Machine Learning 101

1

2

3

4

5

6

Which group does this object belong to?

- How did our brain process the images?
- How did the grouping happen?
- Human brain processed the given images - **learning**
- After learning the brain simply looked at the new image and compared with the groups classified the image to the closest group -**Classification**
- If a machine has to perform the same operation we use Machine Learning
- We write programs for learning and then classification, this is nothing but machine learning

Difference Traditional Models & Machine Learning

- ML is more heuristic
- Focused on improving performance of a learning agent
- Also looks at real-time self learning and robotics – areas not part of data mining
- Some algorithms are too heuristic that there is no one right or wrong answer

Applications of ML

- Biomedical / Biometrics
  - Medicine:
    - Screening
    - Diagnosis and prognosis
    - Drug discovery
- Security:
  - Face recognition
  - Signature / fingerprint / iris verification
  - DNA fingerprinting

Applications of ML

- Computer / Internet
  - Computer interfaces:
    - Troubleshooting wizards
    - Handwriting and speech
    - Brain waves
  - Internet
    - Hit ranking
    - Spam filtering
    - Text categorization
    - Text translation
    - Recommendation

Main Parts in Machine Learning

- Any Machine Learning algorithm has three parts
  - The Output
  - The Objective Function or Performance Matrix
  - The Input

- **Email Spam Classification**
  - **The Output:** Categorize email messages as spam or legitimate.
  - **Objective Function:** Percentage of email messages correctly classified.
  - **The Input:** Database of emails, some with human-given labels

- **Hand Writing Recognition**
  - **The Output:** Recognizing hand-written words
  - **Objective Function:** Percentage of words correctly classified
  - **The Input:** Database of human-labeled images of handwritten words

Machine Learning

- **We will talk about the following methods:**
  - **k**-Nearest Neighbor (Instance based learning)
  - Perceptron (Neural Networks)
  - Support Vector Machines
  - Decision trees
- **With** hands on demonstrations

- **Main question:**
  How to efficiently train
  (build a model/find model parameters)**?**

Machine Learning

- Supervised Learning
  - Have some data with labeled values
- Unsupervised Learning
  - Does not have labeled data

Supervised Learning

- **Example: Spam filtering**

	viagra	learning	the	dating	nigeria	$spam?$
$\vec{x}_1 = ($	1	0	1	0	0 $)$	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0 $)$	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1 $)$	$y_3 = 1$

- **Instance space x ∈ X** (|**X**|= **n** data points)
  - **Binary or real-valued feature vector *x* of word occurrences**
  - ***d*** features (words + other things, **d**~100,000)
- **Class y ∈ Y**
  - ***y***: Spam (+1), Ham (-1)
- **Goal: Estimate** a function **f(x)** so that ***y = f(x)***

More generally: Supervised Learning

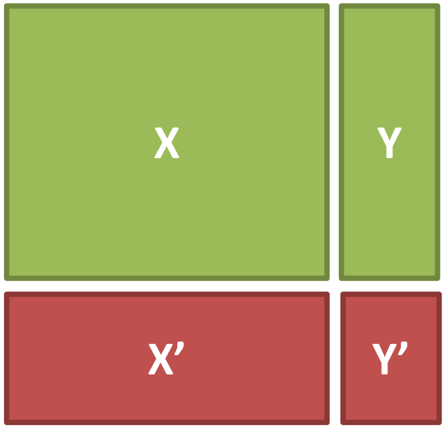
- **Would like to do prediction:**
  estimate a function **f(x)** so that ***y = f(x)***

- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical: Classification**
  - **Complex object:**
    - Ranking of items, Parse tree, etc.

- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

Supervised Learning

- **Would like to do prediction:**
  estimate a function **f(x)** so that ***y = f(x)***
- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical**: Classification
  - Complex object:
    - Ranking of items, Parse tree, etc.
- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

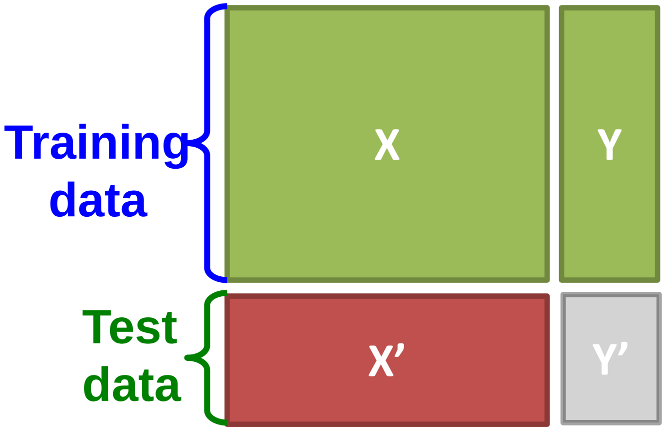


Training and **test** set

Estimate *y = f(x)* on *X,Y.*
Hope that the same *f(x)*
also works on unseen X', Y'

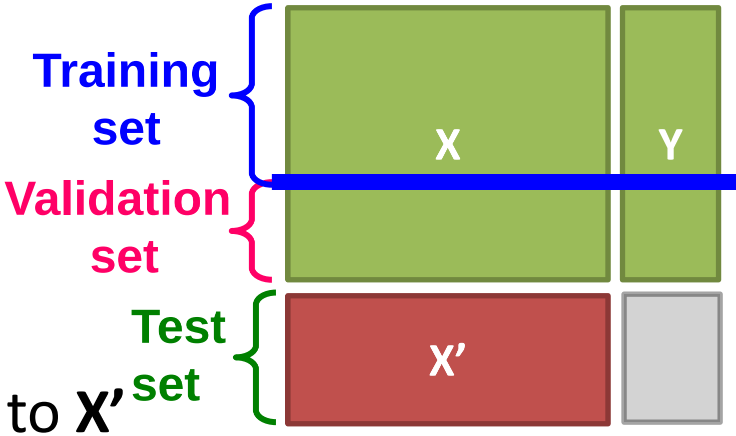
Supervised Learning

- **Task:** Given data **(X,Y)** build a model **f()** to predict **Y'** based on **X'**
- **Strategy: Estimate $y = f(x)$ on (X, Y).**
  Hope that the same $f(x)$ also works to predict unknown Y'
  – The **"hope"** is called **generalization**
    - **Overfitting: If f(x) predicts well Y but is unable to predict Y'**
  – **We want to build a model that <u>generalizes</u> well to unseen data**



Supervised Learning

- **Idea: Pretend we do not know the data/labels we actually do know**
  - Build the model **f(x)** on the training data
    See how well **f(x)** does on the test data
    - If it does well, then apply it also to **X'**



- **Refinement: Cross validation**
  - Splitting into training/validation set is brutal
  - Let's split our data **(X,Y)** into 10-folds (buckets)
  - Take out 1-fold for validation, train on remaining 9
  - Repeat this 10 times, report average performance

Supervised Learning

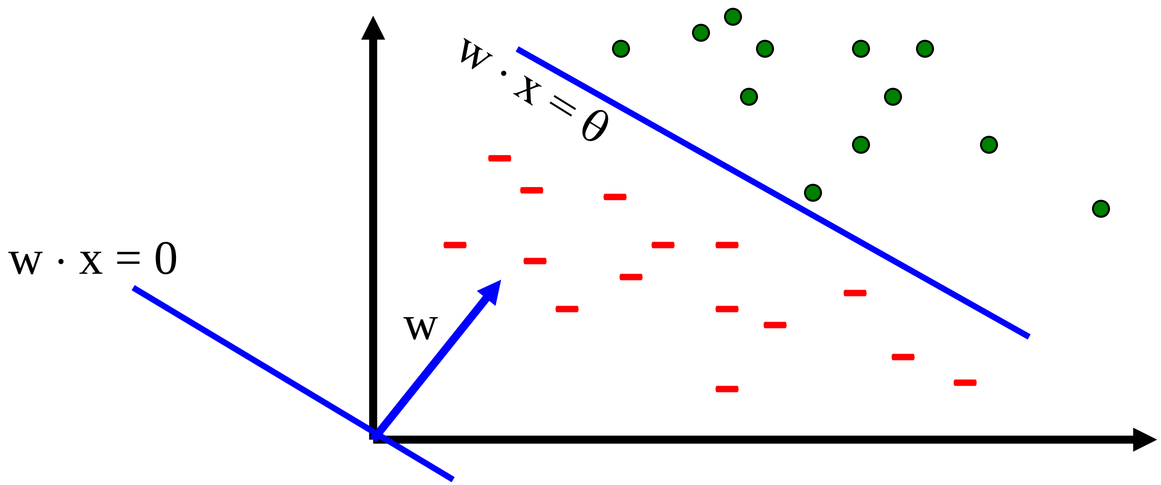
- Models to know:
  - Linear Regressions
  - SVM
  - Decision Trees

Linear models for classification

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad \mathbf{w}^{(1)}\mathbf{x}^{(1)} + \mathbf{w}^{(2)}\mathbf{x}^{(2)} + \ldots \mathbf{w}^{(d)}\mathbf{x}^{(d)} \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

- **Input:** Vectors $\boldsymbol{x}_j$ and labels $\boldsymbol{y}_j$
  - Vectors $\boldsymbol{x}_j$ are real valued where $\|x\|_2 = 1$
- **Goal:** Find vector $\boldsymbol{w} = (w^{(1)}, w^{(2)}, \ldots, w^{(d)})$
  - Each $\boldsymbol{w}^{(i)}$ is a real number



Decision boundary is **linear**

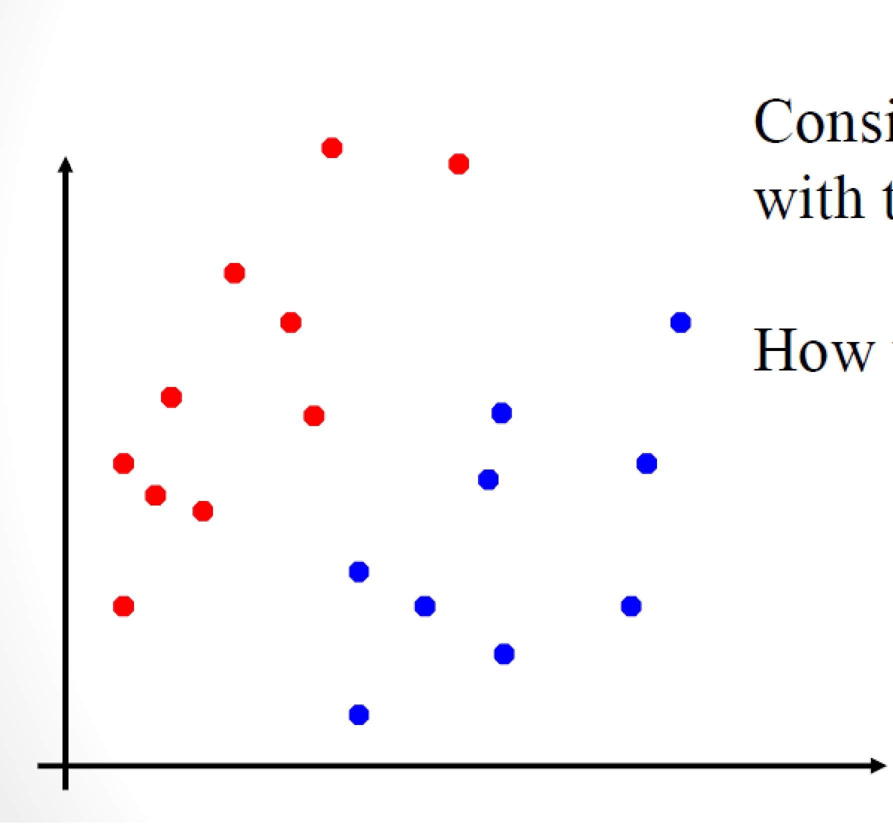
Note:

$$\mathbf{x} \rightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

$$\mathbf{w} \rightarrow \langle \mathbf{w}, -\theta \rangle$$

SVM – Linear Classifier

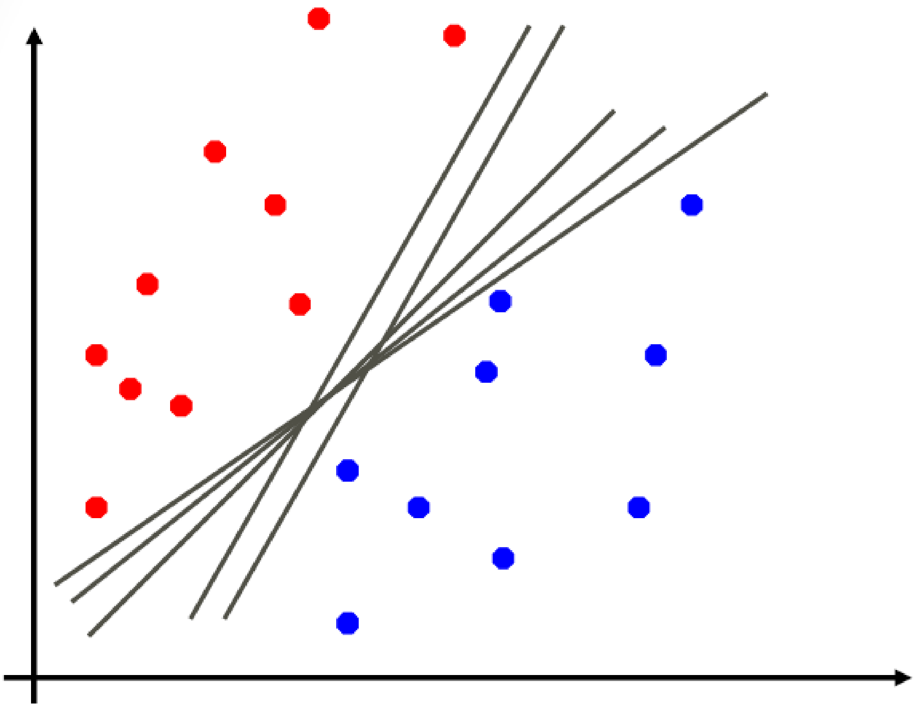
Supervised Learning



Consider a two dimensional dataset with two classes

How would we classify this dataset?

SVM – Linear Classifier

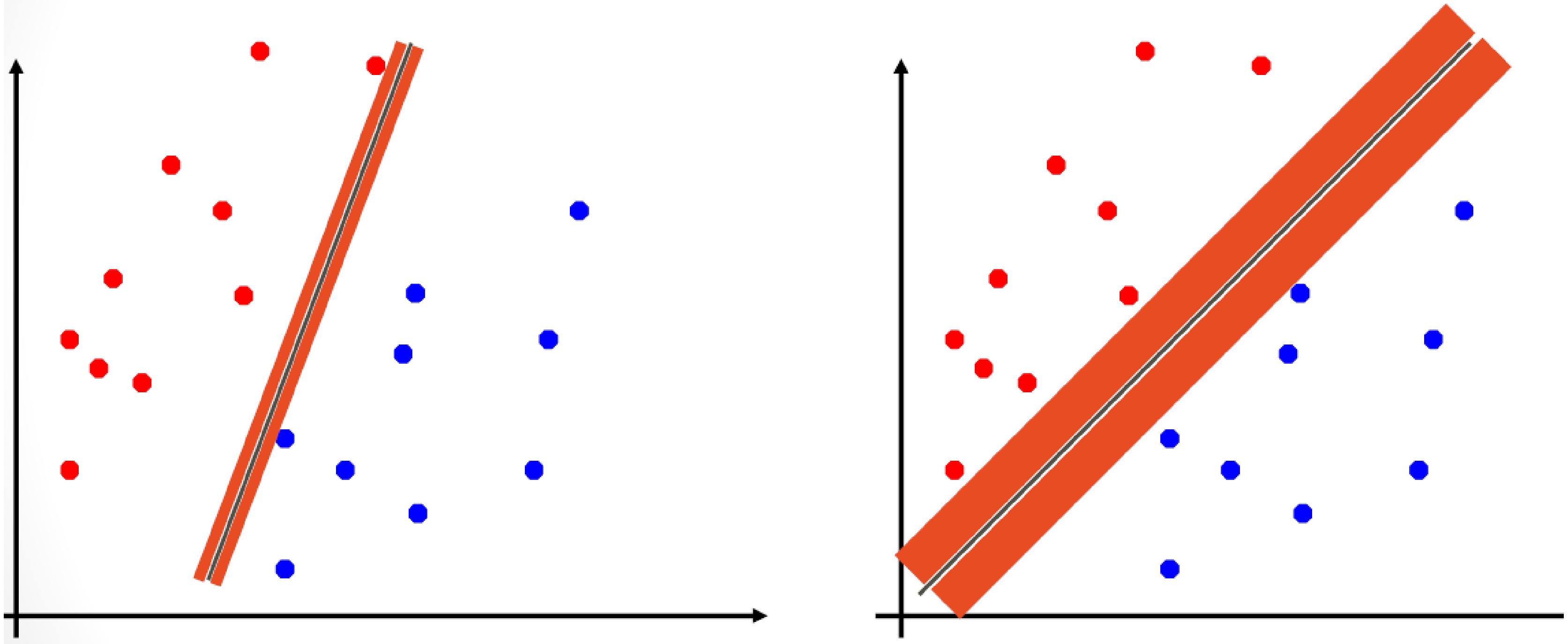


There are many lines that can be linear classifiers.

Which one is the optimal classifier.

SVM - Classifier

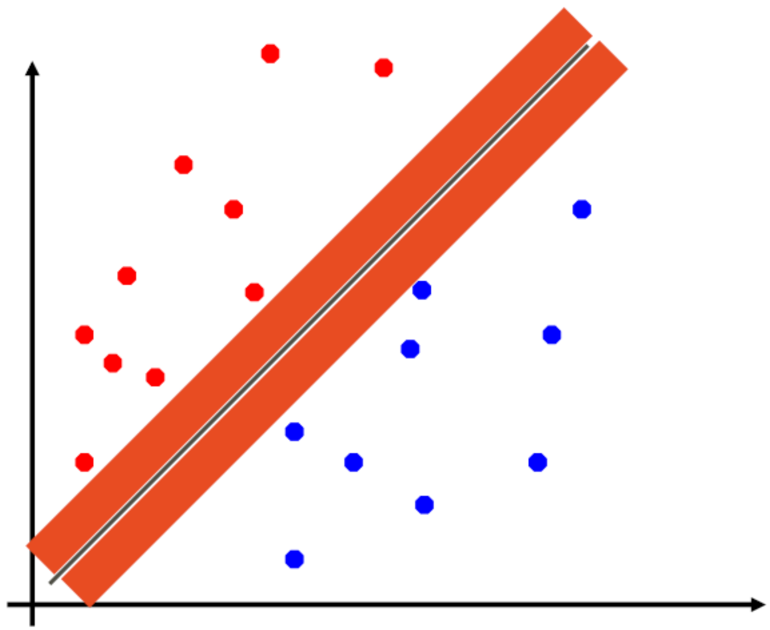
- Margin



Define the **margin** of a linear classifier as the width that theboundary could be increased by before hitting a datapoint.

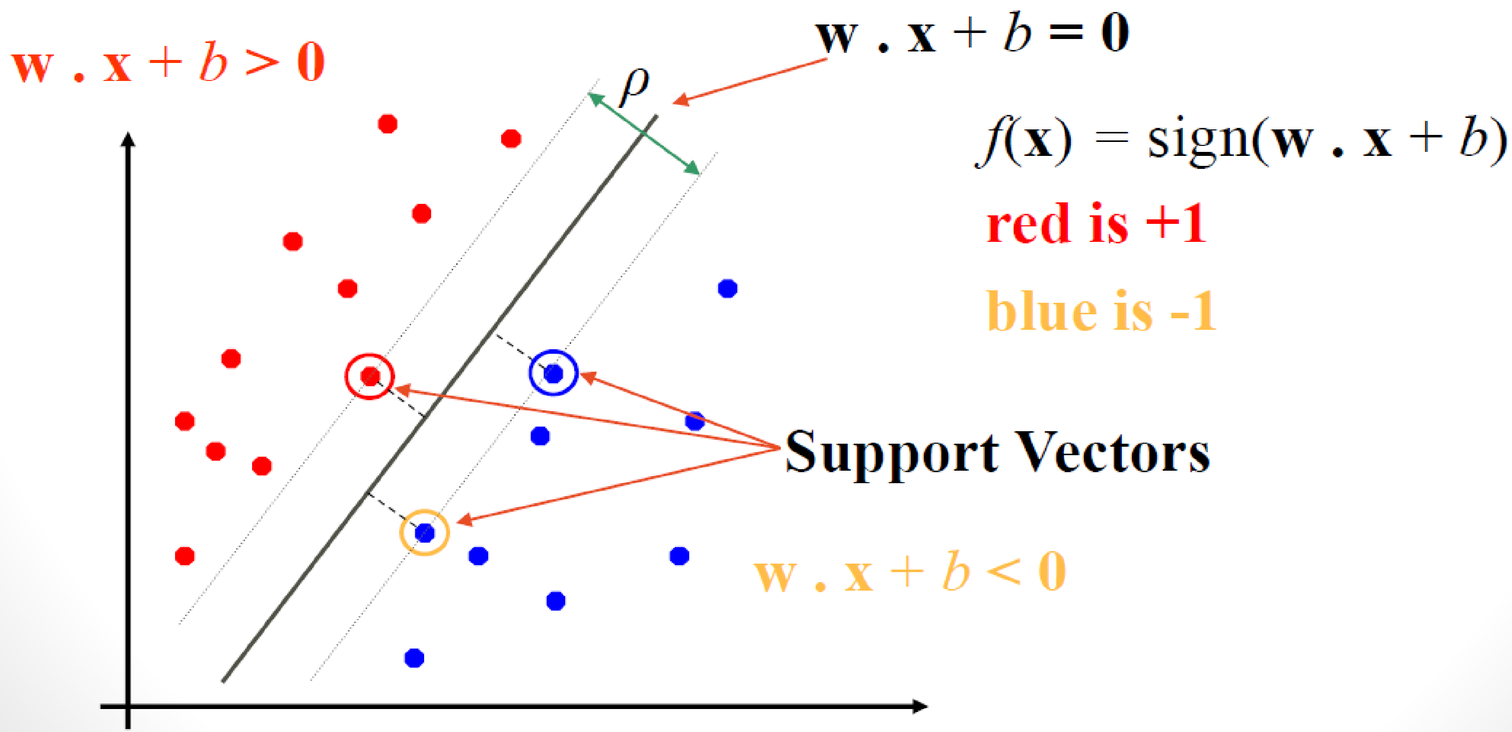
SVM – Classifier

- Maximum Margin
  - The **maximum margin linear classifier** is the linear classifier
  - with the maximum margin.
  - This is the simplest kind of SVM (Called Linear SVM)



SVM - Classifier

- Support Vectors
  - Examples closest to the hyper plane are ***support vectors***.
  - ***Margin*** *ρ* of the separator is the distance between support vectors.



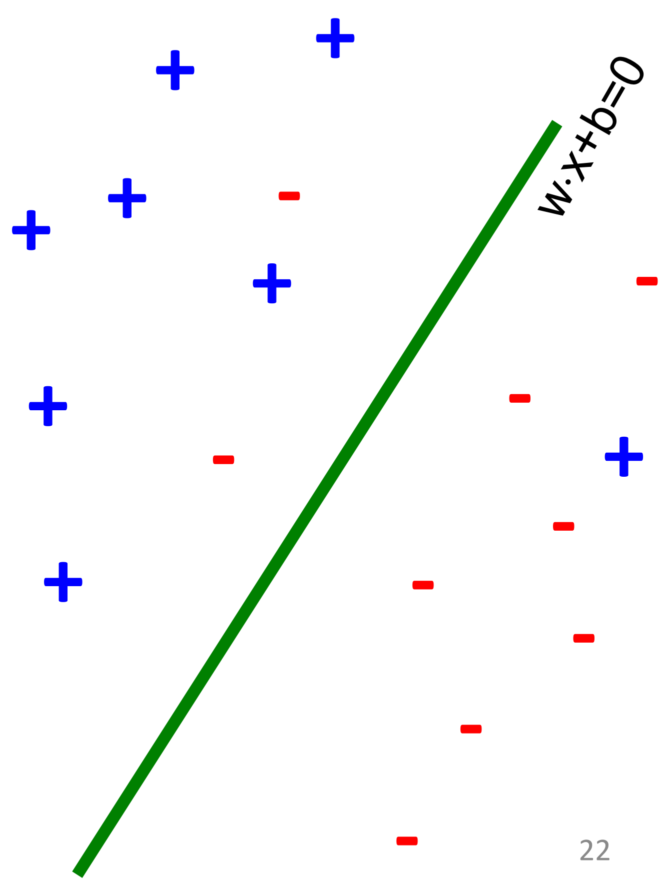
Non-linearly Separable Data

- **If data is not separable introduce penalty:**

$$\min_w \tfrac{1}{2}\|w\|^2 + C \cdot (\#\text{number of mistakes})$$

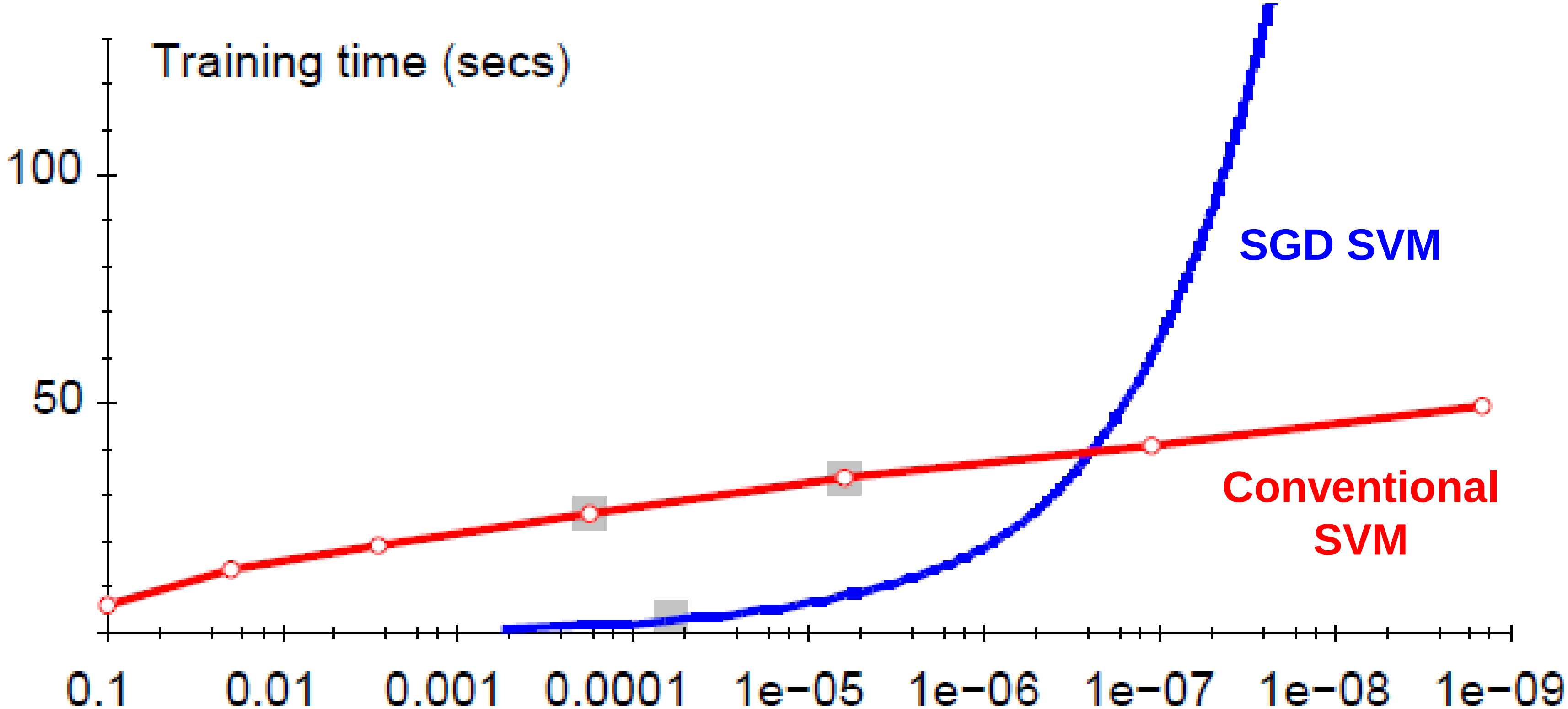
$$s.t. \forall i, y_i(w \cdot x_i + b) \geq 1$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set ***C*** using cross validation



- **How to penalize mistakes?**
  - **All mistakes are not equally bad!**

Optimization "Accuracy"

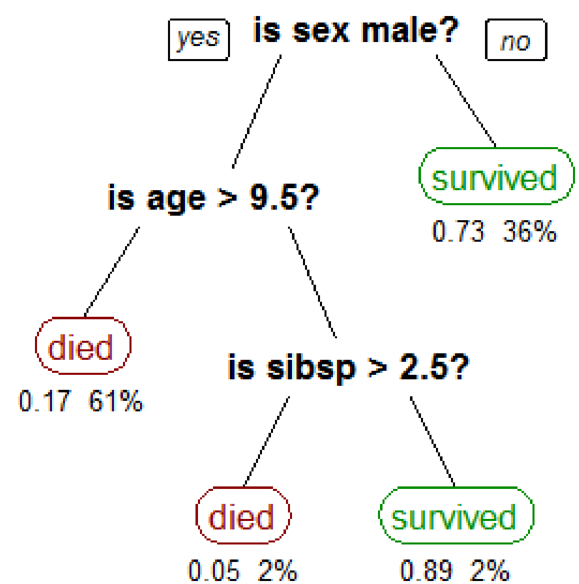


Optimization quality: | *f(w,b)* – *f* (w^{opt},b^{opt}) |

For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

Decision Trees

- Supervised Learning
  - Classification/Rule based
  - Random Forests
    - Multiple trees/Ensemble
      - Subsets of features
    - Interior nodes are feature splits
    - Leaf is desired value
  - Boosted trees: regression



Unsupervised Learning

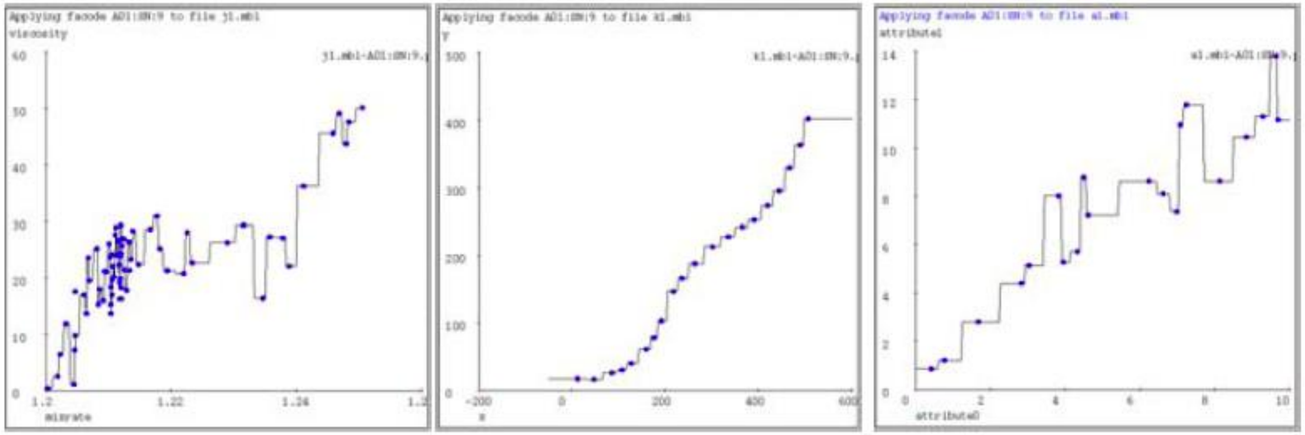
- KNN

Instance Based Learning

- **Instance based learning**
- **Example: Nearest neighbor**
  - Keep the whole training dataset: **{(x, y)}**
  - A query example (vector) ***q*** comes
  - Find closest example(s) $\mathbf{x}^*$
  - Predict $\mathbf{y}^*$
- **Works both for regression and classification**
  - **Collaborative filtering** is an example of k-NN classifier
    - Find ***k*** most similar people to user **x** that have rated movie **y**
    - Predict rating $\mathbf{y}_\mathbf{x}$ of **x** as an average of $\mathbf{y}_\mathbf{k}$

1-Nearest Neighbor

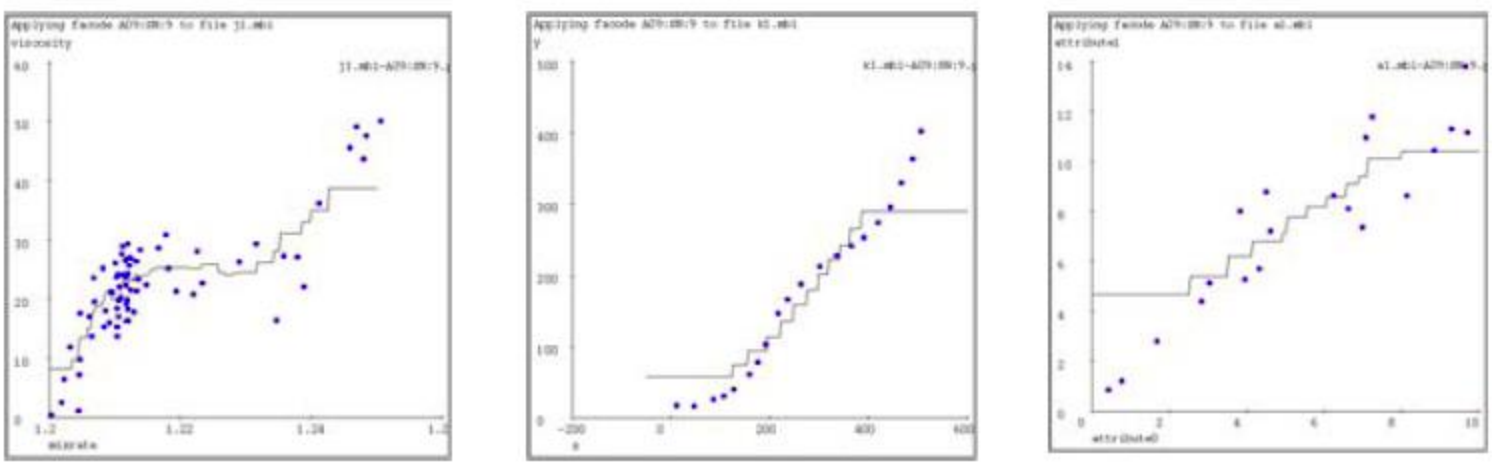
- **To make Nearest Neighbor work we need 4 things:**
  - **Distance metric:**
    - Euclidean
  - **How many neighbors to look at?**
    - One
  - **Weighting function (optional):**
    - Unused
  - **How to fit with the local points?**
    - Just predict the same output as the nearest neighbor



http://www.mmas.org

k-Nearest Neighbor

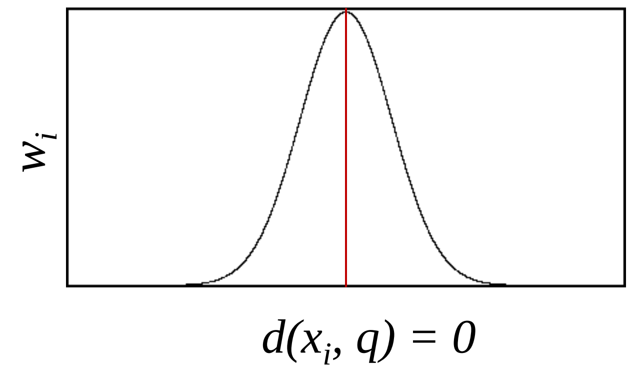
- **Distance metric:**
  - Euclidean
- **How many neighbors to look at?**
  - ***k***
- **Weighting function (optional):**
  - Unused
- **How to fit with the local points?**
  - Just predict the average output among *k* nearest neighbors



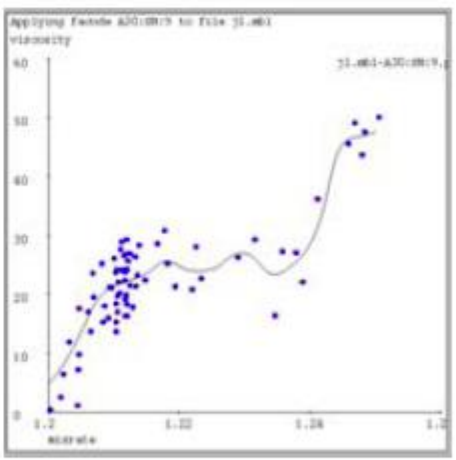
k=9

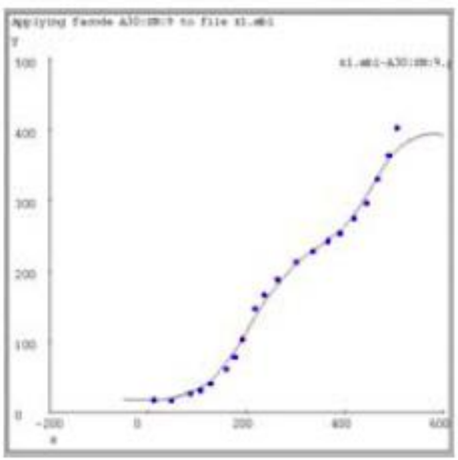
Kernel Regression

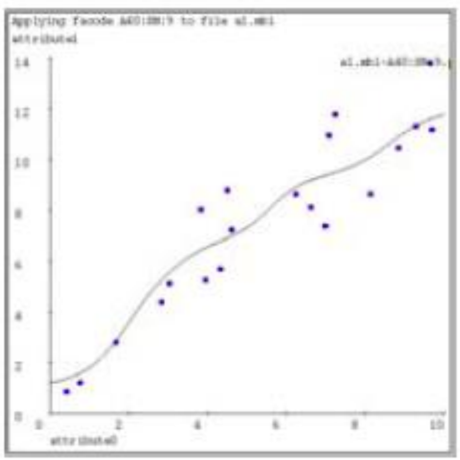
- **Distance metric:**
  - Euclidean
- **How many neighbors to look at?**
  - **All of them (!)**
- **Weighting function:**
  - $w_i = \exp(-\frac{d(x_i, q)^2}{K_w})$
    - Nearby points to query q are weighted more strongly. **K_w**…kernel width.



- **How to fit with the local points?**
  - **Predict weighted average:** $\frac{\sum_i w_i y_i}{\sum_i w_i}$

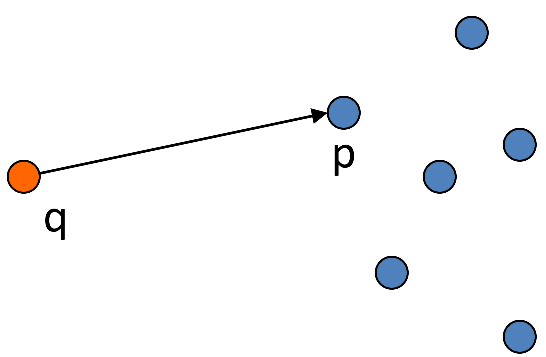






How to find nearest neighbors?

- **Given:** a set ***P*** of ***n*** points in $\boldsymbol{R^d}$
- **Goal: Given a query point *q***
  - **NN:** Find the *nearest neighbor* ***p*** of ***q*** in ***P***
  - **Range search:** Find one/all points in ***P*** within distance ***r*** from ***q***



Algorithms for NN

- **Main memory:**
  - **Linear scan**
  - **Tree based:**
    - Quadtree
    - kd-tree
  - **Hashing:**
    - Locality-Sensitive Hashing
- **Secondary storage:**
  - R-trees

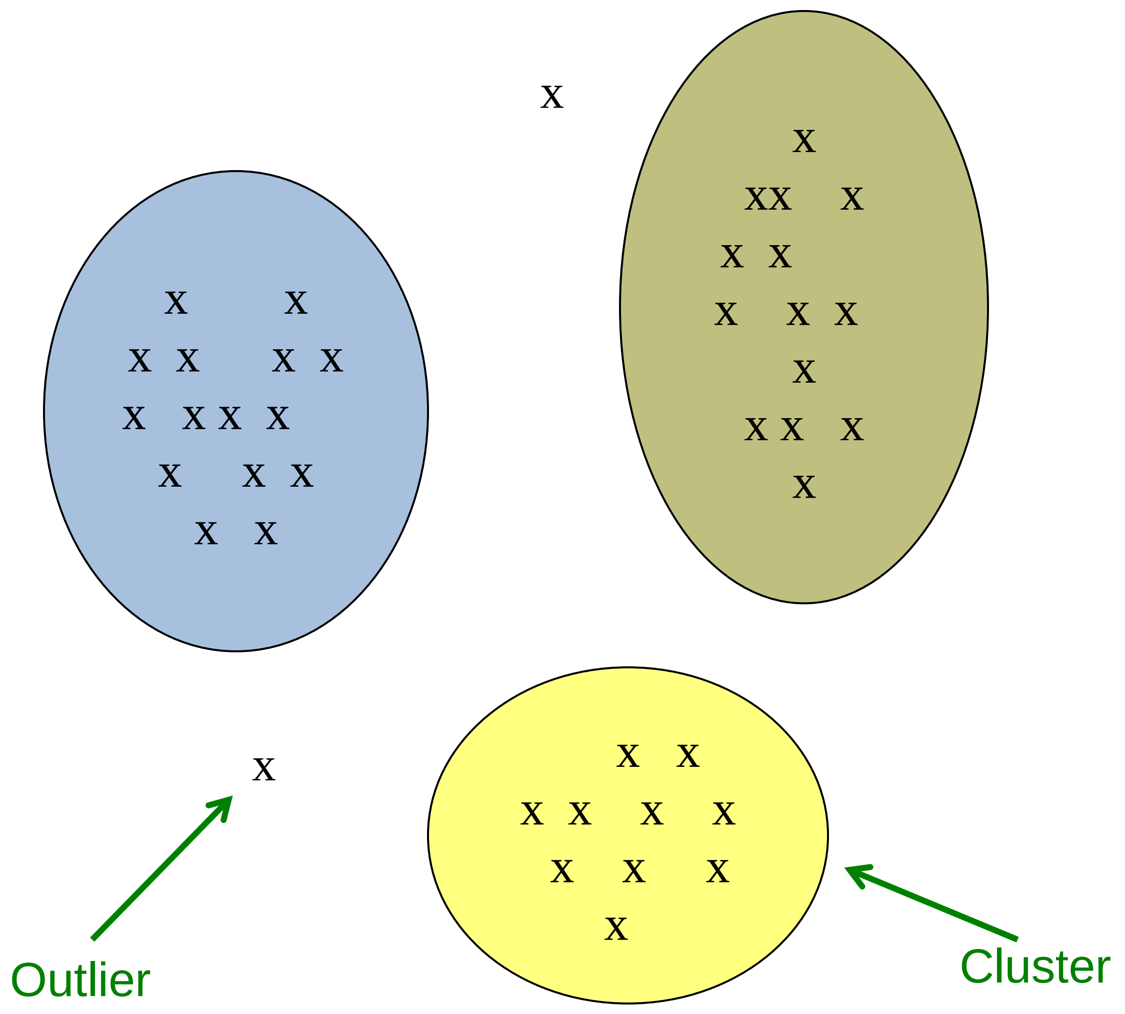
Clustering

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

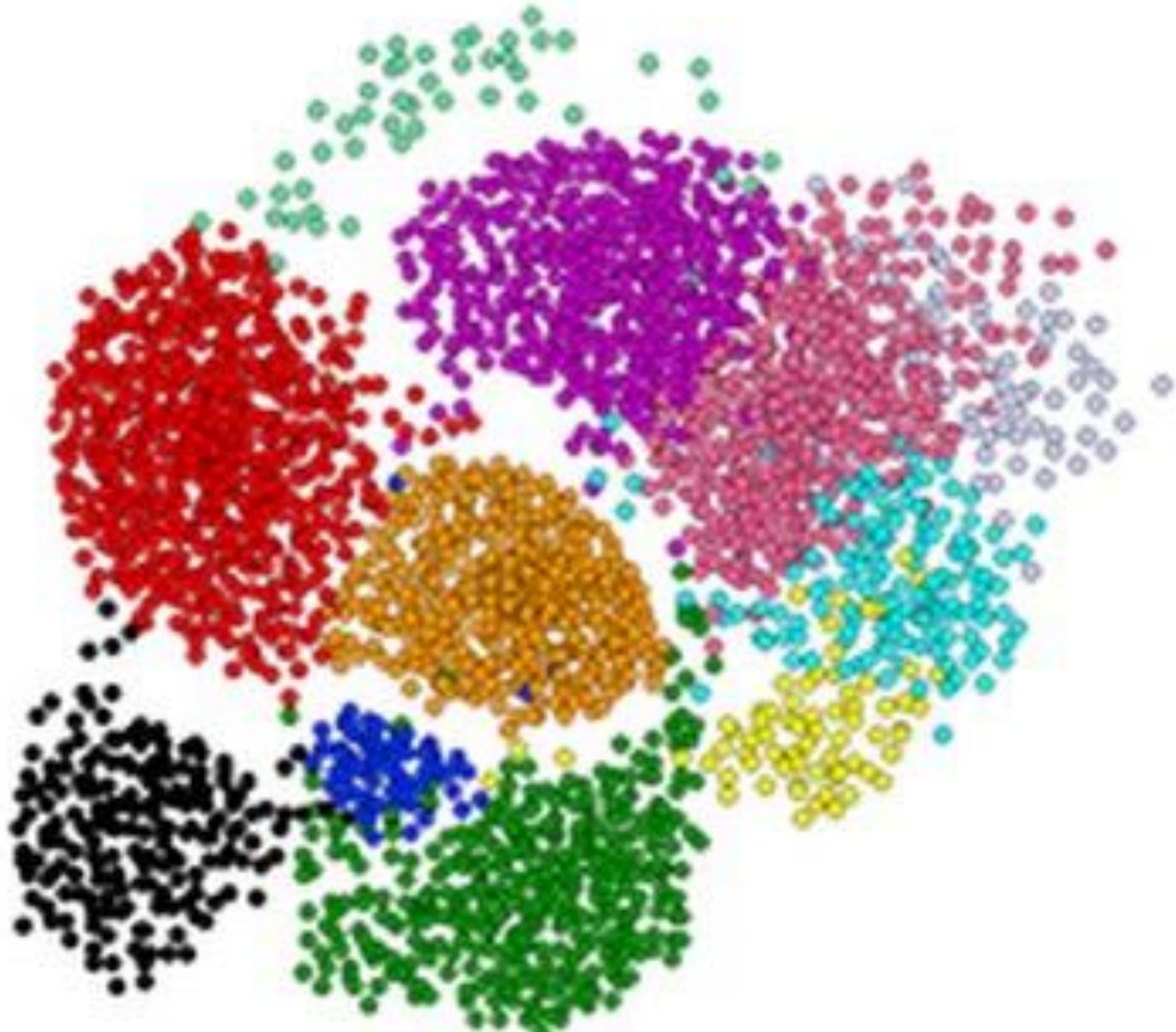
The Problem of Clustering

- 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, Jaccard, edit distance, …

Example: Clusters & Outliers



Clustering is a hard problem!



http://www.mmas.org

Clustering - 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: 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

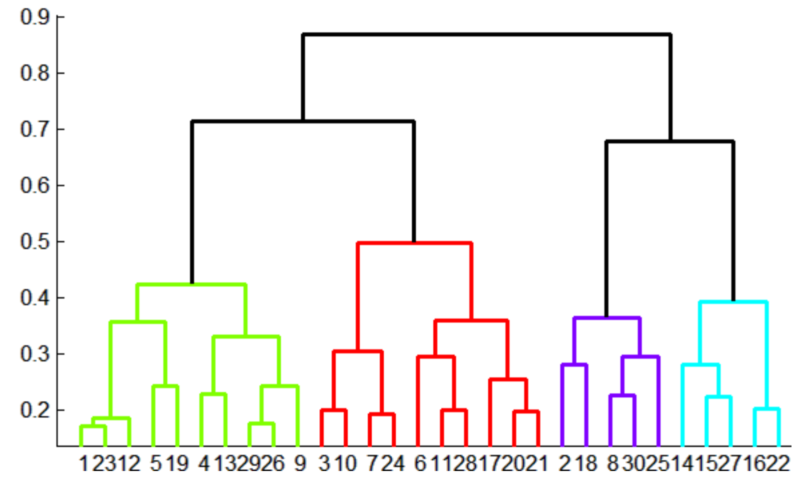
- **Documents with similar sets of words may be about the same topic**

Cosine, Jaccard, and Euclidean

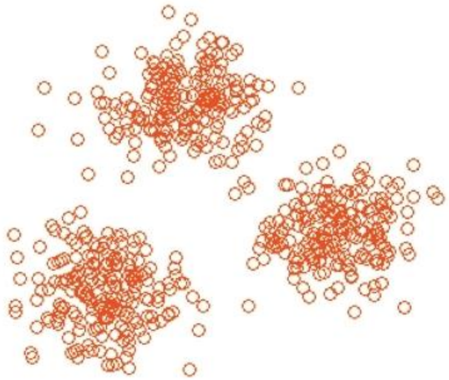
- **As with CDs we have a choice when we think of documents as sets of words or shingles:**
  - **Sets as vectors:** Measure similarity by the **cosine distance**
  - **Sets as sets:** Measure similarity by the **Jaccard distance**
  - **Sets as points:** Measure similarity by **Euclidean distance**

Overview: Methods of Clustering

- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it



- **Point assignment:**
  - Maintain a set of clusters
  - Points belong to "nearest" cluster



Implementation

- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise **distances** between all pairs of clusters, then merge
  - $O(N^3)$

- Careful implementation using priority queue can reduce time to $O(N^2 \log N)$
  - **Still too expensive for really big datasets that do not fit in memory**

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

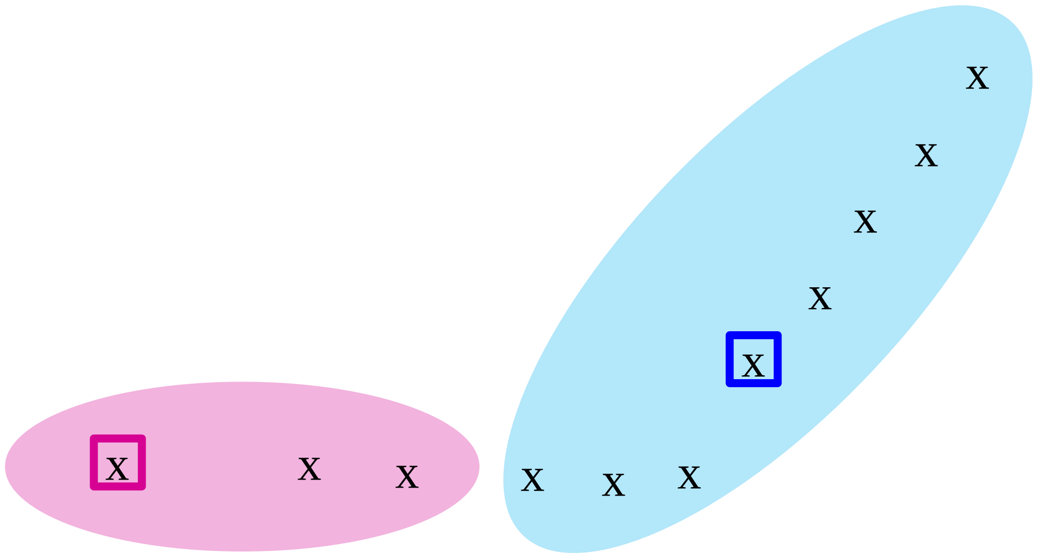
- **1)** For each point, place it in the cluster whose current centroid it is nearest

- **2)** After all points are assigned, update the locations of centroids of the ***k*** clusters

- **3)** Reassign all points to their closest centroid
  - Sometimes moves points between clusters

- **Repeat 2 and 3 until convergence**
  - **Convergence:** Points don't move between clusters and centroids stabilize

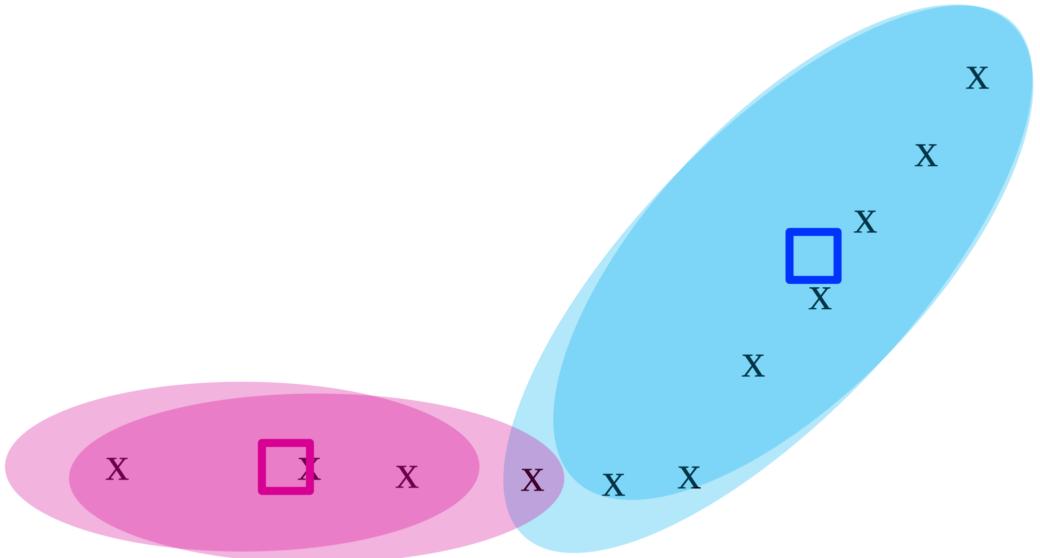
Example: Assigning Clusters



x … data point
… centroid

Clusters after round 1

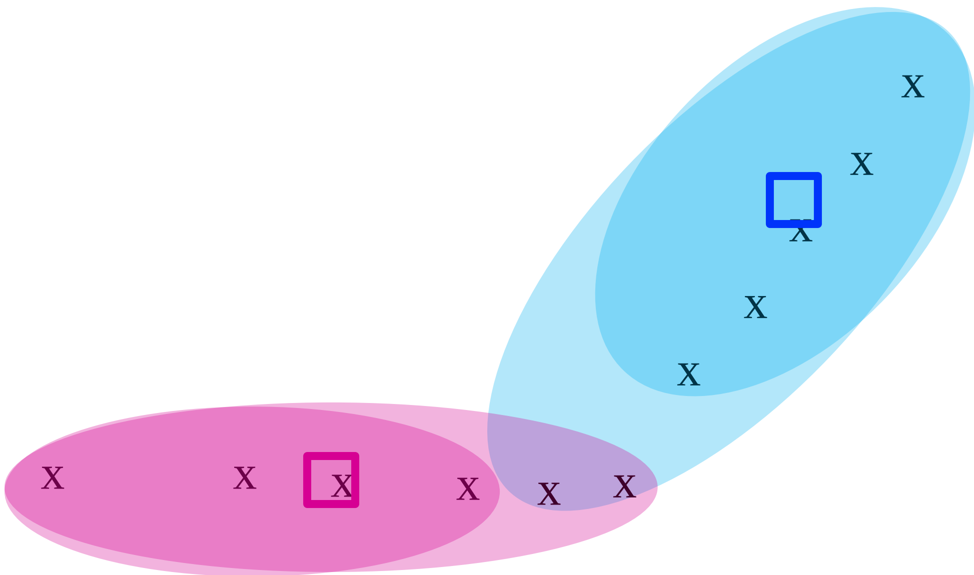
Example: Assigning Clusters



x … data point
… centroid

Clusters after round 2

Example: Assigning Clusters



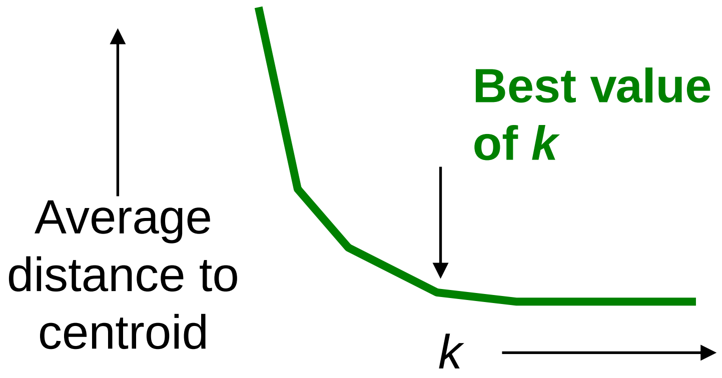
x … data point
… centroid

Clusters at the end

Getting the *k* right

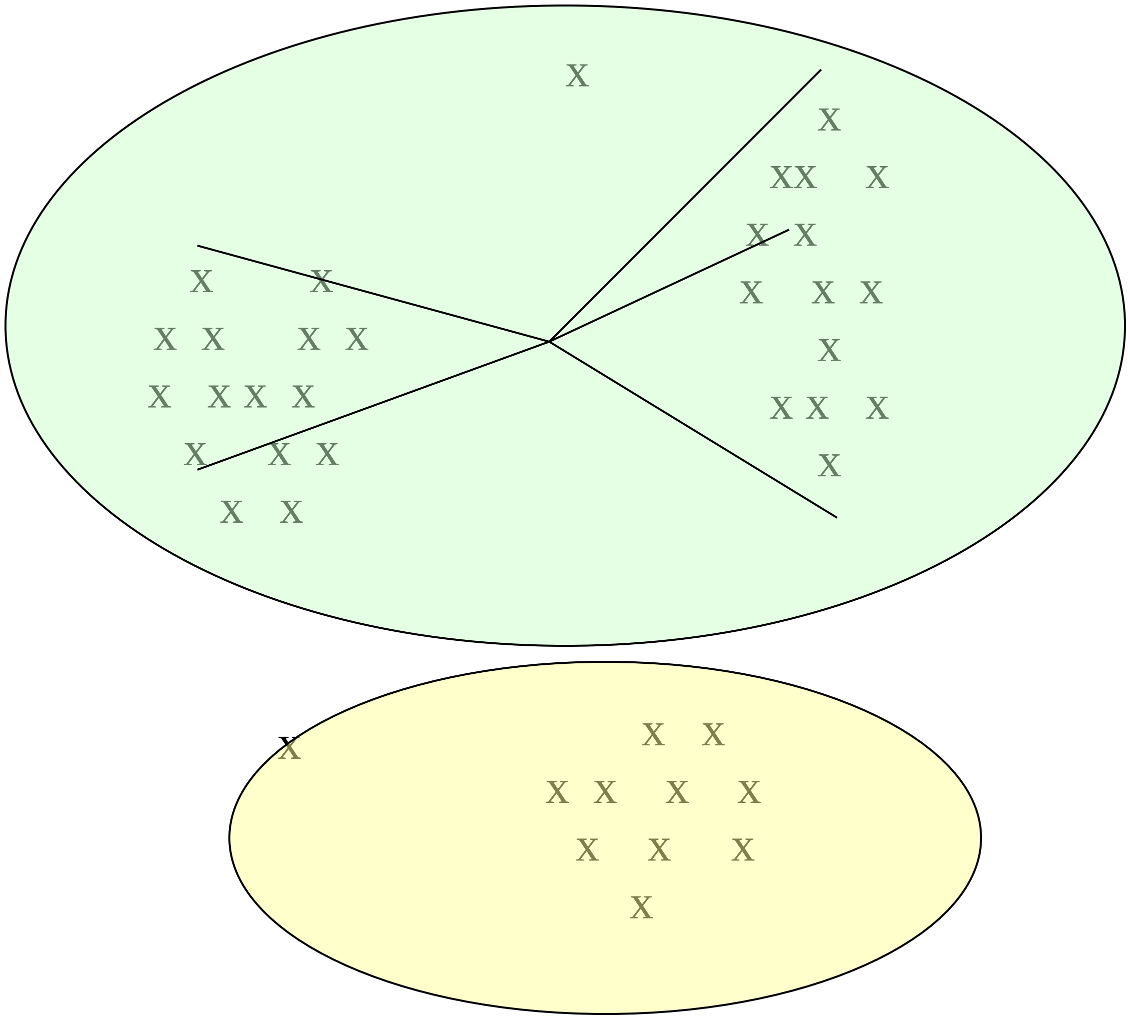
How to select *k*?

- 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



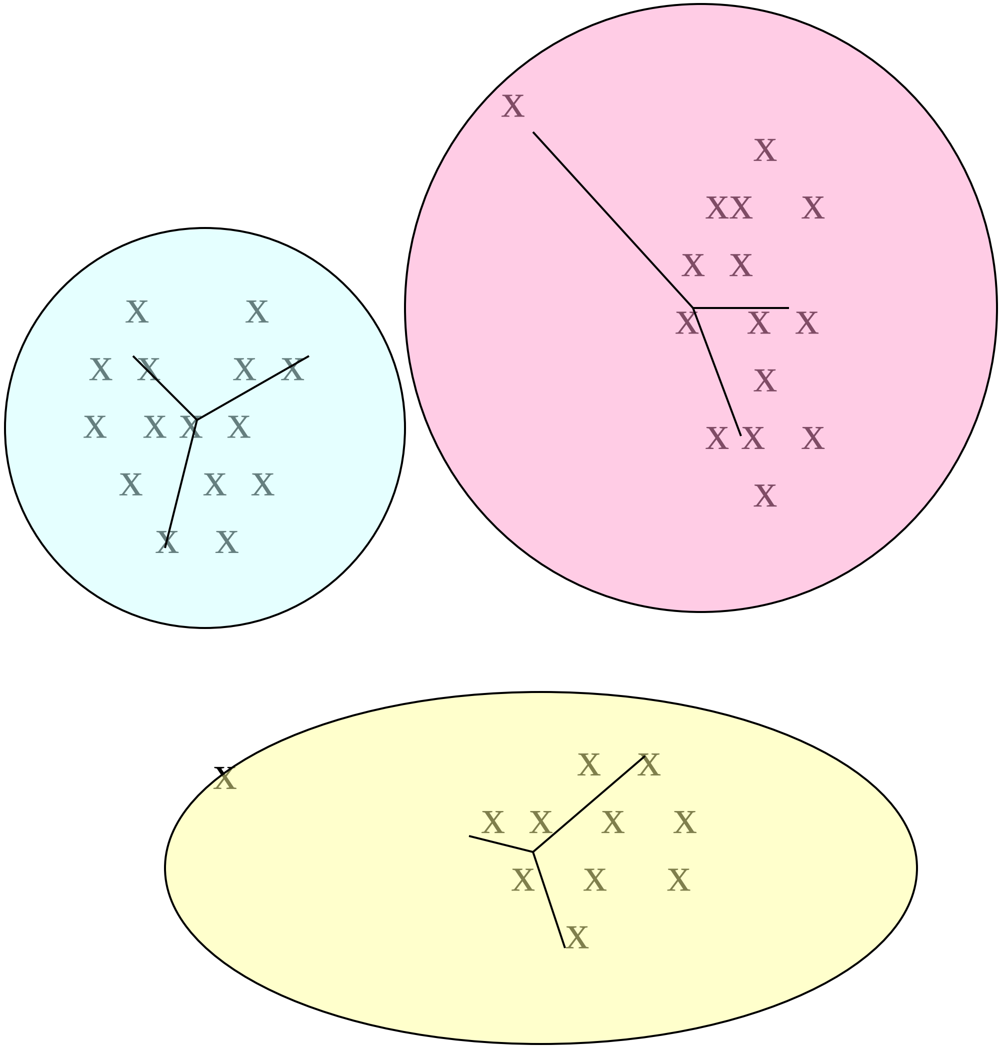
Example: Picking *k*

Too few;
many long distances to centroid.



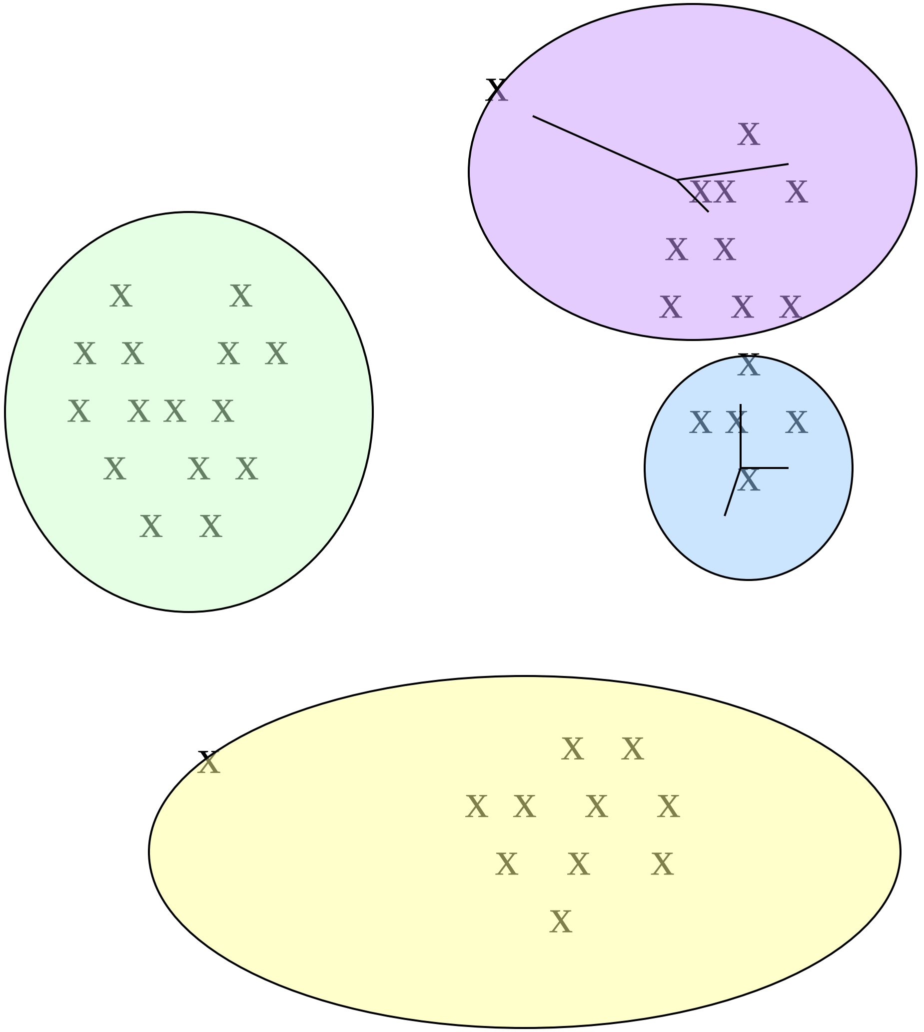
Example: Picking *k*

Just right;
distances
rather short.



Example: Picking *k*

Too many;
little improvement in average distance.



Summary

- **Clustering:** Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***
- **Algorithms:**
  - Agglomerative **hierarchical clustering**:
    - Centroid and clustroid
  - ***k*-means:**
    - Initialization, picking *k*

SUMMARY

- 3 main parts of a machine learning model
- Overfitting / Curse of Dimensionality
- Intuition Fails in high dimensions
- Theoretical guarantees are not what they seem
- Feature engineering is the **key**
- More data beats clever algorithms
- Learn many models, not just one
- Correlation does not imply Causation

SUMMARY

- 3 main parts of a machine learning model
  - Set of possible models to look through
  - A way to test the models
  - A clever way to find find a really good model

SUMMARY

- Overfitting / Curse of Dimensionality
  - Always test your model with out-of-test data
  - More features not necessarily good:
    - More features add noise
    - Correlation between new features might add more noise

SUMMARY

- Intuition Fails in high dimensions
  - Biases in model selection not always right
  - Select/test many models

SUMMARY

– Feature engineering is the **key**

CITY 1 LAT.	CITY 1 LNG.	CITY 2 LAT.	CITY 2 LNG.	DRIVABLE?
123.24	46.71	121.33	47.34	Yes
123.24	56.91	121.33	55.23	Yes
123.24	46.71	121.33	55.34	No
123.24	46.71	130.99	47.34	No

SUMMARY

- More data beats clever algorithms

- Learn many models, not just one

- Correlation does not imply Causation
  - **modeling observational data can only show us that two variables are related**, but it **cannot tell us the "why"**.

Machine Learning - Evaluation

- Did not discuss
  - Performance metrics
    - AUC
    - Others

  See
  - https://github.com/berndbischl/mlr/blob/master/doc/knitted/tutorial/roc_analysis.md

Machine Learning - Summary

- Tools
  - Spark ML
  - Knime
  - H2O
  - R/R-Studio
    - rattle
    - h2o
    - Mlr
    - SparkR