Feature Representations and Unsupervised Learning

Dan Lizotte

2018-10-30

"My data don't look like the simple examples you showed in class..."

- "I don't have nice vectors of features, each of the same length."
- Fair enough. Today, two instances of the following strategy:
  1. Identify the prediction you want to make.
  2. Identify the information you need to make each prediction.
  3. Summarize that information into a feature vector.
  4. Pass the result to a supervised learning method.

Documents: Spam or ham?

BNP PARIBAS 10 HAREWOOD AVENUE, LONDON NW1 6AA TEL: +44 (0) 207 595 2000

Attn:Sir/Madam,

RE: NOTIFICATION OF PAYMENT OF ACCRUED INTEREST OF ONE HUNDRED AND FIFTY THOUSAND BRITISH POUNDS STERLING ONLY (£150,000.00).

This is inform you that your contract/inheritance fund, which was deposited with us since 2010-2013 has accumulated an interest of sum of One Hundred and Fifty Thousand British Pounds Sterling Only (£150,000.00).

In line with the joint agreement signed by the board of trustees of BNP Paribas and the depositor of the said fund, the management has mandated us to pay you the accrued interest pending when approvals would be granted on the principal amount.

In view of the above, you are hereby required to complete the attached questionnaire form, affix your passport photograph, signed in the specified columns and return back to us via email for the processing of the transfer of your fund.

Do not hesitate to contact your assigned transaction manager, Mr. Paul Edward on his direct telephone no. +44 792 4278526, if you need any clarification.

Yours faithfully,

Elizabeth Manning (Ms.) Chief Credit Officer, BNP Paribas, London. Dear

Documents: Spam or ham?

Dan,

How are you? I hope everything is well.

Recently I have collaborated with some colleagues of mine from environmental engineering to research in multi-objective RL (if you are interested you can take a look at one of our papers: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6252759) and I found your approach very interesting.

I want to thank you for replying to my students' email, we have really appreciated your willingness to share your code with us. I know that the code used to produce paper results is often "weak" since it is not produced

for being distributed, and I understand your desire to set up and embellish the code (I do the same thing when other researchers ask for my code). I only want to reassure you that we have no hurry, so having your code in one or two weeks will be perfect.

Unfortunately I will not be in Rome for ICAPS. However, I hope to meet you soon, will you attend ICML?

Documents: Spam or ham?

paribas(0)=3.0
harewood(1)=1.0
avenue(2)=1.0
london(3)=2.0
nw(4)=1.0
aa(5)=1.0
tel(6)=1.0
attn(7)=1.0
sir(8)=1.0
madam(9)=1.0
re(10)=1.0
notification(11)=1.0
of(12)=11.0
payment(13)=1.0
accrued(14)=2.0
interest(15)=3.0
one(16)=2.0
hundred(17)=2.0
and(18)=4.0
fifty(19)=2.0
thousand(20)=2.0
british(21)=2.0
pounds(22)=4.0
sterling(23)=2.0
...
credit(114)=1.0
officer(115)=1.0
of(12)=2.0
one(16)=2.0
and(18)=3.0
only(24)=1.0
is(26)=3.0
you(28)=7.0
that(29)=2.0
your(30)=5.0
with(37)=2.0
us(38)=1.0
since(39)=1.0
in(44)=3.0
the(46)=3.0
to(58)=8.0
when(61)=1.0
be(64)=2.0
are(71)=2.0
email(86)=1.0
for(87)=4.0

do(90)=1.0
…
attend(202)=1.0
icml(203)=1.0

Dictionary-based Representations

- Define "feature detectors" that map an instance (e.g. document) to one feature value.
  - E.g. word features that are 1 if word is in a document, 0 otherwise
  - Or number of times word occurs in document
- Fix a collection of instances, called a *corpus*.
- Your feature set consists of all of your feature detectors that "turn on" for *some* instance in the corpus.
- Often results in a very large number of features.

Bag-of-words

- Little example has vocabulary size 203. In a real corpus, more like 10000.
- Representation is *sparse*: Only non-zero entries recorded
- Nonetheless, still a fixed-size feature vector
- Typical tricks: omit punctuation, omit capitalization, omit stop words
- liblinear, libsvm, and mallet are three excellent tools

Beyond Bag of Words

- Simple! Maybe too simple?
- All **structure**, i.e., relationships between words, is lost.
- "Michael ate the fish."
- "The fish ate Michael."

Structural Features I: Bigrams

- Count all pairs of adjacent words, throw them in with our original bag of words.

…
paribas_harewood(208)=1.0
harewood_avenue(209)=1.0
avenue_london(210)=1.0
london_nw(211)=1.0
…
chief_credit(664)=1.0
credit_officer(665)=1.0
…
of_one(666)=1.0
one_and(667)=1.0
and_only(668)=1.0
only_is(669)=1.0
is_you(670)=1.0

you_that(671)=1.0
…
meet_attend(672)=1.0
attend_icml(673)=1.0

- Trigrams, 4-grams, …, N-grams…

Structural Features II: Parts of Speech

- "Go milk the cow." *milk* is a Verb Phrase
- "Drink your milk." *milk* is a Noun Phrase
- A **part-of-speech tagger** can tell the difference.

milk_VP=1.0
milk_NP=1.0

Structural Features III: Parsing

- Matt Post and Shane Bergsma. "Explicit and Implicit Syntactic Features for Text Classfication" http://cs.jhu.edu/~post/papers/post-bergsma-acl13.pdf

Structural Features III: Parsing

"Do not tell them our products contain asbestos."

"Tell them our products do not contain asbestos."

Extreme Overfitting

- With say 100s of documents and 10000s of features, overfitting is easy.
- E.g., for binary classification with linear separator, if every document has a unique word, give + weight to those for + documents, − weight to those of − documents. Perfect fit!
- For n documents, if each had a unique word, a linear classifier could use n weights out of the 10000 to be nonzero. A decision tree could be built with n nodes.

Sparsity from SVMs

- Recall that SVMs try to minimize the norm of the weights, $||\mathbf{w}||^2$, while getting training points on the correct side of the margin.
- Suppose features coded ± 1. In an SVM, weight for each word would have to be ± 1 to satisfy $y_i \mathbf{w}^\mathsf{T} \mathbf{x}_i \geq 1$; then $||\mathbf{w}||^2 = n$.
- If *one* word can discriminate, can use weight vector with $||\mathbf{w}||^2 = 1$
- SVMs prefer sparse, simple solutions, can avoid overfitting even when $p >> n$.

Sparsity from Other Models

- Recall linear regression and logistic regression both work by finding the $\mathbf{w}$ that minimizes training error.

$$\mathbf{w}^* = \min_{\mathbf{w}} J(\mathbf{w})$$

- If we can exactly fit every point (linear regression) or data are linearly separable (logistic regression) then $\mathbf{w}$ will not be unique, and we are prone to overfitting. (Depending on software, you will see warnings/crashes.)

Regularization [JWHT 6.2]

- Idea: Ask for a weight vector that has low training error but is also small:

$$\mathbf{w}^* = \min_{\mathbf{w}} J(\mathbf{w}) + \lambda||\mathbf{w}||$$

- This is called *regularization*.
  - The $\lambda||\mathbf{w}||$ is called a *penalty*; importance governed by λ
  - If $||\mathbf{w}||$ defined as $\sum_j |w^{(j)}|^2$, this is called a *ridge* penalty
  - If $||\mathbf{w}||$ defined as $\sum_j |w^{(j)}|$, this is an *L1* or *LASSO* penalty
- Ridge shrinks all weights toward zero; many may be nonzero though.
- LASSO will shrink some weights to *exactly* zero, performing **feature selection**

Feature Selection [JWHT Ch 6.1]

- Special case of *model selection* which we saw previously (E.g. with polynomial degree.)
- Add or remove features, check cross-validation error, pick favourite model
- Model selection *criteria* as alternatives to cross-validation are given in [JWHT Ch 6.1.3]. Can save computation.
  - Mallows' C_p, *AIC*, *BIC* all have the same form; training error plus a penalty for the number of parameters
  - Try different subsets of features, choose the one that gives lowest criterion value

Feature Selection vs. Feature Construction

Let n be number of examples, p be number of features.

- If you have e.g. $n > 10p$, you may not need to bother with feature selection.
  - Unless you want to discover that some features are redundant.
- If you have e.g. $n > 1000p$ and you are getting bad performance, you may want to construct features, or use a non-linear classifier, in case the problem is lack of fit.

Feature Selection Scenario

Suppose you have features $x_1, x_2, x_3, ..., x_p$, and label y.

What properties of x_1 might lead you to remove it from the model?

Feature Selection Scenario

Suppose you *only* look at the features $x_1, x_2, x_3, ..., x_p$.

Can you identify some that don't help predict y?

Unsupervised Learning

Unsupervised Learning

- Only features $x_1, ..., x_p$
- None is more important than the others
- Discover relationships among the instances and features
- Difficult to evaluate "performance" because task is ill-defined
- You might use the output of unsupervised learning for supervised learning

Dimensionality reduction

Dimensionality Reduction

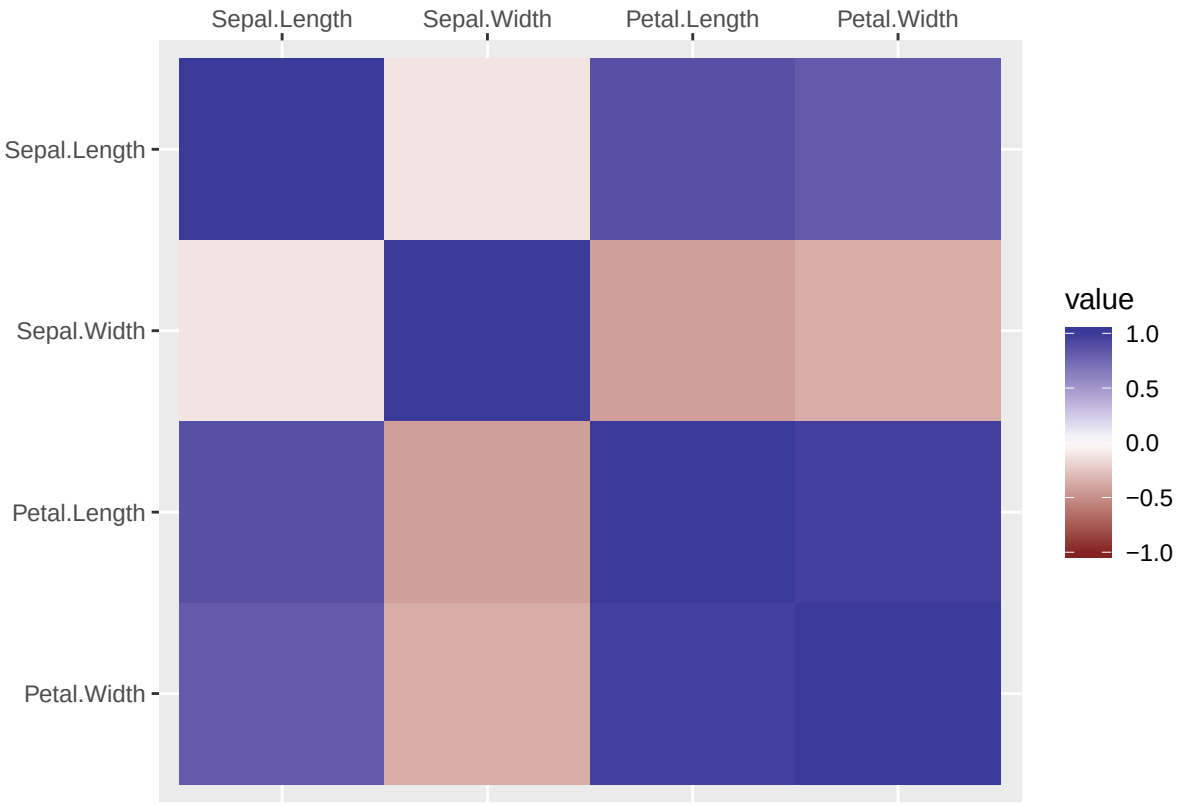
- Dimensionality reduction (or embedding) techniques:
  - Take data that has p features
  - Re-encode as data that has q features, $q < p$
  - Don't lose too much information

Dimensionality Reduction Techniques

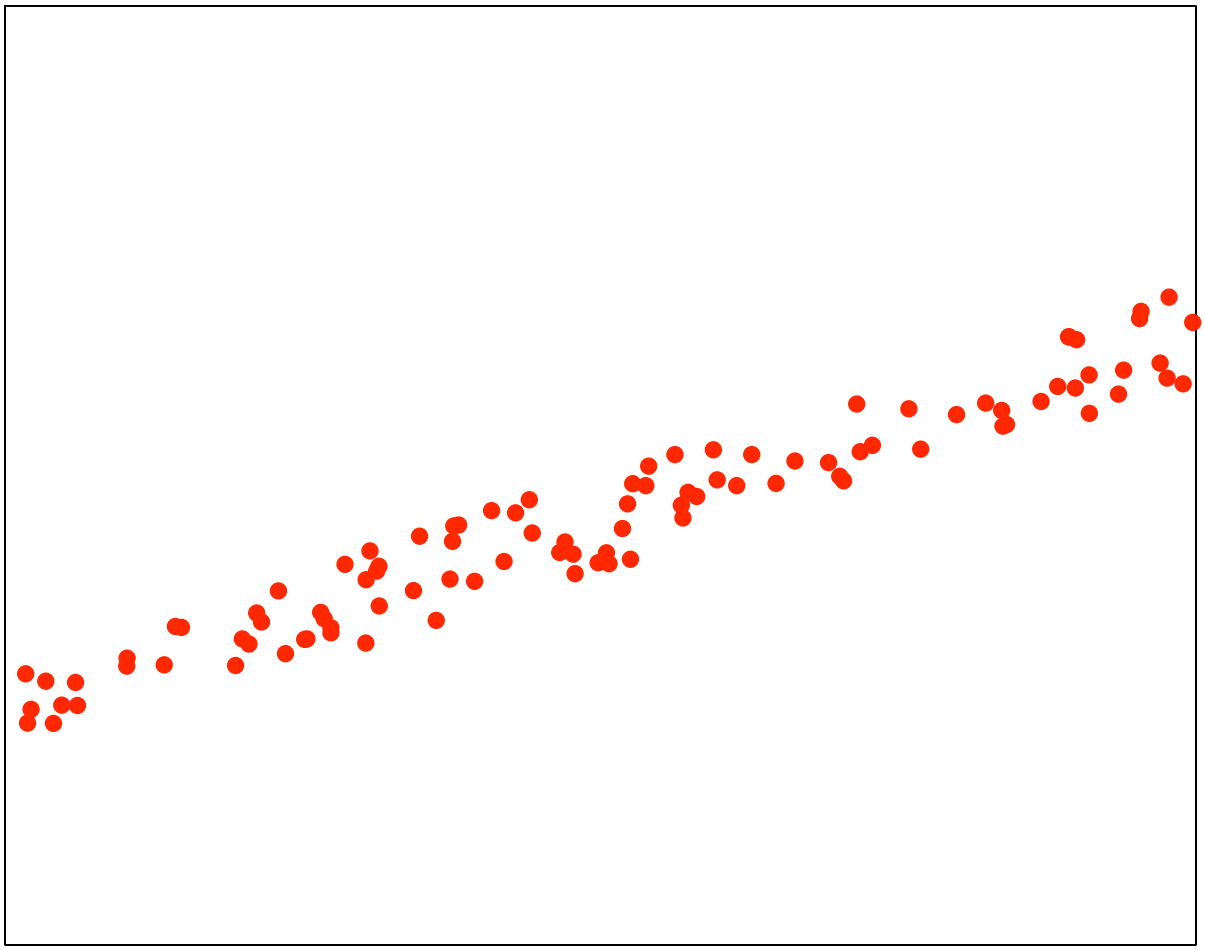
- Axis-aligned: Remove features that are well-predicted by other features
- Linear: **Principal components analysis** creates smaller set of new features that are weighted sums of existing features
- Non-linear: Create small set of new features that are non-linear functions of existing features
  - Kernel PCA
  - Independent components analysis
  - Self-organizing maps
  - Multi-dimensional scaling
  - **t-SNE: t-distributed Stochastic Neighbour Embedding**

Axis-aligned dimensionality reduction

Correlation matrix


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
value
1.0
0.5
0.0
−0.5
−1.0

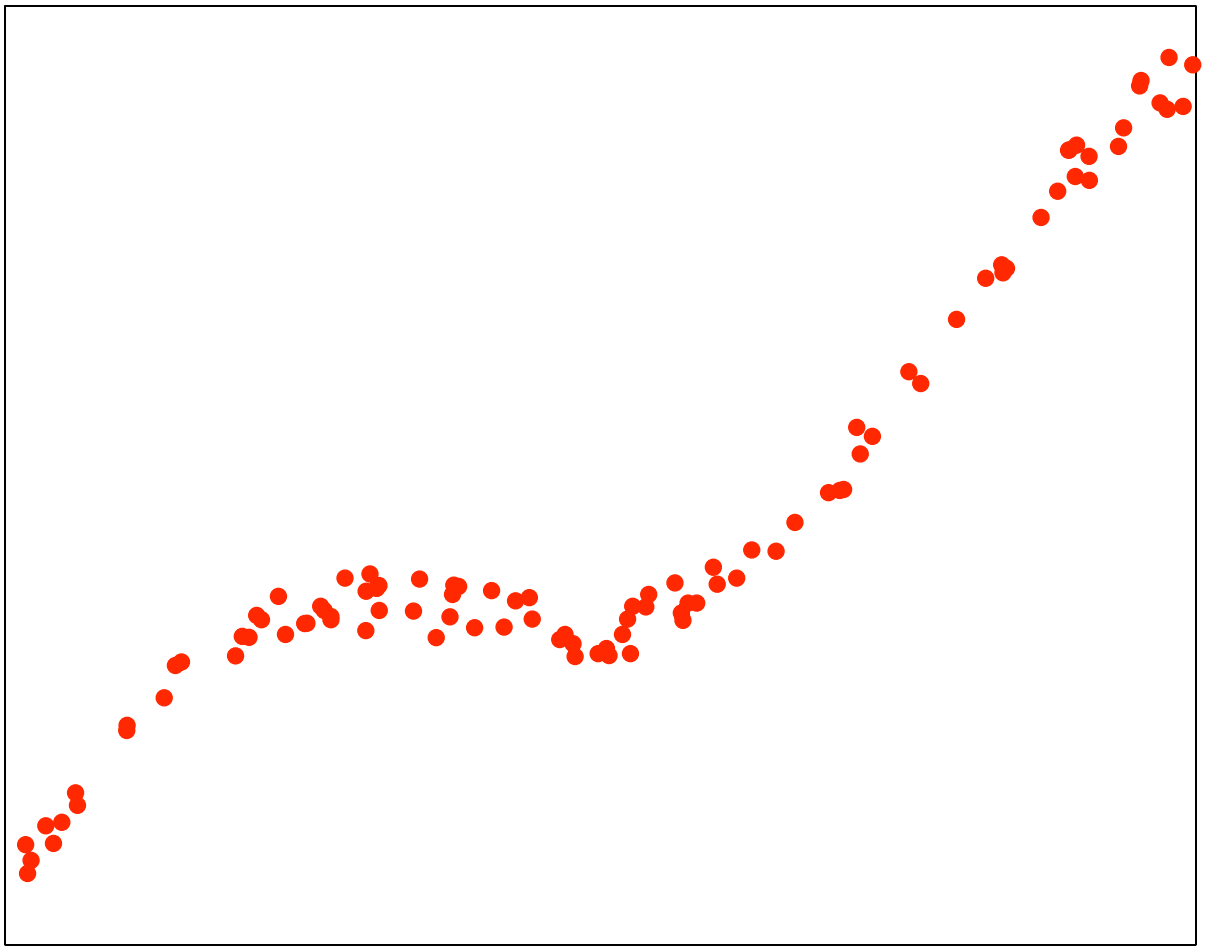
"True dimensionality" of this dataset?



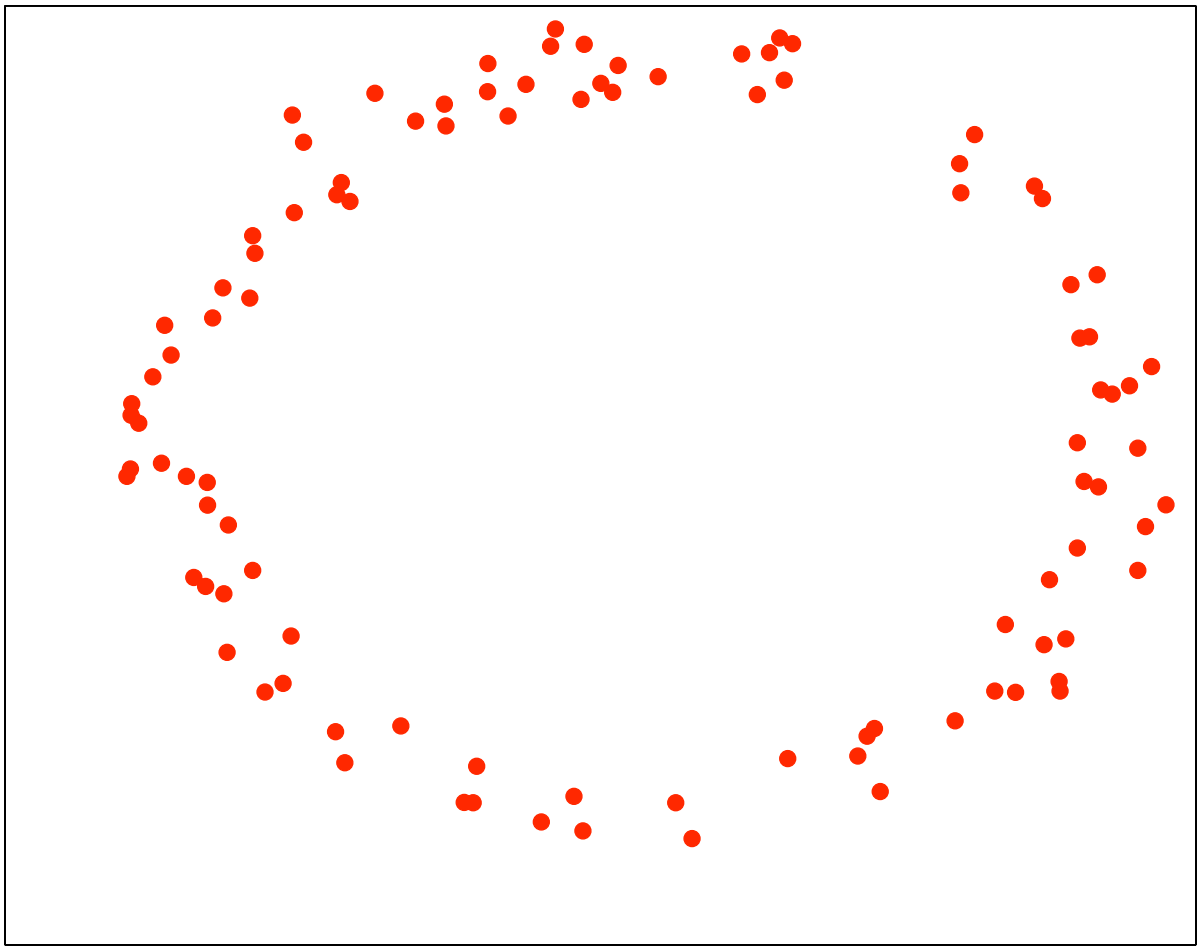
"True dimensionality" of this dataset?

- You may give me a model with $\ll n$ parameters ahead of time.
- How many additional numbers must you send to tell me approximately where a particular data point is?

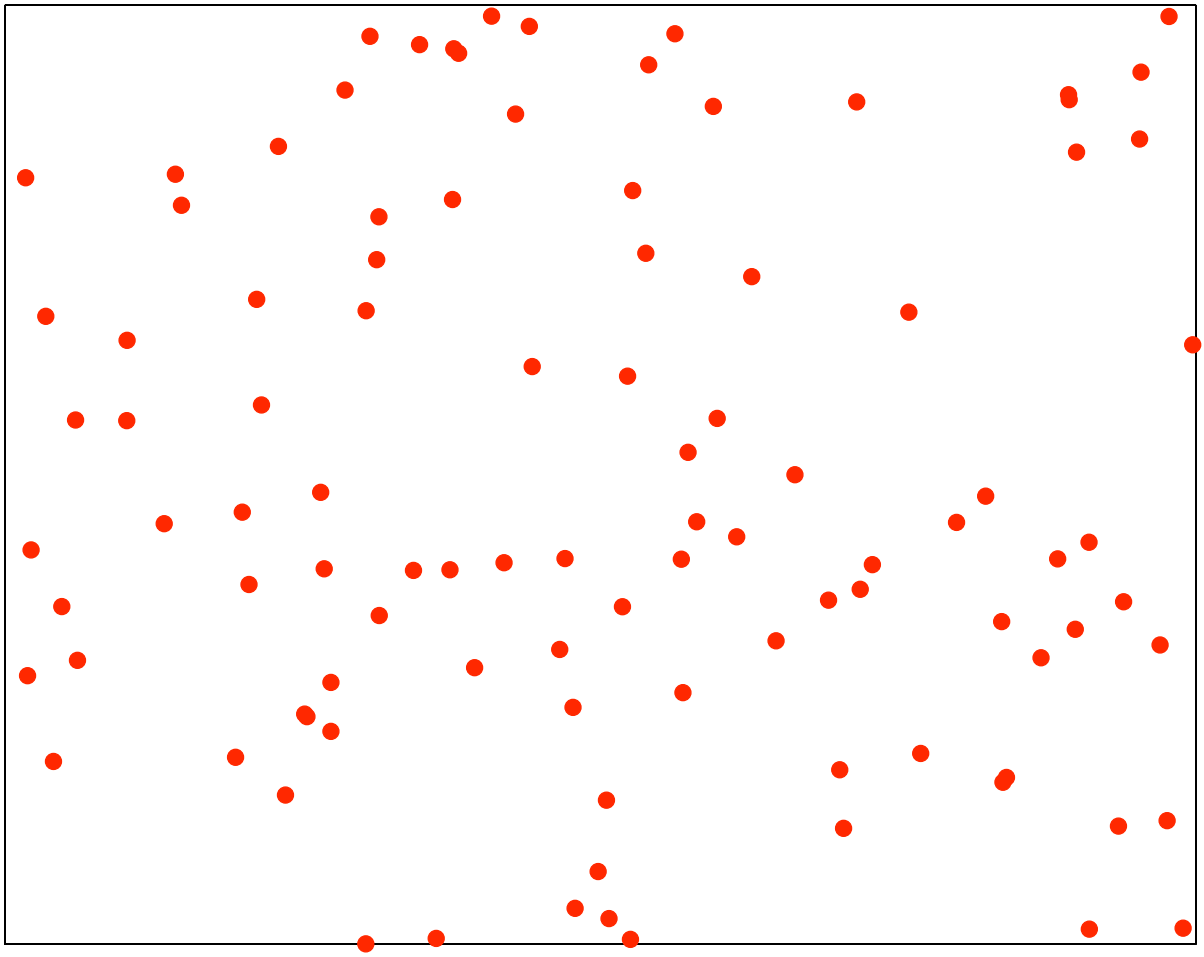
"True dimensionality" of this dataset?



"True dimensionality" of this dataset?



"True dimensionality" of this dataset?



Remarks

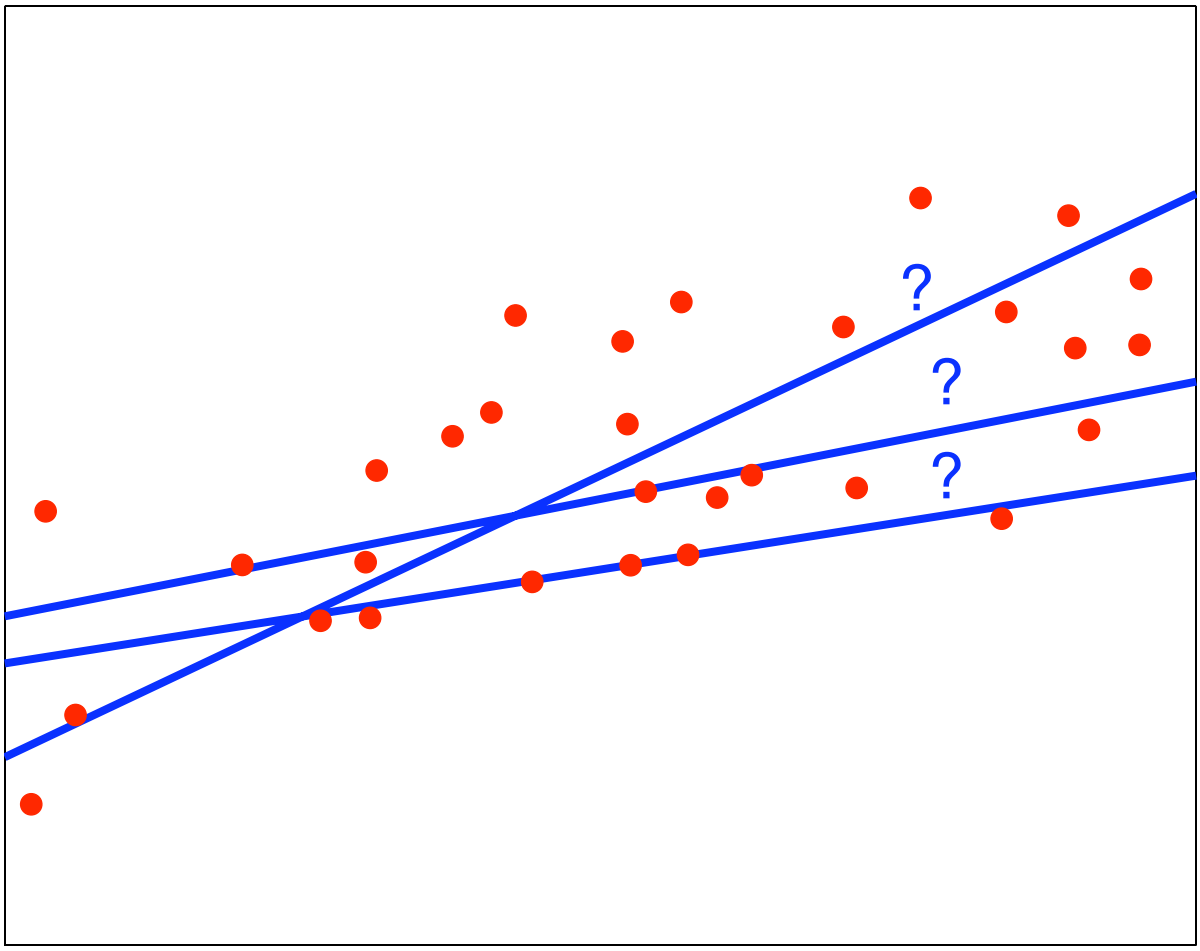
- All dimensionality reduction techniques are based on an implicit assumption that the data lies on (near) some *low-dimensional manifold*
- This is the case for the first three examples, which (almost) lie along a 1-dimensional manifold despite being plotted in 2D
- In the last example, no dimensionality reduction is possible without losing a lot of information

Principal Component Analysis (PCA) [JWHT 10.2]

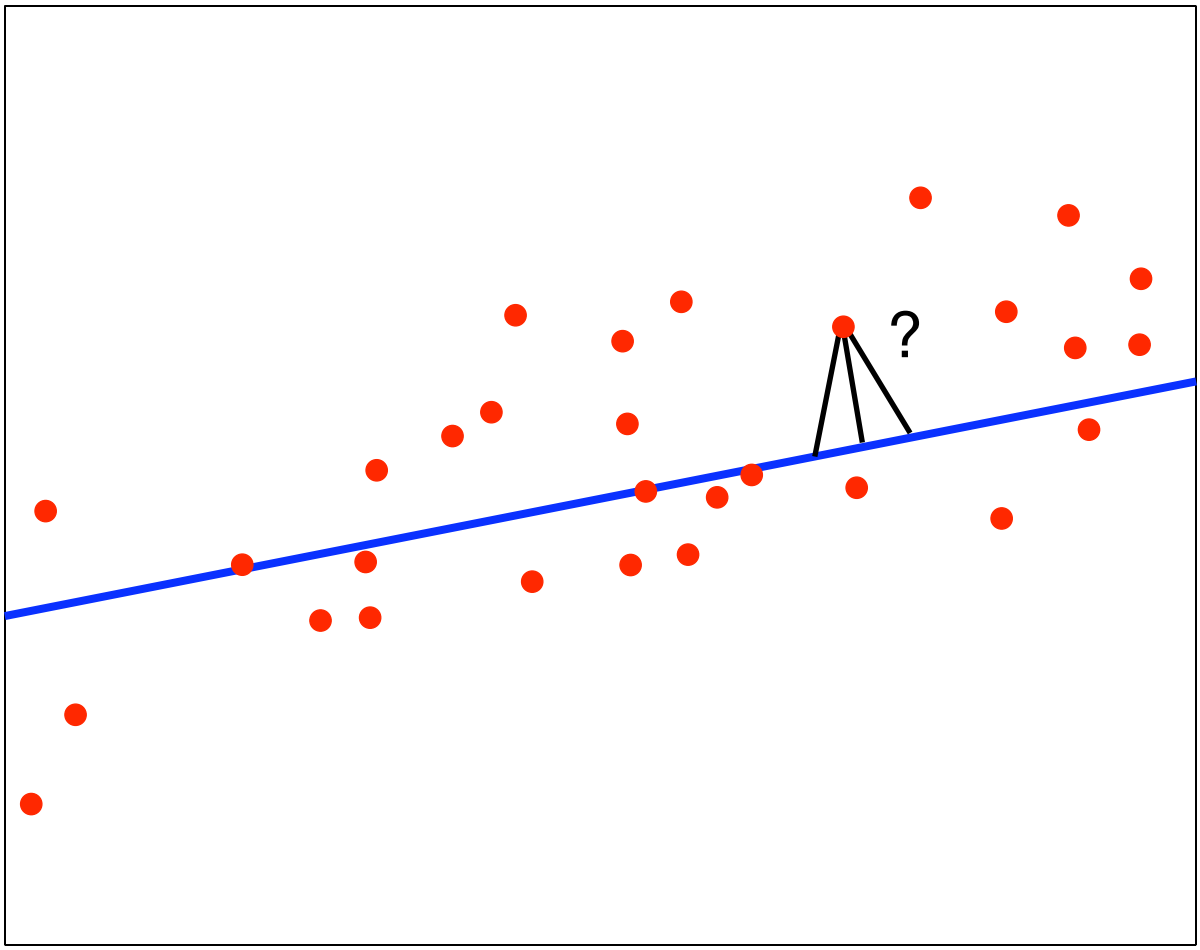
- Given: n instances, each being a length-p real vector.
- Suppose we want a 1-dimensional representation of that data, instead of p-dimensional.
- Specifically, we will:
  - Choose a line in $\mathbb{R}^p$ that "best represents" the data.
  - Assign each data object to a point along that line.

- Identifying a point on a line just requires a scalar: How far along the line is the point?

Which line is best?



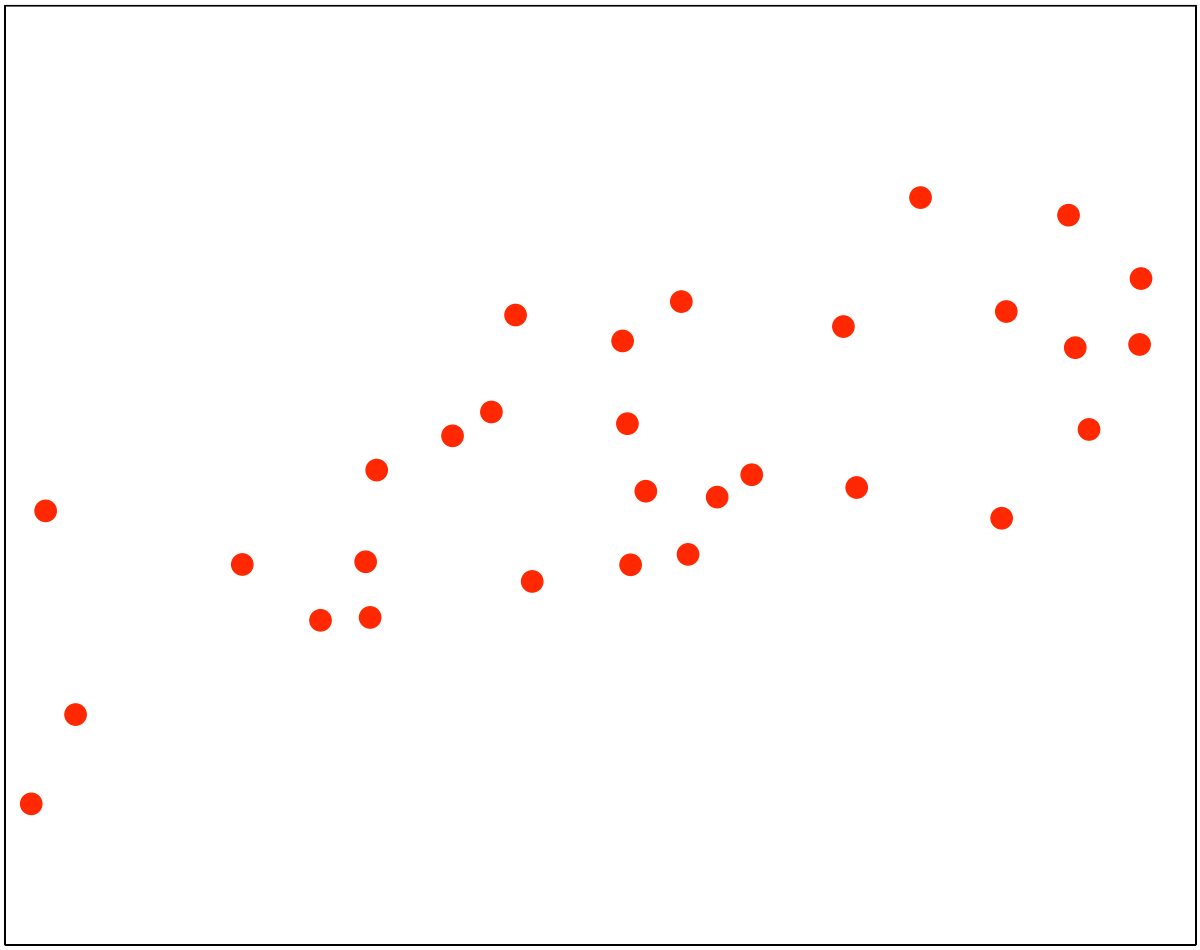
How do we assign points to lines?



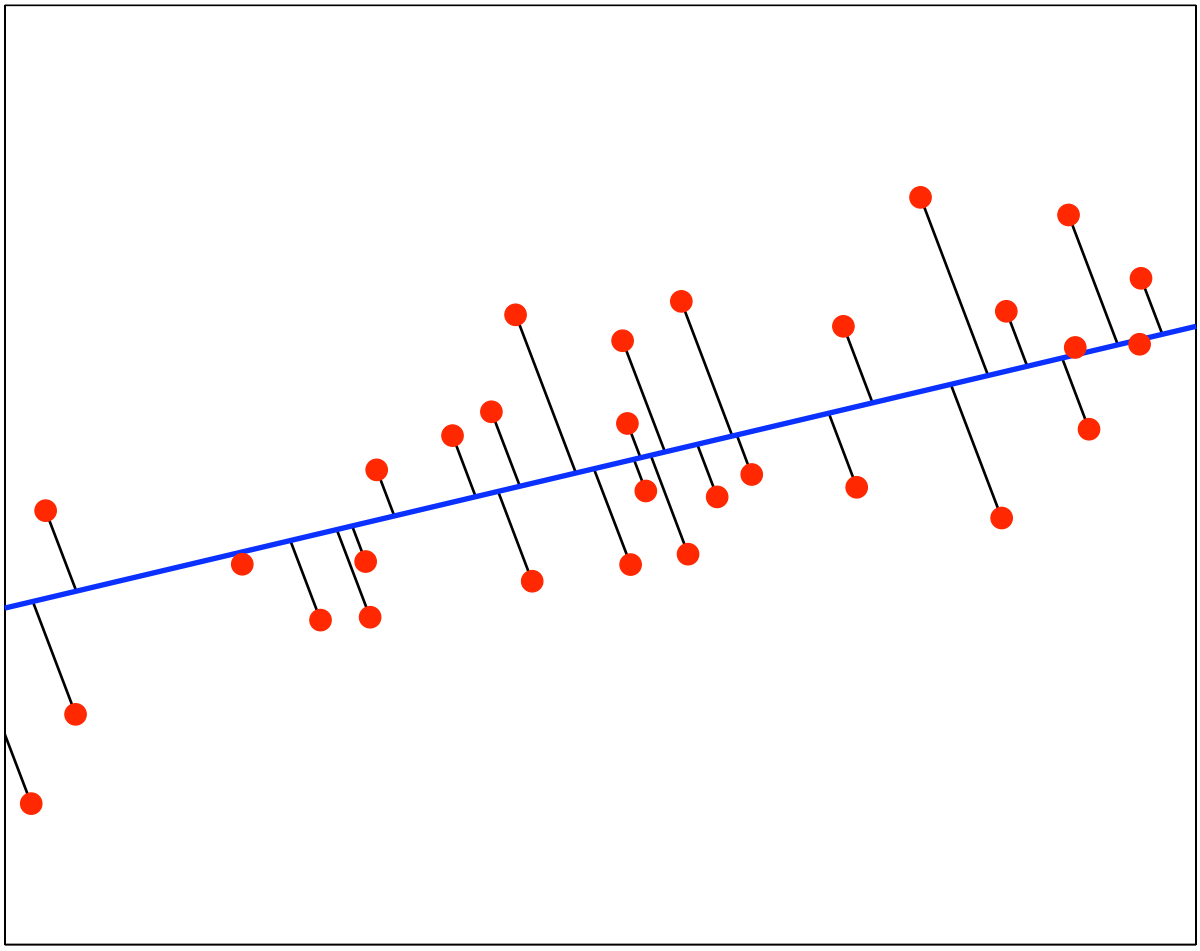
Reconstruction error

- Let the line be represented as $\mathbf{b} + \alpha\mathbf{v}$ for $\mathbf{b}, \mathbf{v} \in \mathbb{R}^p$, $\alpha \in \mathbb{R}$.
  For convenience assume $\|\mathbf{v}\| = 1$.
- Each instance $\mathbf{x}_i$ is associated with a point on the line $\hat{\mathbf{x}}_i = \mathbf{b} + \alpha_i\mathbf{v}$.
  - Instance $\mathbf{x}_i$ is *encoded* as a single α_i
  - This is the new (and only) feature for instance i

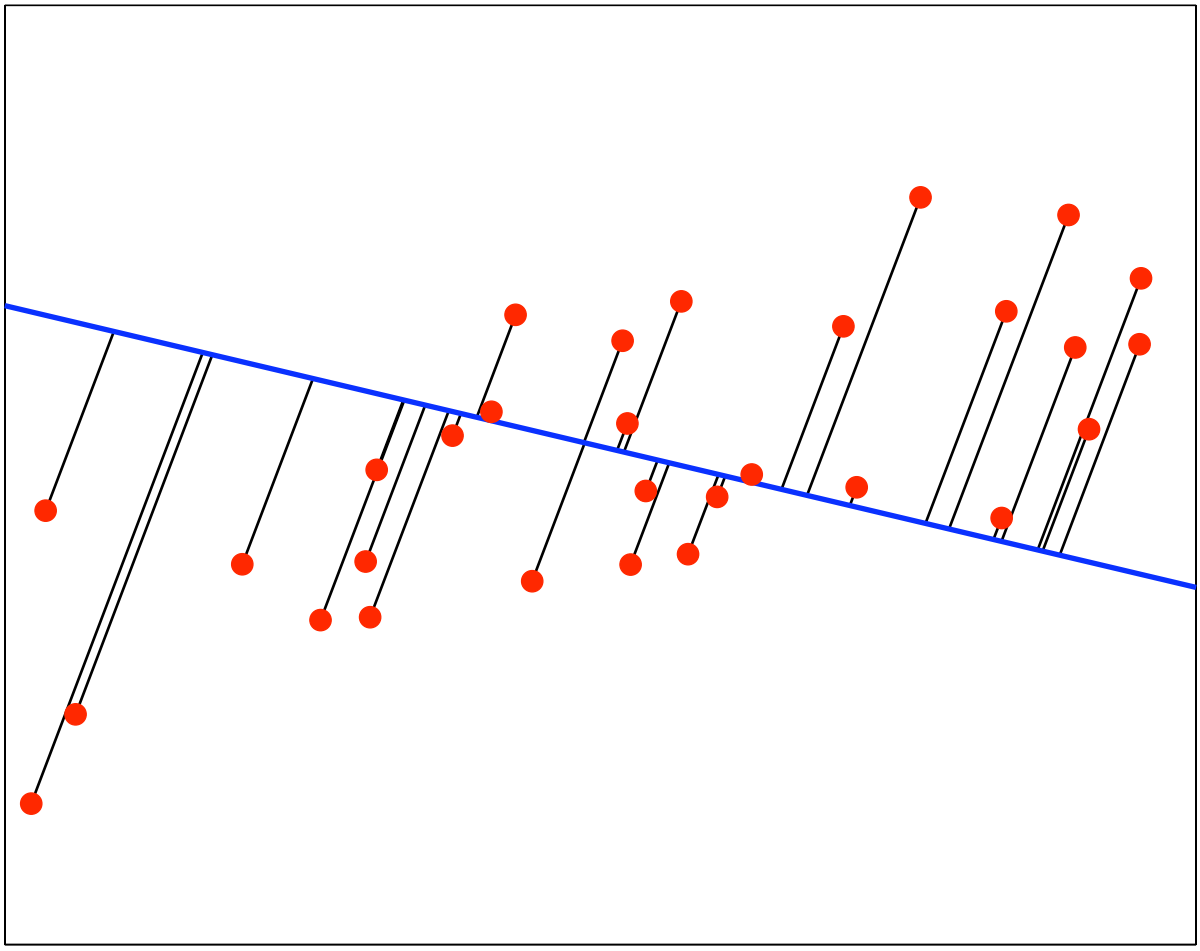
Example data



Example with $\mathbf{v} \propto (1, 0.3)$



Example with $\mathbf{v} \propto (1, -0.3)$



Minimizing reconstruction error

- We want to choose $\mathbf{b}$, $\mathbf{v}$, and the α_i to minimize the total reconstruction error over all data points, measured using Euclidean distance:

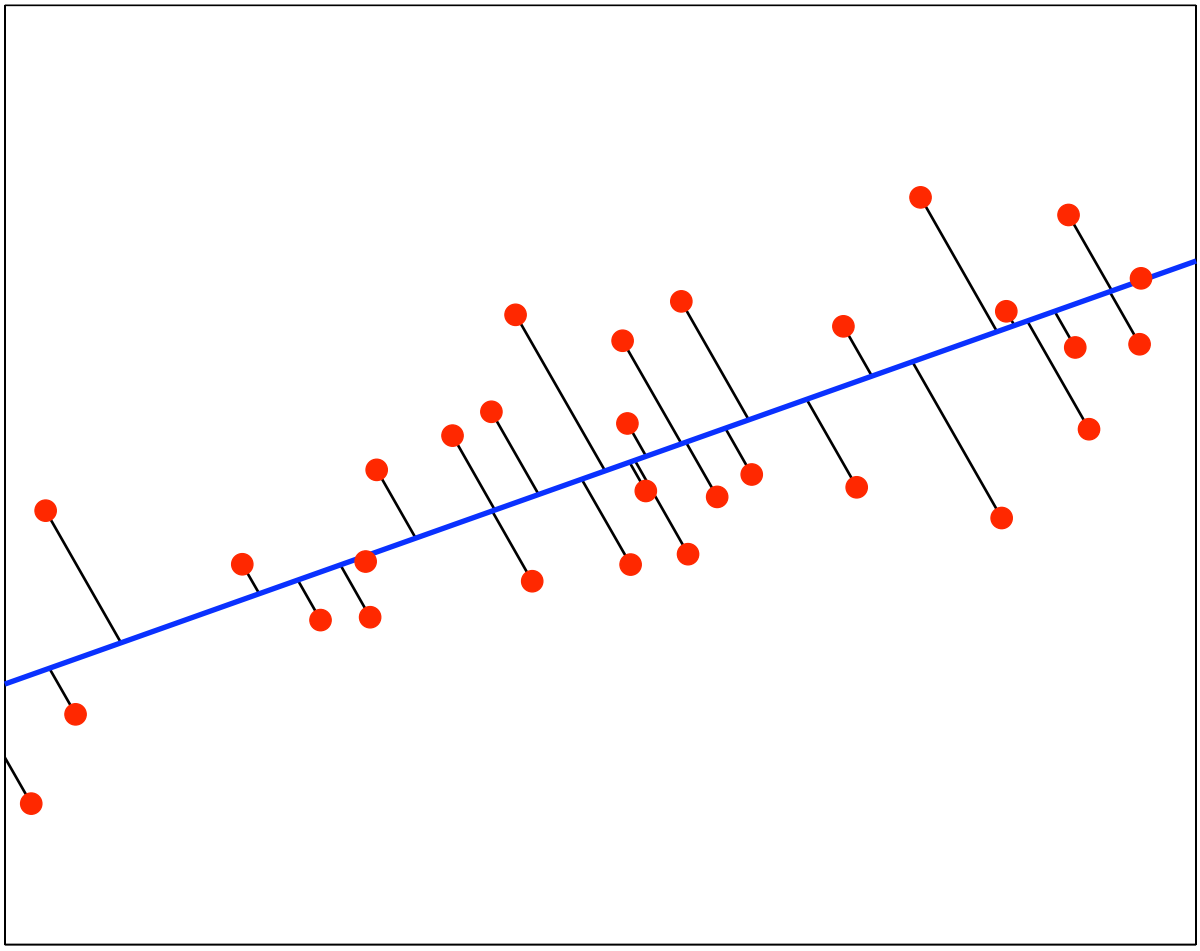
$$R = \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

- ***Difference from regression:*** Given the new feature α_i, reconstruct ***all dimensions of the x_i***. All are equally important.

$$\begin{array}{ll} \min & \sum_{i=1}^{n} \|\mathbf{x}_i - (\mathbf{b} + \alpha_i \mathbf{v})\|^2 \quad \text{w.r.t. } \mathbf{b}, \mathbf{v}, \alpha_i, i = 1, \ldots n \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 \end{array}$$

- Can be computed by ***Singular Value Decomposition***

Example with optimal line: $\mathbf{b} = (0.54, 0.52)$, $\mathbf{v} \propto (1, 0.45)$



Reduction to d dimensions

- $\mathbf{b}$, $\mathbf{v}$, and the α_i can be computed easily in polynomial time. The α_i give a 1D representation.
- More generally, we can create a d-dimensional representation of our data by projecting the instances onto a hyperplane $\mathbf{b} + \alpha^1 \mathbf{v}_1 + \ldots + \alpha^d \mathbf{v}_d$.

Singular Value Decomposition

- $\mathbf{b}$, the eigenvalues λ, the $\mathbf{v}_j$, and the projections of the instances can all be computing in polynomial time, e.g. using (thin) Singular Value Decomposition.

$$X_{n \times p} = U_{n \times p} D_{p \times p} V^{\mathsf{T}}_{p \times p}$$

- Columns of U are left-eigenvectors, diagonal of D are sqrts of eigenvalues ("singular values"), V are right-eigenvectors
- Typically D is sorted by magnitude.
  ***First** d **columns of** U **are new representation of** X.*

- To encode new feature vector $\mathbf{x}$ as a vector $\mathbf{u}$:

  $\mathbf{u} = D^{-1}V^{\mathsf{T}}\mathbf{x}$, take first d elements.

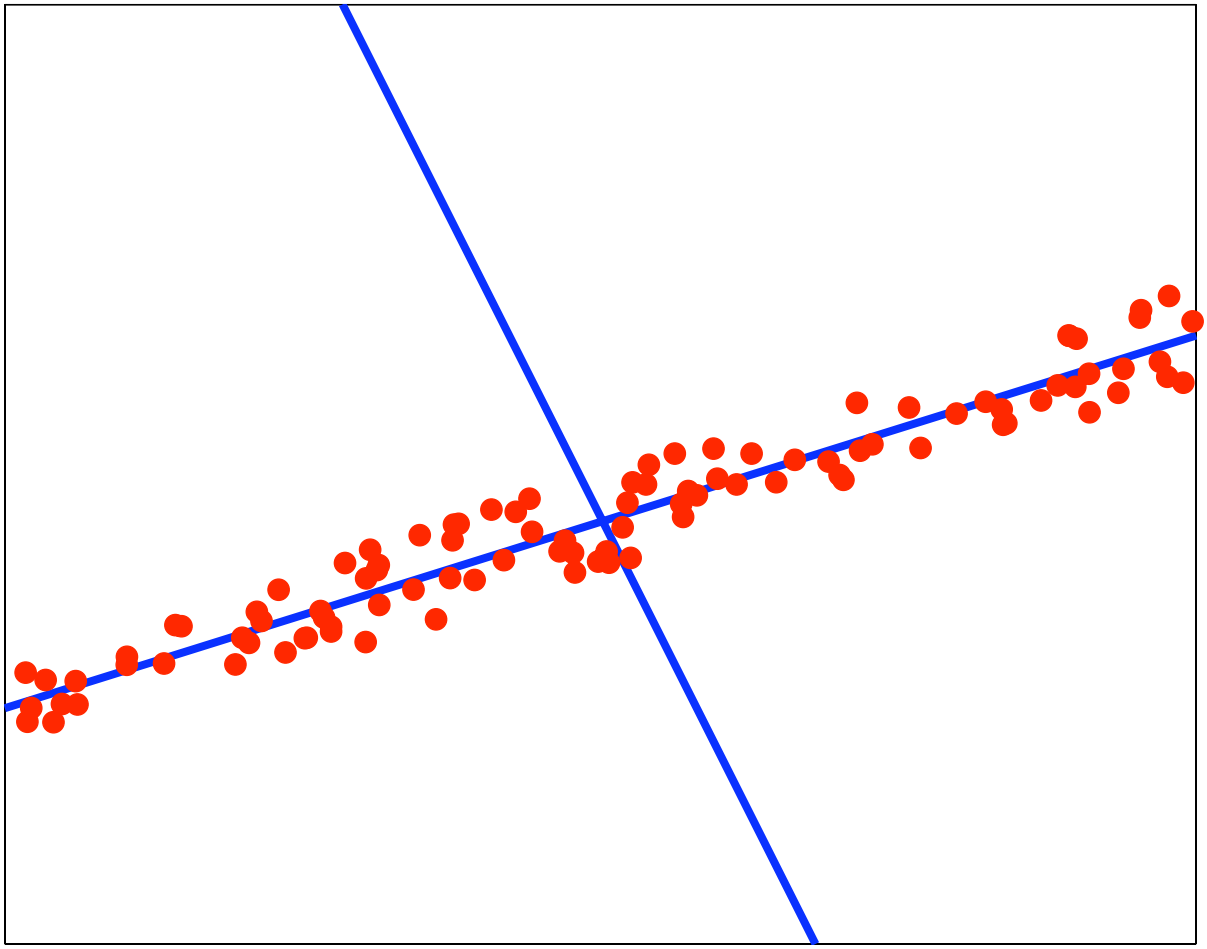
Eigenvalue Magnitudes [JWHT p.383]

- The magnitude of the j^{th}-largest eigenvalue, λ_j, tells how much variability in the data is captured by the j^{th} principal component
- When the eigenvalues are sorted in decreasing order, the proportion of the variance captured by the first d components is:

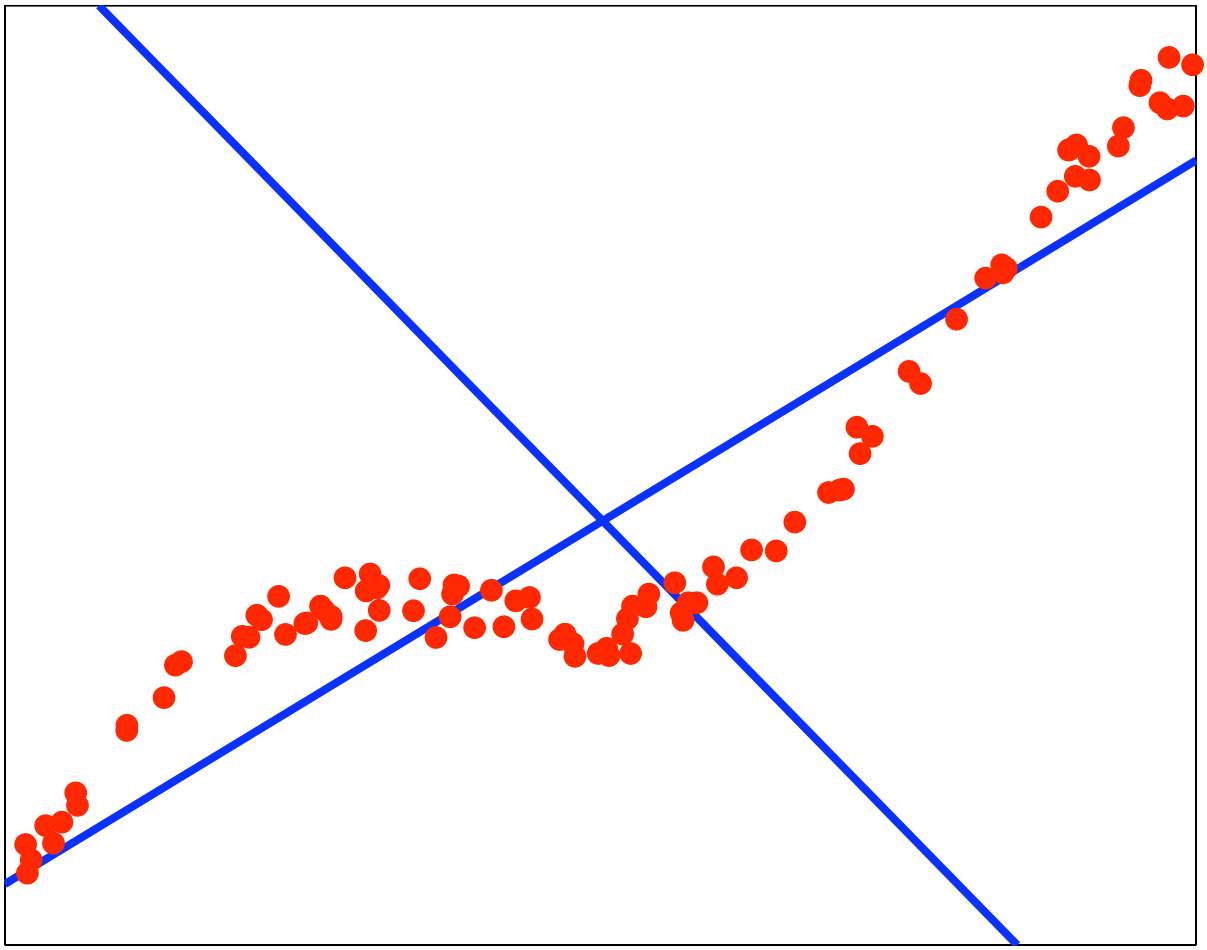
$$\frac{\lambda_1 + \cdots + \lambda_d}{\lambda_1 + \cdots + \lambda_d + \lambda_{d+1} + \cdots + \lambda_n}$$

- So if a "big" drop occurs in the eigenvalues at some point, that suggests a good dimension cutoff

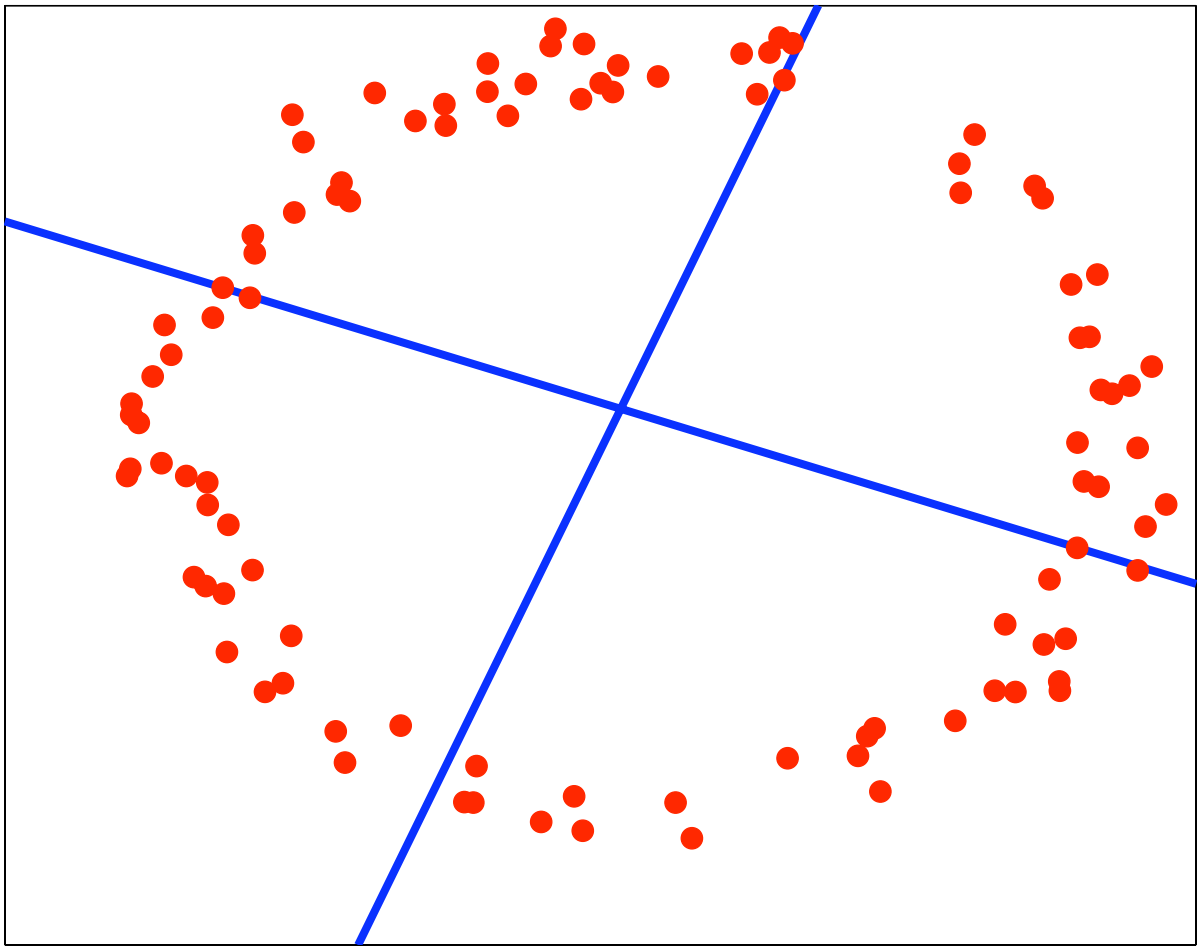
Example: $\lambda_1 = 0.0938, \lambda_2 = 0.0007$



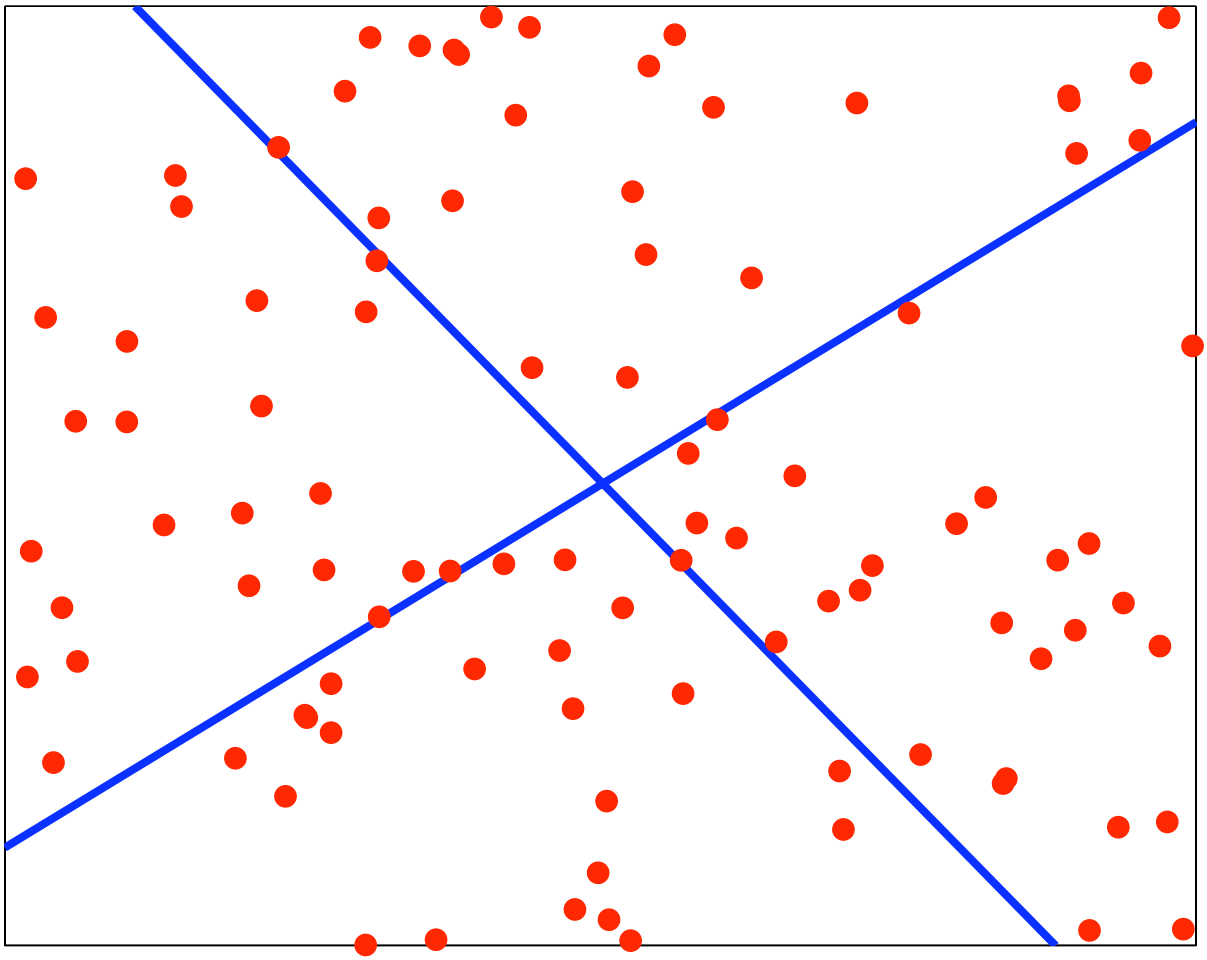
Example: $\lambda_1 = 0.1260, \lambda_2 = 0.0054$



Example: $\lambda_1 = 0.0884, \lambda_2 = 0.0725$



Example: $\lambda_1 = 0.0881, \lambda_2 = 0.0769$



More remarks [JWHT 10.2.3]

- Outliers have a big effect on the covariance matrix, so they can affect the eigenvectors quite a bit
- A simple examination of the pairwise distances between instances can help discard points that are very far away (for the purpose of PCA)
- If the variances in the original dimensions vary considerably, they can "muddle" the true directions. **Typically we normalize each dimension prior to PCA.**
- In certain cases, the eigenvectors are meaningful; e.g. in image processing, they can be displayed as images ("e.g. eigenfaces")

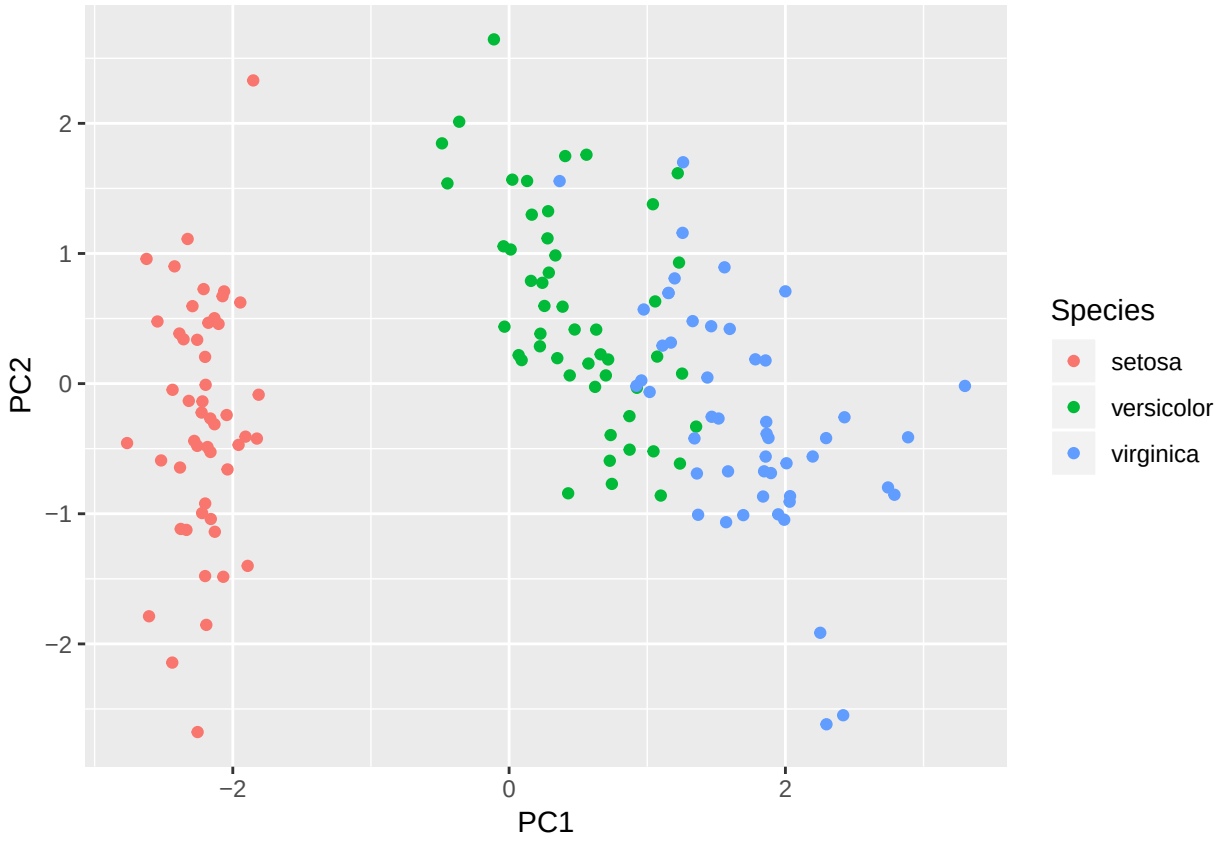
Uses of PCA

- Pre-processing for a supervised learning algorithm, e.g. for image data, robotic sensor data
- Used with great success in image and speech processing
- Visualization

- Exploratory data analysis
- Removing the linear component of a signal (before fancier non-linear models are applied)

Visualization: Iris dataset

```
library(ggplot2)
pca <- prcomp(iris[,1:4], scale. = TRUE, retx = TRUE)
irispca <- cbind(iris, pca$x)
ggplot(data = irispca, aes(x = PC1, y = PC2, colour = Species)) + geom_point()
```



Beyond PCA: Nonlinear dimensionality reduction

- Kernel PCA (but you don't get the eigenvectors)
- Self-Organizing Maps
- Isomap
- Locally Linear Embedding
- http://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

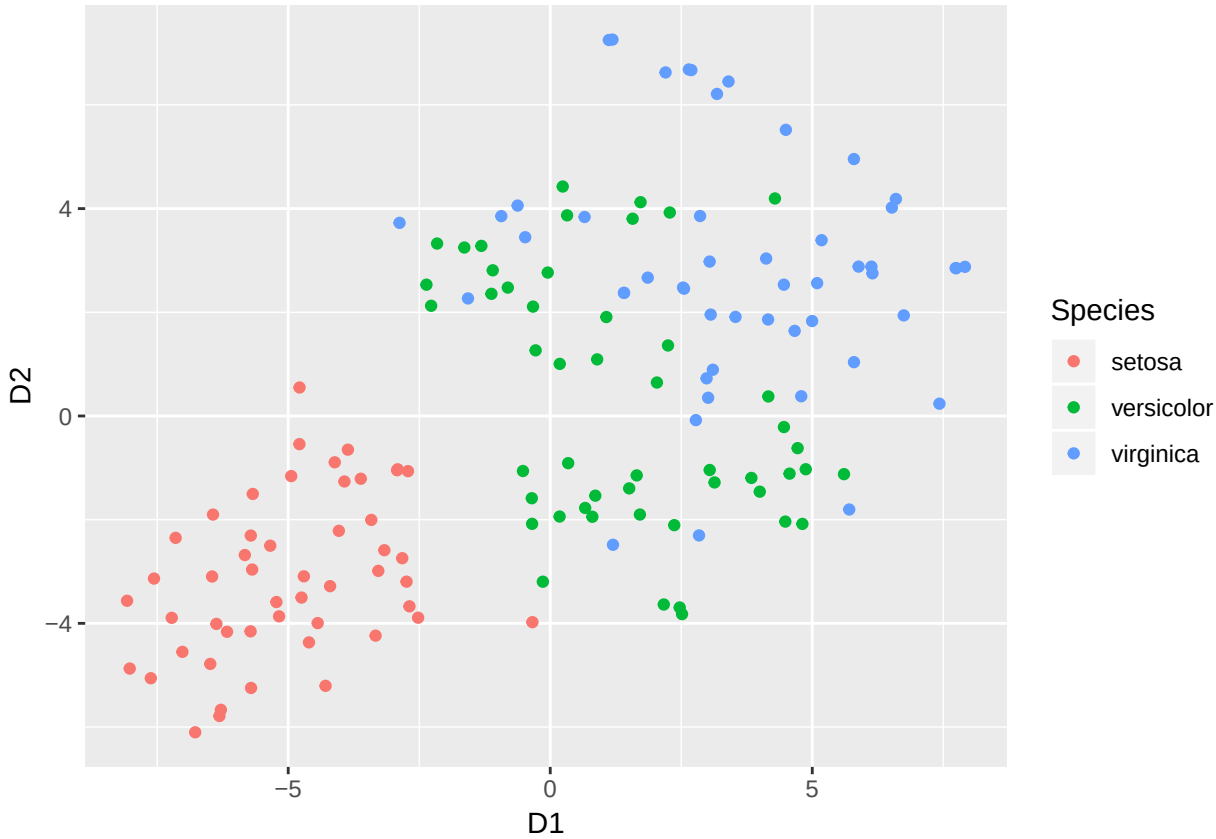
Stochastic Neighbour Embedding

- Given: "High" (>2) dimensional data, similarity measure

- Make up a probability distribution based on similarity that produces pairs of points
  - Probability of pairs with similar features should be larger than pairs of dissimilar features
- Choose smaller dimensionality (usually 2), similarity measure, form of probability distribution for pairs
- Find arrangement of points in low dimensional space that give similar pair distribution as one from high dimensional space
- **Main idea: if a pair is nearby in high dimensions, should be nearby in low dimensions. If a pair is not nearby in high dimensions, don't care about it.**

Iris tSNE

```
library(tsne)
tsne = tsne(iris[,1:4], perplexity=50)
colnames(tsne) <- c("D1","D2"); iristsne <- cbind(iris, tsne)
ggplot(data = iristsne, aes(x = D1, y = D2, colour = Species)) + geom_point()
```



Dimensionality Reduction

- PCA
  - Linear
  - Can embed new points
- tSNE
  - Nonlinear
  - No way to embed new points (without re-running)

Clustering

What is clustering?

- Clustering is grouping similar objects together.
  - To establish prototypes, or detect outliers.
  - To simplify data for further analysis/learning.
  - To visualize data (in conjunction with dimensionality reduction)
- Clusterings are usually not "right" or"wrong" – different clusterings/clustering criteria can reveal different things about the data.

Clustering Algorithms

- Clustering algorithms:
  - Employ some notion of distance between objects
  - Have an explicit or implicit criterion defining what a good cluster is
  - Heuristically optimize that criterion to determine the clustering
- Some clustering criteria/algorithms have natural probabilistic interpretations

K-means clustering

- One of the most commonly-used clustering algorithms, because it is easy to implement and quick to run.
- Assumes the objects (instances) to be clustered are p-dimensional vectors, $\mathbf{x}_i$.
- Uses a distance measure between the instances (typically Euclidean distance)
- The goal is to *partition* the data into K disjoint subsets

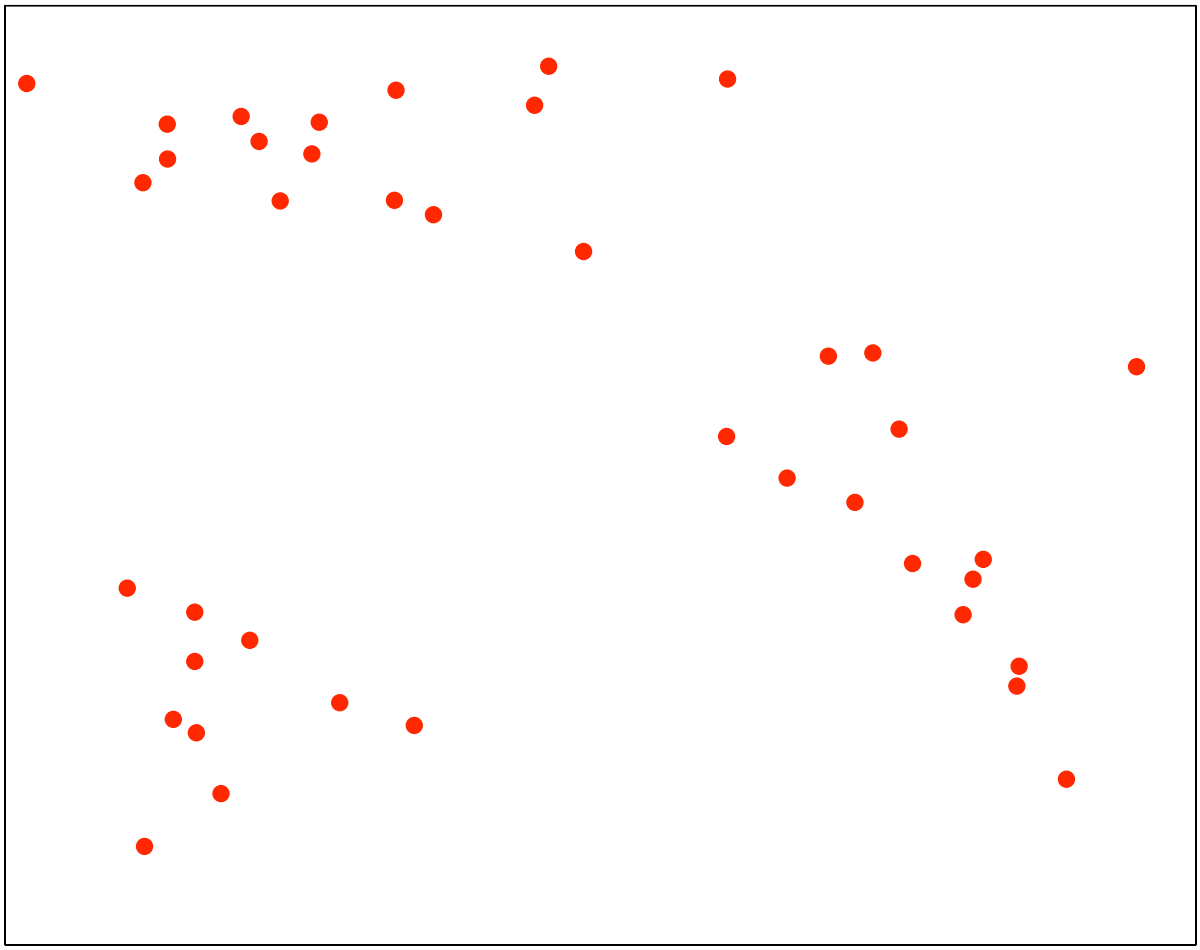
K-means clustering

- Inputs:
  - A set of p-dimensional real vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.
  - K, the desired number of clusters.
- Output: A mapping of the vectors into K clusters (disjoint subsets), $C : \{1, \ldots, n\} \mapsto \{1, \ldots, K\}$.

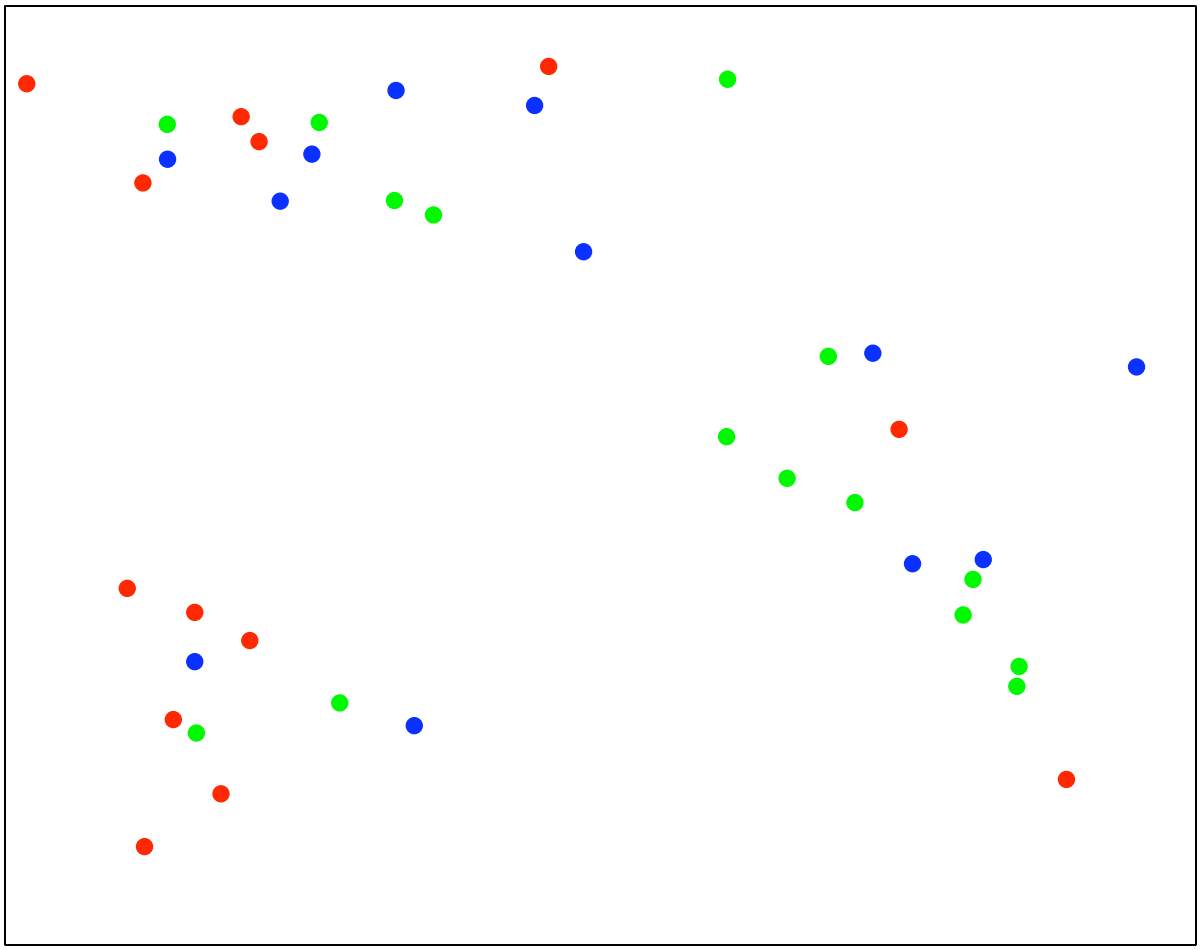
1. Initialize C randomly.
2. Repeat:
   1. Compute the *centroid* of each cluster (the mean of all the instances in the cluster)
   2. Reassign each instance to the cluster with closest centroid

   until C stops changing.

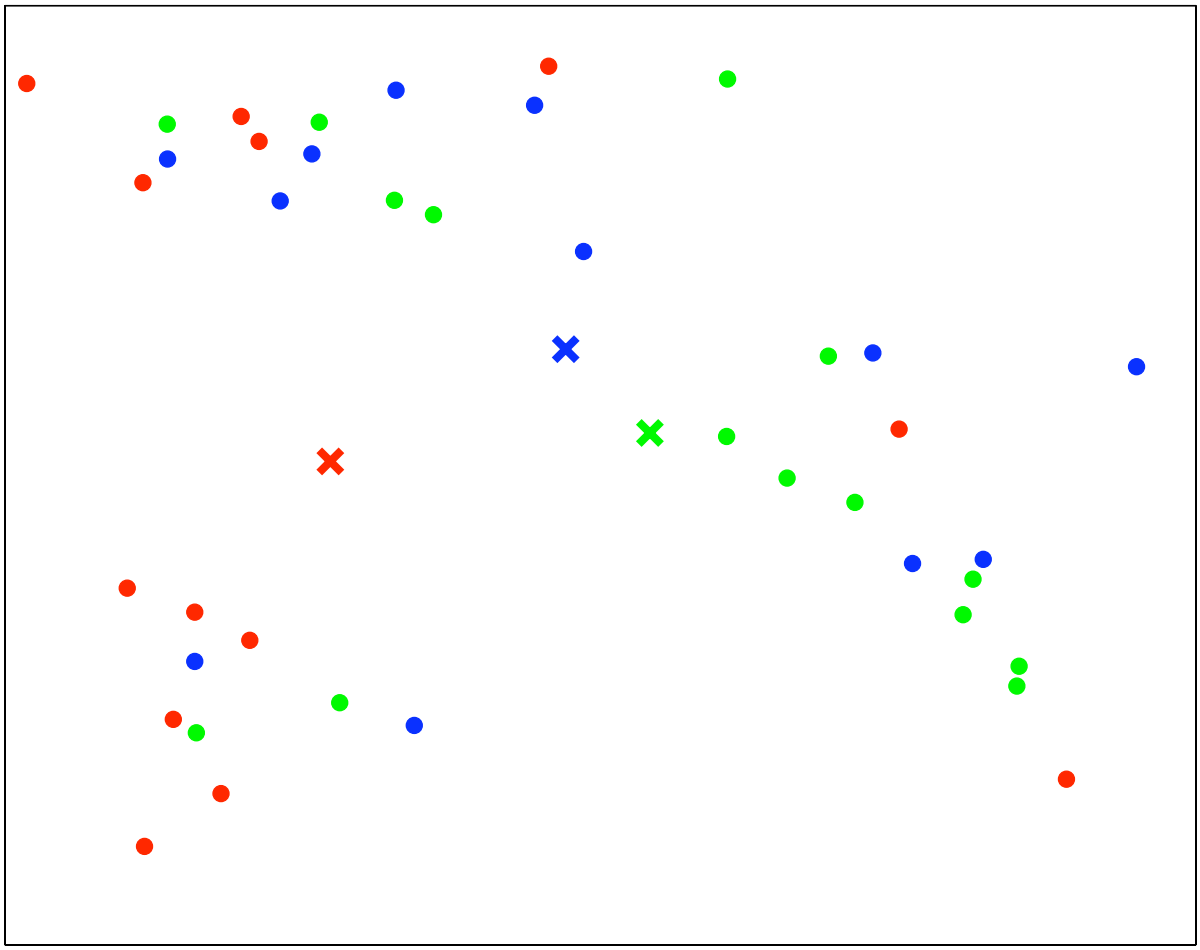
Example: initial data



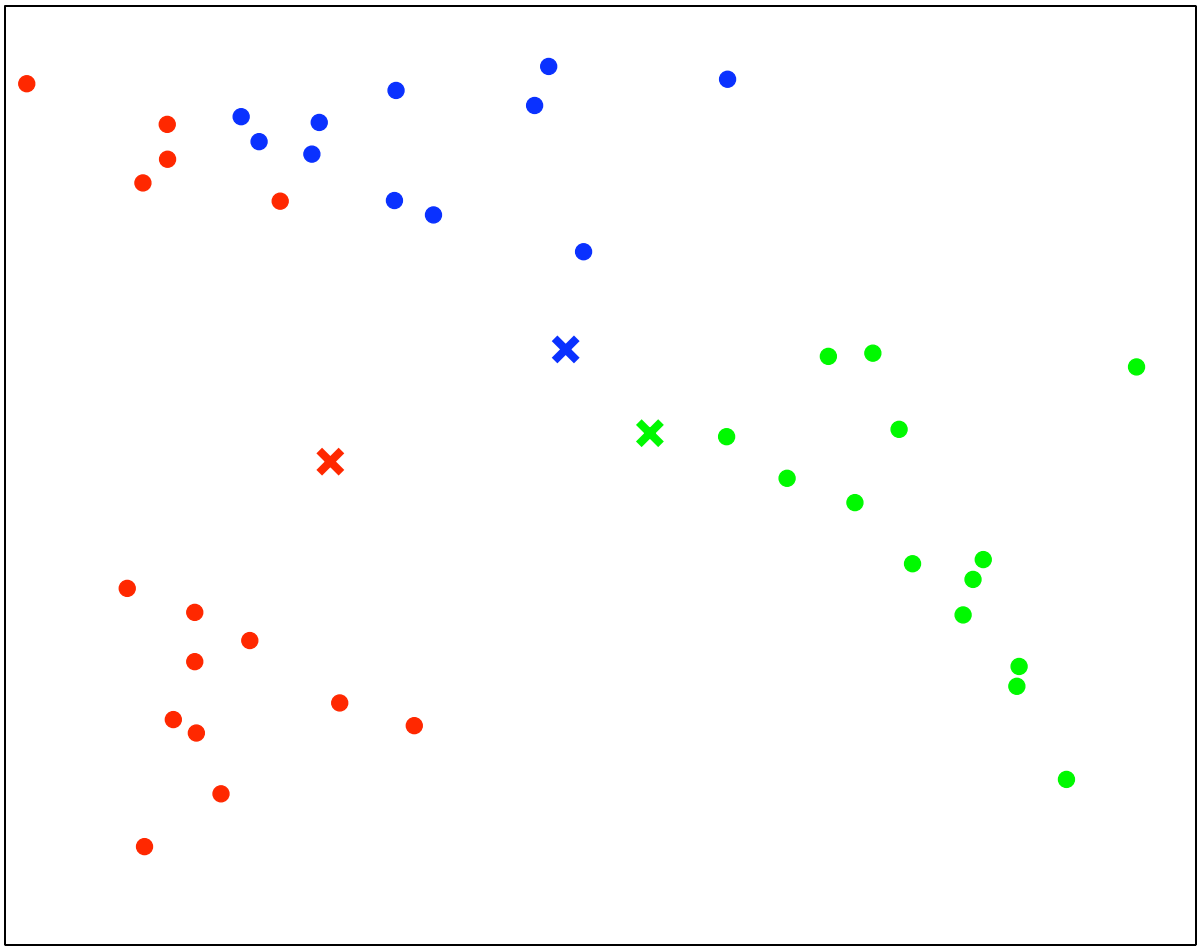
Example: assign into 3 clusters randomly



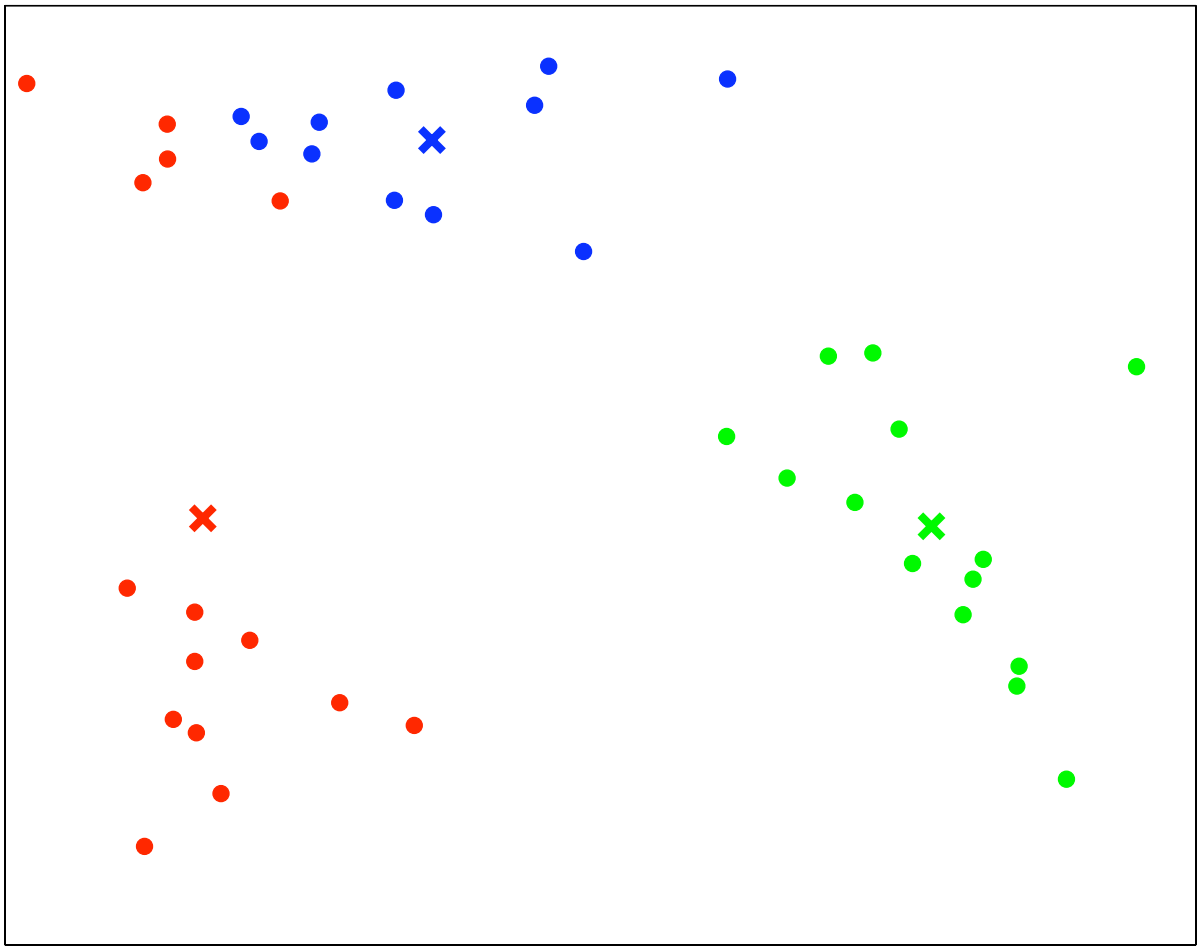
Example: compute centroids



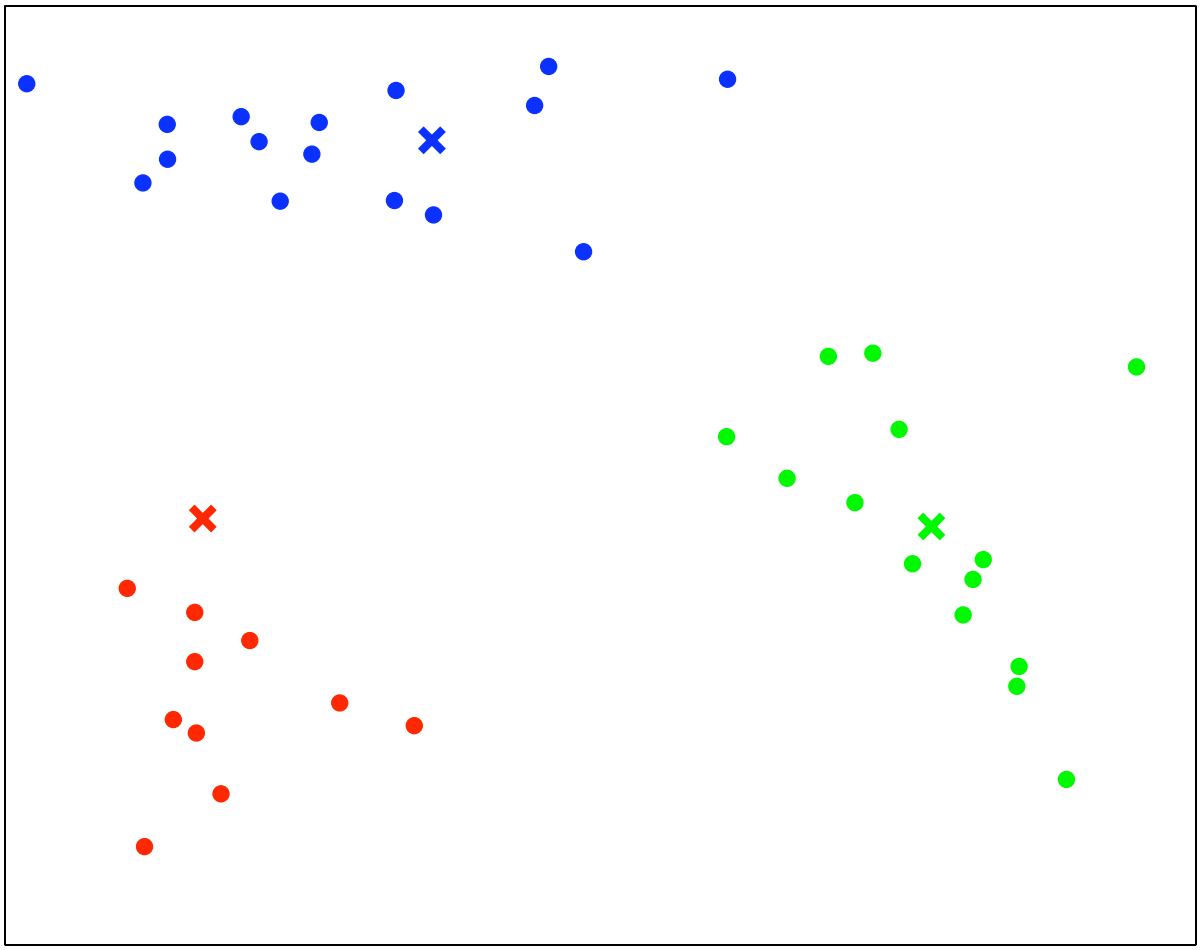
Example: reassign clusters



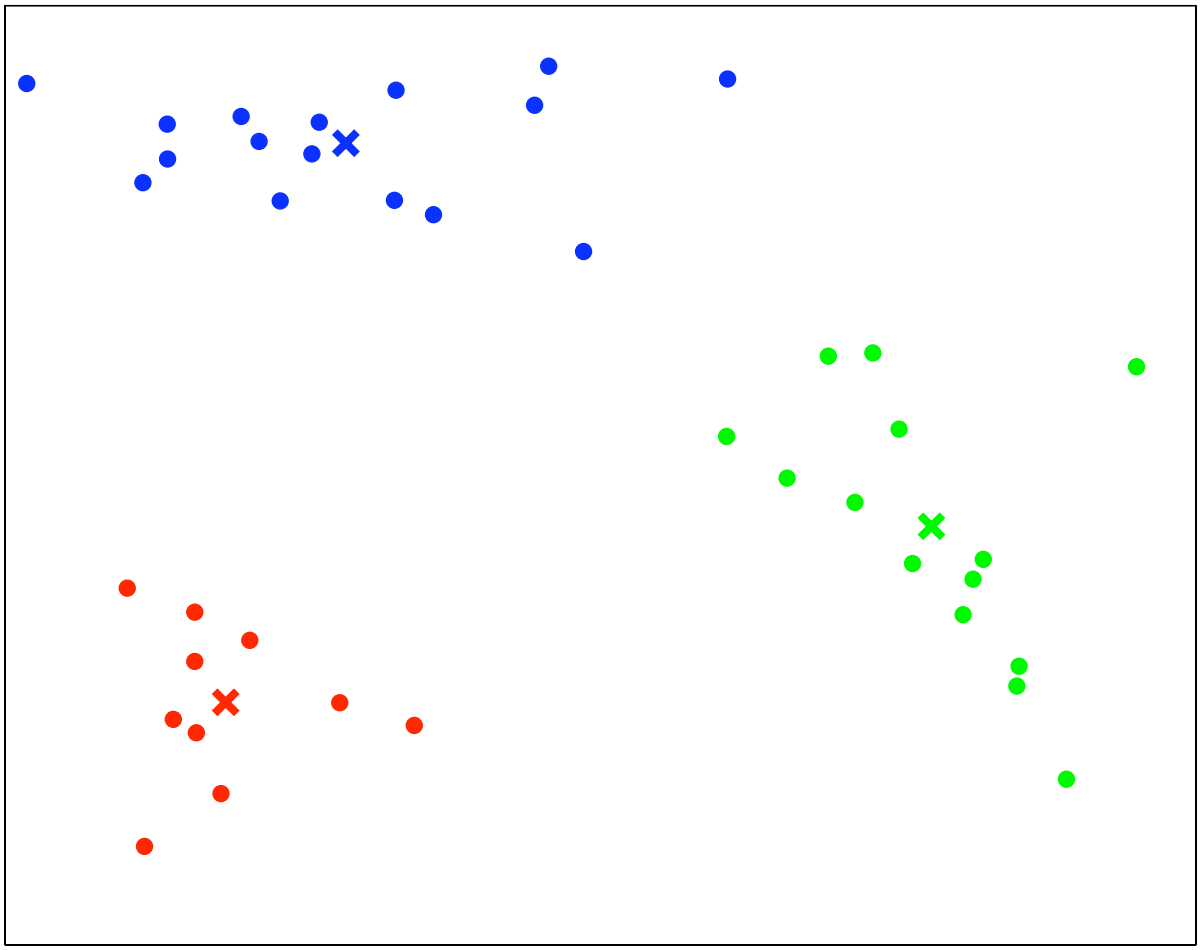
Example: recompute centroids



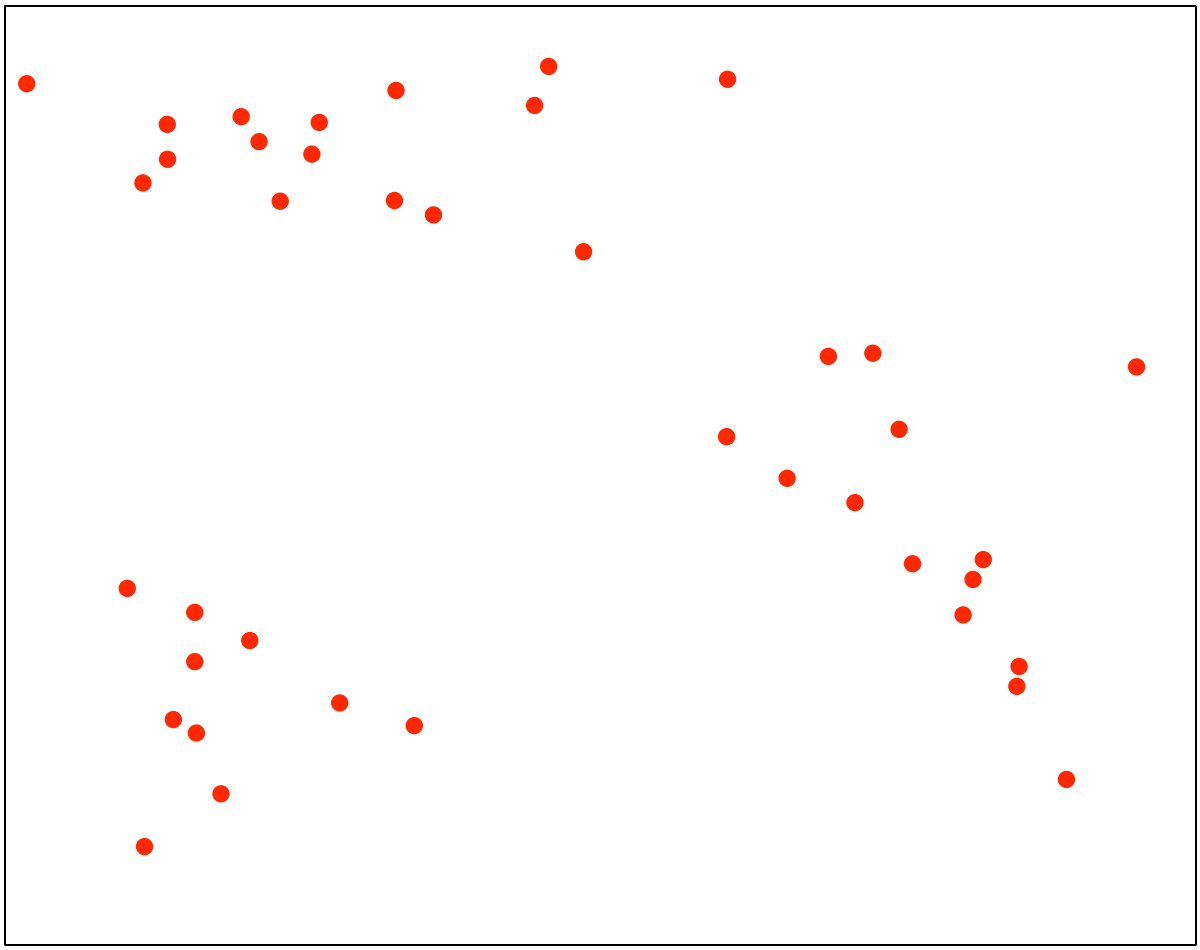
Example: reassign clusters



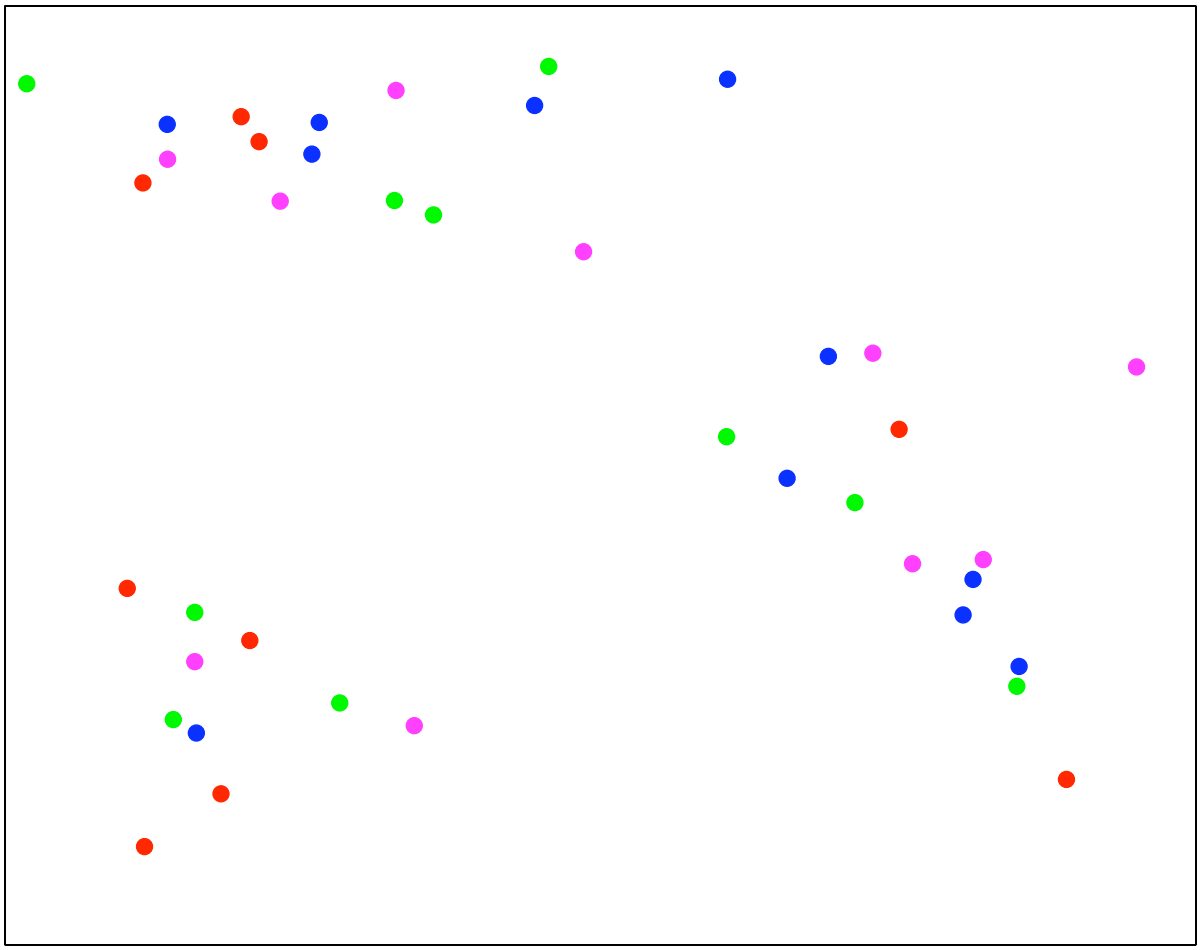
Example: recompute centroids – done!



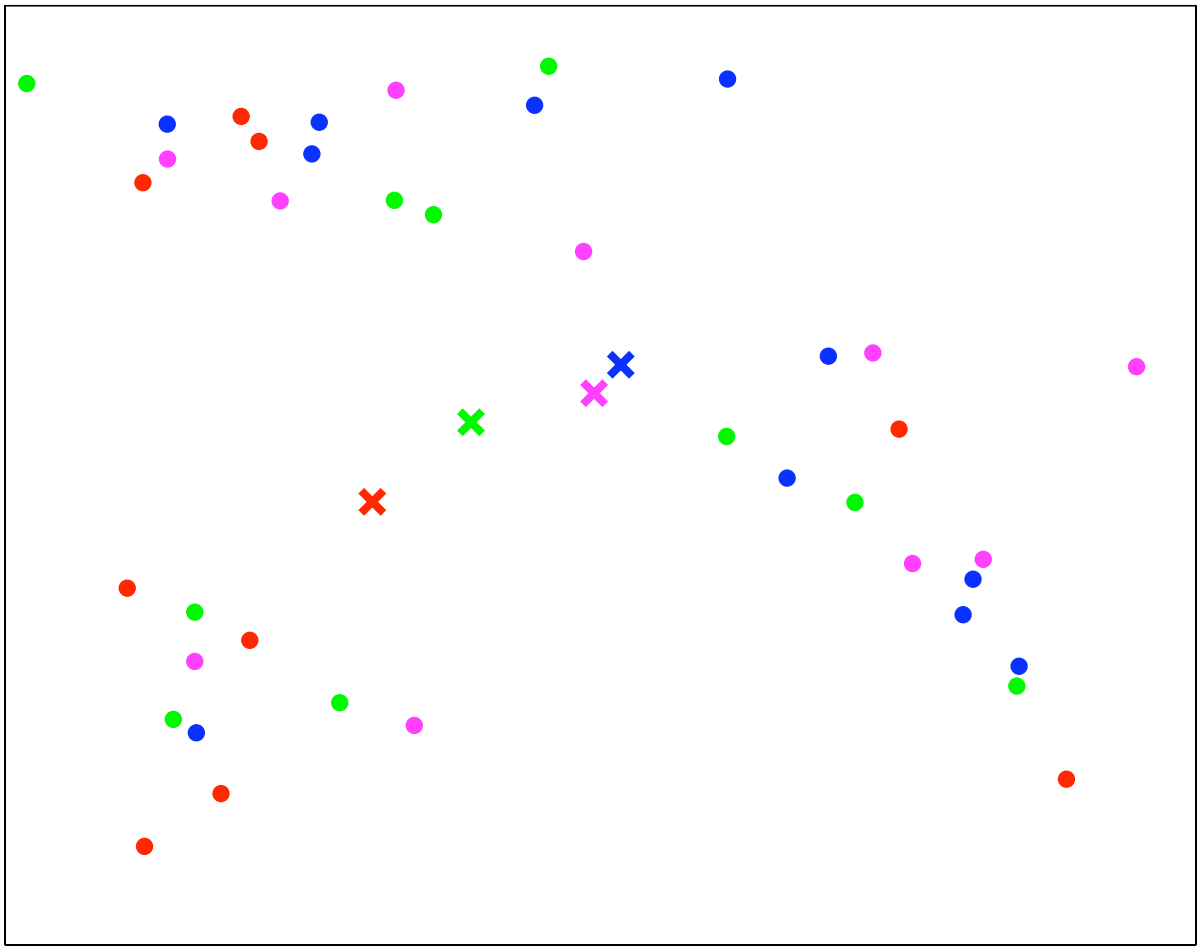
What is the right number of clusters?



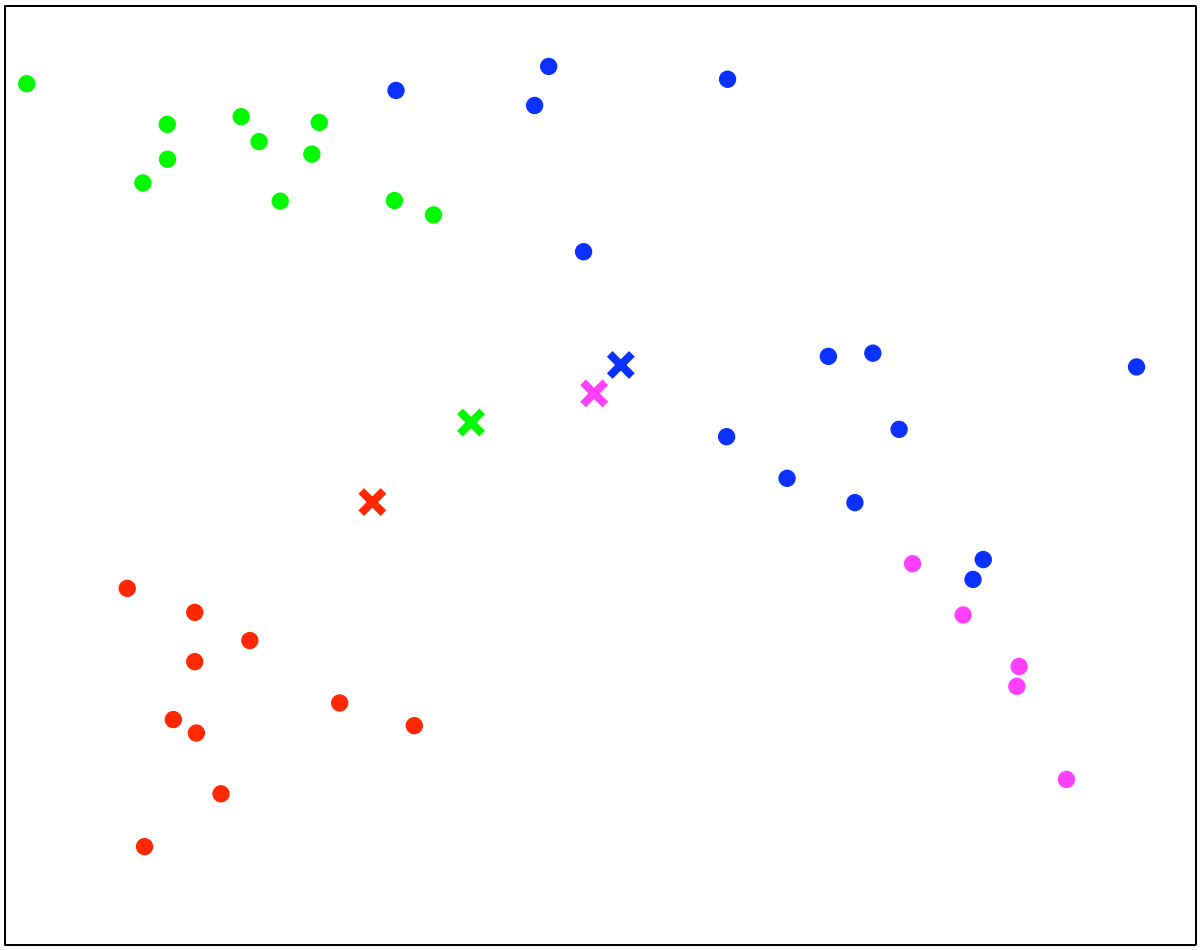
Example: assign into 4 clusters randomly



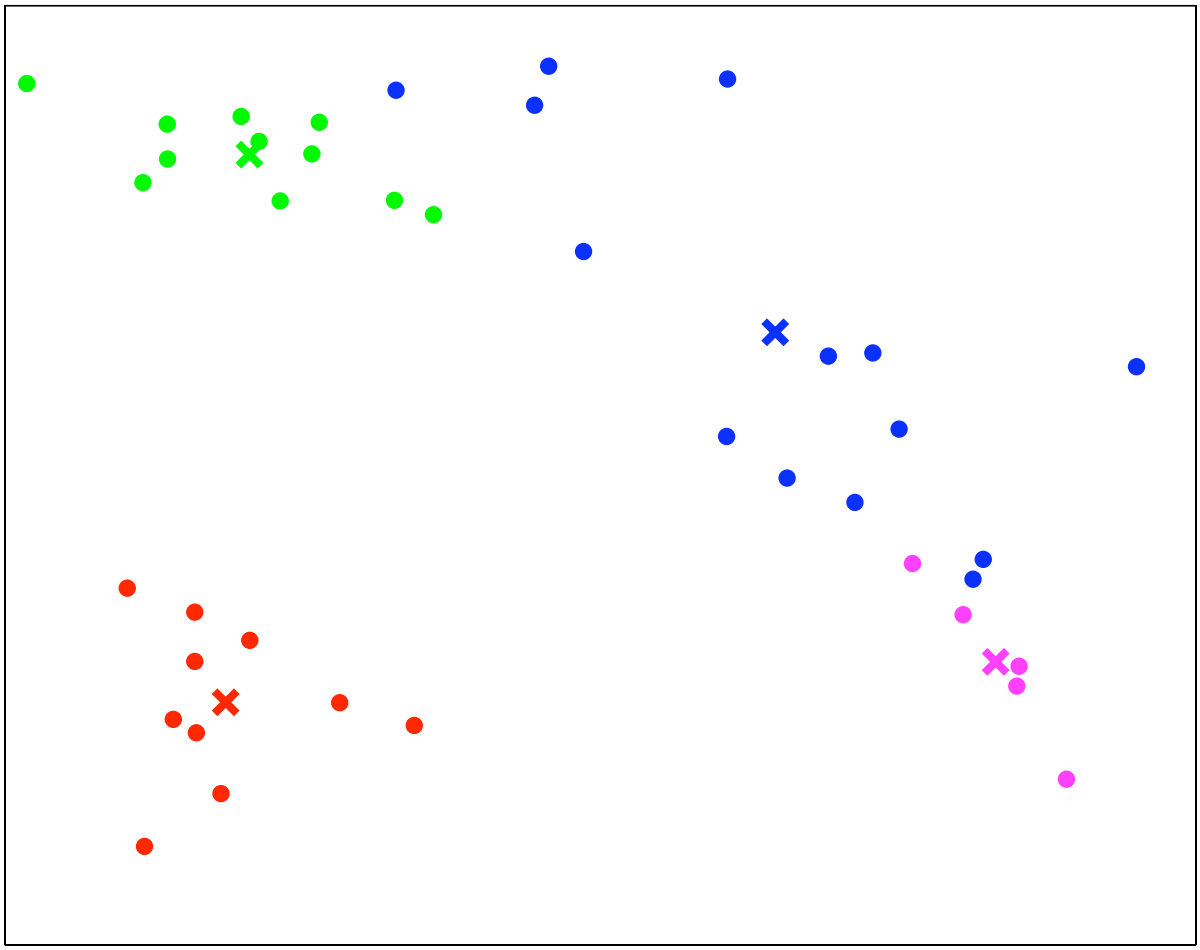
Example: compute centroids



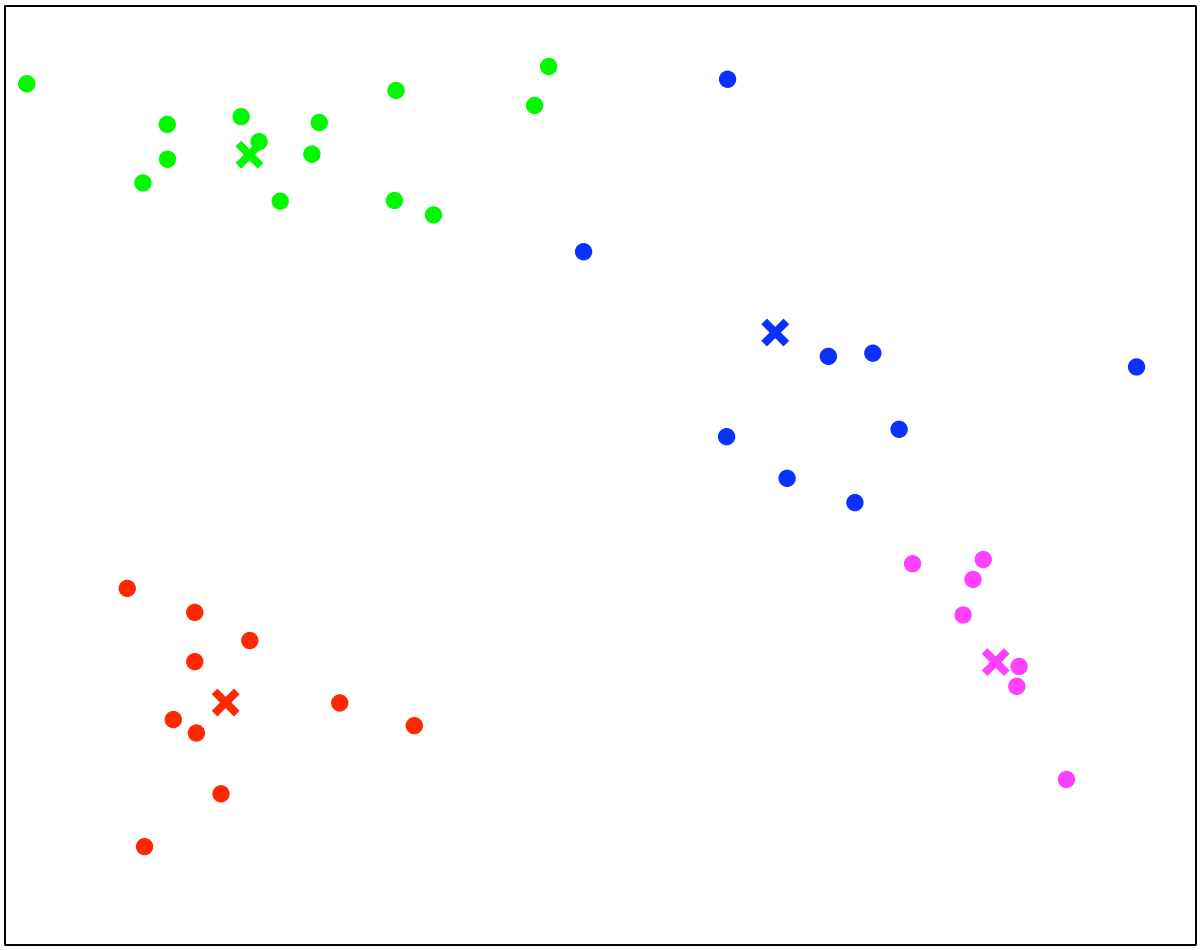
Example: reassign clusters



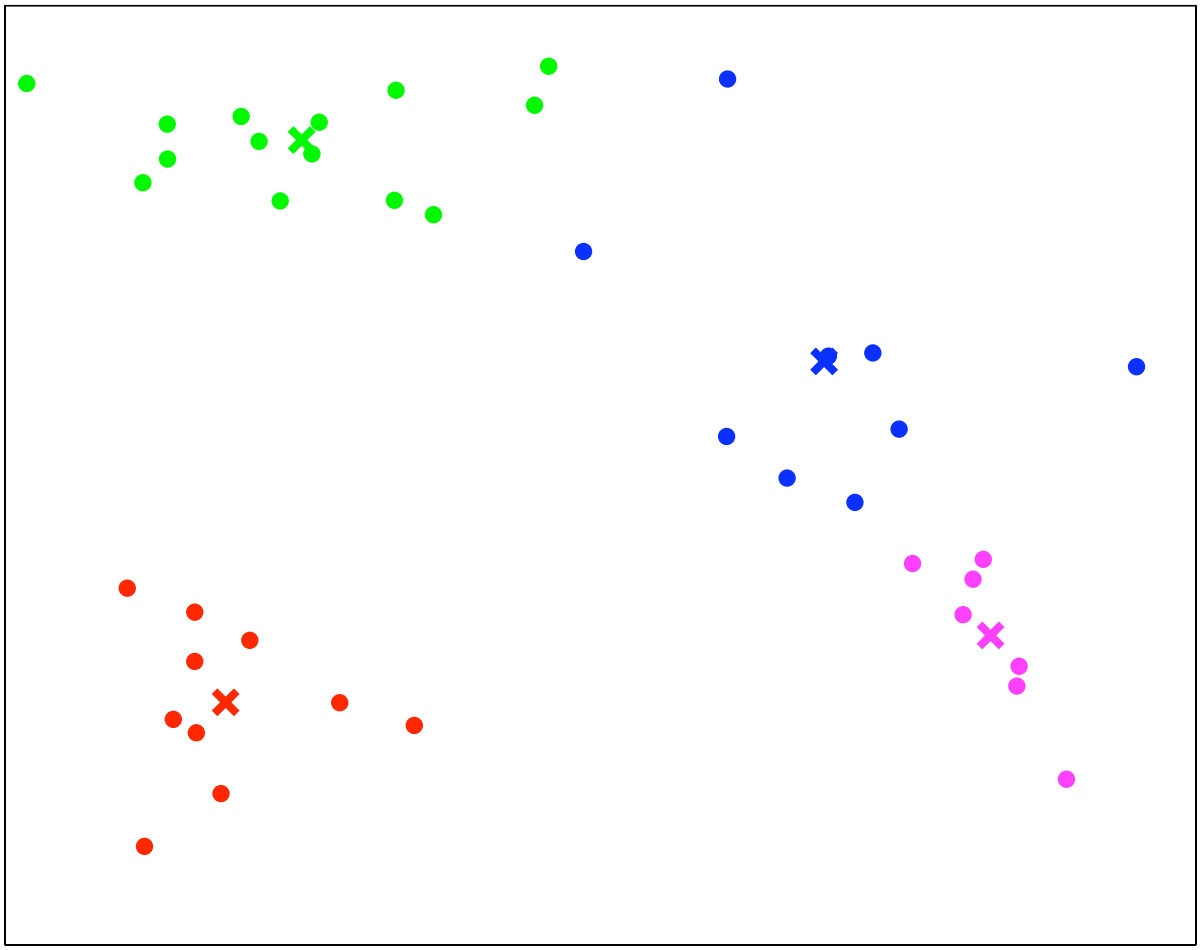
Example: recompute centroids



Example: reassign clusters



Example: recompute centroids – done!



Assessing the quality of the clustering

- If used as a pre-processing step for supervised learning, measure the performance of the supervised learner
- Measure the "tightness" of the clusters: points in the same cluster should be close together, points in different clusters should be far apart
- Tightness can be measured by the minimum distance, maximum distance or average distance between points
- **Silhouette criterion** is sometimes used
- Problem: these measures usually favour large numbers of clusters, so some form of complexity penalty is necessary

Typical applications of clustering

- Pre-processing step for supervised learning
- Data inspection/experimental data analysis

- Discretizing real-valued variables in non-uniform buckets.
- Data compression

Questions

- Will K-means terminate?
- Will it always find the same answer?
- How should we choose the initial cluster centers?
- Can we automatically choose the number of centers?

Does K-means clustering terminate?

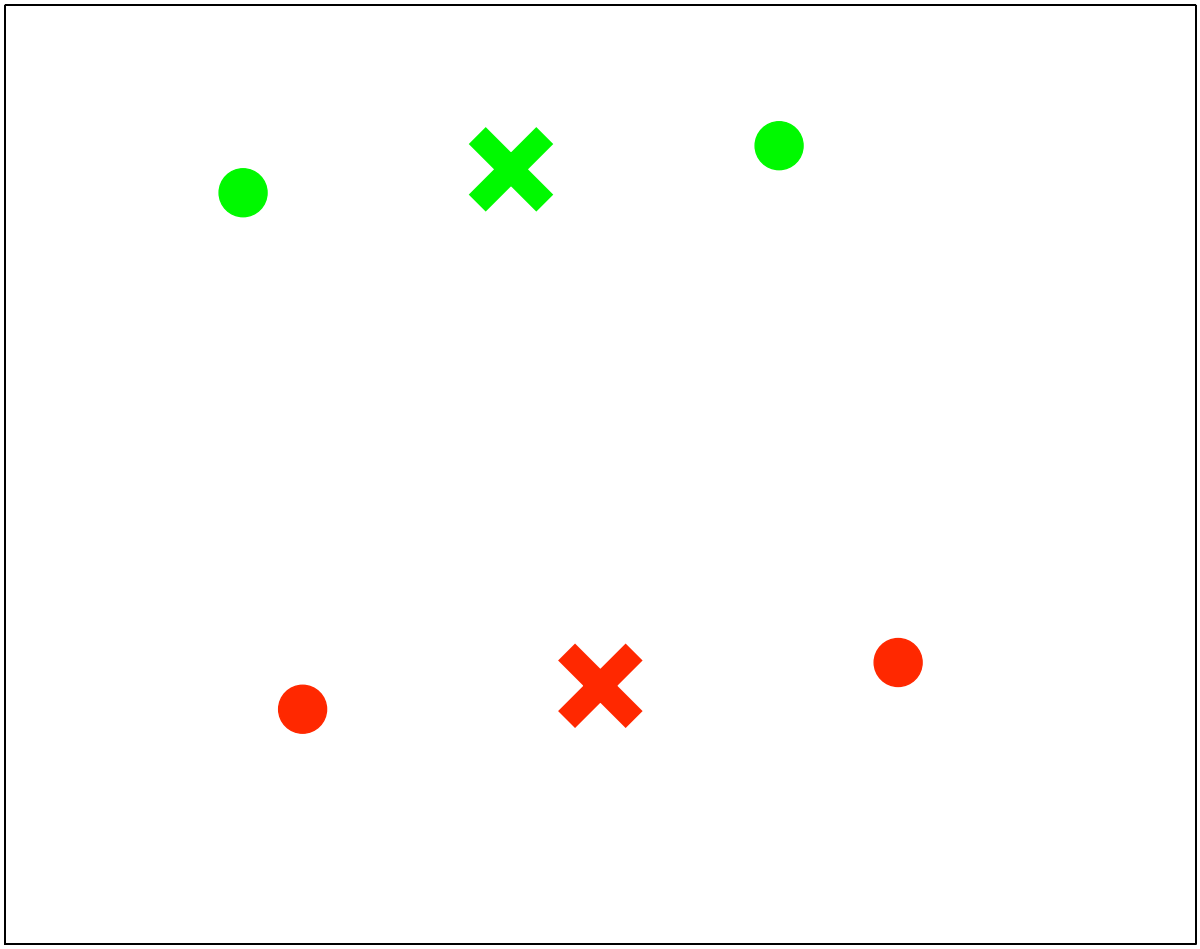
- Yes
- See extra slides for proof sketch

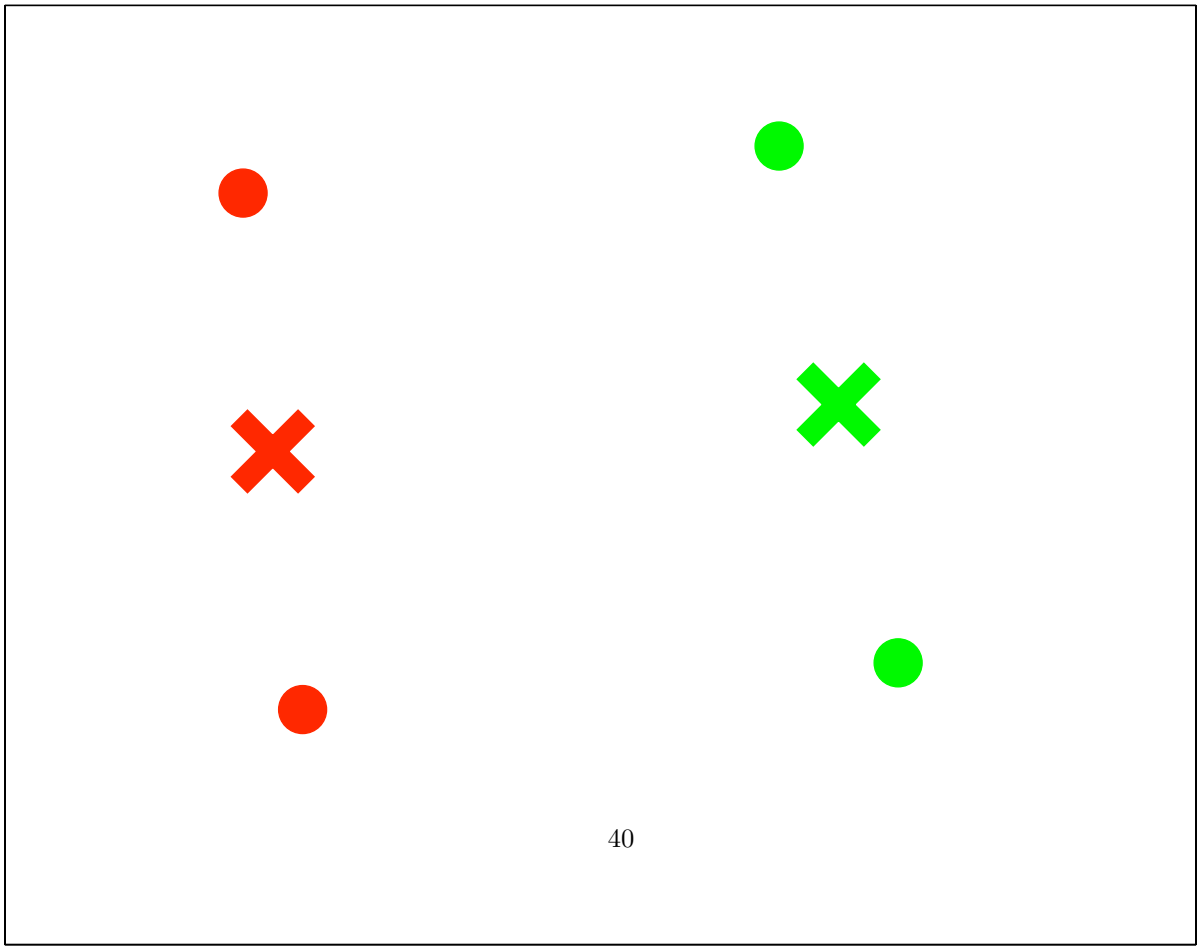
Does K-means always find the same answer?

- K-means is a version of coordinate descent, where the parameters are the cluster center coordinates, and the assignments of points to clusters.
- It minimizes the sum of squared Euclidean distances from vectors to their cluster centroid.
- This error function has many local minima!
- The solution found is *locally optimal*, but *not globally optimal*
- Because the solution depends on the initial assignment of instances to clusters, random restarts will give different solutions

Example - Same problem, different solutions

$J = 0.22870$ $J = 0.3088$



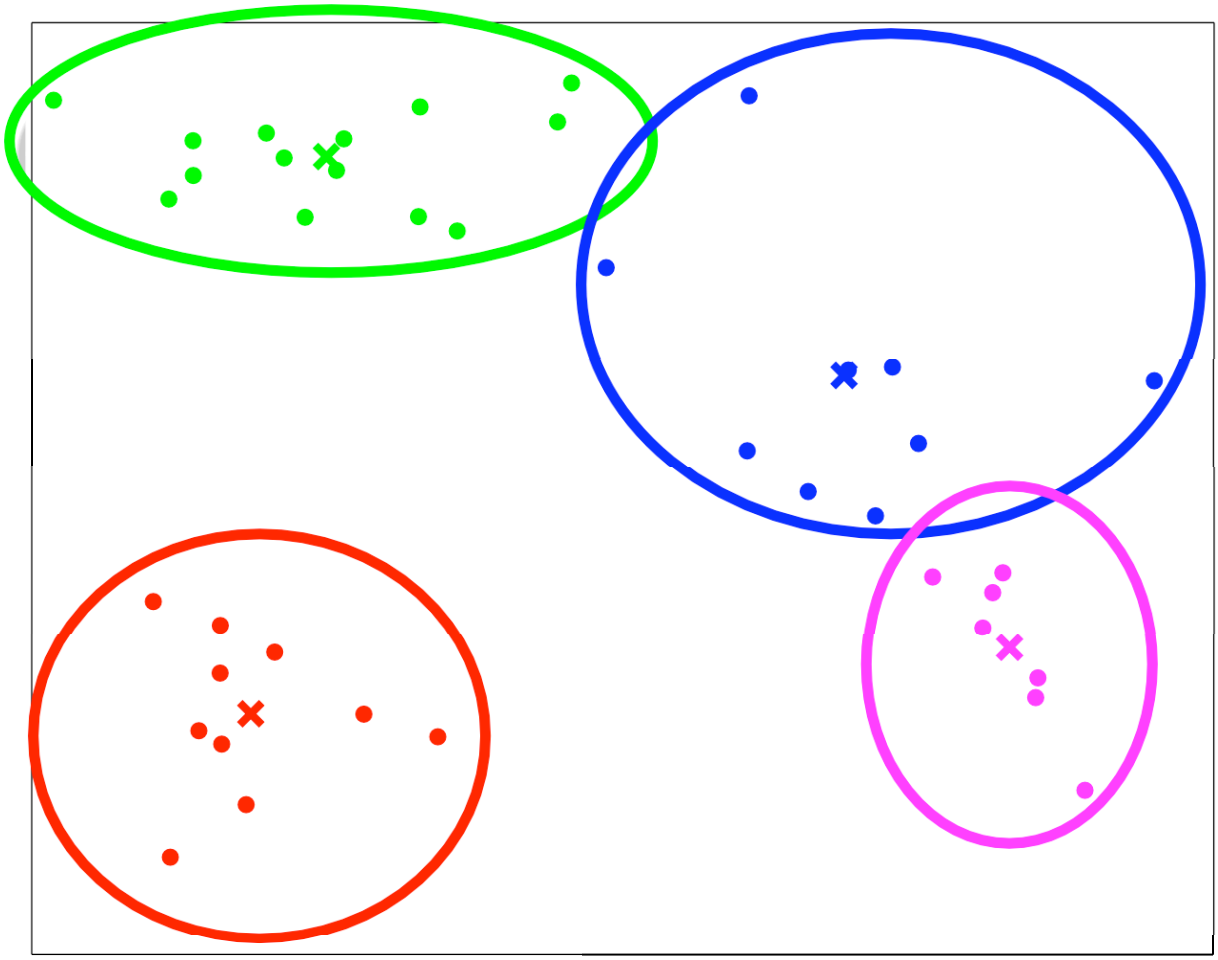


Choosing the number of clusters

- A difficult problem.
- Delete clusters that cover too few points
- Split clusters that cover too many points
- Add extra clusters for "outliers"
- Add option to belong to "no cluster"
- Minimum description length: minimize loss + complexity of the clustering
- Use a hierarchical method first

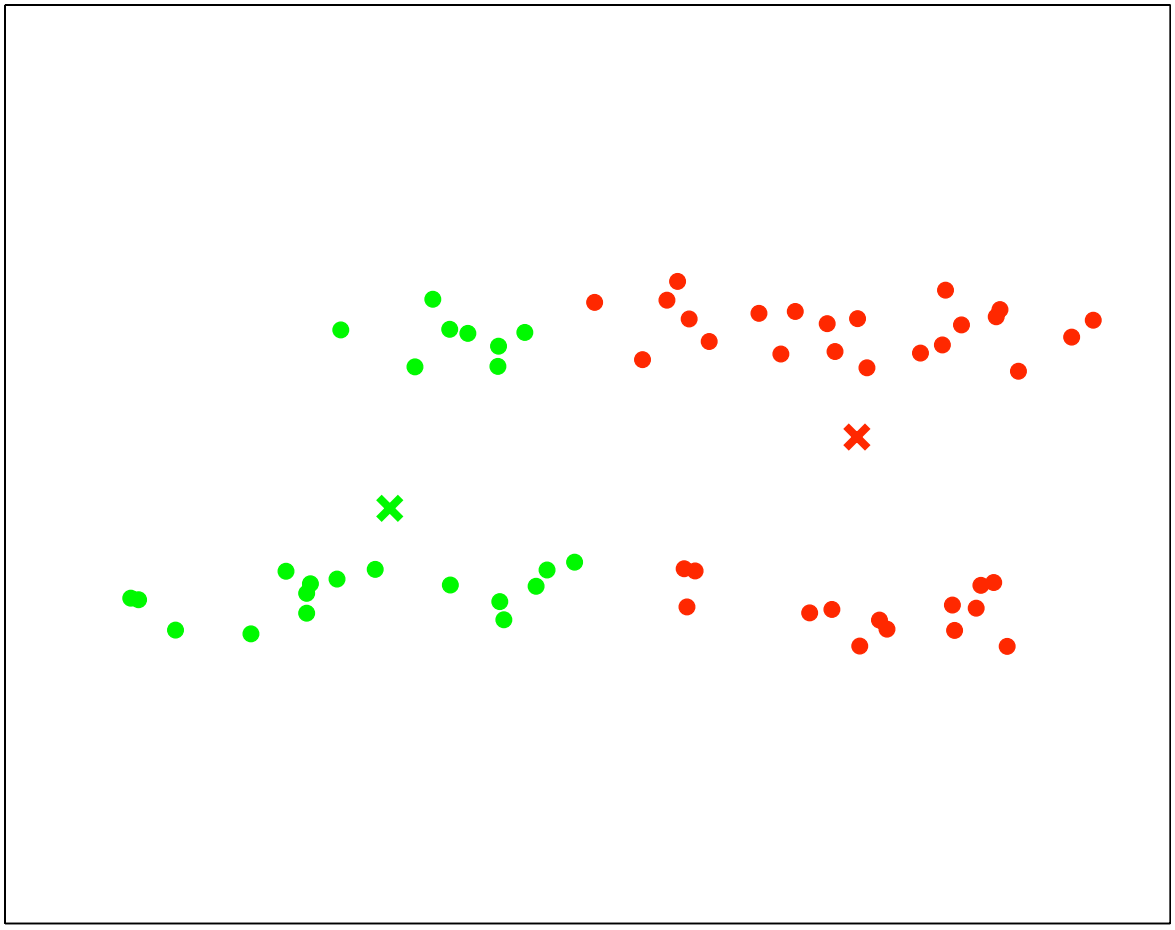
Why Euclidean distance?

Subjective reason: It produces nice, round clusters.



Why *not* Euclidean distance?

1. It produces nice round clusters!



2. Differently scaled axes can dramatically affect results.
3. There may be symbolic attributes, which have to be treated differently

Agglomerative clustering

- Input: Pairwise distances $d(\mathbf{x}, \mathbf{x}')$ between a set of data objects $\{\mathbf{x}_i\}$.
- Output: A hierarchical clustering
  1. Assign each instance as its own cluster on a working list W.
  2. Repeat
     1. Find the two clusters in W that are most "similar".
     2. Remove them from W.
     3. Add their union to W.

     until W contains a single cluster with all the data objects.
  3. Return *all clusters* appearing in W at any stage of the algorithm.

How many clusters after iteration k?

- Answer: $n - k$, where n is the number of data objects.

- Why?
  - The working list W starts with n singleton clusters
  - Each iteration removes two clusters from W and adds one new cluster
  - The algorithm stops when W has one cluster, which is after $k = n - 1$ iterations

How do we measure dissimilarity between clusters?

- Distance between nearest objects ("Single-linkage" agglomerative clustering, or"nearest neighbor"):

$$\min_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

- Average distance between objects ("Group-average" agglomerative clustering):

$$\frac{1}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

- Distance between farthest objects ("Complete-linkage" agglomerative clustering, or"furthest neighbor"):

$$\max_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

- (If reading PDF, note that following examples are available on slides only.)

Example 1: Data

Single	Average	Complete

Example 1: Iteration 30

Single	Average	Complete

Example 1: Iteration 60

Single	Average	Complete

Example 1: Iteration 70

Single	Average	Complete

Example 1: Iteration 78

Single	Average	Complete

Example 1: Iteration 79

Single	Average	Complete

Example 2: Data

Single	Average	Complete

Example 2: Iteration 50

Single	Average	Complete

Example 2: Iteration 80

Single	Average	Complete

Example 2: Iteration 90

Single	Average	Complete

Example 2: Iteration 95

Single	Average	Complete

Example 2: Iteration 99

Single	Average	Complete

Intuitions about cluster similarity

- Single-linkage
  - Favors spatially-extended / filamentous clusters
  - Often leaves singleton clusters until near the end
- Complete-linkage favors compact clusters
- Average-linkage is somewhere in between

Summary of K-means clustering

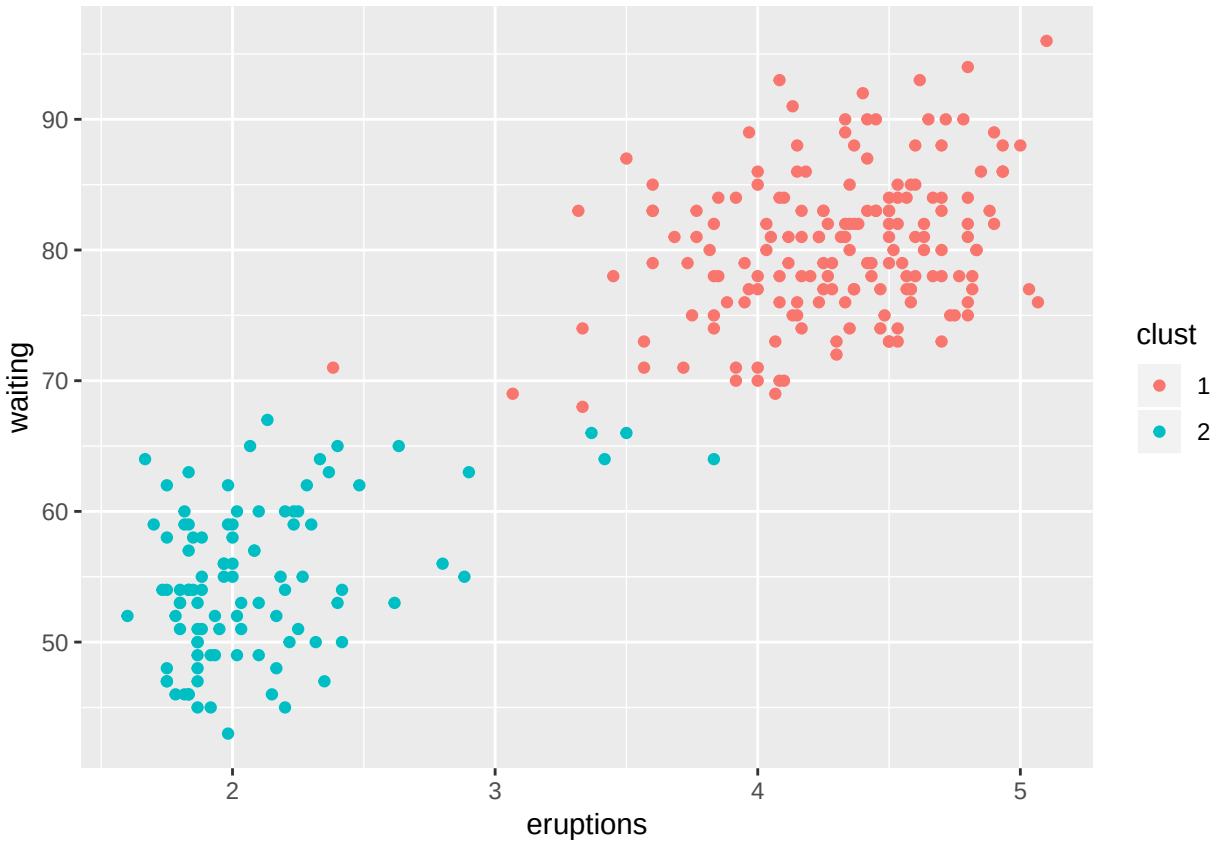
- Fast way of partitioning data into K clusters
- It minimizes the sum of squared Euclidean distances to the clusters centroids
- Different clusterings can result from different initializations
- Natural way to add new points to existing clusters.

Summary of Hierarchical clustering

- Organizes data objects into a tree based on similarity.
- Agglomerative (bottom-up) tree construction is most popular.
- There are several choices of distance metric (linkage criterion)
- No natural way to find which cluster a new point should belong to

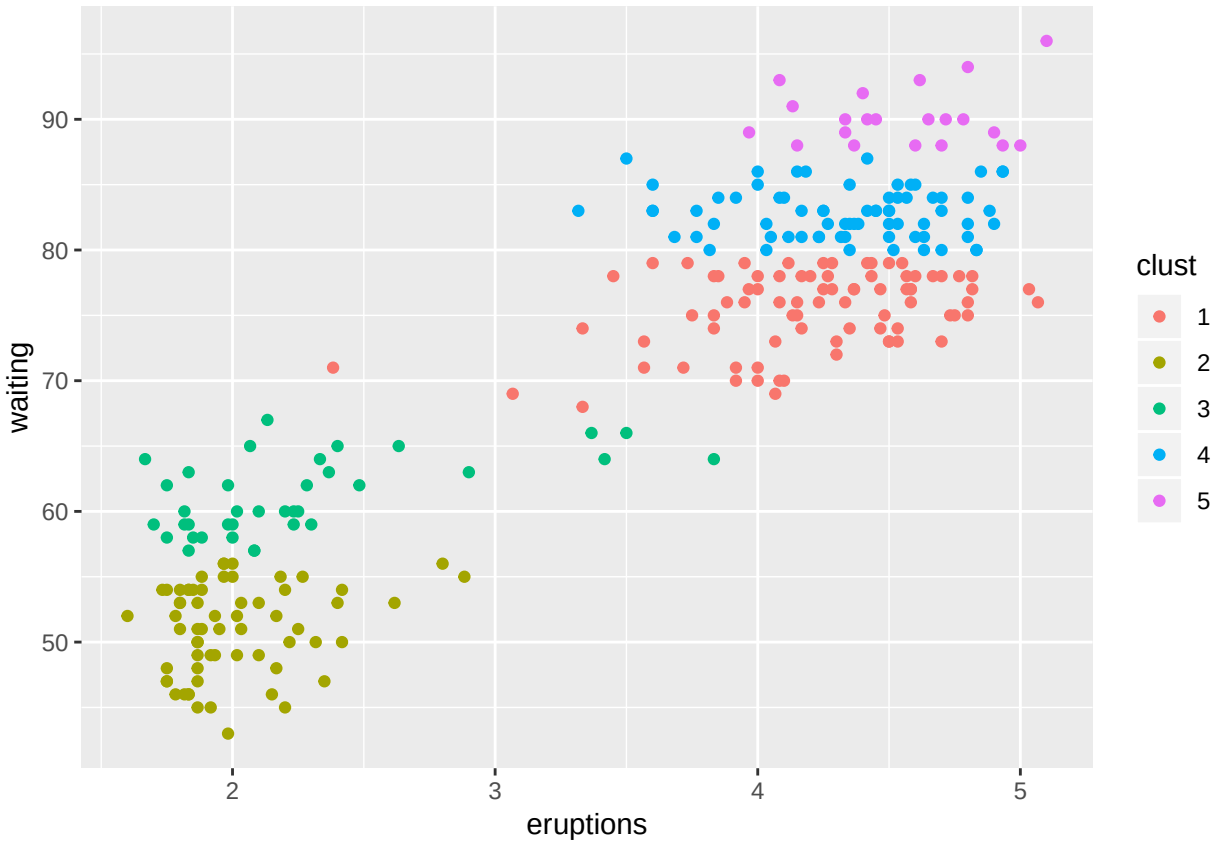
Clustering Example

```
fclust <- hclust(dist(faithful), "ave");
faithful$clust <- as.factor(cutree(fclust,2))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



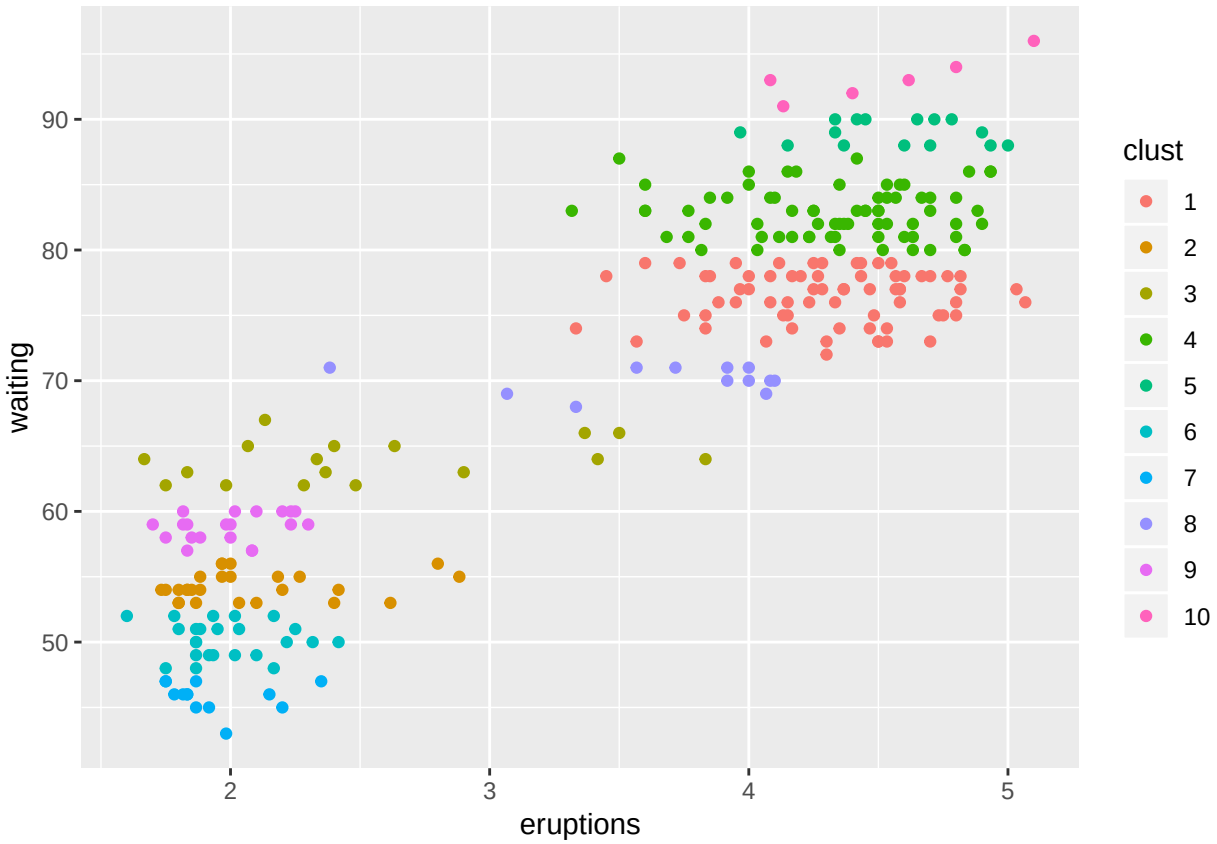
Clustering Example

```
fclust <- hclust(dist(faithful), "ave");
faithful$clust <- as.factor(cutree(fclust,5))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



Clustering Example

```
fclust <- hclust(dist(faithful), "ave");
faithful$clust <- as.factor(cutree(fclust,10))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



Scale matters!

```
sfaithful <- scale(faithful[,c(1,2)])
head(sfaithful,4)
```

```
##     eruptions    waiting
## 1  0.09831763  0.5960248
## 2 -1.47873278 -1.2428901
## 3 -0.13561152  0.2282418
## 4 -1.05555759 -0.6544374
```

```
attr(sfaithful,"scaled:center")
```

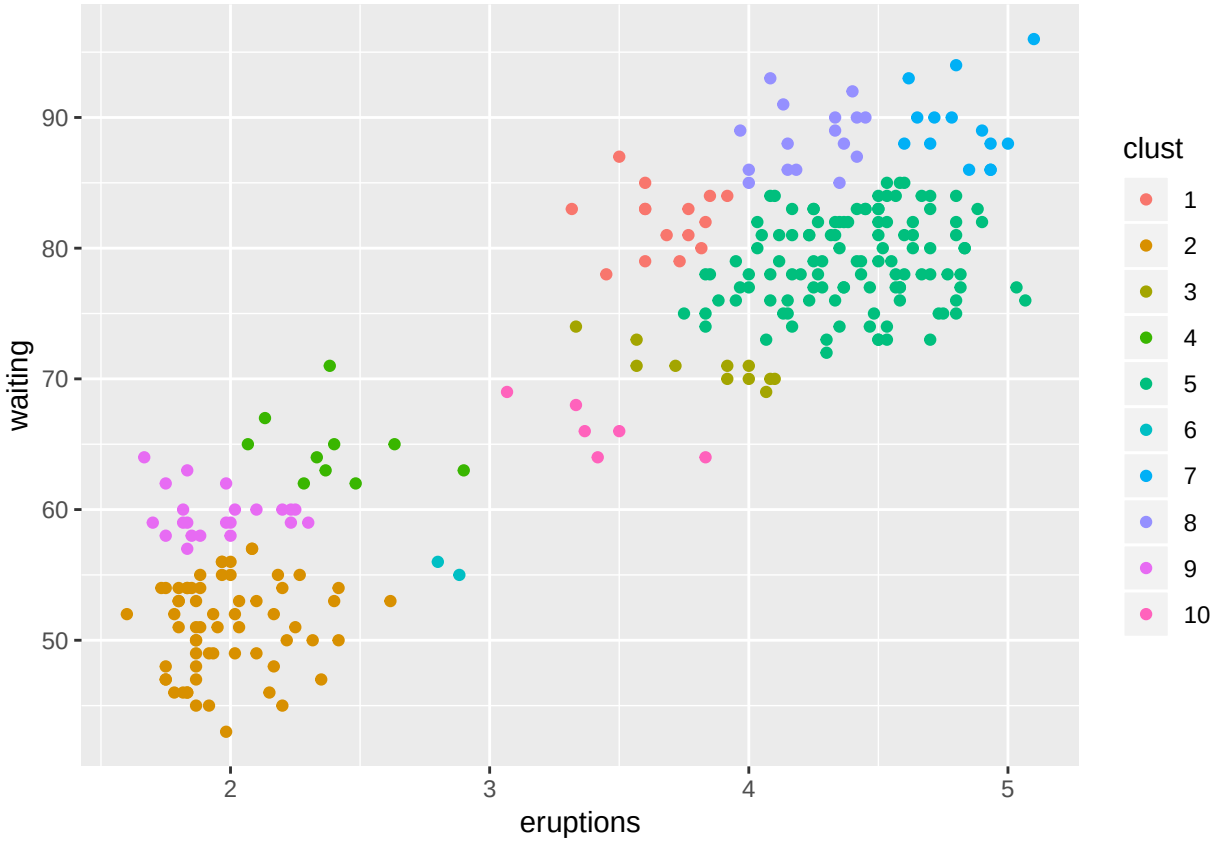
```
## eruptions   waiting
##  3.487783 70.897059
```

```
attr(sfaithful,"scaled:scale")
```

```
## eruptions   waiting
##  1.141371 13.594974
```

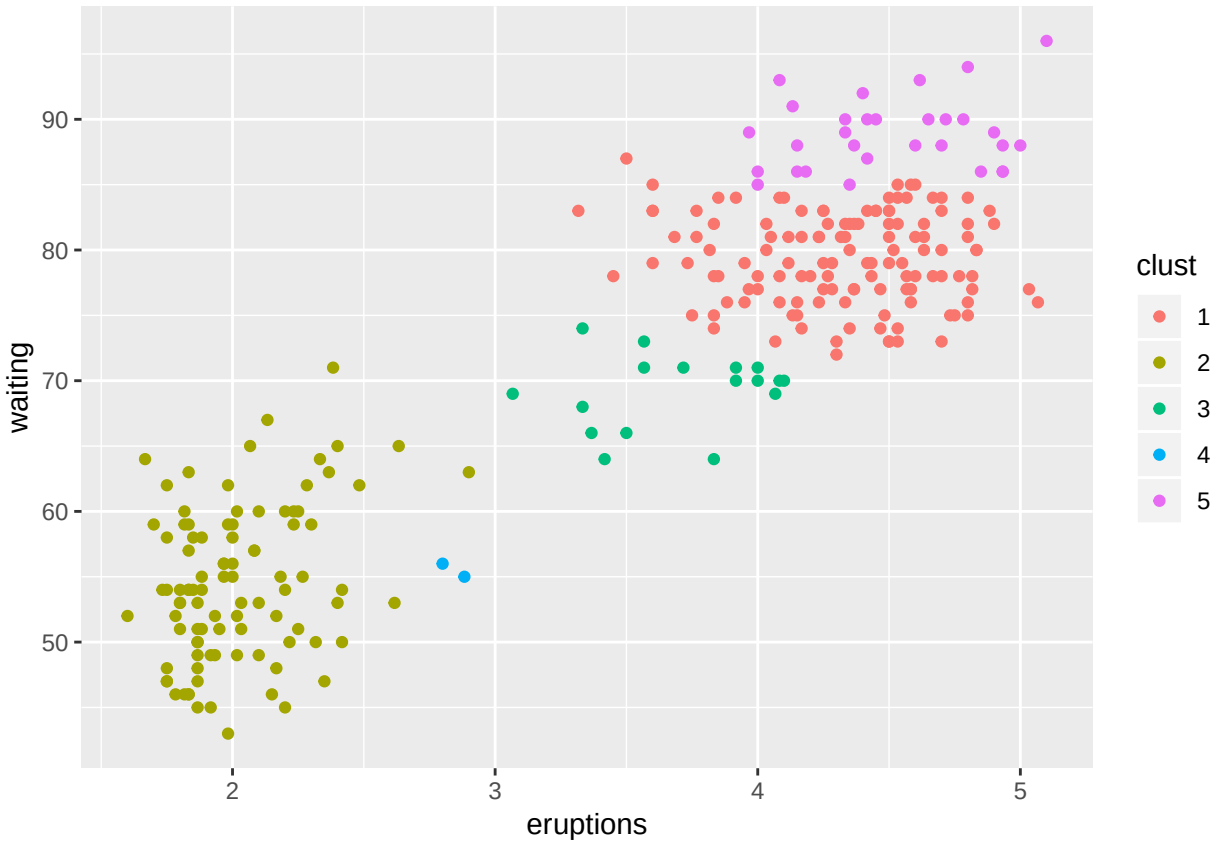
Hierarchical Clustering

```
fclust <- hclust(dist(sfaithful), "ave");
faithful$clust <- as.factor(cutree(fclust,10))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



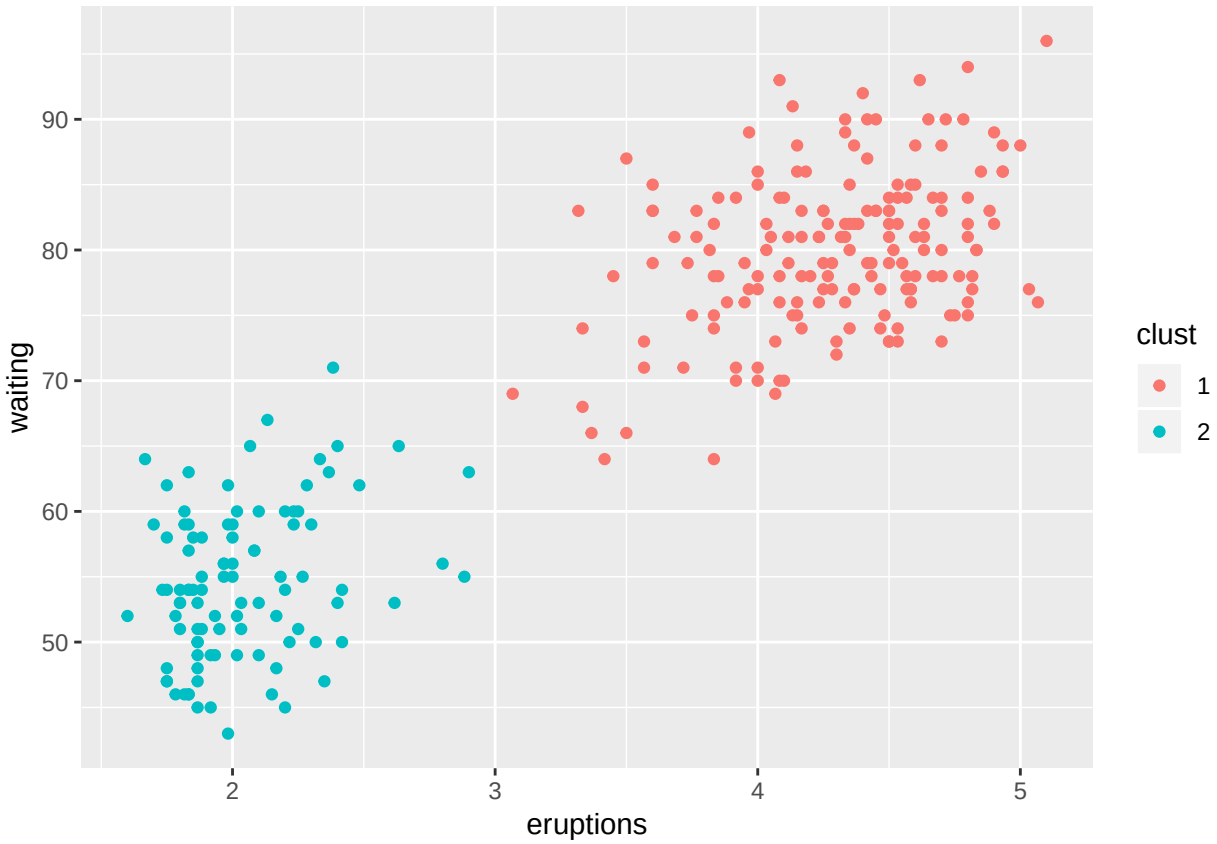
Hierarchical Clustering

```
fclust <- hclust(dist(sfaithful), "ave");
faithful$clust <- as.factor(cutree(fclust,5))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



Hierarchical Clustering

```
fclust <- hclust(dist(sfaithful), "ave");
faithful$clust <- as.factor(cutree(fclust,2))
ggplot(faithful,aes(x=eruptions,y=waiting,colour=clust)) + geom_point()
```



Extra Slides

Does K-means clustering terminate?

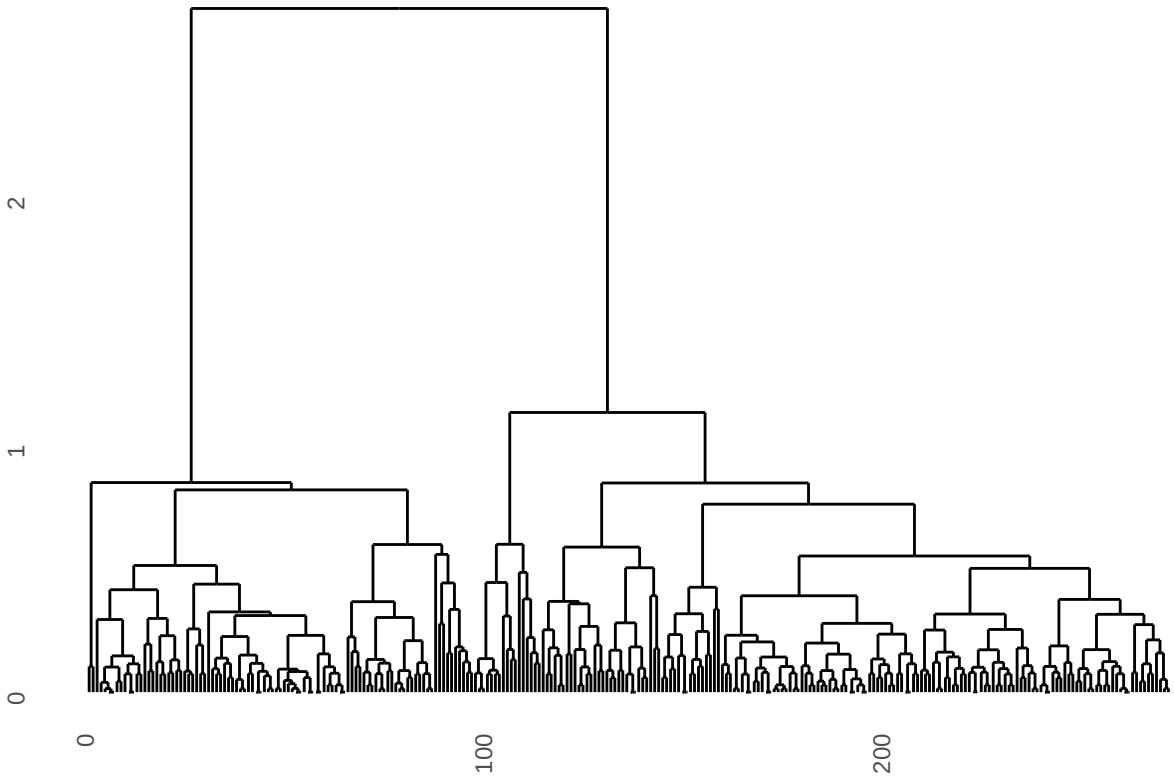
- For given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and a clustering C, consider the sum of the squared Euclidean distance between each vector and the center of its cluster:

$$J = \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{C(i)}\|^2 ,$$

  where $\mu_{C(i)}$ denotes the centroid of the cluster containing $\mathbf{x}_i$.
- There are finitely many possible clusterings: at most K^n.
- Each time we reassign a vector to a cluster with a nearer centroid, J decreases.
- Each time we recompute the centroids of each cluster, J decreases (or stays the same.)
- Thus, the algorithm must terminate.

Dendrogram

```
library(ggdendro)
ggdendrogram(fclust, leaf_labels=F, labels=F)
```



Images: Cat or dog?

1. Identify the prediction you want to make.
2. Identify the information you need to make each prediction.
3. Summarize that information into a feature vector.
4. Pass the result to a supervised learner

Image features

3. Summarize that information into a feature vector.

- Do I need to summarize? Can I just use pixels?
  - The average lolcat has 250,000 pixels
  - Pixels are affected by many non-cat-related issues, including:
    - Color of cat
    - Distance to cat
    - Illumination
    - Background
- Expecting to learn that the important difference is the cat-dog difference rather than some other accidental difference is unrealistic.

What is an image feature?

- A function that given an image produces some (relatively) low-dimensional output but retains some (relatively) interesting information
- "Global" image features:
  - Mean (median, mode) pixel intensity - very low-dimensional, super boring, probably not useful
  - RGB histogram(s) - $(2^8 \cdot 3)$-dimensional vector, no spatial information, might help find some objects (e.g. Canadian flag vs. American flag?)
  - Image "gradient" - 2D vector pointing at direction of increasing brightness
  - …

What is an image feature?

- "Local" image features:
  - Global features applied to little patches of the big image
  - ***Dense*** if we pre-determine the patches (say using a grid)
  - ***Sparse*** if we decide which ones to compute based on the image itself

Dense vs. Sparse

- Dense seems good – fixed-length feature vector!
- What if the important information is between grid cells?
- Too fine a grid is impractical.
- Famous sparse local image features: The Scale-Invariant Feature Transform (SIFT),
  Lowe, David G. 'Distinctive Image Features from Scale Invariant Features', International Journal of Computer Vision, Vol. 60, No. 2, 2004, pp. 91-110

Identifying Interesting Points ("keypoints")

- A point is interesting if it is much darker or brighter than its "neighbors" according to difference-of-Gaussians filter.

Pictures courtesy SadaraX at en.wikipedia

The Scale of an Interesting Point

- "Interestingness" will depend on both the (x,y) location in the image, and on the chosen scale σ of the filter.
- The "scale" of a point is the σ that makes it most interesting.

The Orientation of an Interesting Point

- Based on nearby image gradients
- We can now identify interesting points, assign a scale and an orientation.
- Goal: ability to "match up" interesting points between two different images

A "signature" for an interesting point ("descriptor")

- A 128-dim vector summarizing the image gradients at nearby pixels, relative to interesting point orientation and scale
- **Hope:** in a different image, the "same" interesting point will have the same descriptor, even with different lighting, orientation and scale.

SIFT: Scale-Invariant Feature Transform

- Produces a set of "interesting" points, with the following:
  - A 4-element "frame": (x, y) location, scale, orientation
  - 128-element descriptor
- Are these "features?"
- Goal of sifts: find "the same" points in different pictures. (They will have similar descriptors, possibly different frames.)

SIFT Example

The Goal of SIFTS

- Find **specific** objects in a database of images
- **NOT** to use in machine-learning methods.
- Note: does not produce a fixed-length feature vector given an image
- **But** they have properties we want: invariance to position, scale, rotation, (some) 3D pose, fair bit of lighting
- About 1000-ish features are produced for an average-sized image
- ***SIFTs are to images as words are to documents.***

Vector quantization

- Construct a *dictionary* of vectors labeled $\{1, 2, ..., K\}$
- Given any vector, we can *encode* it by finding the closest (in whatever distance metric) vector in the dictionary

Vector quantization

Feature vectors from VQ

- We can encode any set of vectors into a histogram
- **The histograms (counts) output by VQ are fixed-length feature vectors.**
- Much like bag-of-words. "How many times does a feature close to vector k appear in my image?"
- Possibly as sparse as bag-of-words.

SIFTs as "bag-of-words"

- Hope: If we apply VQ to sifts, images with "similar" shapes/objects in them will have "similar" feature vectors (histograms)
- Trick: Put some spatial information back in:
  http://www.di.ens.fr/willow/pdfs/cvpr06b.pdf
- We can now use standard ML methods (including possibly feature selection) for classification of images
- Similar ideas could apply in other domains where the "raw data" are not in a format conducive to standard methods.
- **But how do we make the dictionary for VQ?**

PCA Math

Solving the PCA optimization problem [HTF Ch. 14.5]

$$\begin{array}{lll} \min & \sum_{i=1}^{n} \|\mathbf{x}_i - (\mathbf{b} + \alpha_i \mathbf{v})\|^2 & \text{w.r.t. } \mathbf{b}, \mathbf{v}, \alpha_i, i = 1, \ldots n \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 & \end{array}$$

- Turns out the optimal $\mathbf{b}$ is just the sample mean of the data, $\mathbf{b} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$
- This means that the best line goes through the mean of the data. Typically, we subtract the mean first. Assuming it's zero:

$$\begin{array}{lll} \min & \sum_{i=1}^{n} \|\mathbf{x}_i - \alpha_i \mathbf{v}\|^2 & \text{w.r.t. } \mathbf{v}, \alpha_i, i = 1, \ldots n \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 & \end{array}$$

- Consider fixing $\mathbf{v}$. The optimal α_i is given by *projecting* $\mathbf{x}_i$ onto $\mathbf{v}$.

Optimal choice of v

$$\begin{array}{lll} \max & \mathbf{v}^\mathsf{T}(X^\mathsf{T}X)\mathbf{v} & \text{w.r.t. } \mathbf{v} \\ \text{s.t.} & \|\mathbf{v}\|^2 = 1 & \end{array}$$

- Forming the Lagrangian of the above problem and setting derivative to zero gives $(X^\mathsf{T}X)\mathbf{v} = \lambda\mathbf{v}$ as feasible solutions.
- Recall: an *eigenvector* $\mathbf{u}$ of a matrix A satisfies $A\mathbf{u} = \lambda\mathbf{u}$, where $\lambda \in \mathbb{R}$ is the *eigenvalue*.
- Fact: The matrix $X^\mathsf{T}X$ has p non-negative eigenvalues and p orthogonal eigenvectors.
- Thus, $\mathbf{v}$ must be an eigenvector of $(X^\mathsf{T}X)$.
- The $\mathbf{v}$ that maximizes $\mathbf{v}^\mathsf{T}(X^\mathsf{T}X)\mathbf{v}$ is the eigenvector of $(X^\mathsf{T}X)$ with the largest eigenvalue

Another view of v

max	$\mathbf{v}^\mathsf{T}(X^\mathsf{T}X)\mathbf{v}$	w.r.t. $\mathbf{v}$
s.t.	$\|\mathbf{v}\|^2 = 1$	

- Recall $\mathbf{x}_i^\mathsf{T}\mathbf{v} = \alpha_i$ is our low-dimensional representation of $\mathbf{x}_i$
- $\mathbf{v}^\mathsf{T}(X^\mathsf{T}X)\mathbf{v} = \sum_i(\mathbf{x}_i^\mathsf{T}\mathbf{v})^2 = \mathrm{Var}(\mathbf{x}_i^\mathsf{T}\mathbf{v})$
- The optimal $\mathbf{v}$ produces an encoding that has as much variance as possible

Optimizing for first principal component...

Let's look at the objective we want to minimize:

- $\sum_{i=1}^n \|\mathbf{x}_i - \alpha_i\mathbf{v}\|^2$, min over $\mathbf{v}, \alpha_i$ s.t. $\|\mathbf{v}\| = 1$
- $\sum_{i=1}^n(\mathbf{x}_i - \alpha_i\mathbf{v})^\mathsf{T}(\mathbf{x}_i - \alpha_i\mathbf{v})$
- $\sum_{i=1}^n \mathbf{x}_i^\mathsf{T}\mathbf{x}_i - 2\alpha_i\mathbf{v}^\mathsf{T}\mathbf{x}_i + \alpha_i^2$ (Assumed $\mathbf{v}$ was a unit vector.)
- $\implies \alpha_i^* = \mathbf{v}^\mathsf{T}\mathbf{x}_i$
- $\sum_{i=1}^n \mathbf{x}_i^\mathsf{T}\mathbf{x}_i - 2\mathbf{v}^\mathsf{T}\mathbf{x}_i\mathbf{v}^\mathsf{T}\mathbf{x}_i + \mathbf{v}^\mathsf{T}\mathbf{x}_i\mathbf{v}^\mathsf{T}\mathbf{x}_i$
- $\sum_{i=1}^n \mathbf{x}_i^\mathsf{T}\mathbf{x}_i - \mathbf{v}^\mathsf{T}\mathbf{x}_i\mathbf{v}^\mathsf{T}\mathbf{x}_i$
- $\sum_{i=1}^n \mathbf{x}_i^\mathsf{T}\mathbf{x}_i - \sum_{i=1}^n \mathbf{v}^\mathsf{T}\mathbf{x}_i\mathbf{v}^\mathsf{T}\mathbf{x}_i$
- $\mathrm{tr}(X^\mathsf{T}X) - \mathbf{v}^\mathsf{T}(X^\mathsf{T}X)\mathbf{v}$

Relation to eigenvectors

- If we assume the $\mathbf{v}_j$ are of unit length and orthogonal, then the optimal choices are:
  - $\mathbf{b}$ is the mean of the data (as before)
  - The $\mathbf{v}_j$ are orthogonal eigenvectors of $X^\mathsf{T}X$ corresponding to its d largest eigenvalues.
  - Each instance is projected orthogonally on the hyperplane.

Eigenfaces

- L. Sirovich and M. Kirby (1987). "Low-dimensional procedure for the characterization of human faces". Journal of the Optical Society of America A 4 (3): 519-524.
- Adapted from Wikipedia: http://en.wikipedia.org/wiki/Eigenface

1. Prepare a training set of face images taken under the same lighting conditions, normalized to have the eyes and mouths aligned, resampled to a common pixel resolution. Each image is treated as one vector, by concatenating the rows of pixels
2. Subtract the mean vector.
3. Calculate the eigenvectors and eigenvalues of $(X^\mathsf{T}X)$. Each eigenvector has the same dimensionality (number of components) as the original images, and thus can itself be seen as an image. The eigenvectors are called eigenfaces. They are the directions in which the images differ from the mean image.
4. Choose the principal components. The eigenvectors (eigenfaces) with largest associated eigenvalue are kept.

Faces

Eigenfaces

Extra Example - Netflix Recommender

Application: Netflix Recommender

- Given: An enormous matrix $Y_{n\times p}$ containing the ratings by n users of p movies. Ratings are all $\in \{1, 2, 3, 4, 5\}$.
- The point is to reconstruct Y. "As well as possible."
- Recall, SVD gives you:
$$Y_{n\times p} = U_{n\times n}D_{n\times p}V^\mathsf{T}_{p\times p}$$
but requires a complete Y, which we don't have.
- First, let's rearrange the decomposition (assume $n > p$):
$$Y_{n\times p} = U_{n\times p}V^\mathsf{T}_{p\times p}$$
- Solving this is way too easy: $U = Y$, $V = I$.

SVD with Missing Data

$$Y_{n\times p} = U_{n\times p}V^\mathsf{T}_{p\times p}$$

- Solving this is way too easy: $U = Y$, $V = I$. We have $n \times p$ plus $p \times p$ parameters to fit $n \times p$ targets (elements of Y). Massive overfitting.
- "Force" generalization by choosing a $c \ll p$ and asserting
$$Y_{n\times p} \approx U_{n\times c}V^\mathsf{T}_{p\times c}$$
- What do we mean by $\approx$? Minimize squared error over the observed data:
$$\min_{U,V} \sum_{i=1}^{n}\sum_{j=1}^{p} \mathbf{IsObs}_{ij}(\mathbf{u}_i\mathbf{v}_j^\mathsf{T} - Y_{ij})^2$$

Final Touch: Regularization

- What do we mean by $\approx$? Minimize squared error over the observed data,

$$\min_{U,V} \sum_{i=1}^{n} \sum_{j=1}^{p} \mathbf{IsObs}_{ij} (\mathbf{u}_i \mathbf{v}_j^\mathsf{T} - Y_{ij})^2 + \lambda \sum_{ij} \mathbf{IsObs}_{ij} \left(\|\mathbf{u}_i\|^2 + \|\mathbf{v}_j\|^2 \right)$$

- Salakhutdinov, Mnih, Hinton, "Restricted Boltzmann Machines for Collaborative Filtering" http://www.machinelearning.org/proceedings/icml2007/papers/407.pdf.svg presents an alternative model also

Bonus: Interpreting the output

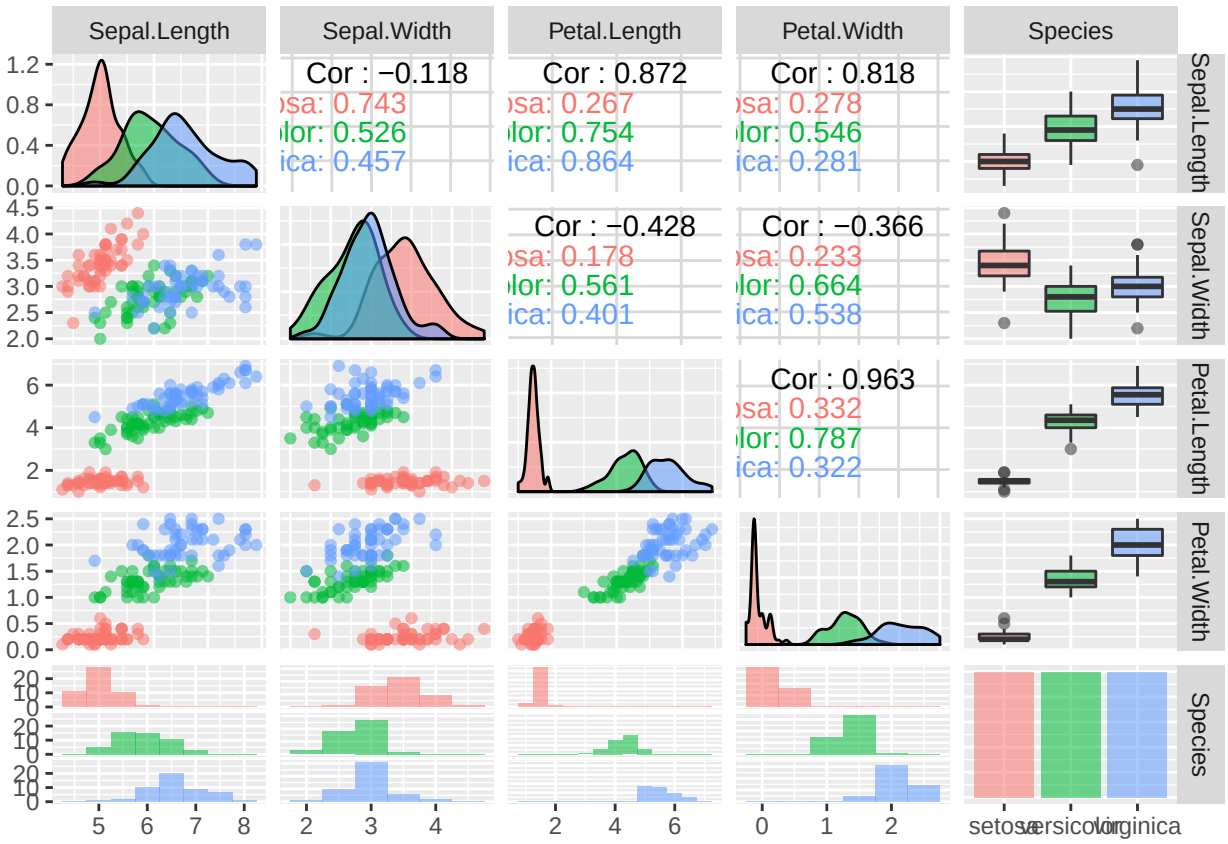
- $\hat{Y}_{ij} = \mathbf{u}_i \mathbf{v}_j^\mathsf{T}$ predicts user i's rating of movie j
  - $\mathbf{u}_i$ is a c-element row vector summarizing person i
  - $\mathbf{v}_j$ is a c-element row vector summarizing movie j
- this is similar to *factor analysis* in statistics

Interpretation

- **Sometimes** (not always) the coefficients can be interpreted if there is structure in the data.
- If a person likes(dislikes) one horror movie, they often like(dislike) other horror movies.
- One way to encode: Use column h of U to represent horror-liking of users, and column h of V to represent horror-ness of movies.
- Sometimes looking at *columns* of U reveal natural user-groupings (e.g. people who like horror movies) and *columns* of V will reveal natural movie-groupings (e.g. horror movies.)
- We can cluster rows of U or V as well.

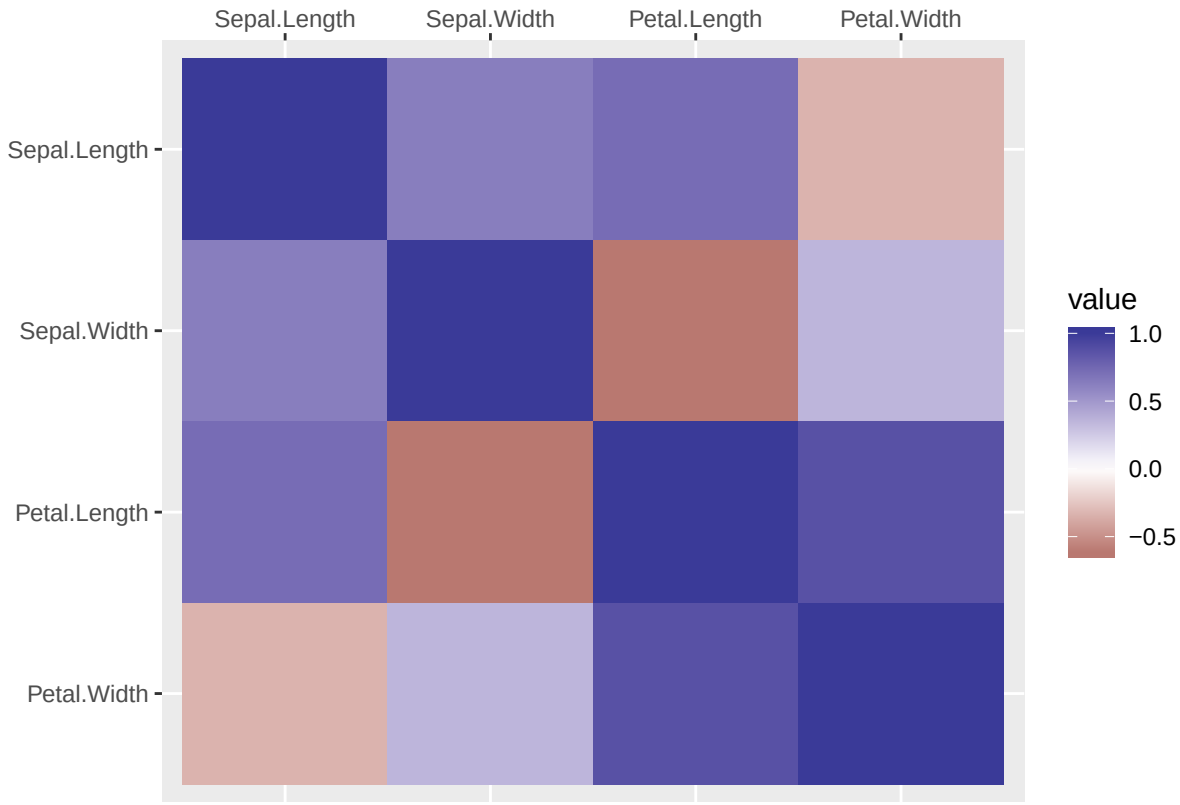
Visualization: Iris dataset

```
library(ggplot2)
library(GGally)
ggpairs(iris, aes(colour = Species, alpha=0.4),lower = list(combo = wrap("facethist", binwidth = 0.5)))
```



Partial correlation matrix

```
library(ggplot2);library(reshape2)
pcormat <- ppcor::pcor(iris[,1:4],method = "pearson")$estimate
melted_pcormat <- melt(pcormat)
ggplot(data = melted_pcormat, aes(x=Var1, y=Var2, fill=value)) +
    scale_y_discrete(name="", limits = rev(levels(melted_pcormat$Var1))) +
    scale_x_discrete(name="", position = "top") +
    geom_tile() + scale_fill_gradient2()
```




```
summary(lm(Sepal.Length ~ Sepal.Width + Sepal.Length + Petal.Width, data = iris))
```

```
## Warning in model.matrix.default(mt, mf, contrasts): the response appeared
## on the right-hand side and was dropped
```

```
## Warning in model.matrix.default(mt, mf, contrasts): problem with term 2 in
## model.matrix: no columns are assigned
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width + Sepal.Length + Petal.Width,
##     data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2076 -0.2288 -0.0450  0.2266  1.1810
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.45733    0.30919   11.18  < 2e-16 ***
## Sepal.Width  0.39907    0.09111    4.38 2.24e-05 ***
## Petal.Width  0.97213    0.05210   18.66  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4511 on 147 degrees of freedom
## Multiple R-squared:  0.7072, Adjusted R-squared:  0.7033
## F-statistic: 177.6 on 2 and 147 DF,  p-value: < 2.2e-16
```