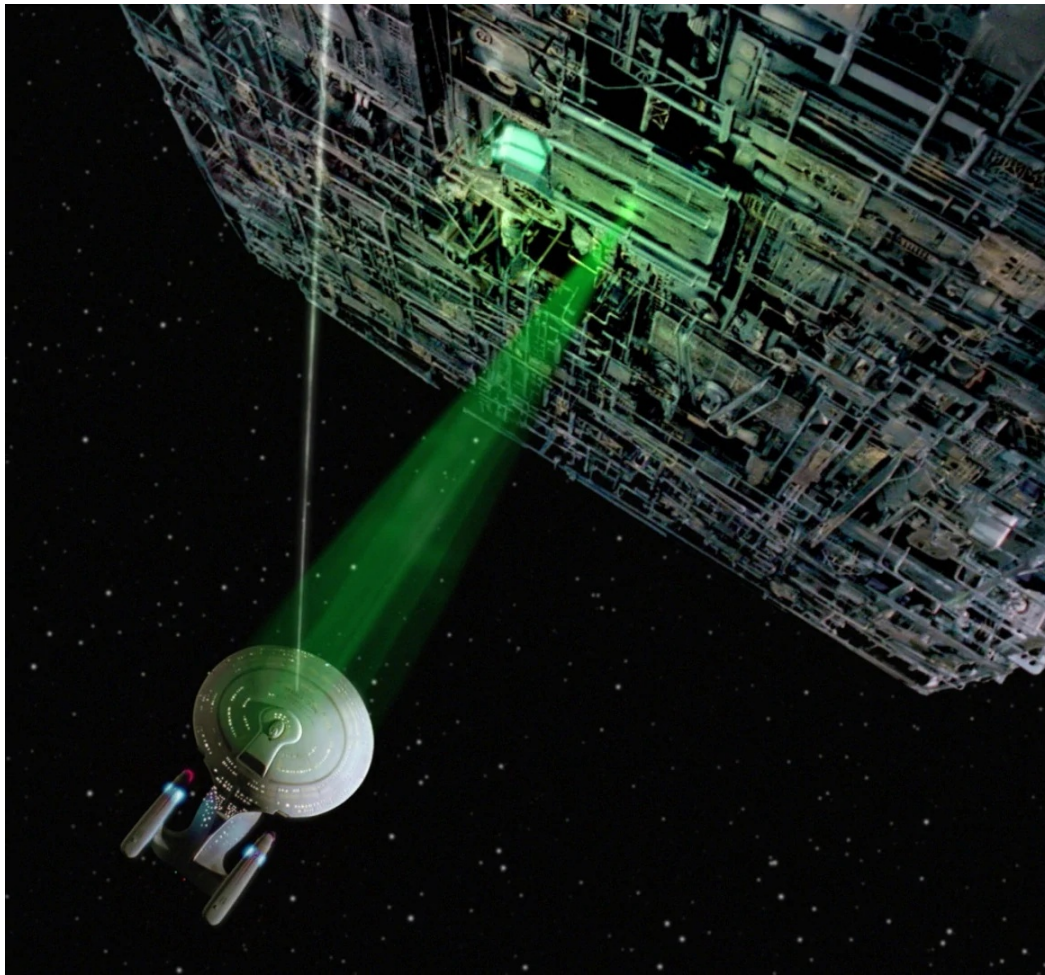
OCIO | Office of the Chief Information Officer

# Can the Data Speak? Reflections on Unsupervised Classification

## In Python with Accessible Examples



**Ashlin Richardson**
**Senior Data Scientist**
**Master of Science, Mathematics**
**Partnerships and Capacity Branch**
**Digital Platforms and Data Division (OCIO)**

# What is classification?

Classification: methods to **distinguish groups in data**

- **apples vs. oranges**
- **sheep vs. goats,**
- **droids we're looking for vs.**

  **not the droids we're looking for, etc.**

# Types of classification?

| | Parametric | Nonparametric |
|---|---|---|
| Supervised | | |
| Unsupervised | | |

**Supervised:** training data are used

**Unsupervised:** training data **not used**

**Parametric:** relatively **strong assumptions** used about the data: e.g. “normal distribution”

**Nonparametric: assumptions weakened:** models which scale as: data become larger or more complex

# Types of classification?

| | Parametric | Nonparametric |
|---|---|---|
| Supervised | | |
| Unsupervised | | X |

**Supervised:** training data are used

**Unsupervised:** training data are **not used**

**Parametric:** fairly **strong assumptions** made: e.g. “normal distribution”

**Nonparametric: assumptions weakened:** models which scale as: data become larger or more complex

# Why unsupervised?

### 4.1.2 *Unsupervised learning*

This branch of machine learning consists of finding interesting transformations of the input data without the help of any targets, for the purposes of data visualization, data compression, or data denoising, or to better understand the correlations present in the data at hand. Unsupervised learning is the bread and butter of data analytics, and it's often a necessary step in better understanding a dataset before attempting to solve a supervised-learning problem. *Dimensionality reduction* and *clustering* are well-known categories of unsupervised learning.

From: "Deep Learning with R" by F. Chollet and J.J. Allaire

# Why Non-parametric?

## Why Non-parametric?

**Relationships** in the data matter!

## Why Non-parametric?

**Relationships** in the data matter!

- Data have **shape**

## Why Non-parametric?

**Relationships** in the data matter!

- Data have **shape**
- The **shape matters!**

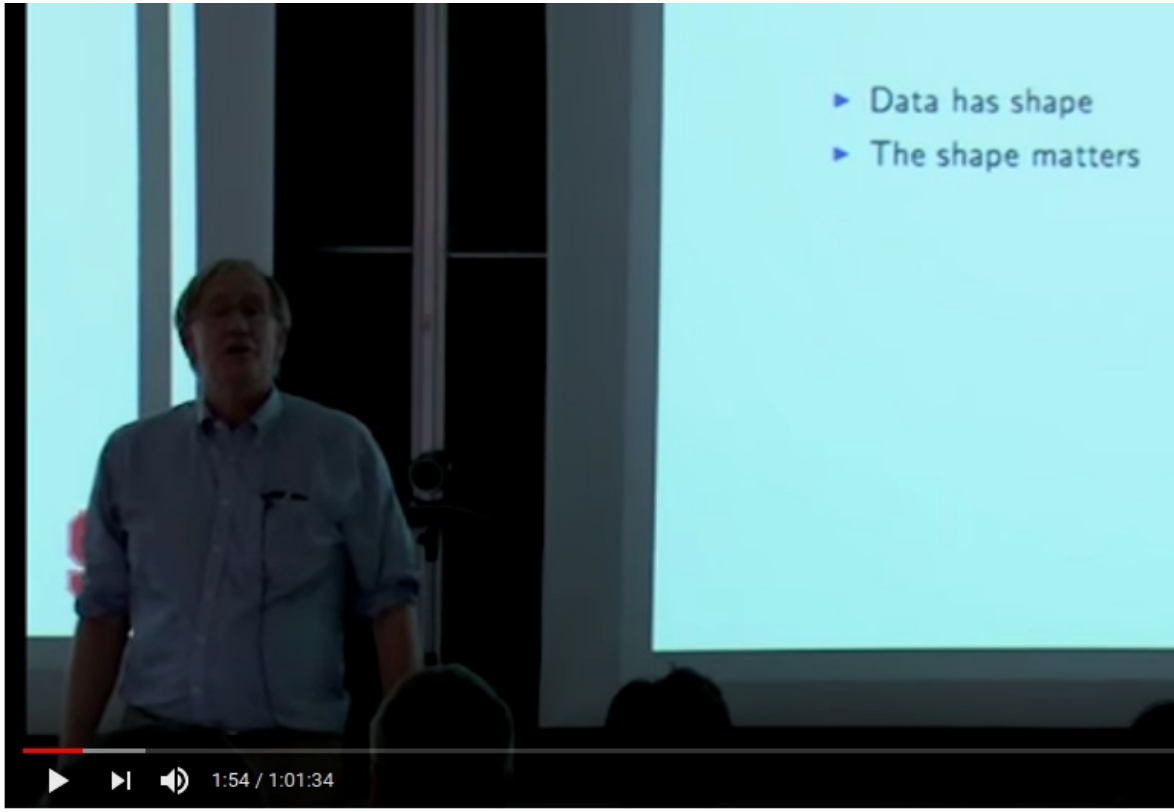
# Why Non-parametric?

**Relationships** in the data matter!

- Data have **shape**
- The **shape matters!**

**Says Gunnar Carlson,** Stanford Math prof and CEO of:

**https://en.wikipedia.org/wiki/Ayasdi**




The Shape of Data - Graduate School of Mathematical Sciences - The University of Tokyo

3,994 views • Aug 5, 2015

AYASDI

Ayasdi
1.61K subscribers

Gunnar Carlson speaks about the Shape of Data at the Graduate School of Mathematical Sciences at The University of Tokyo

https://www.youtube.com/watch?v=iOxLgbnl1u

## Goals for Python implementation:

Devise **procedure to: distinguish groups in data** (classify):

### Accessible

- No code library
- No neural network
- “Simple” programming
- Little user intervention
- Under 50 code statements
- No math except arithmetic and sorting

### Adaptable

- Transferrable to several kinds of data
- HPC technique(s) to help scale: use **parallelism**
- **1) Non-parametric:** model scales with new data: **flexible to data shape**
- **2) Unsupervised:** model doesn’t use training data: groups not decided by us

# Why Non-parametric and Unsupervised?

Flexible to data shape, avoiding confirmation-bias:

**Less violence to data!**

**Do algorithms do violence to data?**

**Potentially, yes.** Caution: choosing algorithms can be like choosing between the Death Star and The Borg Cube.

They're shaped differently:

both "**Weapons of Math Destruction" (WMD)!**

**Intent:** better info products and decision-making Supports, via less-biased "Artificial Intelligence"

# How to accomplish this?

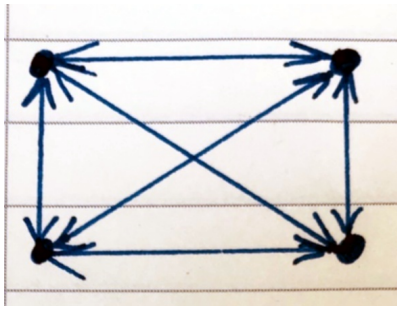
**Recursive hill-climbing on a density estimate! KGC 2010**

Data-driven labelling, without a lot of restrictive assumptions

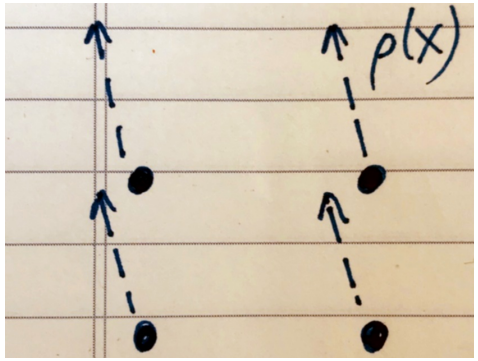
How to predict the future? Let's see if we can "predict the present", first

# Algorithm: KGC 2010

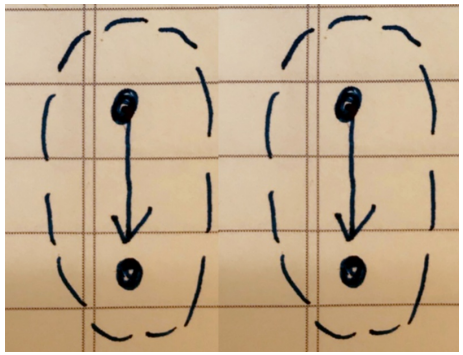
## 1. Calculate “distance matrix”



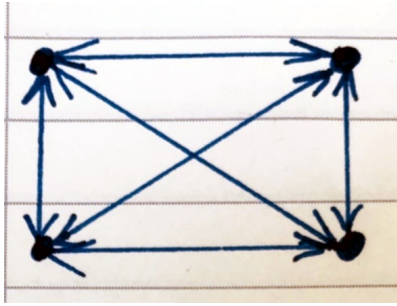
## 2. Estimate density at each point




## 3. Perform recursive hill-climbing



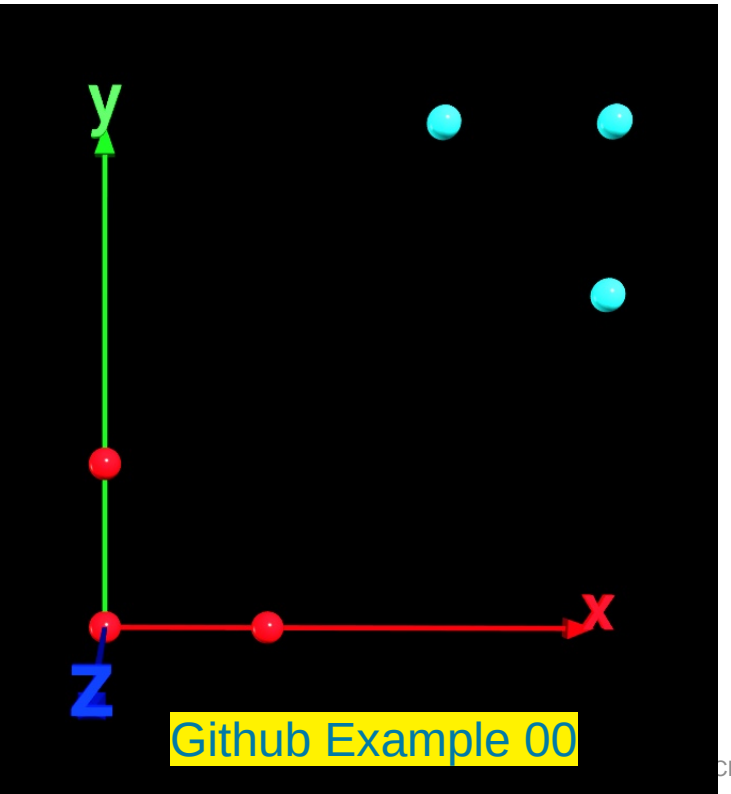
# Algorithm: KGC 2010



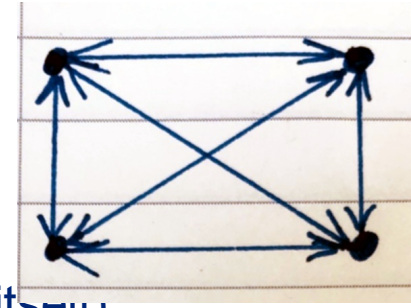
## 1. Calculate “distance matrix”

- One row per data point:

data = [[0.0, 1.0, 0.0], [0.0, 0.0, 0.0], [1.0, 0.0, 0.0], [3.0, 3.0, 0.0], [3.0, 2.0, 0.0], [2.0, 3.0, 0.0]]



# Algorithm: KGC 2010

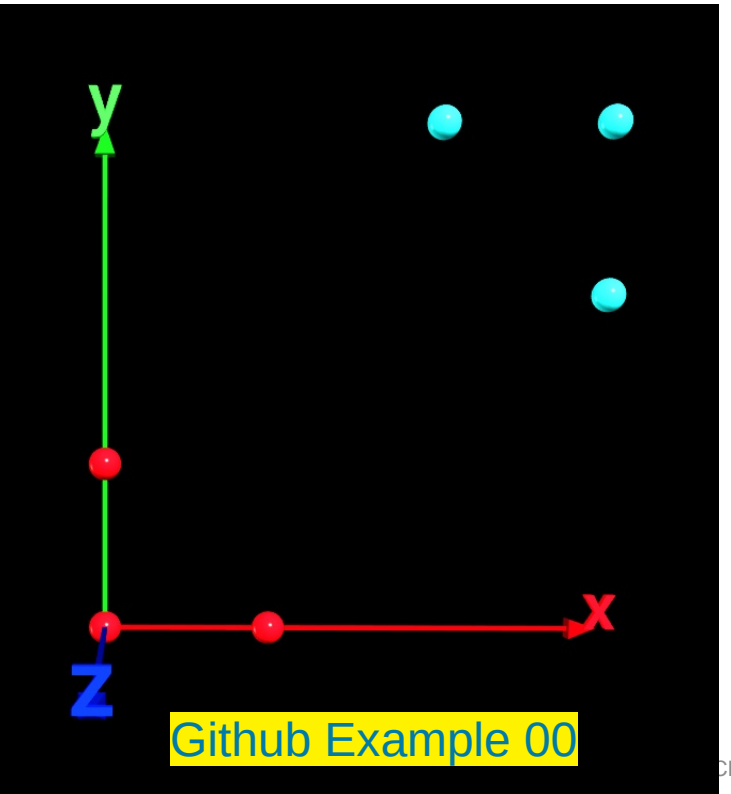


## 1. Calculate “distance matrix”: d(x,y) = ||x-y||

- One row per datapoint: distance from, a point to every point (incl. itself)

E.g., in the first row for point **[0.0, 1.0, 0.0],** d([0.0, 1.0, 0.0], [0.0, 1.0, 0.0]) = 0 , d([0.0, 1.0, 0.0], [0.0, 0.0, 0.0]) = 1, etc.

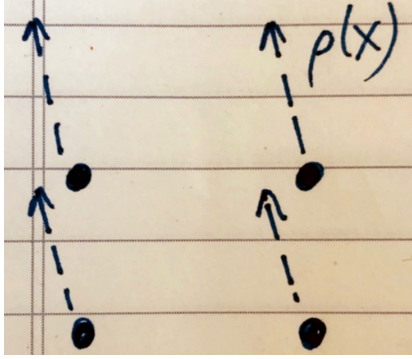
- **Sort each row** so that: “nearest” neighbour is first, second-nearest neighbour is second, and so on.. Below, distance matrix in normal text, Sorted matrix in **bold!**




record distance and neighbour idx as tuple [0.0, 0]:

[[0.0, 0], [1.0, 1], [2.0, 2], [13.0, 3], [10.0, 4], [8.0, 5]]
**[[0.0, 0], [1.0, 1], [2.0, 2], [8.0, 5], [10.0, 4], [13.0, 3]]**
[[1.0, 0], [0.0, 1], [1.0, 2], [18.0, 3], [13.0, 4], [13.0, 5]]
**[[0.0, 1], [1.0, 0], [1.0, 2], [13.0, 4], [13.0, 5], [18.0, 3]]**
[[10.0, 0], [13.0, 1], [8.0, 2], [1.0, 3], [0.0, 4], [2.0, 5]]
**[[2.0, 0], [1.0, 1], [0.0, 2], [13.0, 3], [8.0, 4], [10.0, 5]]**
[[0.0, 4], [1.0, 3], [2.0, 5], [8.0, 2], [10.0, 0], [13.0, 1]]
**[[0.0, 2], [1.0, 1], [2.0, 0], [8.0, 4], [10.0, 5], [13.0, 3]]**
[[13.0, 0], [18.0, 1], [13.0, 2], [0.0, 3], [1.0, 4], [1.0, 5]]
**[[0.0, 3], [1.0, 4], [1.0, 5], [13.0, 0], [13.0, 2], [18.0, 1]]**
[[8.0, 0], [13.0, 1], [10.0, 2], [1.0, 3], [2.0, 4], [0.0, 5]]
**[[0.0, 5], [1.0, 3], [2.0, 4], [8.0, 0], [10.0, 2], [13.0, 1]]**

# Algorithm: KGC 2010

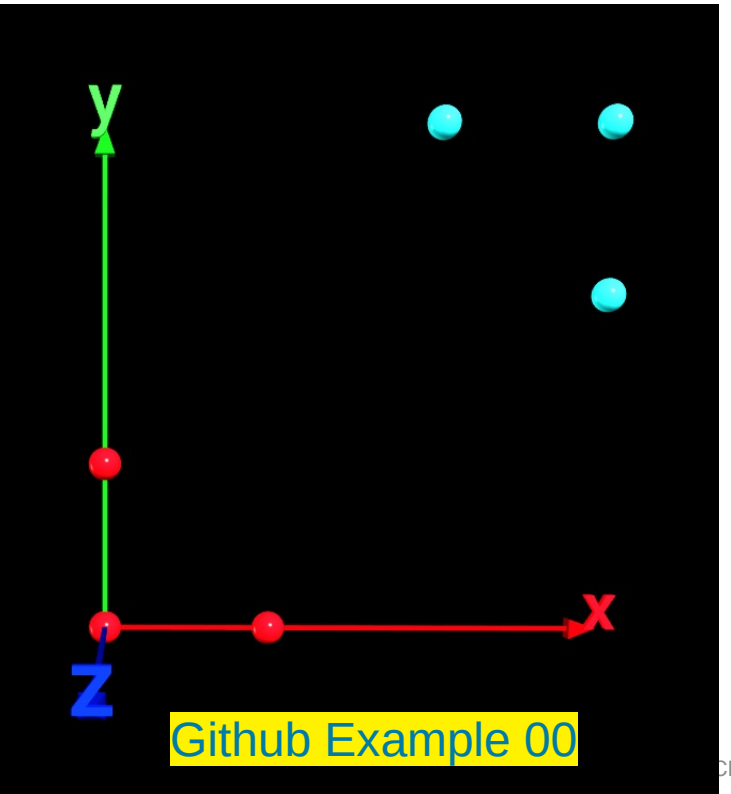



**2. Estimate "density" at each point:**

**A function that gets larger, if the K-nearest neighbours**

**To the point are closer!**

Rho = [0.33.., 0.5, 0.33.., 0.5, 0.33.., 0.33..]




**rho(x) = 1 / sum_y(d(x,y))**
Where y is a K-nearest neighbour of x

Default: K = ceil(sqrt(6)) = 3

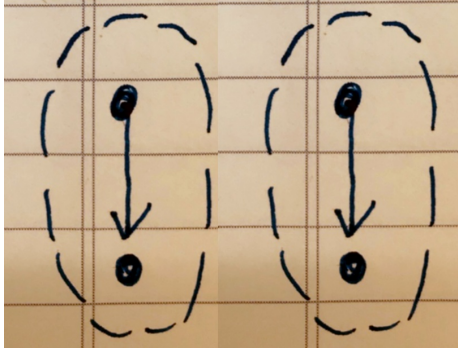
For the sorted distance matrix row 0:
**[[0.0, 0], [1.0, 1], [2.0, 2],..]**

1 / sum(d) = 1 / sum([0, 1, 2]) = 1 / 3 = .33

**[[0.0, 1], [1.0, 0], [1.0, 2],..]**
1/ sum(d) = 1 / sum([0, 1, 1]) = 1/2 = .5

And so on for the third, fourth point Etc..

# Algorithm: KGC 2010



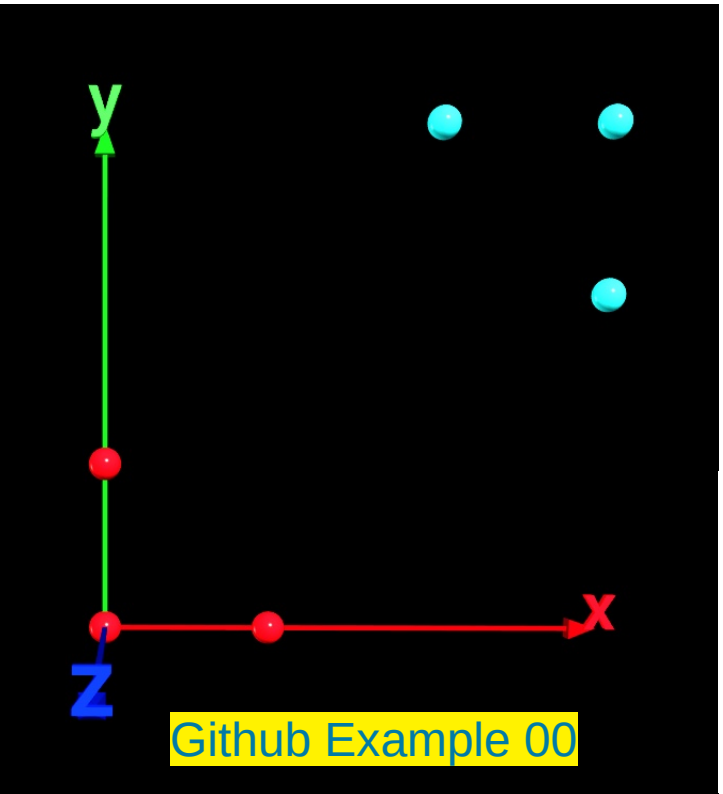
## 3. Perform recursive hill-climbing

A function that calls itself:

Three cases:

**1) already labelled:** return that label

**2) at a hill-top:** create a new label and return it




Github Example 00

**2) not at a hill-top:** return the label from the "highest density" neighbour (recurse/iterate there)

| reference(→) predicted(↓), | 0 | 1 |
|---|---|---|
| 0 | 3 | 0 |
| 1 | 0 | 3 |

climb i= 0
climb i= 1 from 0
  new label 0
climb i= 1
climb i= 2
climb i= 1 from 2
climb i= 3
  new label 1
climb i= 4
climb i= 3 from 4
climb i= 5
climb i= 3 from 5
--> label [0, 0, 0, 1, 1, 1]

# 1. Python: Calculate Distance Matrix

This function **calculates a row Of the "sorted distance matrix"**

**i. Grab a point**

**ii. calculate distances to it!**
With respect to all other points

**iii. Sort the list: [(distance, index of neighbour)]**
on distance

Order: by increasing distance..

```
# calculate one sorted row of the distance matrix
def dmat_row(i):
    global n, points

    row = []  # output: distance matrix row
    pi = points[i]  # point i data

    for j in range(0, n):
        d = 0.  # distance from point i to point j
        pj = points[j]  # point j data
        for k in range(0, n_d):
            d += math.pow(pi[k] - pj[k], 2)
        row.append([d, j])

    # sort on distance: lowest to highest
    row.sort(key=lambda x: x[0])
    return row

# calculate distance matrix in parallel
print("calculating distance matrix")
dmat_rows = parfor(dmat_row, indices)
```

## 2. Python: Estimate Density at Each Point

**i. Set density values (rho) To Zero!**

**ii. grab a Row from Distance-matrix**

**iii. Add up distances to k-nearest Neighbours!**

**iv. Divide by summed distance:**

```
def rho(): # density estimation
    global data, idx, knn_k, dmat, rho
    rho = []
    for i in idx:  # for each point
        row, r = dmat[i], 0.  # distance matrix row, dens. est.
        for j in range(0, knn_k):
            r += row[j][0] # add dist. to j-th nearest neighbour
        try: rho.append(1. / r)
        except Exception: pass
```

Smaller average distance --> higher density!

Larger average distance --> lower density!

# 3. Python: Recursive Hill-Climbing

**i. Check if point already labelled:** if so, return

**ii.** otherwise, **list the K-nearest Neighbours** of this point, plus indices: [(rho[j], j)]
So we know what points the Distances are for..

**iii. sort the k-nearest Neighbours list** by density

**iv. If density "here" is higher** than at the highest-density neighbour, Declare a "hill top" (new label)

**v. otherwise, climb to the highest-density neighbour** and grab the label from there! **"Recursively"**

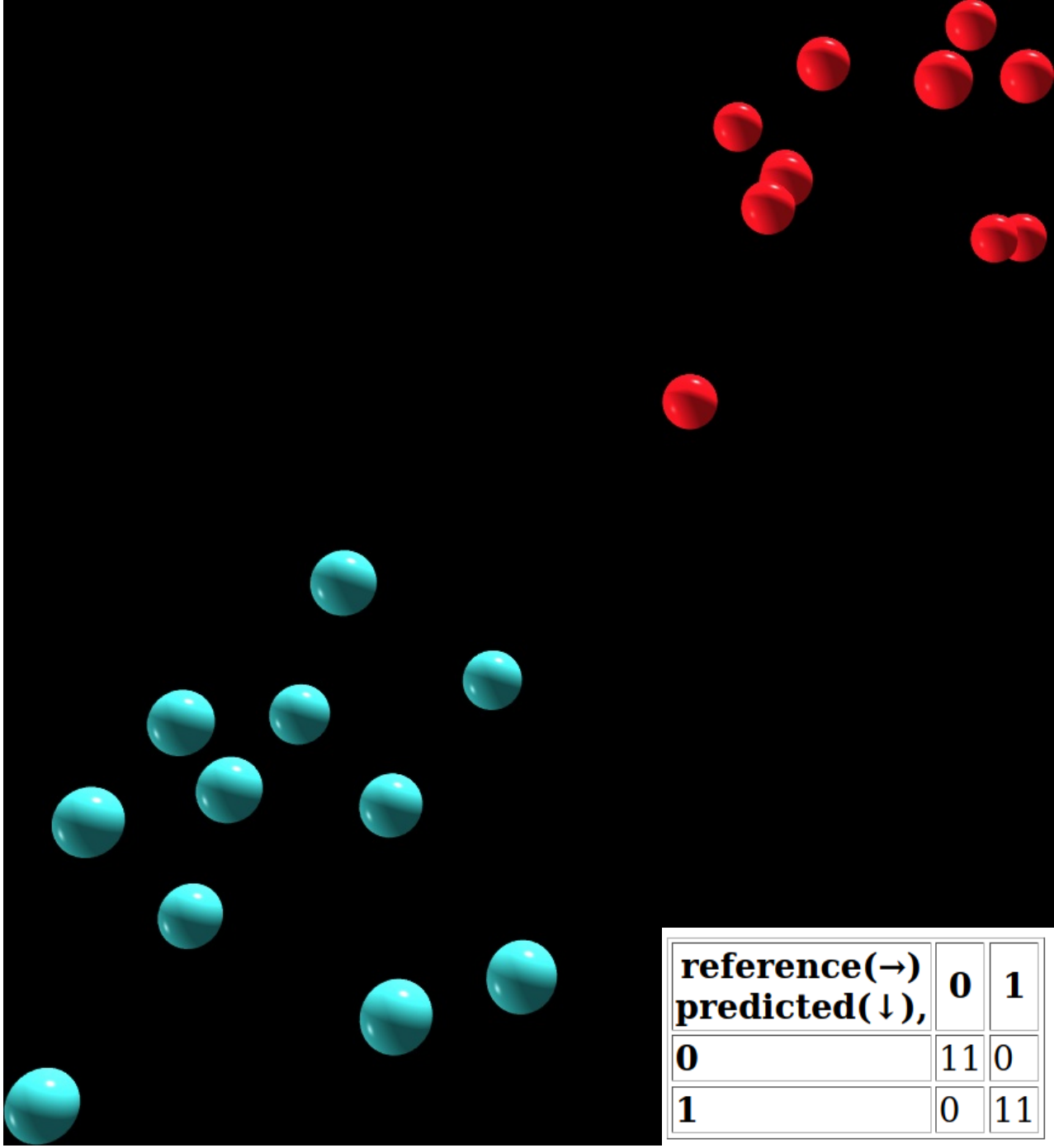
```
def labelme(i):
    global rho, label, knn_k, dmat_rows, next_label

    if label[i] >= 0:
        return label[i]  # base case: already labelled
    else:
        dmat_row = dmat_rows[i]

        # [rho[j], j]: density estimate and index j:
        #   for k-nearest neighbour j
        rho_knn = [[rho[j], j] for j in
                   [dmat_row[k][1] for k in range(1, knn_k)]]

        rho_knn.sort(key=lambda x: -x[0])  # sort by density

        # are we at hill-top? If so return new label
        if rho[i] > rho_knn[0][0]:
            label[i] = next_label
            next_label += 1  # create new label
        else: # If not, keep climbing
            label[i] = labelme(rho_knn[0][1])

        return label[i]  # return label
```

dmat_row[k] was a tuple [d, j]
So dmat_row[k][1] is the index j..

..Want to go over **indices of all of my neighbours that "aren't me":** hence the double List-comprehension!

# Example 01: bimodal

**Output labels: {0, 1} coloured by going around the colour wheel**



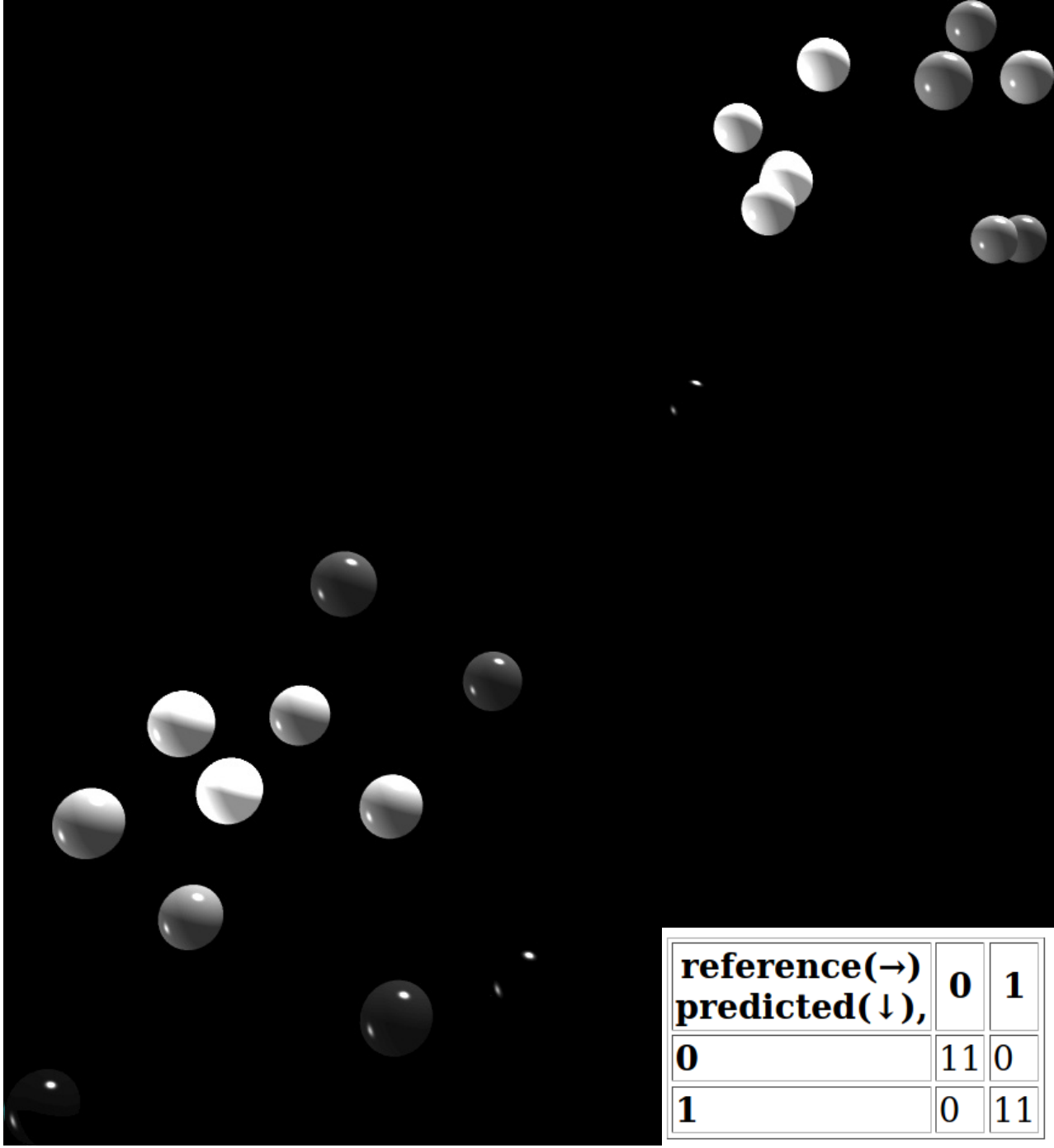
| reference(→) predicted(↓), | 0 | 1 |
|---|---|---|
| 0 | 11 | 0 |
| 1 | 0 | 11 |

Github Example 01

# Example 01: bimodal

**“Density” estimate rho(x):**

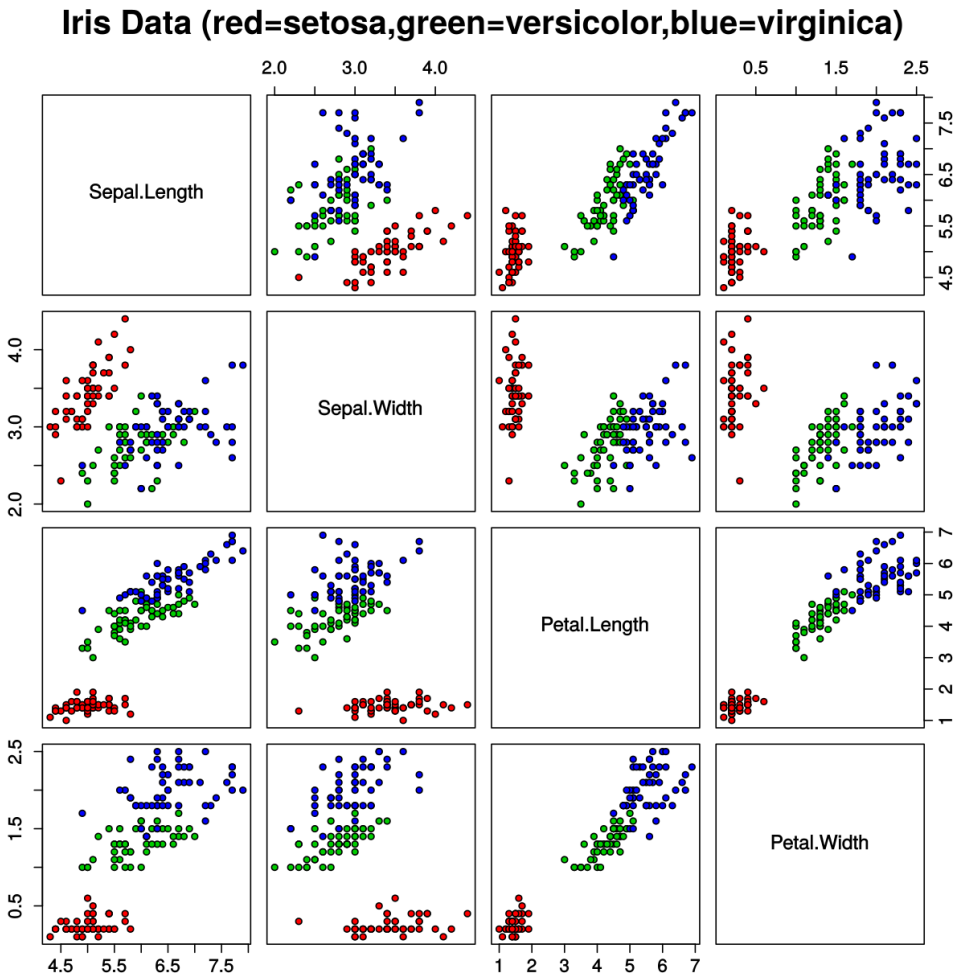
**Scaled to [0, 1]**



| reference(→) predicted(↓), | 0 | 1 |
|---|---|---|
| 0 | 11 | 0 |
| 1 | 0 | 11 |

Github Example 01

# Less contrived example: Iris Data Set




**Heads up:** the versicolor and virginica are not perfectly resolvable..
..they interpenetrate in space!

Thanks Wikipedia for the plot!

88.7% Accuracy

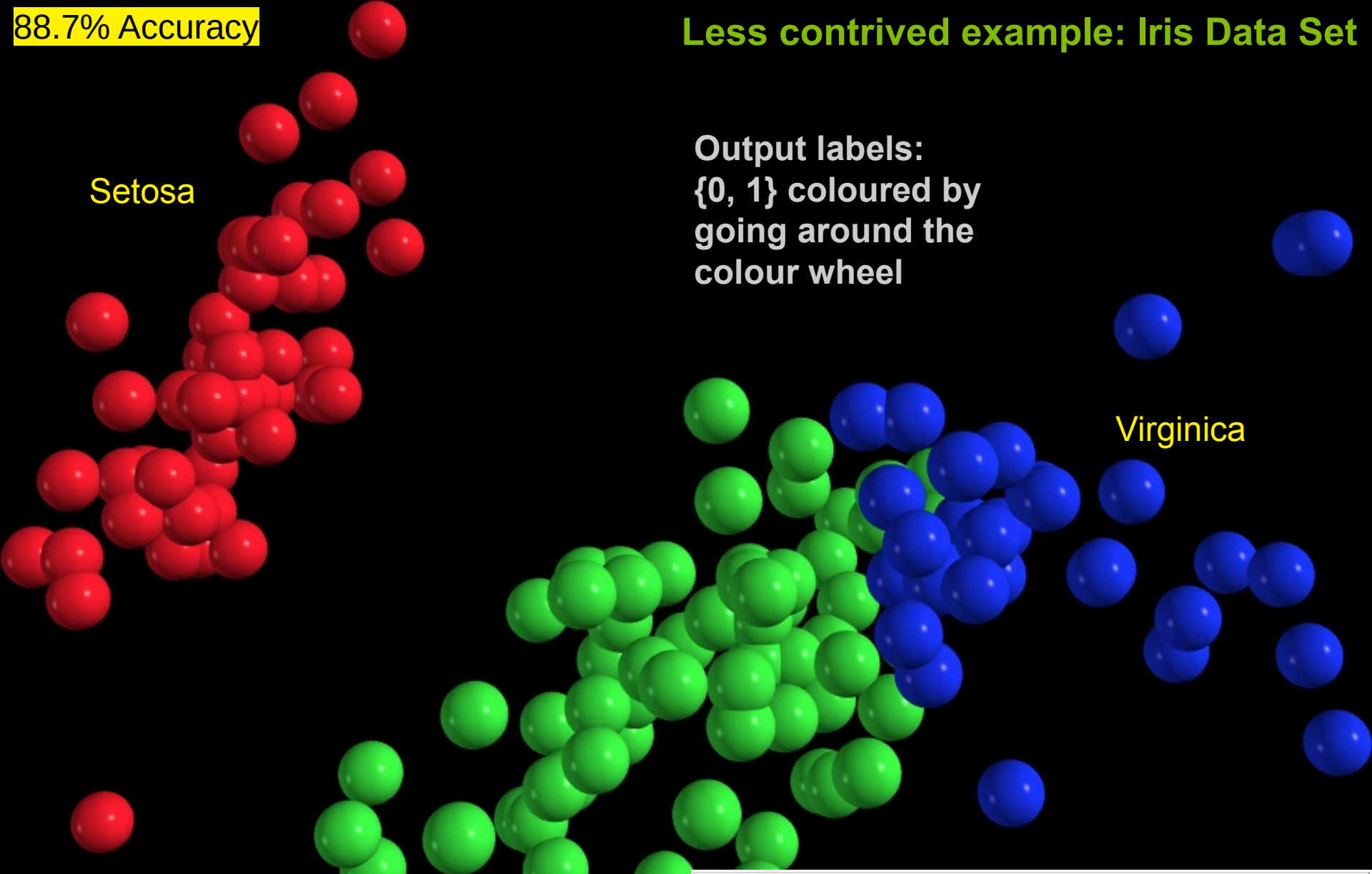
# Less contrived example: Iris Data Set

Setosa

Output labels:
{0, 1} coloured by going around the colour wheel

Virginica

Versicolor

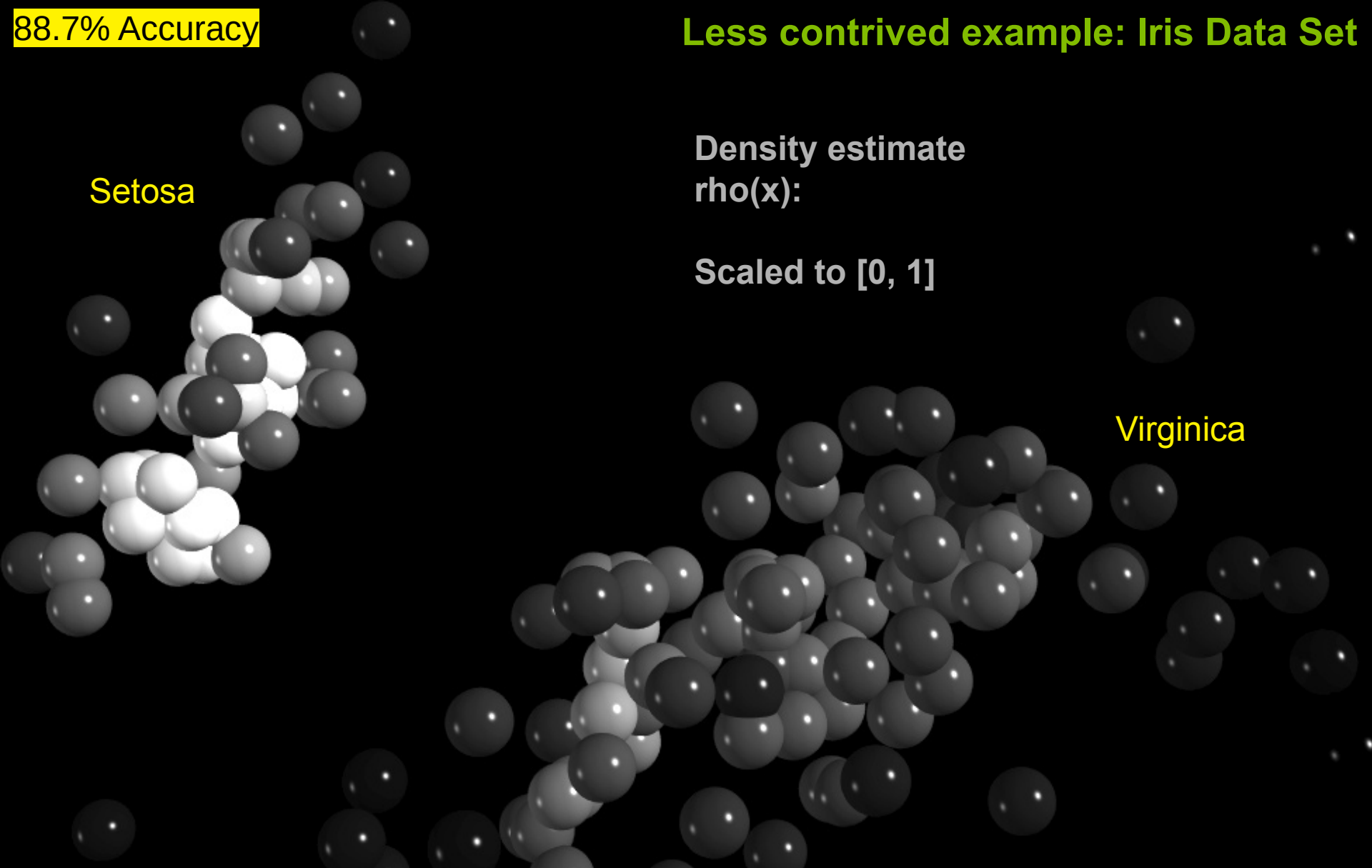


| reference(→) predicted(↓), | Iris-setosa | Iris-versicolor | Iris-virginica |
|---|---|---|---|
| Iris-setosa | 50 | 0 | 0 |
| Iris-versicolor | 0 | 50 | 17 |
| Iris-virginica | 0 | 0 | 33 |

Github Example 2

88.7% Accuracy

# Less contrived example: Iris Data Set




| reference(→) predicted(↓), | Iris-setosa | Iris-versicolor | Iris-virginica |
|---|---|---|---|
| Iris-setosa | 50 | 0 | 0 |
| Iris-versicolor | 0 | 50 | 17 |
| Iris-virginica | 0 | 0 | 33 |

Github Example 2

88.7% Accuracy

# Less contrived example: Iris Data Set

Setosa

Density estimate rho(x):

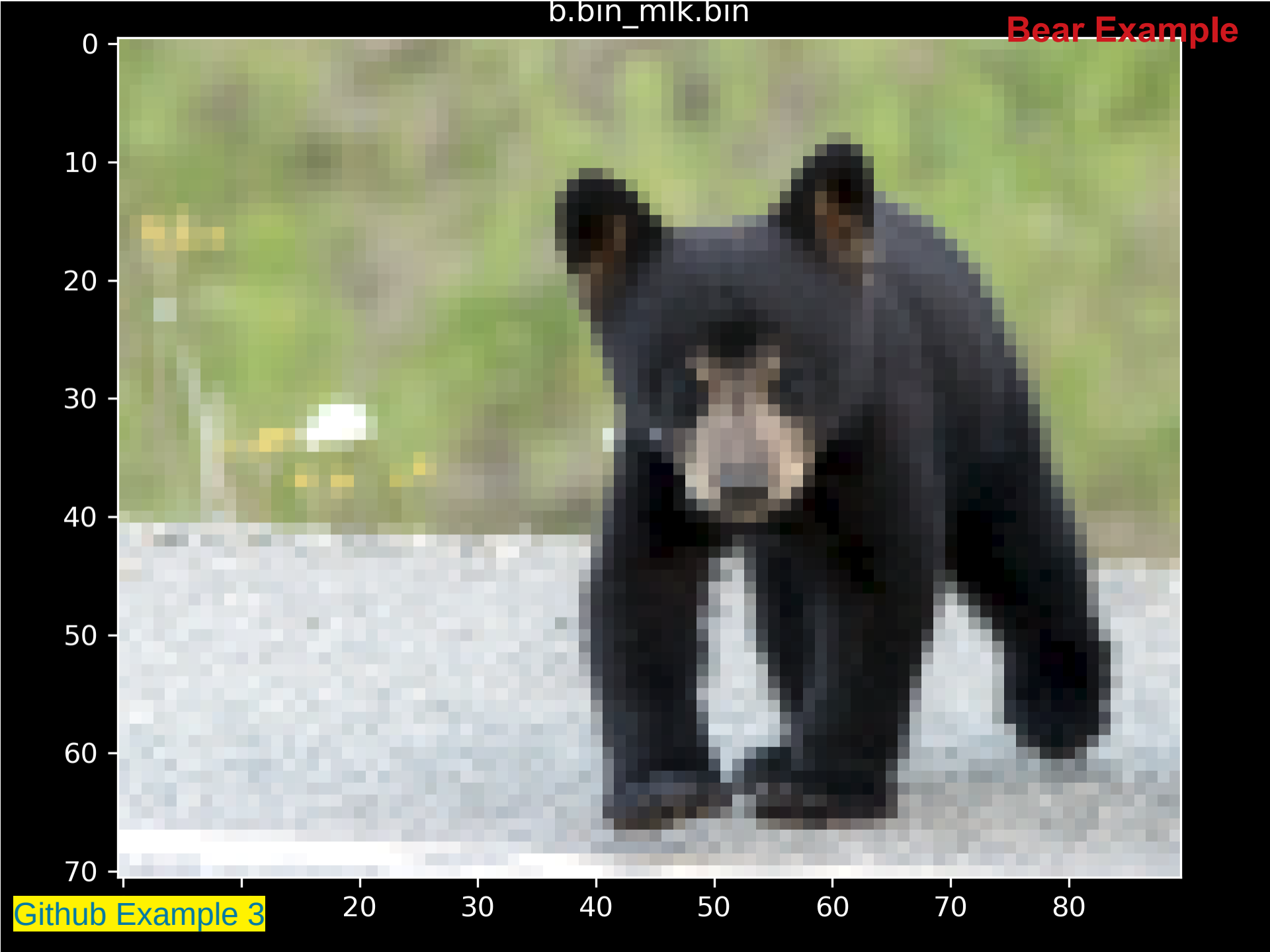
Normalized within cluster, so that each hill-top has rho(x) = 1

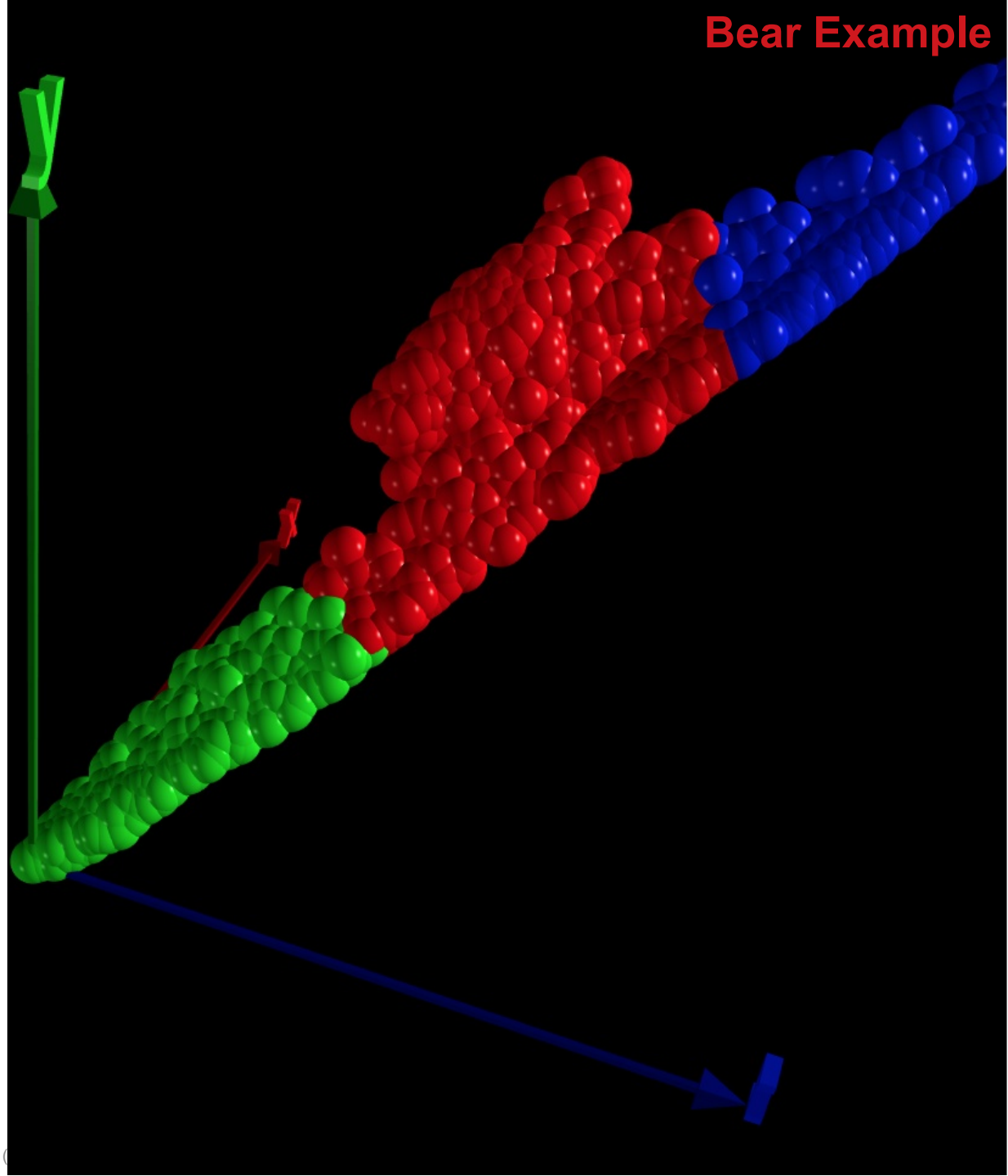
Virginica

Versicolor

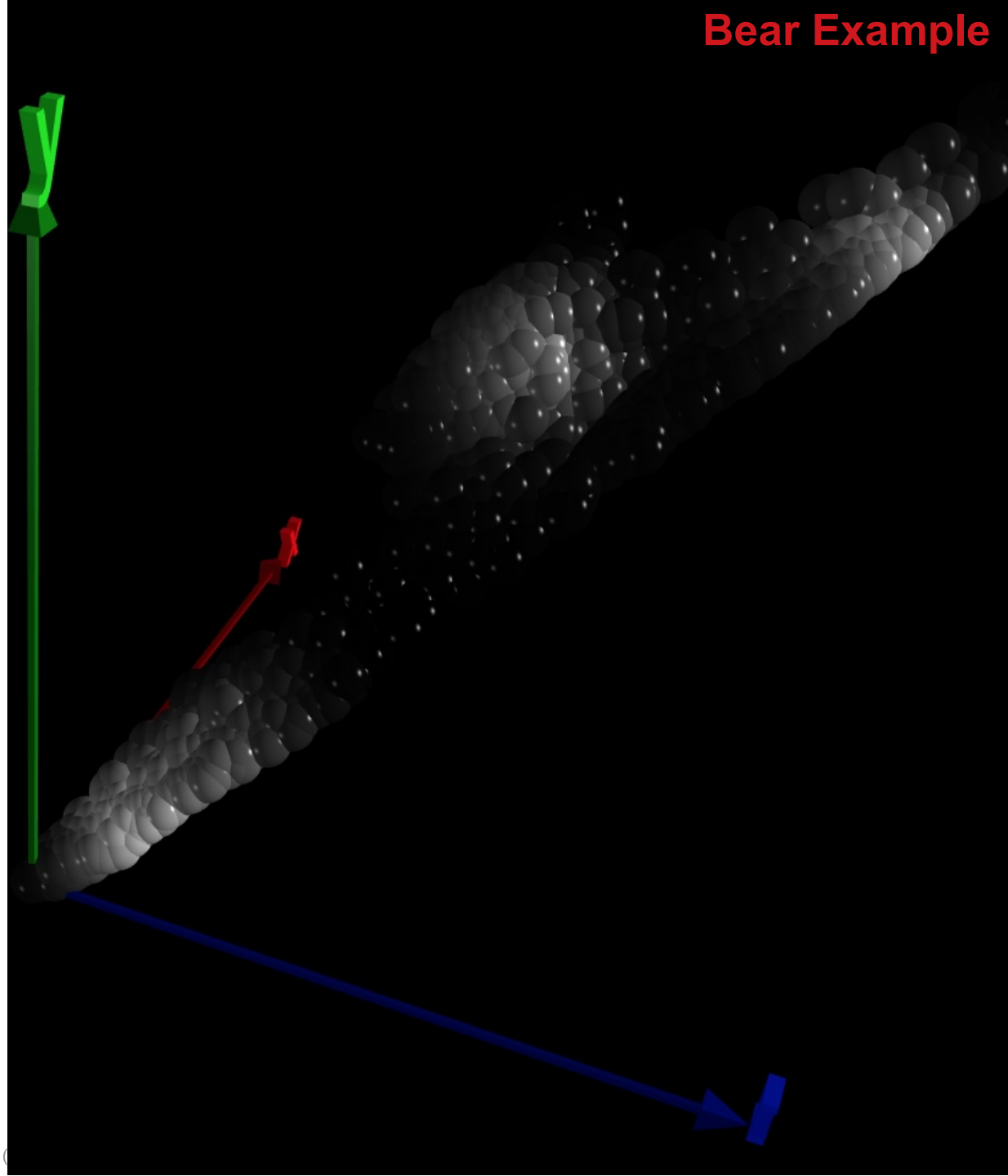
| reference(→) predicted(↓), | Iris-setosa | Iris-versicolor | Iris-virginica |
|---|---|---|---|
| Iris-setosa | 50 | 0 | 0 |
| Iris-versicolor | 0 | 50 | 17 |
| Iris-virginica | 0 | 0 | 33 |

Github Example 2


b.bin_mlk.bin
Bear Example
0
10
20
30
40
50
60
70
20
30
40
50
60
70
80
Github Example 3




Github Example 3




Github Example 3

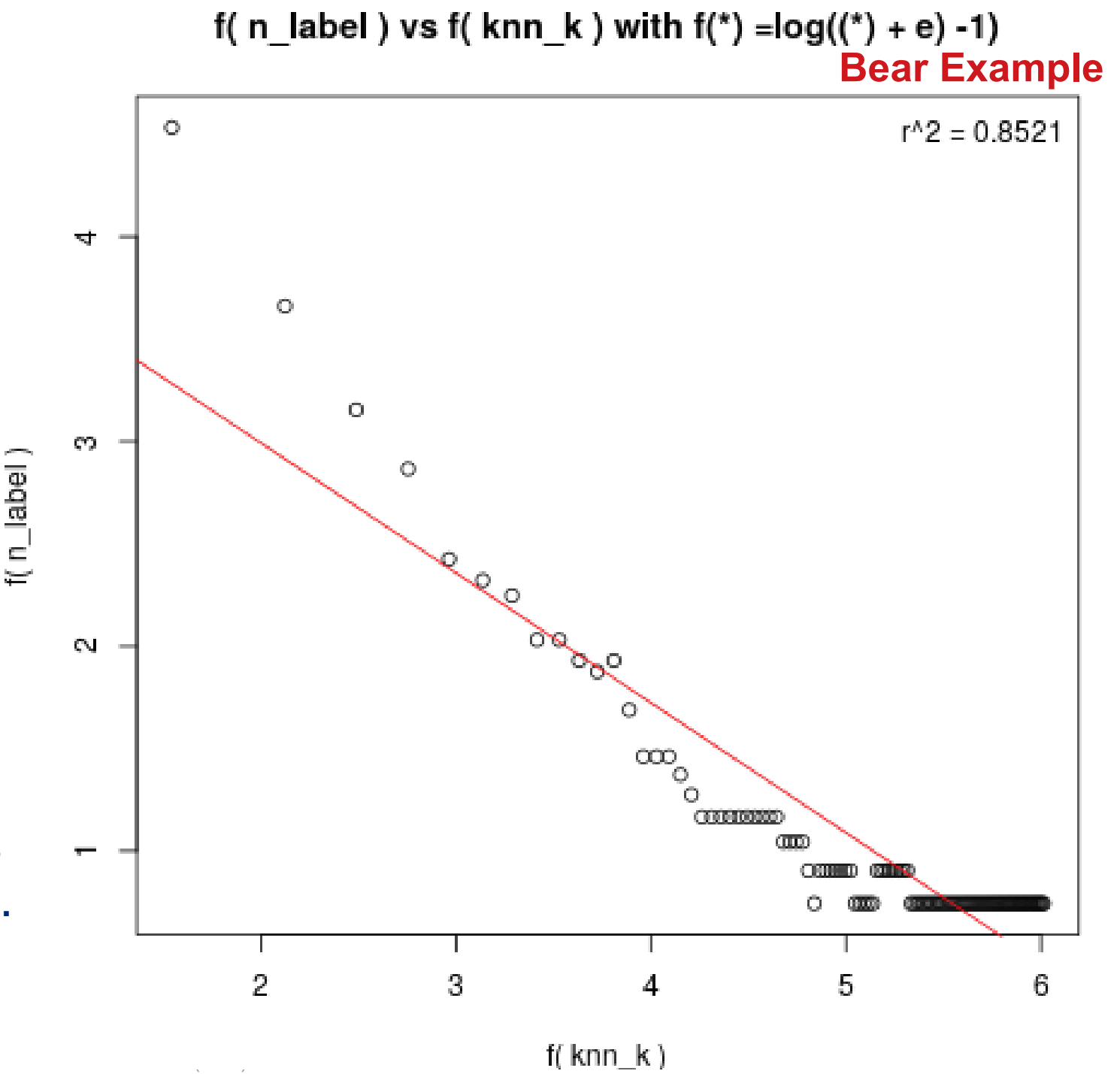
**Number of labels vs. knn_n: log plot! "scree plot"**

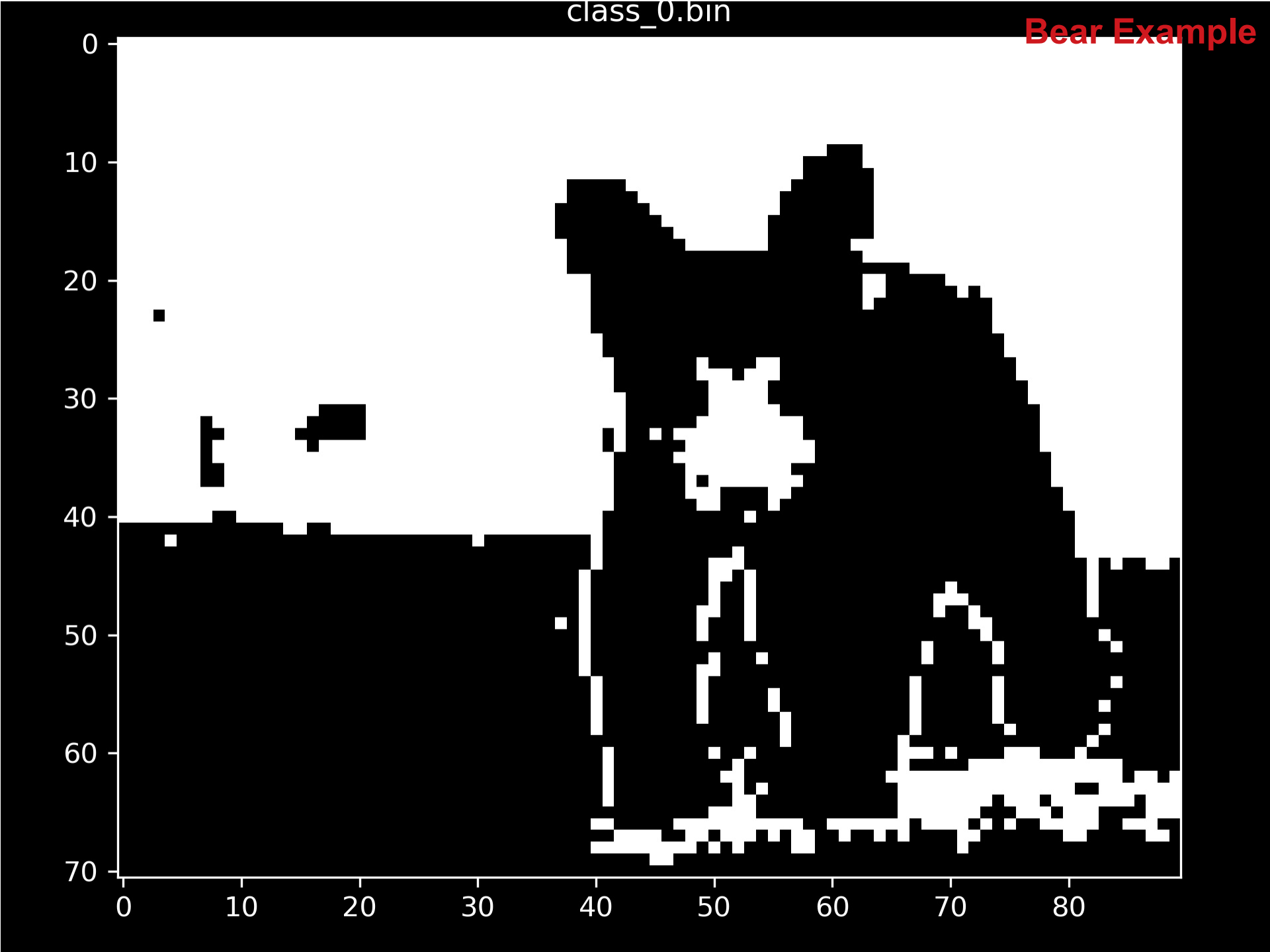
**1) Set KNN_K = 1000**

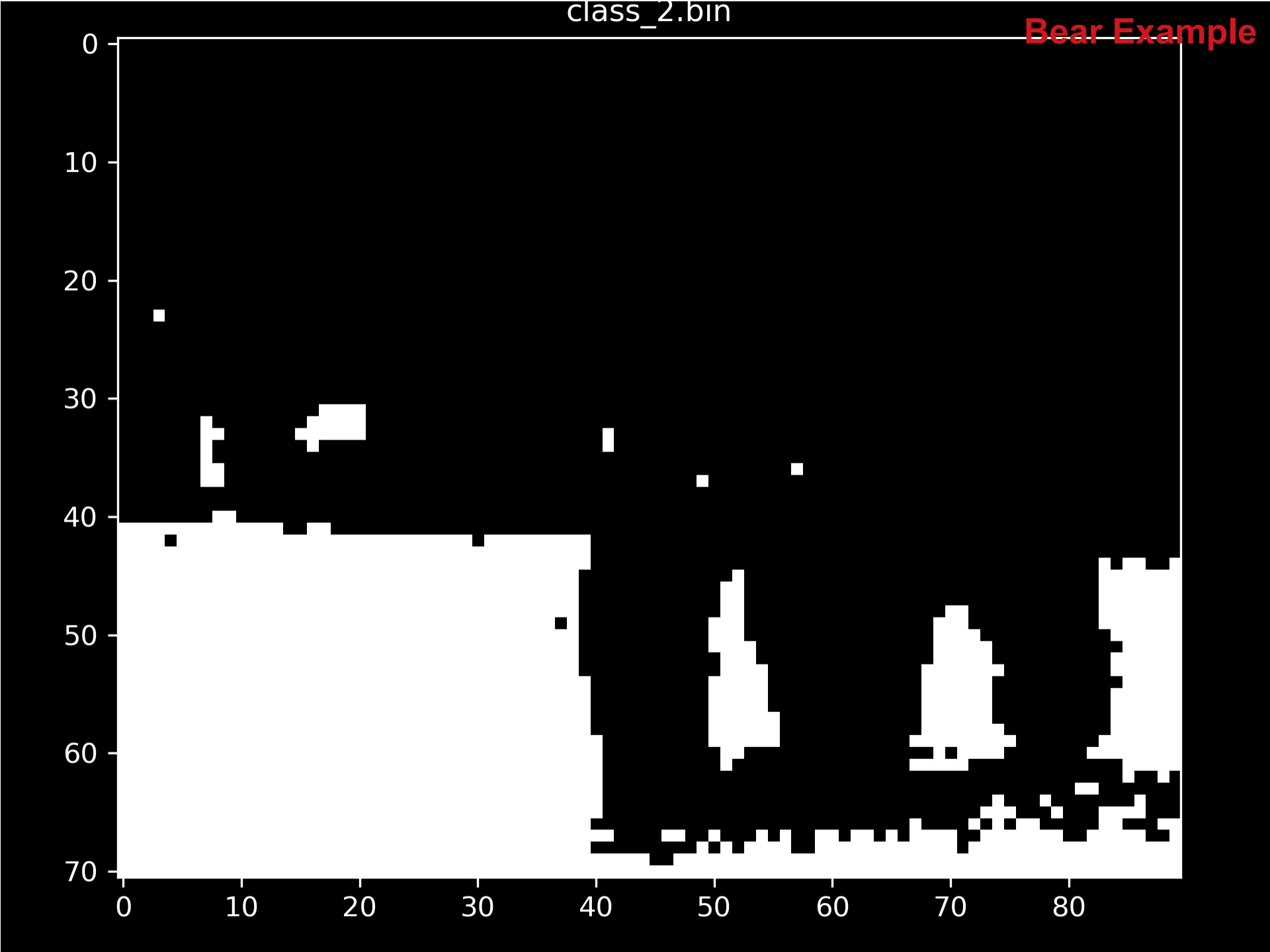
**2) Result: Three labels**

3) plot the three Labels in Image Coordinates (next slide)

**Q: Did it work? Bad news first..**




class_0.bin
Bear Example
0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70
80


class_2.bin
Bear Example
0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70
80

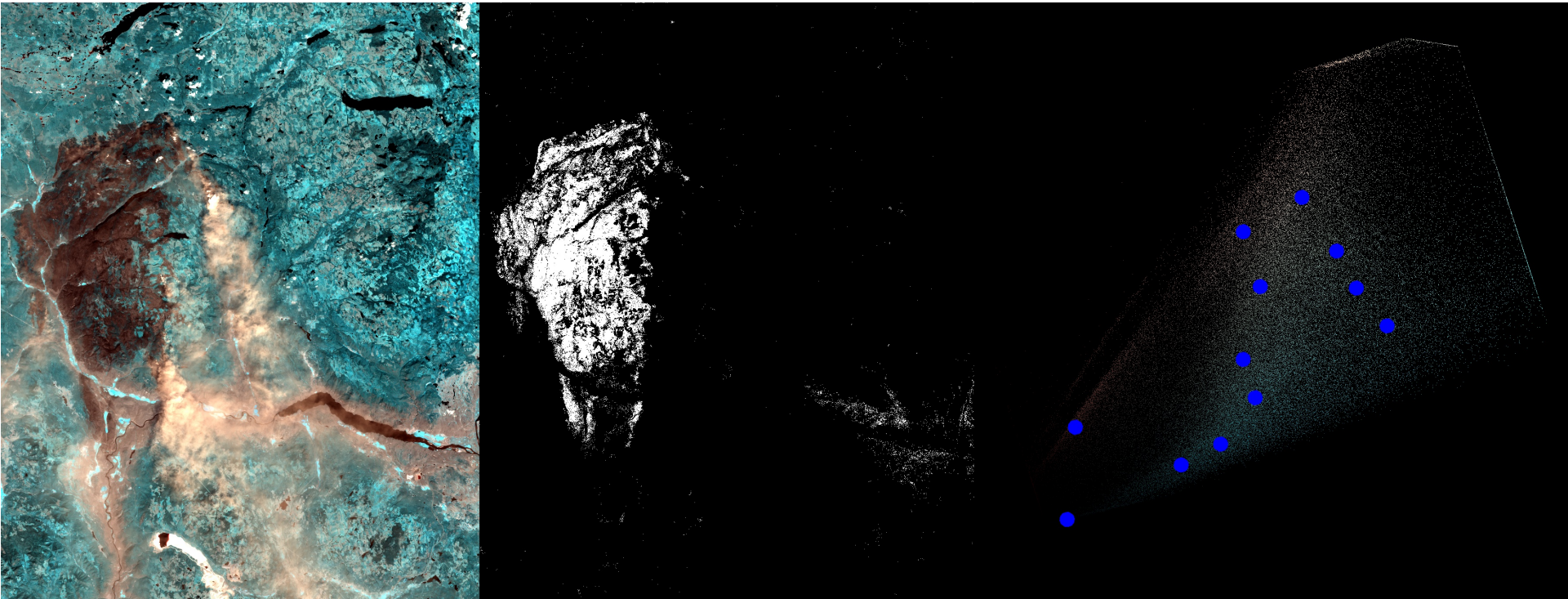

class_1.bin
Bear Example
0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70
80

## Smart-scissors for wildfires--
## --not in python. Can live-demo this another time!

**Left: Sentinel-2 data over Kamloops (false color image) 2017**

**Binary classification image**

**Interactive Scatter-plot with cluster centres (hill-tops)**



# Success Story

Was presented to int'l audience by a top Remote Sensing expert who pioneered a technique to 3d-map the environment despite cloud cover

Also invited for submission to int'l radar-imaging software package, that's administered by ESA. The invitation is still open!

# Radar Polarimetry for Forestry Applications: ALOS and Radarsat-2 studies in Canada


**by**

S. R. Cloude[1], A. Marino[2], D. Goodenough[3], H Chen[3], A. Richardson[3], B. Moa[4]

(1) AEL Consultants, Edinburgh,Scotland, UK, e-mail : aelc@mac.com

(2) University of Edinburgh, School of Geosciences, Scotland, UK

(3) Pacific Forestry Centre, Natural Resources Canada, Victoria, BC, Canada

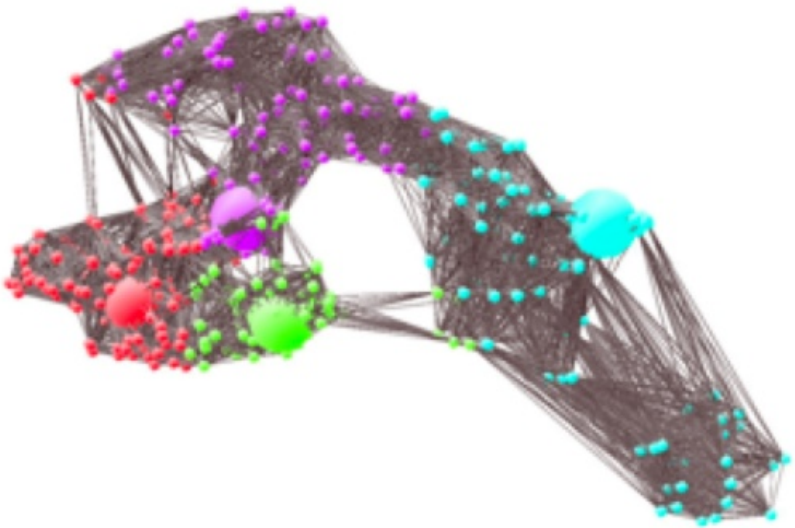
(4) Department of Computer Science, University of Victoria, Victoria, BC, Canada

# Fire Scar Classification: New Unsupervised Manifold Method - KNN Graph Clustering (KGC)*

A new unsupervised classification was devised.

- It is ***data driven***, with ***only one parameter (K).***
- It is intended to ***find clusters of arbitrary shape***.
- It ***doesn't directly assume the number of clusters***.
- It ***doesn't assume the data has any underlying parameterized distribution*** (e.g. Wishart).
- ***Is stable*** in terms of both the number of clusters, and the cluster shapes (with respect to the parameter K).

KGC uses an ***approximate manifold (the K Nearest Neighbor Graph)*** to describe arbitrary cluster shapes in N dimensions..here N = 3



*A. Richardson et al "Unsupervised Classification of Polarimetric SAR Data using the K-nearest Neighbour Graph", Proc. IGARSS 2010, Hawaii, USA, July 2010

# KGC-Fire-Scar Application


- Binary classification results for two values of the parameter K (2002 burned area class).
- Increasing K ("blurring" the density estimate) resulted in less accurate class boundaries.
- For speed, only 1/30th of the 737x249 data points were used.
- A random algorithm was used to assign classes rapidly.
- Increased computational power will further improve the results.

# Questions? Feedback?

Thank you!