Ricardo Aler Mur

- The main goals of this lecture are:
  - To describe why it is important to evaluate models, by explaining the concept of generalization and overfitting, both in classification and regression.
  - Then, several methods of evaluation are explained: train / test (holdout), repeated train/test, and crossvalidation, the latter one being the recommended method.
  - There are many evaluation measures, both for classification and regression. Two of them are explained in more detail: success rate (for classification) and root mean squared error (for regression).

- Finally, the process of hyper-parameter tuning is introduced.
- In the initial point, it was shown how the complexity of the model and overfitting are related (the more complex the model, the more likely it is overfitting, although there is always the possibility of underfitting)
- It is explained that all machine learning algorithms have some hyper-parameter that determine the performance of the model, and specifically, its complexity.
- Those hyper-parameters need to be adjusted, or tuned. The process of grid-search, by which all possible combinations of parameters are tested, is explained.

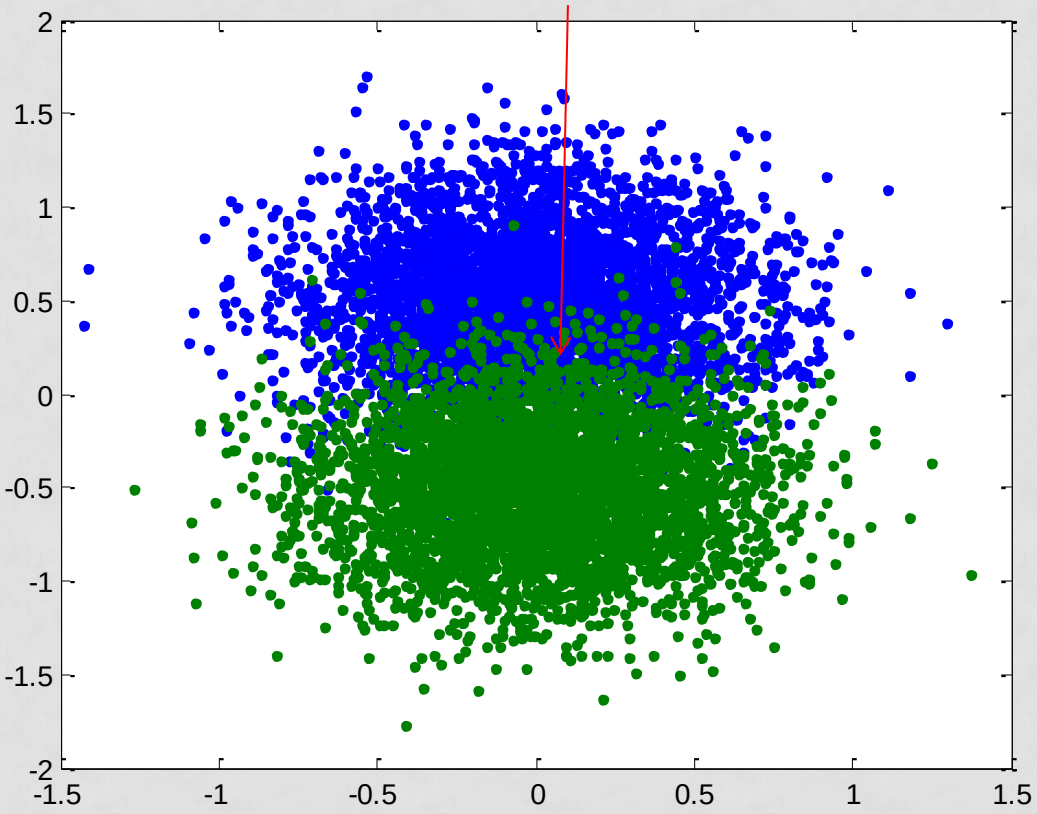
EVALUATION

EVALUATION

- Once the model has been obtained it is important to estimate its future performance
- Why? If a student is evaluated (exam) with the same exercises s/he used for learning, probably s/he will get high grades
- But s/he is not showing that s/he has generalized beyond the training exercises. S/he might have just memorized them
- Models suffer from the same issue
- Solution: use fresh (new) data for testing the model
- Divide the available data into a training partition and a test partition (typically 70% vs. 30%)

EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

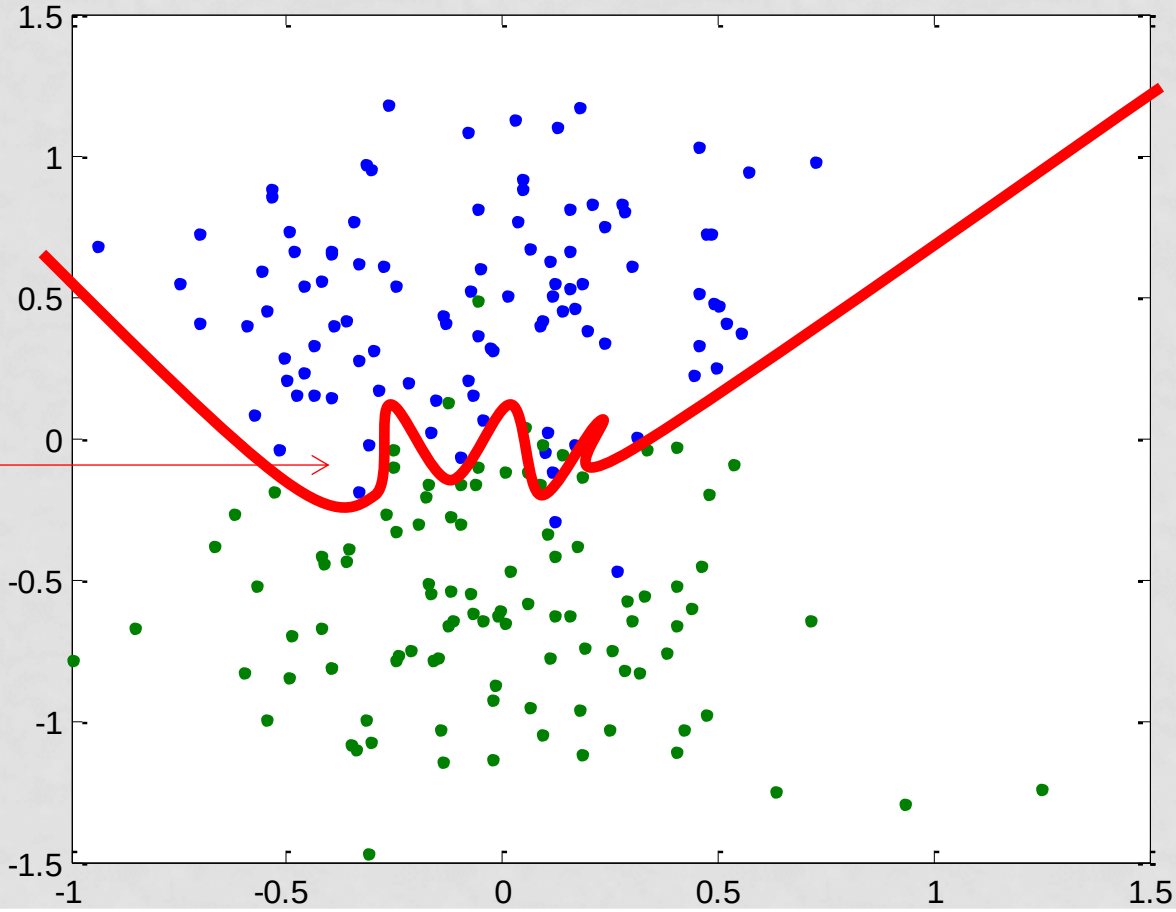
Let's suppose we want to learn a model that is able to separate class "blue" from class "green". Below we can see instance space with thousands of instances. Notice that both classes overlap in the middle



EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

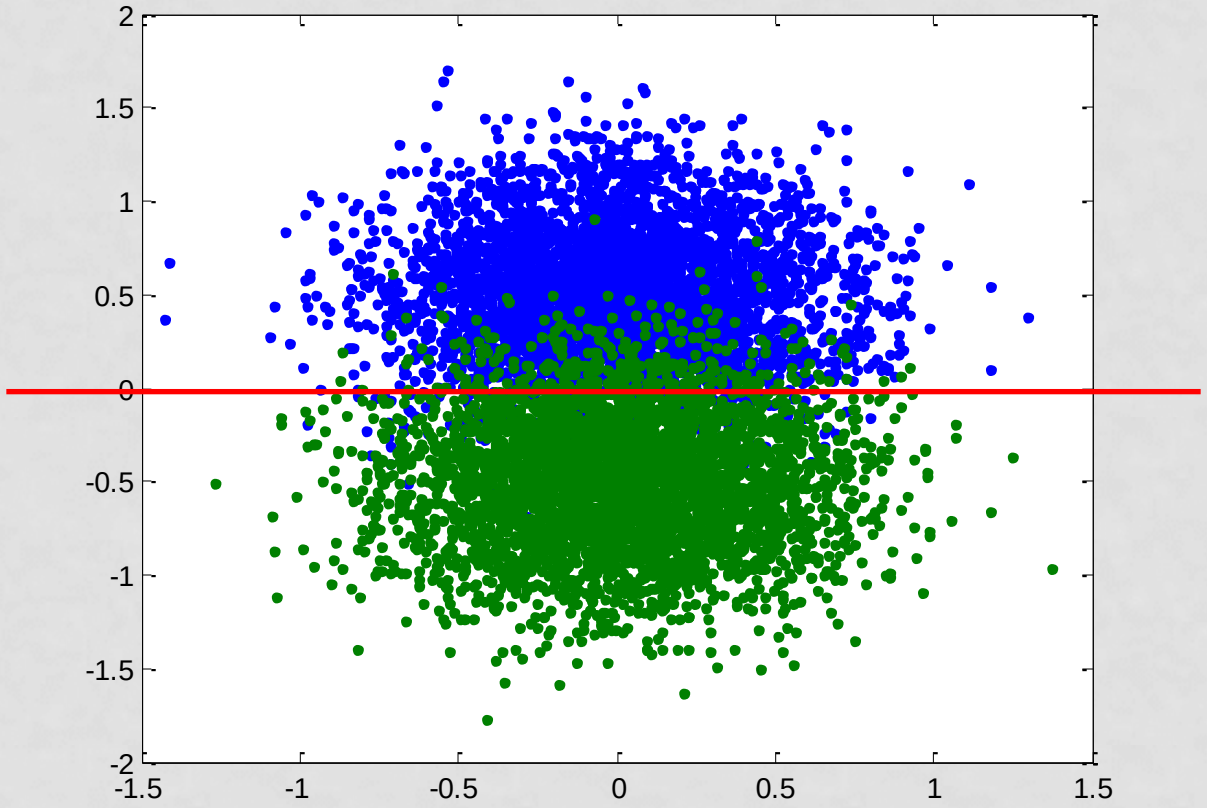
- But if we have few data for training, the following model might be learned
- The model is obviously not generalizing well. It is memorizing the data, or **overfitting** the data

This curve has been learned because there are no green instances here.
But this happened by chance. If we had more instances, probably there would be green instances in that region



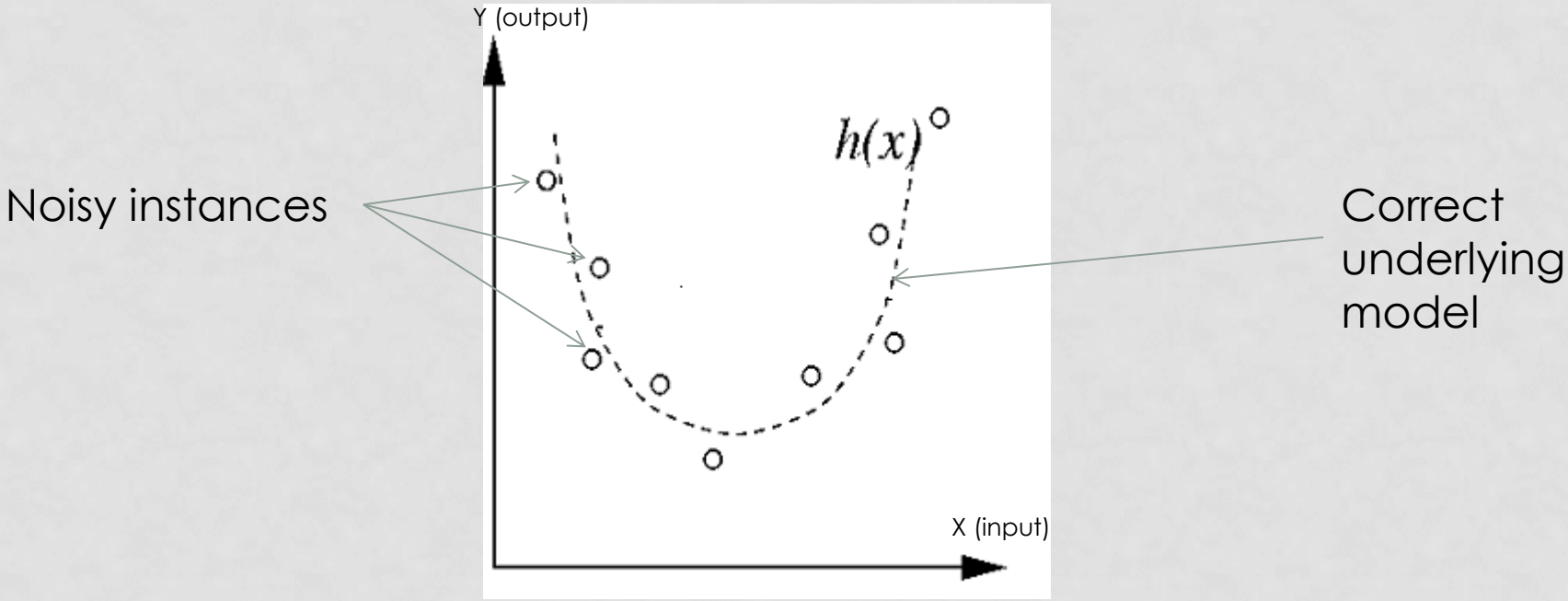
EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

If we had lots of data, the following (correct) model would be learned



EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

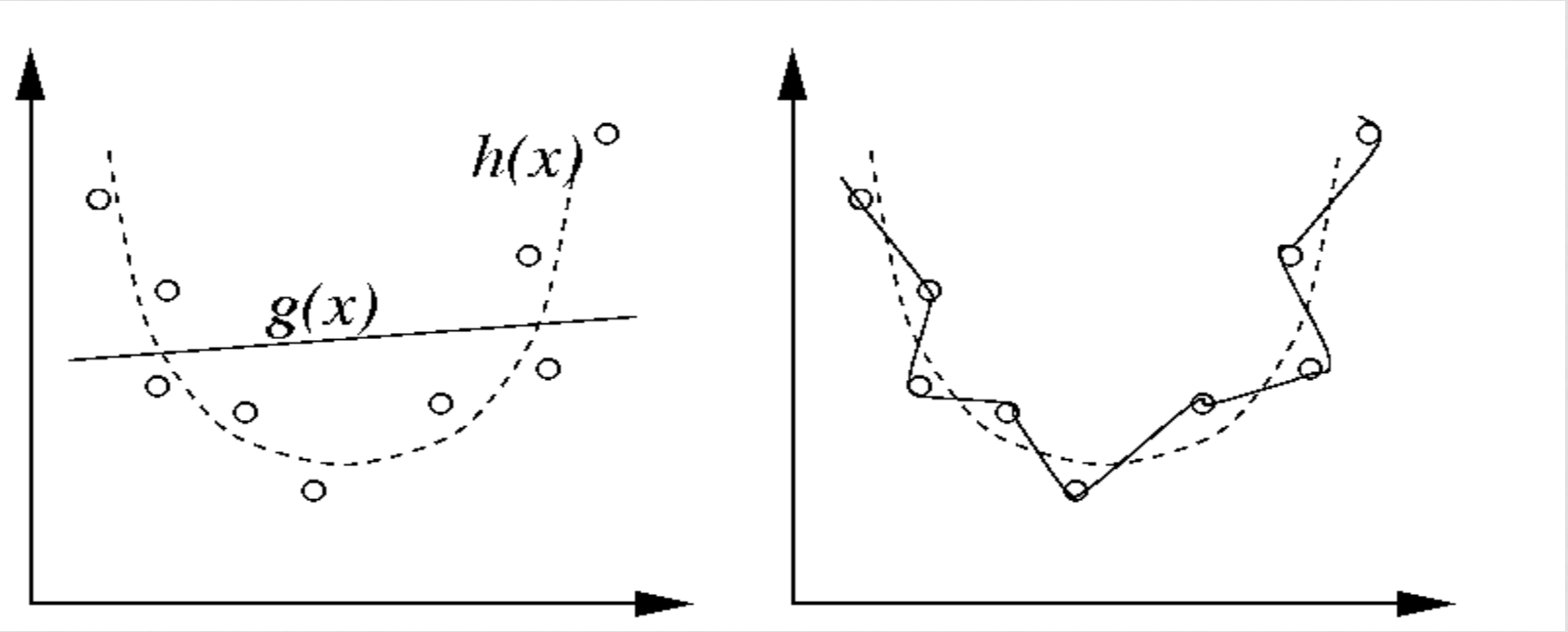
- Let's suppose that the underlying model is a parabola, but instances have some noise
  - For example, **y** might be "temperature", but the thermometer used to measure it is not very accurate



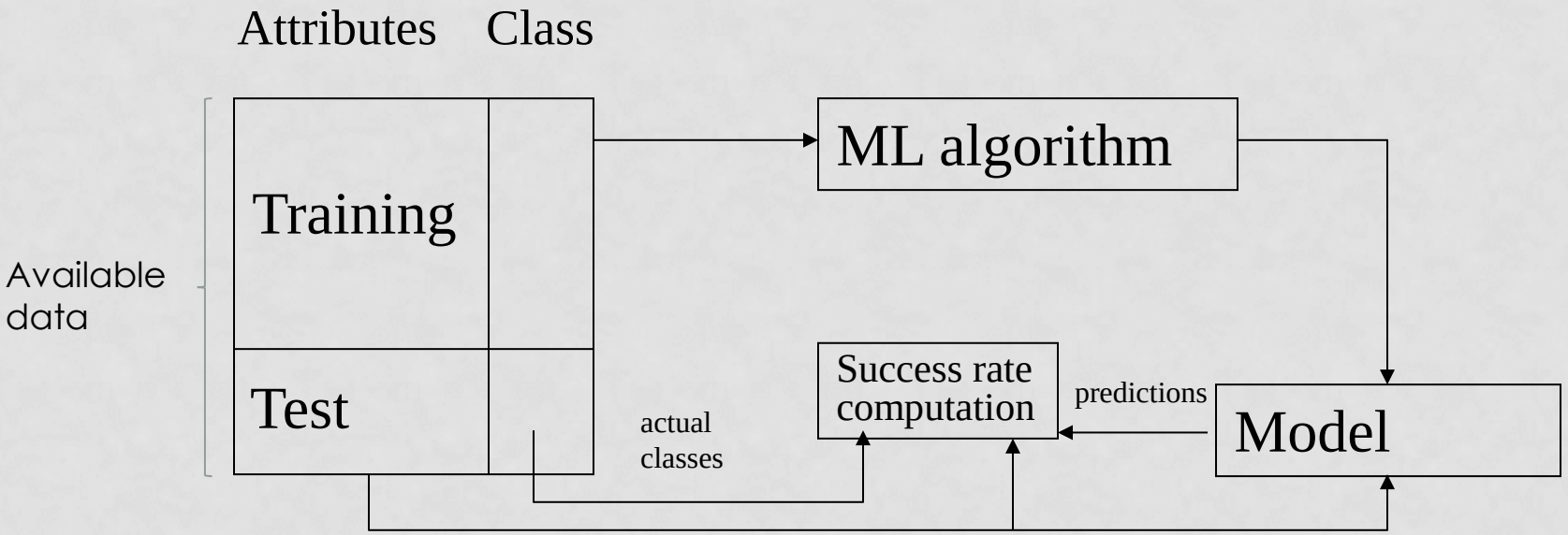
EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

Underfitting

Overfitting



TRAIN / TEST EVALUATION METHOD



Main problem: if available data is not very large, it is possible that the training and/or the test partitions are **biased**

Example of **bias**: There is data about whether persons like playing tennis. In the training partition, there are 51% men and 49% women and the model learns that it is more likely for men to like playing tennis. Maybe it is a random bias?

REPEATED TRAIN / TEST

- Repeat train / test many times
- Given that biases happen randomly, biases that appear in some of the train / test partition will not appear in other train / test partitions. In general random biases will cancel each other after averaging.
- **Method:**
  - **Repeat many times:**
    1. **Sort available data randomly**
    2. **Take the first 70% of instances for training and 30% for test**
    3. **Learn the model with the training partition and test the model with the test partition. Obtain success rate on the test partition**
  - **Average all test success rates**

REPEATED TRAIN / TEST

- Main problem: the different test partitions might overlap: they are not independent
- In an extreme (and unlikely) case, if all test partitions were exactly the same, computing the average of all of them would be useless
- This extreme case never happens, but there is always some overlap between test partitions

CROSSVALIDATION

- The available data (originally called "training data") is divided into k folds (k partitions). With k=3, three partitions A, B, and C.
- The process has k steps (3 in this case):
  - Learn model with A, B, and test it with C (T1 = success rate on C)
  - Learn model with A, C, and test it with B (T2 = success rate on B)
  - Learn model with B, C and test it with A (T3 = success rate on A)
  - Success rate T = (T1+T2+T3)/3
- The final classifier CF is learned **with the whole dataset (A, B, C)**. It is assumed that T is a good estimation of the success rate of CF
- k=10 is commonly used

PERFORMANCE MEASURES

- What is meant by “error” (or “success”):
  - In classification, **success rate**: count how many times the model gives the correct answer and divide by the number of test data
  - In regression: **root mean squared quadratic error**

Performance measure	Formula
mean-squared error	$\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}$
root mean-squared error	$\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{n}}$
mean absolute error	$\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{n}$
relative squared error	$\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}$, where $\bar{a}=\frac{1}{n}\sum_i a_i$
root relative squared error	$\sqrt{\frac{(p_1-a_1)^2+\ldots+(p_n-a_n)^2}{(a_1-\bar{a})^2+\ldots+(a_n-\bar{a})^2}}$
relative absolute error	$\frac{\lvert p_1-a_1\rvert+\ldots+\lvert p_n-a_n\rvert}{\lvert a_1-\bar{a}\rvert+\ldots+\lvert a_n-\bar{a}\rvert}$
correlation coefficient	$\frac{S_{PA}}{\sqrt{S_P S_A}}$, where $S_{PA}=\frac{\sum_i (p_i-\bar{p})(a_i-\bar{a})}{n-1}$, $S_p=\frac{\sum_i (p_i-\bar{p})^2}{n-1}$, and $S_A=\frac{\sum_i (a_i-\bar{a})^2}{n-1}$

* p are predicted values and a are actual values.

Basic criteria for evaluating classification models

- Let's suppose that there are two classes. A successful model has to obtain a success rate larger than 50%. Otherwise, tossing a coin would give better results
  - For M classes, the baseline result is 1/M * 100
- In classification problems, a successful model has to obtain a success rate larger that the percentage of the majority class.
  - The reason is that some classification problems are imbalanced, i.e. there are many more instances for the "Yes" class than for the "No" class.
  - Let's suppose that 90% of instances belong to "No" and only 10% of instances belong to "Yes". A **trivial classifier** that always says "No" would already obtain 90% success rate!! (by doing nothing)
  - Our model has to do better than that

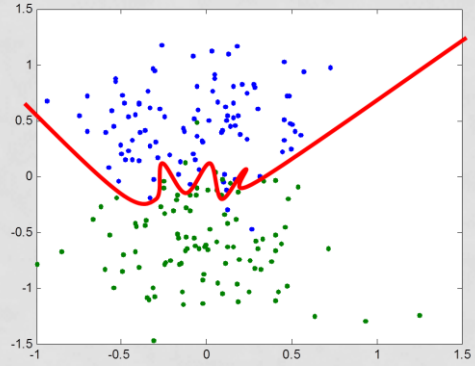
HYPERPARAMETER TUNING

HYPER-PARAMETERS

- Each machine learning algorithm has one or several parameters (called hyper-parameters)
- For instance, KNN has *K* (the number of neighbors)
- For instance, decision trees have:
  - *max_depth*: the maximum depth of the tree
  - *min_samples_split*: the minimum number of instances to split a node (the default value is 2: if a node contains fewer than 2 instances, the node is not split)
- Finding the correct value of a hyper-parameter may result in improved performance of the classifier

HYPER-PARAMETERS

- Finding the correct value of a hyper-parameter may result in improved performance of the classifier
- Hyperparameters control, directly or indirectly, the complexity of a classifier
- In general, the more complex a model is, the more likely it will overfit the data (but if it is not complex enough, it will underfit the data)
- Example of a complex classifier (the curve is allowed to turn around many times):



HYPER-PARAMETER TUNING

- In decision trees, if *max_depth* is very large or *min_samples_split* is very small, the resulting tree will be large, and therefore, complex
- We can try to give appropriate values to the hyper-parameters by hand (trying and testing)
- But there is an automatic way of tuning the hyperparameters which is called **grid-search**

GRID SEARCH

MAX_DEPTH	2	4	6	8
MIN_SAMPLES				
2	(2,2)	(2,4)	(2,6)	(2,8)
4	(4,2)	(4,4)	(4,6)	(4,8)
6	(6,2)	(6,4)	(6,6)	(6,8)

Grid search means: try all possible combinations of values for the two (or more) hyper-parameters. For each one, carry out a crossvalidation, and obtain the success rate. Select the combination of hyper-parameters with best success-rate.

MAX_DEPTH	2	4	6	8
MIN_SAMPLES				
2	70%	75%	76%	68%
4	72%	73%	**81%**	70%
6	68%	70%	71%	67%