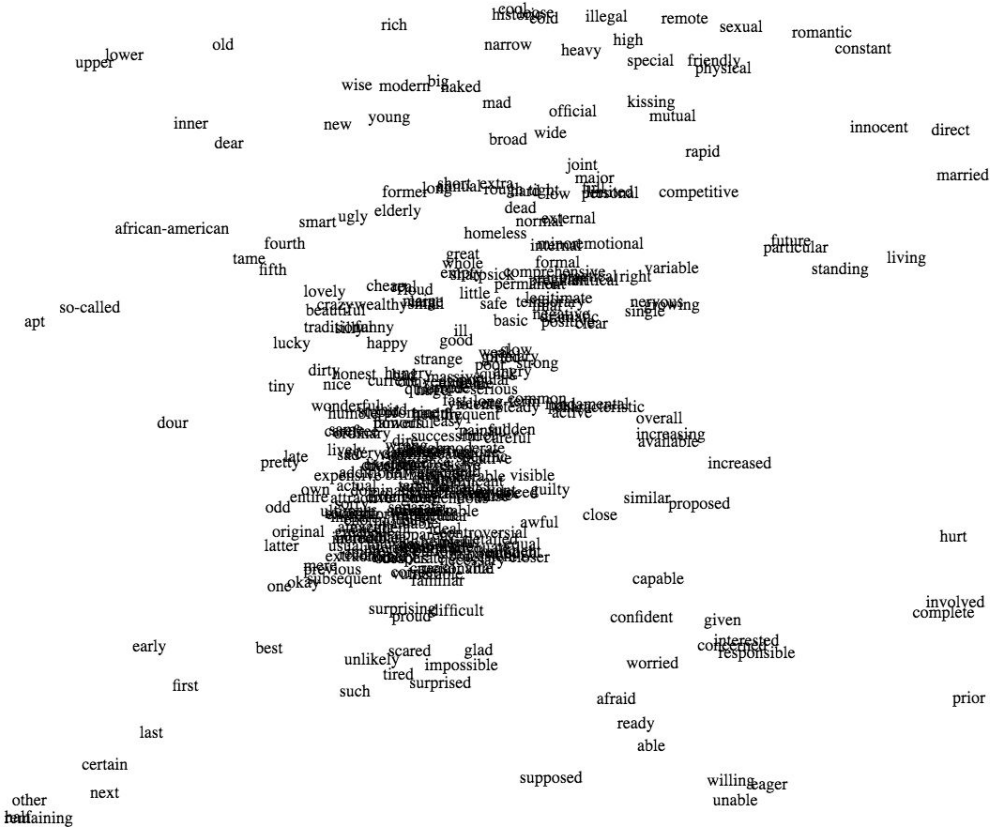
Personal rendering of Collobert and Weston Word Embeddings - Take 2



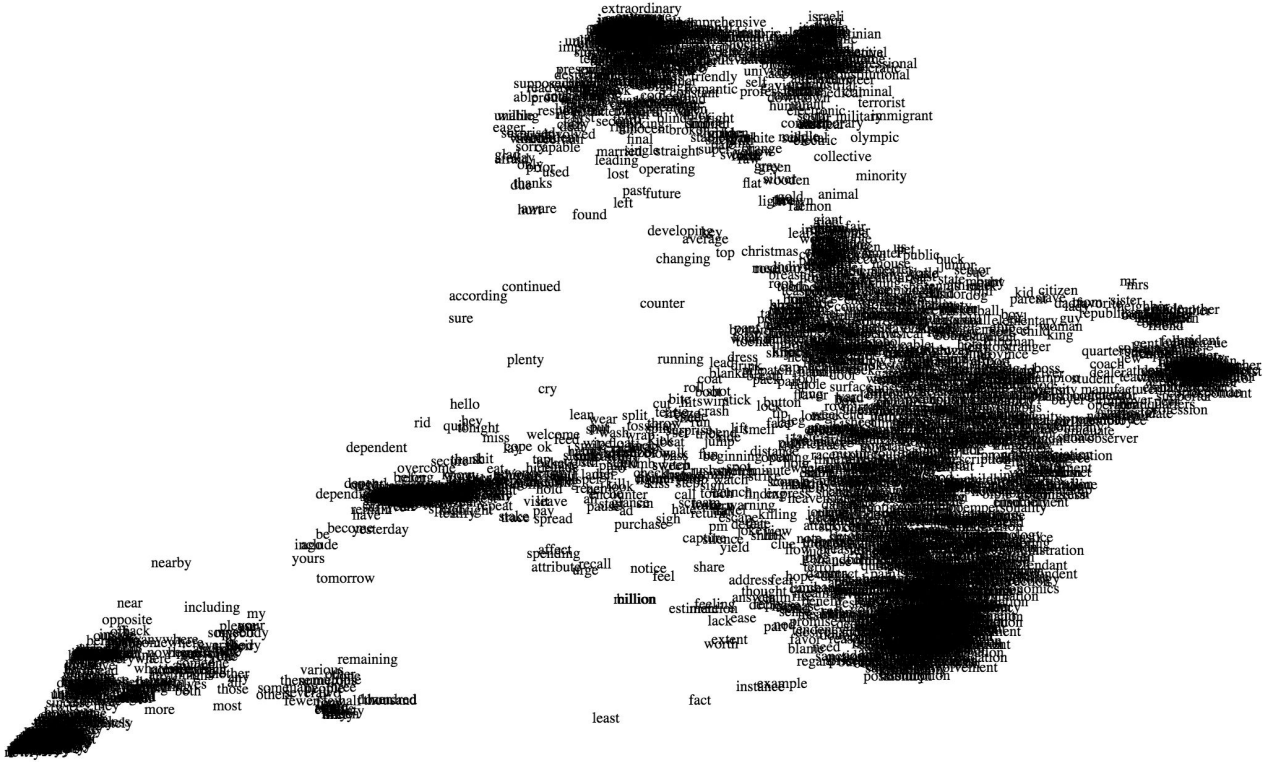
North America



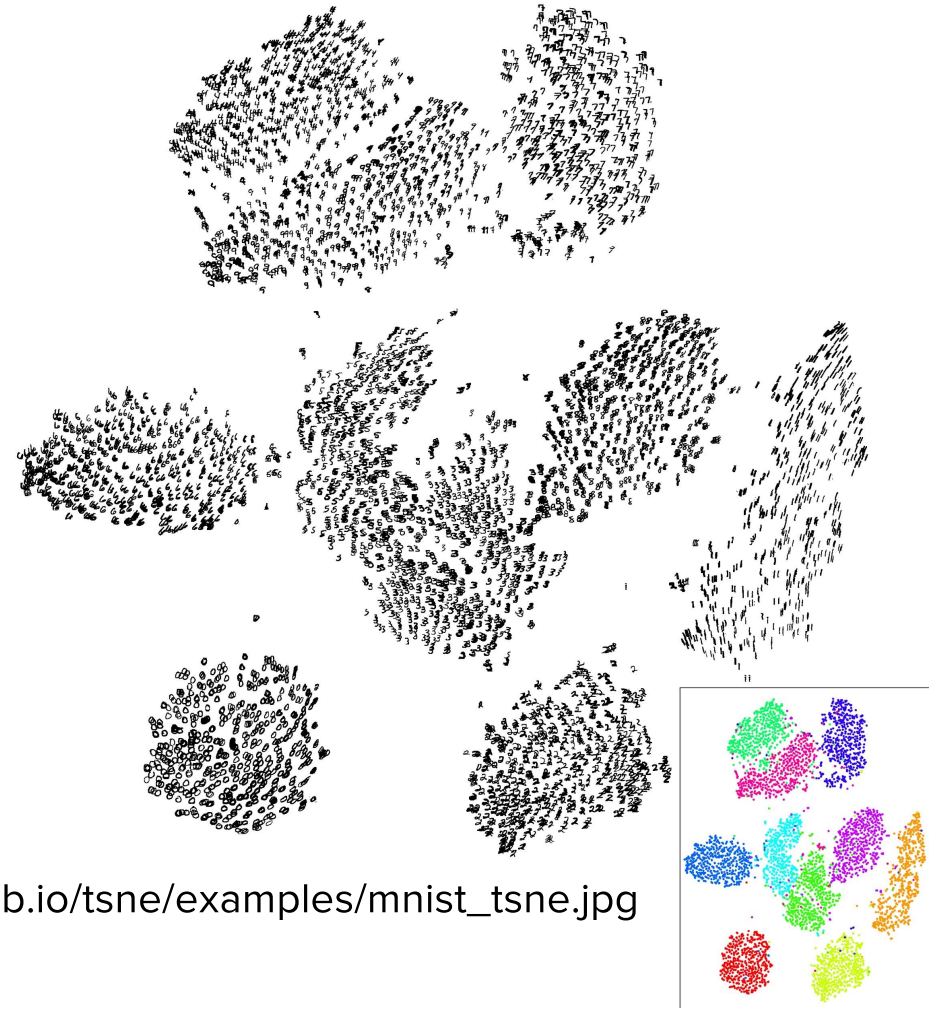
North American Capitals



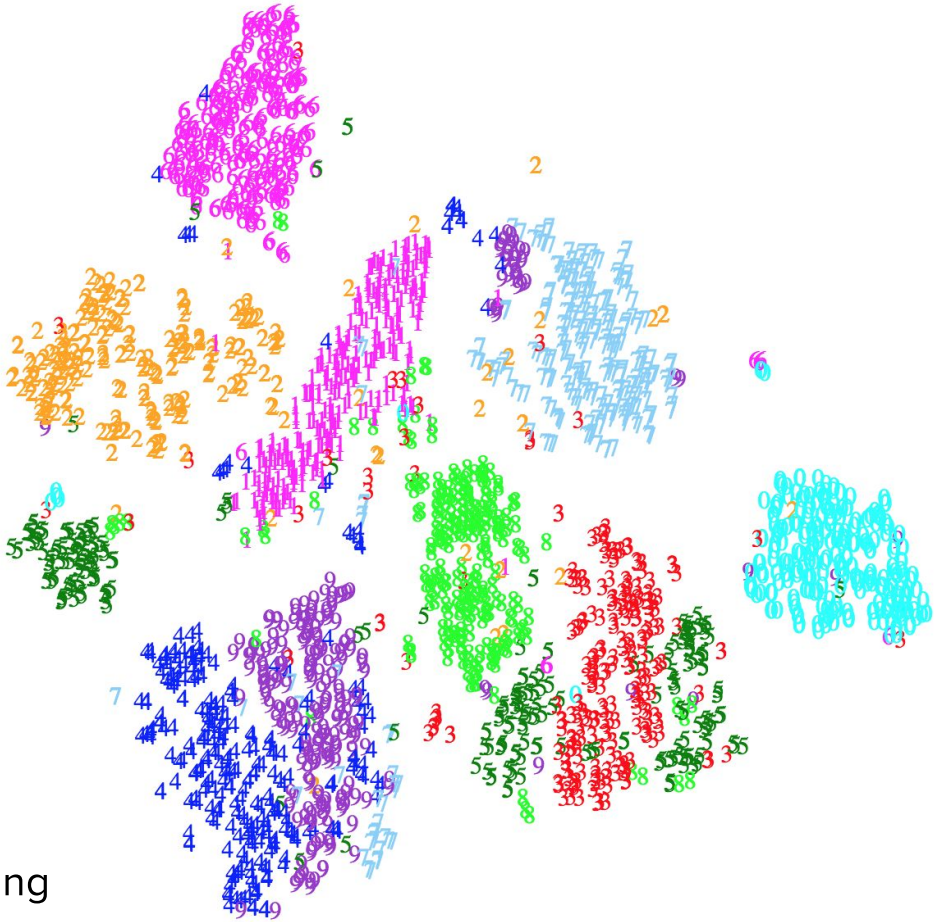
Adjectives (loosely)



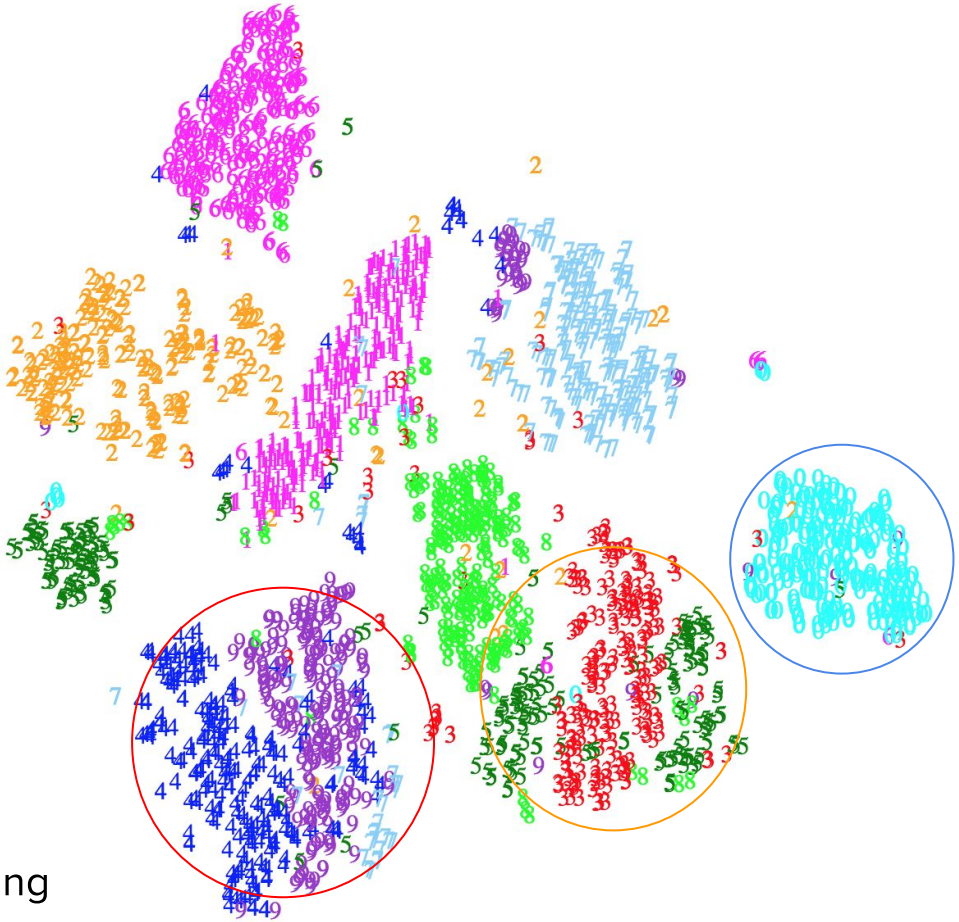
Rendering without places, just 3000 most popular english words



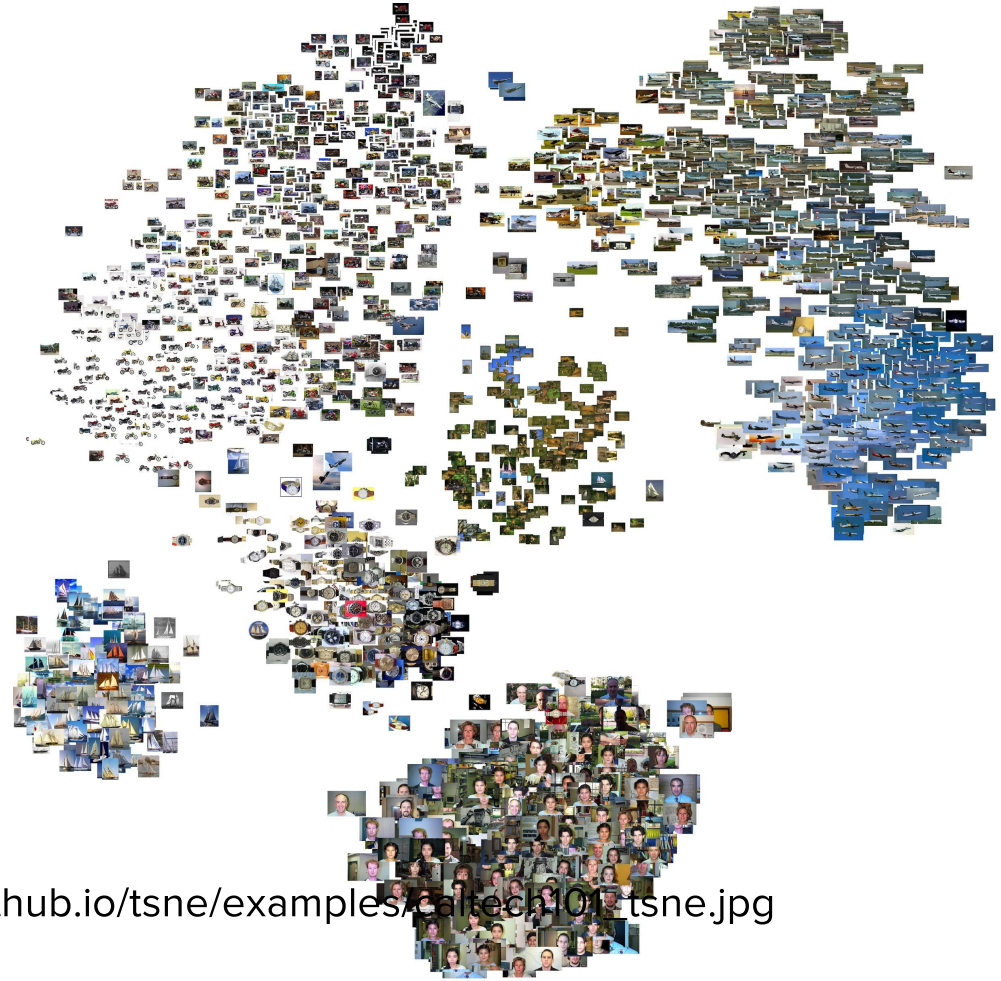
https://lvdmaaten.github.io/tsne/examples/mnist_tsne.jpg



Personal MNIST rendering



Personal MNIST rendering



https://lvdmaaten.github.io/tsne/examples/caltech101_tsne.jpg

https://lvdmaaten.github.io/tsne/examples/caltech101_tsne.jpg

https://lvdmaaten.github.io/tsne/examples/caltech101_tsne.jpg

https://lvdmaaten.github.io/tsne/examples/caltech101_tsne.jpg

Thanks to:

circleci Stitch RJMETRICS A Magento Company *Cursive*


Hinton


Laurens van der Maaten
lvdmaaten


Joseph Turian
turian


Matthew Bilyeu
bi1yeu


Peter Taoussanis
ptaoussanis


Collobert


Weston

mikera / core.matrix