Predictive Models

Iaroslav Shcherbatyi

Agenda

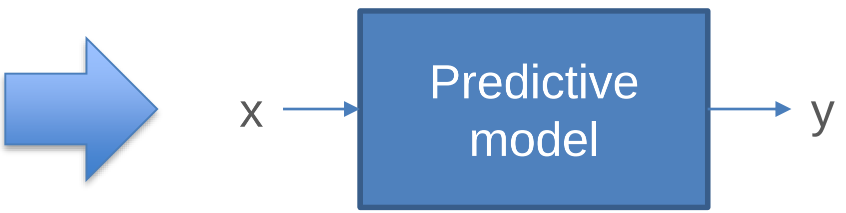
- What is predictive modeling?
- How to perform predictive model selection?
- Aspects and take-aways of predictive models: overfitting, missing information
- Specifics of predictive model classes:
  - Linear models
  - K Nearest Neighbors
  - Kernel Support Vector Machines
  - Decision trees
  - Gradient Boosting

Predictive modelling

Extract generalizable models from data.

Set of example inputs and outputs: a *dataset*.

x_1	x_2	y
0.61	0.21	151
-0.51	-0.26	75
-0.11	-0.36	206
-0.36	0.21	135
…		



Predictive model estimates outputs accurately for previously unseen inputs.

See dataset examples in models.ipnb

Example datasets

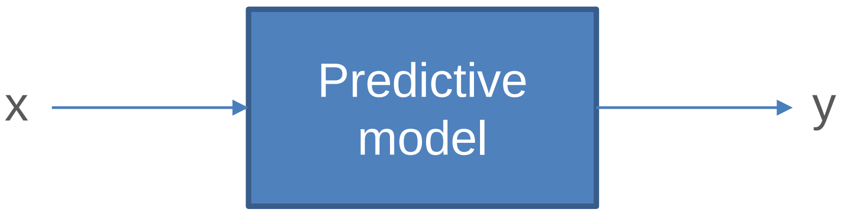
Age	Gender	Pain type	Blood pressure	Oldpeak	Sick
70	male	4	130	2.4	yes
67	female	3	115	1.6	no
57	male	2	124	0.3	yes
64	male	4	128	0.2	no
74	female	2	120	0.2	no
65	male	4	120	0.4	no
56	male	3	130	0.6	yes
59	male	4	110	1.2	yes
60	male	4	140	1.2	yes
63	female	4	150	4	yes
59	male	4	135	0.5	no

Example datasets

Image	Caption
	A car driving near the forest
	People playing Frisbee on the beach
	A dog playing with a soccer ball

Notation

Dataset is represented as n instances of inputs and outputs.



Separate inputs are generally denoted as x and outputs as y.

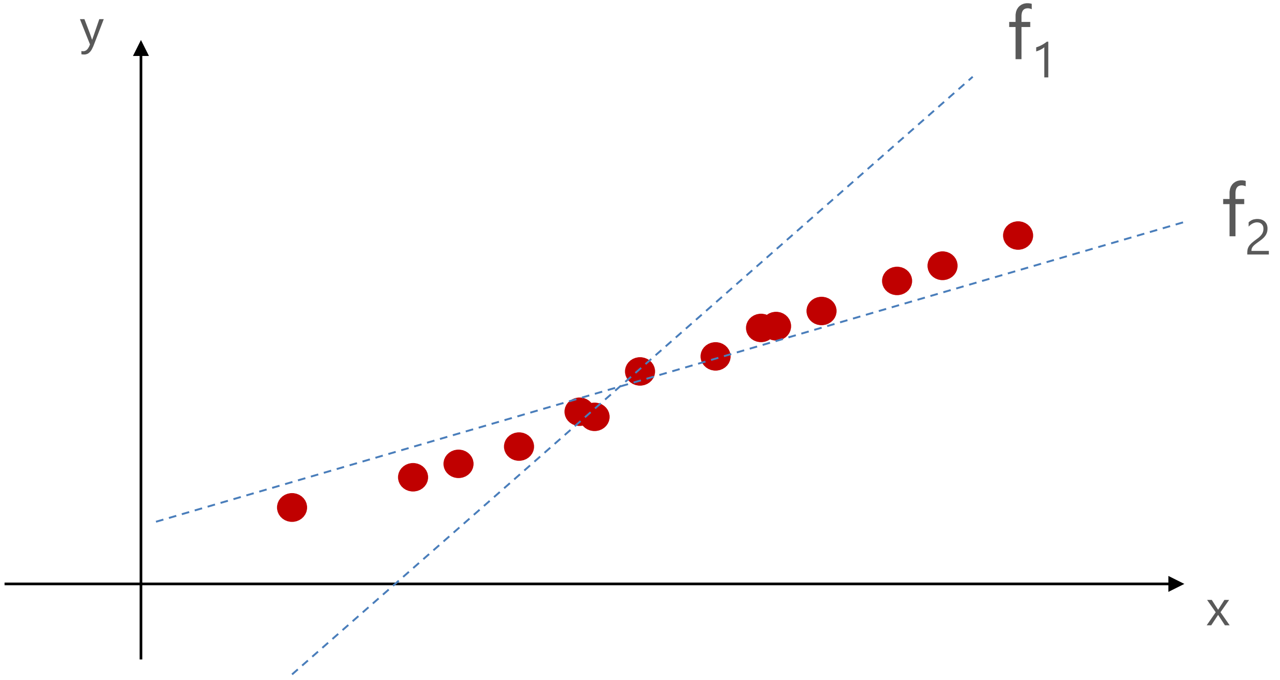
All available inputs and outputs are denoted as X and Y.

All possible inputs and outputs are denoted in this lecture as X* and Y*.

Model fitting

Core element of supervised learning

How to find a model which is most accurate on available data?



Model fitting

- **How to define a model?**
- How to compare different models?
- How to perform a model fitting?

Model parameters

Predictive model is a function of the form $f : X^* \to Y^*$

Every model is defined by a set of its paramters $w \epsilon W$

Example: For linear model, parameters w is a vector of the length *n:*

$$f\left(w, x\right) = w^T x = \sum_{i \epsilon 1,..,n} w_i x_i$$

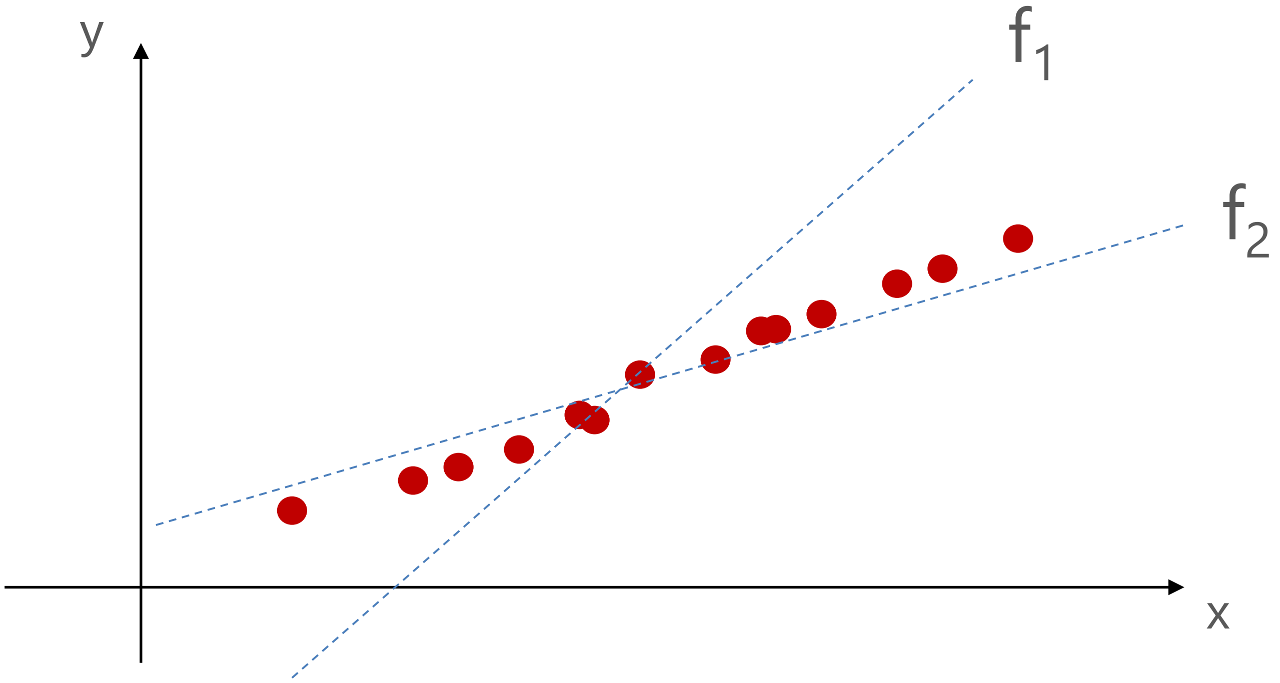
Set of all such vectors defines the set of all parameters.

Model fitting

- How to define a model?
- **How to compare different models?**
- How to perform a model fitting?

Model fitting: model error

Assume we have two models, f_1 and f_2. How to choose between two?



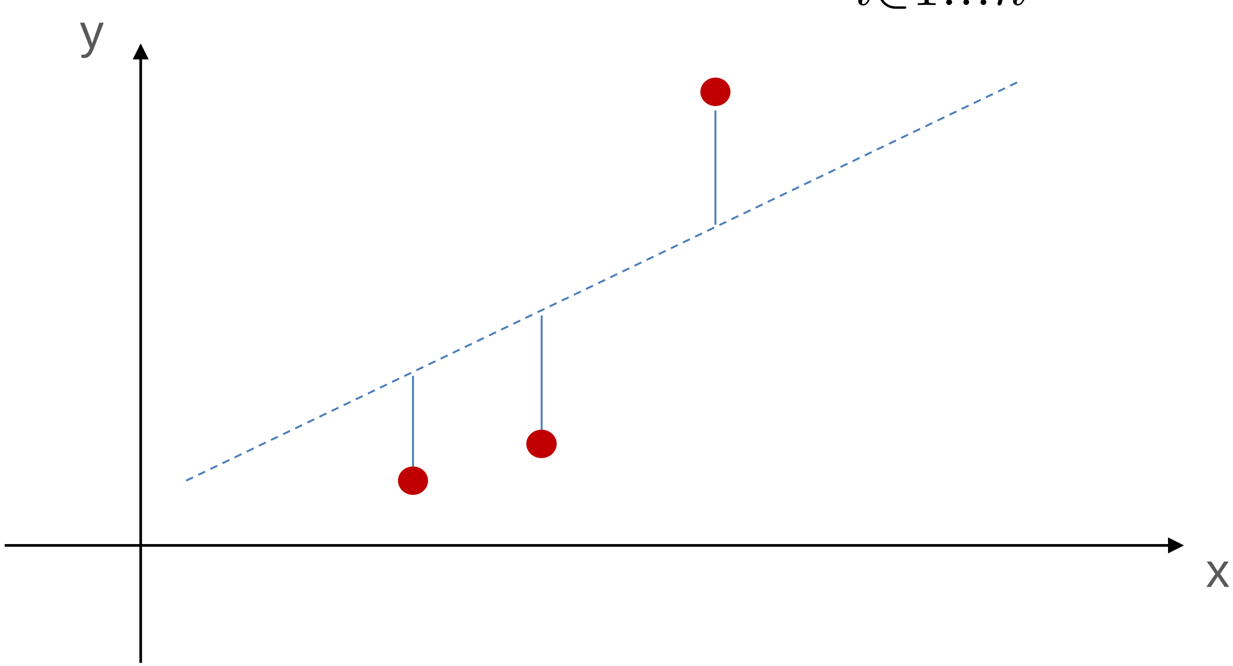
Model fitting: model error

Regression fitting problem: outputs of the model are real numbers.

Loss function: measures how good the fitted function aligns with data.

(e.g., least squares for regression)

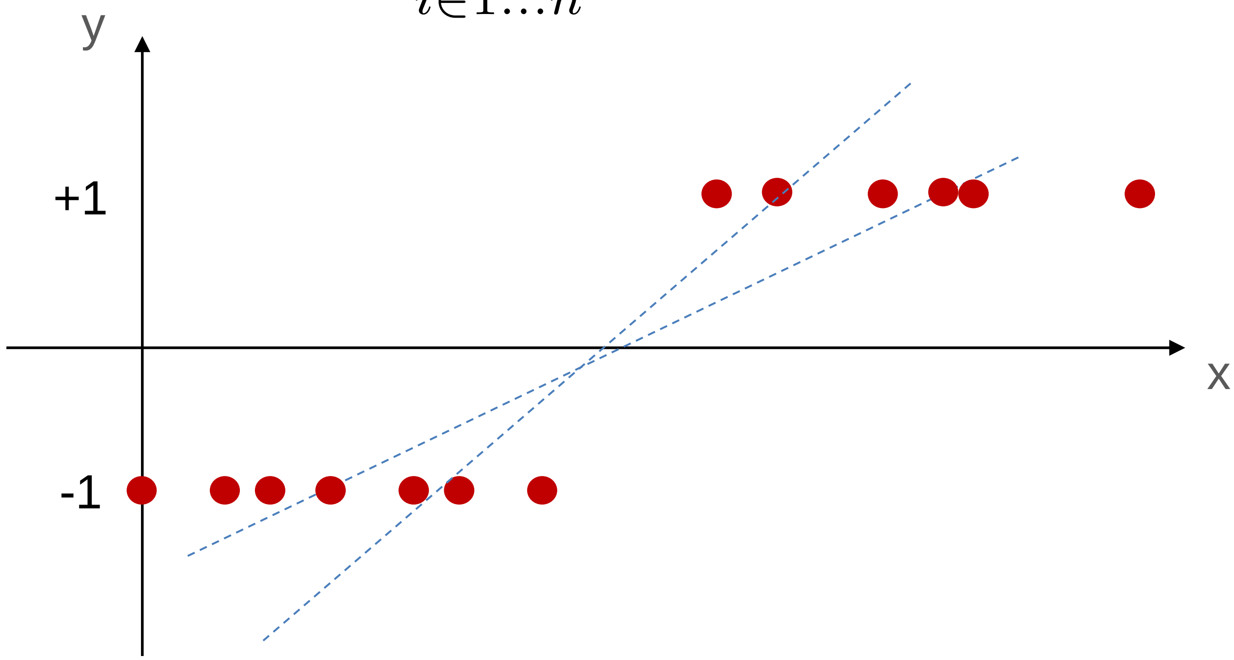
$$L(p, y) = 0.5 * \|p - y\|_2^2 = 0.5 * \sum_{i \in 1 \ldots n} (p_i - y_i)^2$$



Model fitting: model error

Binary classification problem: model output is binary
e.g.: sick / healthy.
Loss function for binary classification: misclassification rate.

$$L(p, y) = \sum_{i \in 1 \ldots n} |sign(p_i) - y_i|$$



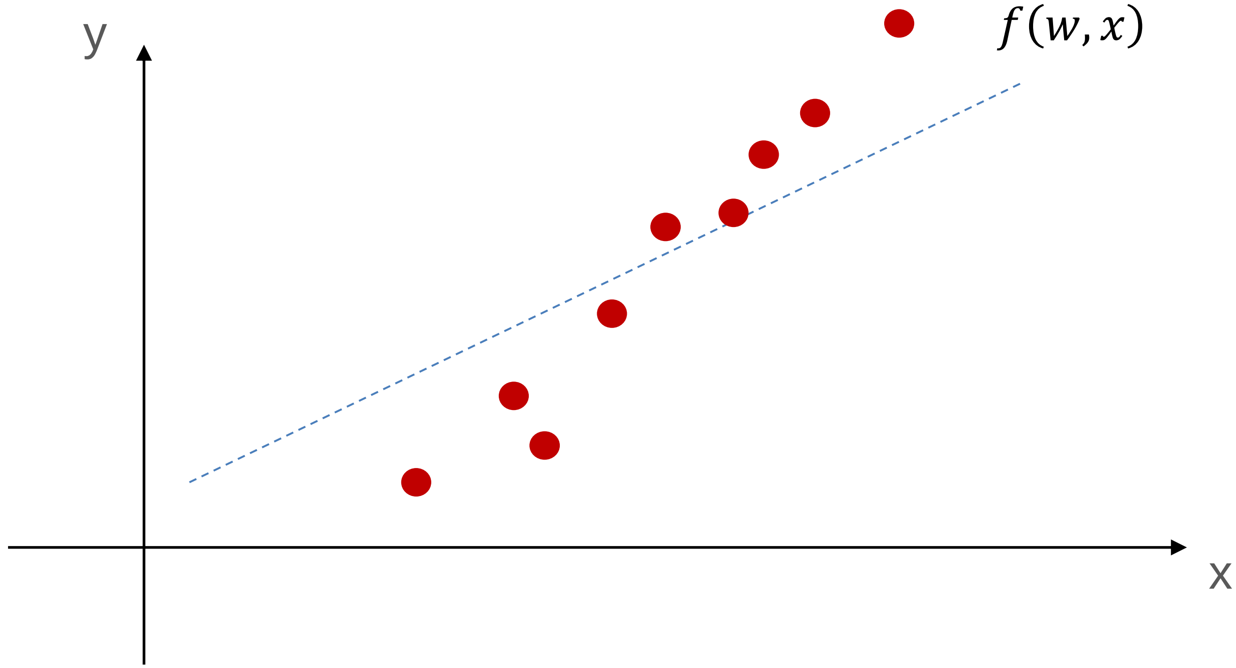
Model fitting

- How to define a model?
- How to compare different models?
- **How to perform a model fitting?**

Model search?

Loss function: $$L\left(p,y\right) = 0.5\|p-y\|_2^2 = \sum_{i\in 1\dots n}\left(p_i-y_i\right)^2$$

Model function: $$f\left(w,x\right) = w^T x = \sum_{i\in 1\dots n} w_i x_i$$

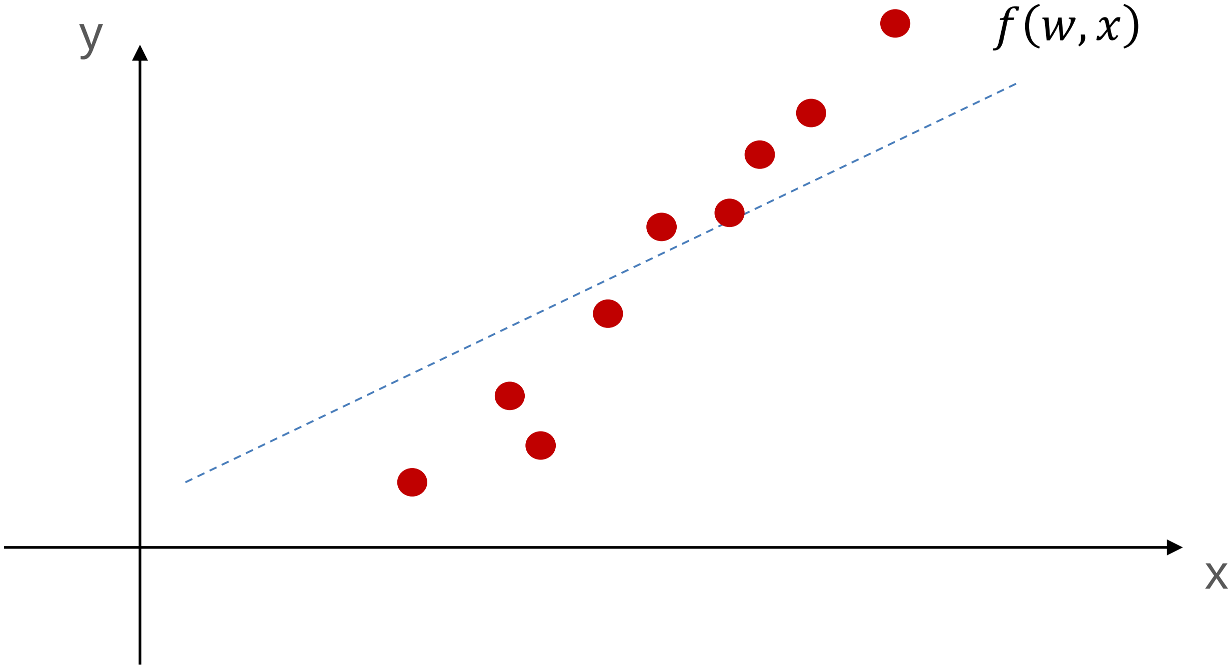


Model search

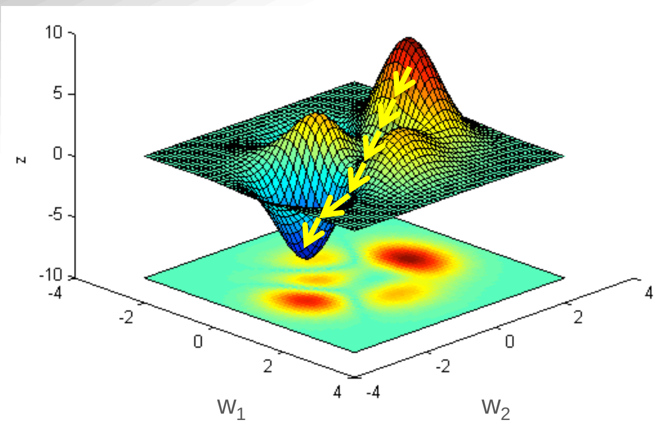
Solve

$$\min_{w \in W} \sum_{i \in 1 \ldots m} L\left(f\left(w, x_i\right), y_i\right)$$

Using ~~brute force~~, gradient descent or your favorite heuristic



Gradient Descent



Algorithm *Gradient Descent*

$$Initialize \quad w = [0, ..., 0]$$

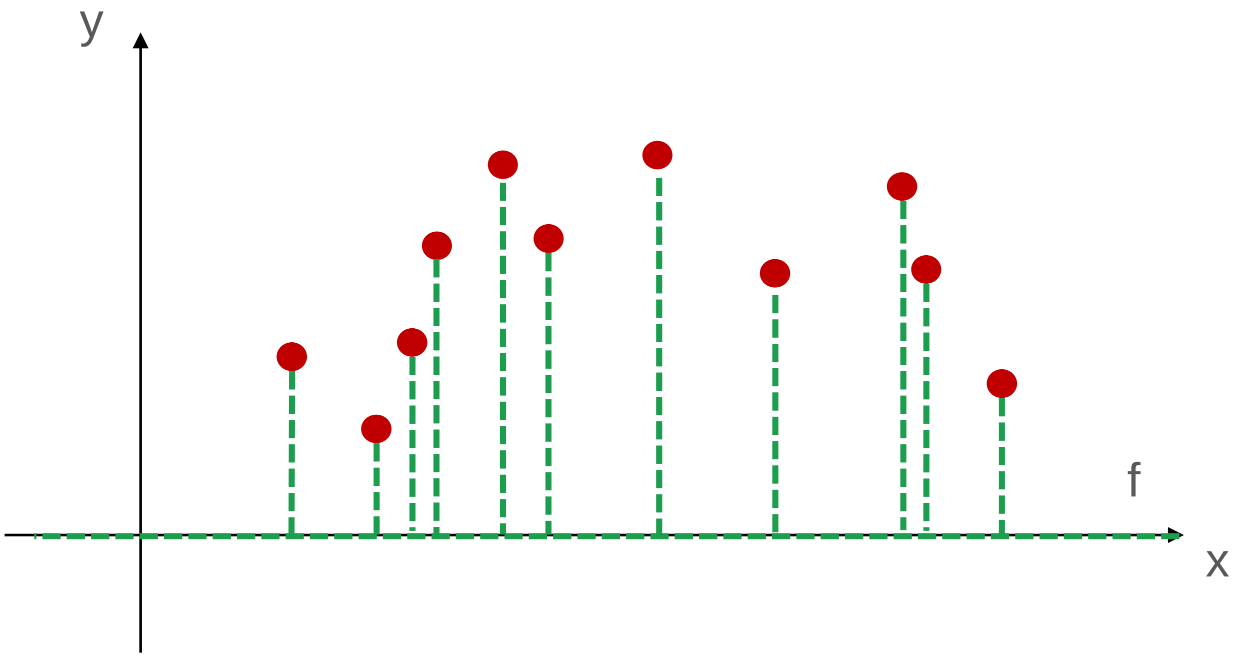
$$for \quad t = 1, ..., T:$$

$$w := w - \underset{\text{stepsize}}{\eta} \underbrace{\nabla_w L(f(w, x), y)}_{\text{gradient}}$$

Hyperparameters: stepsize η and iterations T (e.g., η=0.1 and T=100)

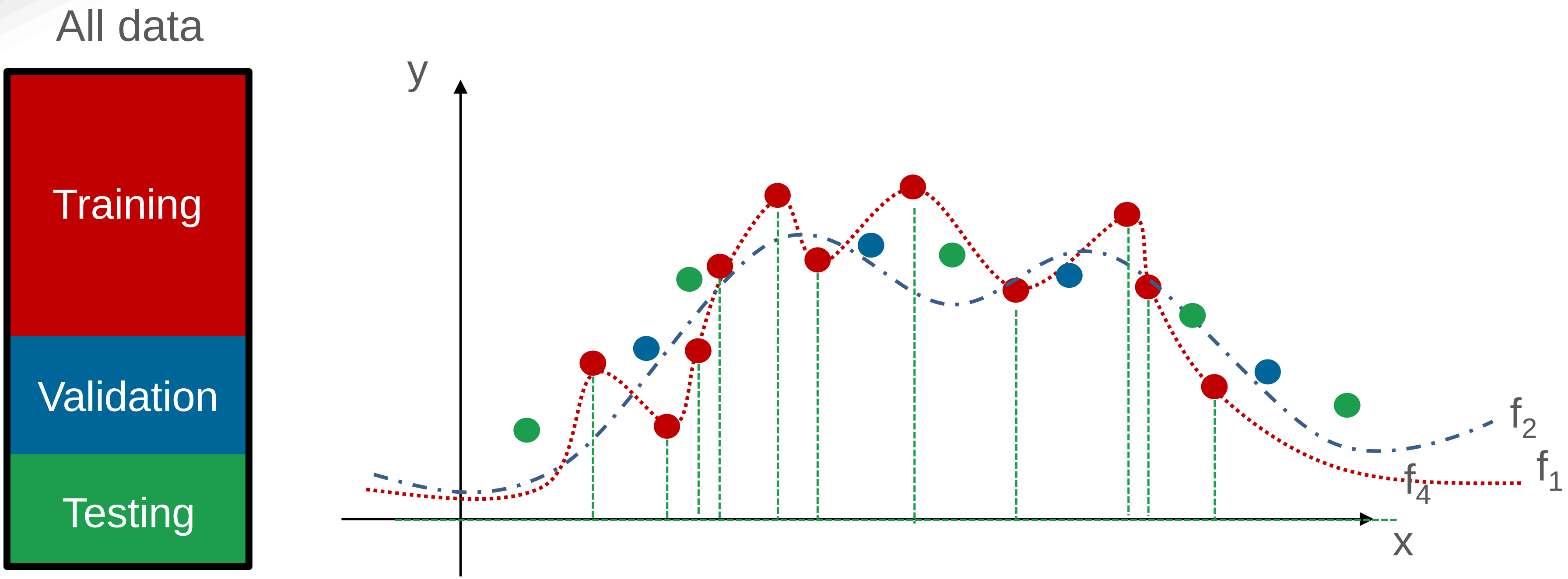
Question

Does good fit implies good generalization?



Avoiding overfitting

Estimate of accuracy on unseen data can be given with training, validation and test split of all available data.



Why test set?

Parameter p: sets model class (SVM, KNN, …)

$$f : X \times W \times P \rightarrow R$$

Can overfit too!

Model selection as bilevel optimization:

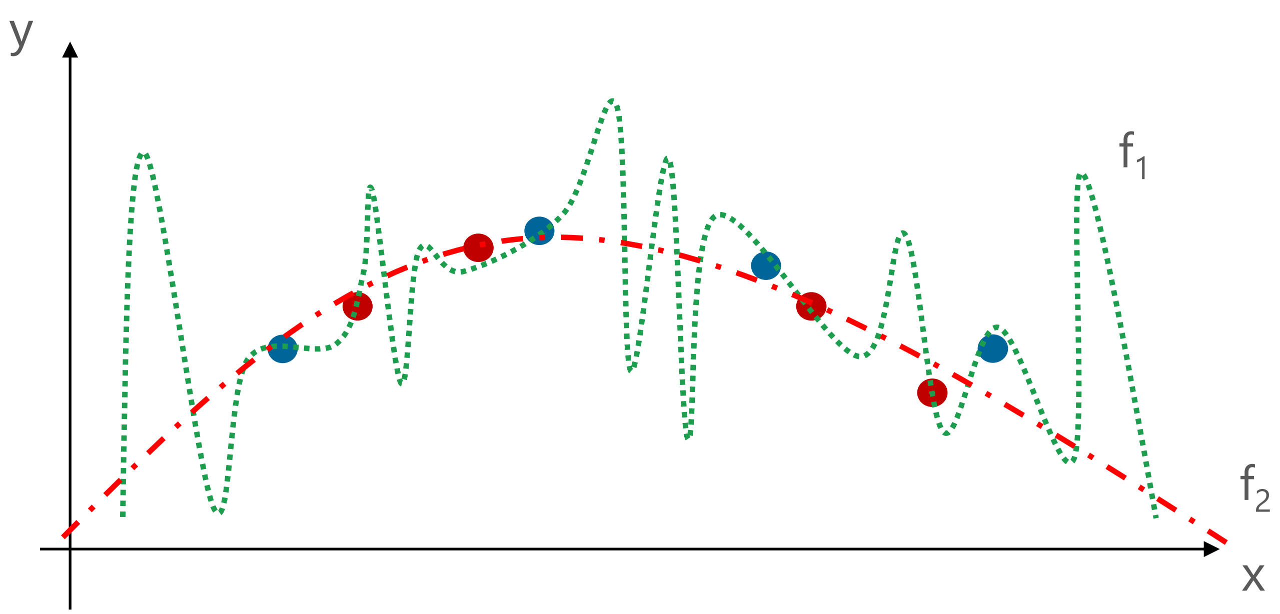
$$\min_{p \in P} \sum_{i \in I_{val}} L(f(x_i, w^*, p), y_i)$$

Subject to

$$w^* = argmin_{w \in W(p)} \sum_{i \in I_{train}} L(f(x_i, w, p), y_i)$$

Complexity control

Which of the models do you expect to have a better performance on a test set?



Complexity control

Complexity control: *explicit* – e.g. number of neurons in neural network

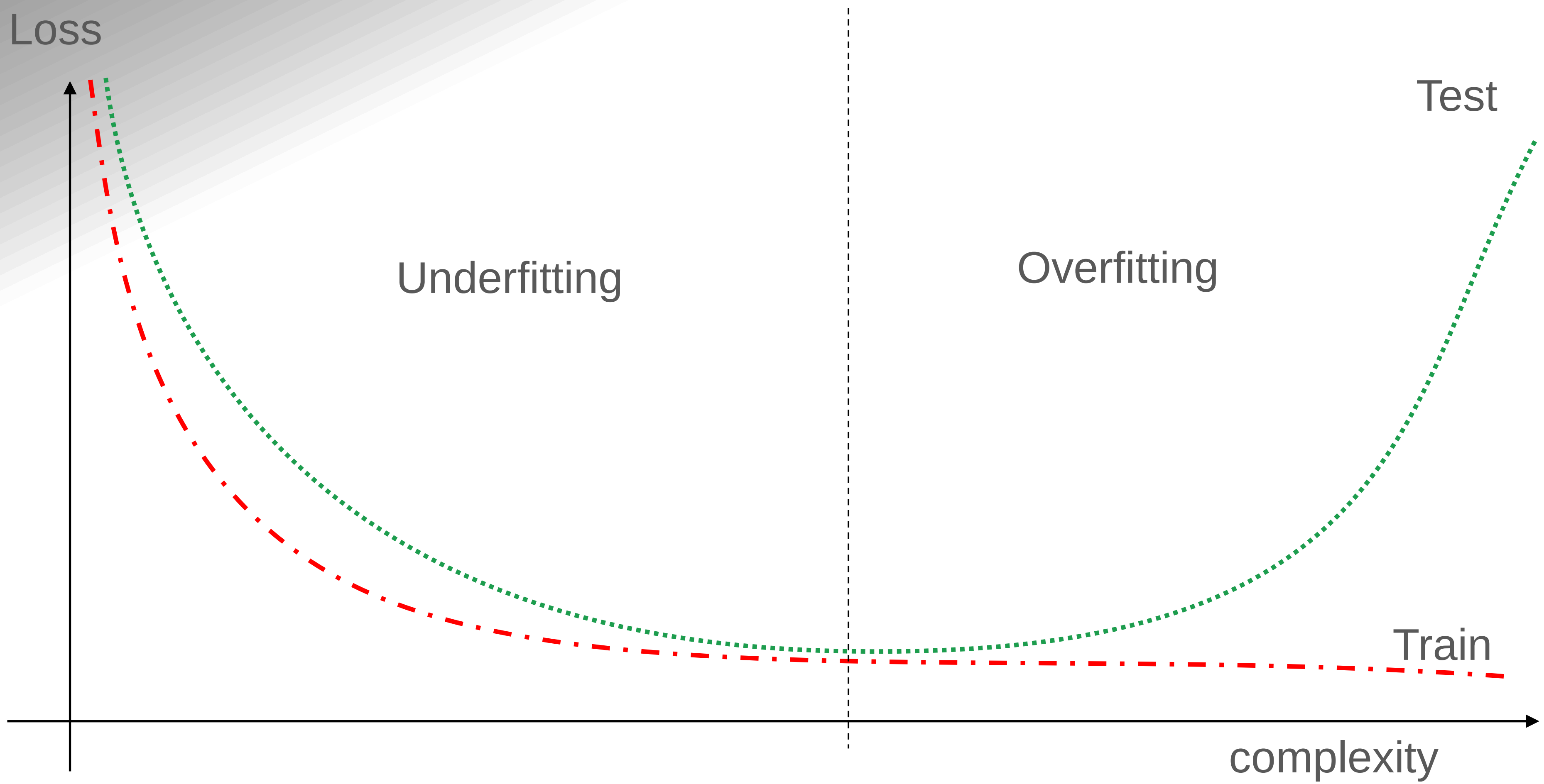
Complexity control: *using complexity function* $r : W \to R_+$

$$\min_{w \in W} \left[r(w) + C \sum_{i \in 1 \dots n} L(f(w, x_i), y_i) \right]$$

L_2 complexity function: smooth model outputs; $\sum_{i \in 1 \dots n} \|w_i\|_2^2$

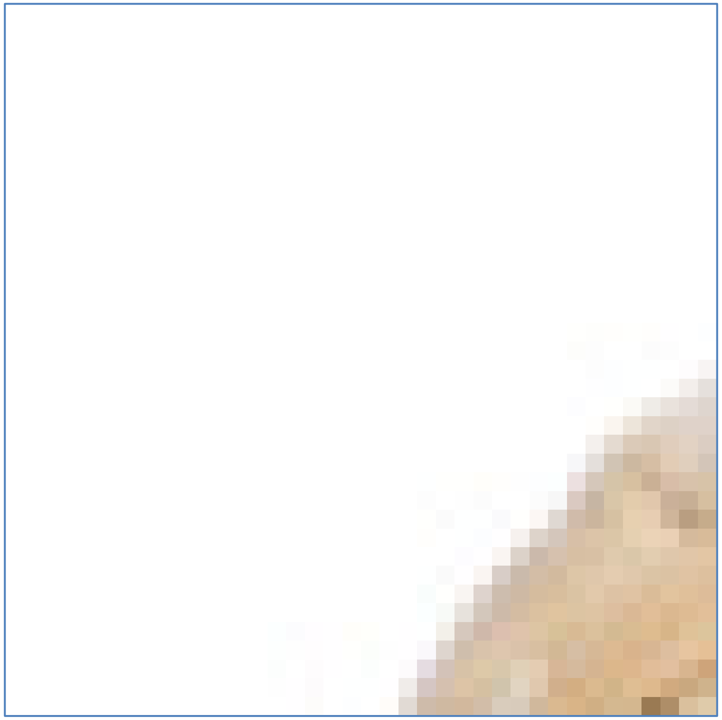
L_1 complexity function: smooth model outputs + sparse model parameters. $\sum_{i \in 1 \dots n} |w_i|_1$

Complexity control



Fundamental limitations

What is shown on this image?



Fundamental limitations

Predictive model is good to the extend to which the data is good.

Fundamental limitations

Predictive model is good to the extend to which the data is good.

Predict who is a student

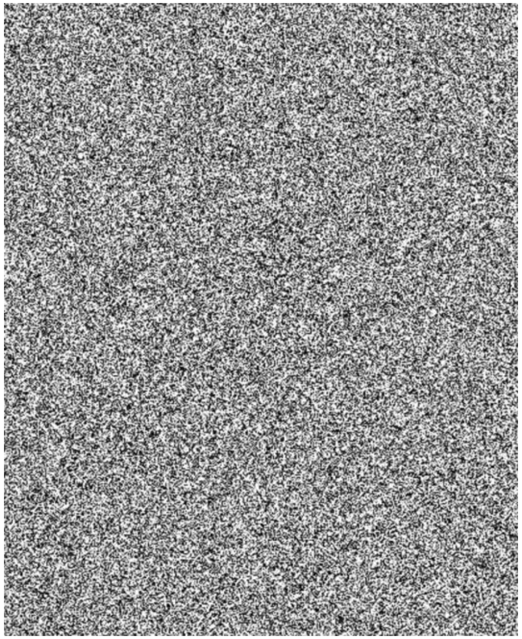
Eye color	Blood type	Is student
Green	1	?
Brown	2	?
Green	1	?
Blue	2	?
Brown	1	?

Data representation

Predictive model is good to the extend to which the data is good.

Unencrypted

Robust bcrypt hashing



Data representation

Predictive model is good to the extend to which the data is good.

Simple

Type 1	Type 2	Type 3	Type 4
1	0	0	0
0	0	1	0
0	1	0	0
0	0	0	1
1	0	0	0

Complex

Blood type
1
3
2
4
1

Hands on

Source: https://github.com/iaroslav-ai/ed3s-2017

Predictive models

- Linear models
- K Nearest Neighbors
- Kernel Support Vector Machines
- Decision trees
- Boosting models

Choice of models inspired by https://arxiv.org/pdf/1708.05070.pdf

Linear models

Definition of the model:

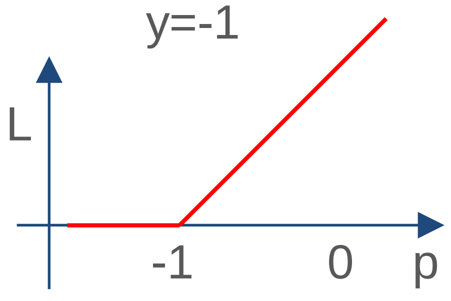
For regression: $$f(x) = b + x^T w = b + \sum_{i=1\ldots n} x_i w_i$$

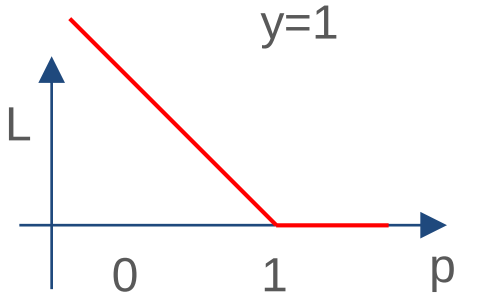
For binary classification: $$f(x) = sign(b + x^T w)$$

Loss functions

Hinge loss: max(1-yp, 0)

Classification:

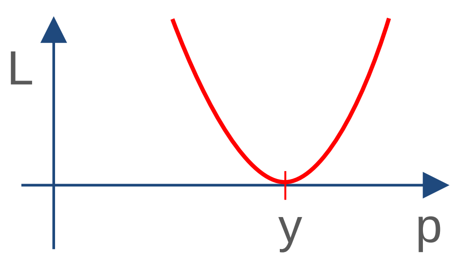


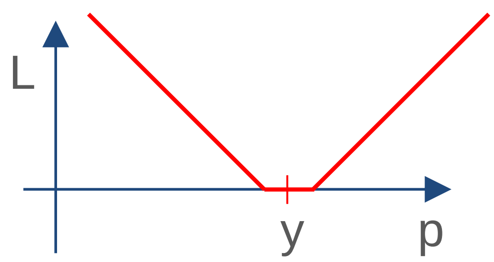


Sq. error: $(y-p)^2$

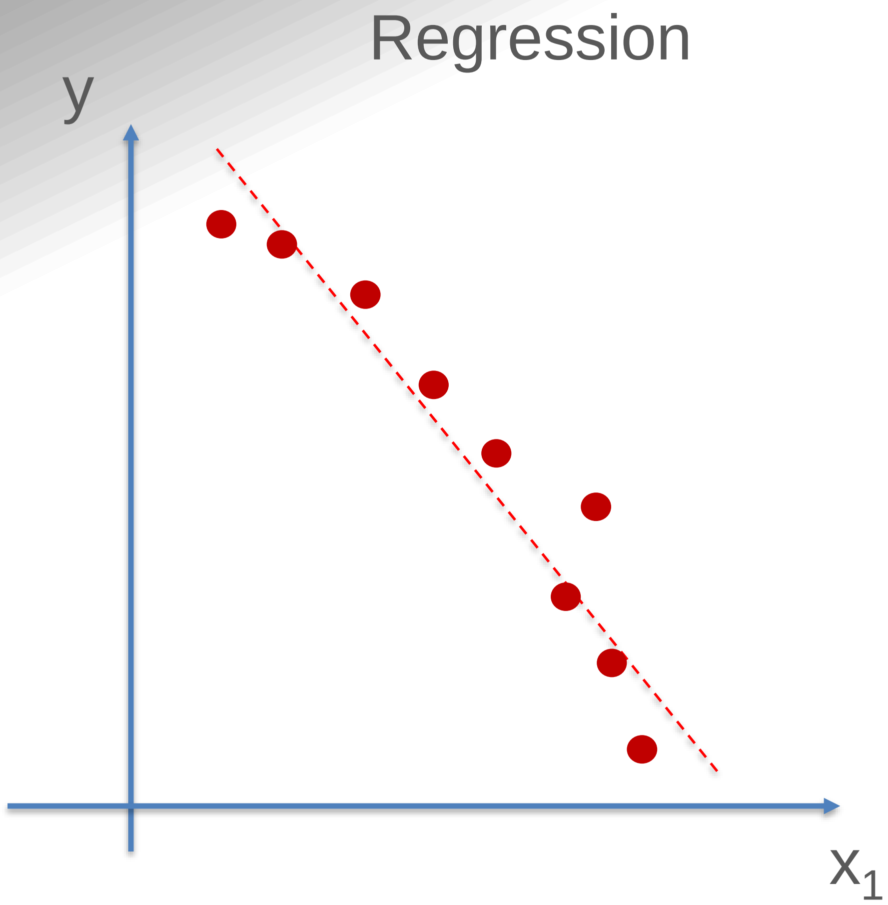
Xi - insensitive: max(|y-p|-c, 0)

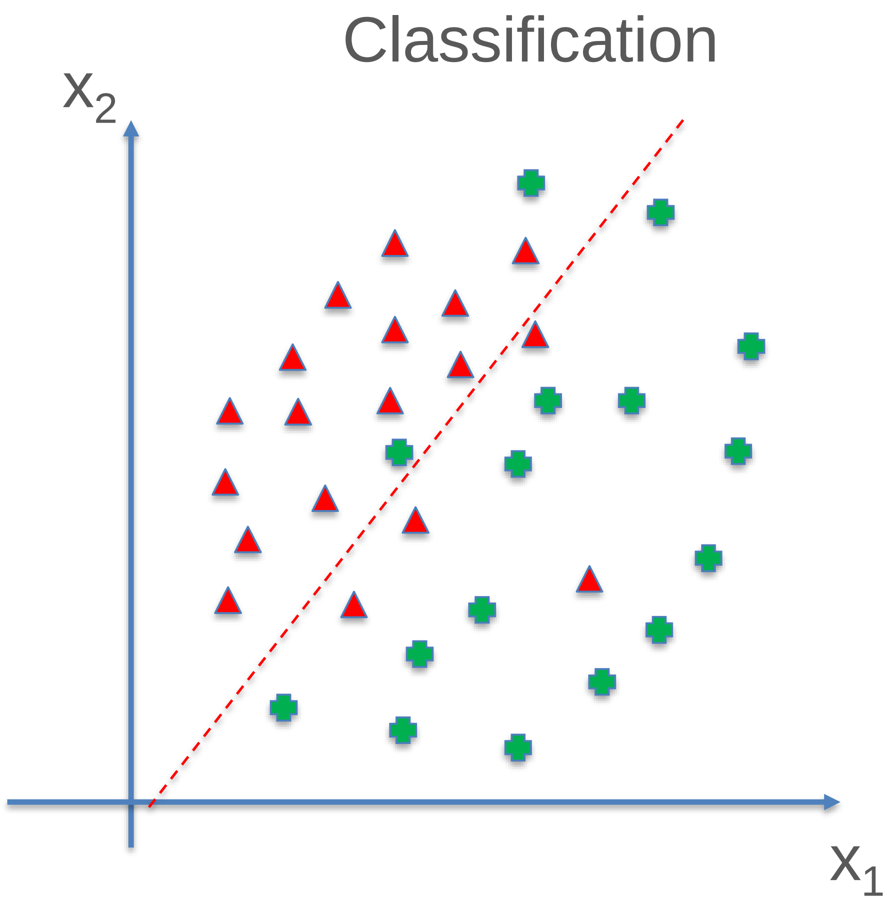
Regression:





Linear models: example





Linear models: pros and cons

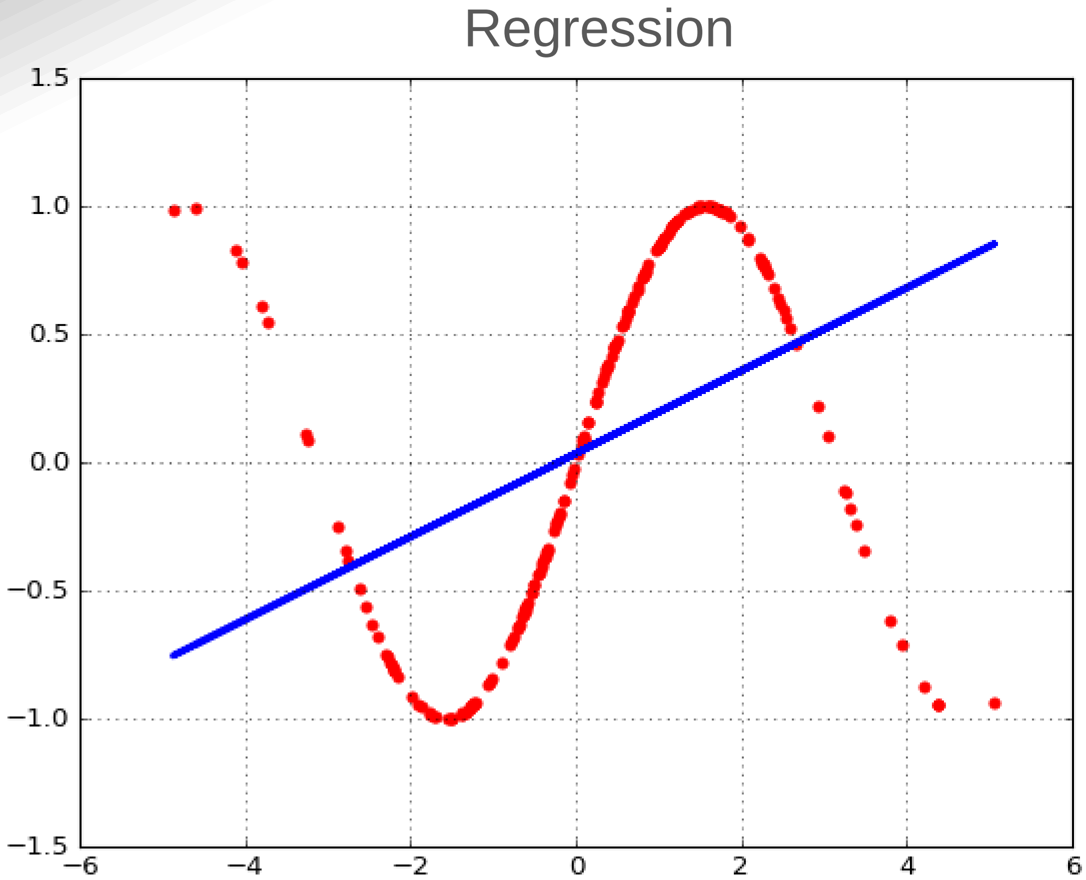
Pros:

- Best fitting set of parameters can be found in polynomial time
- Easy to interpret
- Fast evaluation

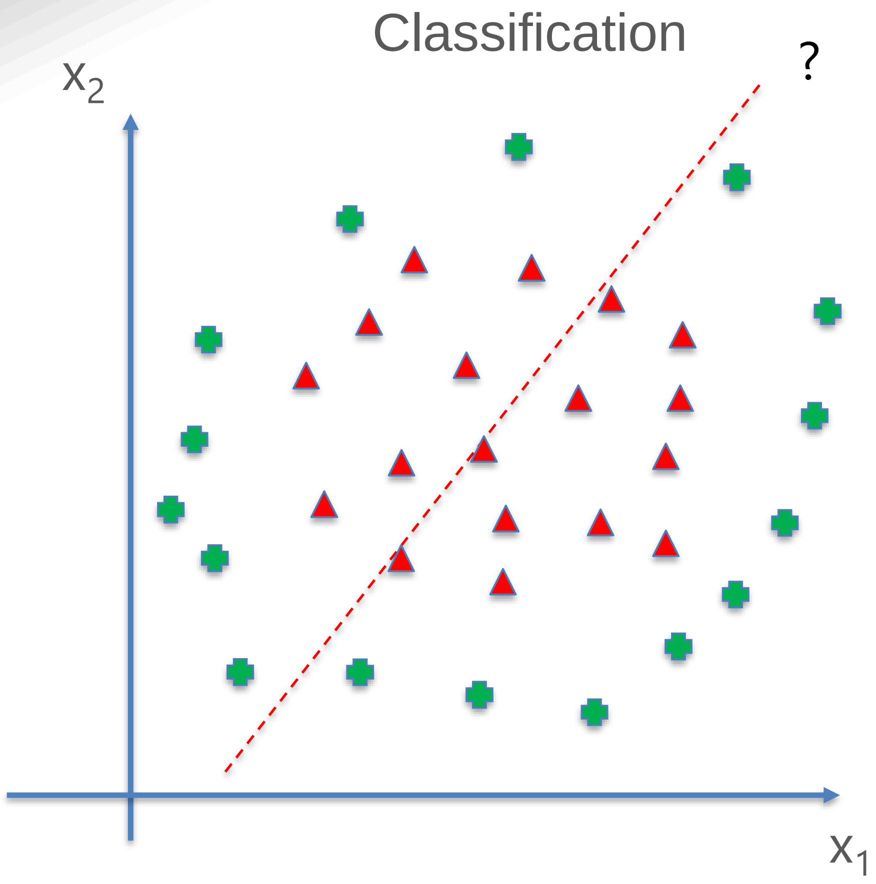
Cons:

- Low modelling power – assumes linear dependency between inputs and outputs.

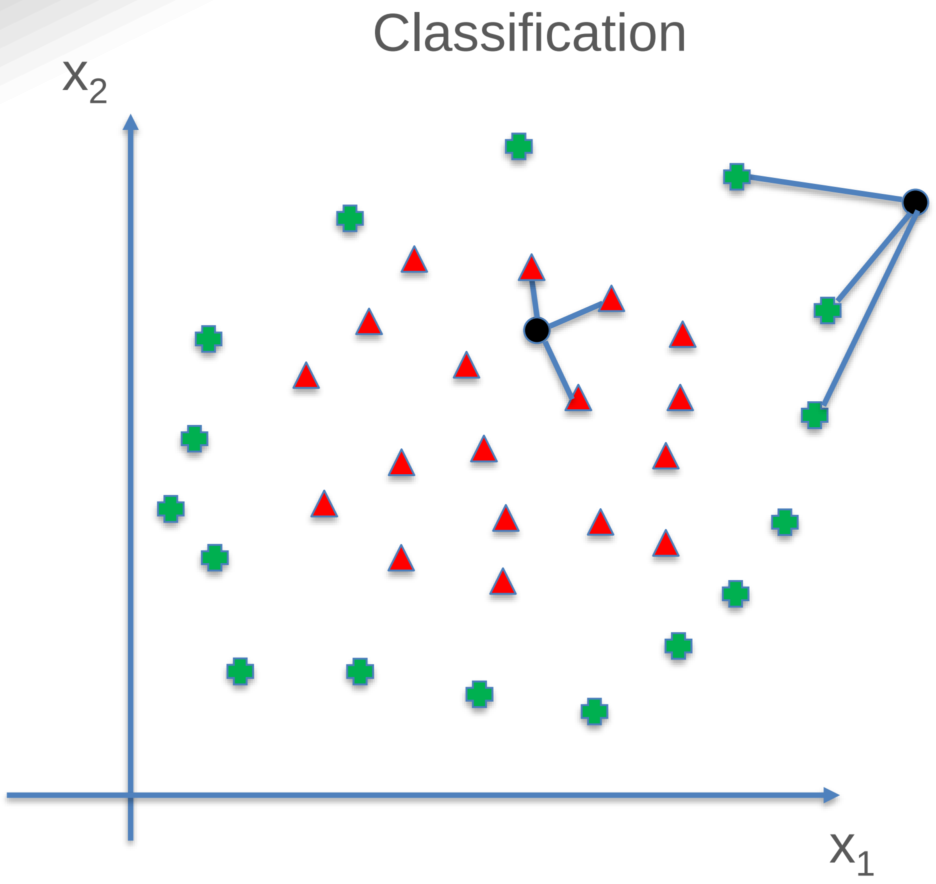
Linear models: limitations



Linear models: limitations



K Nearest Neighbors: KNN



KNN algorithm

Input:

1. **Training data** as set of pairs (x_i, y_i), $i = 1... N$,
   New data point $x^* \in X^*$ to be classified
2. **Distance metric** $d: X^* \times X^* \mapsto R$ that measures how different two points are.

Begin:

1. Select an index set I with least $d(x_i, x^*)$, i in I

2. For the classification task:

assign to x^* most frequent label in $\{y_i \mid \forall$ i in S$\}$

For the regression task:

assign to x^* mean of set $\{y_i \mid \forall$ i in S$\}$

End

KNN: pros and cons

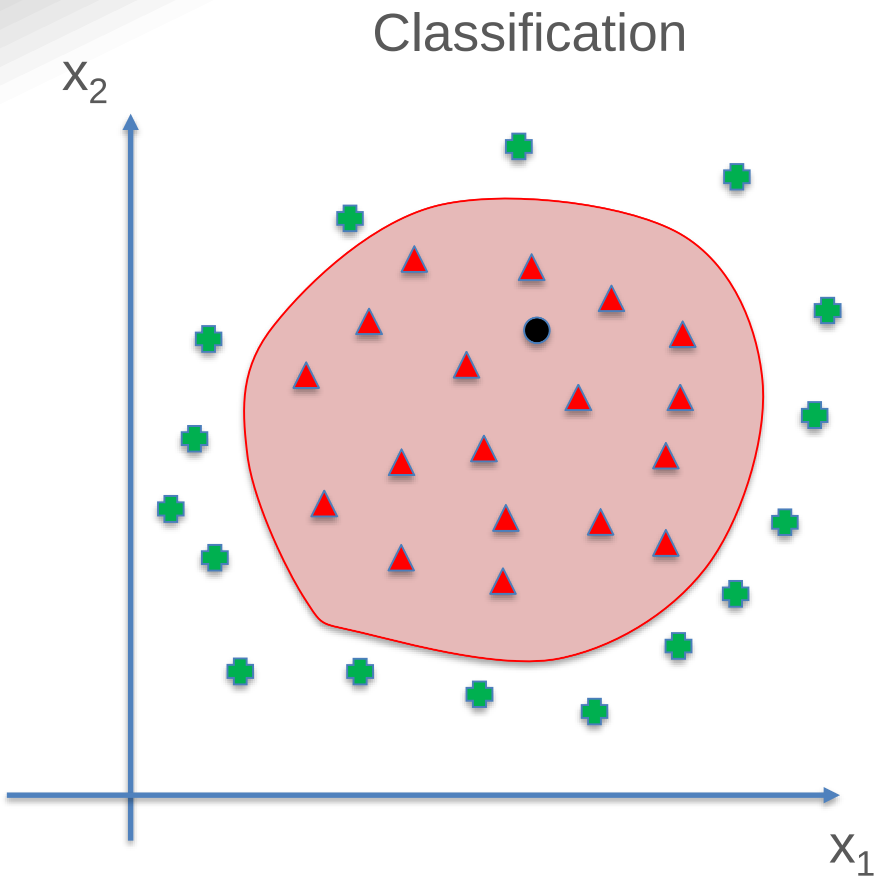
Pros:

- Can represent non – linear relations
- No fitting procedure!
- Can be fast to evaluate for small dimensional features

Cons:

- Can be slow for large feature vectors
- Requires whole dataset for predictions
- Susceptible to noise, for small number of neighbors

Kernel SVM



Kernel Support Vector Machines

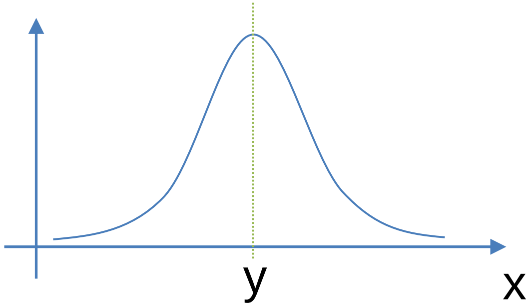
Definition of the model:

For regression: $$f(x) = b + K(x)^T w = b + \sum_{i=1\ldots m} k(x, x_i) w_i$$

For binary classification: $$f(x) = sign(b + \sum_{i=1\ldots m} k(x, x_i) w_i)$$

Kernel function: $k : X^* \times X^* \to R$ (dis) similarity between inputs.

A popular choice: $$k_{RBF}(x, y) = e^{-\gamma ||x-y||_2^2}$$



Schölkopf, Bernhard, and Alexander J. Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press, 2002.

Kernel SVM: pros and cons

Pros:

- Well studied class of models
- Relatively small number of hyperparameters
- Clear control over complexity of the model

Cons:

- Requires subset of dataset for predictions
- Training time grows quadratically with increase of dataset size
- Black box model

Decision Trees

Internal nodes: Nodes where the decision branching happens. One feature is tested in every decision node.

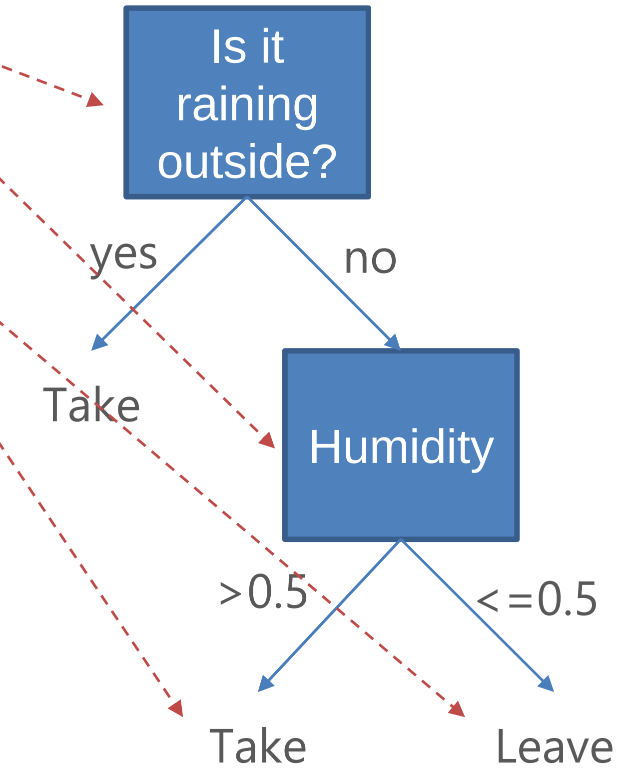
Leaf nodes: define outputs of the decision process that ends in their branch.

Regression task: leaf nodes yield real numbers

Classification task: leaf nodes yield category

Some features might not be used in the decision tree.

Take umbrella?



Decision Trees: pros and cons

Pros:

- Well suited for big data – evaluation time does not depend on size of dataset!
- Can capture non – linear dependencies
- Can be analyzed and interpreted by humans

Cons:

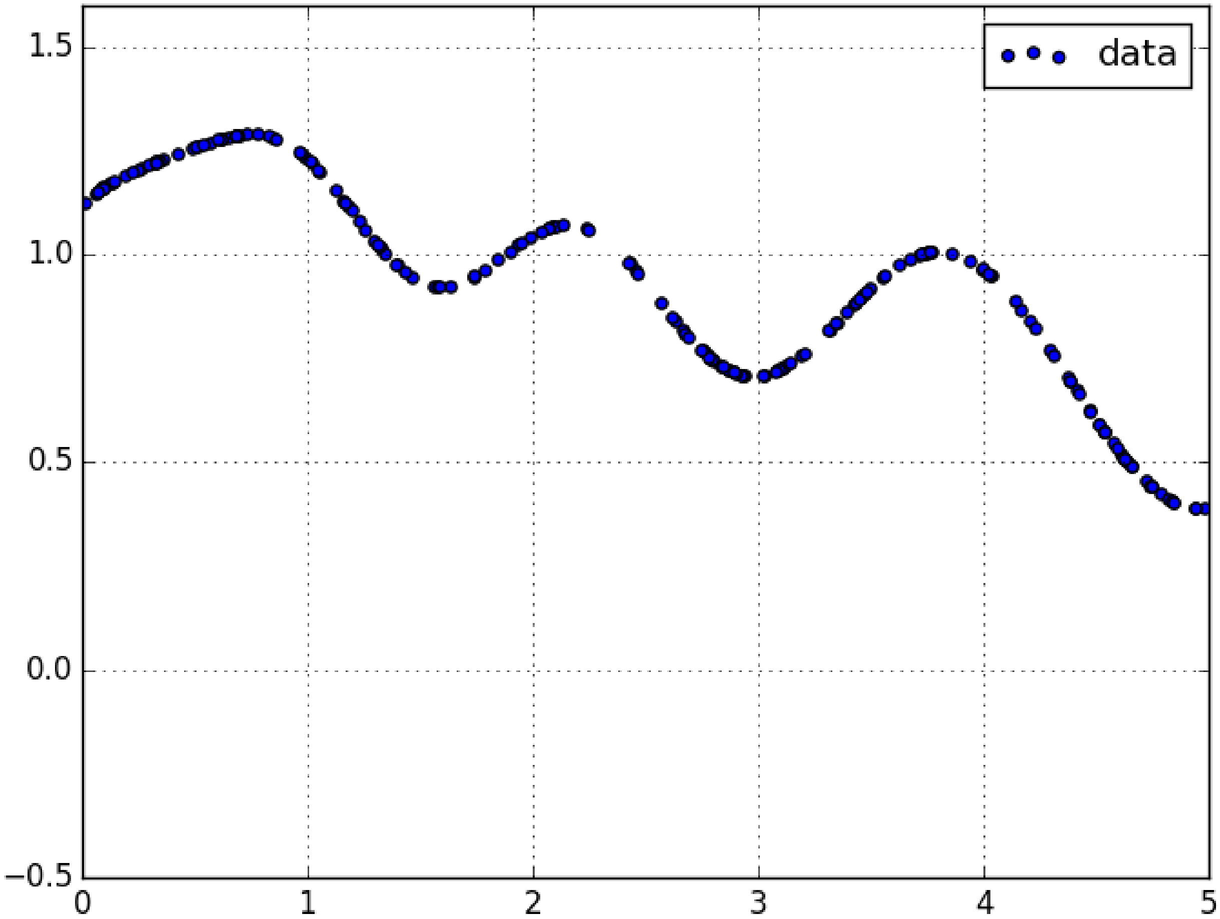
- Performance not as good as for other methods (eg Kernel SVM) in black box setting
- Performance depends on training heuristic used

Boosting

Add models iteratively at feature space locations where the ensemble does not perform well.

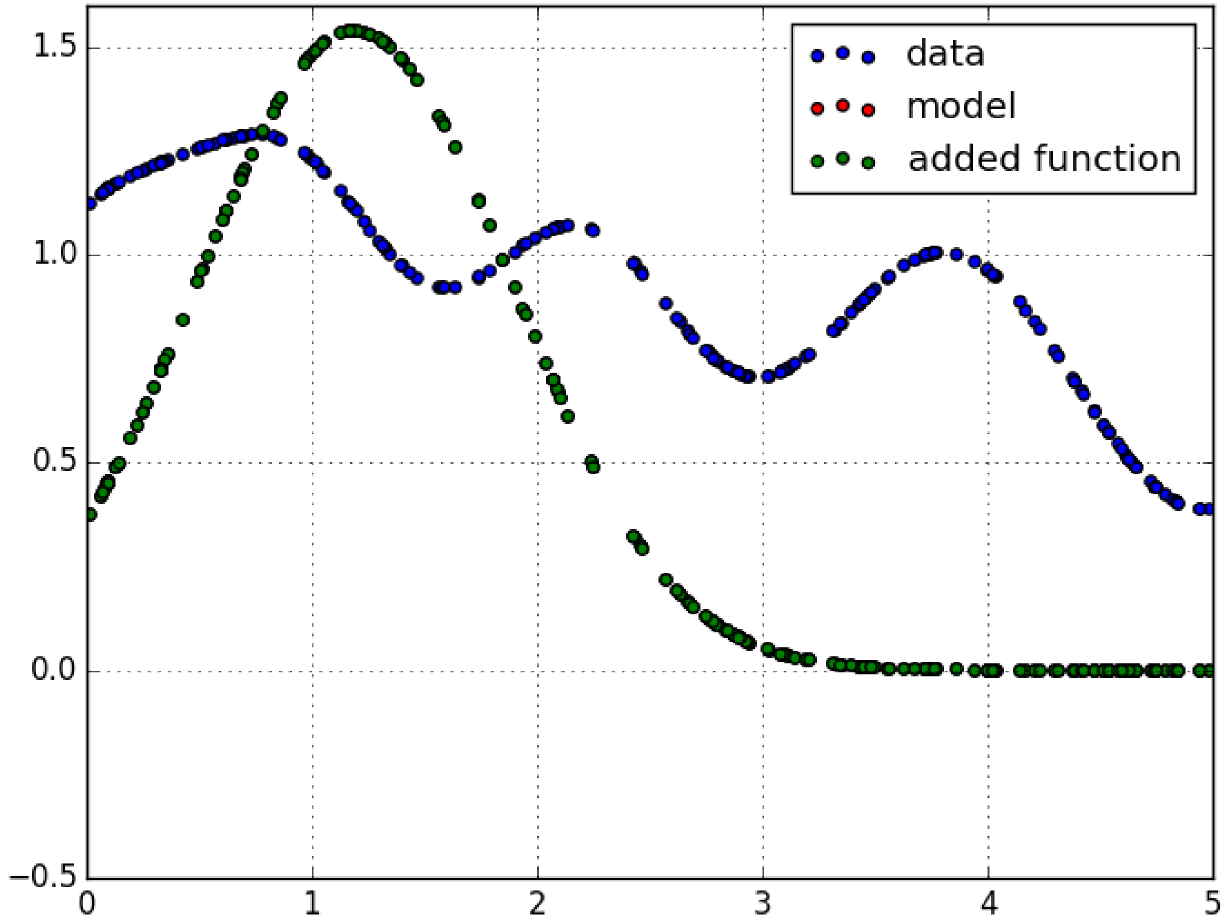
Example: ensemble of Gaussians in 1D.

Boosting



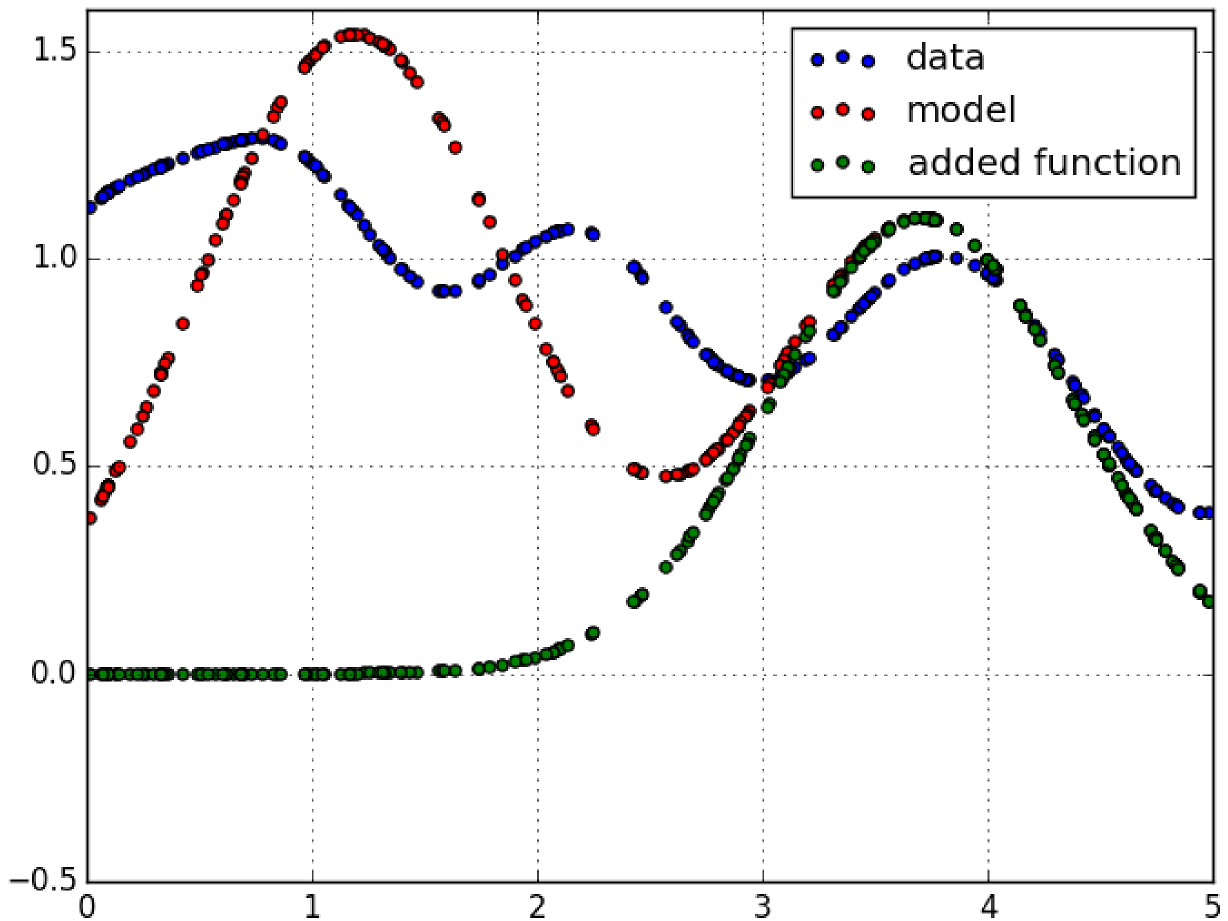
Use boost_gaussian.py to reproduce plots

Boosting



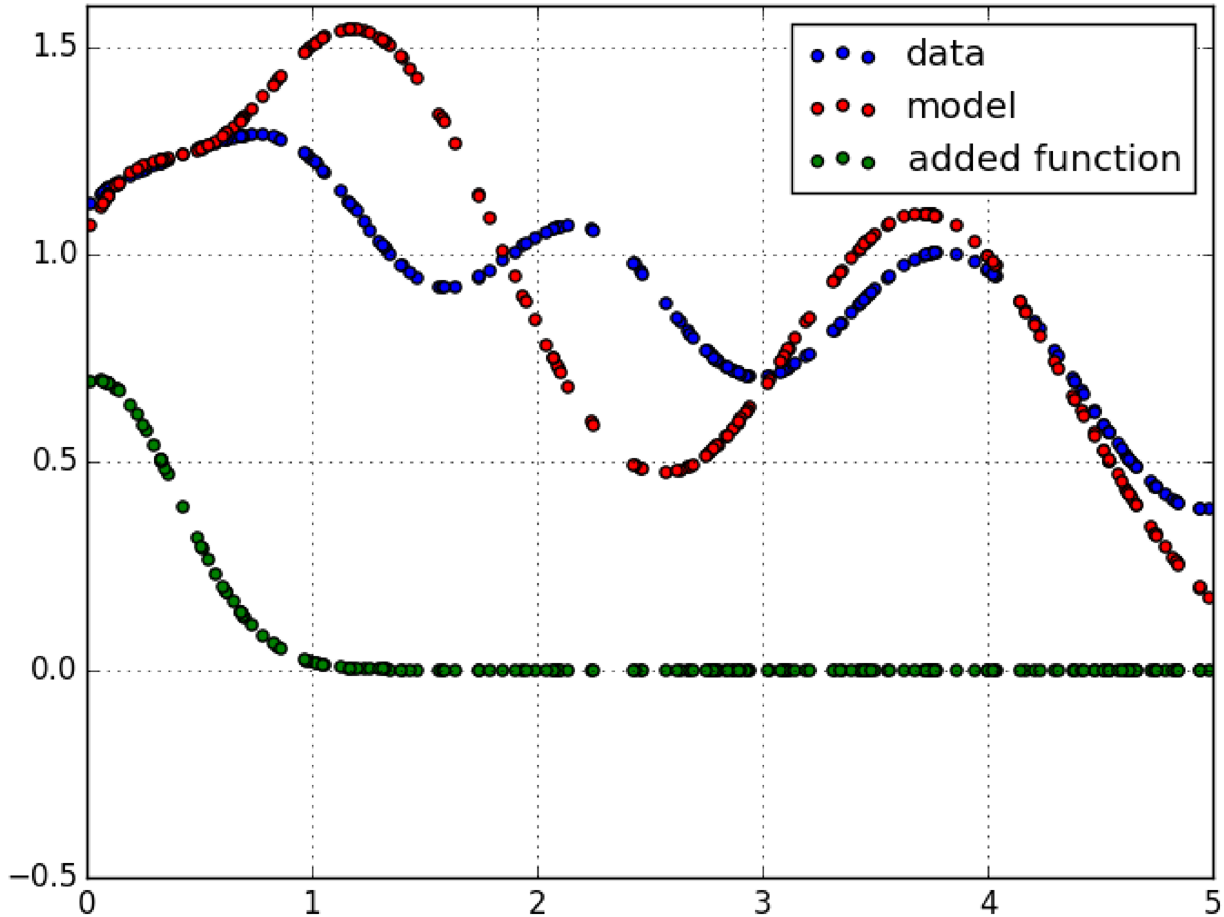
Use boost_gaussian.py to reproduce plots

Boosting



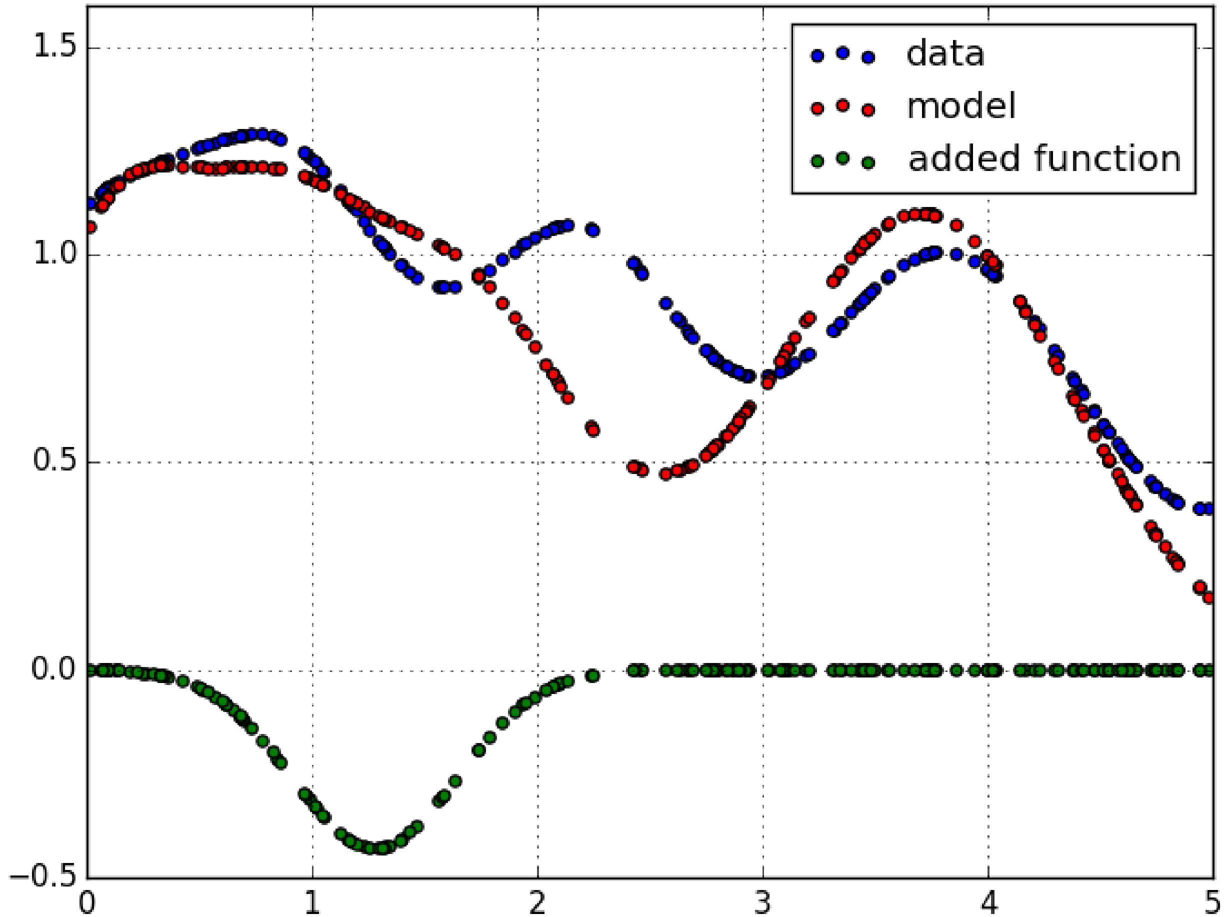
Use boost_gaussian.py to reproduce plots

Boosting



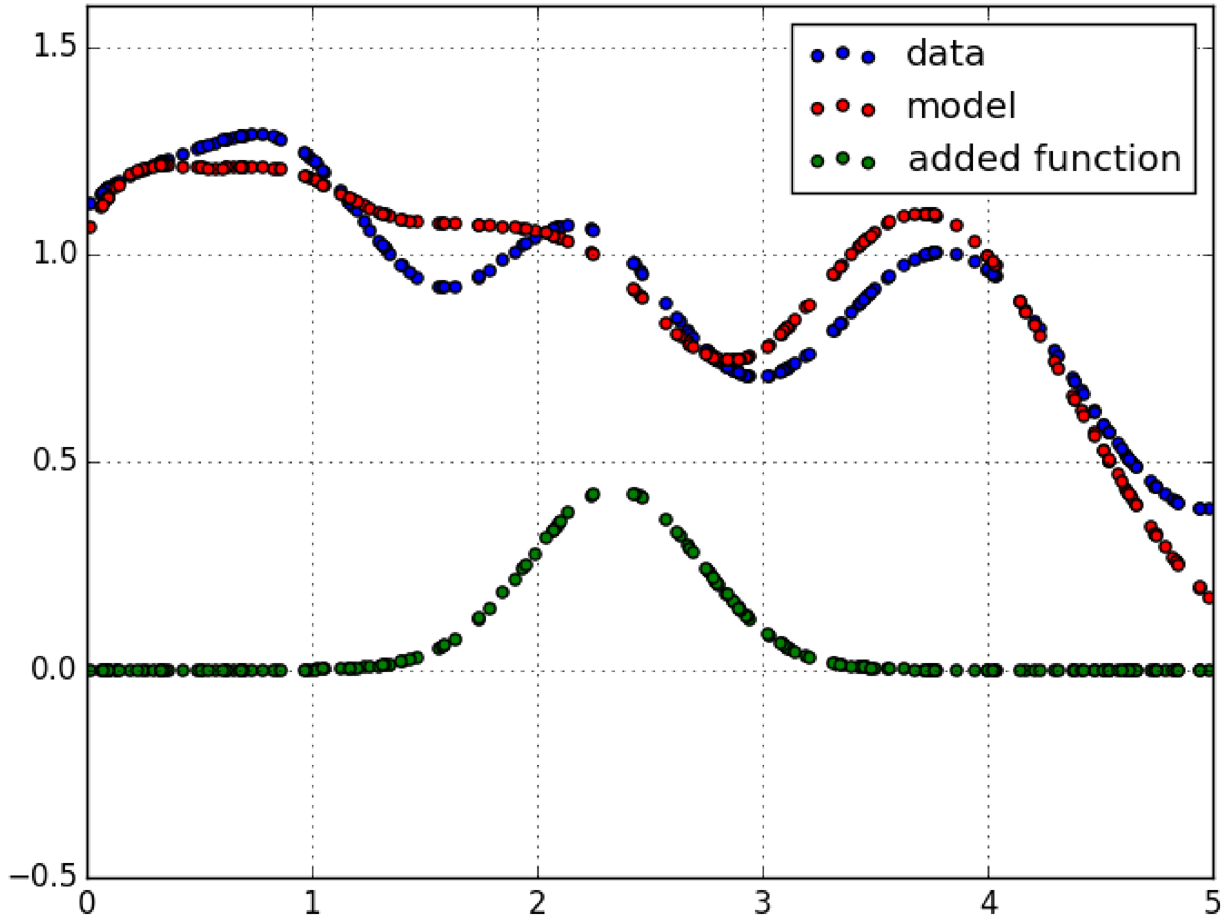
Use boost_gaussian.py to reproduce plots

Boosting



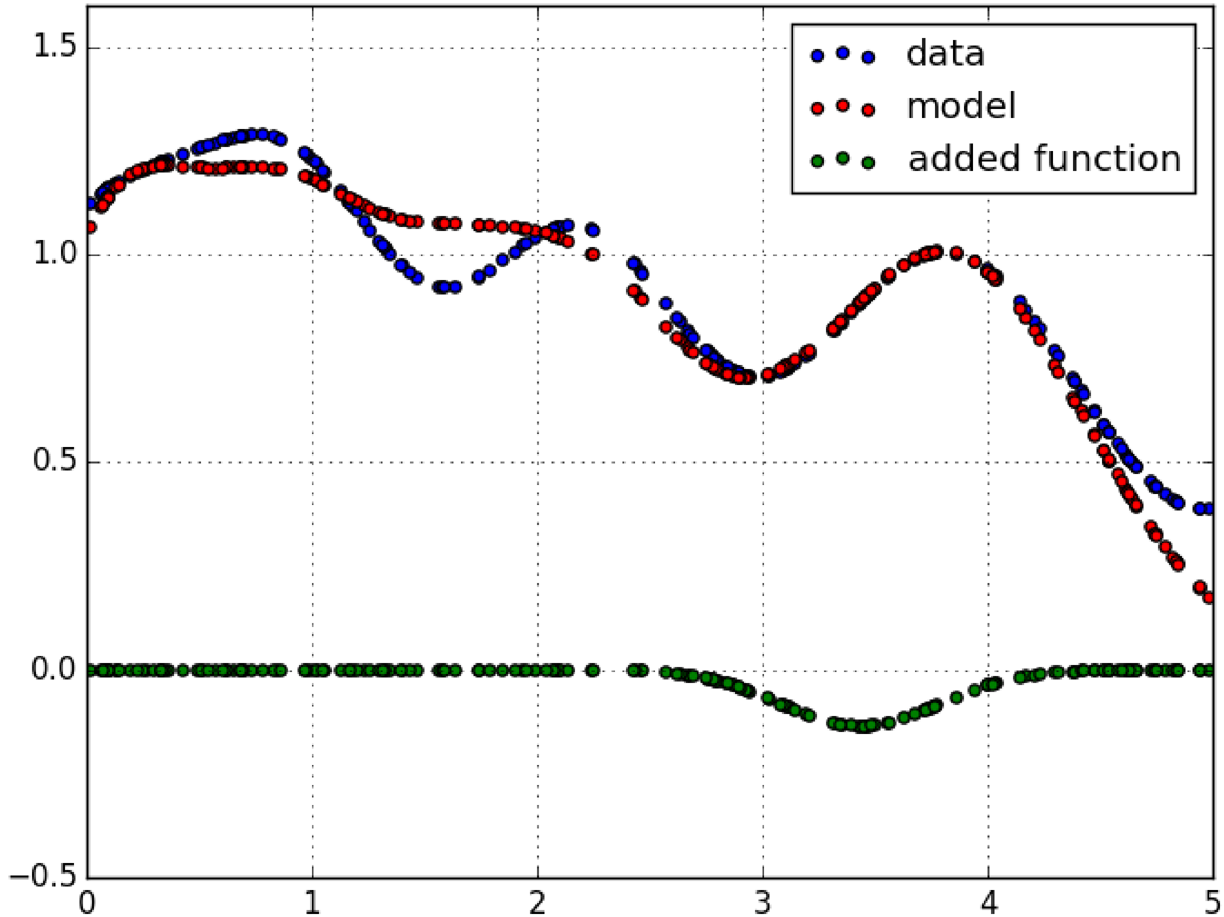
Use boost_gaussian.py to reproduce plots

Boosting



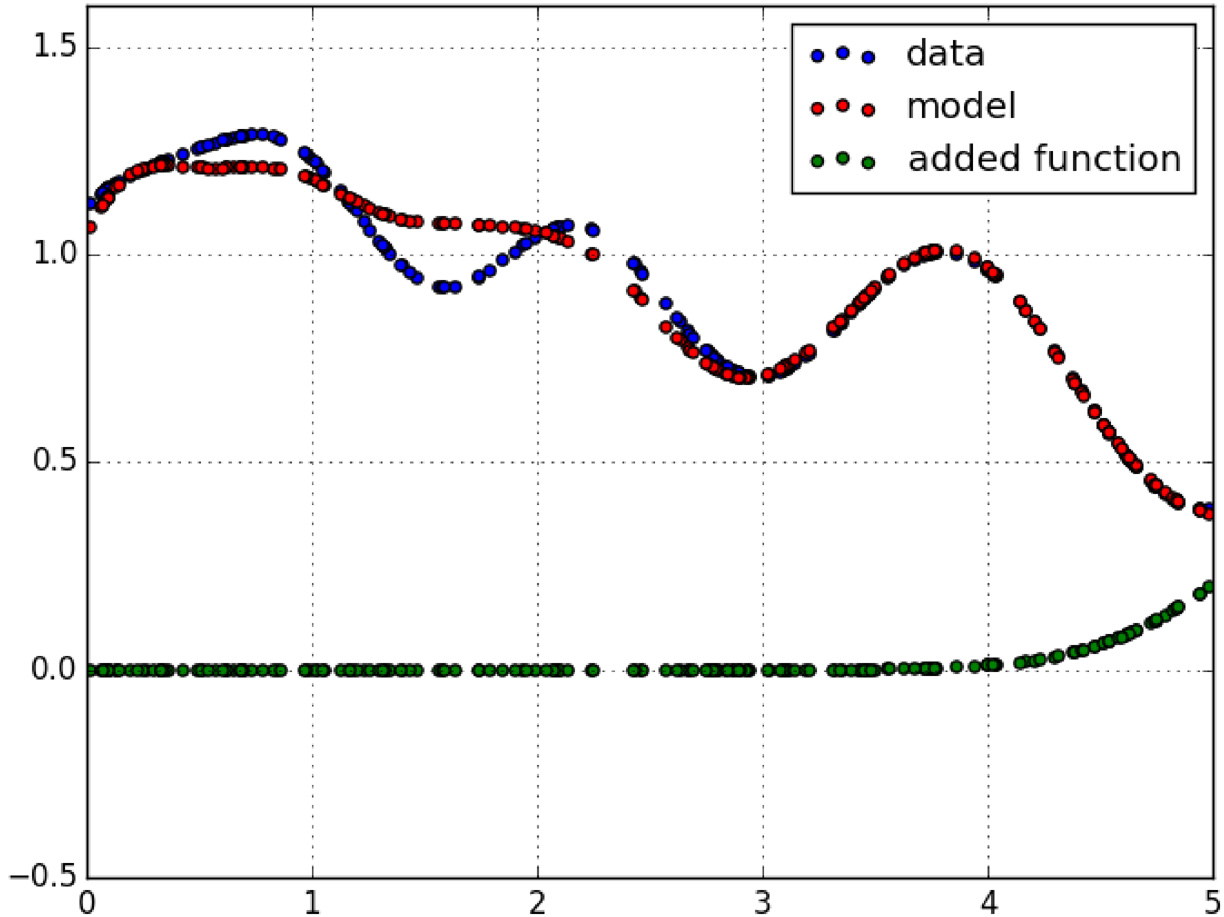
Use boost_gaussian.py to reproduce plots

Boosting



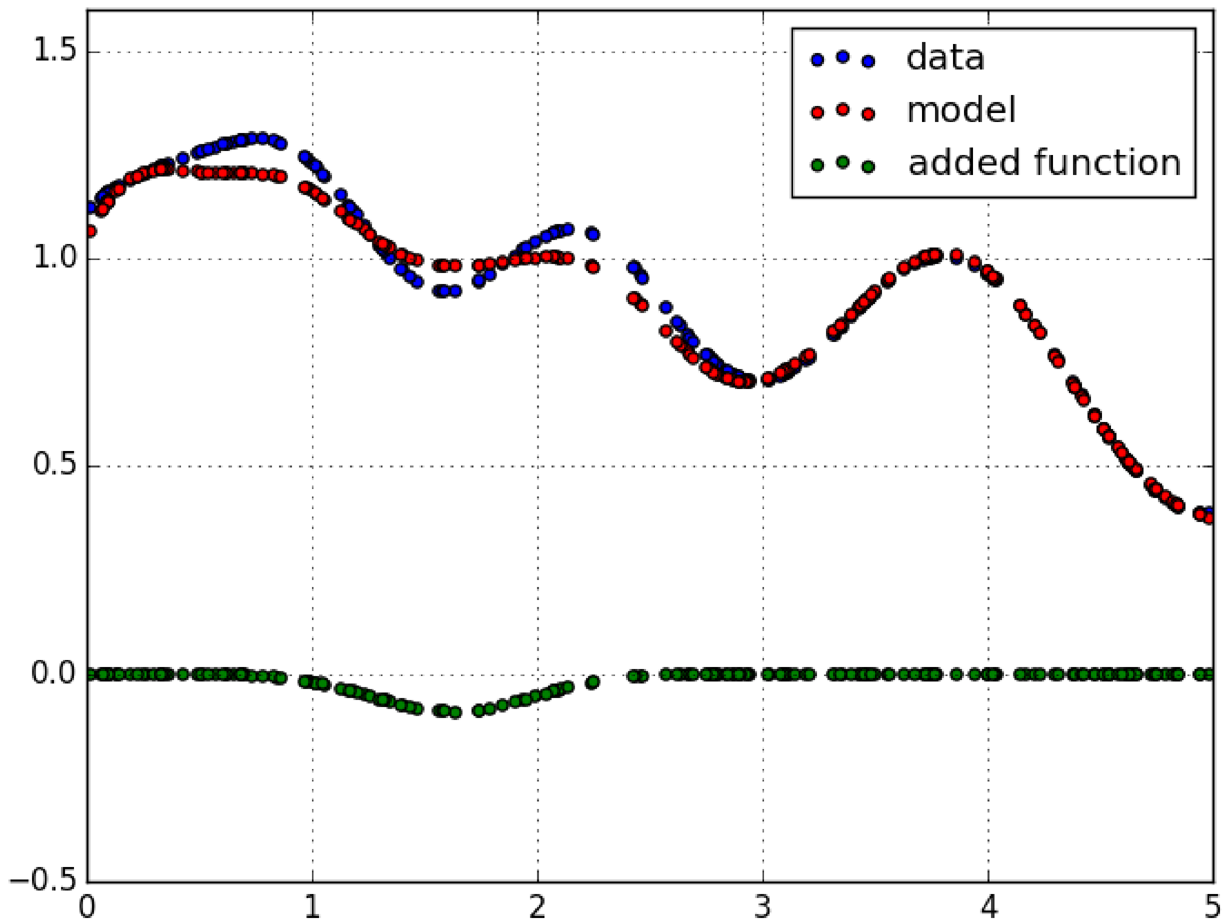
Use boost_gaussian.py to reproduce plots

Boosting



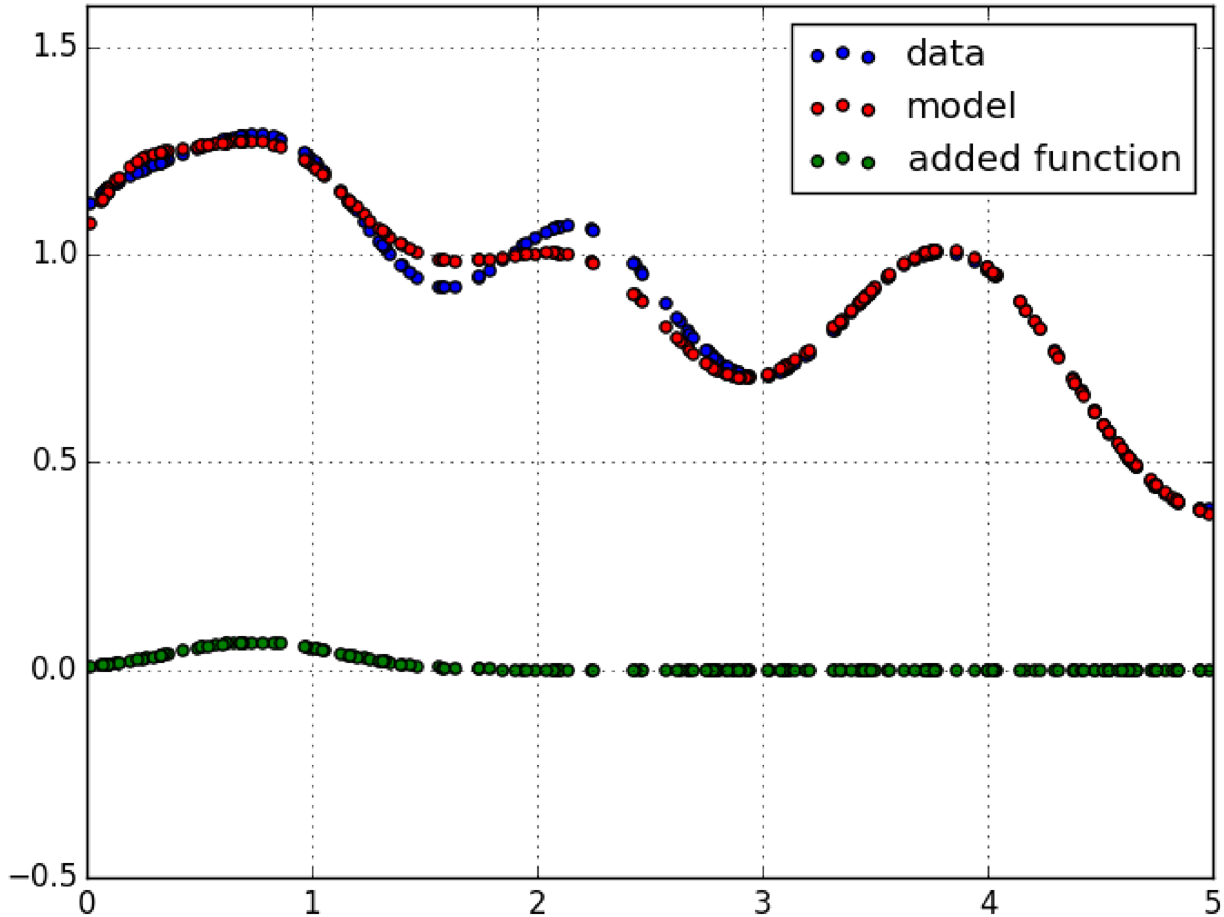
Use boost_gaussian.py to reproduce plots

Boosting



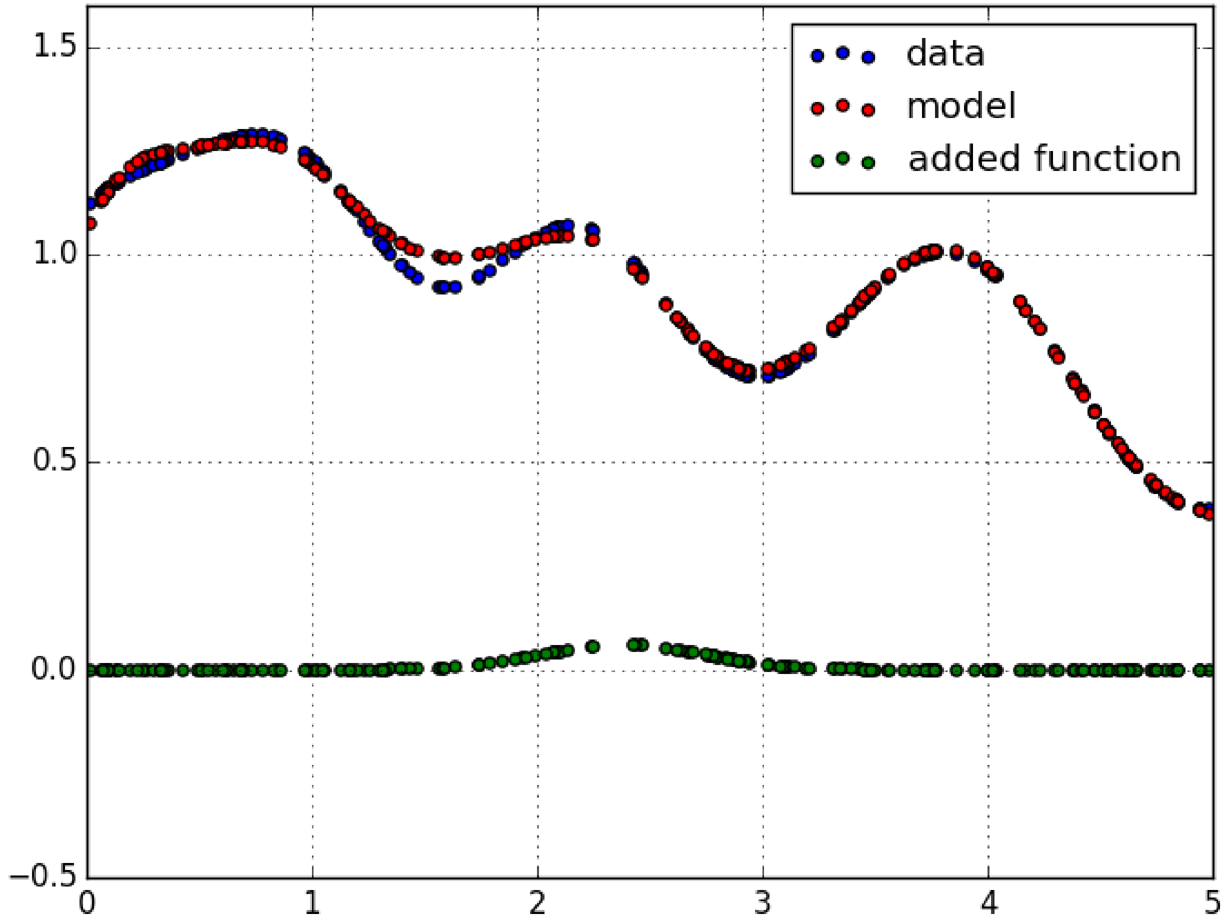
Use boost_gaussian.py to reproduce plots

Boosting



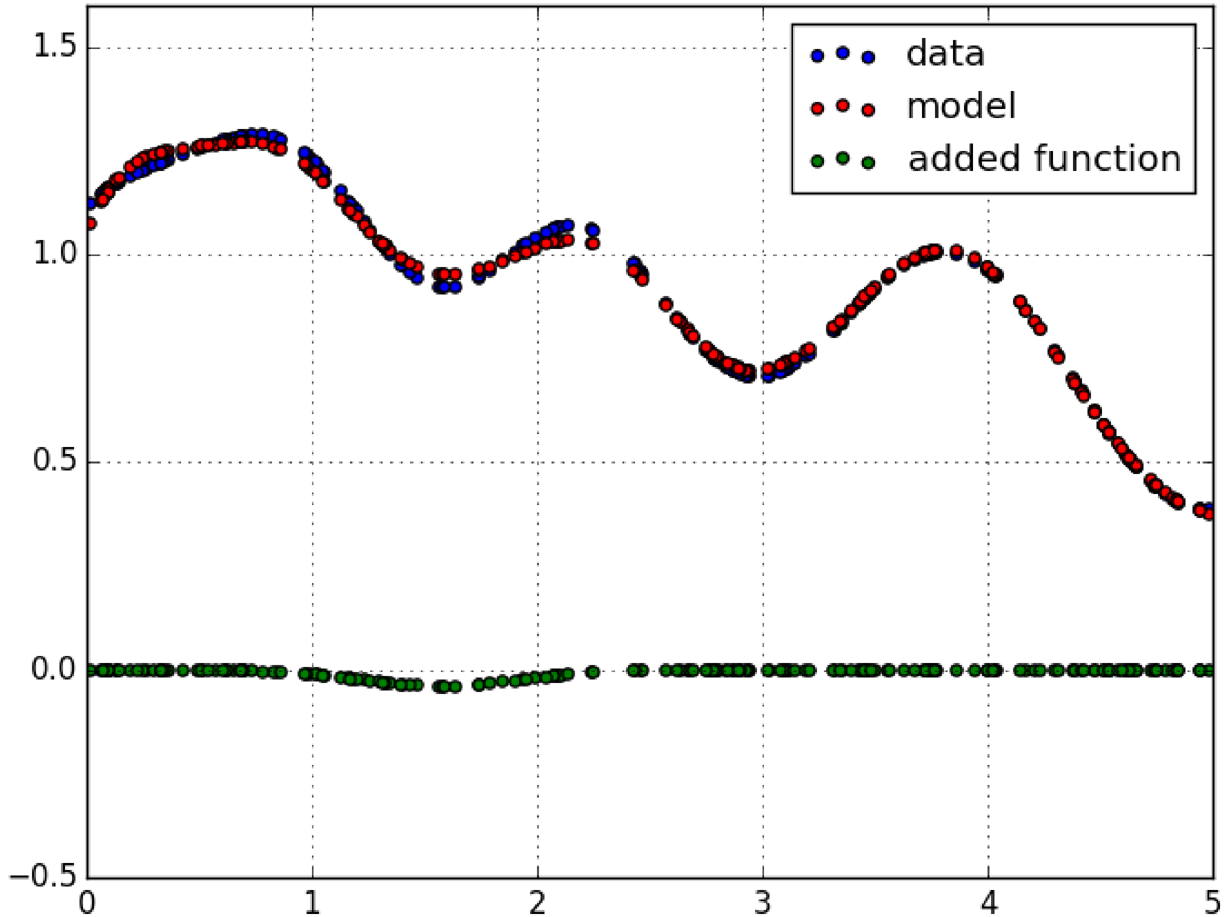
Use boost_gaussian.py to reproduce plots

Boosting



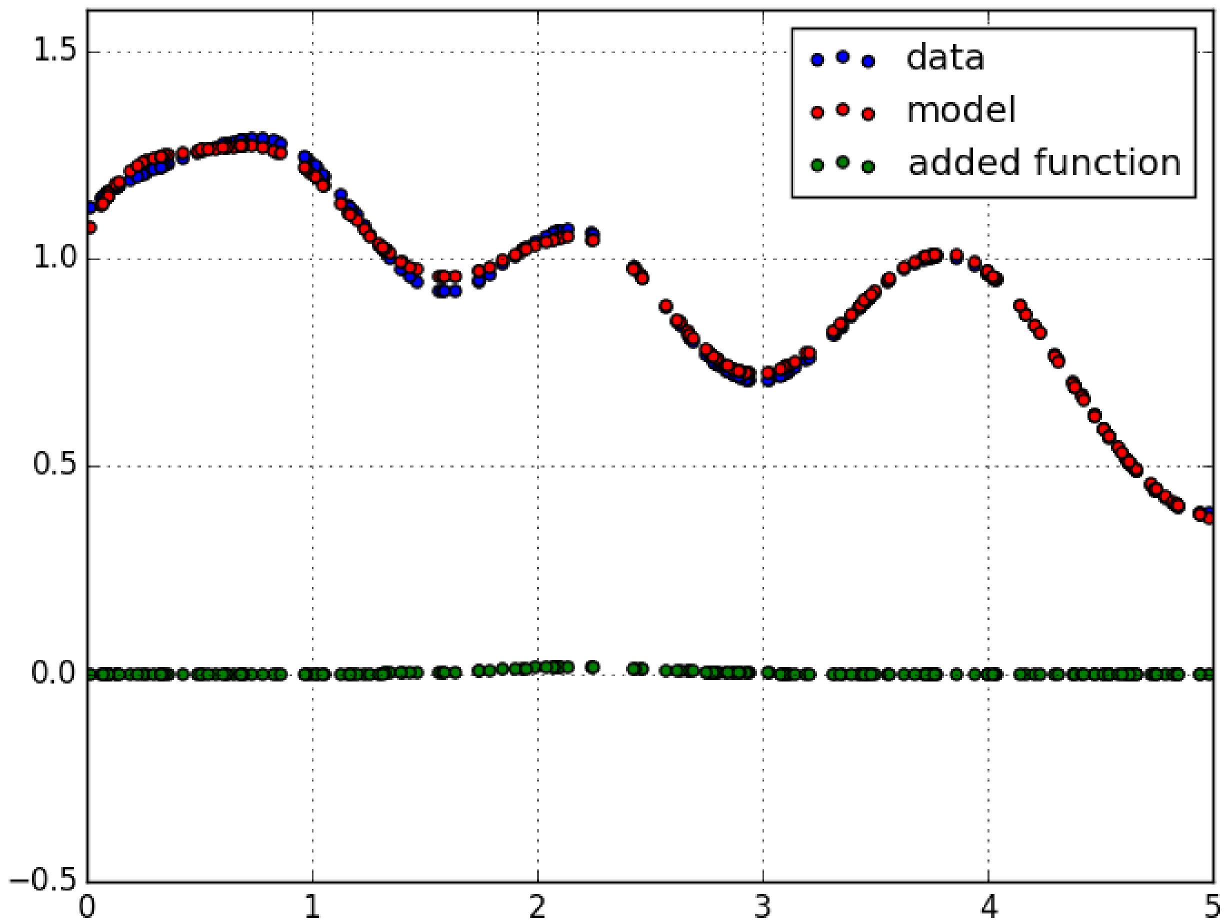
Use boost_gaussian.py to reproduce plots

Boosting



Use boost_gaussian.py to reproduce plots

Boosting



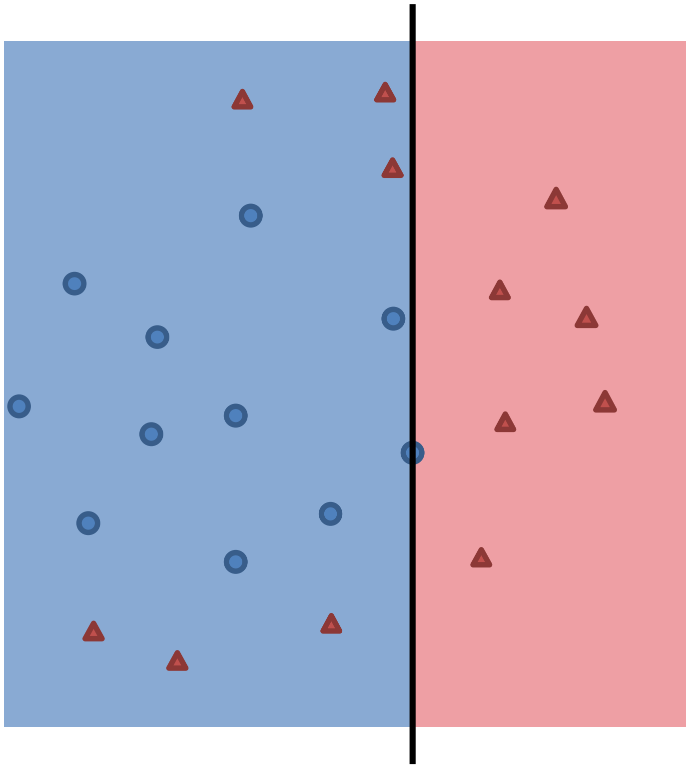
Use boost_gaussian.py to reproduce plots

Boosting: weak learner

Non-linear model of the form

$$f : X \to Y$$

- Chosen only slightly better than random, e.g. models that depend only on one feature.
- This usually implies that such models are easy to compute.



Gentle AdaBoost

Allows to achieve arbitrary accuracy when number of weak learners M can be arbitrarily large.

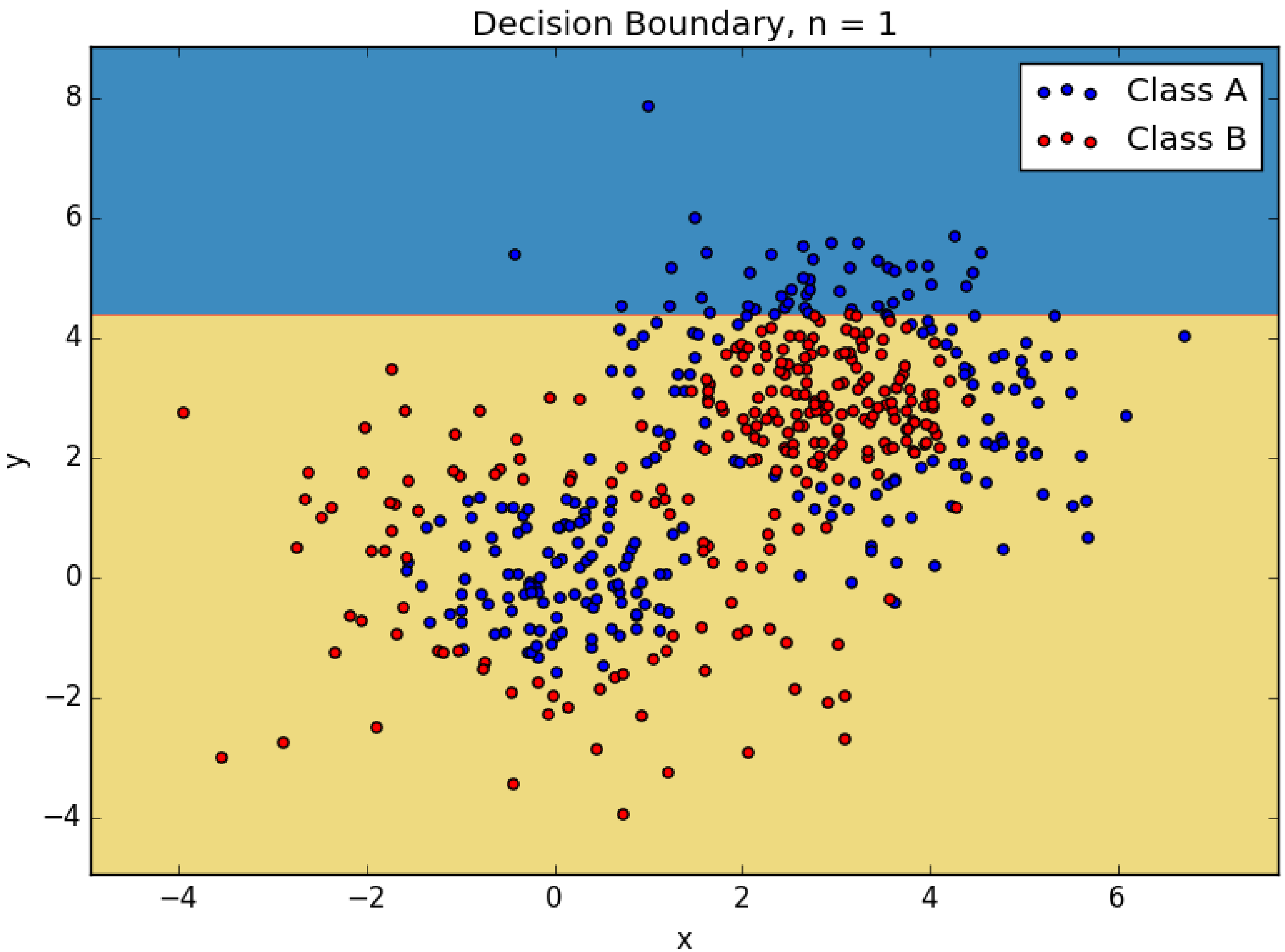
1. Start with weights $w_i = 1/N, i = 1, 2, ..., N, F(x) = 0$
2. Repeat for $m = 1, 2, ..., M$:

   (a) Find $f_m = argmin_{f_m \in F_m} \sum_{i=1...n} w_i (f(x_i) - y_i)^2$

   (c) Update w_i using the formula: $w_i \leftarrow w_i \exp\left(-y_i f_m(x_i)\right)$

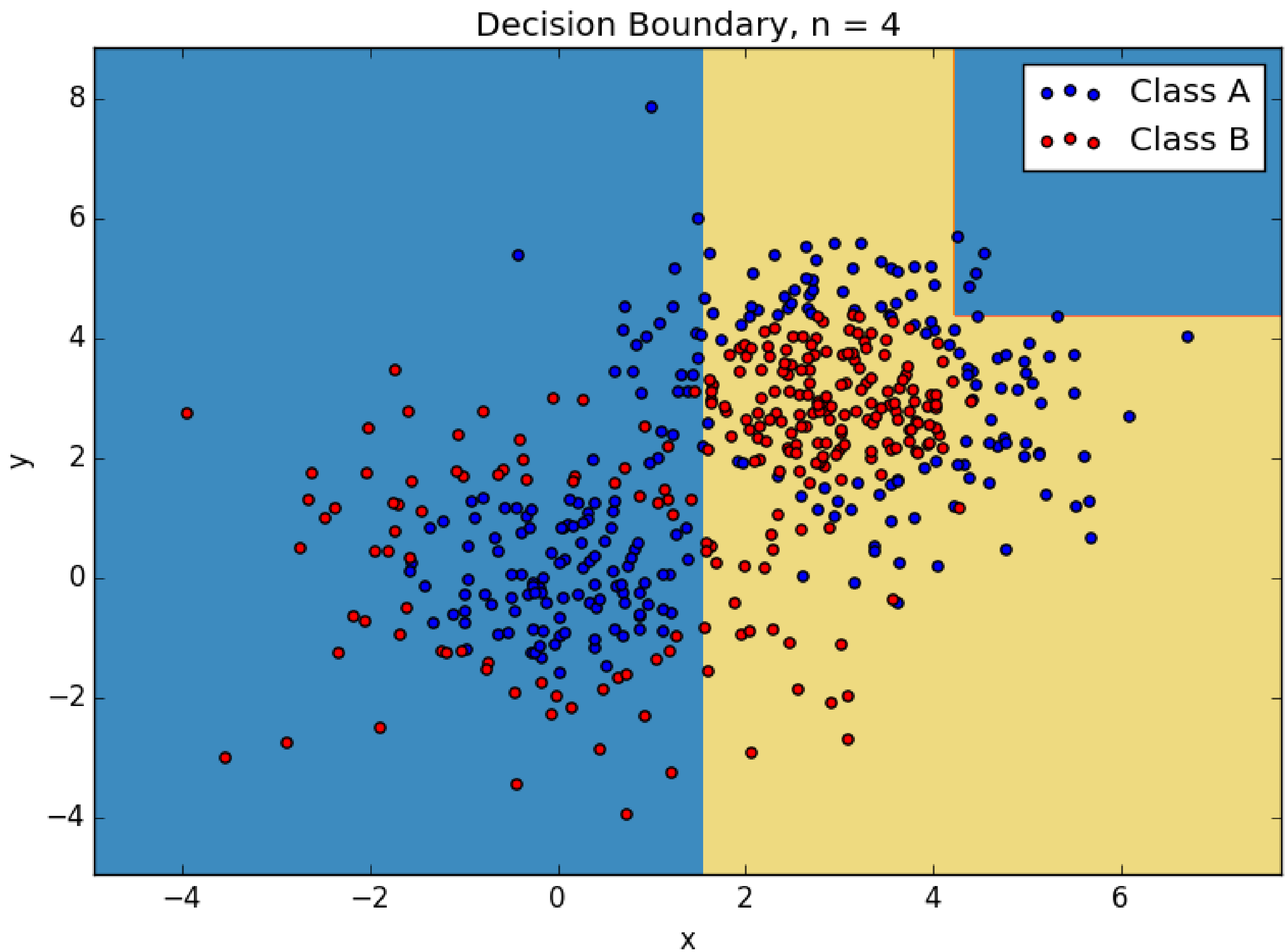
3. Output the classifier $sign\left[F(x)\right] = sign\left[\sum_{m=1}^{M} f_m(x)\right]$

Gentle AdaBoost



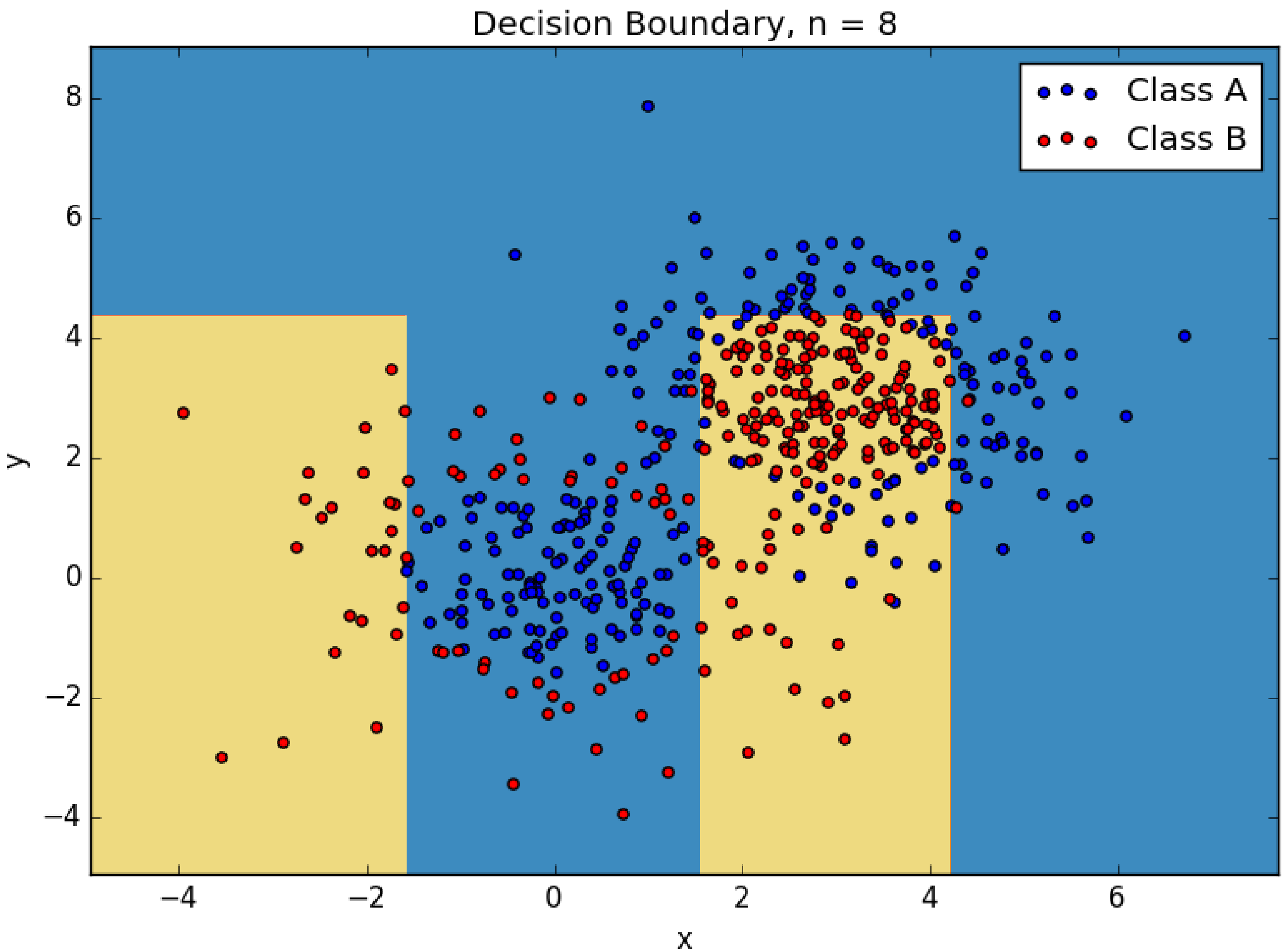
Use adaboost.py to reproduce plots

Gentle AdaBoost



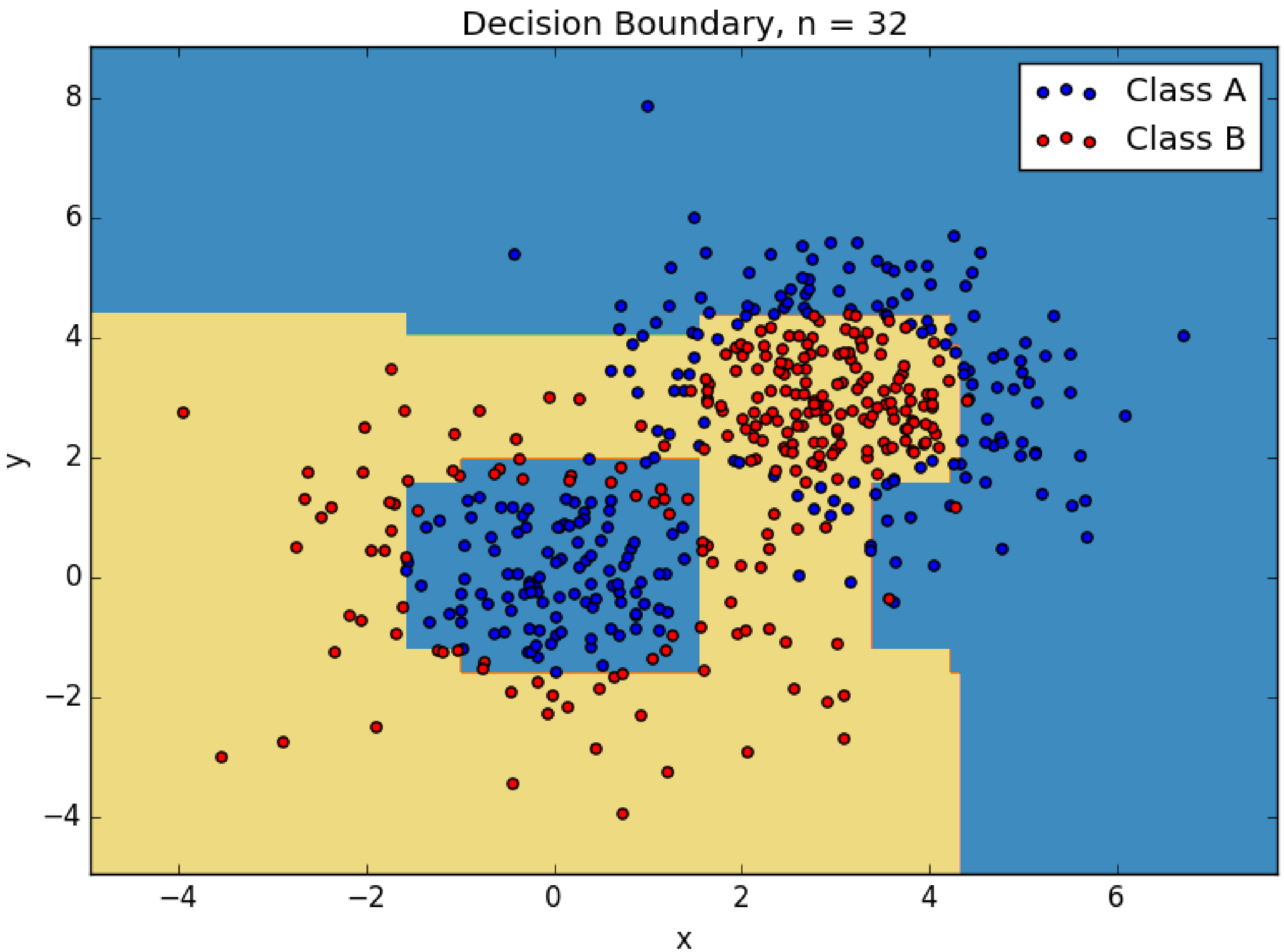
Use adaboost.py to reproduce plots

Gentle AdaBoost



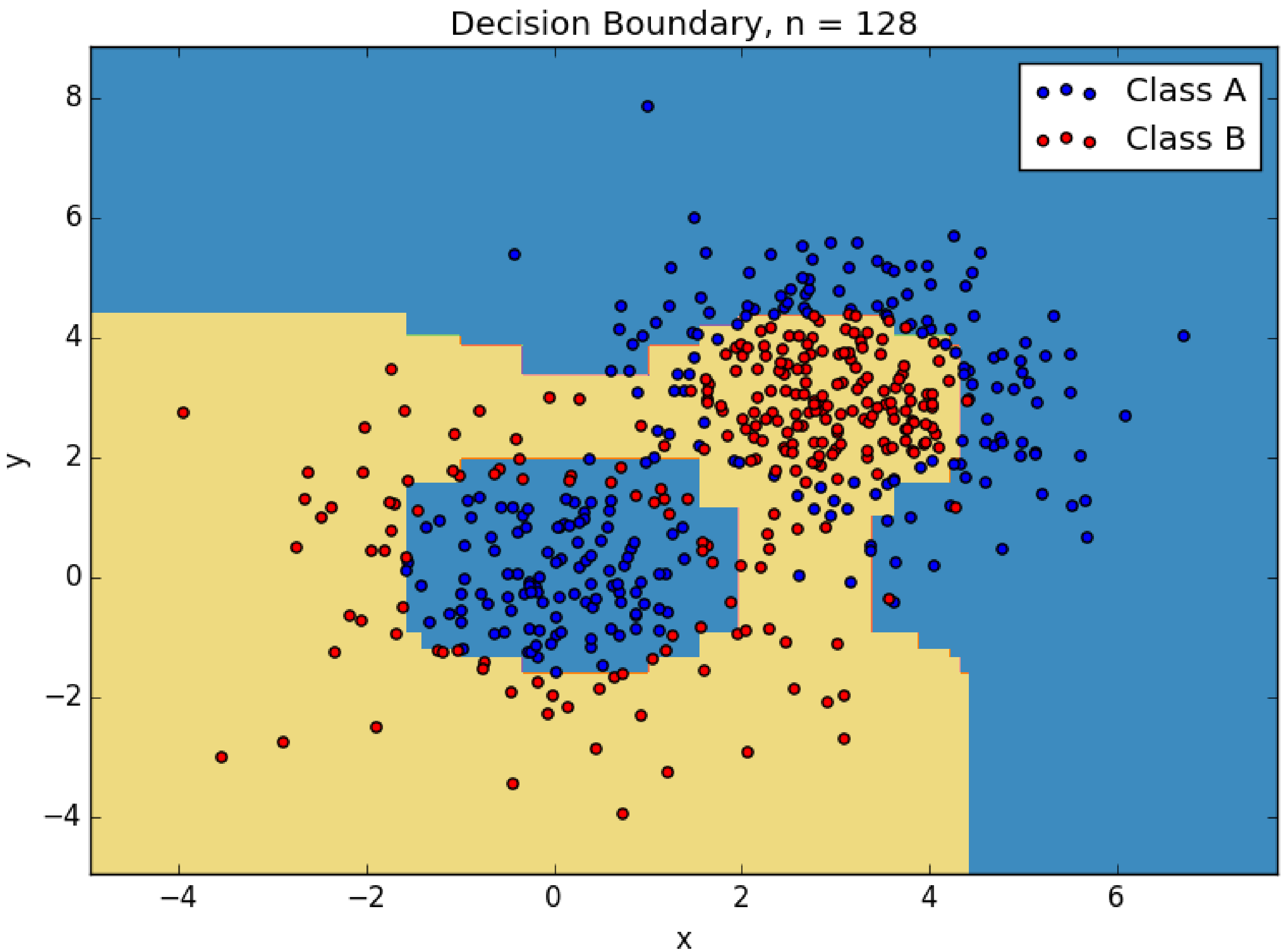
Use adaboost.py to reproduce plots

Gentle AdaBoost



Use adaboost.py to reproduce plots

Gentle AdaBoost



Use adaboost.py to reproduce plots

Literature

- Hastie, T., Tibshirani, R. „Statistical Learning: Linear regression", Stanford, 2016,
- Schölkopf, Bernhard, and Alexander J. Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press, 2002.
- Pfister, H., Blitzstein, J., Kaynig, V. „CS109 Data Science Classification & PCA", Harvard, 2013

These slides are largely based on slides and material from 'Introduction into Data Science' course taught at the Saarland University.