Mining Massive Datasets

Lecture 2

Artur Andrzejak, Diego Costa

http://pvs.ifi.uni-heidelberg.de

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Note on Slides

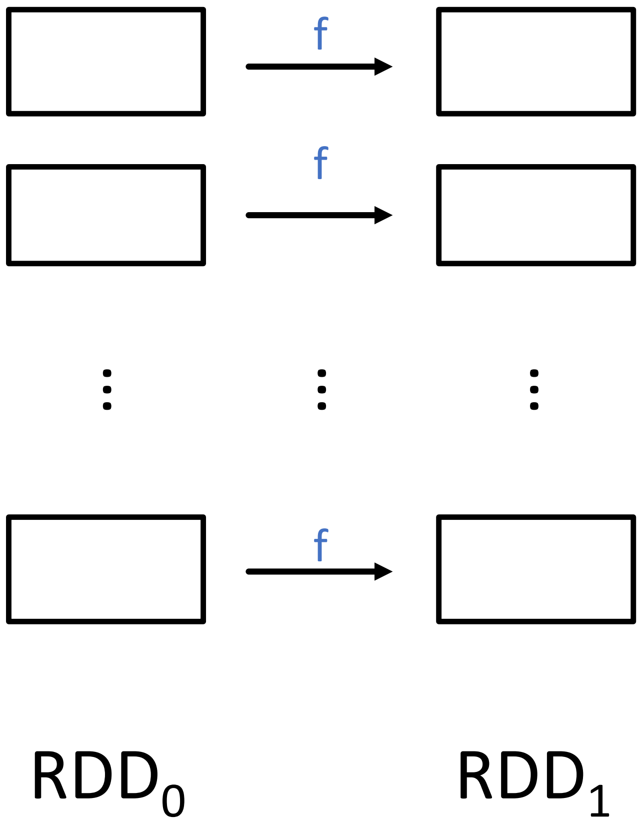
A substantial part of these slides come (either verbatim or in a modified form) from the book
Mining of Massive Datasets
by Jure Leskovec, Anand Rajaraman, Jeff Ullman
(Stanford University).
For more information, see the website accompanying the book: http://www.mmds.org.

Spark Programming: Introduction

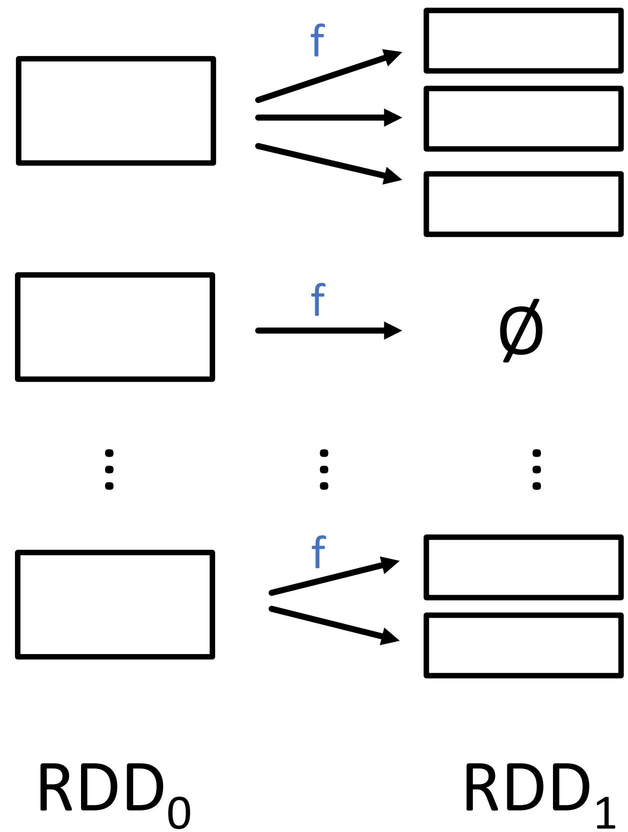
… continued

Essential Transformations

map with f:val -> val

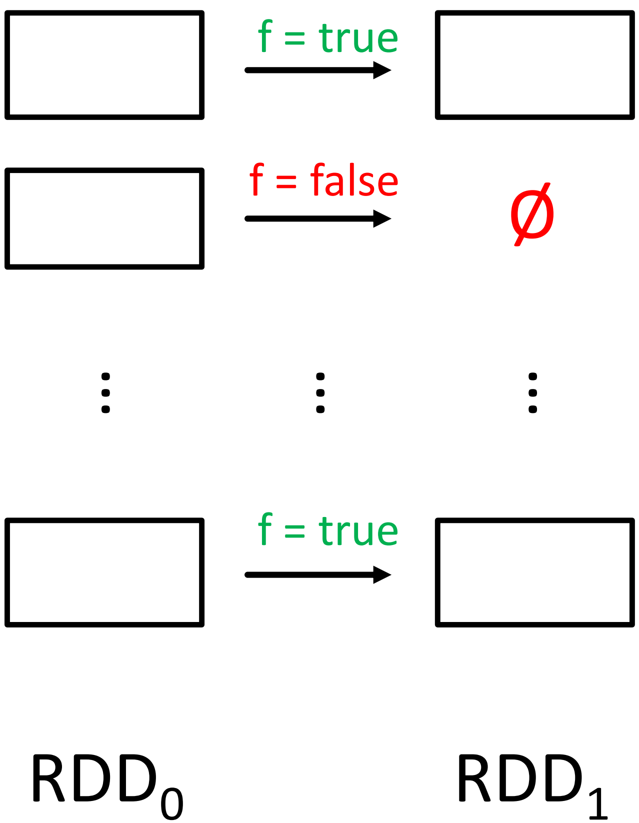


flatMap with
f:val -> [val,…,val]



Essential Transformations

filter with f:val -> boolean

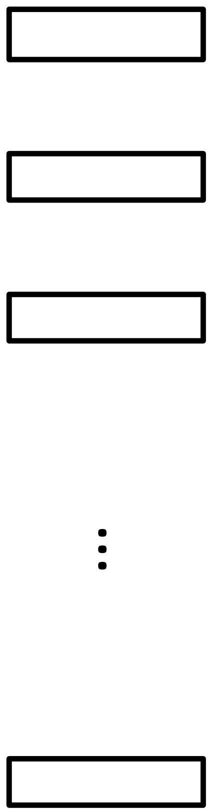


Others (useful):

- union
- distinct
- join
- groupByKey …

„Reduce"-Action

reduce with f: val, val -> val



Other RDD Operators

- `map`
- `filter`
- `groupBy`
- `sort`
- `union`
- `join`
- `leftOuterJoin`
- `rightOuterJoin`
- `reduce`
- `count`
- `fold`
- `reduceByKey`
- `groupByKey`
- `cogroup`
- `cross`
- `zip`
- `sample`
- `take`
- `first`
- `partitionBy`
- `mapWith`
- `pipe`
- `save`
- `...`

More details: spark-project.org/docs/latest/

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

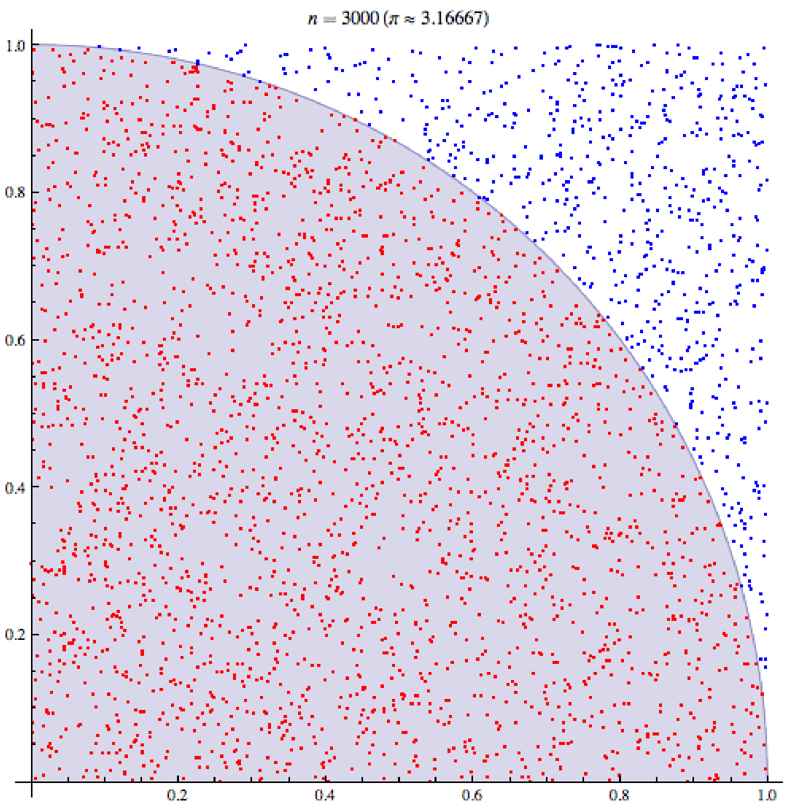
Passing Values to/from Functions

- Functions in transformations/actions are **closures**: they have read-only access to all driver variables visible at closure definition
  - Related: **broadcast** variables
- Transformations
  - NOT possible to return values to driver process
  - NO exchange of values between code executing on different records of a RDD

Spark Programming: Example

Estimating π

- We use a Monte Carlo method to estimate the value of Pi (π)
- Idea: Count the share of random points (x, y) whose distance to (0,0) is 1 or less
- Naturally parallelizable



Source: Wikipedia

Estimating Pi/4

Generates two pseudo-random numbers in [0.0, 1.0)

True iff distance of (x,y) to origin is < 1

Generates an iterable ("lazy list") from 0 to N-1

```
N = 1000000
def inCircle(p):
    x, y = random(), random()
    return 1  if x*x + y*y < 1 else 0

myplus = lambda a, b: a + b

rawDataRDD = sc.parallelize(range(0,N), partitions)
inCircleRDD = rawDataRDD.map(inCircle)
count = inCircleRDD.reduce(myplus)

print("Pi is roughly %f" % (4.0 * count / N))
```

A Stand-Alone Program

```
import sys
from random import random
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
"""Usage: pi [partitions]"""

sc = SparkContext (appName="PythonPi")
partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
N = 100000 * partitions

... [code as above, without N = 1000000] ...

sc.stop()
```

Execution on the VM

- Open terminal window
- cd spark/examples/src/main/python
- spark-submit pi.py 1
  - => Pi is roughly 3.139168
- spark-submit pi.py 2
  - => Pi is roughly 3.138352
- spark-submit pi.py 3
  - => Pi is roughly 3.142220
- spark-submit pi.py 4
  - => Pi is roughly 3.142624

Clustering: Introduction

Clustering

High dim. data
- Locality sensitive hashing
- Clustering
- Dimensio-nality reduction

Graph data
- PageRank, SimRank
- Community Detection
- Spam Detection

Infinite data
- Filtering data streams
- Web advertising
- Queries on streams

Machine learning
- SVM
- Decision Trees
- Perceptron, kNN

Apps
- Recommen der systems
- Association Rules
- Duplicate document detection

Programming in Spark & MapReduce

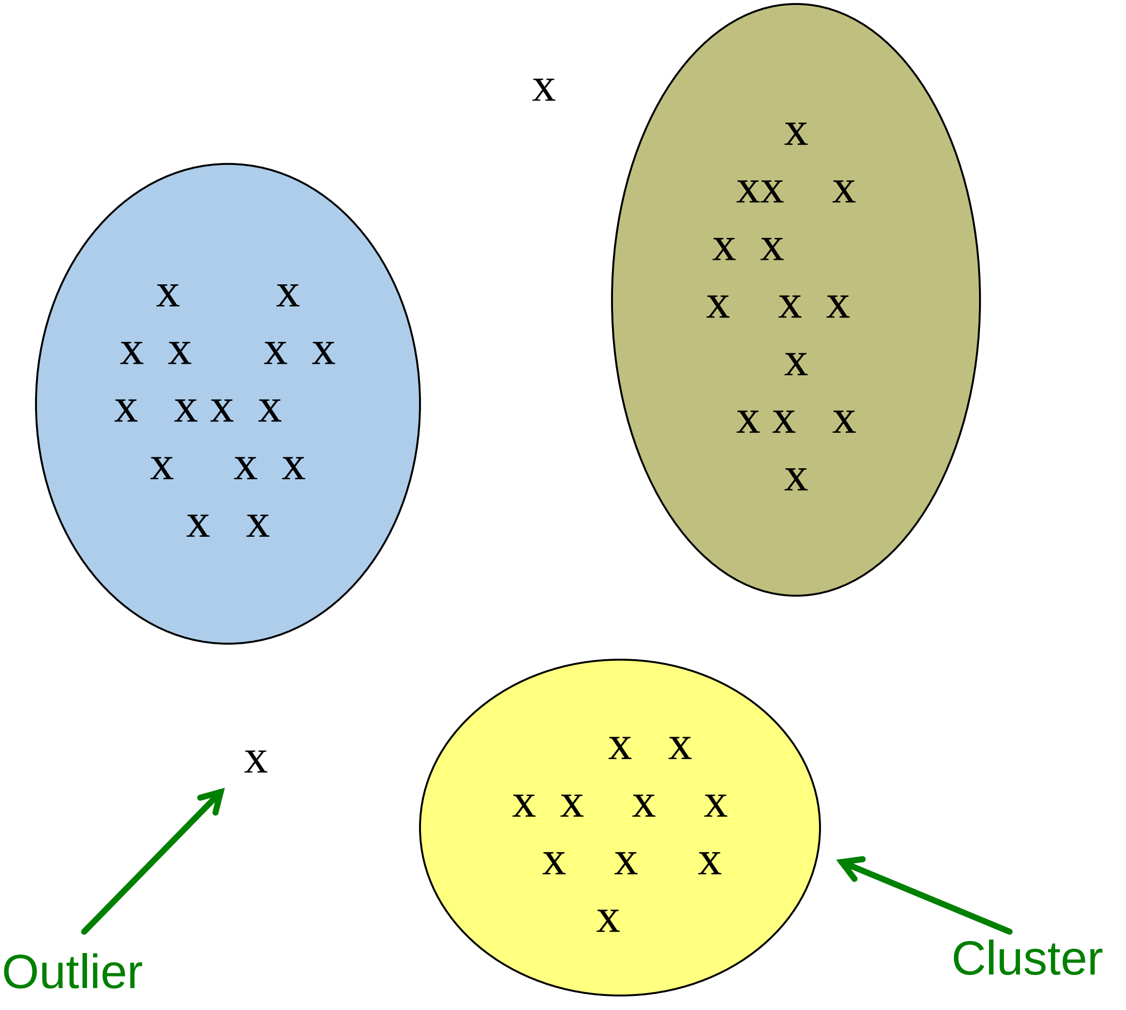
High Dimensional Data

- **Given a cloud of data points we want to understand its structure**

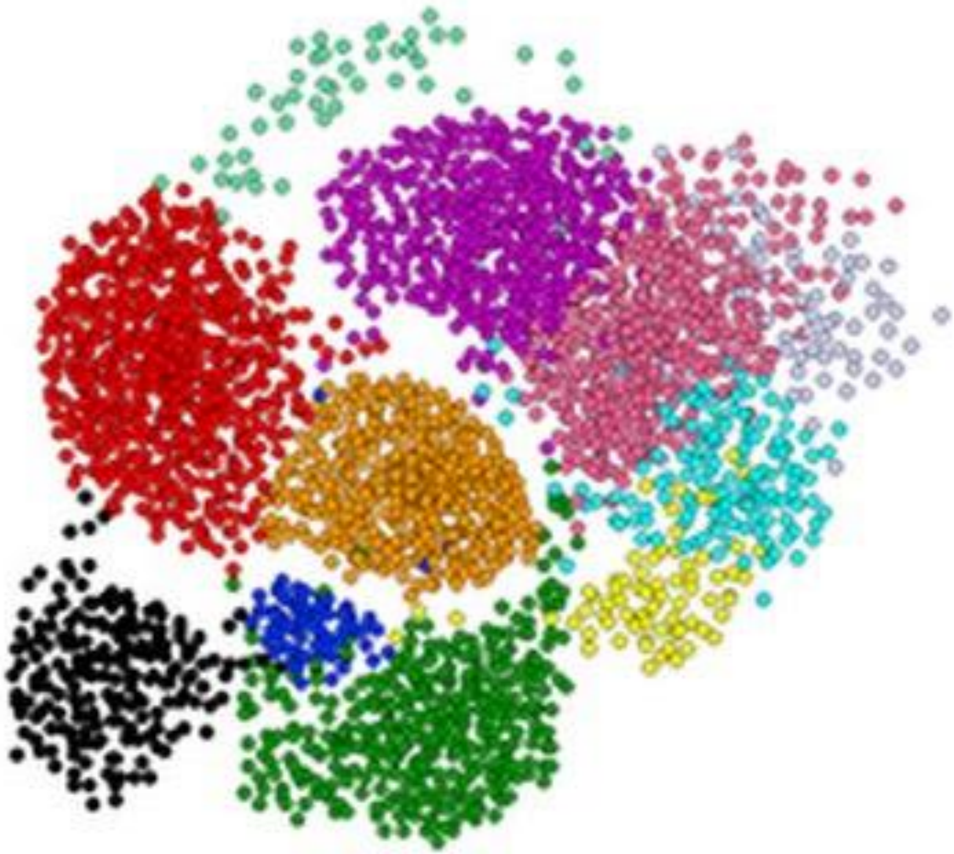
The Problem of Clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, so that
  - Members of a cluster are close/similar to each other
  - Members of different clusters are dissimilar
- **Usually:**
  - Points are in a high-dimensional space
  - Similarity is defined using a distance measure
    - Euclidean, Cosine, edit distance, …

Example: Clusters & Outliers



Clustering is a hard problem!



Why is it hard?

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance

Clustering Problem: Galaxies

- **A catalog of 2 billion "sky objects" represents objects by their radiation in 7 dimensions (frequency bands)**
- **Problem: Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.**
- **Sloan Digital Sky Survey**

Clustering Problem: Music CDs

- **Intuitively: Music divides into categories, and customers prefer a few categories**
  - But what are categories really?
- Represent a CD by a set of customers who bought it:
- Similar CDs have similar sets of customers, and vice-versa

Clustering Problem: Music CDs

Space of all CDs:

- Think of a space with one dim. for each customer
  - Values in a dimension may be 0 or 1 only
  - A CD is a point in this space $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} customer bought the CD
- For Amazon, the dimension is tens of millions
- **Task:** Find clusters of similar CDs

Clustering Problem: Documents

Finding topics:

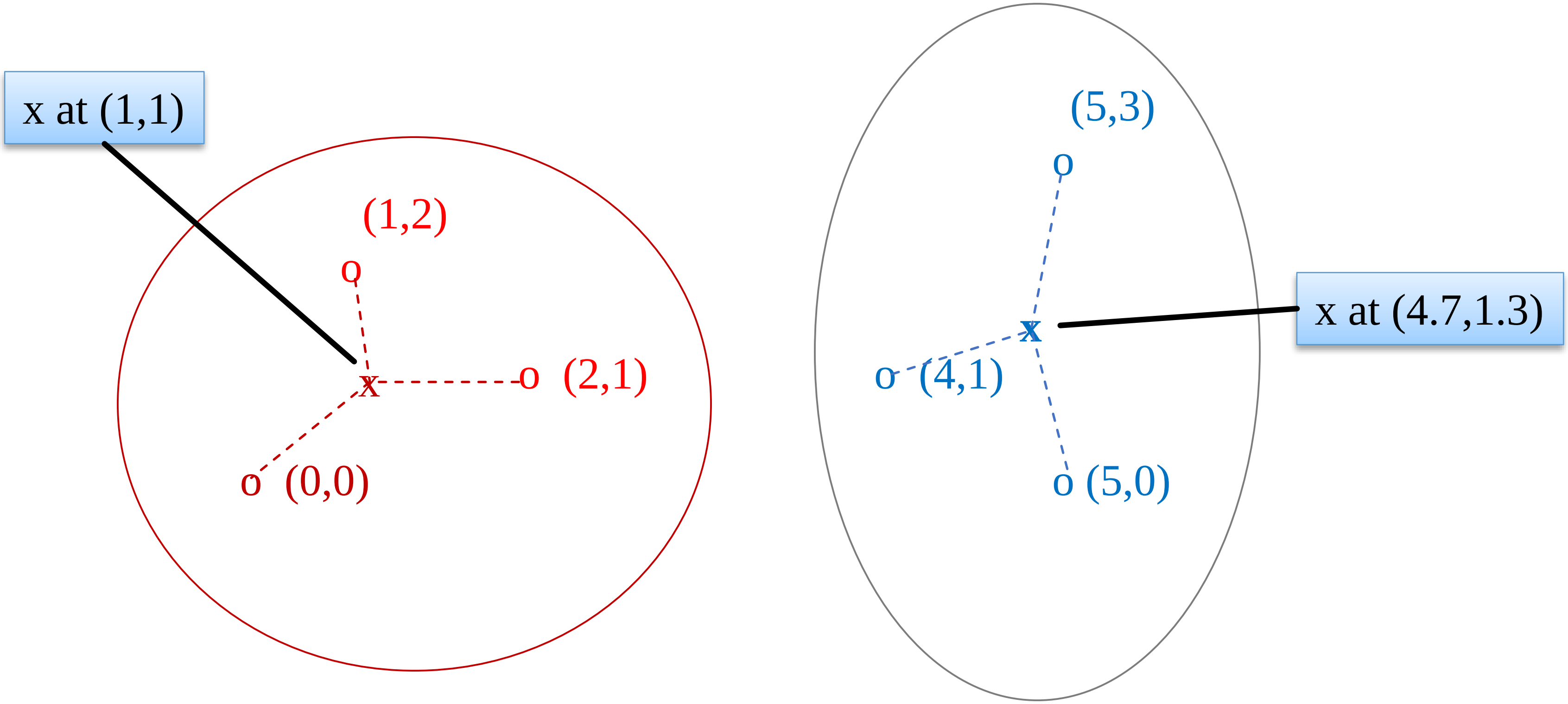
- Represent a document by a vector $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
  - It actually doesn't matter if k is infinite; i.e., we don't limit the set of words

- **Documents with similar sets of words may be about the same topic**

Centroids

- **How to represent a cluster of many points?**
  - **Key problem:** How do you represent the "location" of each cluster, to tell which pair of clusters is closest?
  - **Euclidean case:** each cluster has a **centroid** = average of its (data) points

Example: Centroids



Data:
o … data point
x … centroid

And in the Non-Euclidean Case?

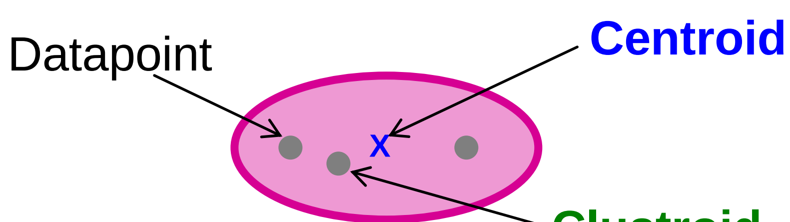
What about the Non-Euclidean case?

- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

- **Approach:**
  - **clustroid** = (data)point "**closest**" to other points in the cluster

"Closest" Point?

- **How to represent a cluster of many points?**
  clustroid = (data)point "**closest**" to other points
- **Possible meanings of "closest":**
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points
    - For distance metric ***d*** clustroid ***c*** of cluster ***C*** is: $\min_{c} \sum_{x \in C} d(x,c)^2$



Cluster on
3 datapoints

Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.
Clustroid is an **existing** (data)point that is "closest" to all other points in the cluster.

k-Means Clustering

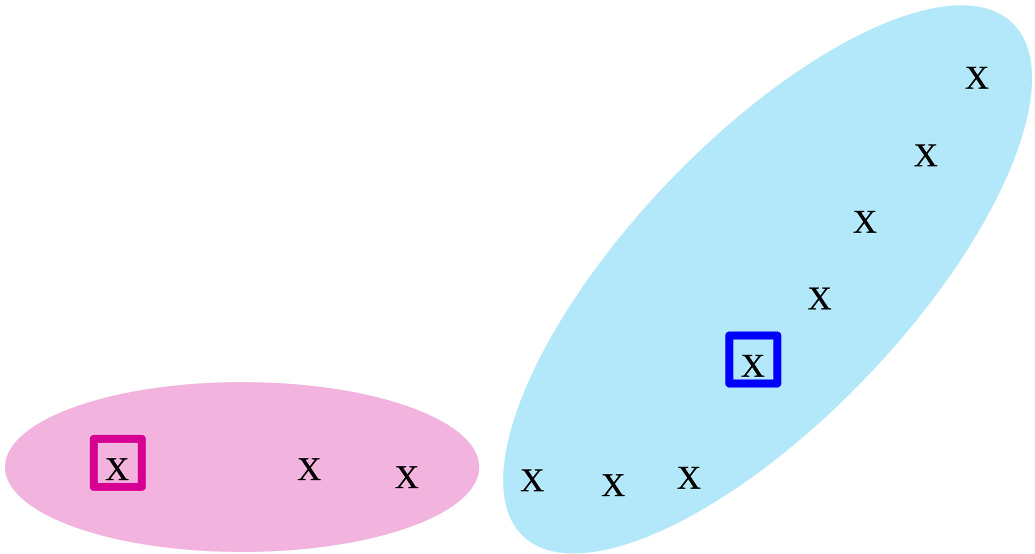
k–means Algorithm(s)

- Assumes Euclidean space/distance
- Start by picking ***k***, the number of clusters
- Initialize clusters by picking one point per cluster
  - **Example:** Pick one point at random, then ***k*-1** other points, each as far away as possible from the previous points

Populating Clusters

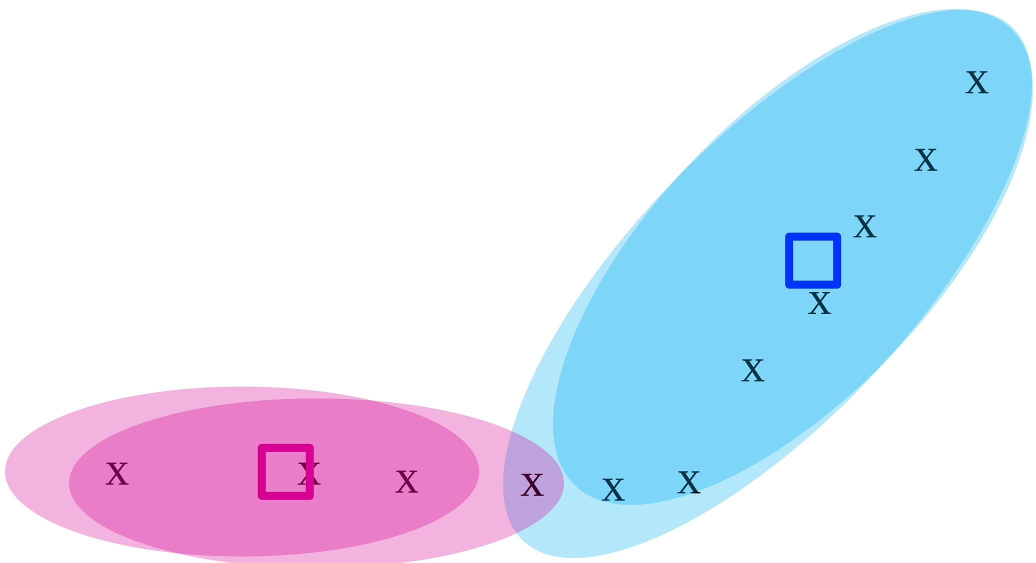
- **1)** For each point, place it in the cluster whose current centroid it is nearest
- **2)** After all points are assigned, update the locations of centroids of the ***k*** clusters
- **Repeat 1 and 2 until convergence**
  - **Convergence:** Points don't move between clusters and/or centroids stabilize
    - "stabilize":
      - e.g. sum of (squared) centroid changes < threshold

Example: Assigning Clusters



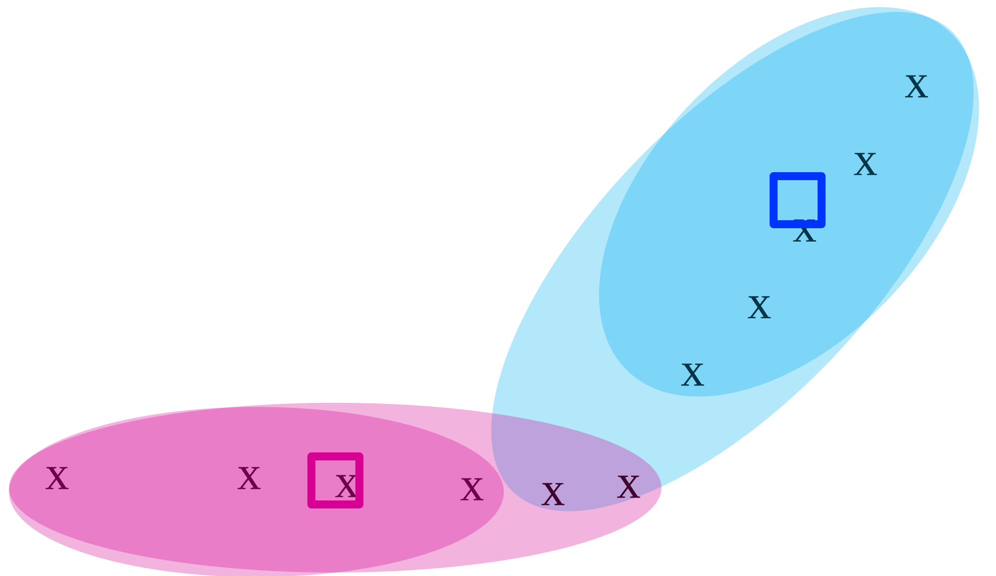
x … data point
… centroid

Example: Assigning Clusters



x … data point
… centroid

Example: Assigning Clusters



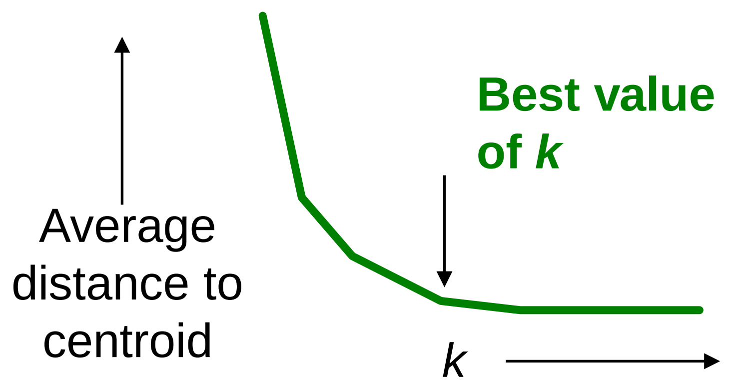
x … data point
… centroid

Clusters at the end

Getting the *k* right

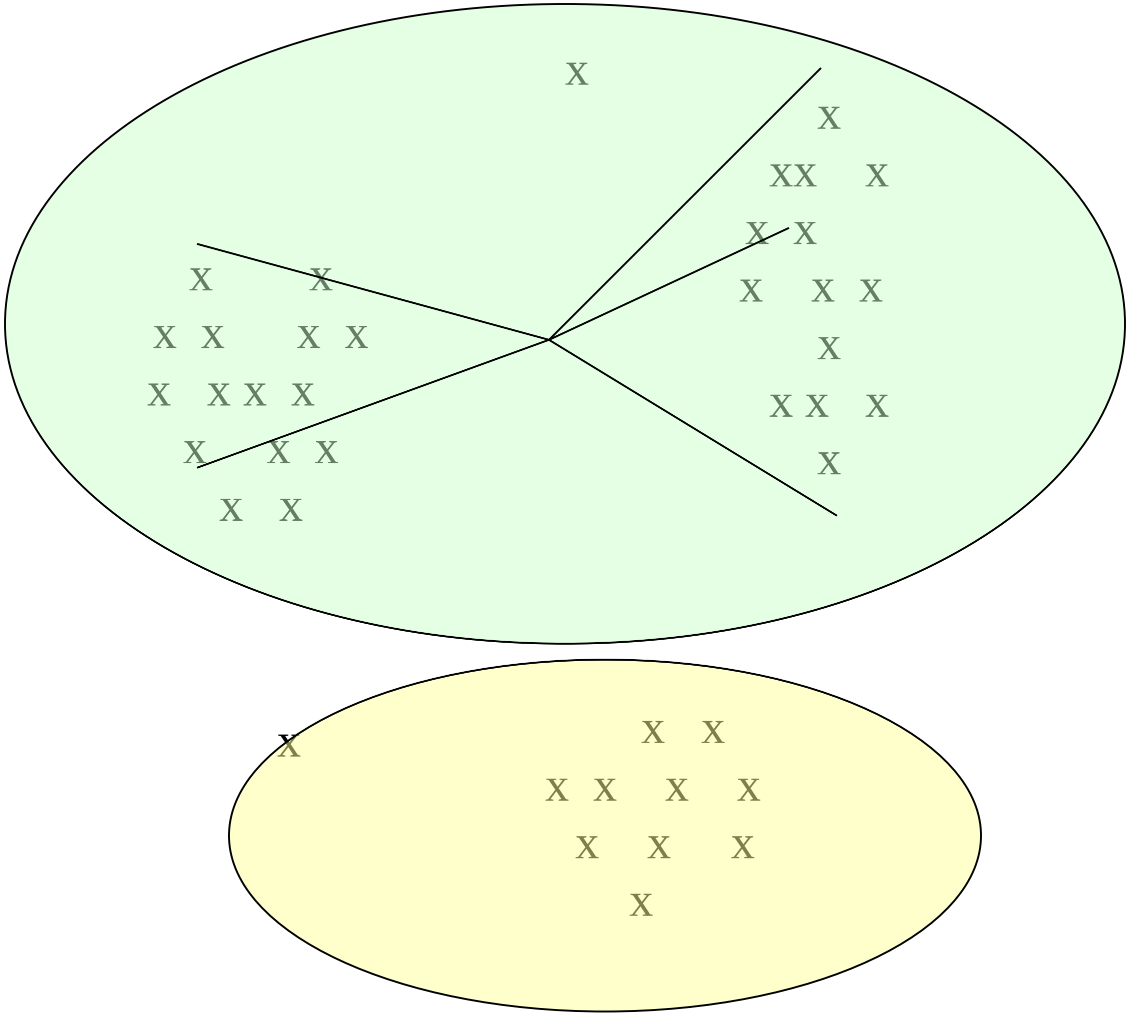
How to select *k*?

- Try different **k**, looking at the change in the average distance to centroid as **k** increases
- Average falls rapidly until right **k**, then changes little



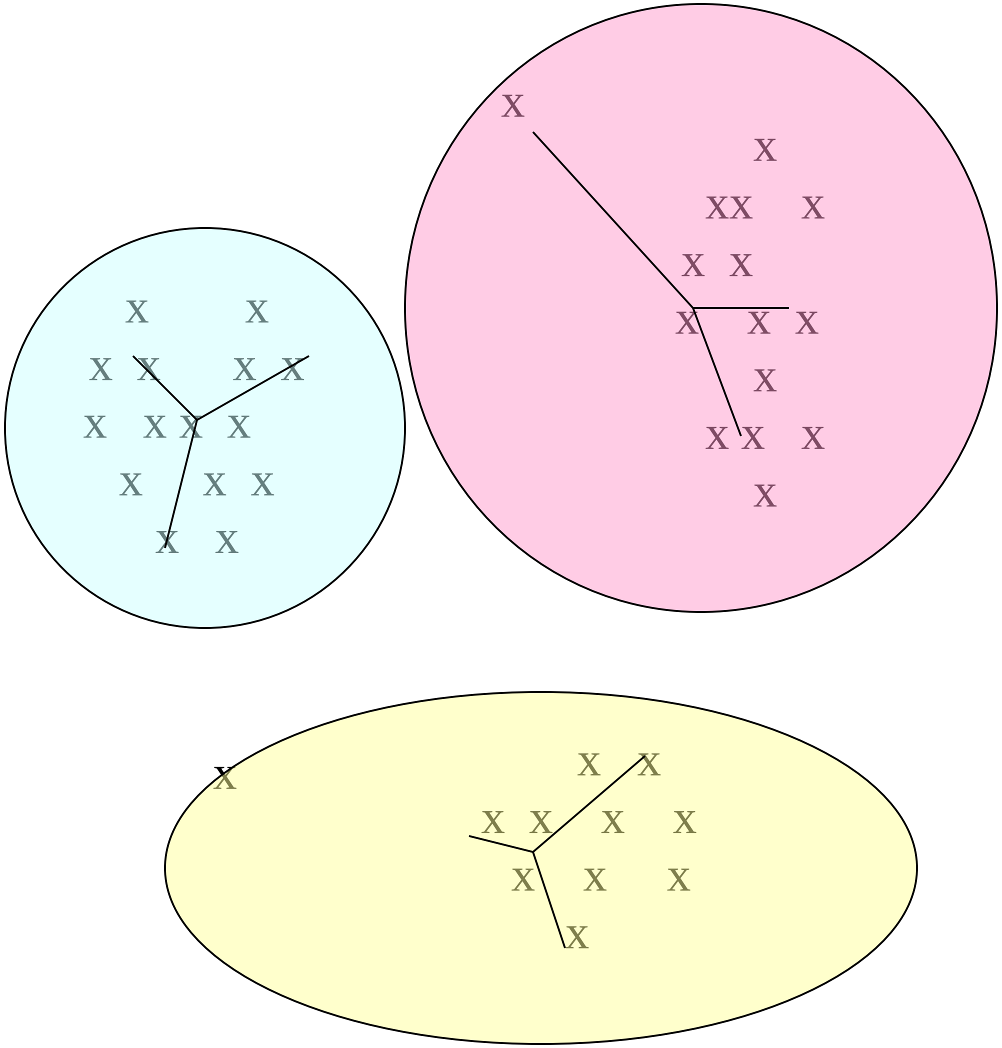
Example: Picking *k*

Too few;
many long
distances
to centroid



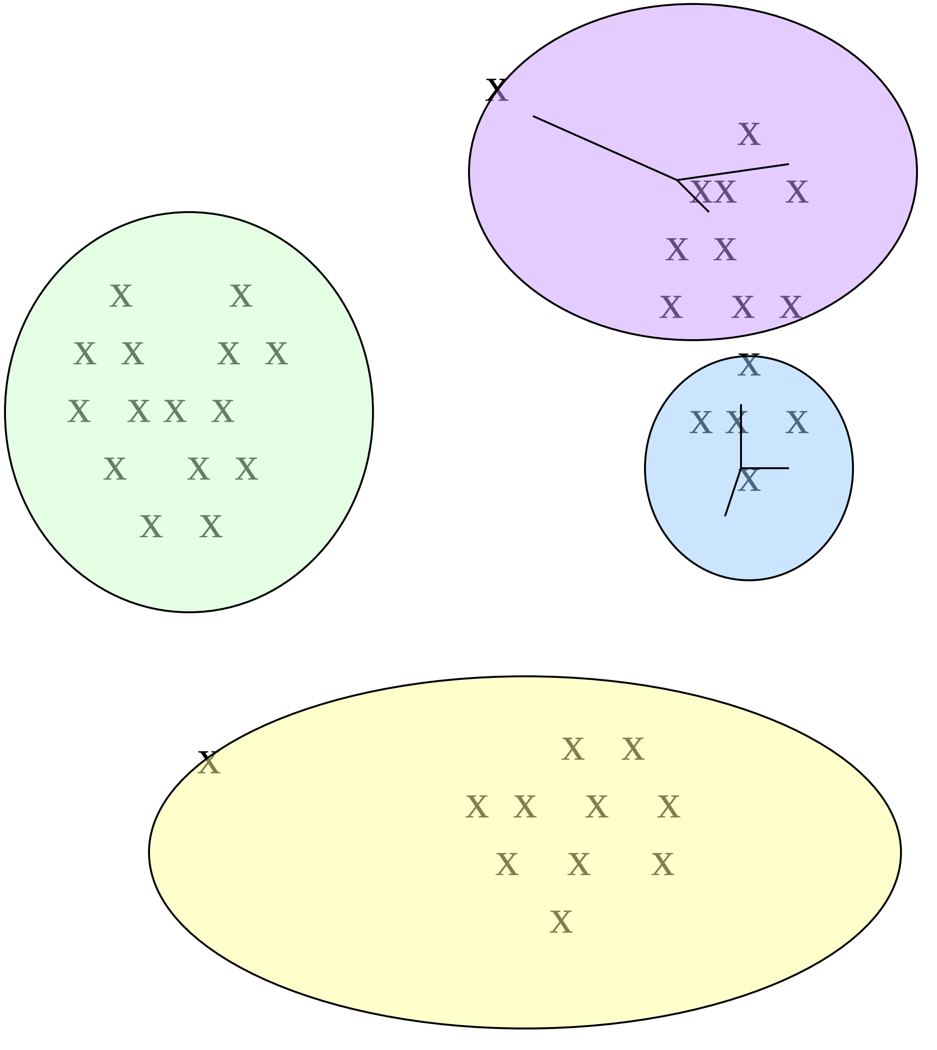
Example: Picking *k*

Just right;
distances
rather short



Example: Picking *k*

Too many;
little improvement
in average
distance



Implementing k-Means in Spark

Reading Points

Read-in points into RDD
Pick k points at random
Repeat until convergence

1) Place each point it in the cluster with nearest centroid
2) Update the locations of centroids of the ***k*** clusters

NumPy: optimized library for arrays and linear algebra

```
import numpy as np
from pyspark import SparkContext

def parseVector(line):
        return np.array([float(x) for x in line.split(' ')])

sc = SparkContext(appName="PythonKMeans")
lines = sc.textFile(sys.argv[1])
data = lines.map(parseVector).cache()
```

Function parseVector turns a text line with numbers into a numpy-vector

Created RDD has numpy-vectors as records; cached in memory

Picking k Initial Centroids

Read-in points into RDD
Pick k points at random
Repeat until convergence
  1) Place each point it in the cluster with nearest centroid
  2) Update the locations of centroids of the ***k*** clusters

2nd argument given to python process is parsed as K

Collection with K numpy-vectors

Spark action: samples K records and returns to the driver

```
K = int(sys.argv[2])

centroids = data.takeSample(False, K, 1)
newCentroids = centroids[:]      # copy array
```

Testing Convergence

Read-in points into RDD
Pick k points at random
Repeat until convergence
1) Place each point it in the cluster with nearest centroid
2) Update the locations of centroids of the *k* clusters

Threshold for convergence

Computes sum of squared Euclidean distances between old and new centroids

```
convergeDist = float(sys.argv[3])

def distanceCentroidsMoved(oldCentroids, newCentroids):
    sum = 0.0
    for index in range(len(oldCentroids)):
        sum += np.sum( (oldCentroids[index] - newCentroids[index]) ** 2 )
    return sum
```

```
tempDist = 2* convergeDist            # constant larger than convergeDist
while  tempDist > convergeDist:
        <k-Means iteration, use centroids, compute newCentroids>
        tempDist = distanceCentroidsMoved(centroids, newCentroids)
        centroids = newCentroids[:]   # copy
```

Finding Closest Centroids /1

Read-in points into RDD
Pick k points at random
Repeat until convergence
1) Place each point it in the cluster with nearest centroid
2) Update the locations of centroids of the ***k*** clusters

Input is a point p and list of K centroids

```
def closestPoint(p, centroids):
    bestIndex = 0
    closest = float("+inf")
    for index in range(len(centroids)):
        tempDist = np.sum((p - centroids[index]) ** 2)
            if tempDist < closest:
                closest = tempDist
                bestIndex = index
    return bestIndex
```

For a point p (=numpy vector) computes index of the closest centroid in centroids

Finding Closest Centroids /2

Read-in points into RDD
Pick k points at random
Repeat until convergence
1) Place each point it in the cluster with nearest centroid
2) Update the locations of centroids of the *k* clusters

What is the "1" for?

```
while tempDist > convergeDist:
    closest = data.map(
        lambda p: (closestPoint(p, centroids), (p, 1)) )
```

Point p assigned to cluster with index j becomes a record (j, (p,1)) in a new RDD (i.e. record is a nested tuple)

Update the Location of Centroids /1

Read-in points into RDD
Pick k points at random
Repeat until convergence
- **1)** Place each point it in the cluster with nearest centroid
- **2)** Update the locations of centroids of the ***k*** clusters

```
while tempDist > convergeDist:
    closest =  ...
    for cIndex in range(K):
        closestOneCluster=closest.filter(lambda d: d[0] == cIndex)
                    .map(lambda d: d[1])
```

Each d is a record (j, (p,1)), so d[0] is cluster index j of p

This RDD contains tuples (p_0,1), (p_1,1),… for all points in cluster with index = cIndex

Update the Location of Centroids/2

Read-in points into RDD
Pick k points at random
Repeat until convergence

1) Place each point it in the cluster with nearest centroid
2) Update the locations of centroids of the *k* clusters

Each d is a record (j, (p,1)), so d[0] is cluster index j of p

```
while tempDist > convergeDist:
    closest =  ...
    for cIndex in range(K):
        closestOneCluster=closest.filter(lambda d: d[0] == cIndex)
                .map(lambda d: d[1])
        sumAndCountOneCluster=closestOneCluster.reduce(
                lambda p1, p2: (p1[0]+p2[0], p1[1]+p2[1]) )
        vectorSum = sumAndCountOneCluster[0]
        count = sumAndCountOneCluster[1]
        newCentroids[cIndex] = vectorSum / count
```

The Complete Loop

```
tempDist = 2* convergeDist
while  tempDist > convergeDist:
        closest = data.map(lambda p: (closestPoint(p, centroids), (p, 1)) )
        for cIndex in range(K):
                closestOneCluster=closest.filter(lambda d: d[0] == cIndex)
                        .map(lambda d: d[1])
                sumAndCountOneCluster=closestOneCluster.reduce(
                        lambda p1, p2: (p1[0]+p2[0], p1[1]+p2[1]))
                vectorSum = sumAndCountOneCluster[0]
                count = sumAndCountOneCluster[1]
                newCentroids[cIndex] = vectorSum / count

        tempDist = distanceCentroidsMoved(centroids, newCentroids)
        centroids = newCentroids[:]
```

Improvements Are Possible

- We need several RDDs (e.g. closestOneCluster) for <u>each cluster</u> but have same operations
- Idea: use Sparks group operations, like
  - reduceByKey
  - groupByKey
- E.g.: closest = (j, (p,1)), (j', (p',1)), (j'', (p'',1)), ...,
  - Transformation closest.groupByKey() gives RDD with (0, group0), (1, group1), (2, group2), ...,
  - where groupX is collection [(p,1), (p',1), (p'',1), ...] of all "points" in cluster with index X

Summary

- We have written a complete & scalable implementation of k-means on <1 page*
  - Runs for arbitrarily large data sets
- Difficult?

- If you agree, wait for the MapReduce version ☺
- Code download k-means in Spark: https://1drv.ms/f/s!Arb2LwF7ECx4h4Mot6gx6xJ0rn_Vbg

* = Using 4-pixel font size or smaller

Further Reading / Information

- Book “Mining of Massive Datasets”, Chapter 7 (in particular 7.3)
- The Data Science Lab, Clustering With K-Means in Python, http://datasciencelab.wordpress.com/2013/12/12/clustering-with-k-means-in-python/
- Installing numpy on Ubuntu (our VM):
  - sudo apt-get install python-numpy
  - See http://askubuntu.com/questions/359254/how-to-install-numpy-and-scipy-for-python
- Link to code (also for future lectures):
  - https://1drv.ms/f/s!Arb2LwF7ECx4h4Mot6gx6xJ0rn_Vbg

Thank you.

Questions?