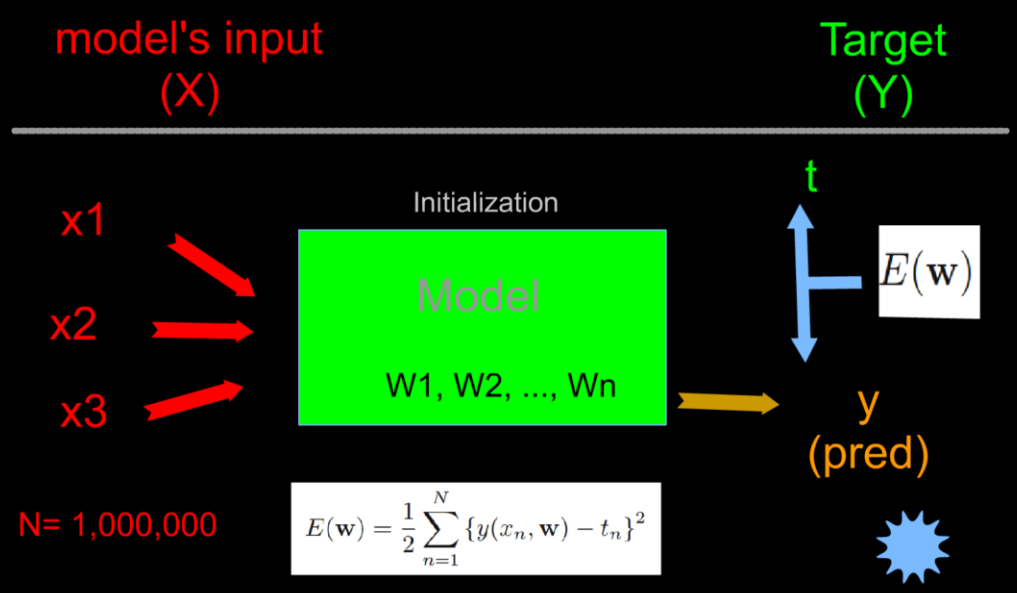
Optimization method:

1-When we can not train a model (finding its parameters) in a closed form.
2-Then we might want to get closer and closer to the actual values (the target) by tunning a model's parameters, step by step and gradually.
3-We start by choosing the parameters randomly (initialization).
4-Then we can tune the parameters by an algorithm (Gradient Descent) using our training set to get better and better results (less errors).
5-The training is usually done in different steps (epochs)
6-If we satisfy, then we can stop training.

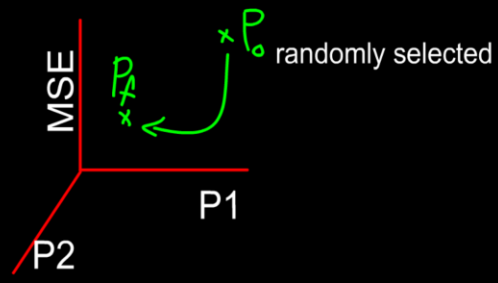
Training steps

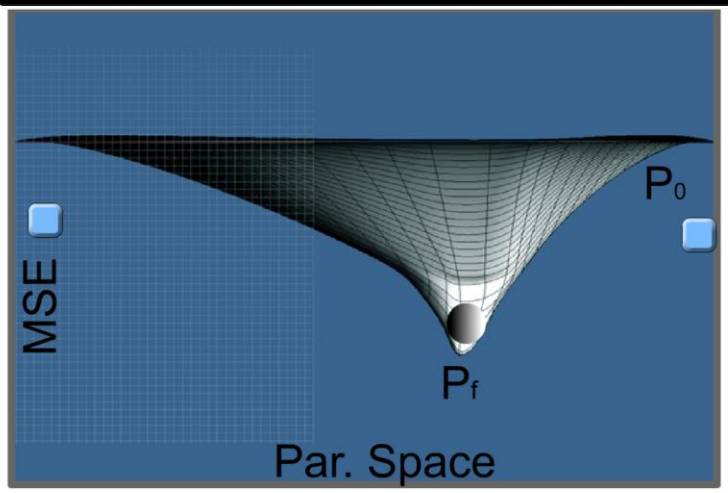


One iteration:
batch process – min batch - stochastic

The difference between Y and PY values can be measured by a performance function that should be minimized. (Mean Square Error function, MSE)

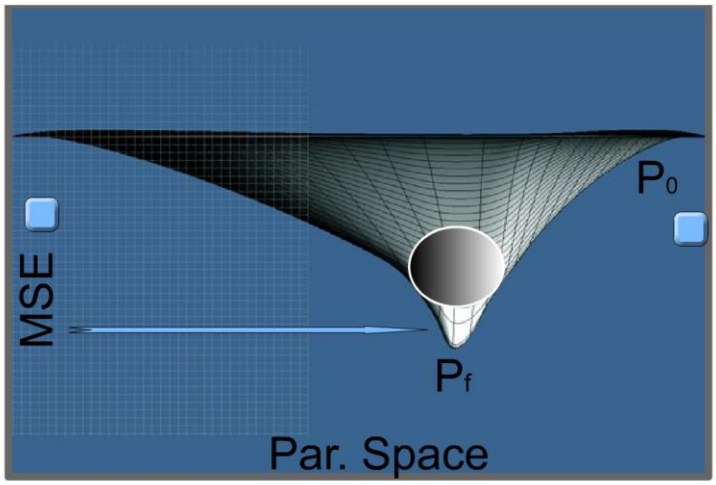
This is a function of model's parameters:
MSE=*f*(P1,P2,...Pn)



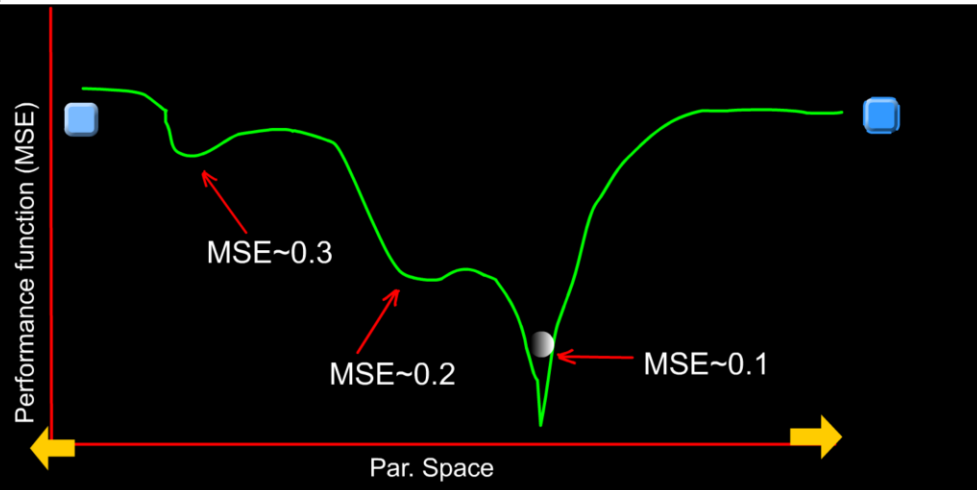


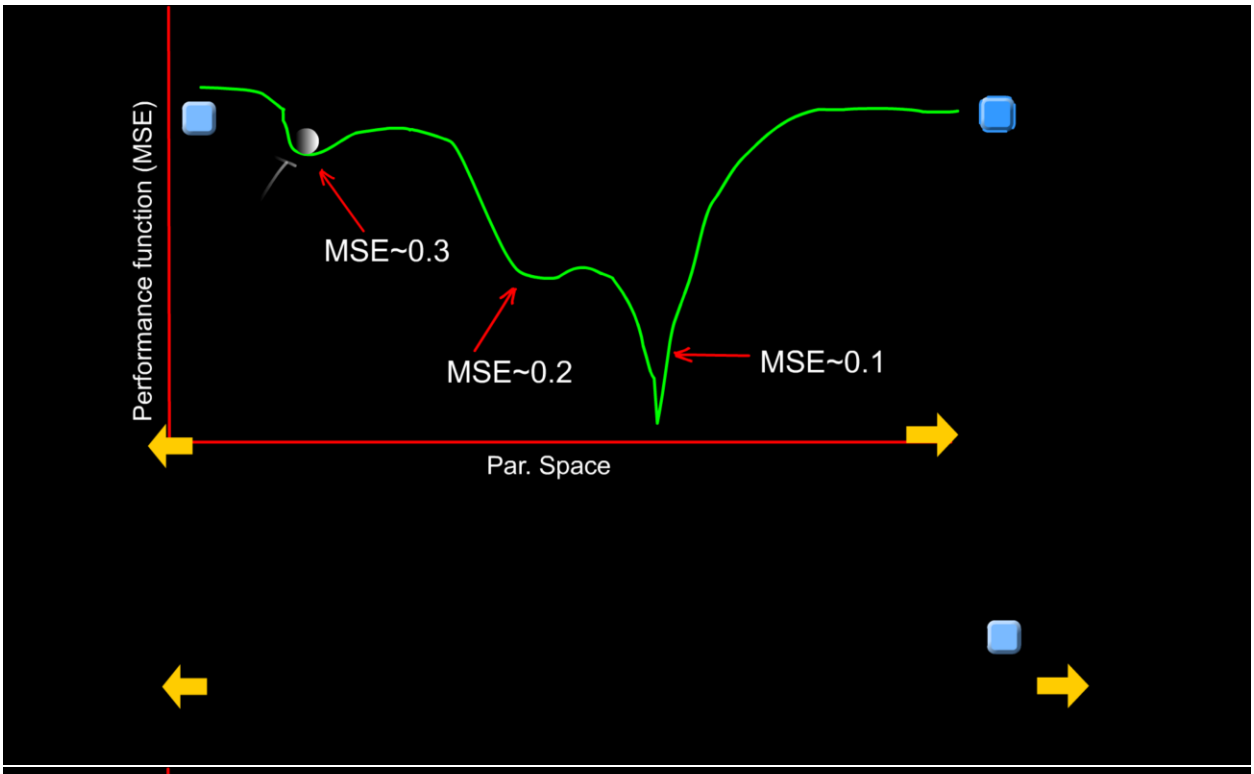
6 iterations

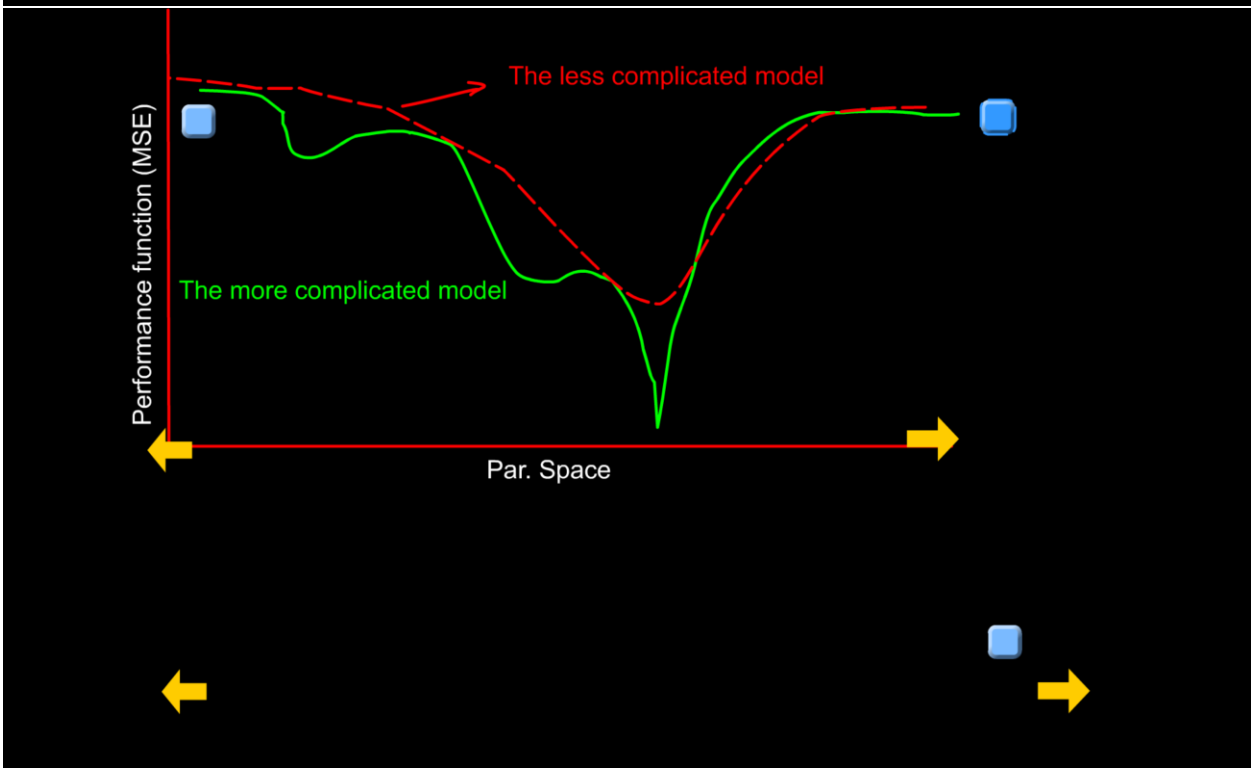
1-We can test all parameters in the model to find best minmum, Pf. (time issue)
2- Or, we can use an algorithm to find Pf in a shorter time.
3-Ones we find the model parameters we can fix them and use the model.
5- When the paramerts are found it is said that the machine is trained
4-We should validate our model with an independent data-set.

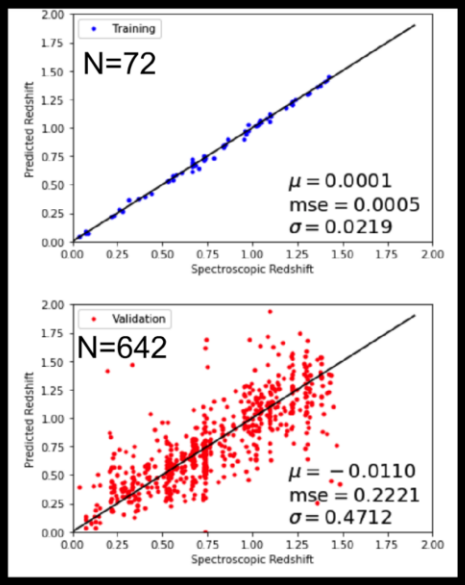


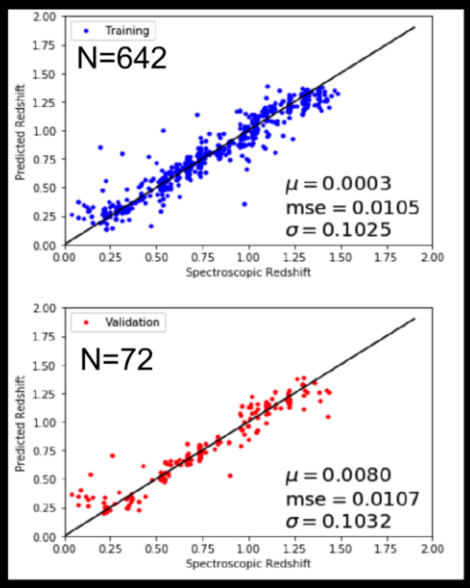
1-We can test all parameters in the model to find best minmum, Pf. (time issue)
2- Or, we can use an algorithm to find Pf in a shorter time.
3-Ones we find the model parameters we can fix them and use the model.
5- When the paramerts are found it is said that the machine is trained
4-We should validate our model with an independent data-set.




Performance function (MSE)
MSE~0.3
MSE~0.2
MSE~0.1
Par. Space

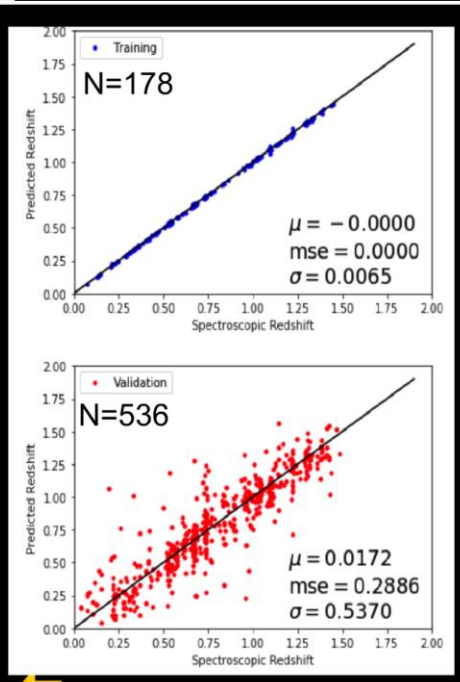

Performance function (MSE)
The less complicated model
The more complicated model
Par. Space

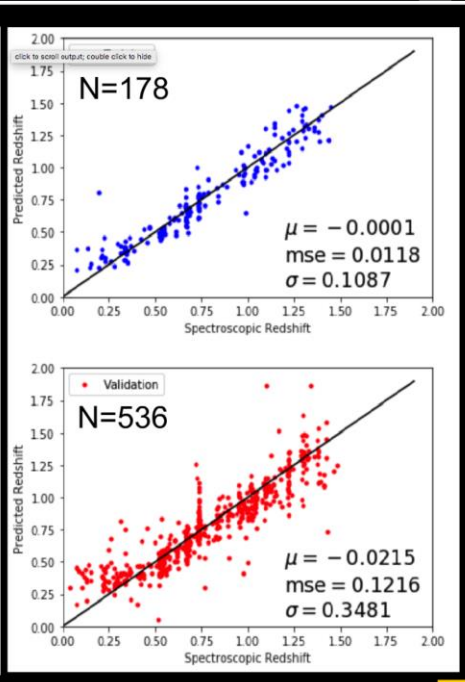




A powerful model

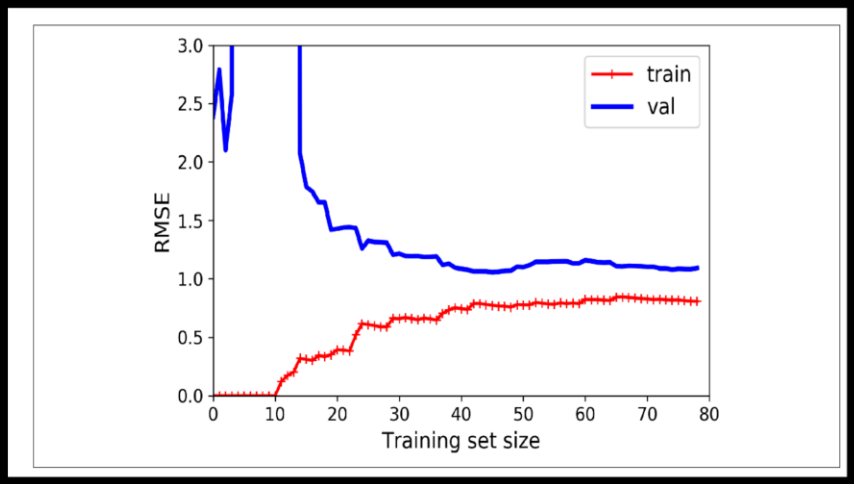
The same model





A powerful model

the less powerful model



$$\underset{\text{Posterior}}{P(w|M)} = \frac{\overset{\text{likelihood}}{P(M|w)} \times \overset{\text{prior}}{P(w)}}{\underset{\text{evidence}}{P(M)}}$$

Bayes' theorem

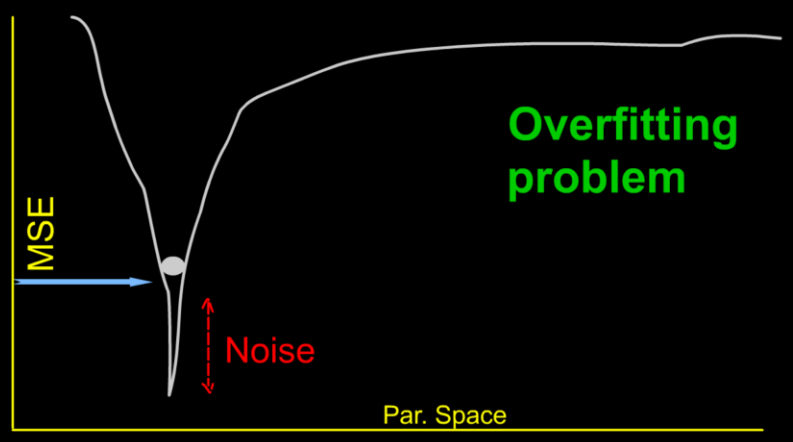
Regularization
Bayesian Regression

$$\begin{aligned} -\ln p(\mathbf{w}|x_{1:N}, y_{1:N}) &= -\sum_i \ln(p(y_i|x_i, \mathbf{w})) - \ln(p(\mathbf{w})) + \ln(p(y_{1:N}|x_{1:N})) \\ &= \frac{1}{2\sigma^2}\sum_i (y_i - f(x_i))^2 + \frac{\alpha}{2}||\mathbf{w}||^2 + \text{constants} \end{aligned}$$

```
x_t, x_v, y_t, y_v = train_test_split(inp, tar, test_size=0.25)
#  reg = linear_model.LinearRegression()
# reg = linear_model.Ridge(alpha=1)

reg = linear_model.Lasso(alpha=1.0, fit_intercept=True, normalize=False,
precompute=False, copy_X=True, max_iter=1000, tol=0.0001)

pred_y_v= reg.predict(x_v)  # photometric redshift for validation set
pred_y_t= reg.predict(x_t)    # photometric redshift for training set
```

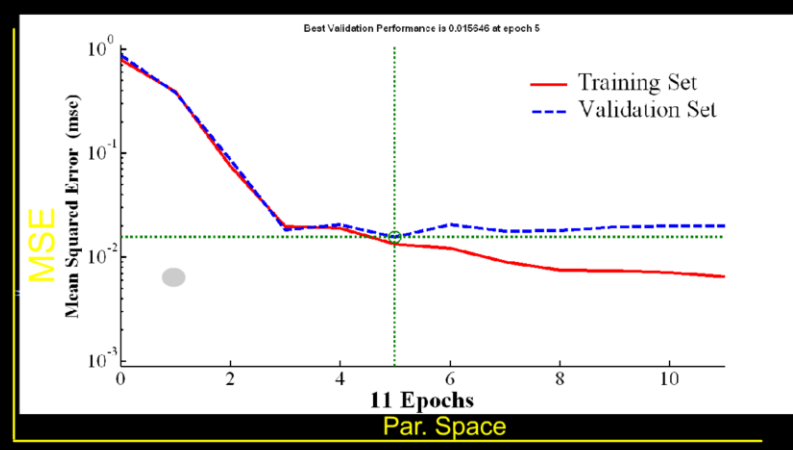


During the training step, one important problem that can occur is the overfitting issue. In this problem, the error on the training set is small but the error becomes large when the network is applied to a new data set.

How to avoid overfitting:

1- increase the training size

2- use less powerful models



During the training step, one important problem that can occur is the overfitting issue. In this problem, the error on the training set is small but the error becomes large when the network is applied to a new data set.

How to avoid overfitting:

1- increase the training size

2- use less powerful models

k-nearest neighbors algorithm

From Wikipedia, the free encyclopedia

Not to be confused with k-means clustering.

In statistics, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951,[1] and later expanded by Thomas Cover.[2] It is used for classification and regression. In both cases, the input consists of the *k* closest training examples in a data set. The output depends on whether *k*-NN is used for classification or regression:

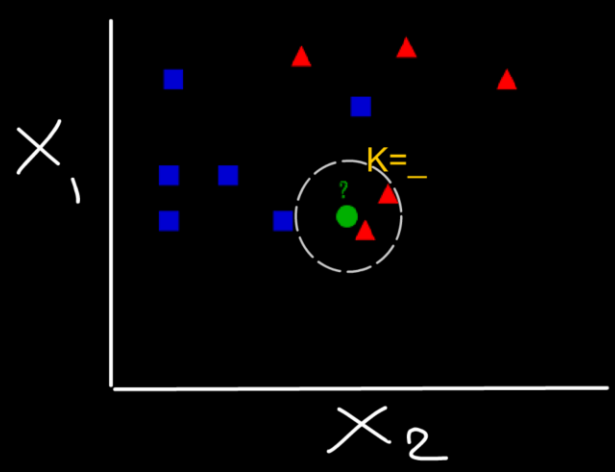
- In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of *k* nearest neighbors.

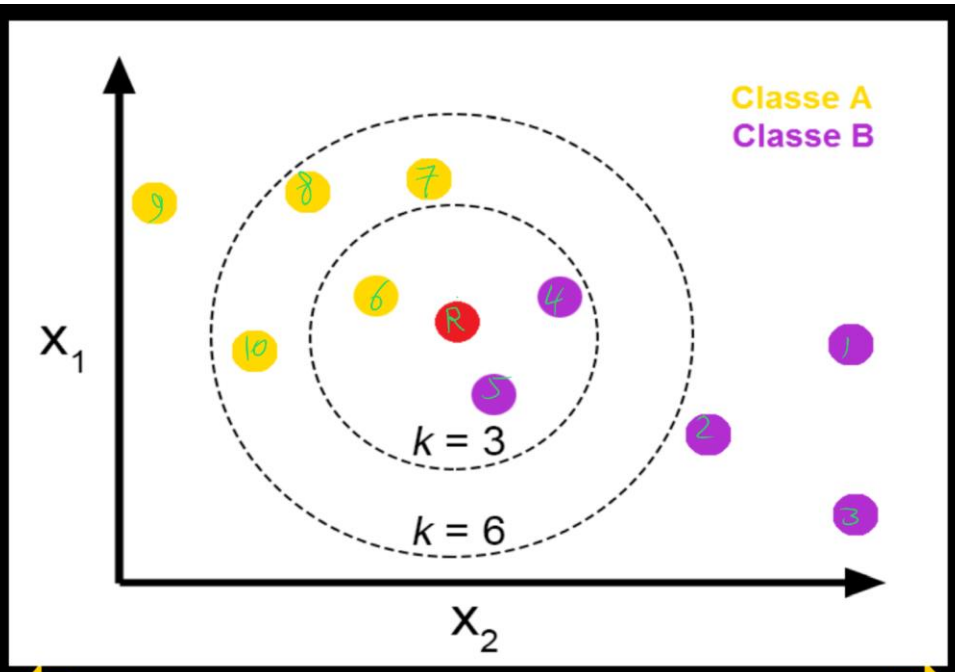
K-nearest neighbor algorithm (KNN)

1- Find K examples that have similar features (e.g., similar color and mass).

2- If classification: Find the majority vote, and assign that class to the new record.

3- If regression: Find the average among those similar examples, and predict that average for the new record.





"In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space" (Wikipedia)

So, we need a metric to find the k closest training examples. Two examples of metrics:

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}$$

$$|x_1 - u_1| + |x_2 - u_2| + \cdots + |x_p - u_p|$$

$$R = (2, 11)$$

	X_1	X_2	Y
D_1 ①	2.5	6	C_1
D_2 ②	5.5	5	C_2
D_3 ③	1.2	12	C_3

$$D_1 = \sqrt{(2-2.5)^2 + (11-6)^2} = 5.02$$

$$D_2 = \sqrt{(2-5.5)^2 + (11-5)^2} = 6.94$$

$$D_3 = \sqrt{(2-1.2) + (11-12)^2} = 1.28 \checkmark$$

Class R?

we have 10 data point with two labels

1- find D_{R1}, D_{R2}, ⋮, D_{R10} → sort

$\boxed{1.2, 1.3, 1.8}$ $k=3$ → $\boxed{B, A, B}$

2, 3, 5.5, 6, 7.8, 9.1, 12 find the associated targets → A, A, A, B, A, B, B, B

if
- regression → average the three targets
- classification → majority vote of the three targets

1- The curse of dimensionality
2- Sorting problems (data structure)
3- Scaling problems
4- Very sensitive to outliers

However:

1- Simple
2- Versatile (regression and classification)
3- No assumptions about data (linear or nonlinear data?)