CS 649 Big Data: Tools and Methods
Fall Semester, 2022
Doc 19 Clustering
Mar 10, 2022

Clustering

Unsupervised machine learning

- Algorithm “looks” for structure in the data

Clustering

- Groups data that is similar to each other in some way

Uses for Clustering

Bioinformatics
- Sequence analysis
  - Group sequences into gene families
- Human genetic clustering
  - Infer ancestral background

Market research
- Partition consumers into market segments based on surveys & test panels

Image segmentation
- Divide image into regions for border detection or object recongnition

Recommender Systems

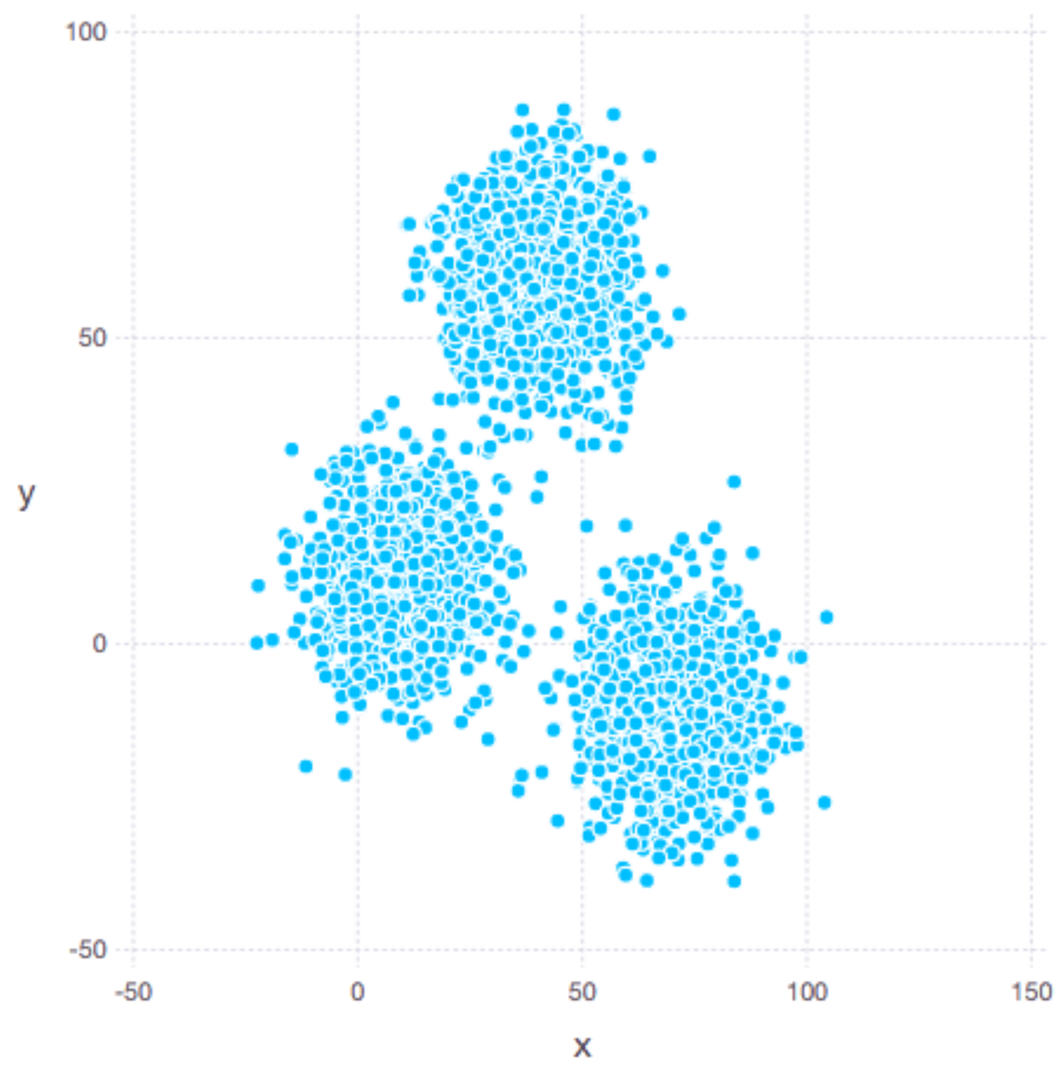
Examples
- Last.fm
- Pandora Radio
- Netflix recommendations
- Amazon recommendations
- Facebook friend recommendations

Machine Learning algorithms used
- Bayesian Classifiers
- Cluster analysis
- Decision trees
- Artificial neural networks

Clustering

Clustering algorithms group data based on distance

- What is distance?
- Normalizing data affects distance



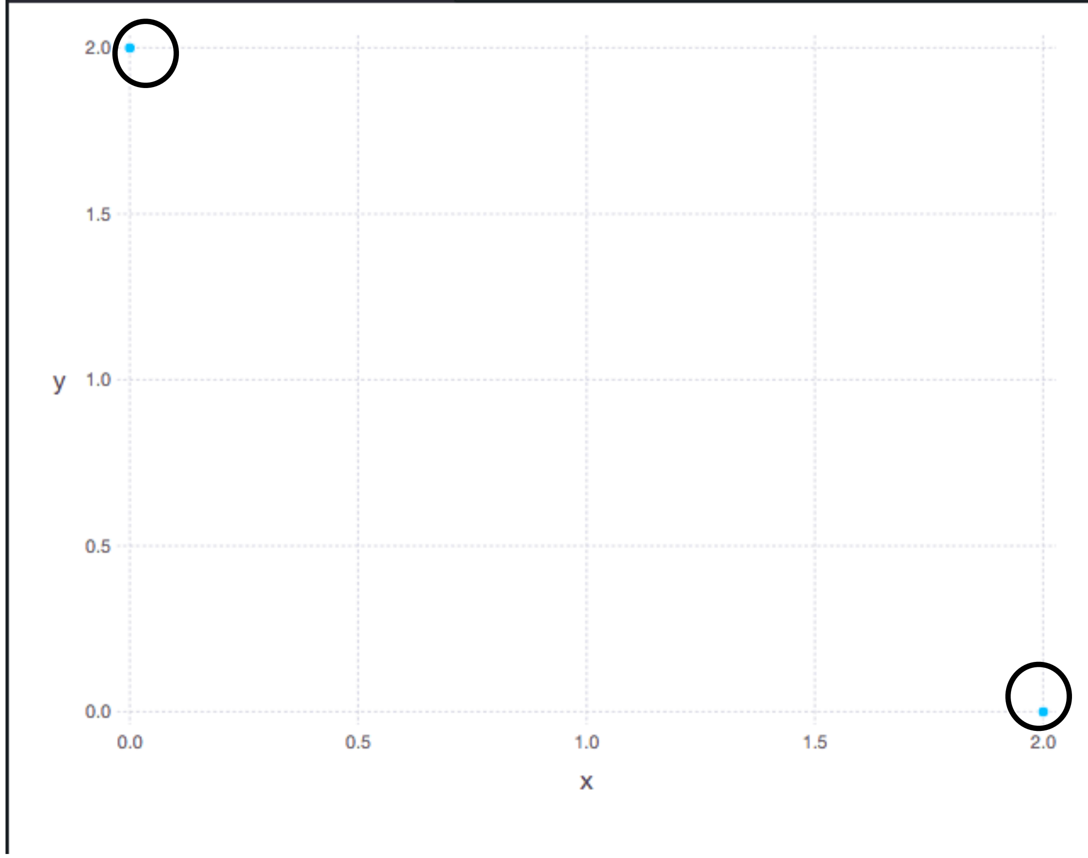
100
50
0
-50
y
-50
0
50
100
150
x

Distance

Distances.jl

Euclidean distance
Squared Euclidean distance
Periodic Euclidean distance
Cityblock distance
Total variation distance
Jaccard distance
Rogers-Tanimoto distance
Chebyshev distance
Minkowski distance
Hamming distance
Cosine distance
Correlation distance
Chi-square distance
Kullback-Leibler divergence
Generalized Kullback-Leibler divergence
Rényi divergence
Jensen-Shannon divergence
Mahalanobis distance
Squared Mahalanobis distance
Bhattacharyya distance
Hellinger distance
Haversine distance
Mean absolute deviation
Mean squared deviation
Root mean squared deviation
Normalized root mean squared deviation
Bray-Curtis dissimilarity
Bregman divergence

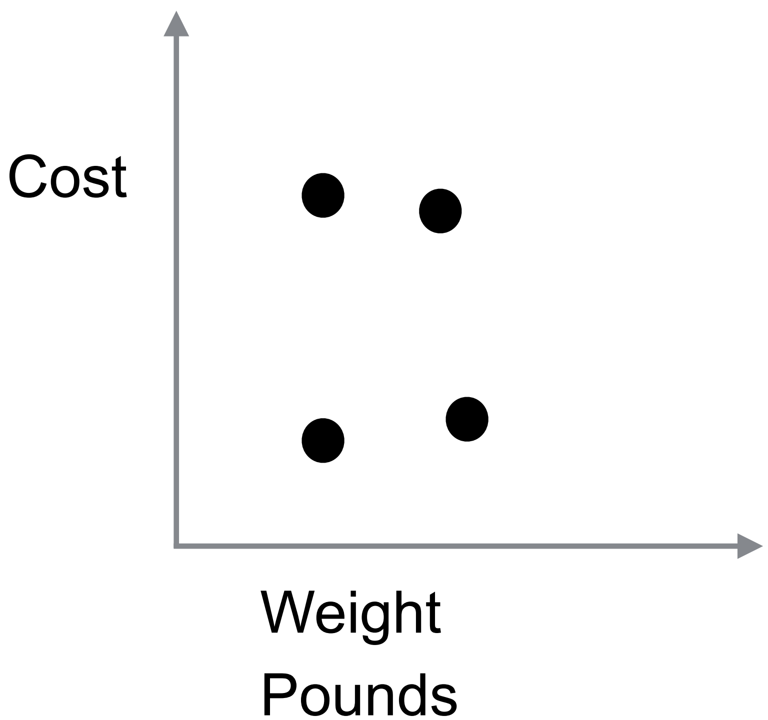
using Distances

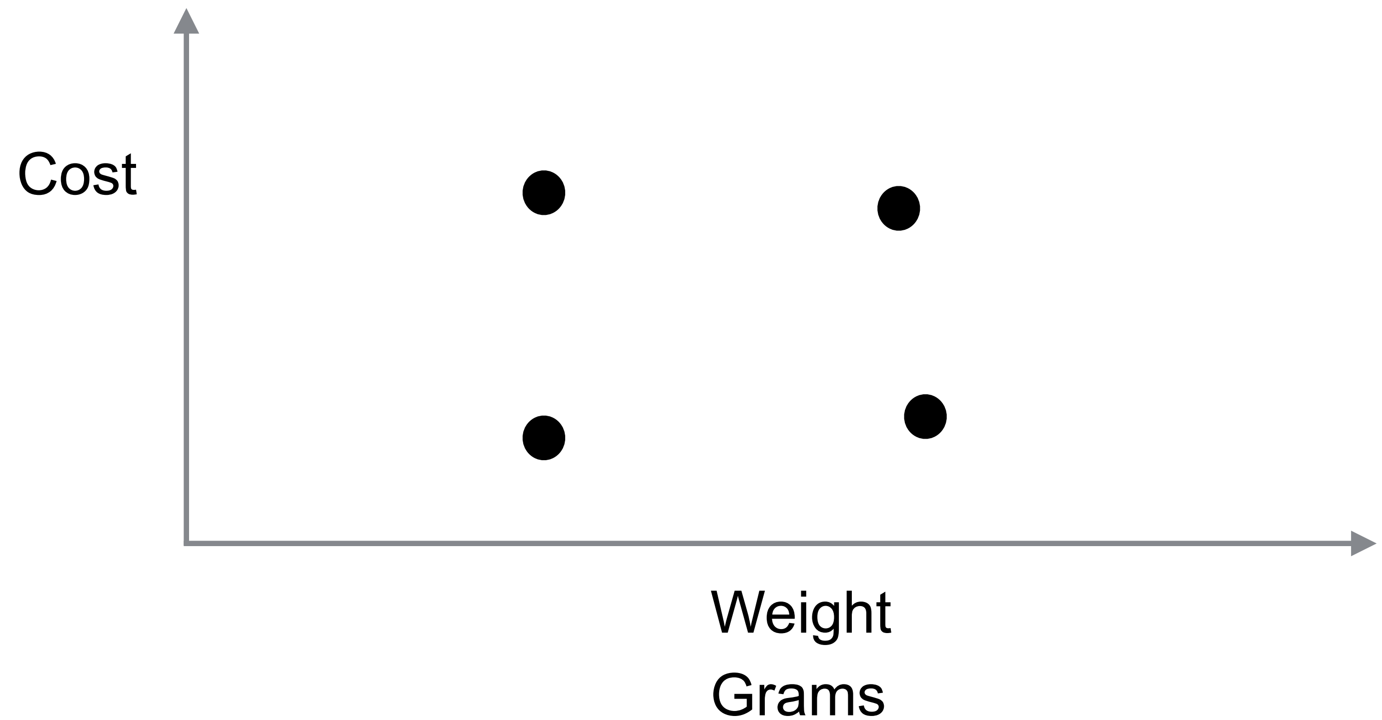



```
euclidean(x, y) =    sqrt(sum((x - y) .^ 2))
   euclidean([2,0],[0,2]) == 2.83

cityblock(x, y) =     sum(abs(x - y))
   cityblock([2,0],[0,2]) == 4

hamming(x, y) =    sum(x .!= y)
   hamming([2,0],[0,2]) == 2
   hamming([9,0],[0,2]) == 2

cosine_dist(x,y) = cos(x,y)
   cosine_dist([2.0,0.0], [0.0,2.0])) == 1
   cosine_dist([2.0,0.0], [10.0,0.0])) == 0

jaccard(x, y)   = 1 - sum(min(x, y)) / sum(max(x, y))
   jaccard([2,0],[0,2])  == 1
```

Normalization

Clustering relies on distance between data points which scale can affect





Normalization

Clustering relies on distance between data points which scale can affect

Max-min

Mean-standard deviation

Sigmoidal normalization

Softmax

Max-min

min_max_norm(x) = (x - minimum(x)) / (maximum(x) - minimum(x))

maps data -> [0, 1]
Cheap to compute
Outliers compress the data

1	0.0
2	0.0526316
3	0.105263
4	0.157895
9	0.421053
20	1.0

1	0.0
2	0.00050025
3	0.0010005
4	0.00150075
9	0.004002
20	0.00950475
2000	1.0

Mean-standard deviation (Z-score)

mean_std_norm(x) = (x - mean(x)) / std(x)

Unbounded, but mainly in [-3, 3]

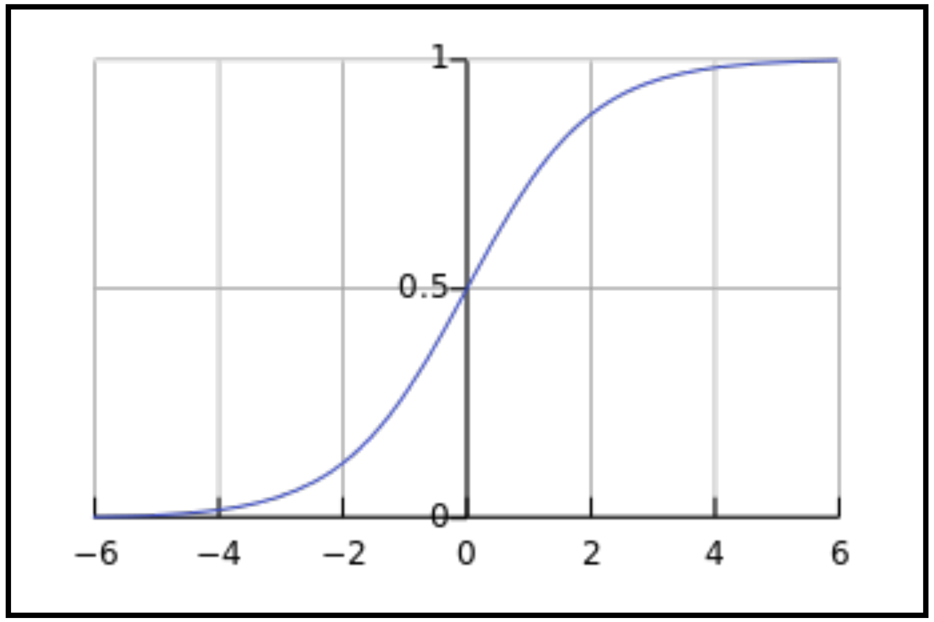
Contains negative numbers

Has outlier issues

1	-0.766406
2	-0.62706
3	-0.487713
4	-0.348367
9	0.348367
20	1.88118

1	-0.385249
2	-0.383922
3	-0.382595
4	-0.381268
9	-0.374632
20	-0.360034
2000	2.2677

Sigmoidal Normalization



sigmoidal_norm(x) = 1 ./ (1 + exp(-x))

Range (0, 1)
Not very useful as given in text

1	0.731059
2	0.880797
3	0.952574
4	0.982014
9	0.999877
20	1.0

1	0.731059
2	0.880797
3	0.952574
4	0.982014
9	0.999877
20	1.0
2000	1.0

Logistic Function

logistic_norm(x,k,c) = 1 ./(1 + exp(-k*(x - c)))

c = 0

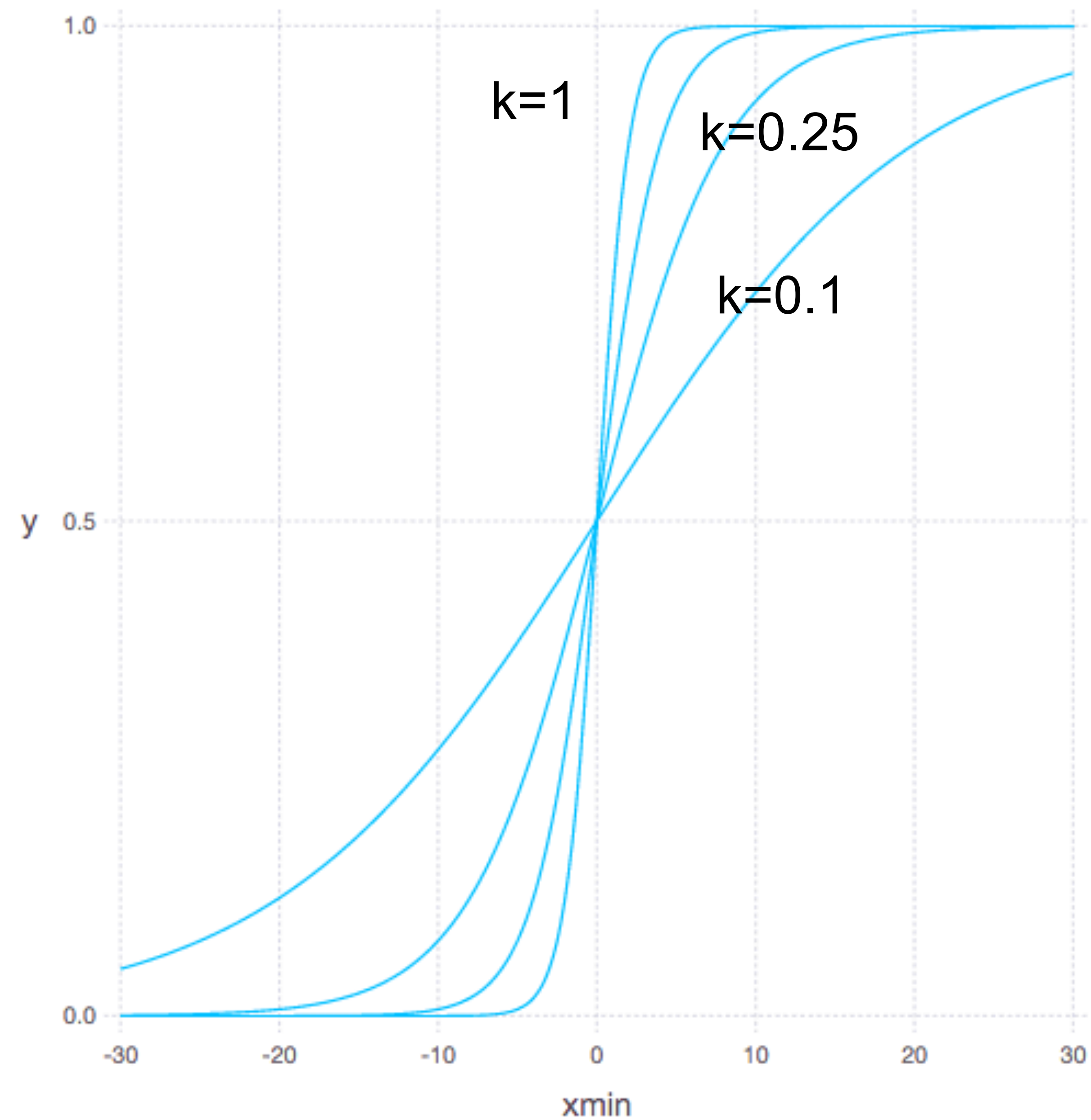
k = 1, 0.5, 0.25, 0.1

Range (0, 1)

Need to select k & c

Commonly used in neural networks

Bases of Elo ranking system



Logistic Function

logistic_norm(x,k,c) = 1 ./(1 + exp(-k*(x - c)))

	k= 1, c= 0	k= 0.5, c= 0	k= 0.2, c= 0	k= 0.2, c= 9
1	0.731059	0.622459	0.549834	0.167982
2	0.880797	0.731059	0.598688	0.197816
3	0.952574	0.817574	0.645656	0.231475
4	0.982014	0.880797	0.689974	0.268941
9	0.999877	0.989013	0.858149	0.5
20	1.0	0.999955	0.982014	0.90025

1	0.731059	0.622459	0.549834	0.167982
2	0.880797	0.731059	0.598688	0.197816
3	0.952574	0.817574	0.645656	0.231475
4	0.982014	0.880797	0.689974	0.268941
9	0.999877	0.989013	0.858149	0.5
20	1.0	0.999955	0.982014	0.90025
2000	1.0	1.0	1.0	1.0

Softmax Normalization

softmax_norm(x) = 1 ./(1 + exp(-(x - mean(x))/std(x)))

Range (0, 1)

mean -> 0.5

Near linear within standard deviation of mean

Keeps outliers, but reduces their influence

1	0.317257
2	0.348178
3	0.380432
4	0.413779
9	0.586221
20	0.867747

1	0.404861
2	0.405181
3	0.405501
4	0.405821
9	0.407422
20	0.410951
2000	0.906166

Text Normalization

Extracting text from xml, json

tokenizing

Punctuation & non text characters ()

Non relavent word
  the, and, this, ...

Root (stem) words
  like, liked

Stem Words

worked
working

worker
workers

sleep
sleeping
slept

Text & Distance - Jaccard Distance

Let A and B be sets

The Jaccard index or Jaccard similarty coefficient is

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Range [0, 1]

If A == B then J(A,B) = 1

Jaccard Distance for sets

dj(A, B) = 1 - J(A, B)

Example

```
a = StringDocument("Music is the food of love")
b = StringDocument("War is the locomotive of history")
c = StringDocument("It's lovely that you're musical")
```

```
jaccard_dist(a,b) == 0.667
jaccard_dist(a,c) == 1.00
```

Example Revisited

```
a = StringDocument("Music is the food of love")
b = StringDocument("War is the locomotive of history")
c = StringDocument("It's lovely that you're musical")

normalize_text!(a)
normalize_text!(b)
normalize_text!(c)

jaccard_dist(a,b) == 1.00
jaccard_dist(a,c) ==  0.333
```

Text as Vectors - Term Frequency

Find all unique words in your text - say n words

Map each word to a number from 1 - n

That number becomes the words location in a vector

Count the number of time the word appears

Place that number in the vectors location

Example

"Music is the food of love"
"War is the locomotive of history"
"It's lovely that you're musical"

"music food love"
"war locomotive histori"
"love music"

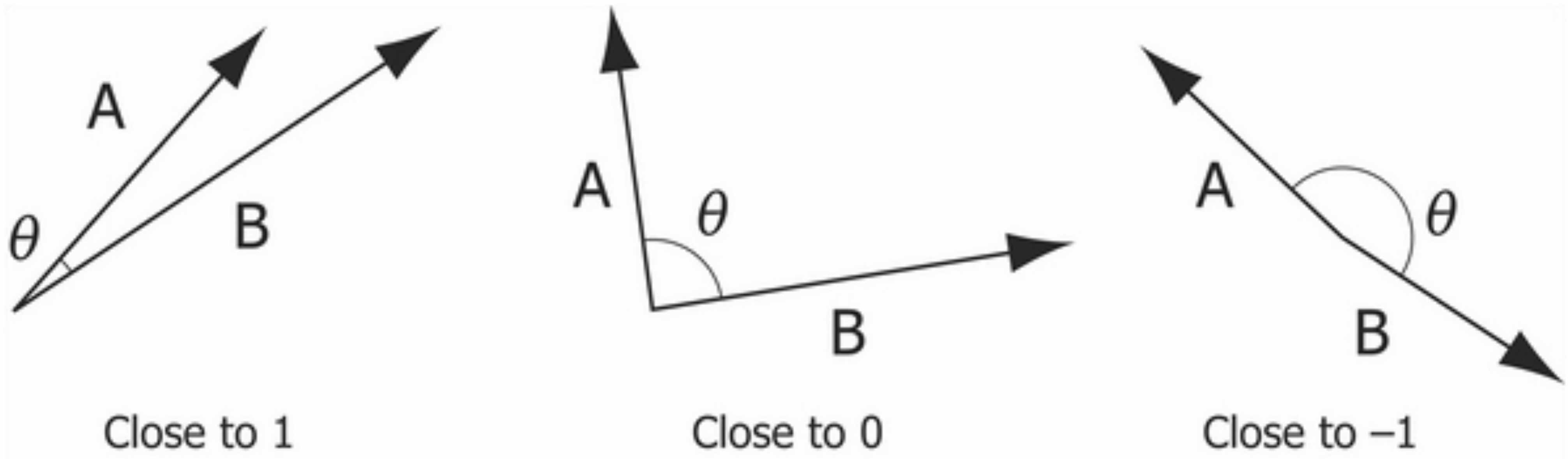
"food" = 1
"histori" = 2
"locomotive" = 3
"love" = 4
"music" = 5
"war" = 6

"music food love" -> [1, 0, 0, 1, 1, 0]

"war locomotive histori" -> [0, 1, 1, 0, 0, 1]

"love music" -> [0, 0, 0, 1, 1, 0]

Cosine Distance



cos(0) = 1.0

cos(deg2rad(90)) = 6.12e-17

cos(deg2rad(180)) = -1.00

Cosine Distance

"music food love" -> [1, 0, 0, 1, 1, 0]

"war locomotive histori" -> [0, 1, 1, 0, 0, 1]

"love music" -> [0, 0, 0, 1, 1, 0]

"music food love" verses "war locomotive histori"

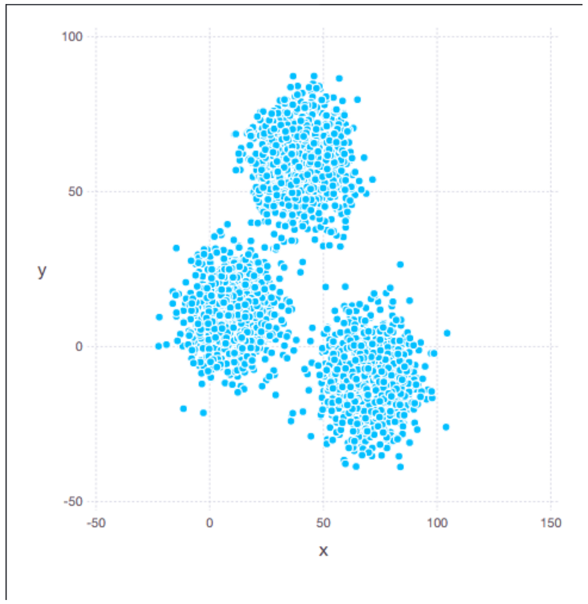
cosine_dist([1, 0, 0, 1, 1, 0], [0, 1, 1, 0, 0, 1]) = 1.00

"music food love" verses "love music"

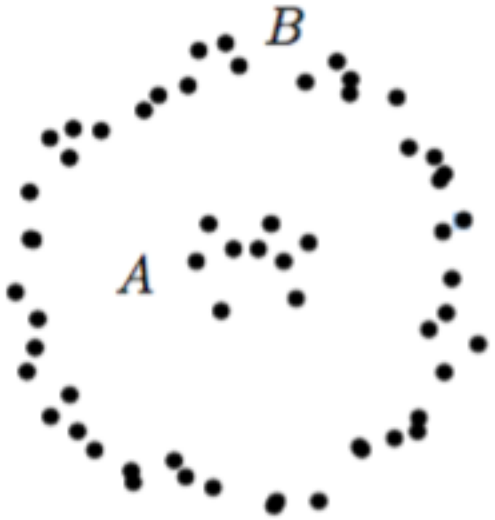
cosine_dist([1, 0, 0, 1, 1, 0]), [0, 0, 0, 1, 1, 0]) = 0.184

Types of Clustering

Center-based Cluster Algorithms
- k-nearest neighbor
- k-means
- k-medoids
- Affinity propagation



Density clusters
- DBSCAN



K-Clustering - Basic Idea

Pick k points to be start of each cluster

1. Add each data point to the nearest cluster

2. Readjust the k points for each cluster

Repeat 1 & 2 until clusters are stable or reach given number of iterations

K-means

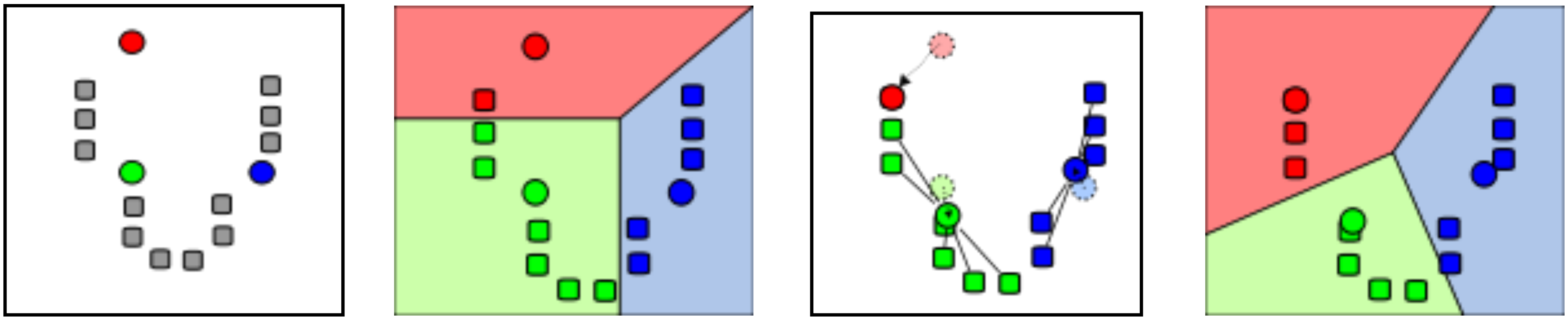
Select k points $m_1^1, m_2^1, \ldots, m_k^1$

For each data point x assign it to the mean that it is closest to form k clusters
Use square of the (Euclidiean) distance

For each cluster compute the mean of that cluster
Get new means $m_1^2, m_2^2, \ldots, m_k^2$

If points changed clusters repeat

Example



K-mediods

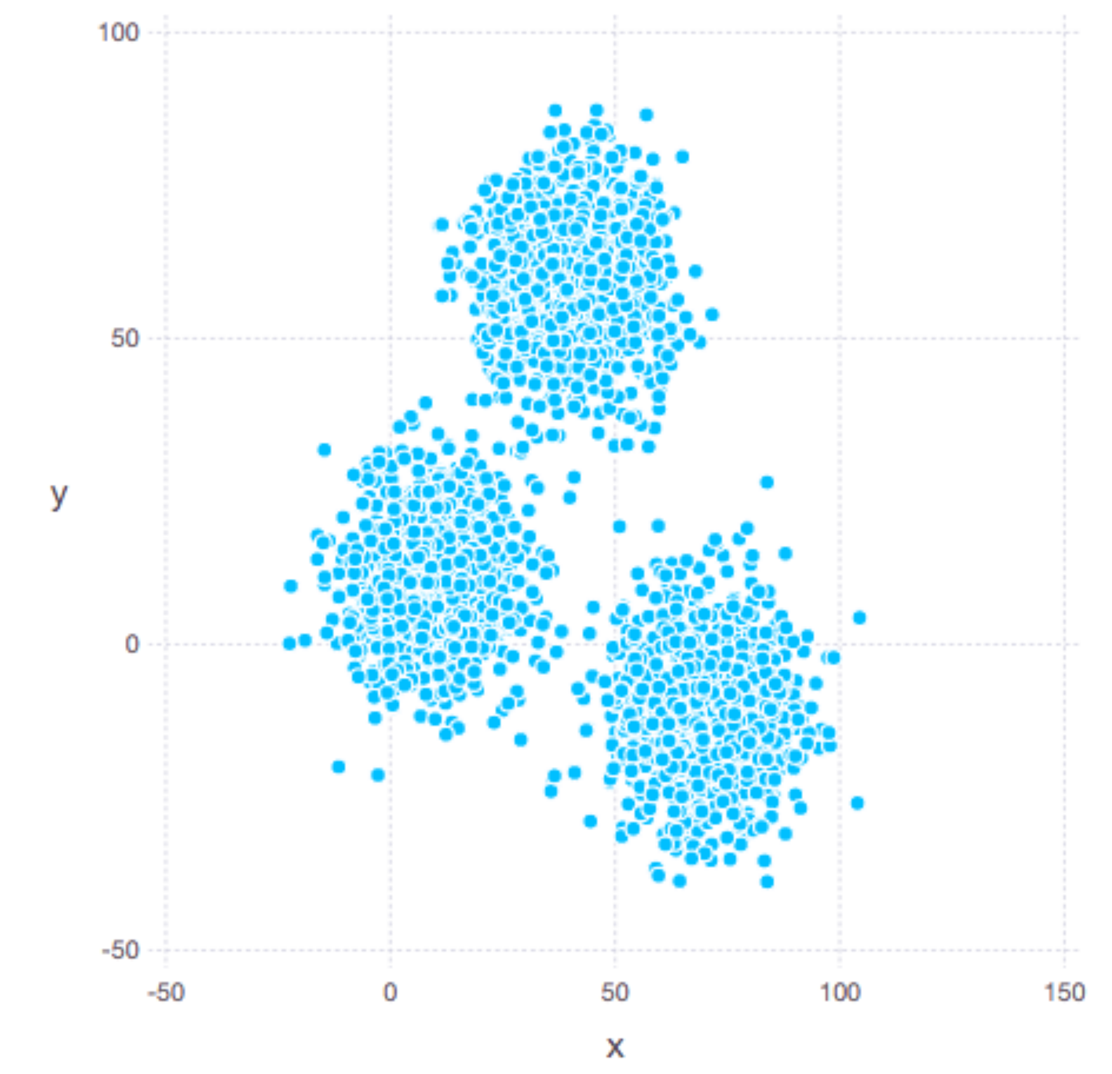
Differs from K-means in two ways

Centers of each cluster is data point nearest the mean point

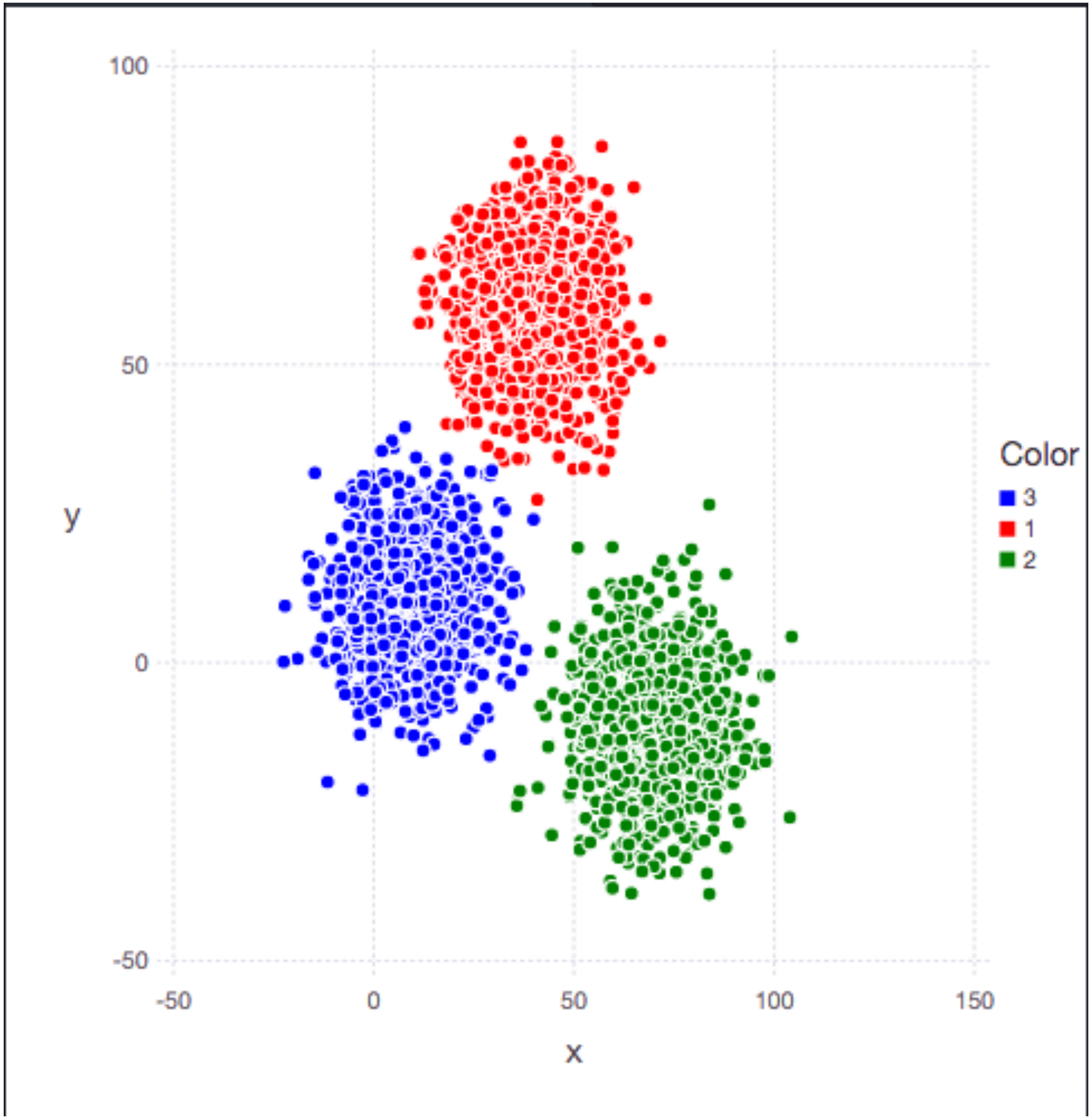
Uses distance matrix so can use any definition of distance

Sample Dataset

xclara = dataset("cluster", "xclara"



K-Means k= 3



Issues

Picking initial means

Picking number of clusters

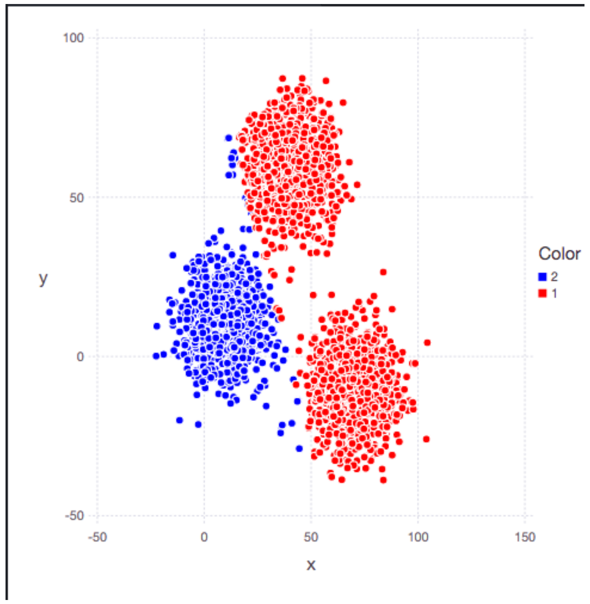
Measuring how good the clusters are

Normalization of data

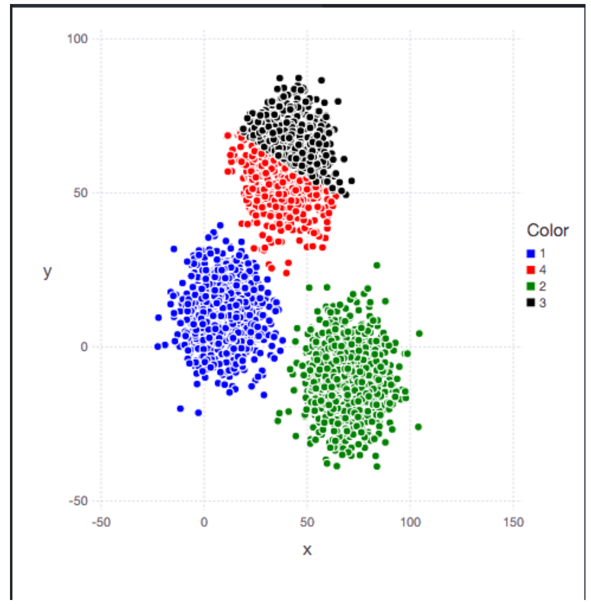
What is distance

Varying k

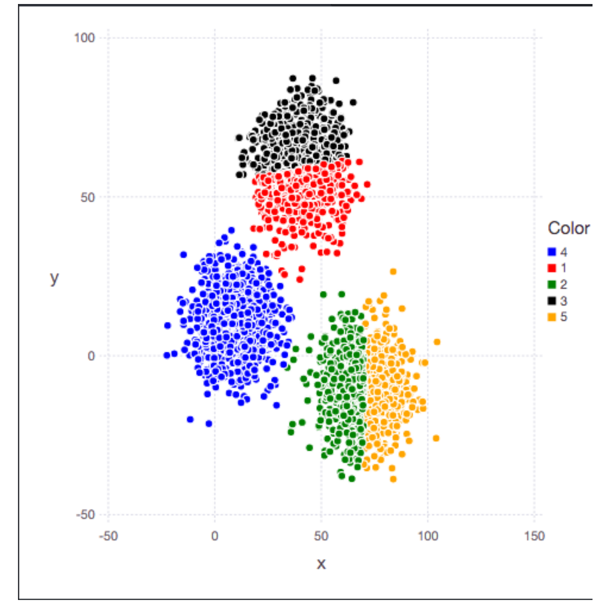
2



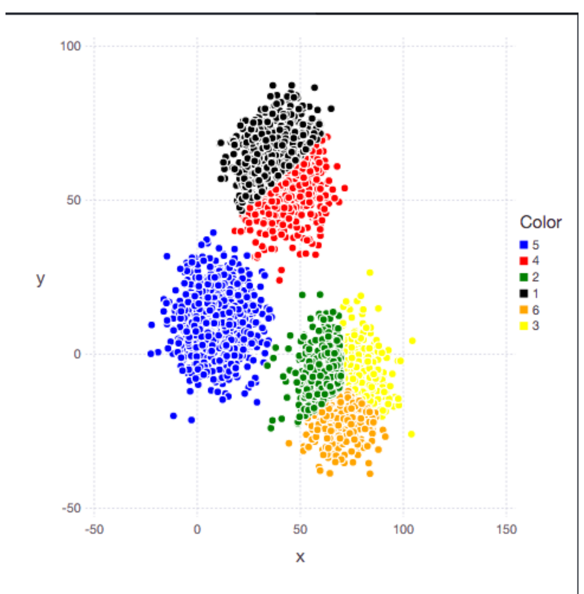
4



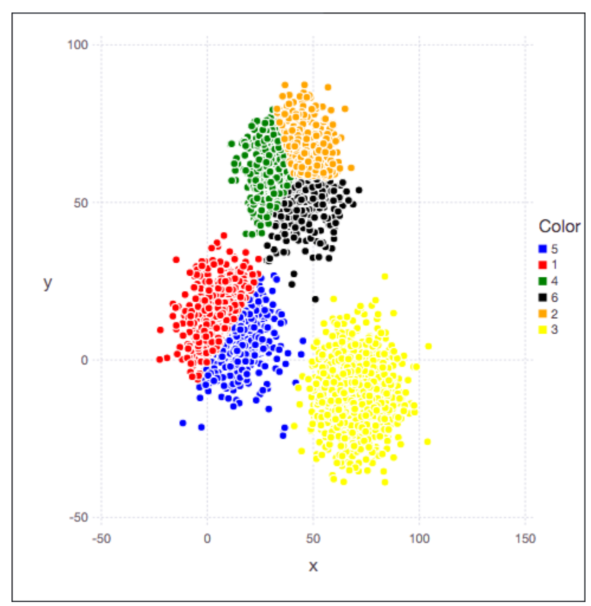
5



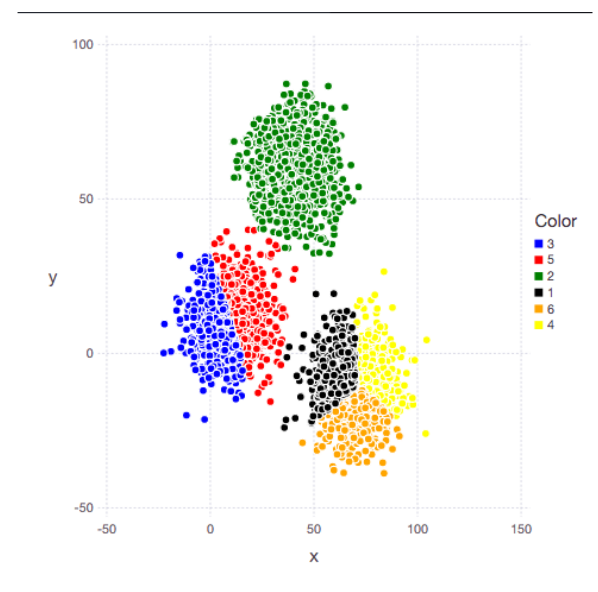
6



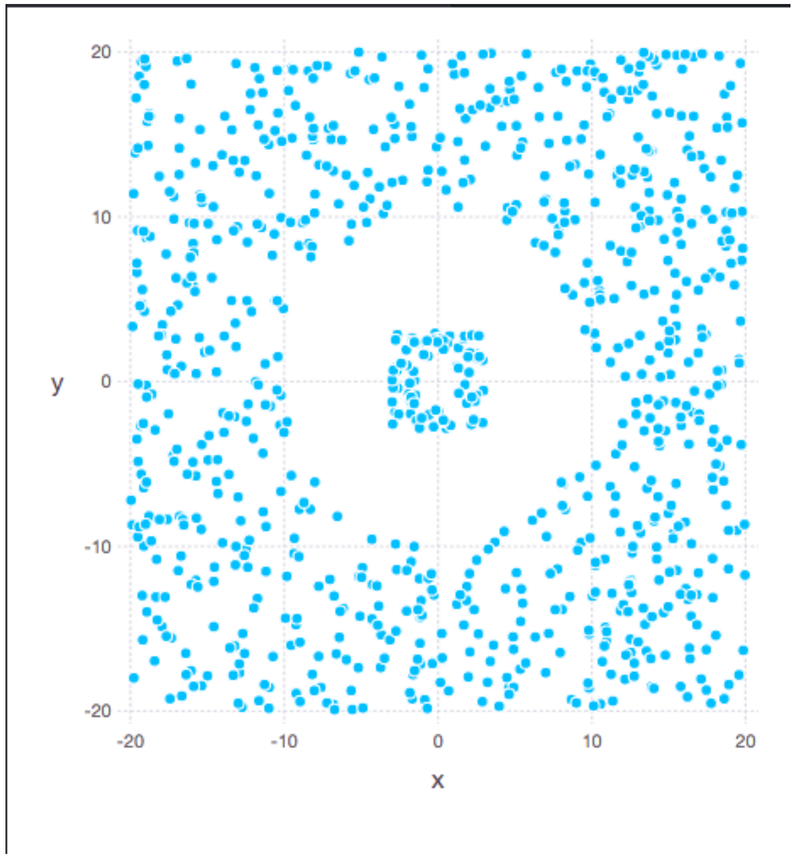
6

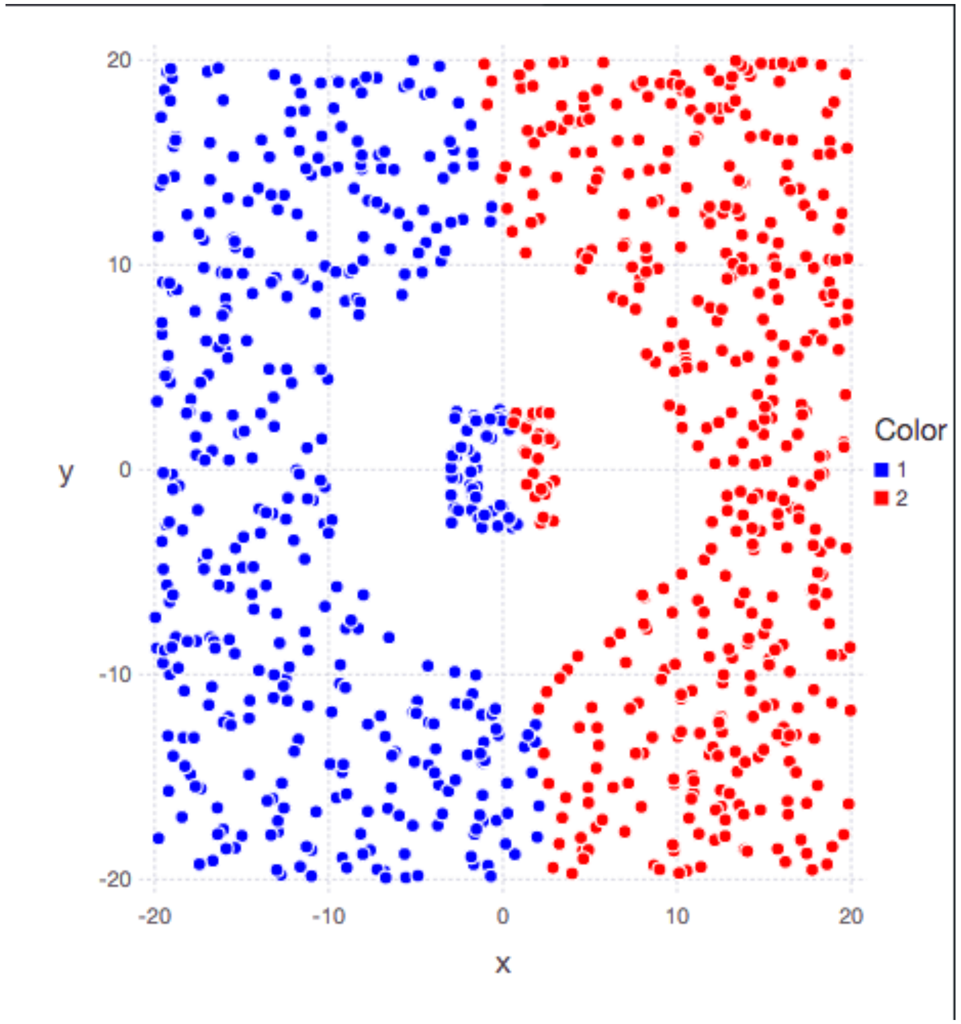


6



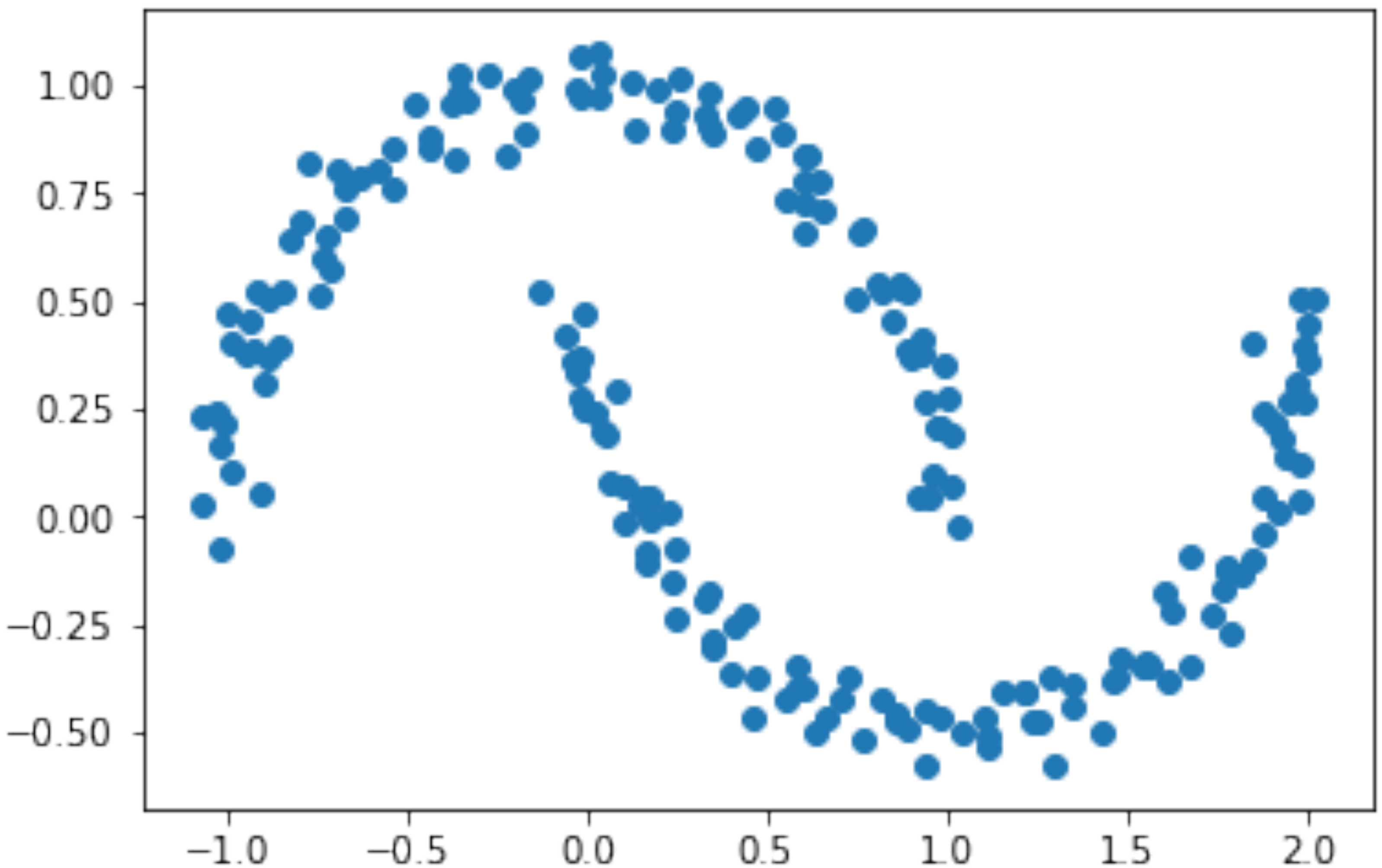
k-Means & Clusters with no center





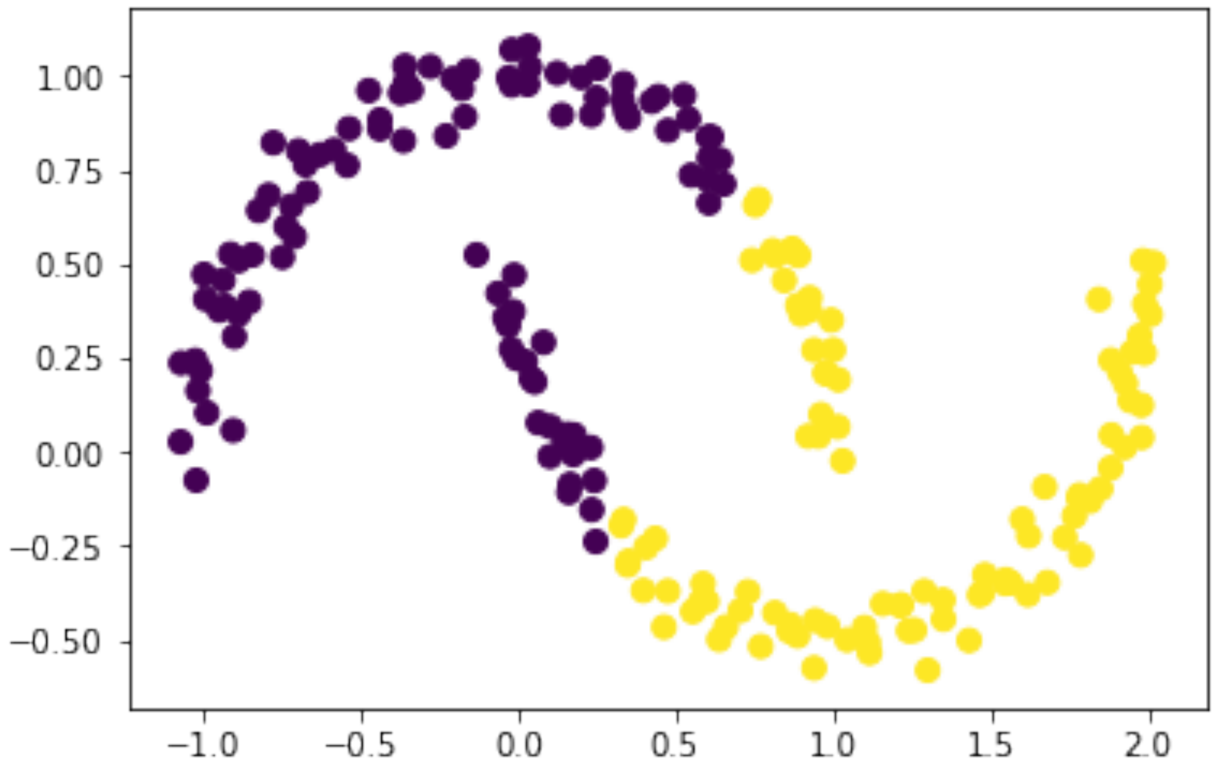
k-Means & Clusters with no center

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=.05, random_state=0)

plt.scatter(X[:, 0], X[:, 1]);
```



k-Means & Clusters with no center

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,
            s=50, cmap='viridis');
```



k-clustering Algorithms

Assume that

Each cluster is centered around a point

Clusters are convex

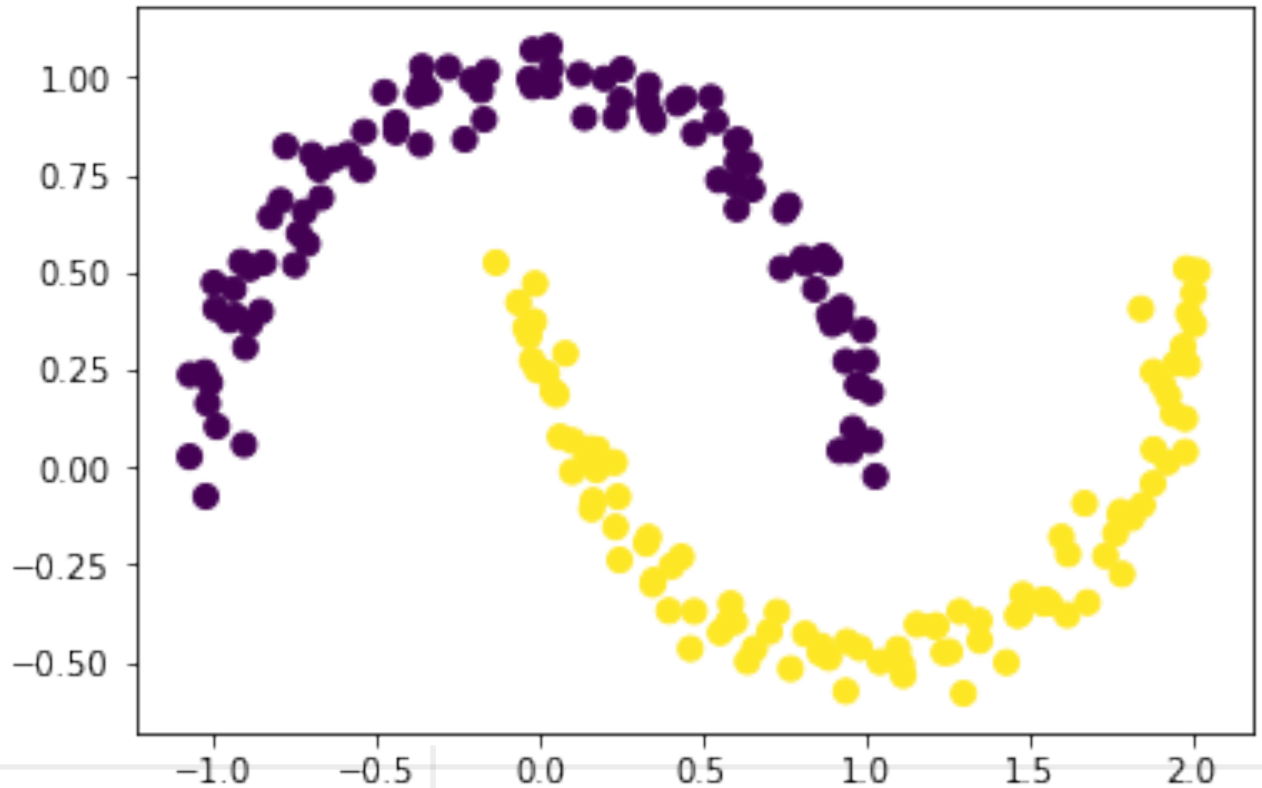
You know how many clusters there should be

SpectralClustering

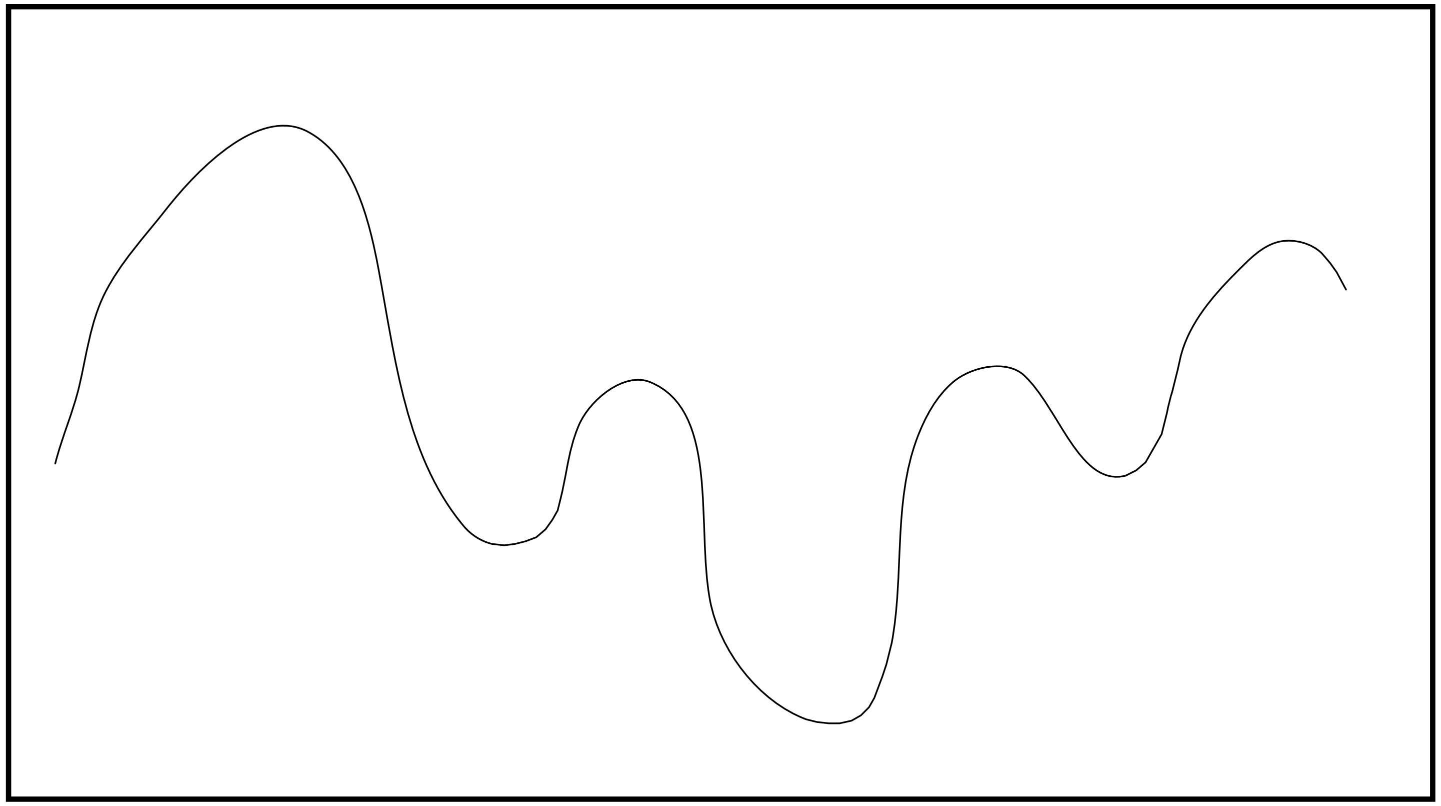
Transforms data then uses K-menas

useful when the structure of the individual clusters is highly non-convex

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',
                           assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,
            s=50, cmap='viridis');
```



Picking initial Seeds for Clusters



Clustering algorithms try to find the best clusters

But can get stuck in local extrema

DBSCAN

Density-based spatial clustering of applications with noise

Groups points together that are closely packed together

Developed in 1996

One of most commonly used clustering algorithms

Most cited in scientific literature

Terms

Parameters ϵ- distance

minPts

p is a core point if

There are minPts within distance ϵ of p including p

Directly reachable points

All points within distance ϵ of a core point p are directly reachable from p

q is reachable from p if

There is a path $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = q$,

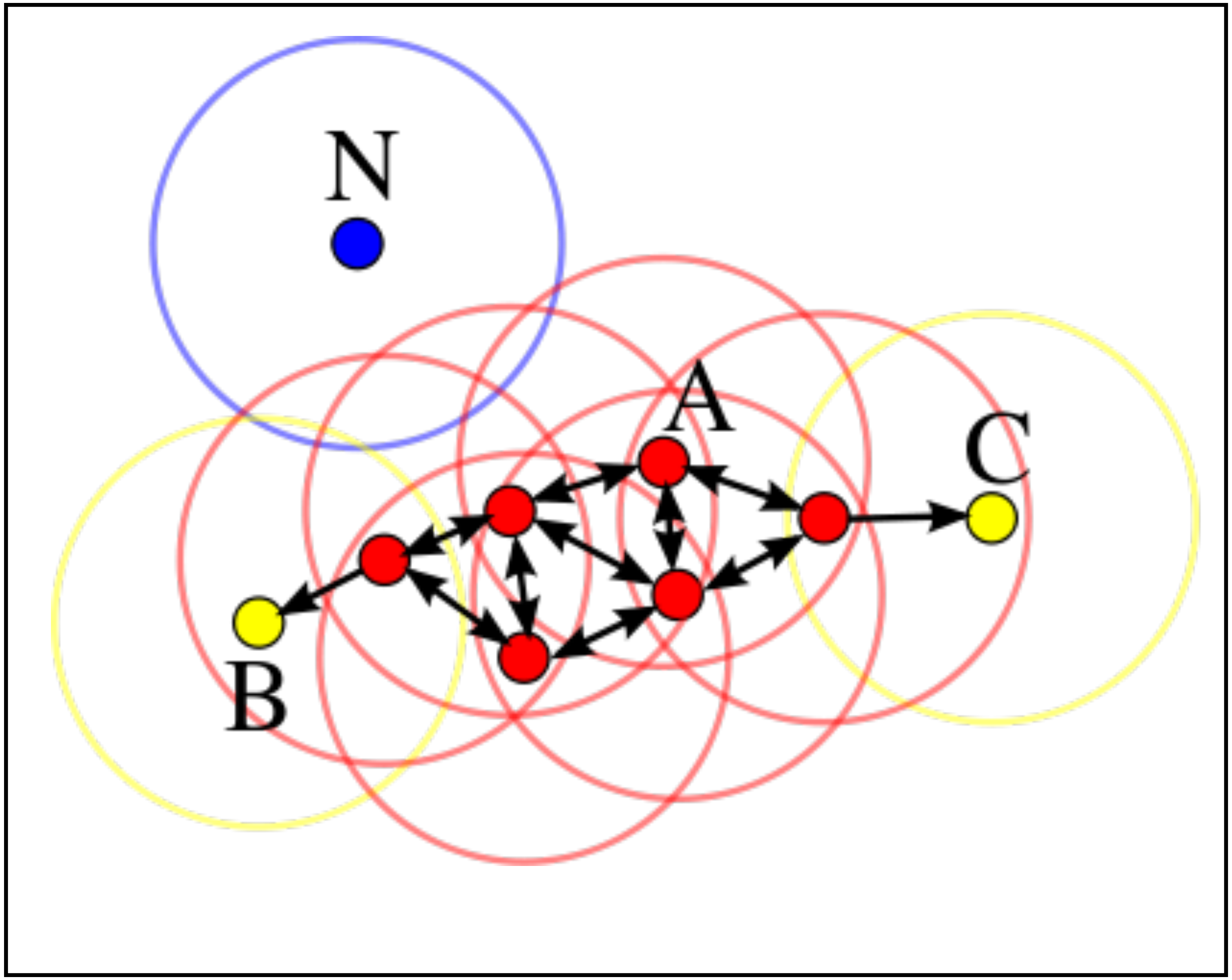
p_{i+1} is directly reachable from p_i

Outlier

Points not reachable from any other points

A core point and all points reachable from it form a cluster

Example - minPts = 4



DBSCAN Issues

ϵ & minPts determine the clusters

No need to determine number of clusters

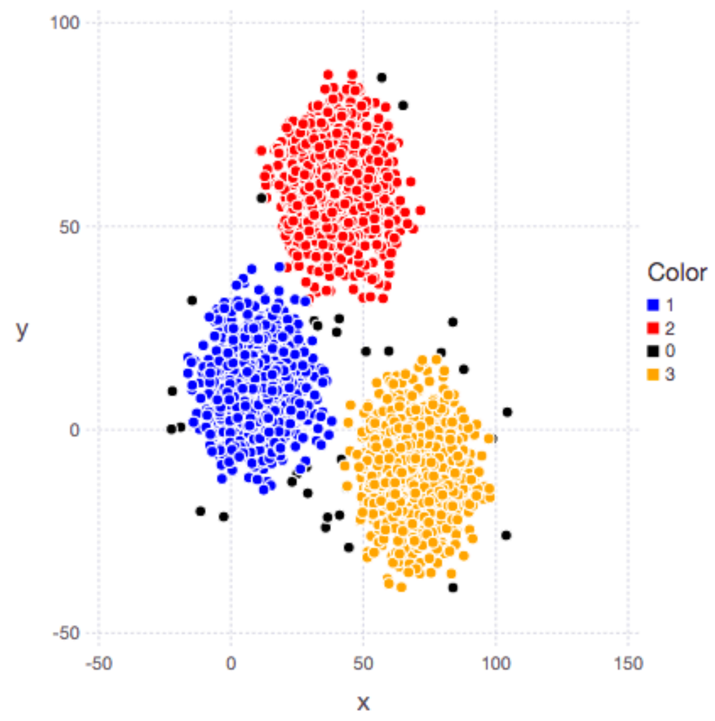
Robust to outliers

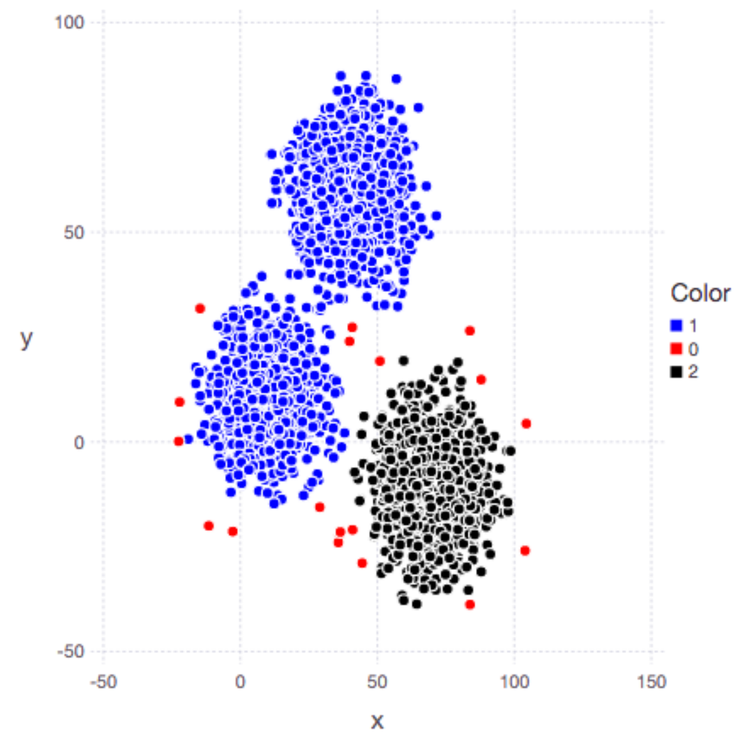
Can be implemented with runtime O(n log n)

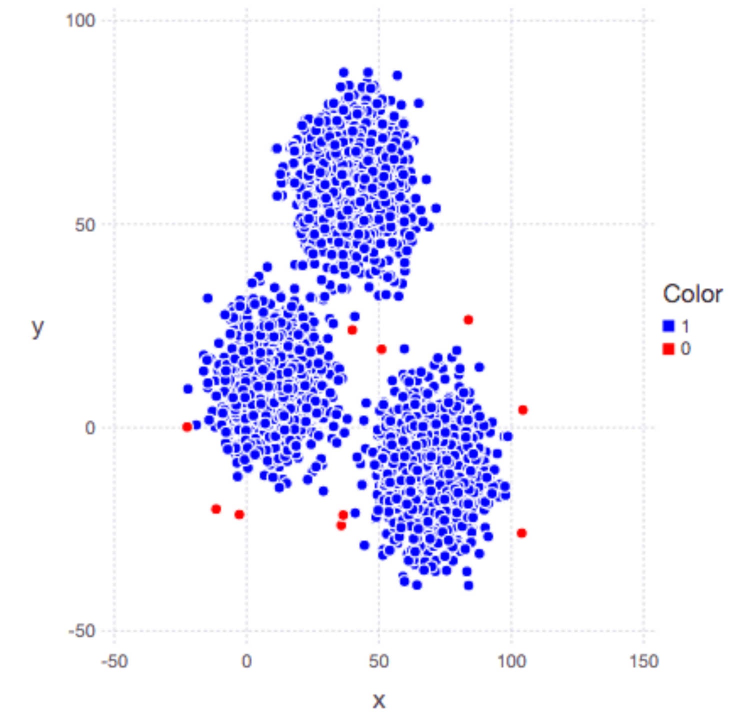
Can not handle data with varying densities

High demensional data causes problems with selecting ϵ & minPts

DBSCAN with varying eps





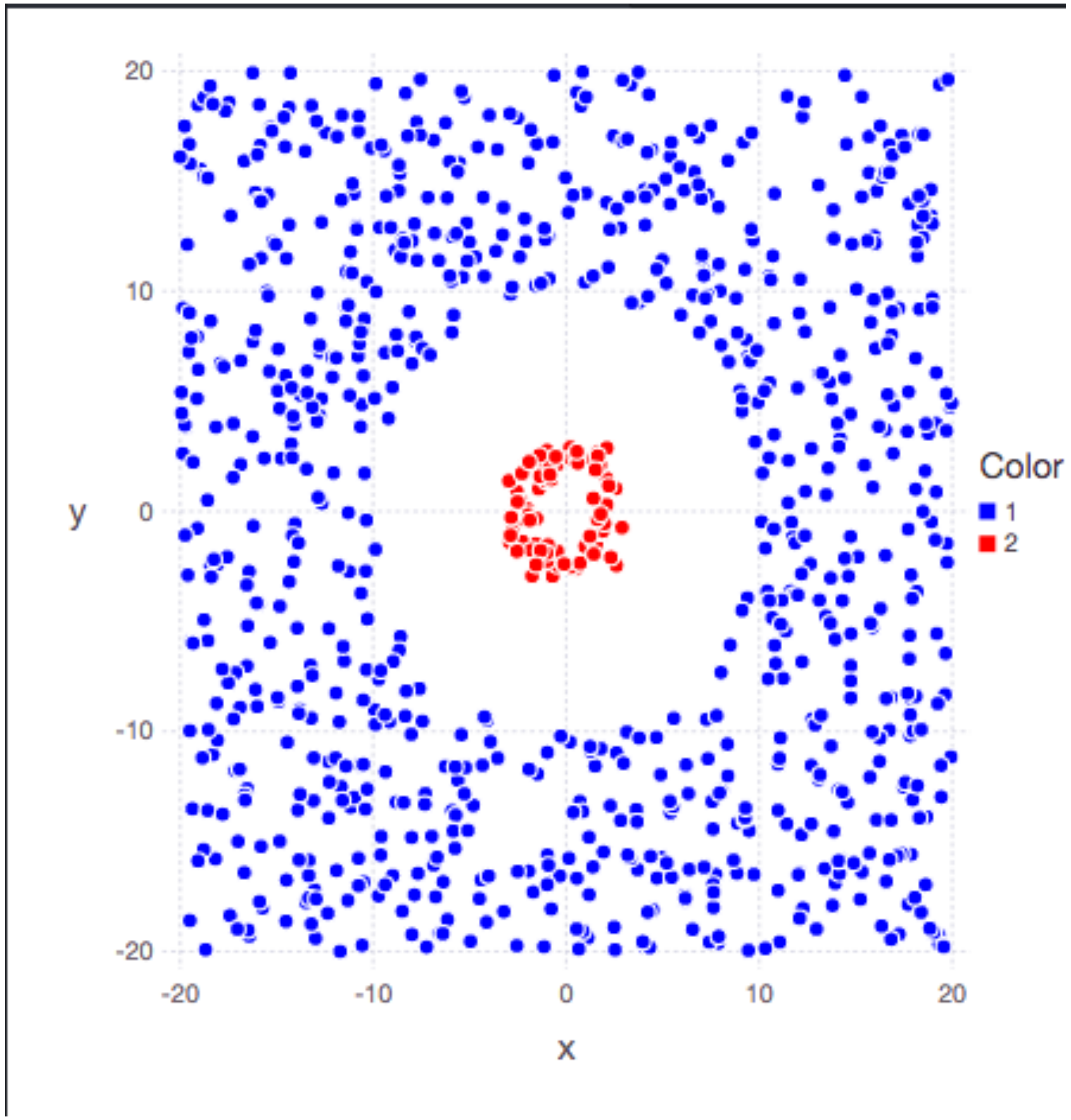


eps = 6
minpts = 10

eps = 7
minpts = 10

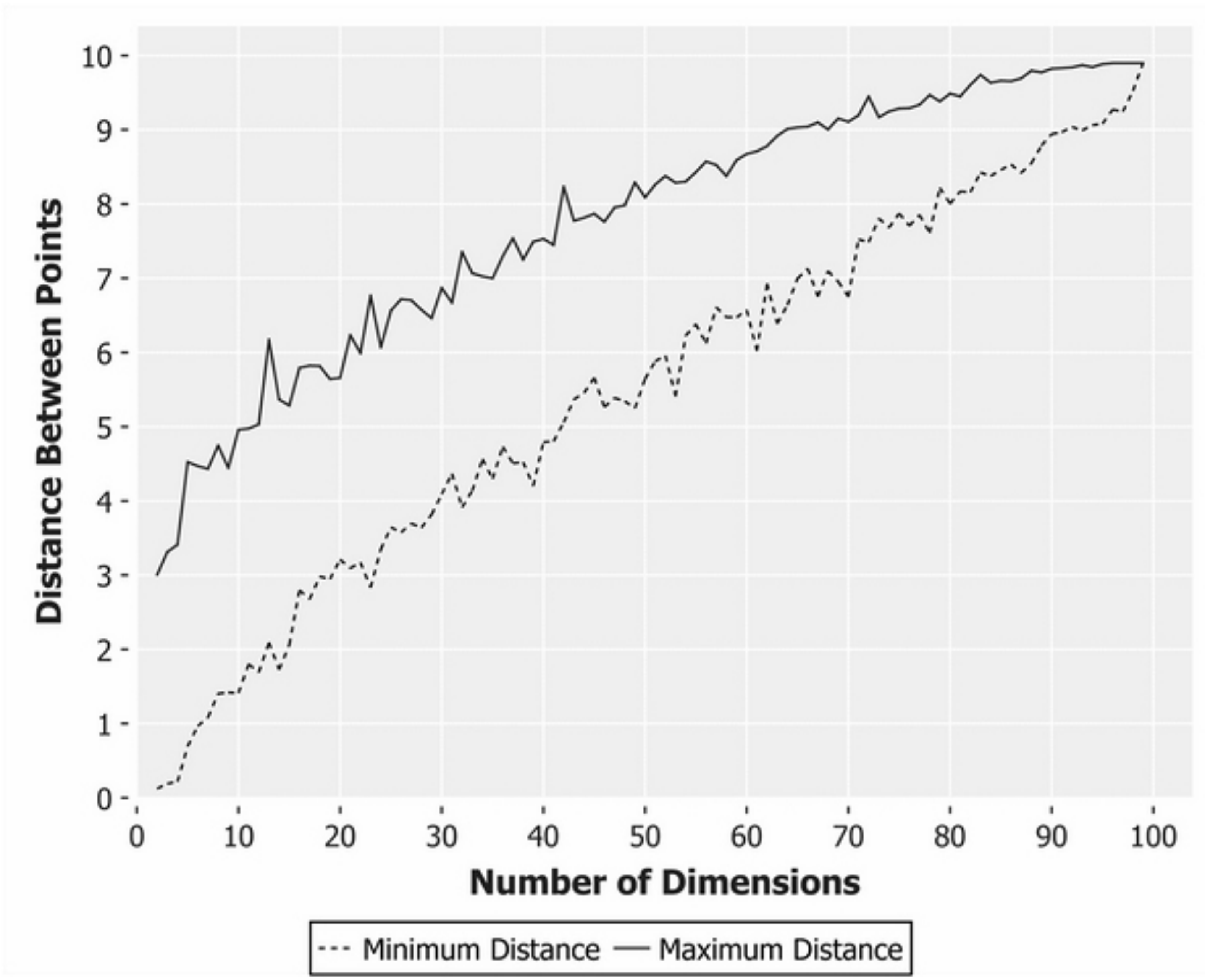
eps = 8
minpts = 10

DBSCAN & Non centered clusters



Curse of Dimensionality

As dimensions rise every point tends to become equally far from every other point

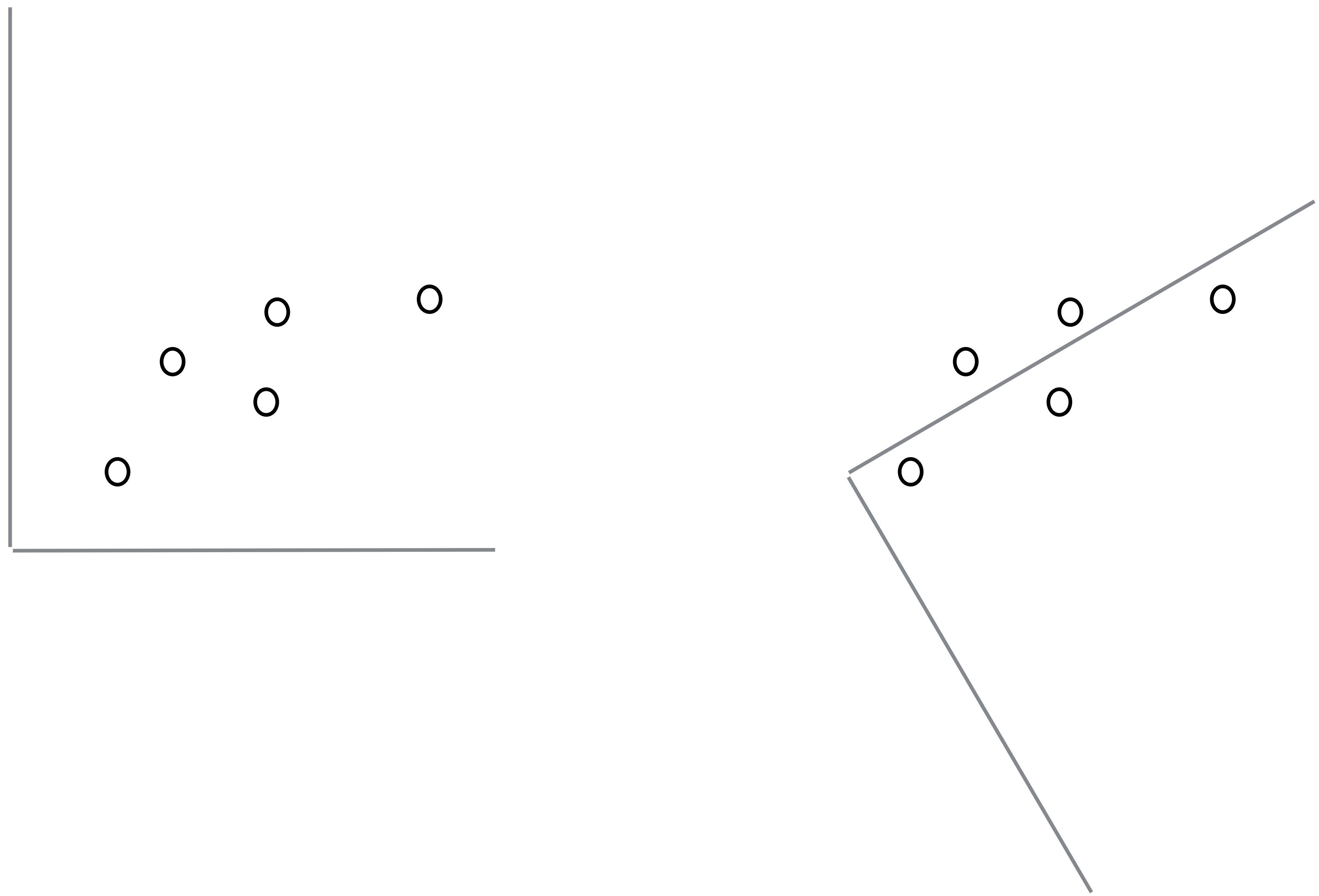


Reducing Dimensions

Some dimensions in a data set have less variation that others

So contribute less

These dimensions may not be the ones given in the data



PCA - Principle Component Analysis

Used to reduce the dimensionality of data

Changes the dimension of the data so

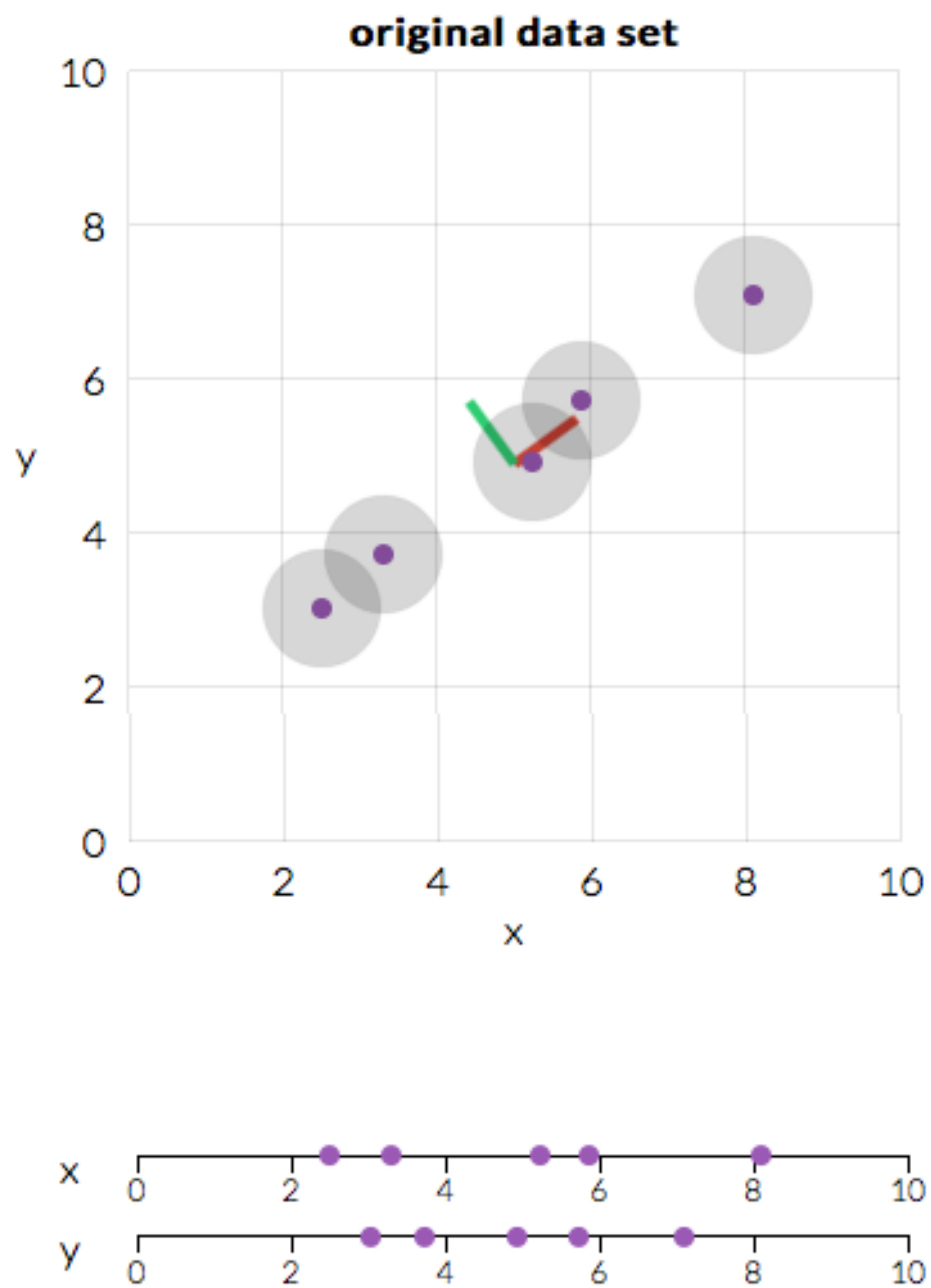
- First dimension has the greatest variance
- Second dimension has second greatest variance
- ...

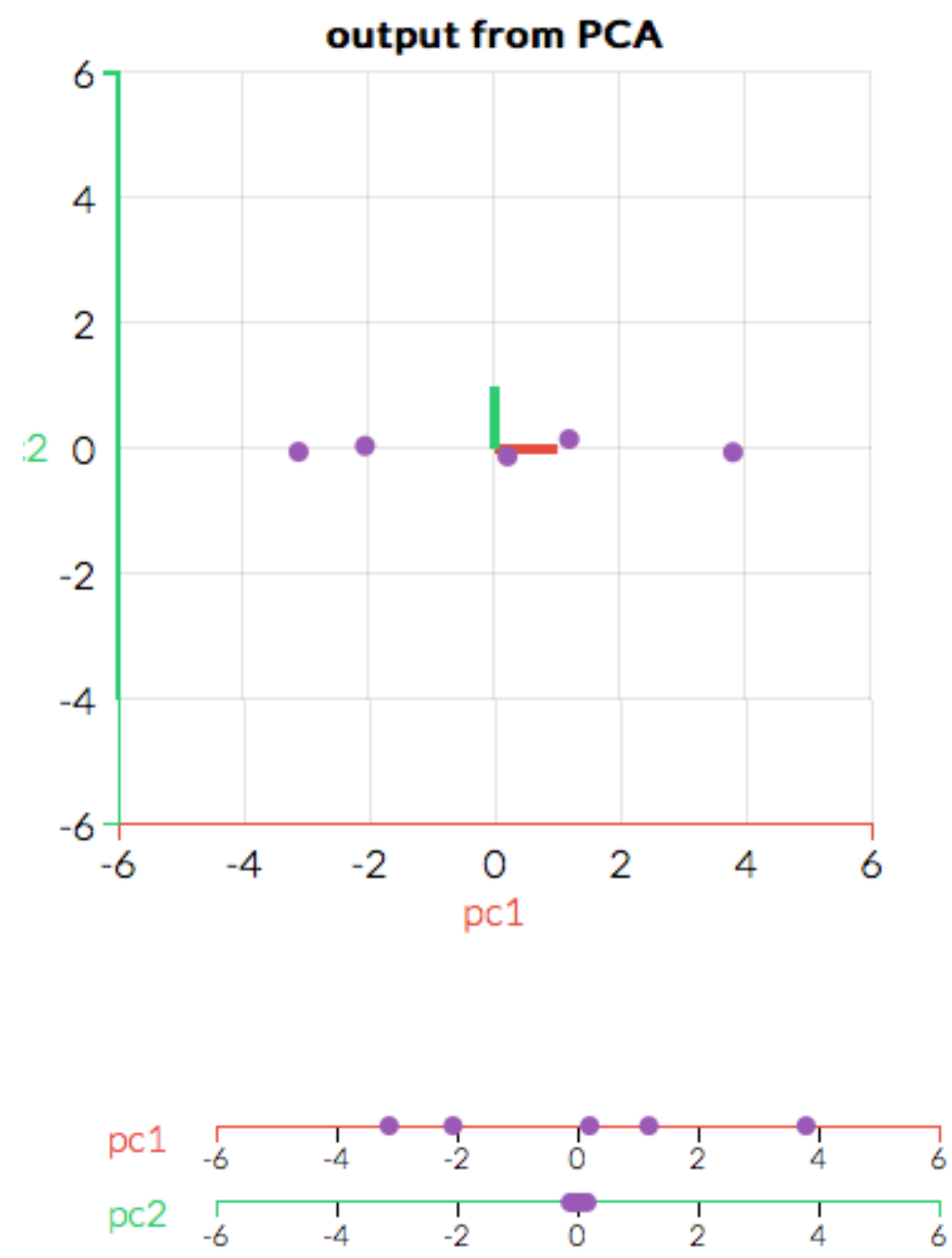
Can then select first K dimensions to work with

Data is transformed into different coordinate system

Example

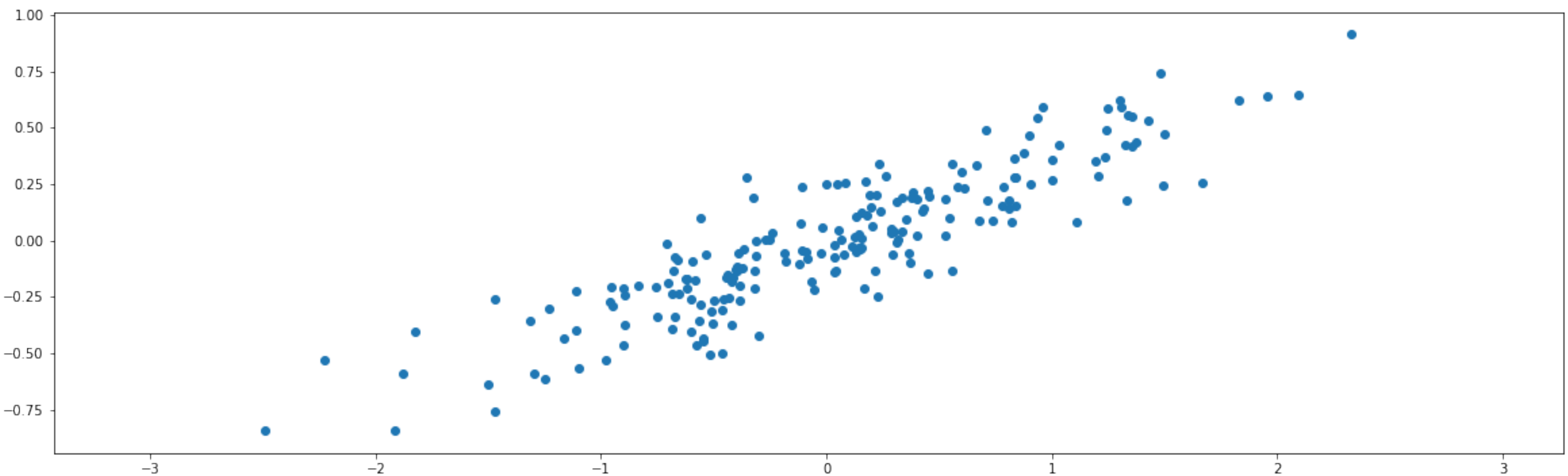
http://setosa.io/ev/principal-component-analysis/





Example - Generate Data

```
import numpy as np
from matplotlib import pyplot as plt
plt.figure(figsize=(20,6))
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
```



Example - Compute PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

Vector of two Components

```
print(pca.components_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

How much variation on each axis

```
print(pca.explained_variance_)
```

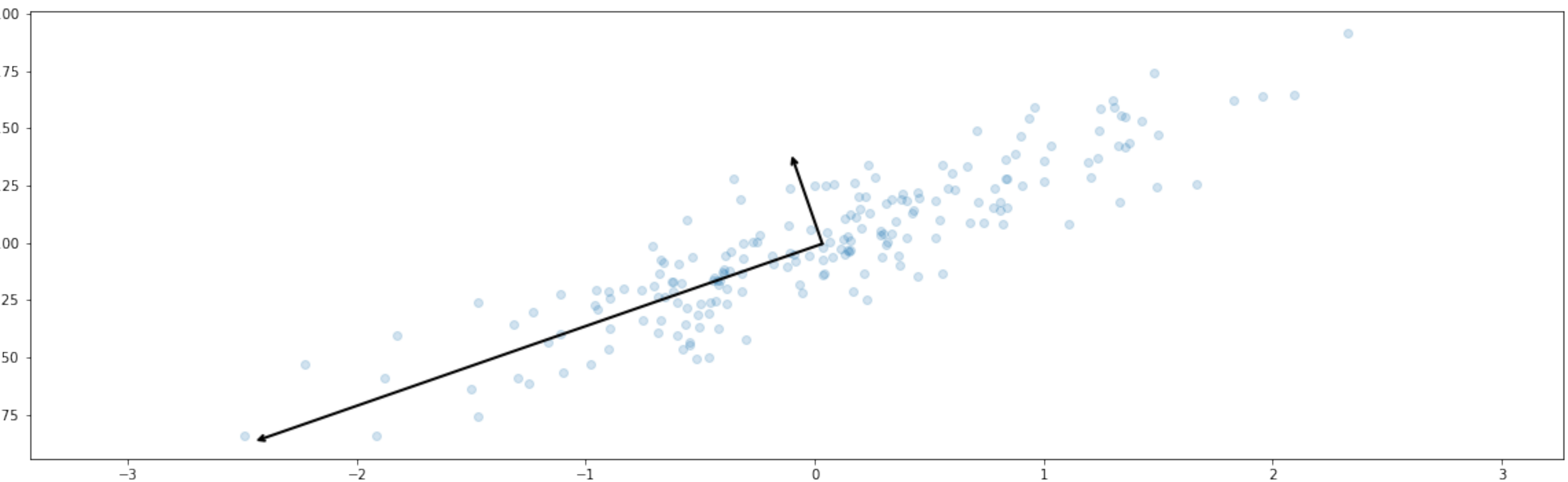
```
[0.7625315 0.0184779]
```

Center of Data

```
print(pca.mean_)
```

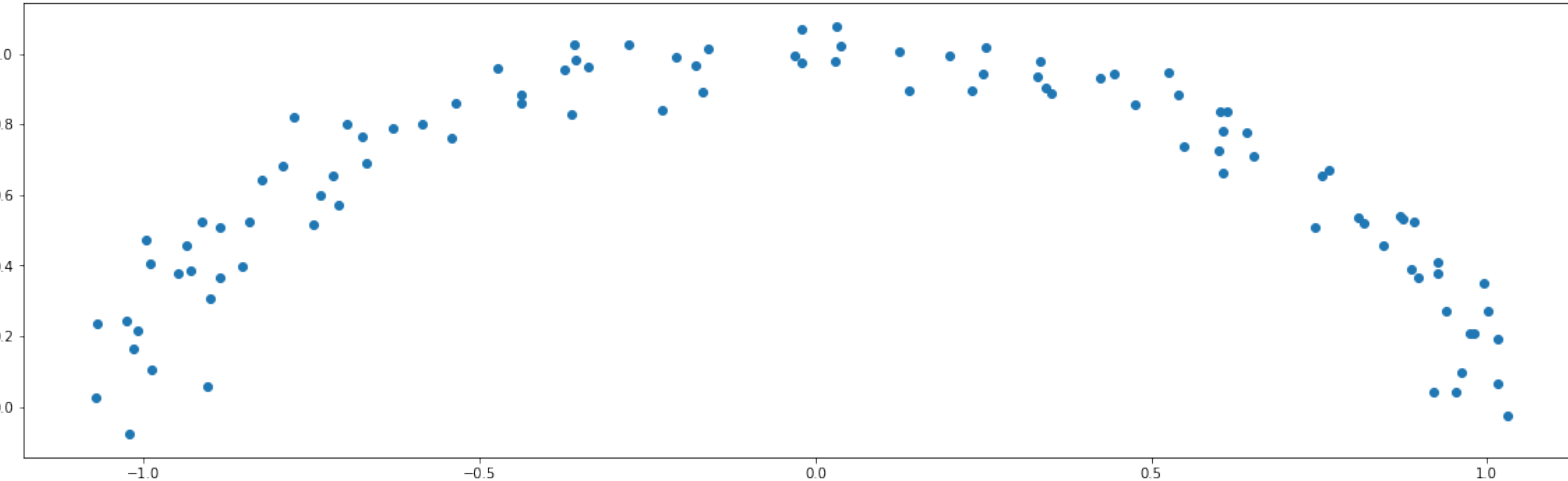
```
[ 0.03351168 -0.00408072]
```

New Axis

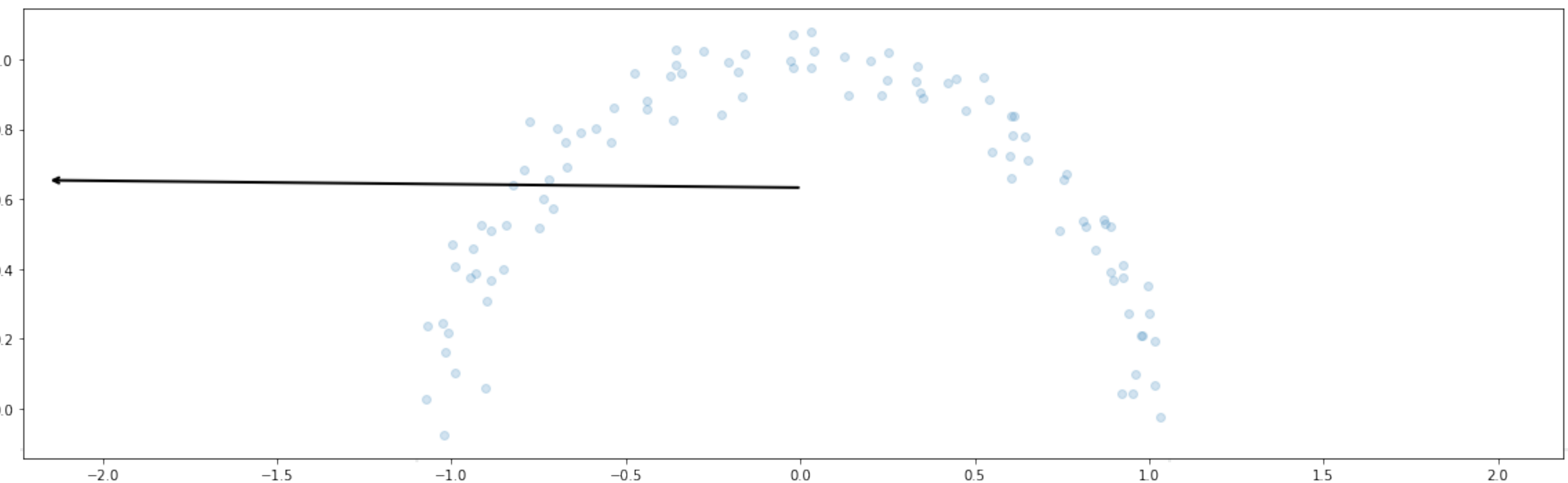


If we project all data on the long axis

- 1 dimensional data
- 76% of variation



How much variation on each axis [0.51123202 0.09867101]



Drawing Vector

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.figure(figsize=(20,6))
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```

Creating one Moon

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=.05, random_state=0)
moon = X[y == 0]
plt.figure(figsize=(20,6))
plt.scatter(moon[:, 0], moon[:, 1]);
```

```
from sklearn.decomposition import PCA
pca_moon = PCA(n_components=2)
pca_moon.fit(moon)
print(pca_moon.explained_variance_)
```