Supervised and Unsupervised Learning

V. Ashley Villar
PSU

PSU Astrostats Summer School

Unsupervised vs Supervised Learning

- Clustering
- Feature selection
- Dimensionality Reduction

- Classification
- Regression
- Forecasting

How to fit a model using MOO

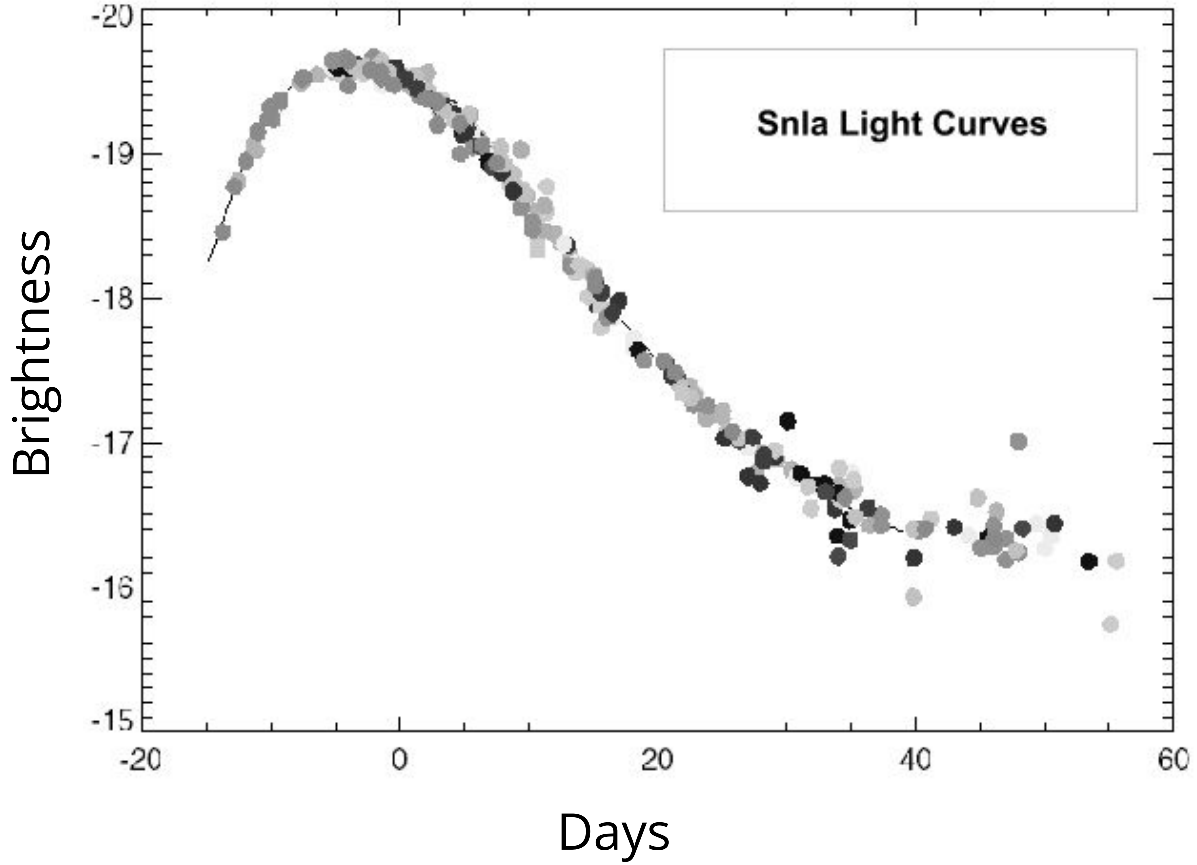
1. **Model** to fit the data (e.g. physics).
2. **Objective Function** (or ‘loss/cost function’) which is a metric that you will choose to quantify how well the model fits the data (e.g. chi-squared).
3. **Optimization Method** which you will use to find the best model (e.g. gradient descent).

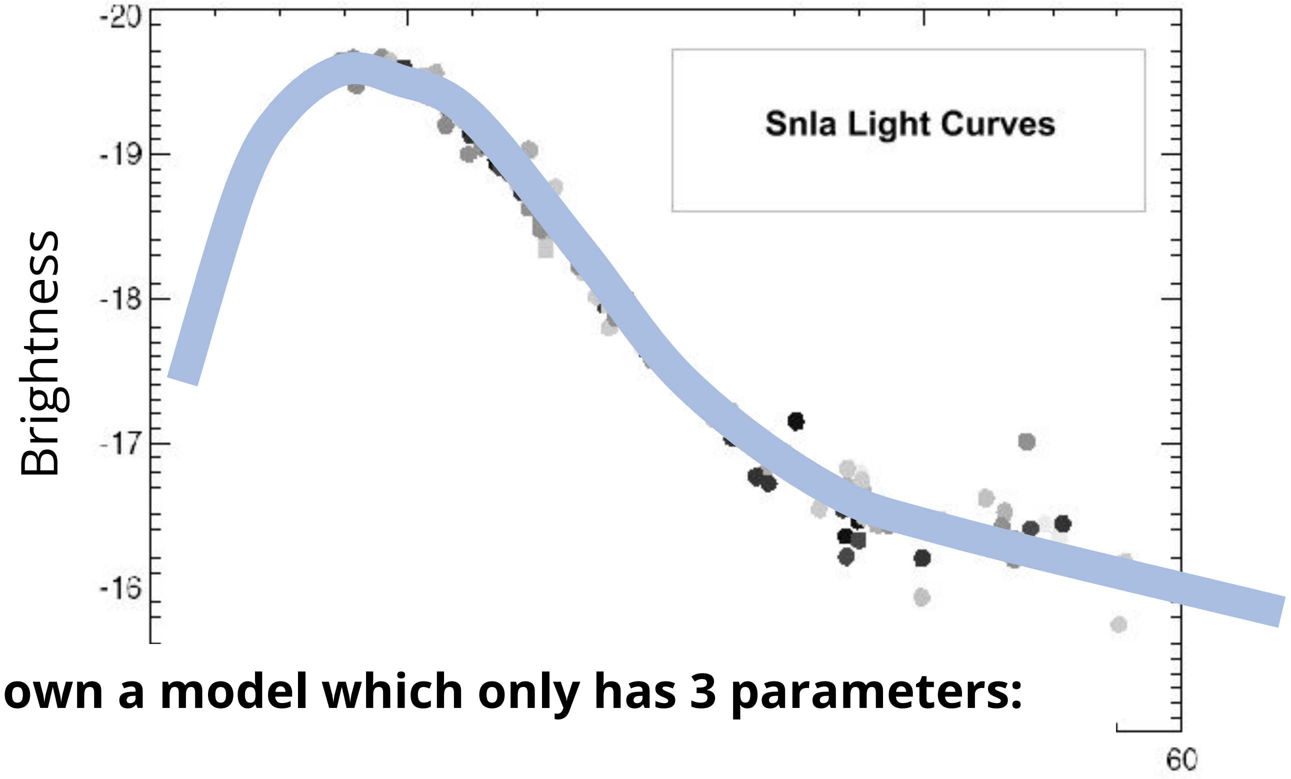
Unsupervised Learning

Learning **structures** within our data

First, let’s focus on **reducing dimensionality**

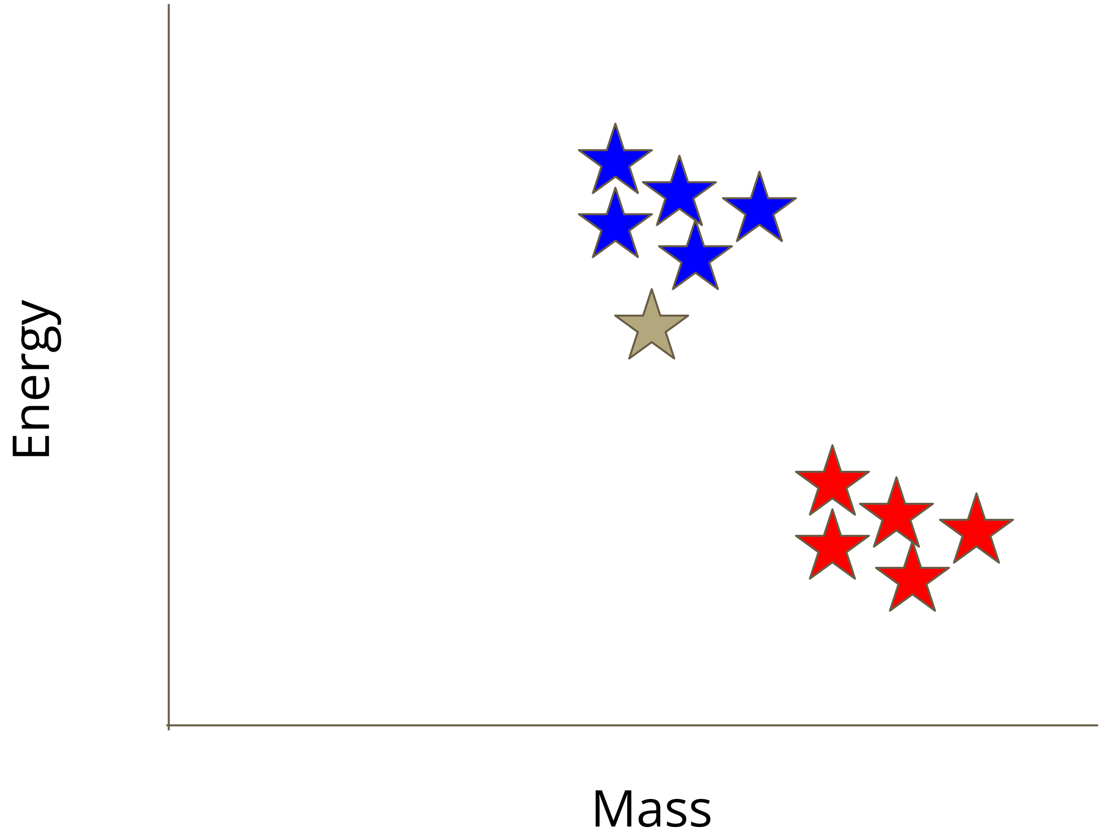
Then, we’ll focus on **clustering**


SnIa Light Curves
Brightness
Days
-20
-19
-18
-17
-16
-15
-20
0
20
40
60



I can write down a model which only has 3 parameters:
Energy,
Mass,
Amount of radioactive material

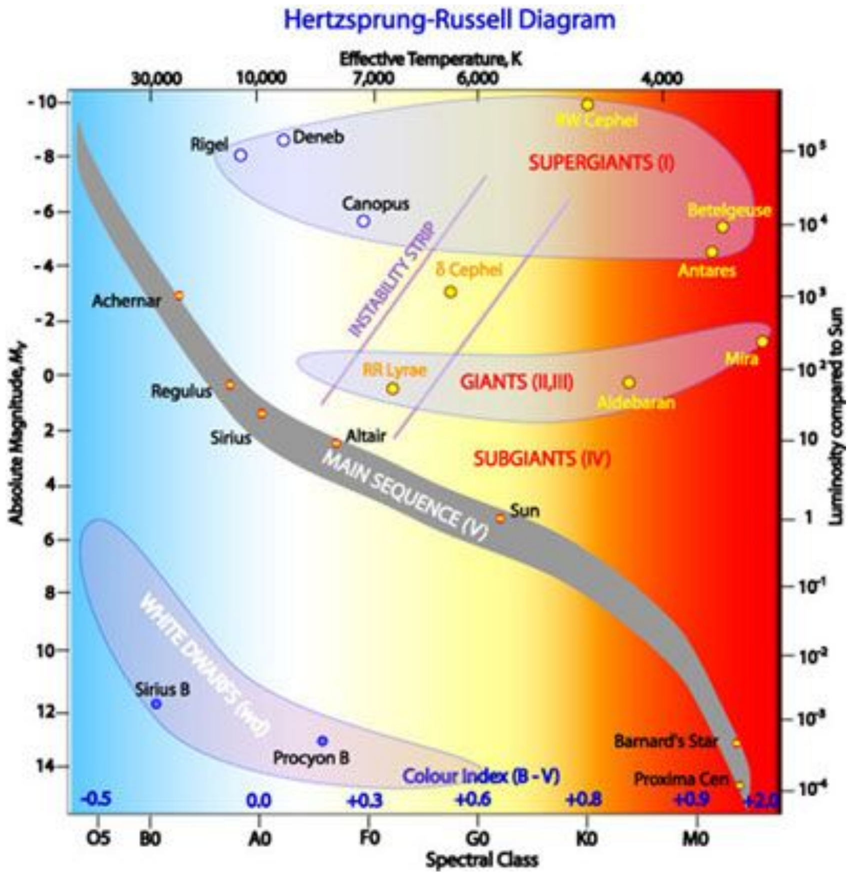
Energy
Mass


Energy
Mass

Let’s call this space a “latent” space. Why “latent”?

How is this different (if at all) from physical model parameters (e.g., mass and energy)?

Why do we care about latent spaces?



Once our data is in a low-dimensional space, we can complete a number of tasks:

1. Better feature selection for classification
2. Anomaly detection
3. Data simulations
4. Physical interpretability

Dimensionality reduction methods

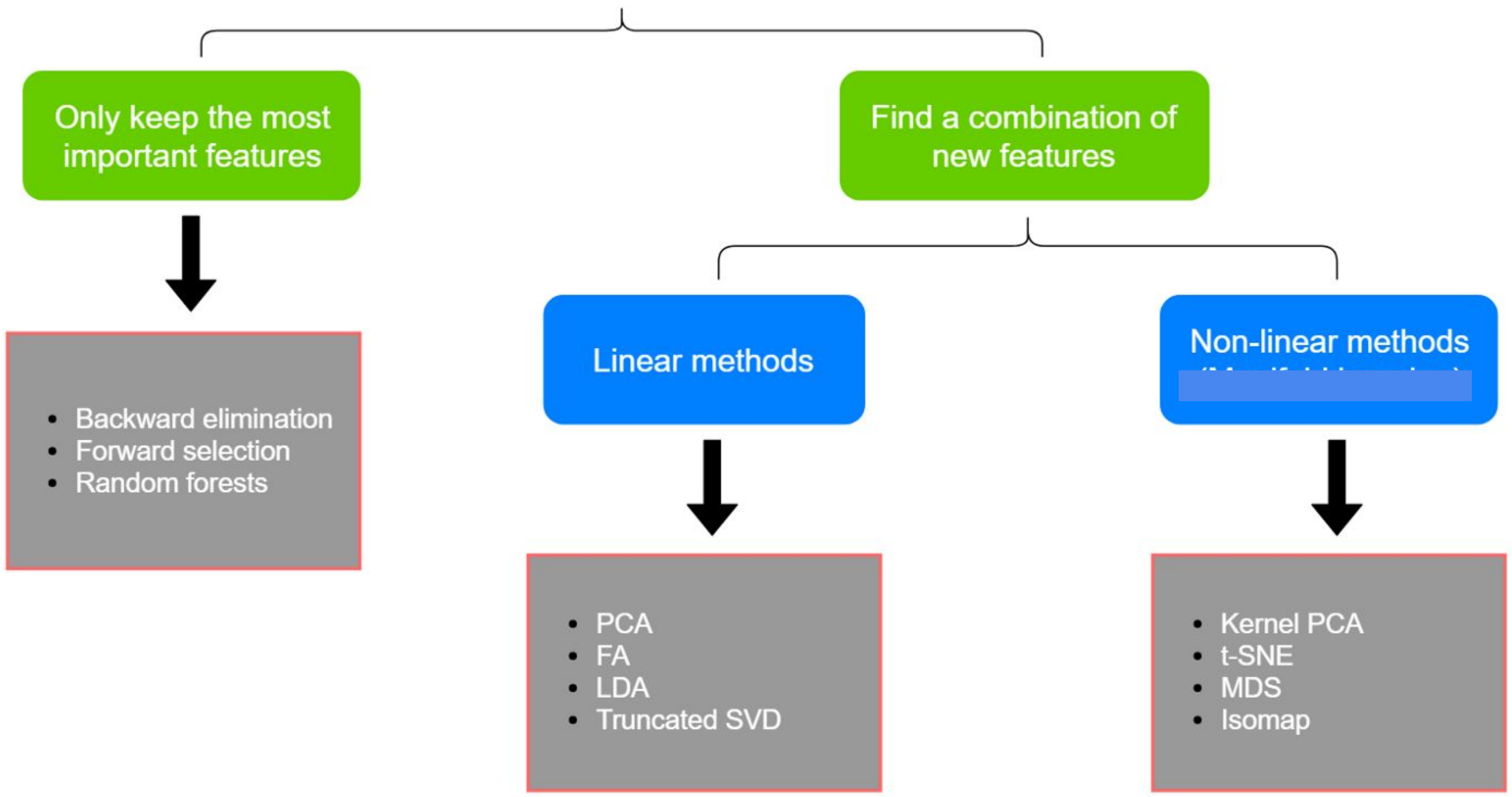
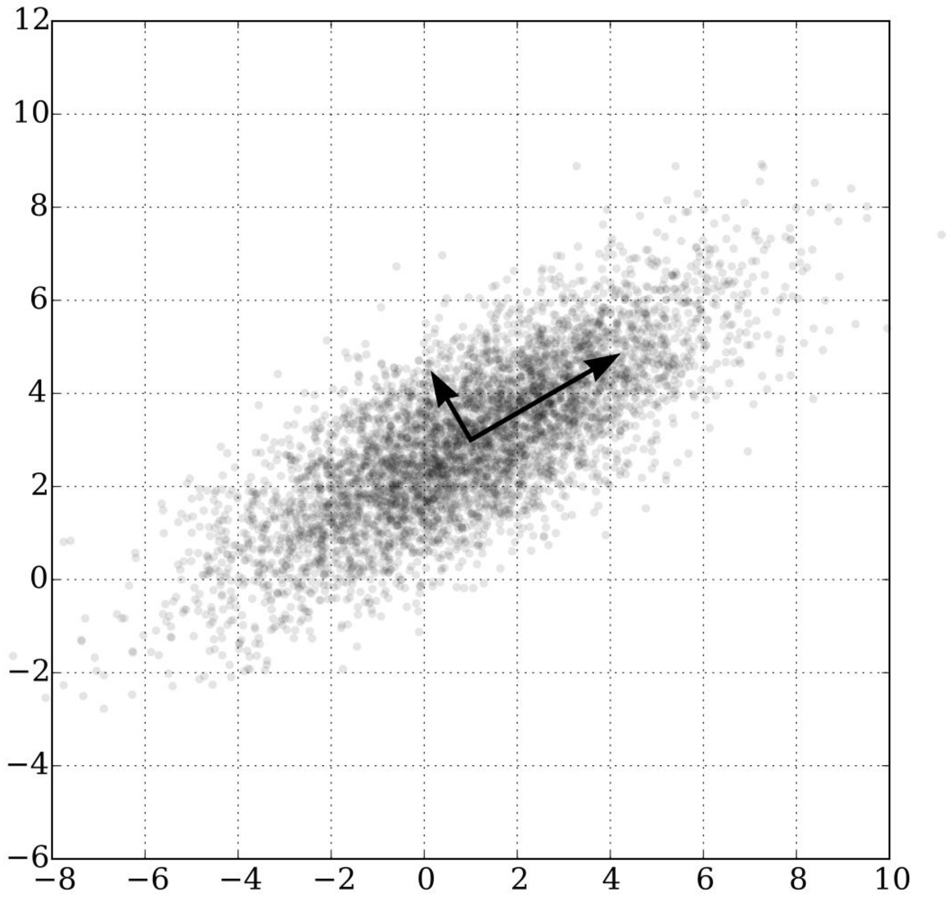


Image copyright: Rukshan Pramoditha

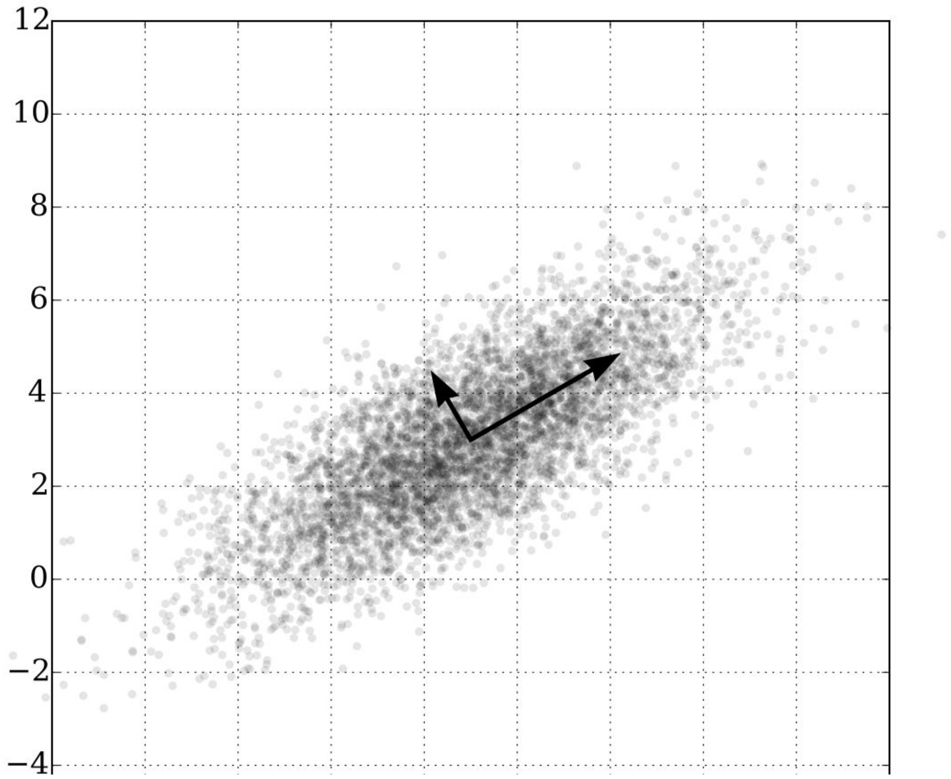
Removing information can be an effective way to remove dimensionality

How can we create a low-dimensional representation without directly fitting a physical model?

The simplest solution is to break down data into **basis vectors**

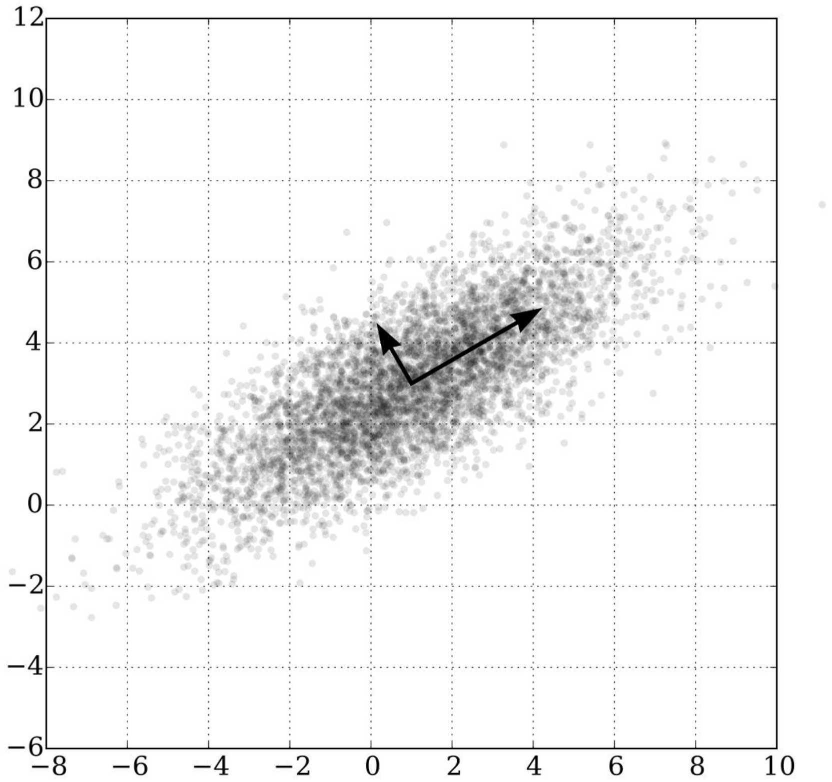


The simplest solution is to break down data into **basis vectors**

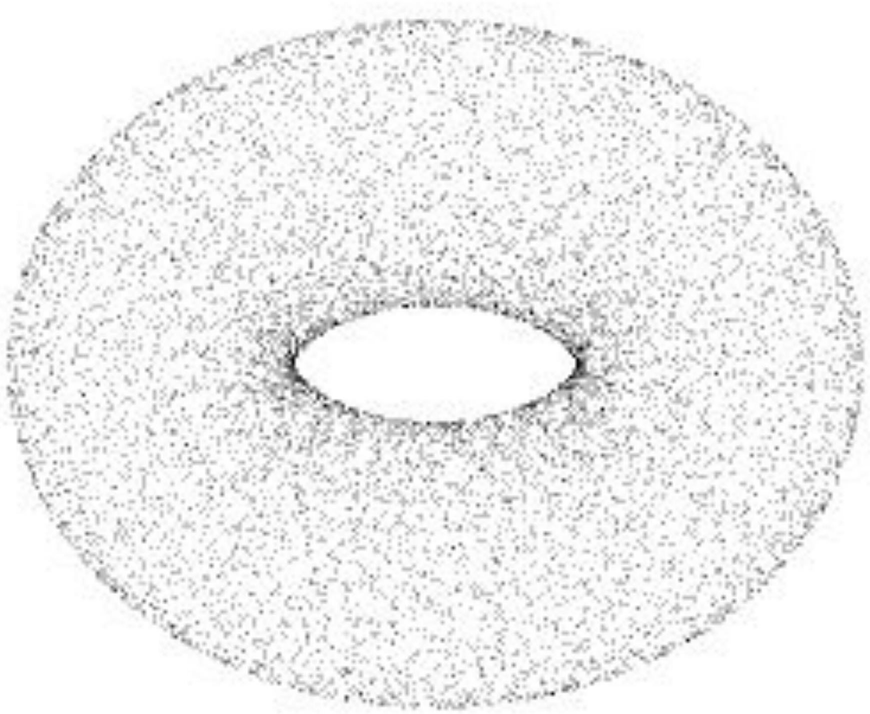


An eigenbasis is one with orthogonal, normalized vectors

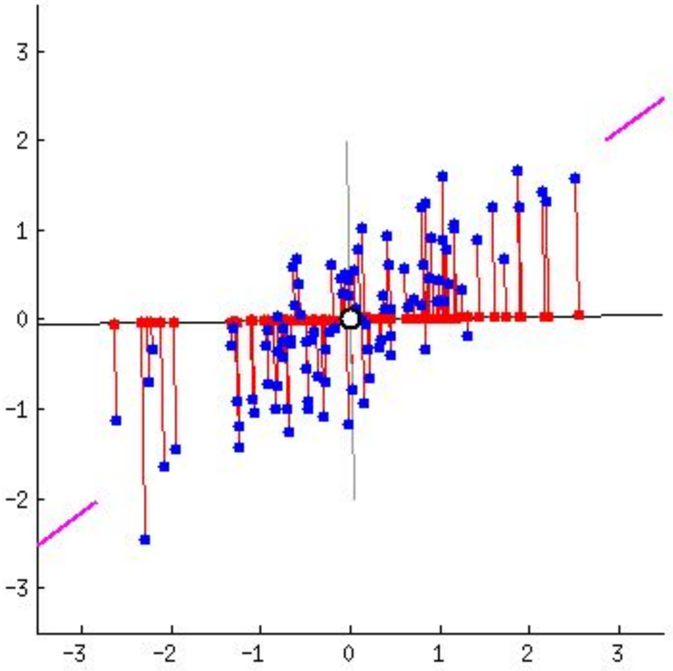
In this 2D example, every point is exactly described by the sum of two eigenvectors



In higher dimensions, 2 detectors may describe ‘most’ of the variance within our data



Each observed datapoint can be projected onto a basis vector

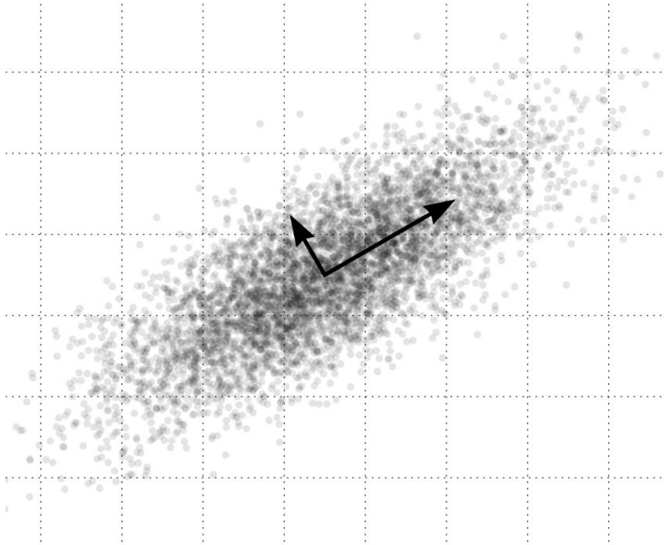


Principal Component Analysis (roughly) has the following steps:

- Find the eigenvectors of the dataset

Principal Component Analysis (roughly) has the following steps:

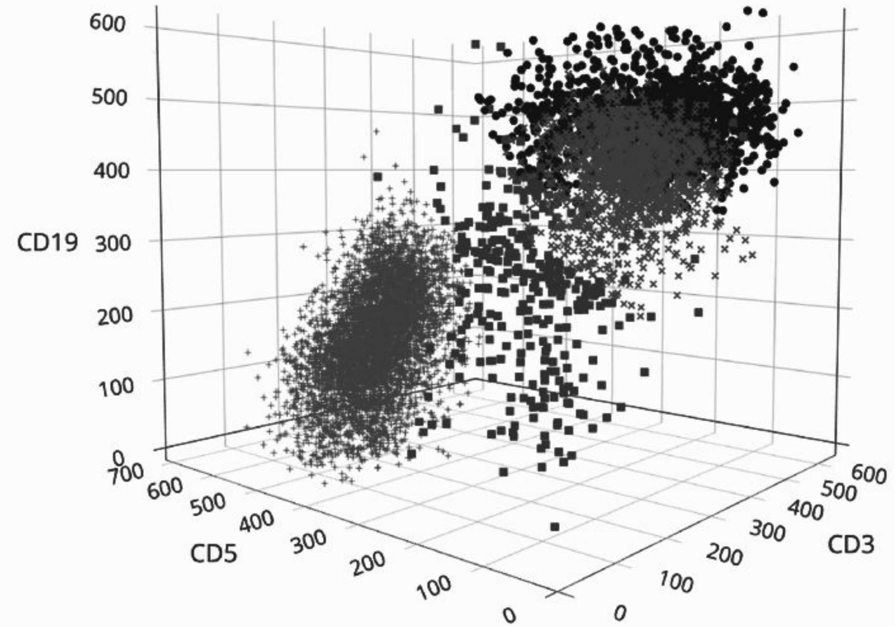
- Find the eigenvectors of the dataset
- Sort the eigenvectors by explained variance
    - Which ones explain the majority of the scatter within the data?

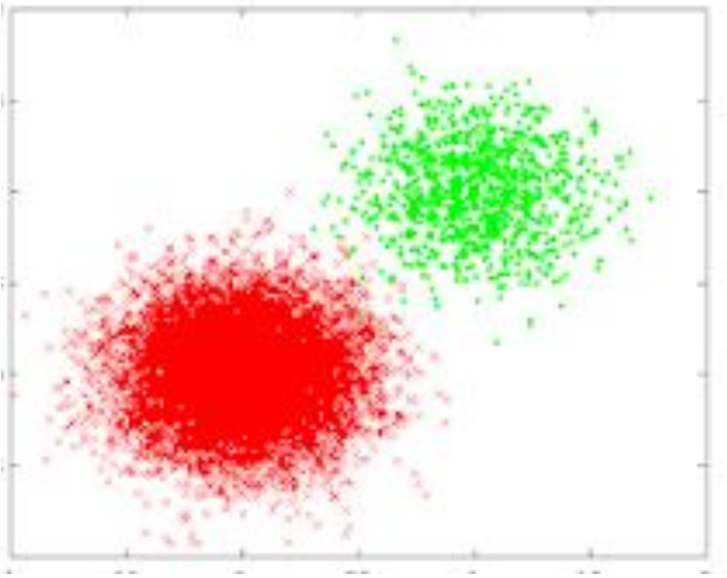


Principal Component Analysis (roughly) has the following steps:

- Find the eigenvectors of the dataset
- Sort the eigenvectors by explained variance
- Project the data onto each basis, tracking the weights
    - For N-dimensional data, each point should be exactly described with N-vectors. So we want to grab the M vectors which describe most of the variance in our data, with $M<N$

PCA transforms our high-dimensional observed space, to a low-dimension **latent space**



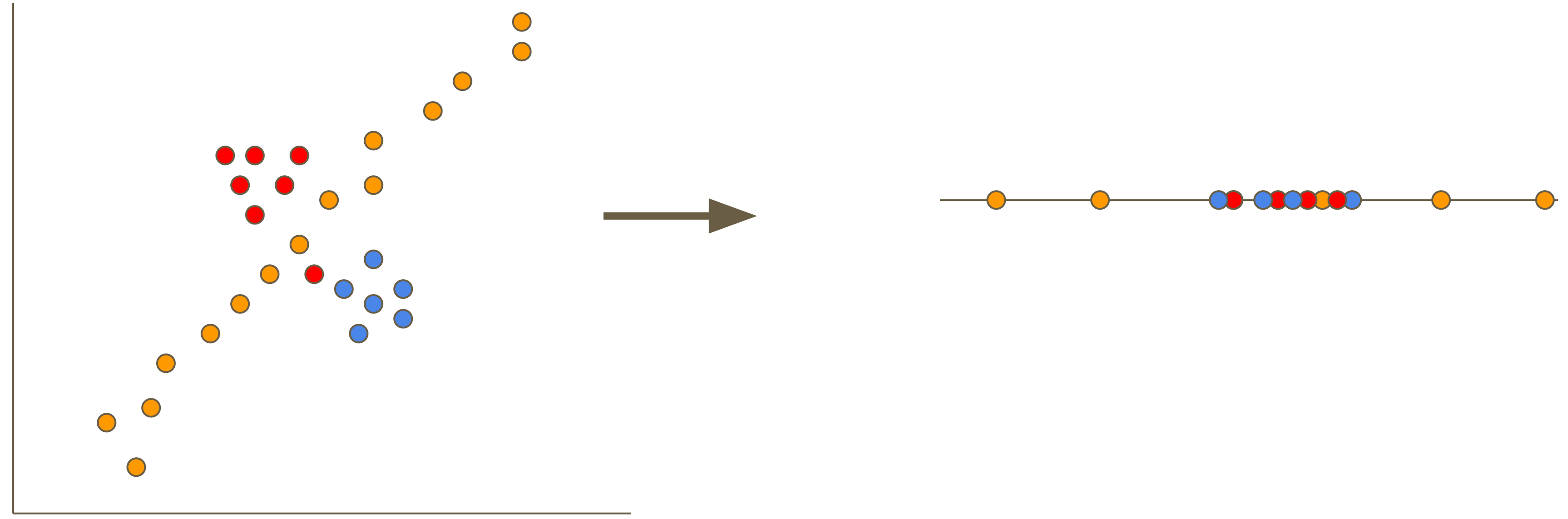


What if my data is high dimensional?

It can be computational expensive to find *every* eigenvector if our data-space is high-dimensional. Instead, we can iteratively find the top k eigenvectors using the **power iteration method**:

1. Given $M=X^TX$, select a random vector v_0
2. For i = 1, 2, . . . , let $v_i = M_i v_0$.
3. If $v_i / |v_i| \approx v_{i-1} / |v_{i-1}|$, then return $v_i/|v_i|$ as an approximation for the first component (w_1)
4. Project our data orthogonally to w_1 Repeat steps 1-3 to find the next PCA component. Repeat for k components

PCA is limited by it linear nature



Group chat - discuss the following questions

1. What does it mean to have a ‘linear’ transformation of our data?
2. Can you give an example of a non-linear transformation in astronomical data?

PCA is limited in its linear assumption, but other, more sophisticated, methods exist as well

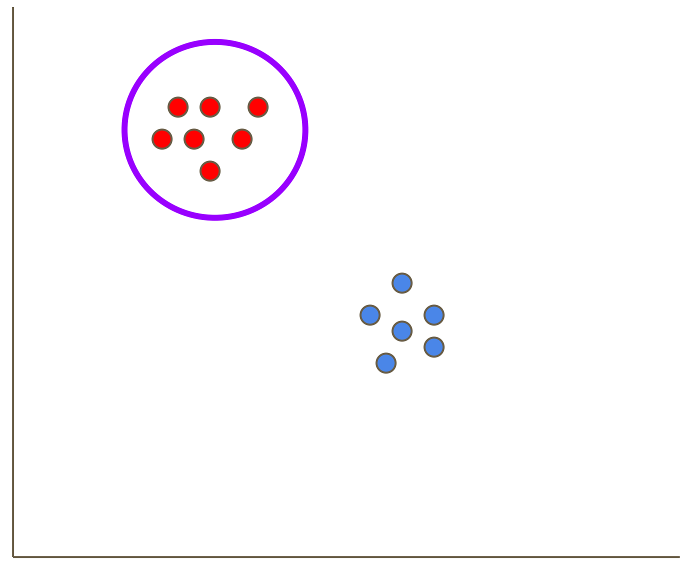
(e.g., t-sne, UMAP)

t-distributed stochastic neighbor embedding

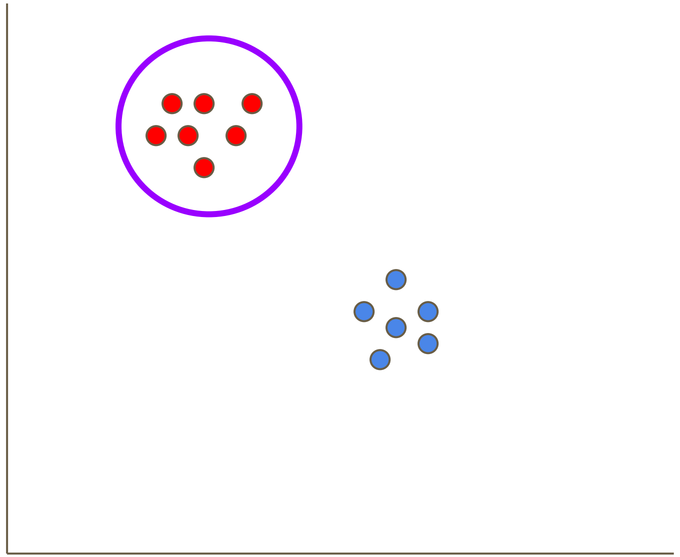
(t-SNE or “tee-snee”)

t-distributed stochastic neighbor embedding

Neighbor embedding: quantify which observations are similar

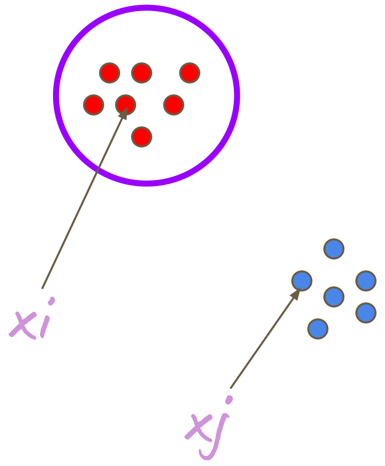


Define a distance metric (e.g., Euclidean distance)



$$d_{i,j}^2 = ||x_i - x_j||^2$$

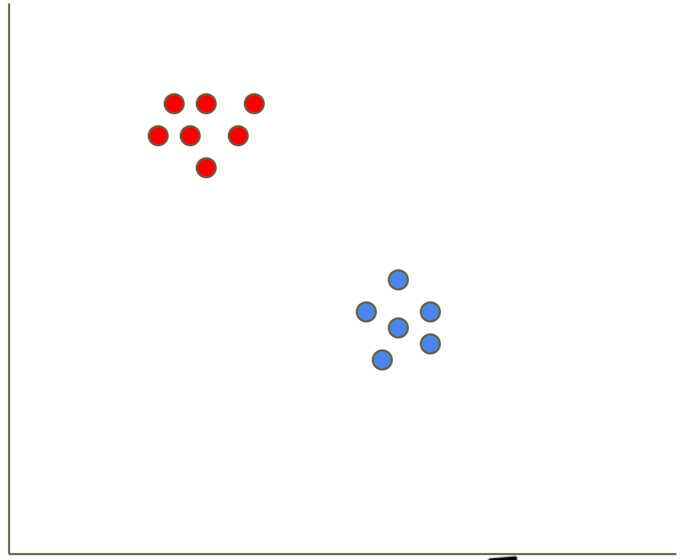
Think of this distance as being proportional to the chance that observations are “neighbors”

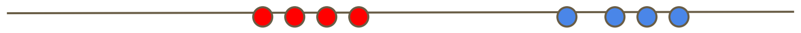


$$p_{i|j} \propto \exp\left(-\frac{||x_i - x_j||^2}{2\sigma_j^2}\right)$$

Free parameter to tune

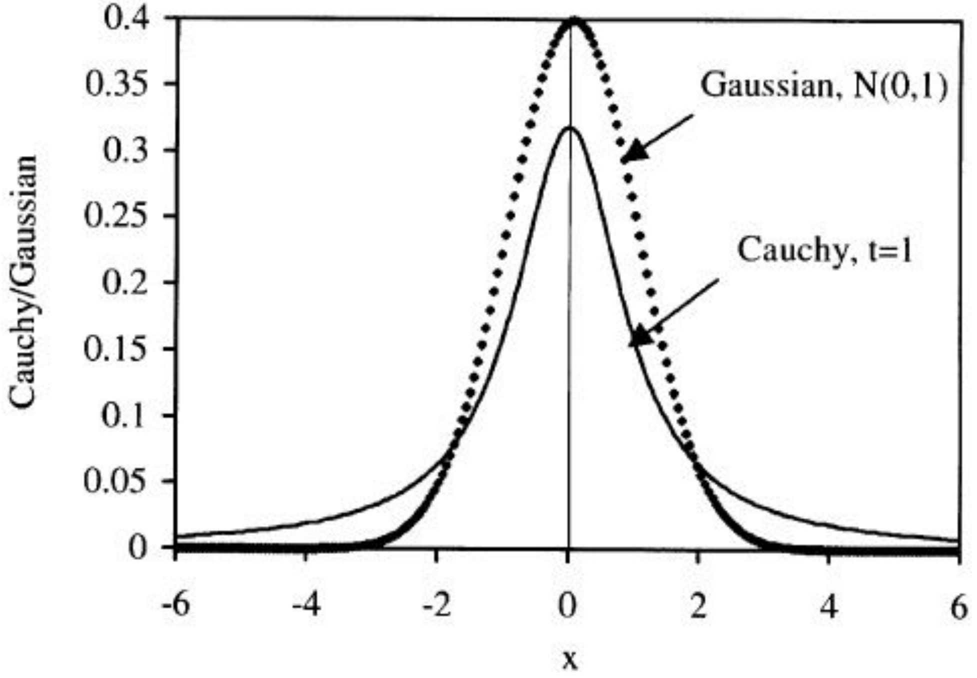
t-SNE aims to learn an embedding which preserves distance measures in the latent space





$$d_{i,j} \simeq d'_{i,j}$$

There are cases in which we *can't* achieve this embedding perfectly, but to help, we will use a Student **t-distribution**



Intuition: Try to match distances for neighbors, but the distance between neighborhoods can be fudgey

In observed space:

$$p(x_i|x_j)$$

In observed space:

$$p(x_i|x_j)$$

In latent space:

$$q(x'_i|x'_j)$$

In observed space:

$$p(x_i|x_j)$$

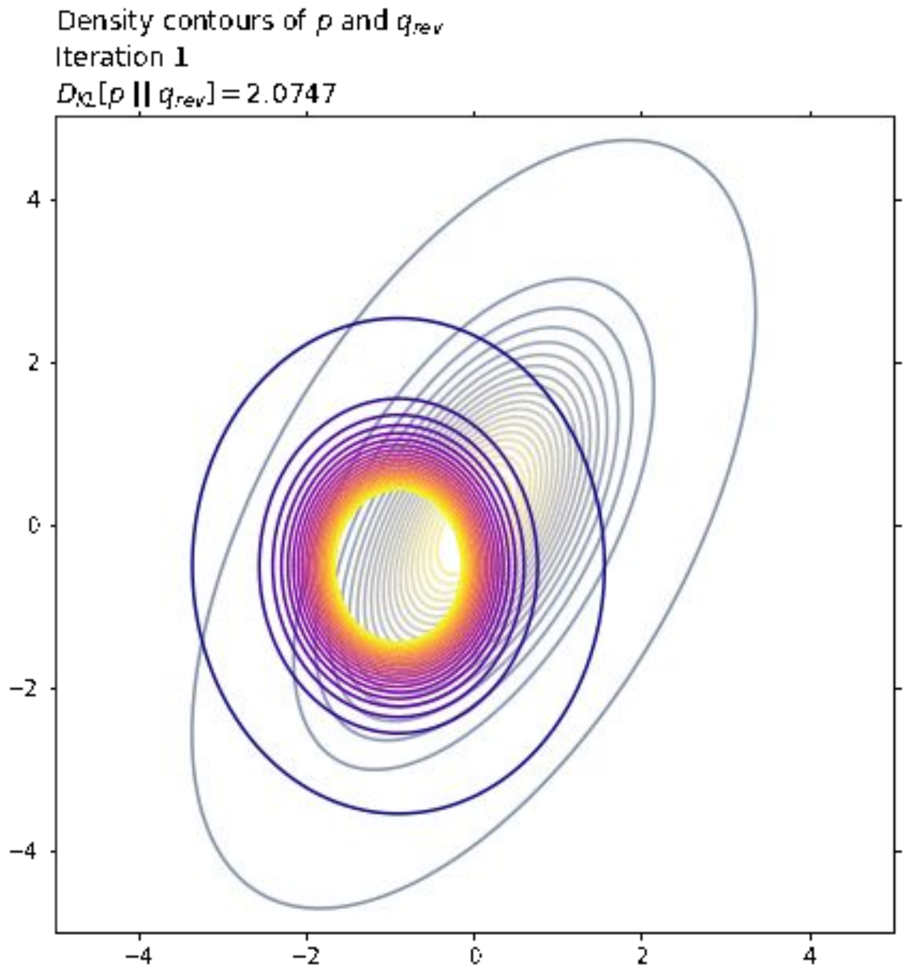
In latent space:

$$q(x'_i|x'_j)$$

We want to minimize the difference between these distributions

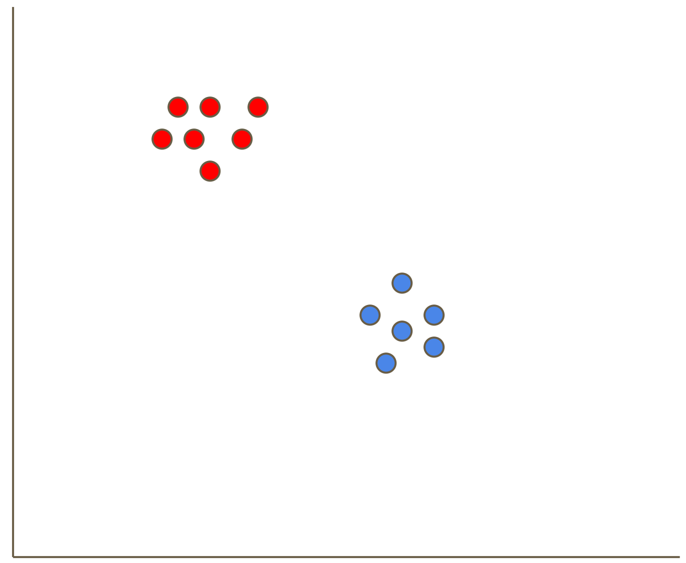
KL divergence of KL(p|q) tries to match the probability distributions of p(xi|xj) and q(x'i|x'j)

We will minimize the KL divergence using gradient descent



t-distributed stochastic neighbor embedding

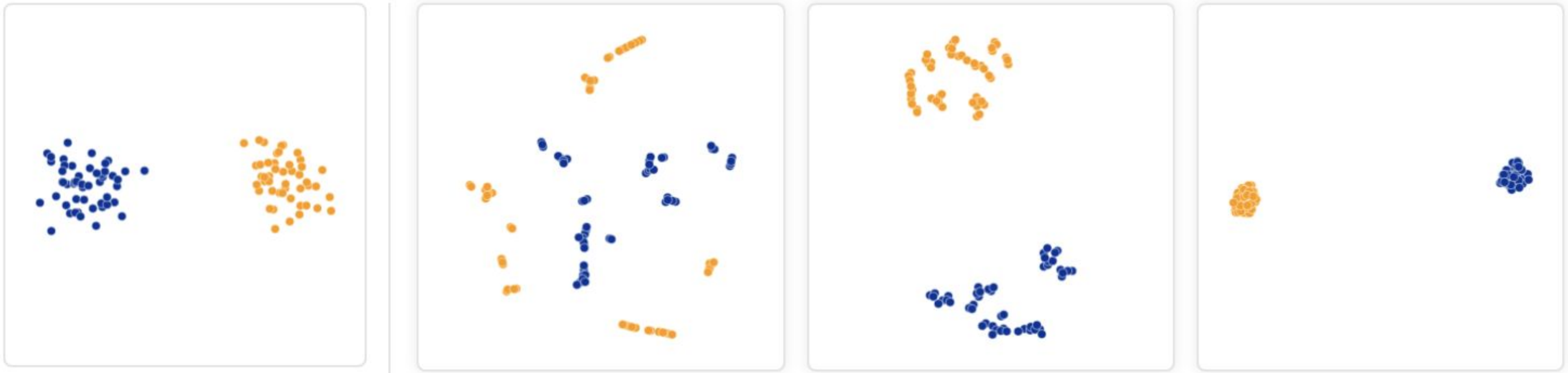
Randomly choose points while optimizing distance metrics….



So while PCA is deterministic, t-SNE is a stochastic method

“Perplexity” is important!

Perplexity ~ number of points expect in each cluster, a hyperparameter



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

https://distill.pub/2016/misread-tsne/

Pros and Cons of each method

PCA	t-sne
Computationally fast	Preservation of local structure
Simple interpretations of latent spaces	Performs especially well in 2D cases
Limited expressiveness due to linearity	Extremely expensive
	Complex interpretation

Clustering

We will explore three algorithms: Gaussian mixture models, K-means and DBScan

M(odel):

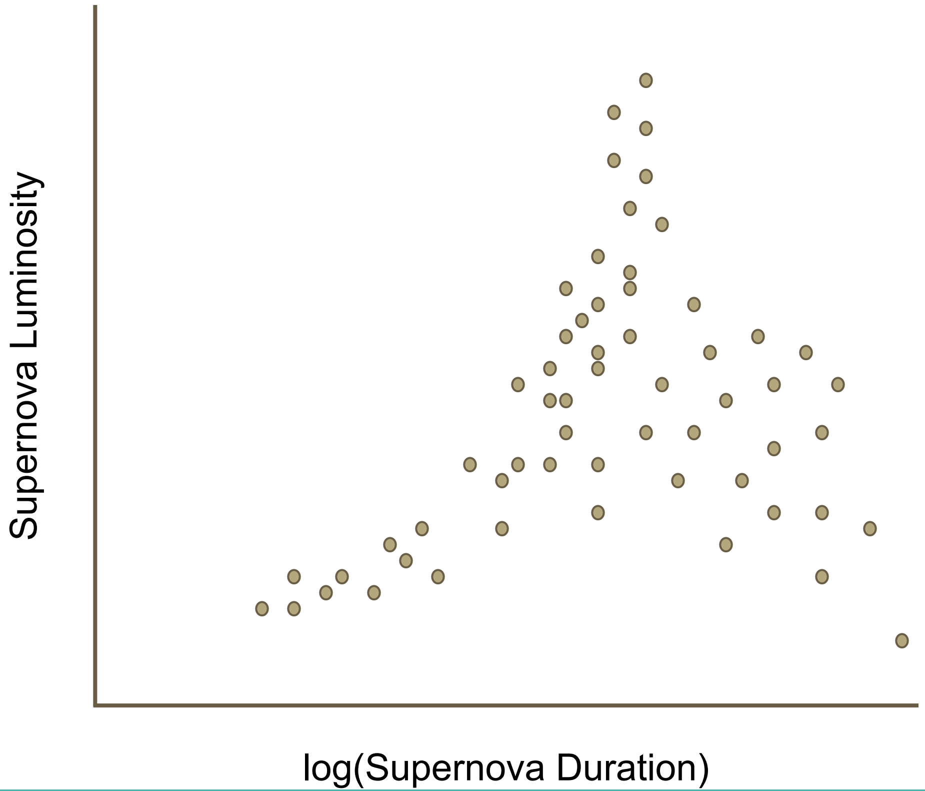
O(objective function):

O(ptimization):

K-means

M(odel):

K-clusters, of any shape. K must be chosen by the user.



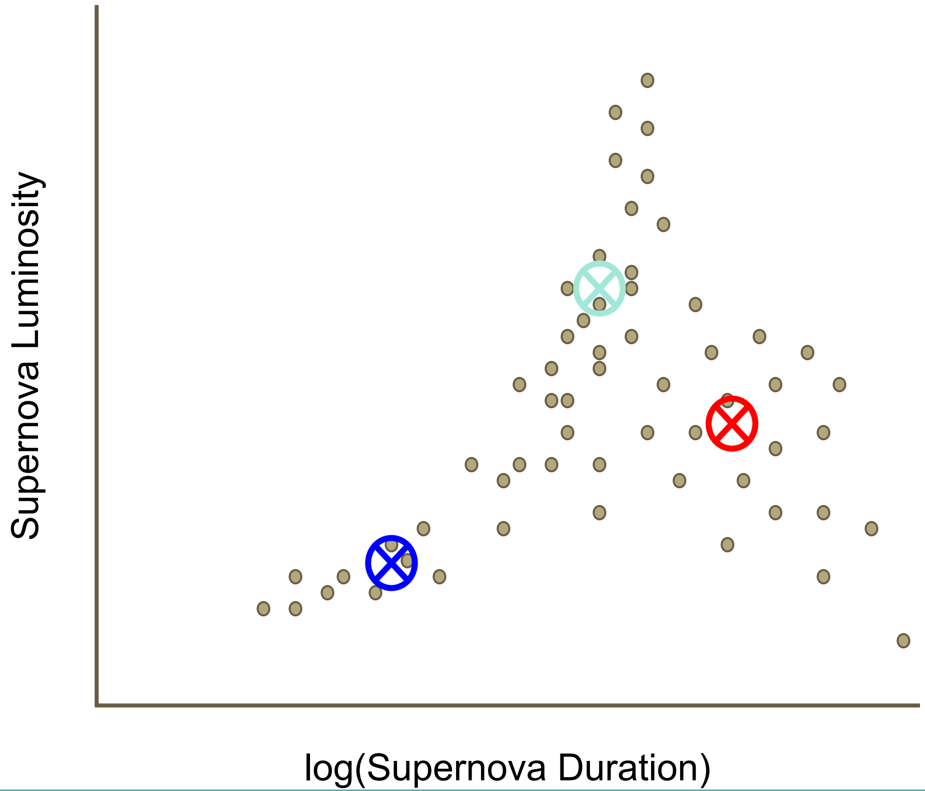
Type II Supernovae

Type Ia Supernovae

Type Ic Supernovae?

M(odel):

Each centroid has 1 free parameters: the mean μ_k



Type II Supernovae

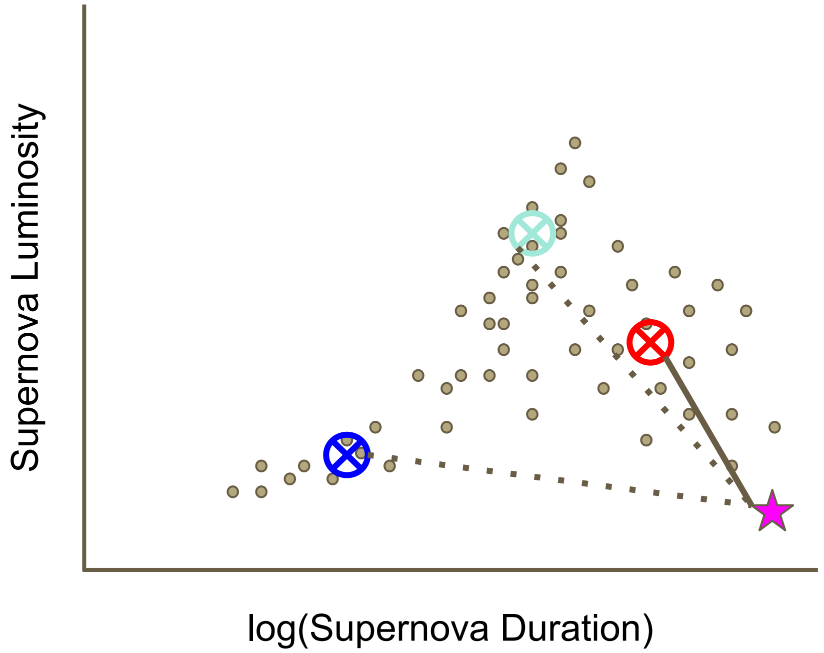
Type Ia Supernovae

Type Ic Supernovae?

$$\Theta = \{\mu_1, \mu_2 \ldots \mu_K\}$$

M(odel): Each observation comes from 1 of K clusters $\Theta = \{\mu_1, \mu_2 ... \mu_K\}$

O(bjective function):



$$\sum_{i=1}^{N} \min_{c_k \in C} (||x_i - \mu_k||^2)$$

This is called "inertia"

M(odel): Each observation comes from 1 of K clusters $\Theta = \{\mu_1, \mu_2 ... \mu_K\}$

O(bjective function): $$\sum_{i=1}^{N} \min_{c_k \in C} (||x_i - \mu_k||^2)$$

O(ptimization): gradient descent?

M(odel): Each observation comes from 1 of K clusters $\Theta = \{\mu_1, \mu_2 ... \mu_K\}$

O(objective function): $$\sum_{i=1}^{N} \min_{c_k \in C} (||x_i - \mu_k||^2)$$

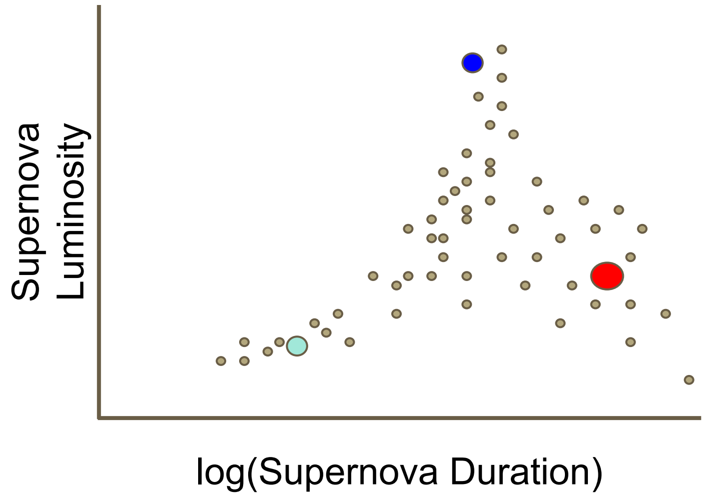
😱

O(ptimization): gradient descent?

How do I take the gradient of a "minimum" function??

O(ptimization): Expectation-maximization!

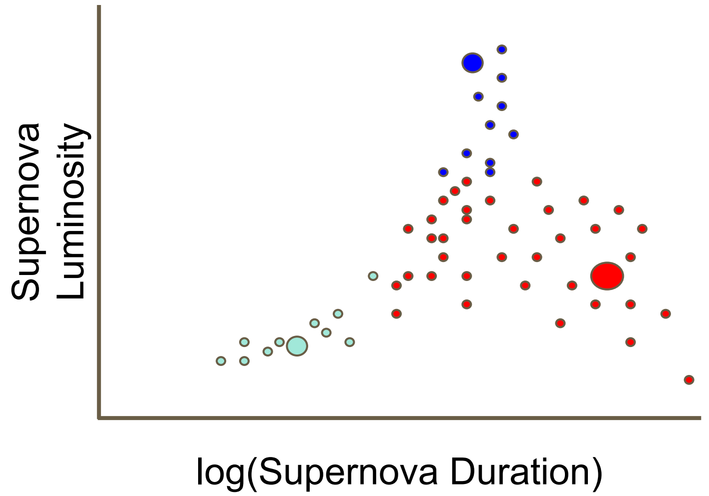
Step 0: Randomly choose K data points and call those your centers



O(ptimization): Expectation-maximization!

Step 0: Randomly choose K data points and call those your centers

Step 1 (Expectation): Assign each observed datapoint to a cluster



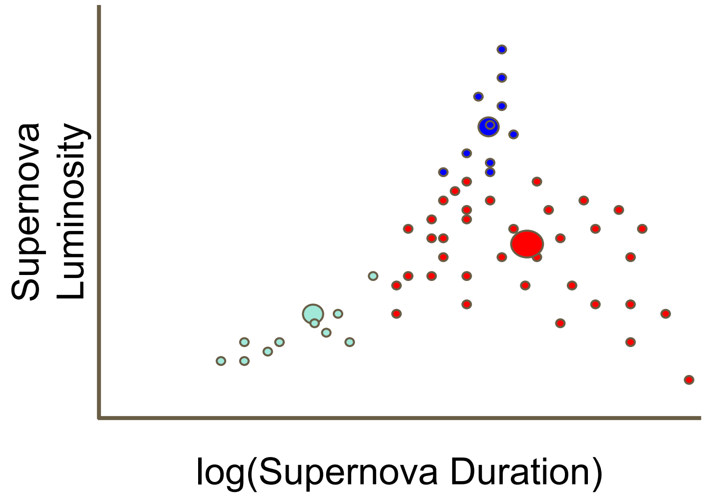
O(ptimization): Expectation-maximization!

Step 0: Randomly choose K data points and call those your centers

Step 1 (Expectation): Assign each observed datapoint to a cluster

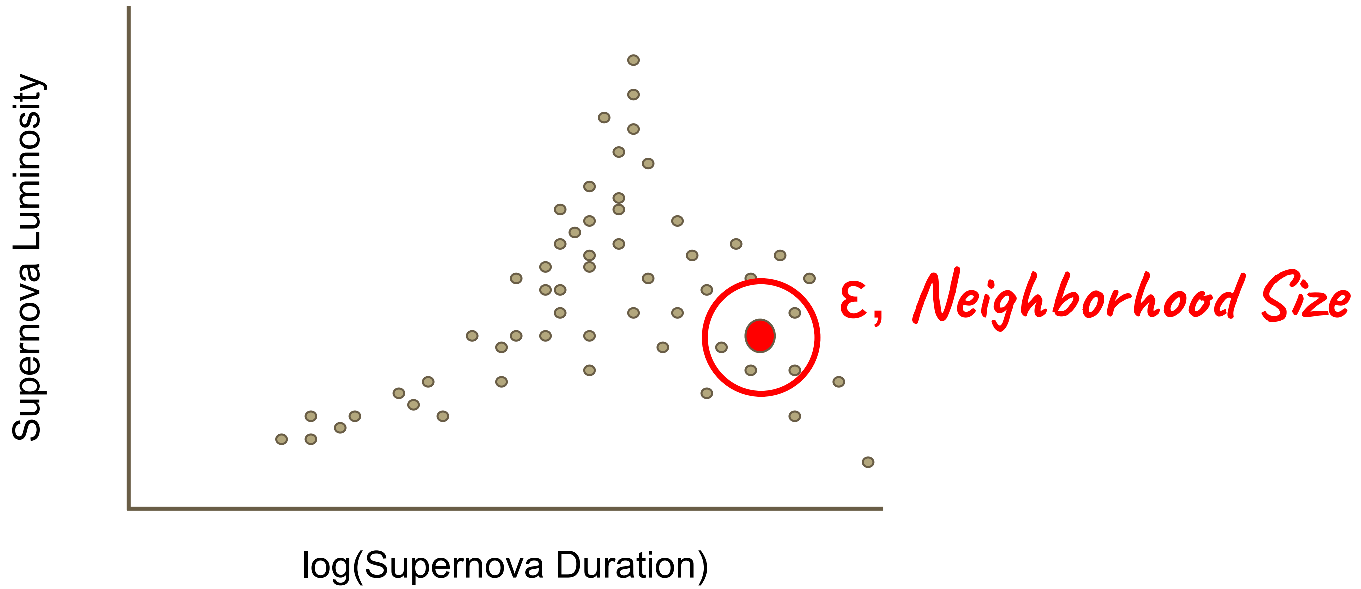
Step 2 (Maximization): Choose new centers of each cluster

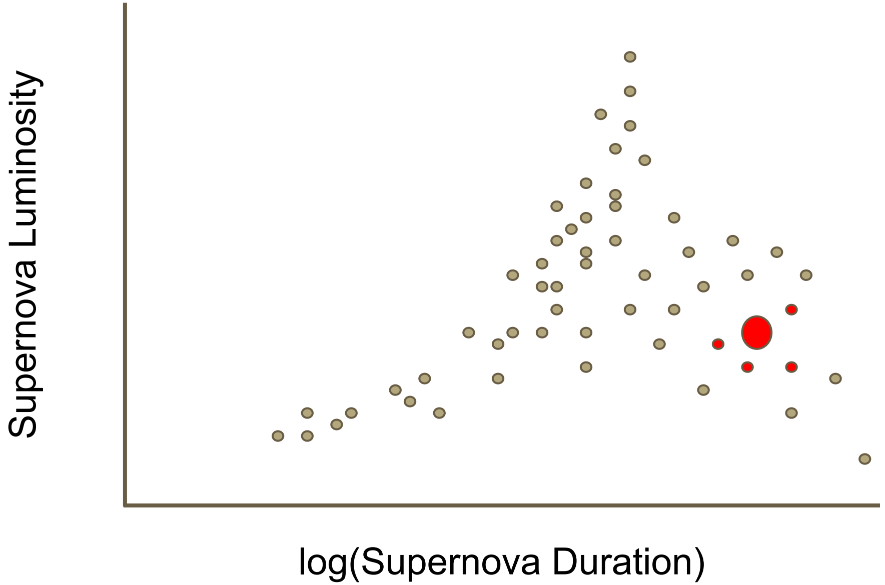
Repeat steps 1/2 until convergence

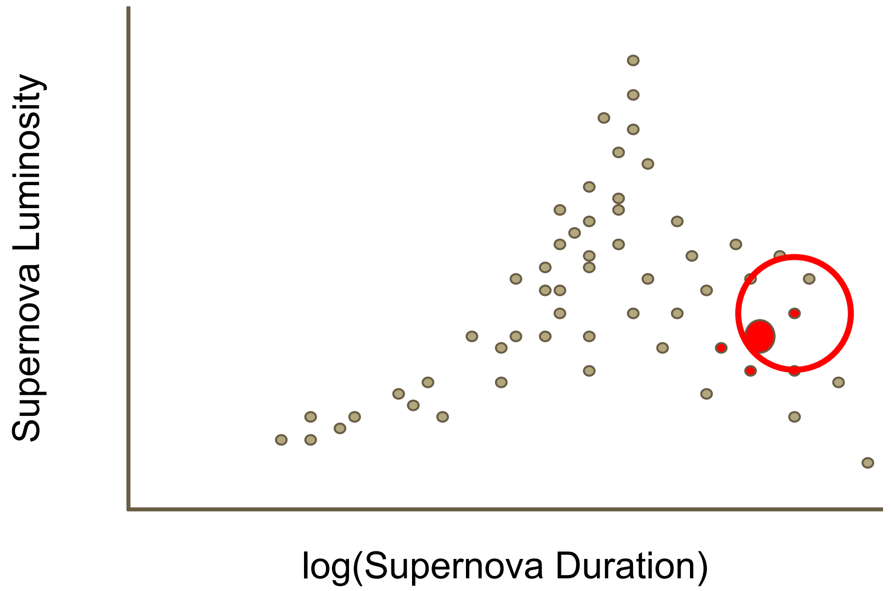


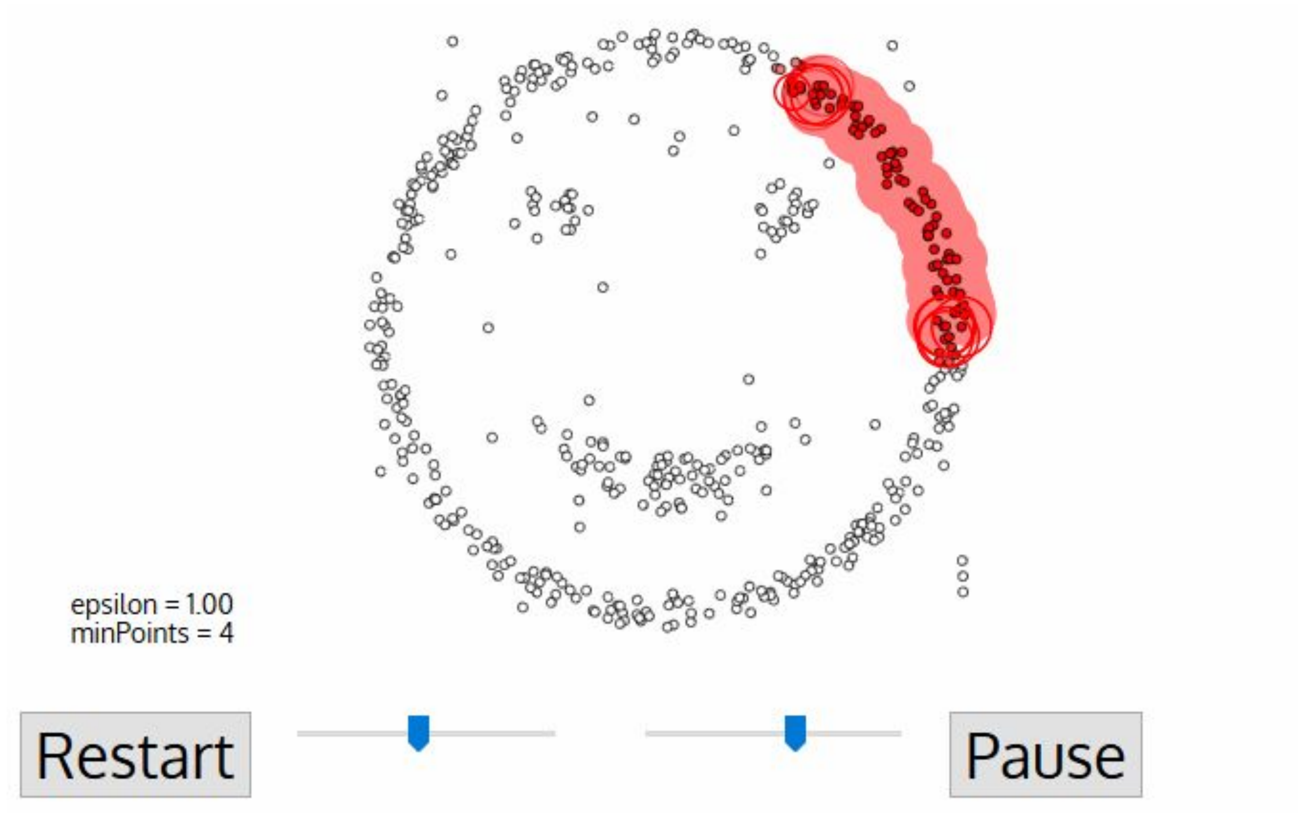
DB-SCAN

(An extremely abbreviated description)


Supernova Luminosity
log(Supernova Duration)
ε, Neighborhood Size

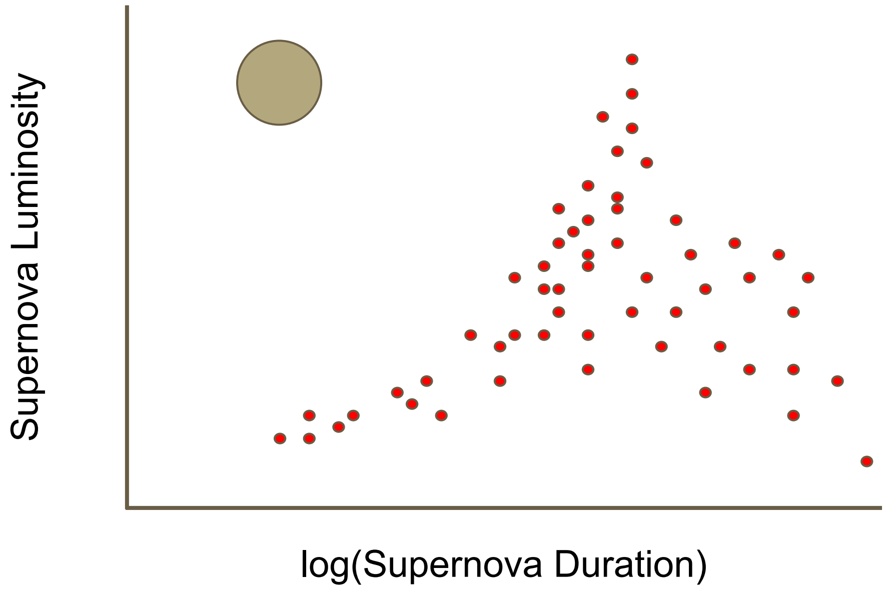

Supernova Luminosity
log(Supernova Duration)


Supernova Luminosity
log(Supernova Duration)

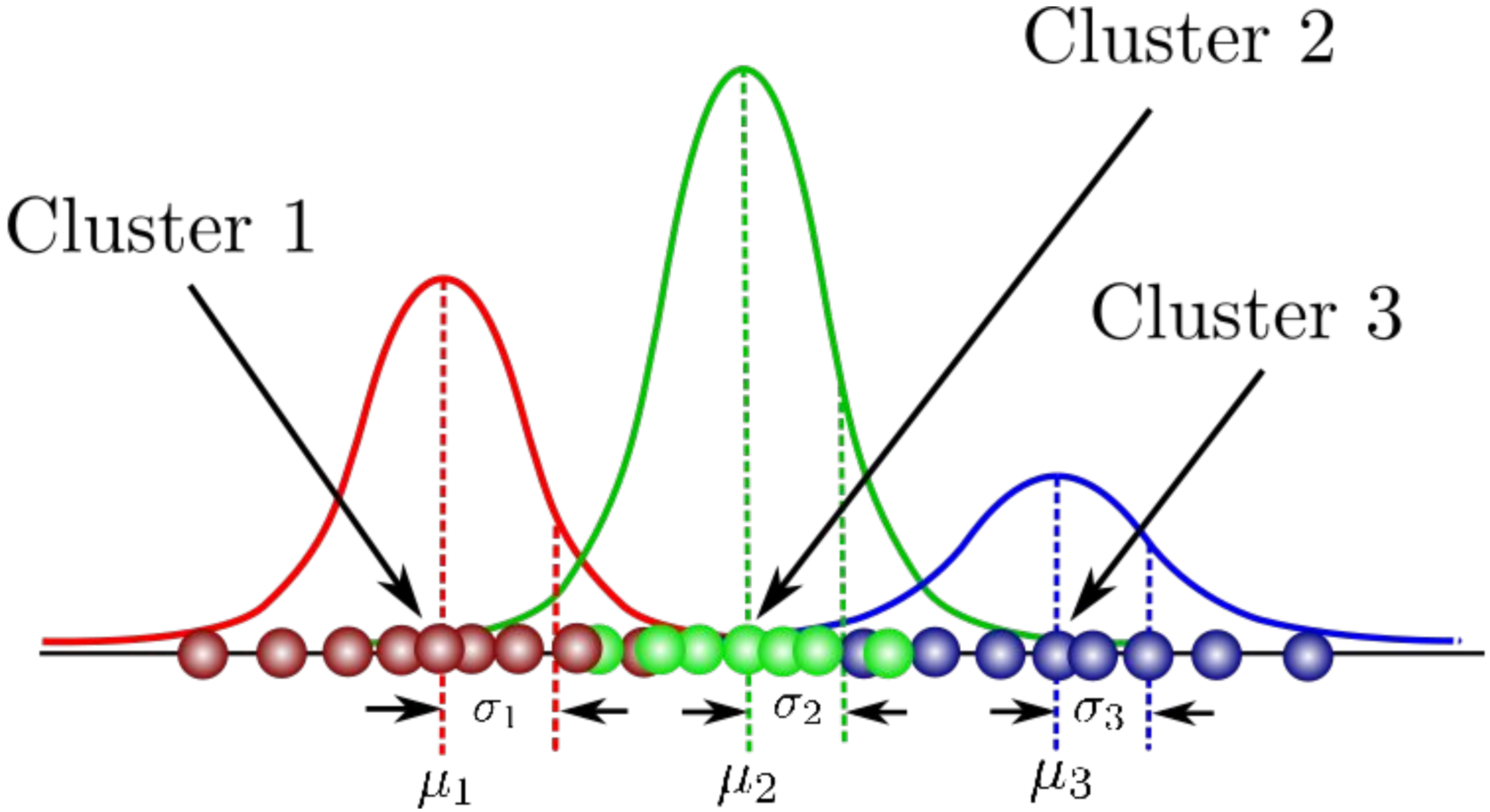


epsilon = 1.00
minPoints = 4

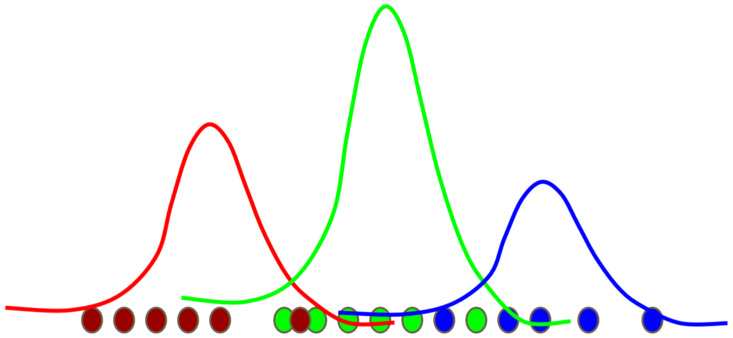
If some points cannot be reached, they will not be clustered and considered an outlier!



Gaussian Mixture Models

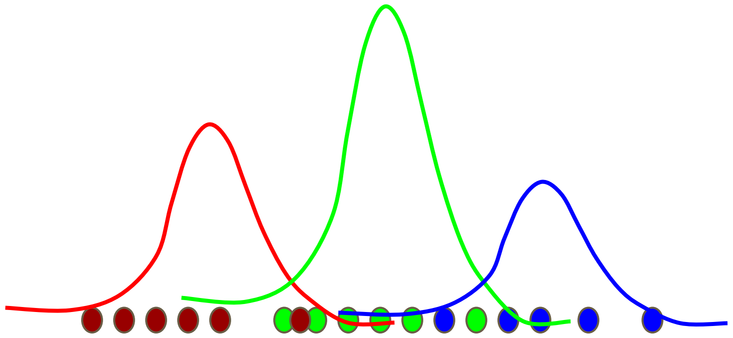


Example: I measure the masses of different degenerate objects, including BHs and NSs. Now, with a new mass measurement, what is the probability that it is a NS?



$$p(x|\Theta) = \sum_{k=1}^{K} \alpha_k p_k(x|\mu_k, \sigma_k)$$

$$\Theta = \{\alpha_1, \mu_1, \sigma_1, \ldots \alpha_k, \mu_k, \sigma_k\}$$



Mixture weights, this of this as the fraction of events which should fall in each component

Mixture components

$$p(x|\Theta) = \sum_{k=1}^{K} \alpha_k \overbrace{p_k(x|\mu_k, \sigma_k)}$$

Membership weights, or the probability that xi comes from the kth component

$$w_{i,k} = \frac{\alpha_k p_k(x_i|\mu_k, \sigma_k)}{\sum_{m=1}^{K} \alpha_k p_m(x_i|\mu_m, \sigma_m)}$$

Note, we're looking at the 1D case, but this is easily extended into N-dimensions

$$p_k(x|\mu_k, \Sigma_k) = \frac{\exp(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k))}{(2\pi)^{d/2} |\Sigma_k|^{1/2}}$$

M(odel):

O(bjective function):

O(ptimization):

M(odel): A mixture of Gaussians

O(bjective function):

O(ptimization):

M(odel): A mixture of Gaussians

O(bjective function) Log-likelihood! :

$$\sum_{i=1}^{N} \log(p(x_i|\Theta))$$

O(ptimization):

M(odel): A mixture of Gaussians

O(bjective function) Log-likelihood! :

$$\sum_{i=1}^{N} \left(\log \sum_{k=1}^{K} \alpha_k p_k(x_i | \mu_k, \sigma_k) \right)$$

O(ptimization): gradient descent?

Optimization, why not gradient descent?

Positive, semidefinite

$$p_k(x|\mu_k, \Sigma_k) = \frac{\exp(-\frac{1}{2}(x-\mu_k)^T \boxed{\Sigma_k^{-1}}(x-\mu_k))}{(2\pi)^{d/2}|\Sigma_k|^{1/2}}$$

$$\sum_{i=1}^{N}\left(\log \sum_{k=1}^{K} \alpha_k p_k(x_i|\mu_k, \sigma_k)\right)$$

And mixture weights must sum to 1!

AND I have to take the derivative of a sum in a log 😱

Optimization, why not gradient descent?

Tough constraints and a tough likelihood make it hard to calculate the gradient, required for gradient descent!

“Expectation Maximization” Algorithm (EM) for optimization

1. Choose a random set of **parameter values** which satisfy our constraints:

$$\Theta = \{\alpha_1, \mu_1, \sigma_1, \dots \alpha_k, \mu_k, \sigma_k\}$$

You can be more clever than ‘totally random’; for example, using a clustering technique!

“Expectation Maximization” Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (Θ) which satisfy our constraints
2. Expectation step: Compute the expected value of our log-likelihood. In words: *what is the most likely component a single datapoint belongs to?* A probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)

$$\begin{vmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,K} \\ w_{2,1} & \cdots & \cdots & \cdots \\ \cdots & \cdots & w_{i,k} & \cdots \\ w_{N,1} & \cdots & \cdots & w_{N,K} \end{vmatrix}$$

$$w_{i,k} = \frac{\alpha_k p_k(x_i|\mu_k, \sigma_k)}{\sum_{m=1}^{K} \alpha_k p_m(x_i|\mu_m, \sigma_m)}$$

“Expectation Maximization” Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (ϴ) which satisfy our constraints
2. Expectation step: Compute a probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)

$$\left|\begin{array}{cccc} w_{1,1} & w_{1,2} & \cdots & w_{1,K} \\ w_{2,1} & \cdots & \cdots & \cdots \\ \cdots & \cdots & w_{i,k} & \cdots \\ \hline w_{N,1} & \cdots & \cdots & w_{N,K} \\ \hline \end{array}\right|$$

Rows sum to 1, or in words: there is a 100% chance a point has to belong to one of the components!

“Expectation Maximization” Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (ϴ) which satisfy our constraints
2. Expectation step: Compute a probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)
3. Maximization step:
    a. We have new cluster params! Calculate the effective number of samples in each cluster

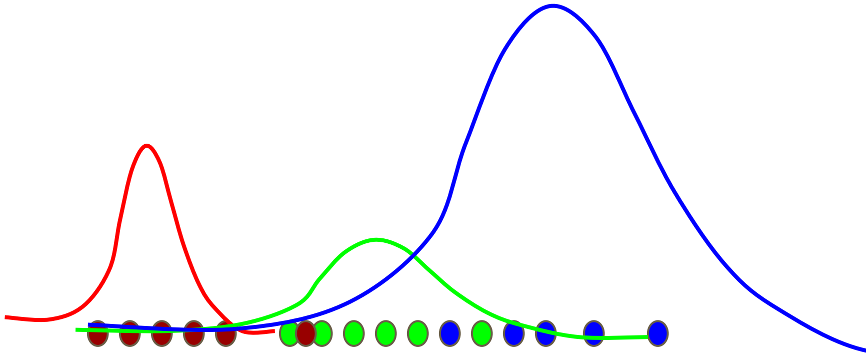
$$\boxed{N_k \equiv \sum_{i=1}^{N} w_{i,k}}$$

$$\left.\begin{matrix} \boxed{\begin{matrix} w_{1,1} \\ w_{2,1} \\ \cdots \\ w_{N,1} \end{matrix}} & \begin{matrix} w_{1,2} \\ \cdots \\ \cdots \\ \cdots \end{matrix} & \begin{matrix} \cdots \\ \cdots \\ w_{i,k} \\ \cdots \end{matrix} & \begin{matrix} w_{1,K} \\ \cdots \\ \cdots \\ w_{N,K} \end{matrix} \end{matrix}\right|$$

"Expectation Maximization" Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (ϴ) which satisfy our constraints
2. Expectation step: Compute a probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)
3. Maximization step:
   a. Calculate the effective number of samples in each new cluster
   b. After doing ~math~, the most likely value of the new component weights is:

$$\alpha_k^{\text{new!}} = \frac{N_k}{N}$$



“Expectation Maximization” Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (Ɵ) which satisfy our constraints
2. Expectation step: Compute a probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)
3. Maximization step:
   a. Calculate the effective number of samples in each new cluster
   b. Calculate the new parameters, then the Gaussian parameters:

$$\mu_k^{\text{new!}} = \frac{1}{N_k} \sum_{i=1}^{N} w_{i,k} x_i \qquad (\sigma_k^2)^{\text{new!}} = \frac{1}{N_k} \sum_{i=1}^{N} (x - \mu)^2$$

"Expectation Maximization" Algorithm (EM) for optimization

1. Choose a random set of **parameter values** (ϴ) which satisfy our constraints
2. Expectation step: Compute a probability matrix which enumerates the probability of each data point (1-N) belonging to each component (1-K)
3. Maximization step:
   a. Calculate the effective number of samples in each new cluster
   b. Calculate the new parameters, then the Gaussian parameters
4. Repeat steps 2/3, each time checking the likelihood. The algorithm has converged when the likelihood remains ~constant

We now have a tool box of 3 clustering techniques:

Gaussian Mixture Models

K-Means

DBSCAN

Parameters?

Gaussian Mixture Models

Properties of each Gaussian component

Hyperparameter: Number of components

K-Means

Centers of each cluster

Hyperparameter: Number of Clusters

DBSCAN

Hyperparameter: Neighborhood Size

Hyperparameter: Minimum density

Note: Hyperparameter means that the algorithm by itself does not optimize this parameter

Pros and Cons

Gaussian Mixture Models

Pro: Extremely interpretable

Pro: Density estimate available everywhere

K-Means

Pro: Minimal parameters

Pro: Computationally scalable to large number of samples

DBSCAN

Pro: Minimal parameters

Pro: Arbitrary cluster shapes

Pros and Cons

Gaussian Mixture Models

Pro: Extremely interpretable

Pro: Density estimate available everywhere

Con: Challenging to scale to high dimensions

Con: Strong assumption on cluster shape

K-Means

Pro: Minimal parameters

Pro: Computationally scalable to large number of samples

Con: Can be non-trivial to pick the "correct" number of clusters

DBSCAN

Pro: Minimal parameters

Pro: Arbitrary cluster shapes

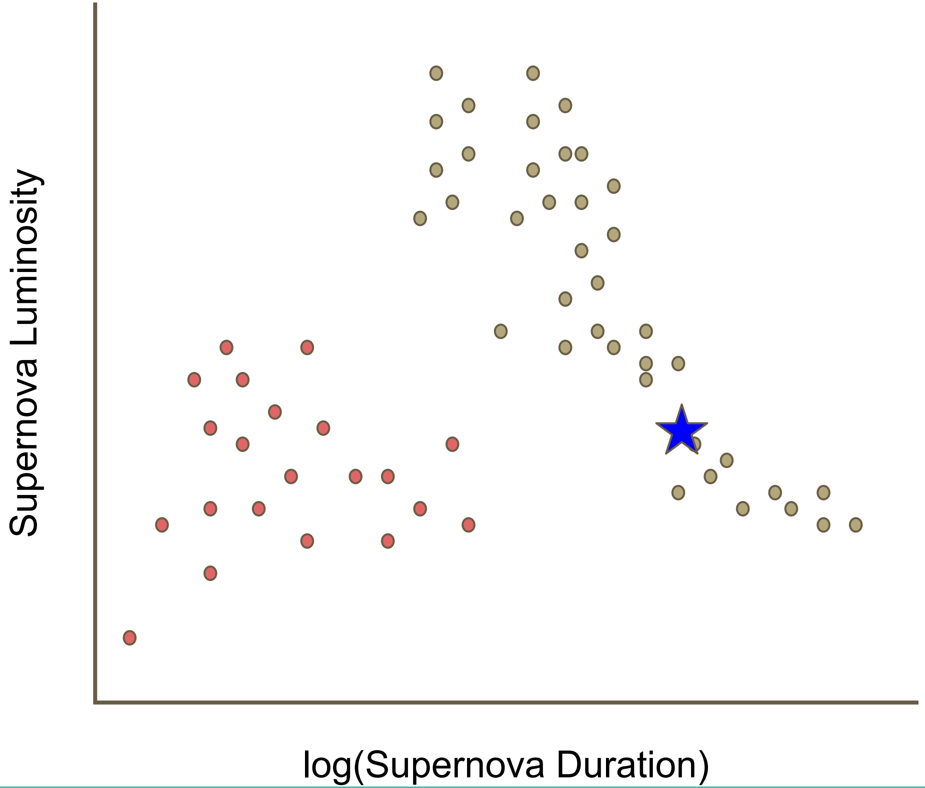
Pro/Con: Not every point will end up in a cluster

Supervised Learning

In supervised learning, we know the “answers” apriori for a training set, and we want to train an algorithm to report some quantity (a classification, a luminosity, etc) given a new observation.

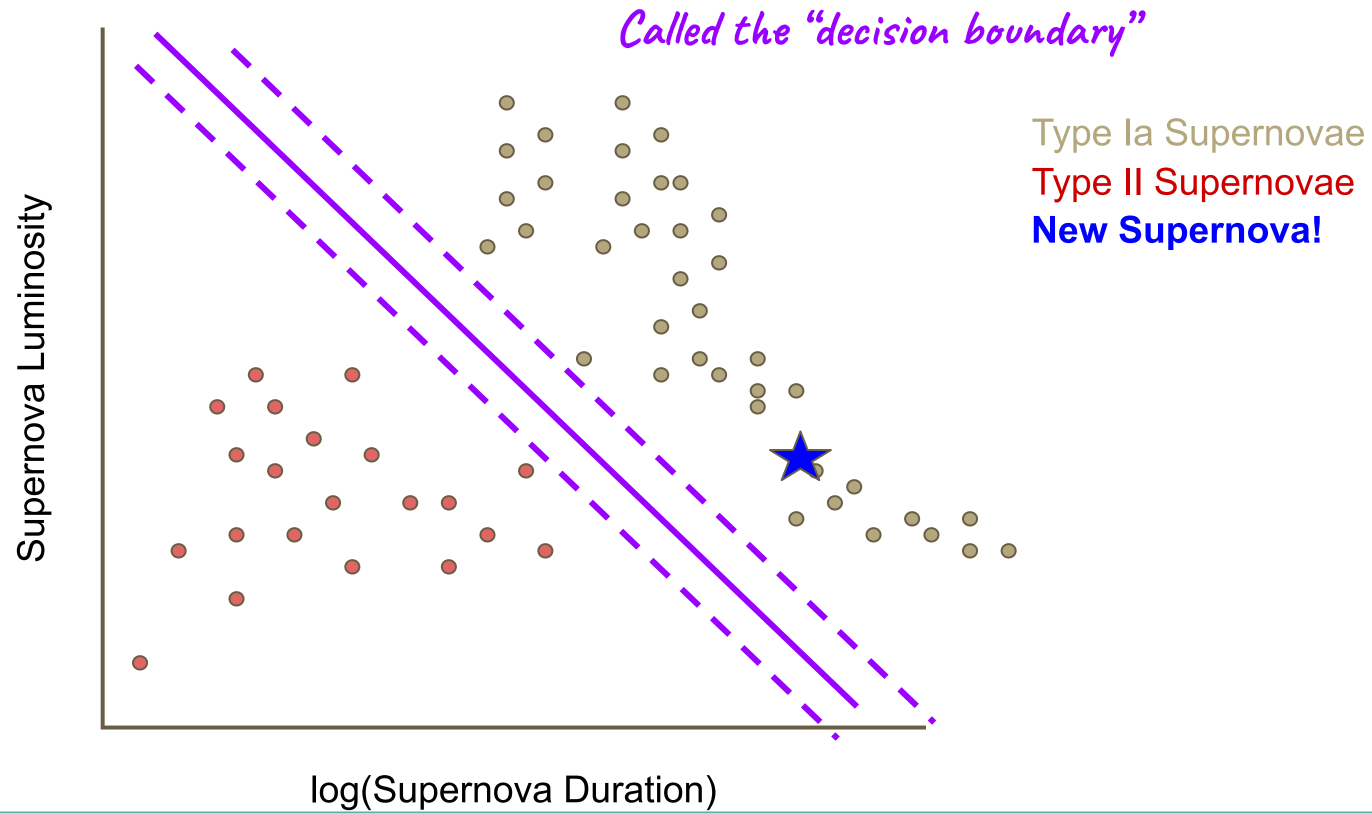
Here we will explore just 2 algorithms: Support Vector Machines and Random Forests. In both cases, we’ll explore **classification**

Classification: Can I build a model to label a new supernova as a Type I vs Type II?

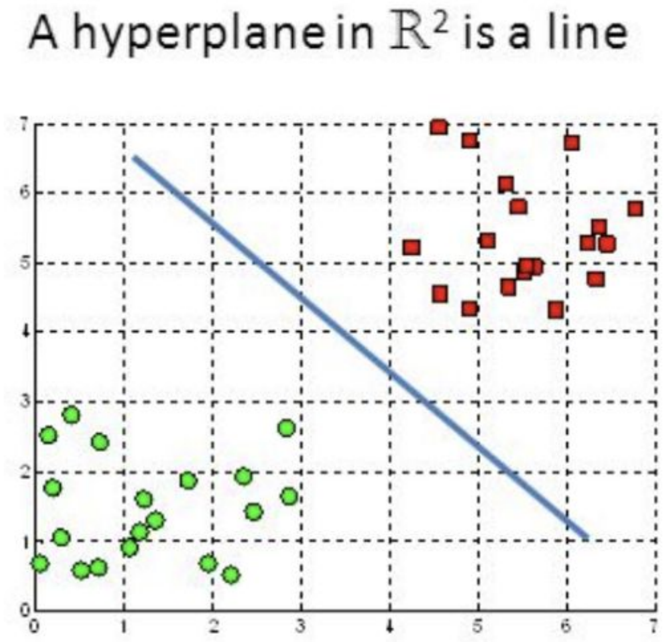


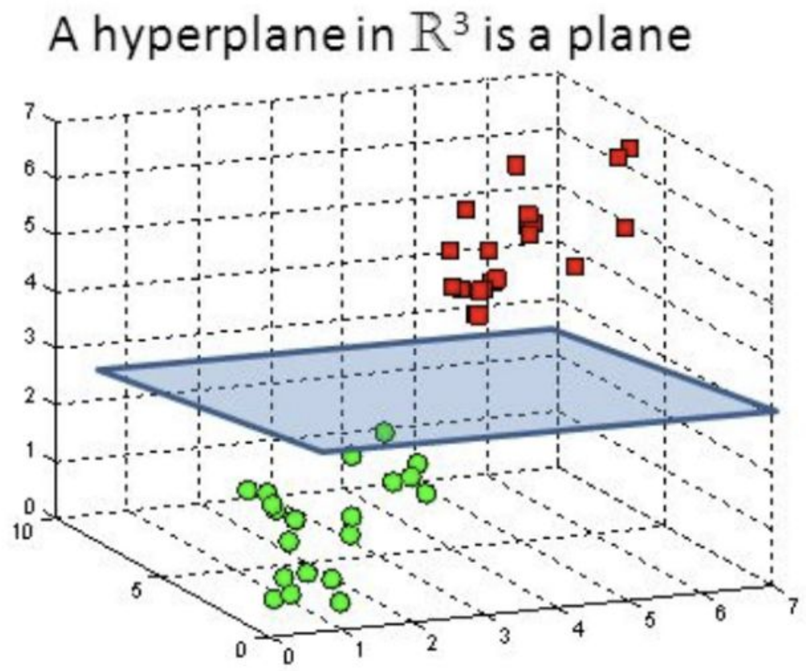
Type Ia Supernovae
Type II Supernovae
New Supernova!

Support Vector Machine: Find the best line to divide the classes



More generally, we find the hyperplane (of N-1 dimensions) for N-dimensional data

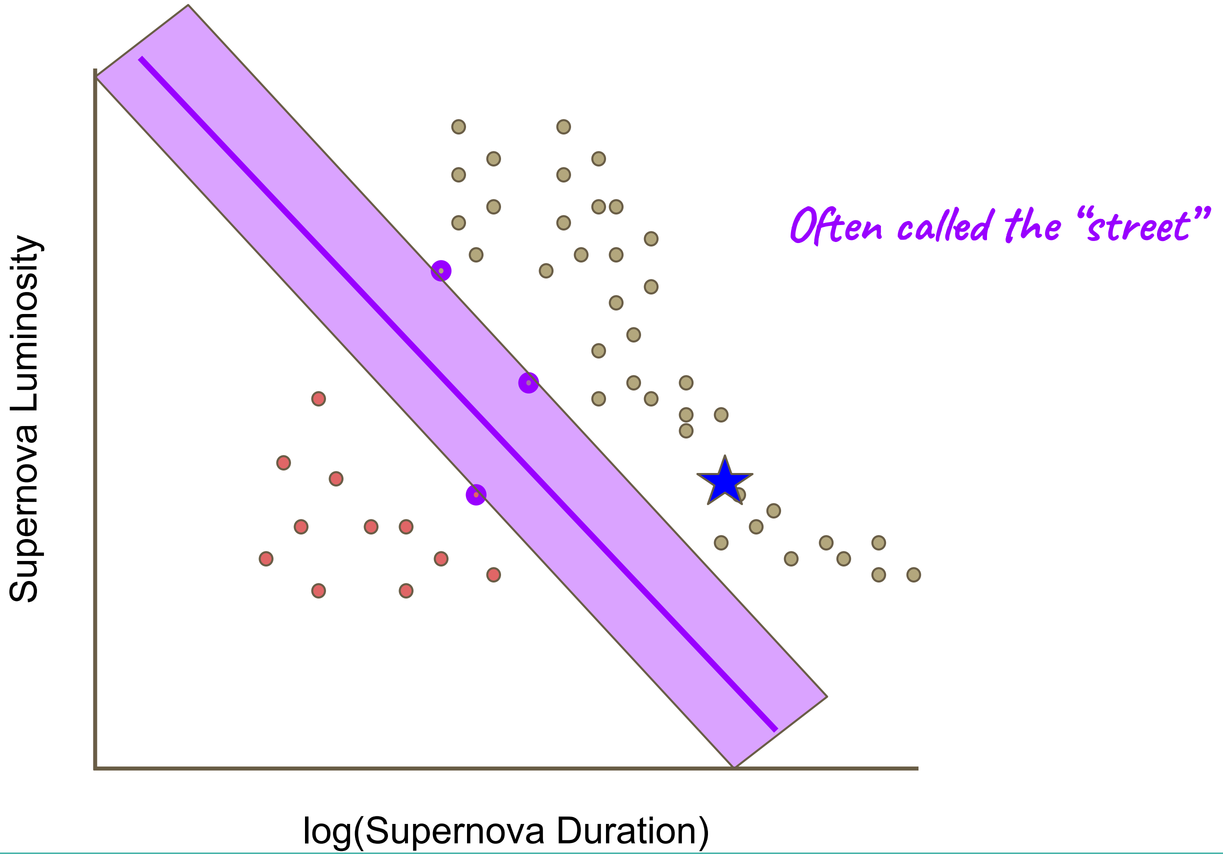




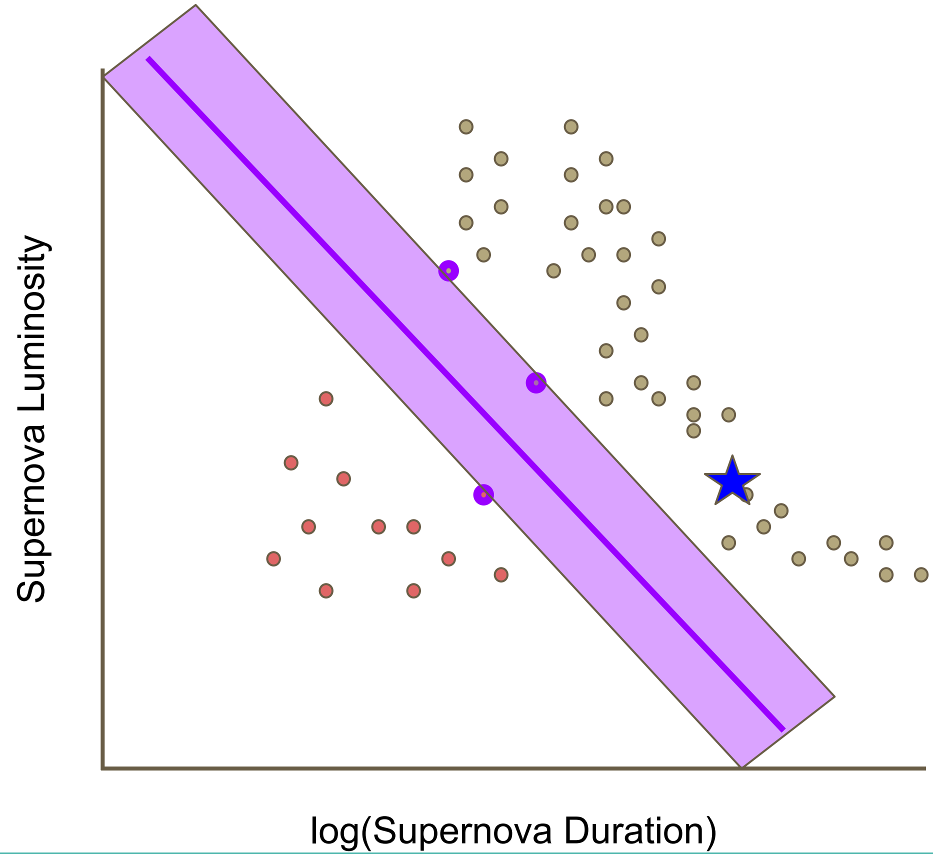
Let’s only think about the 2D case. Let’s define our objective function:

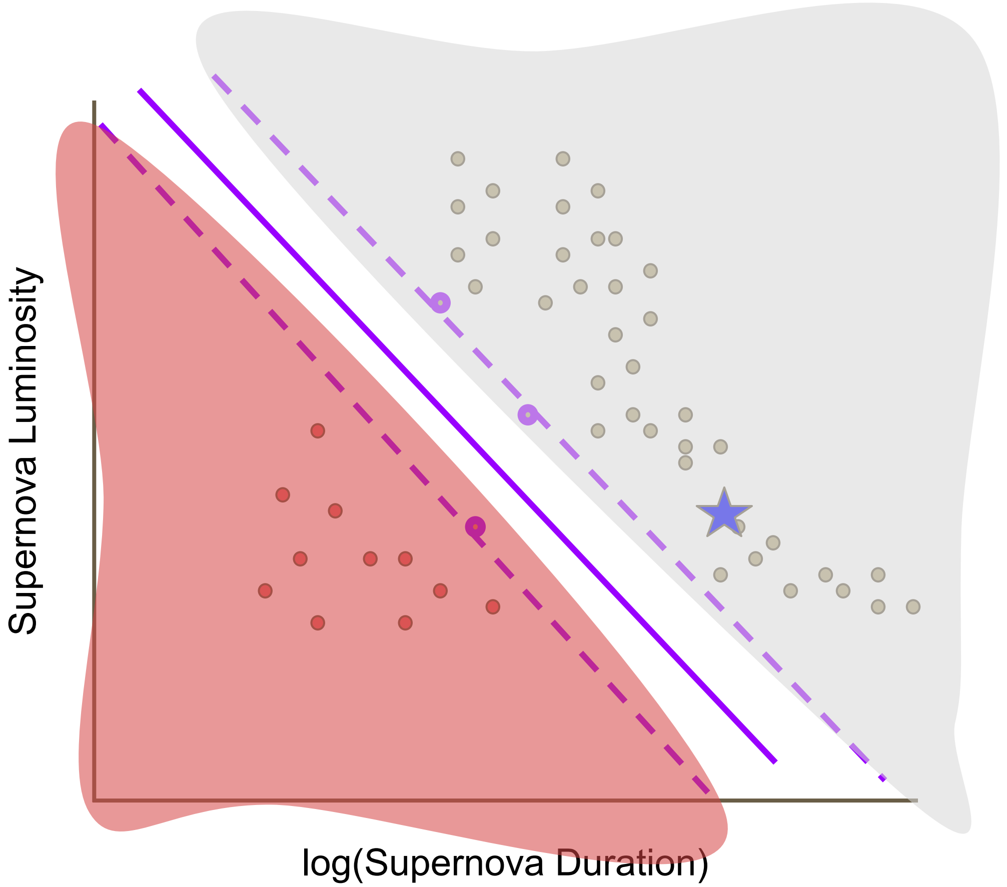
How do we choose the “best” line to divide the classes?

We want to draw our line such that we have the widest separation between our classes.

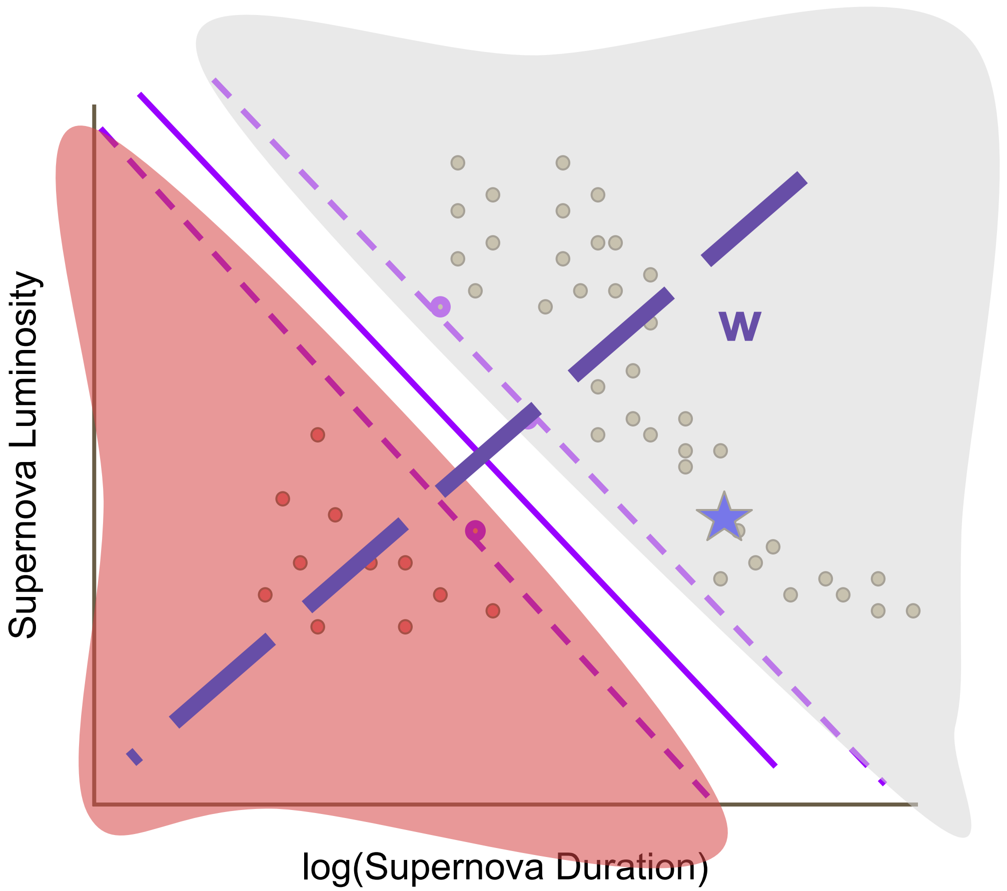


Three points are needed to define two parallel planes which separate the classes. These are called the **support vectors**



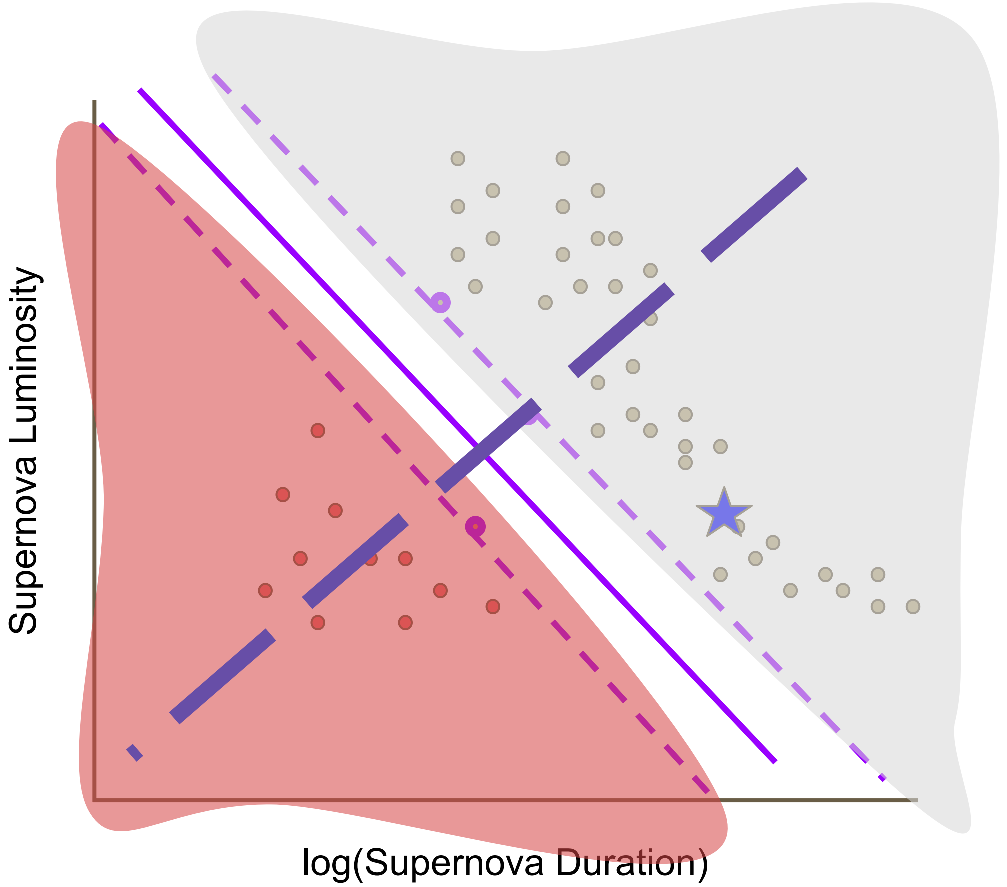


Our class will be determined by which side of the “street” we sit on…



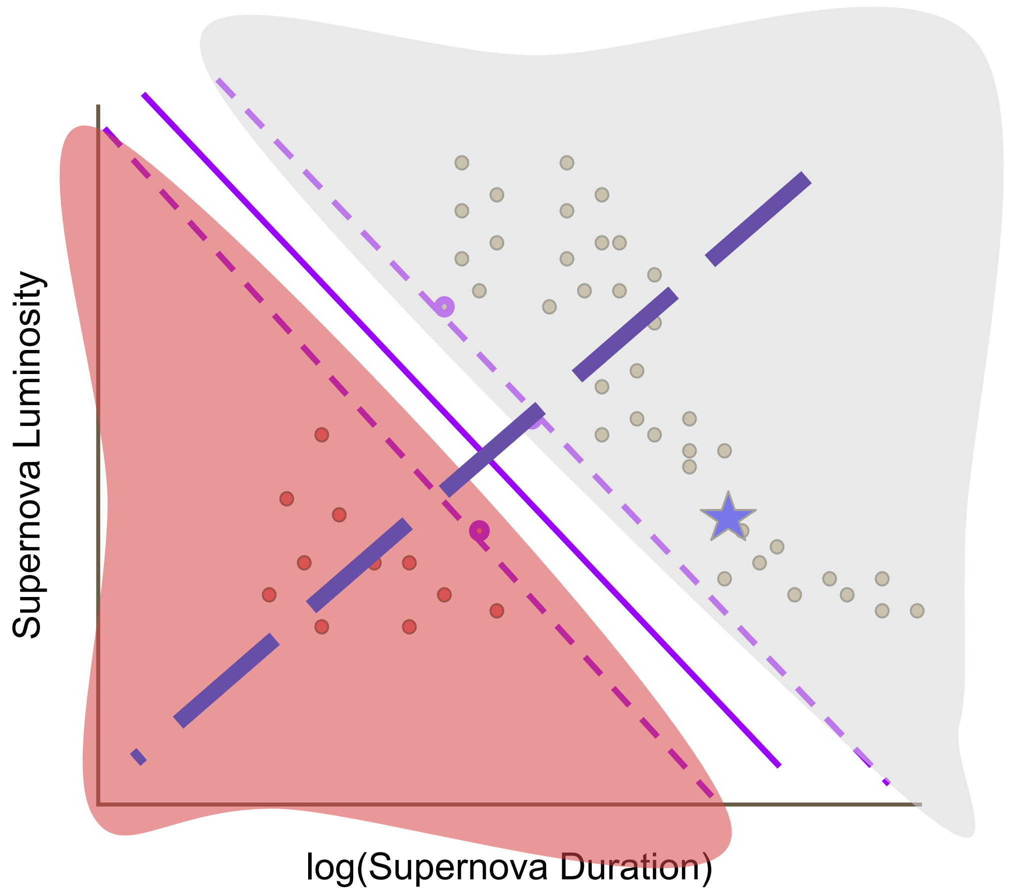
Our class will be determined by which side of the “street” we sit on…

Or mathematically, we can define the perpendicular plane to the street and determine the distance along this vector:



$$\vec{x}_i \cdot \vec{w} + b \geq +1$$ Type Ia

$$\vec{x}_i \cdot \vec{w} + b \leq -1$$ Type II



$$\vec{x_i} \cdot \vec{w} + b \geq +1$$

$$\vec{x_i} \cdot \vec{w} + b \leq -1$$

We want to maximize the width of the street, while subject to the two equality constraints listed above.

Defining our **objective function** will be the job of Lagrangian math

From the detailed math*, three important pieces of information on the objective function emerge:

1. The objective function only depends on the support vectors - i.e., the observational points which lie on the very edge of my street

*See notes or reading for detailed math

From the detailed math*, three important pieces of information on the objective function emerge:

1. The objective function only depends on the support vectors - i.e., the observational points which lie on the very edge of my street
2. The objective function will be differentiable, so we can optimize via gradient descent

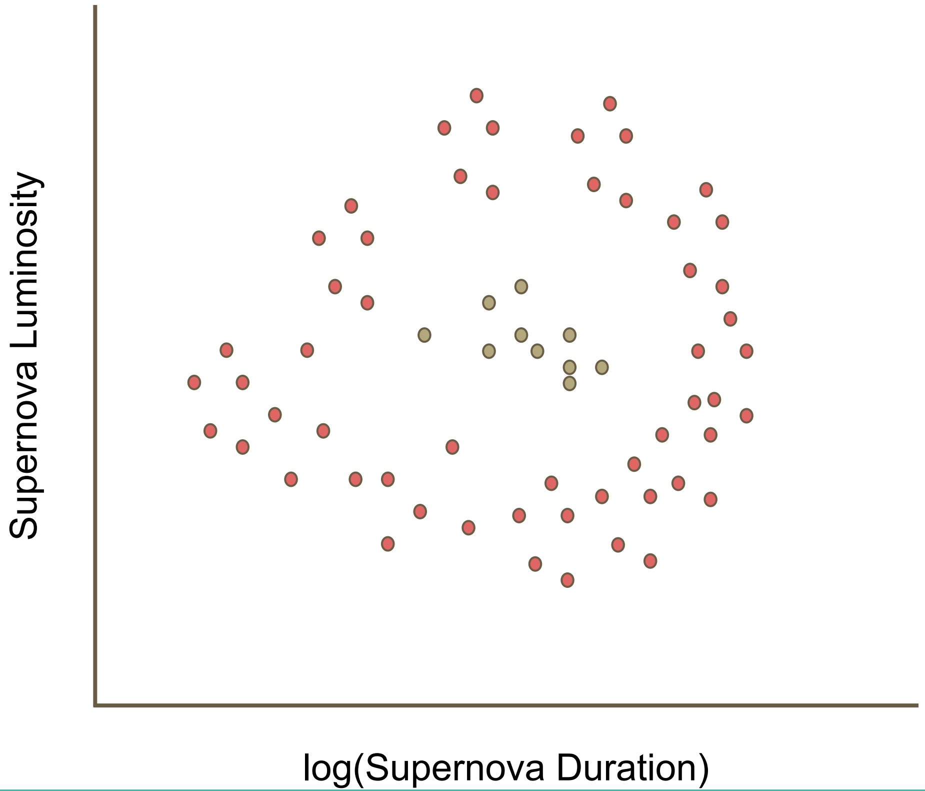
See notes or reading for detailed math

From the detailed math*, three important pieces of information on the objective function emerge:

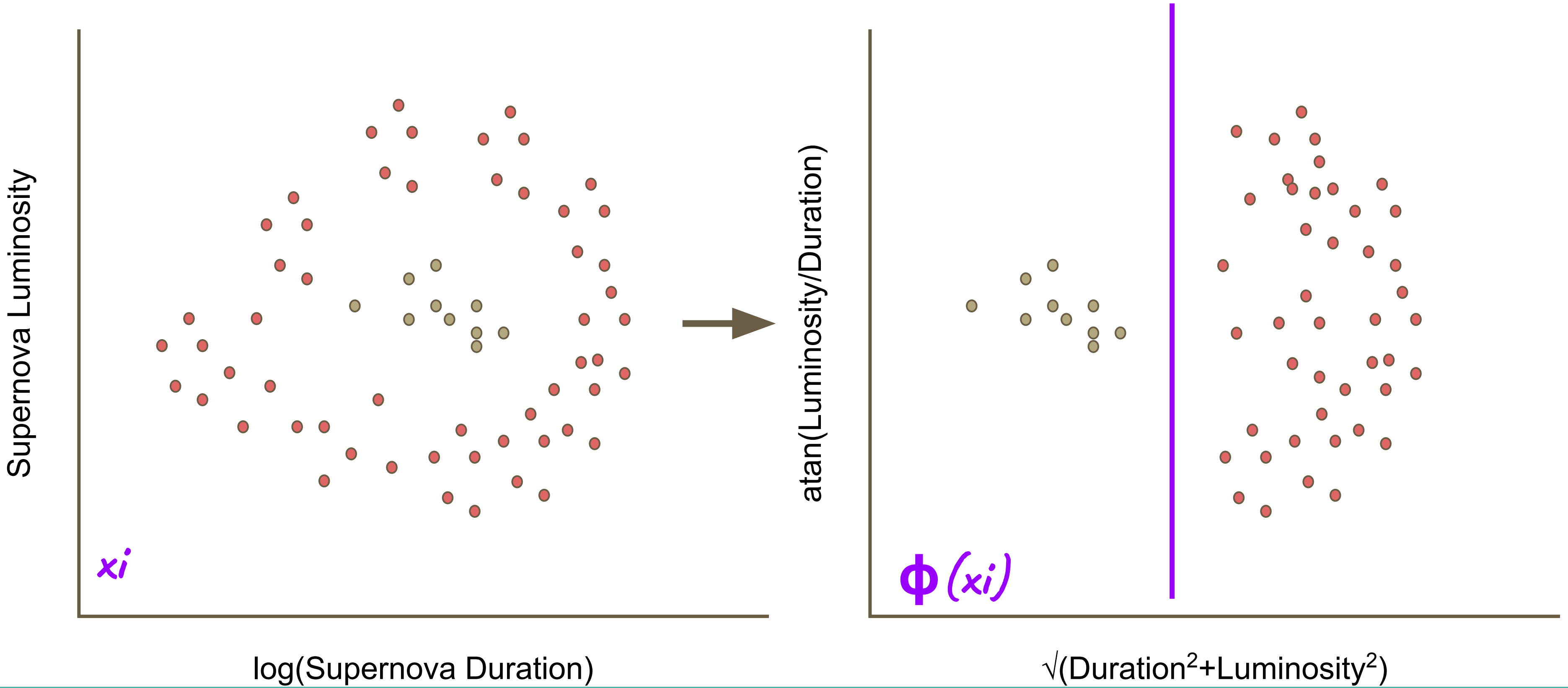
1. The objective function only depends on the support vectors - i.e., the observational points which lie on the very edge of my street
2. The objective function will be differentiable, so we can optimize via gradient descent
3. The objective function only depends on **the similarity (dot product) between pairs of support vectors, $\mathbf{x}_i \cdot \mathbf{x}_j$**

See notes or reading for detailed math

What if my data cannot be separated by a line (linearly separable)?



In previous classes, we learned about mapping one feature space into another latent space



But, I don’t actually need to know what $\Phi(x)$ is, because my objective function only depends on $\Phi(xi) \cdot \Phi(xj)$

But, I don't actually need to know what Φ(x) is, because my objective function only depends on Φ(xi) ·Φ(xj) ...

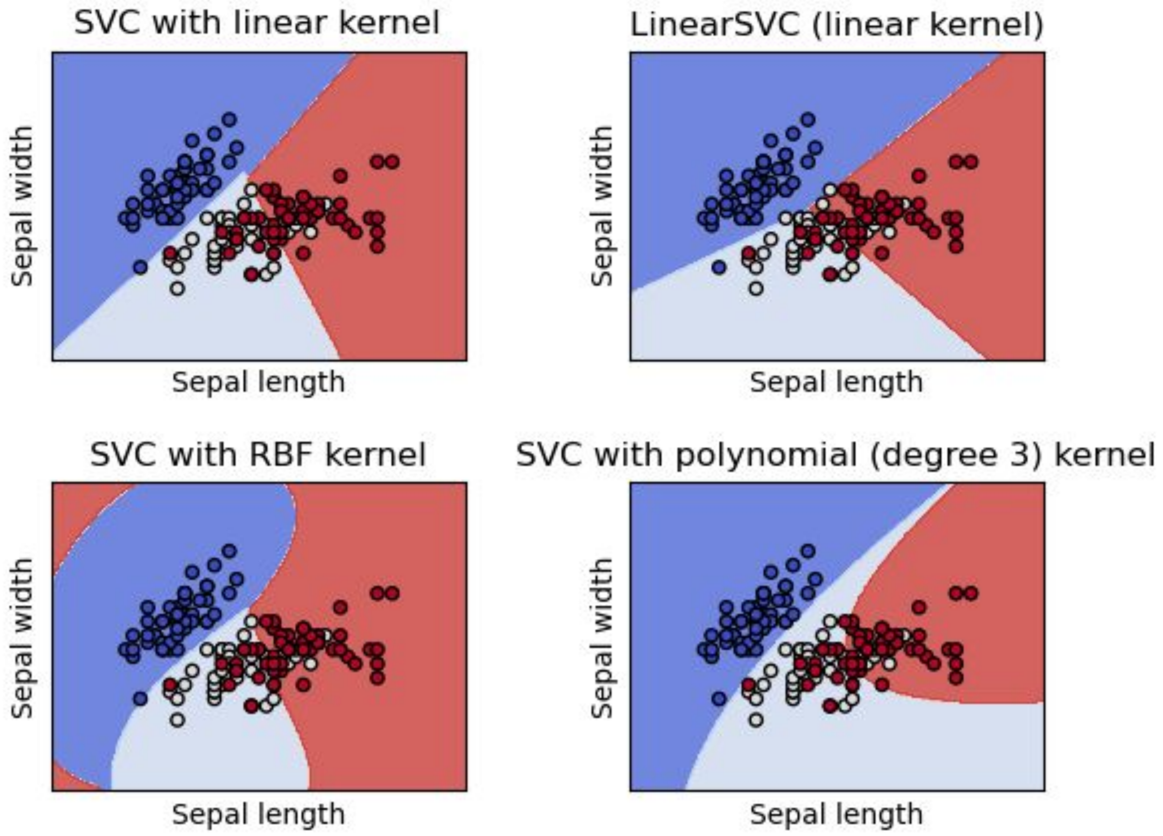
So let us define a function, which we'll call a Kernel, which defines dot products in this new space:

$$K(x_i, x_j) \equiv \phi(x_i) \cdot \phi(x_j)$$

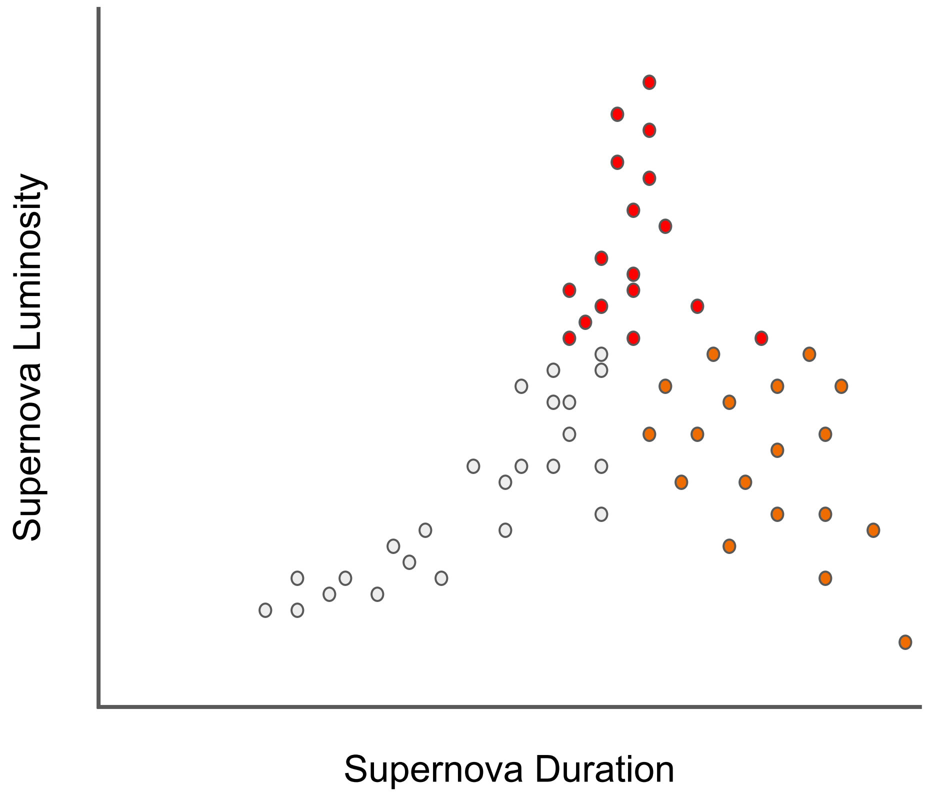
This is called the "kernel trick!"

One popular example of a kernel function:

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$$


SVC with linear kernel
LinearSVC (linear kernel)
SVC with RBF kernel
SVC with polynomial (degree 3) kernel
Sepal width
Sepal length

Decision Trees as a Supervised Classification Method

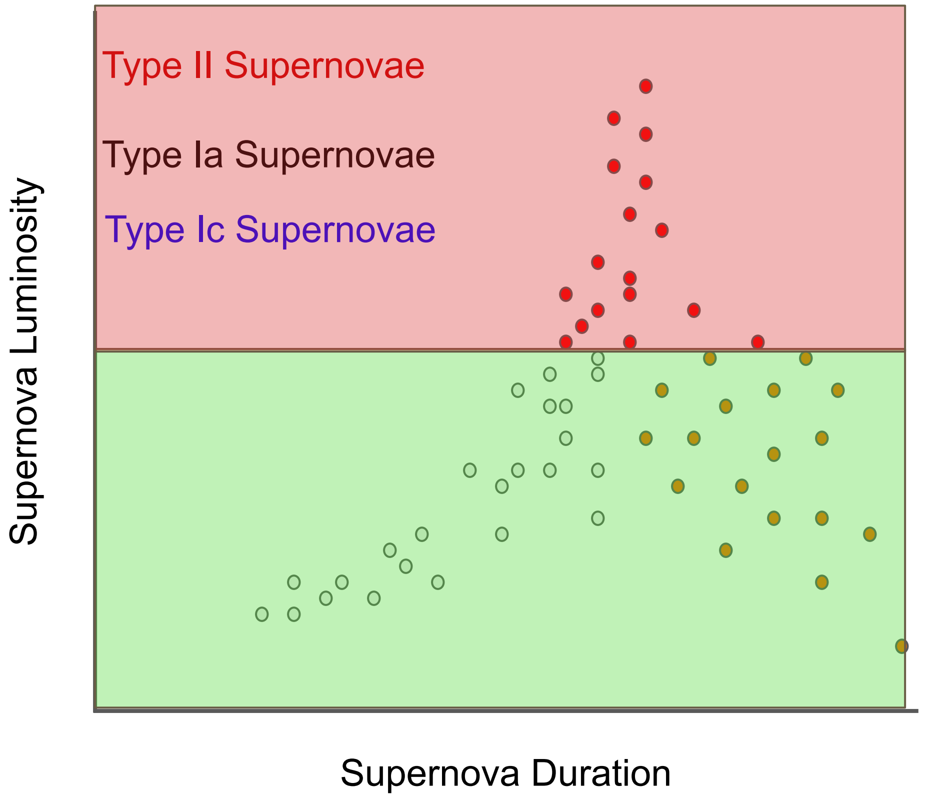


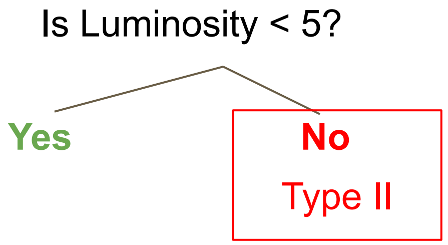
Type II Supernovae

Type Ia Supernovae

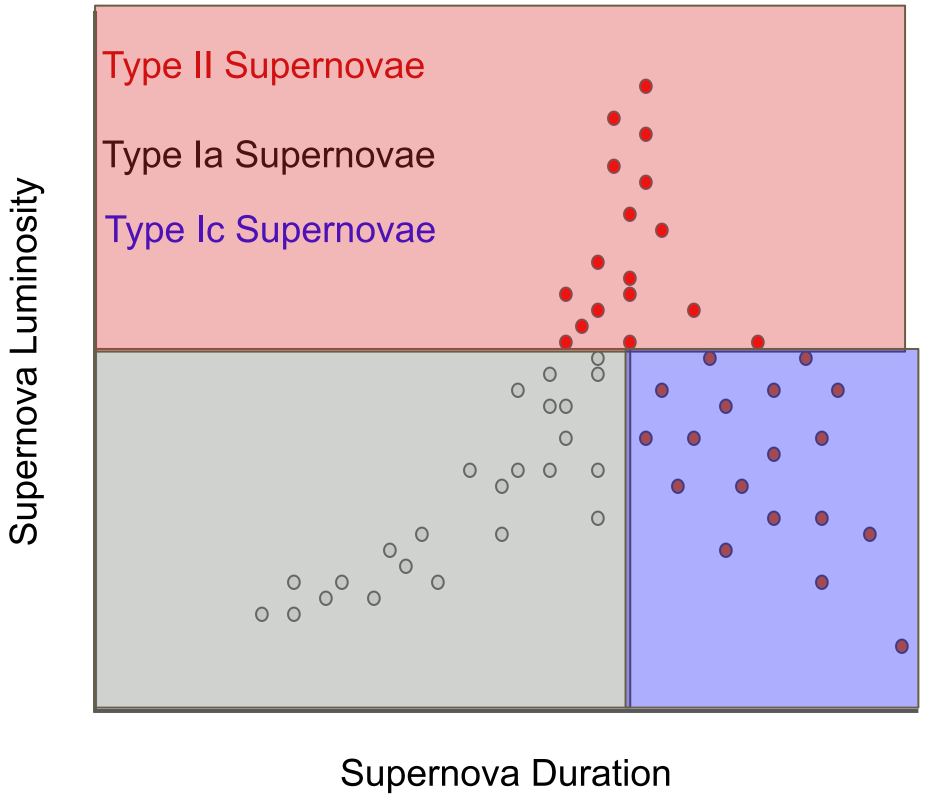
Type Ic Supernovae

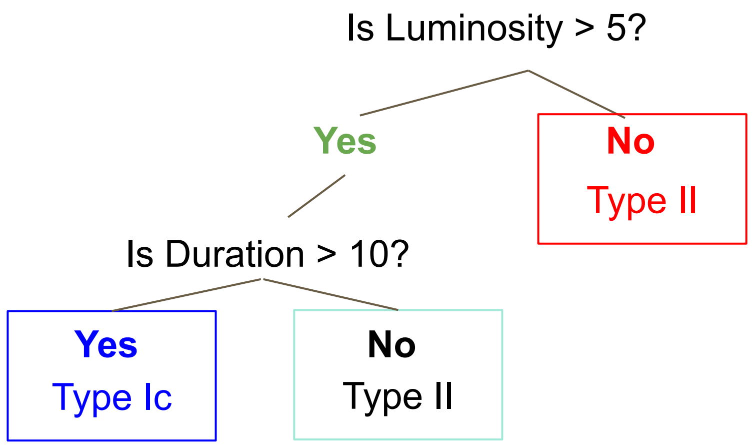
A decision tree makes a series of binary cuts to sort data



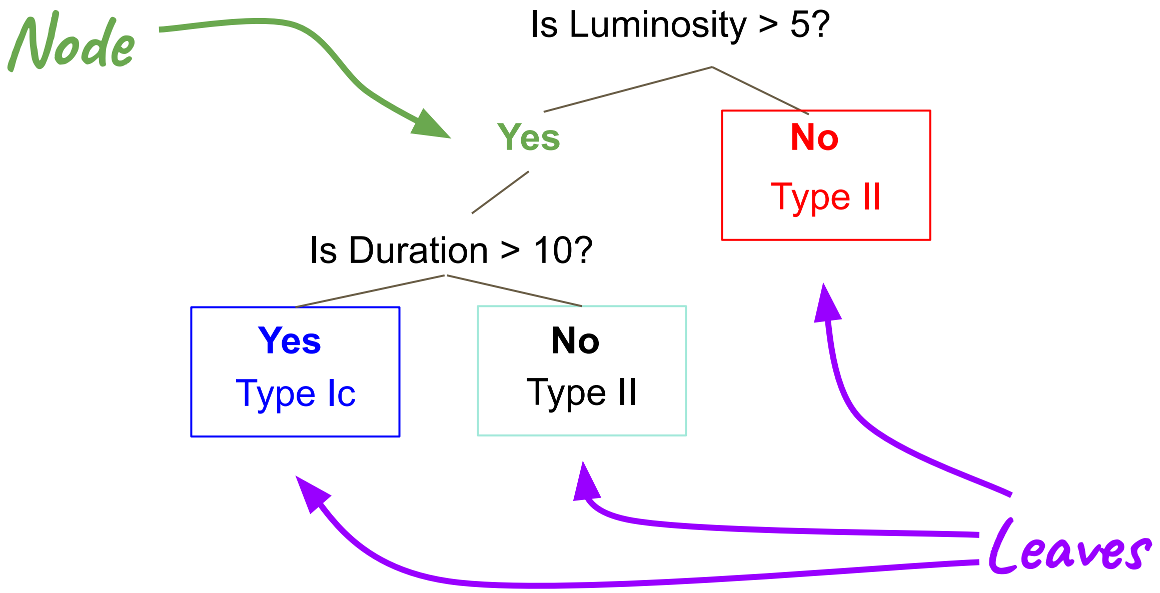


A decision tree makes a series of binary cuts to sort data

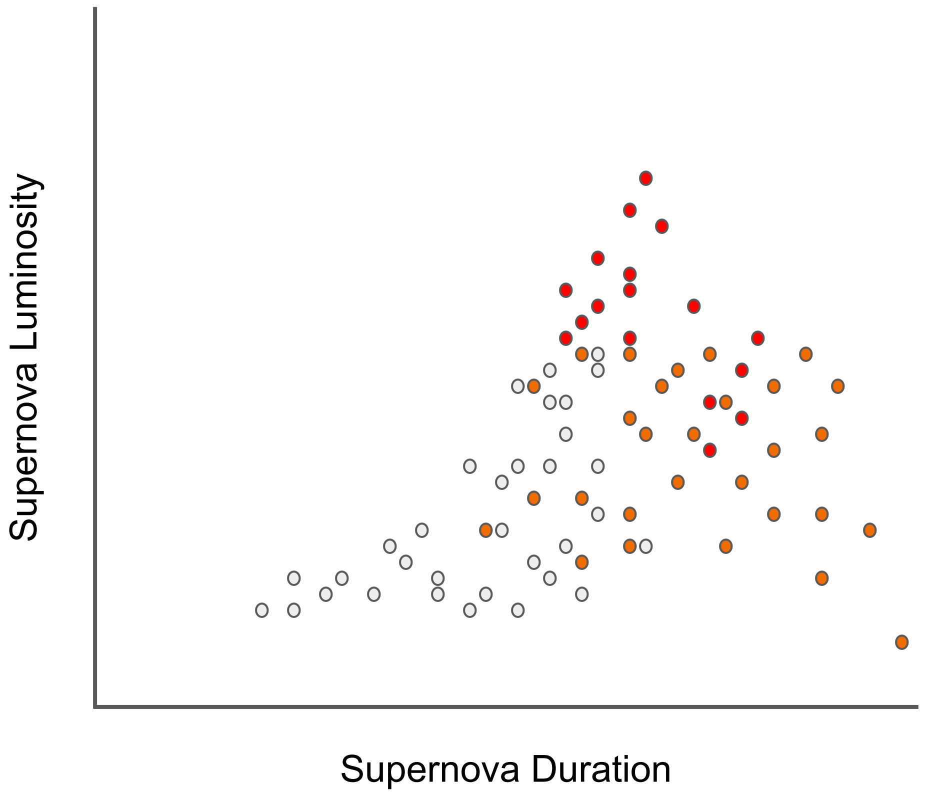




Some terminology



That was pretty easy! What if our classes were more mixed?

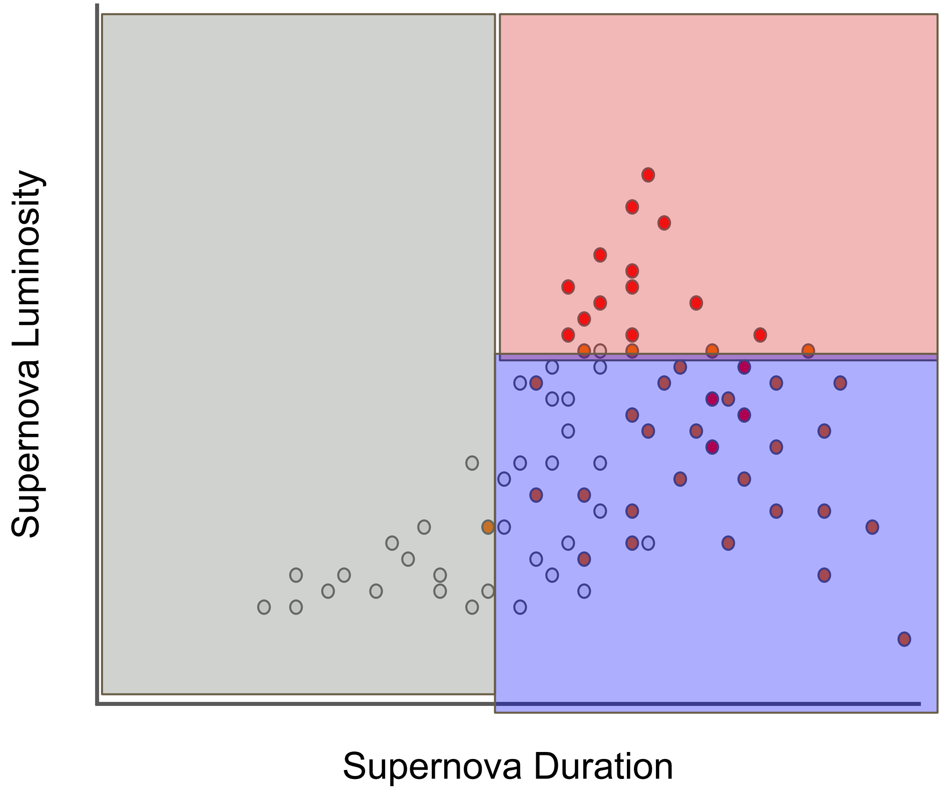


Type II Supernovae

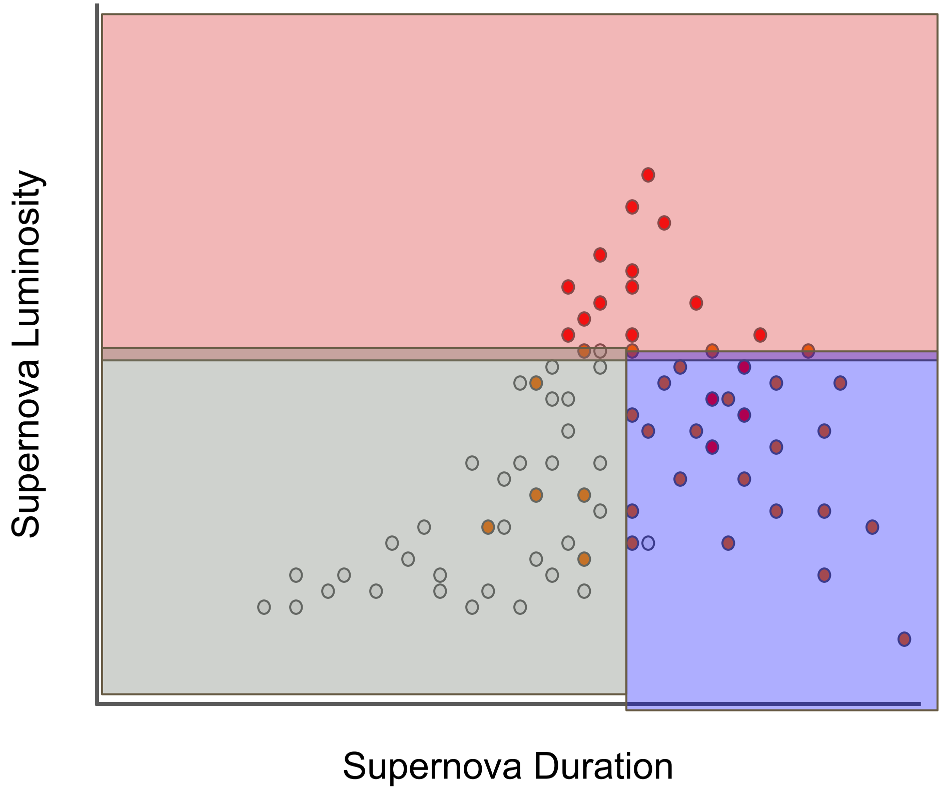
Type Ia Supernovae

Type Ic Supernovae

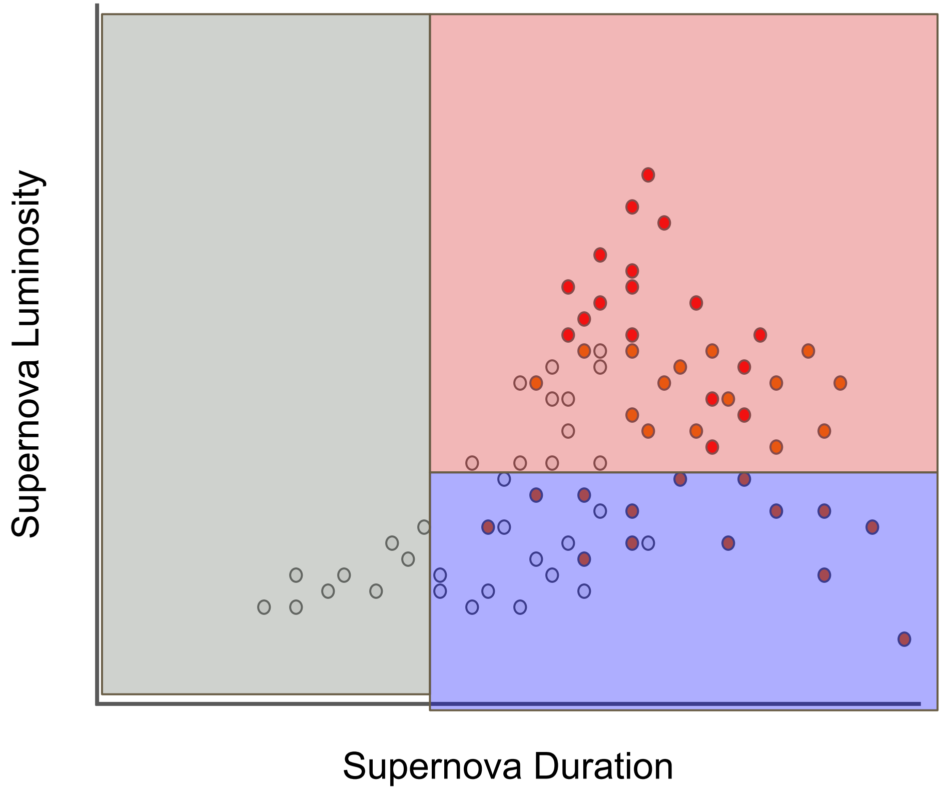
That was pretty easy! What if our classes were more mixed?



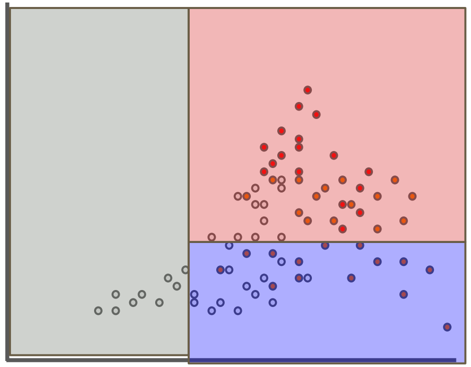
That was pretty easy! What if our classes were more mixed?

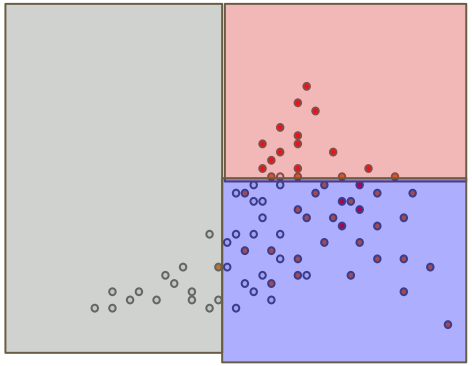


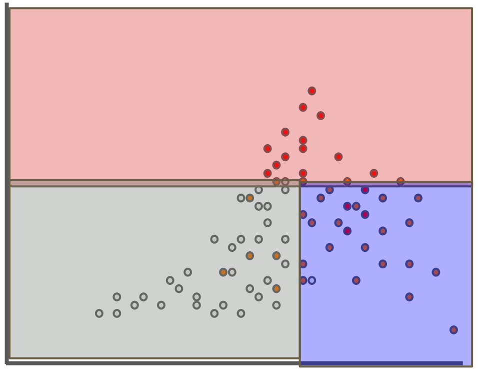
That was pretty easy! What if our classes were more mixed?



A random forest combines these decision trees to decide on a final classification





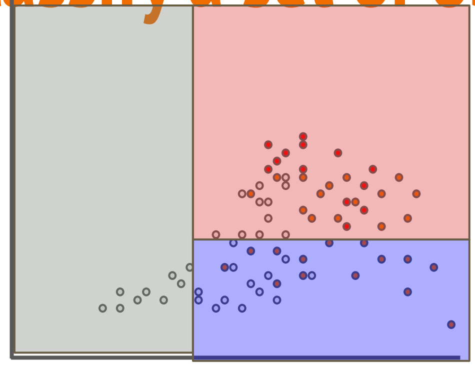


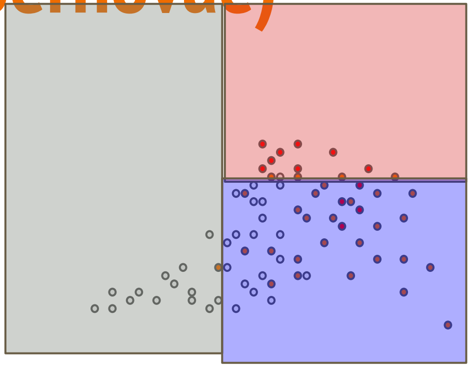
M(odel):

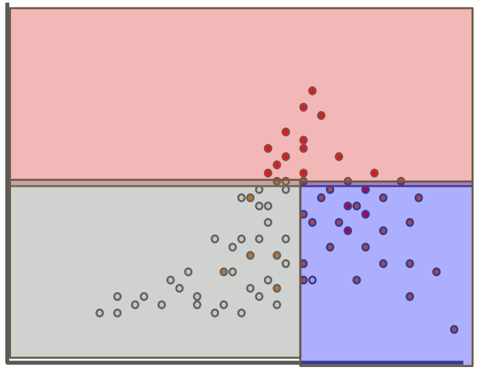
O(bjective function):

O(ptimization):

M(odel): A series of set of decision trees which attempt to classify a set of objects (like supernovae)



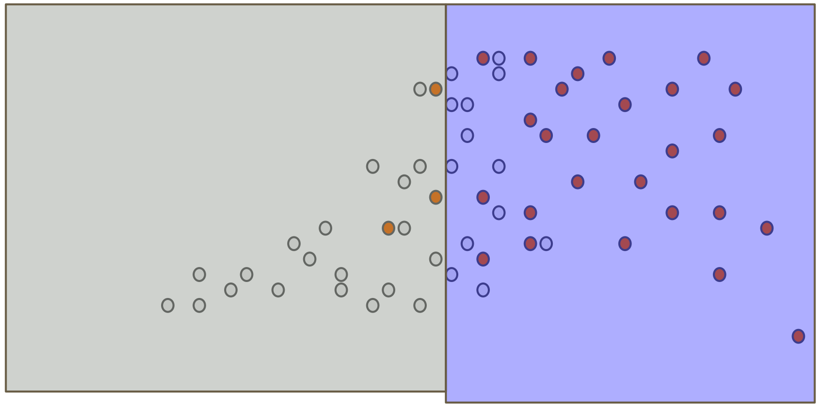




Let's zoom into one tree. How do we choose decision boundaries?

$$1 - \sum_{k}^{K} p_k^2$$

O(bjective function): Gini Impurity

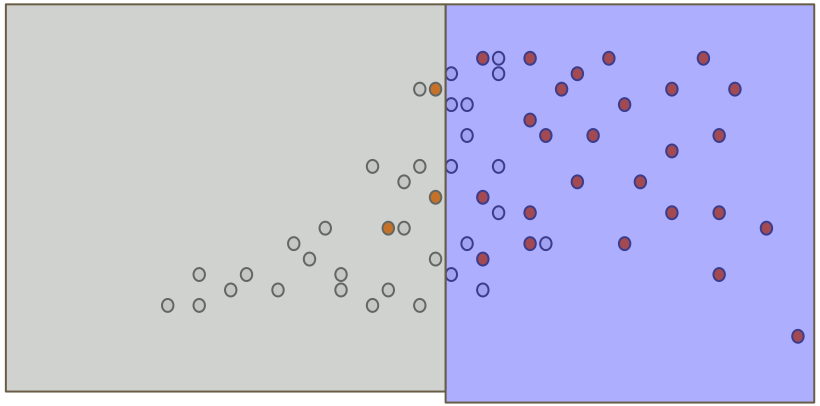


Total SNe*: 23
Total Type Ia: 20
Total Type Ic: 3

Total SNe*: 37
Total Type Ia: 13
Total Type Ic: 24

*SNe = supernovae

O(bjective function): Gini Impurity



Total SNe*: 23
Total Type Ia: 20
Total Type Ic: 3

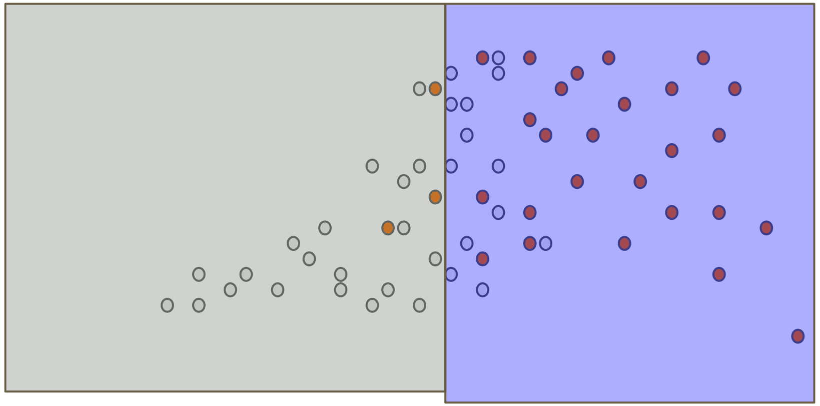
p(Ia) = 20/23 = 0.87
p(Ic) = 3/23 = 0.13

Total SNe*: 37
Total Type Ia: 13
Total Type Ic: 24

p(Ia) = 13/37 = 0.35
p(Ic) = 24/24 = 0.65

*SNe = supernovae

O(bjective function): Gini Impurities (for each leaf)



Total SNe*: 23
Total Type Ia: 20
Total Type Ic: 3

p(Ia) = 20/23 = 0.87
p(Ic) = 3/23 = 0.13

Total SNe*: 37
Total Type Ia: 13
Total Type Ic: 24

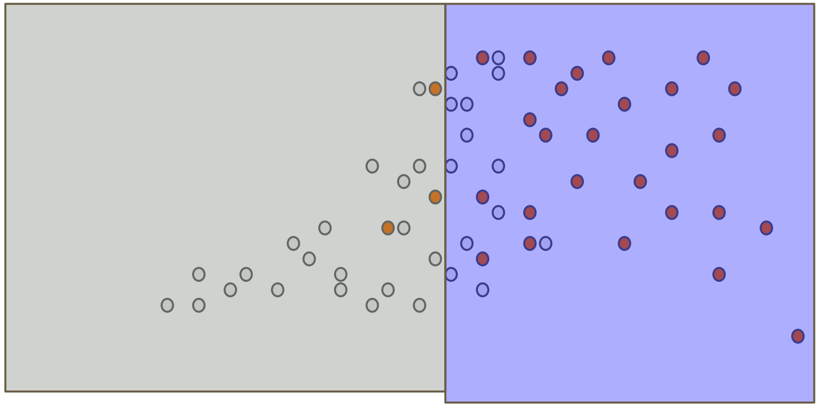
p(Ia) = 13/37 = 0.35
p(Ic) = 24/24 = 0.65

$$1 - (0.87^2 + 0.13^2) = 0.23$$

$$1 - (0.35^2 + 0.65^2) = 0.46$$

*SNe = supernovae

O(bjective function): Weighted Mean Gini Impurities



$\boxed{23/60 * 0.23}$ + $\boxed{37/60 * 0.46}$

= 0.37!

Total SNe*: **23**
Total Type Ia: 20
Total Type Ic: 3

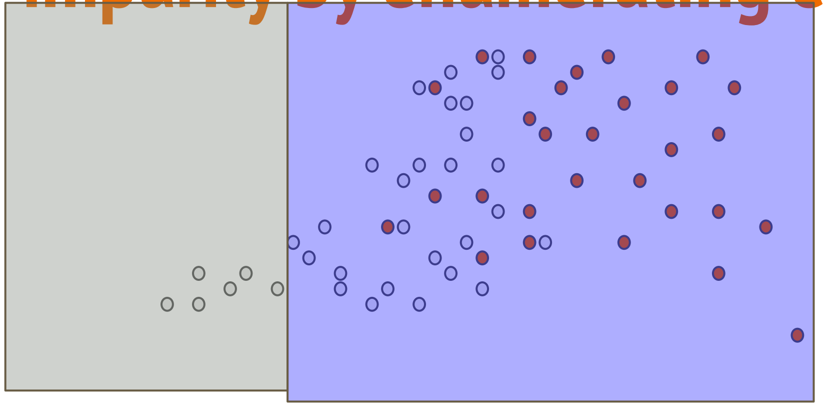
p(Ia) = 20/23 = 0.87
p(Ic) = 3/23 = 0.13

Total SNe*: **37**
Total Type Ia: 13
Total Type Ic: 24

p(Ia) = 13/37 = 0.35
p(Ic) = 24/24 = 0.65

*SNe = supernovae

A random forest minimizes this weight mean Gini impurity by enumerating every option



$$\boxed{6/60 * 0} + \boxed{54/60 * 0.75}$$

Total SNe*: **6**
Total Type Ia: 6
Total Type Ic: 0

p(Ia) = 6/6 = 1
p(Ic) = 0/6 = 0

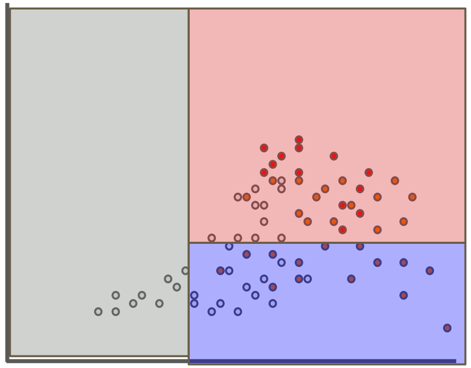
Total SNe*: **54**
Total Type Ia: 27
Total Type Ic: 27

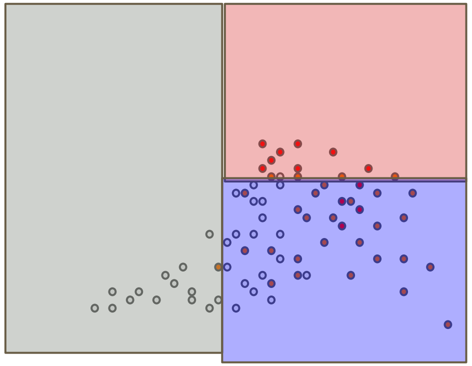
p(Ia) = 13/37 = 0.5
p(Ic) = 24/24 = 0.5

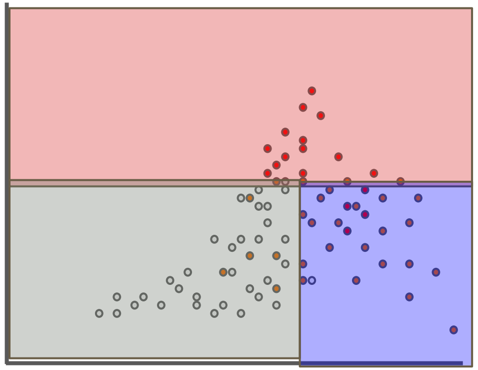
= 0.68!

*SNe = supernovae

Each tree works on a random subset of the data







Conclusions

Supervised/Unsupervised Learning are families of statistical problems which can aid in: understanding relationships between variables, dimensionality reduction, clustering, classification and regression.

Today we covered a wide variety of topics, from k-means to random forests.

Questions?