

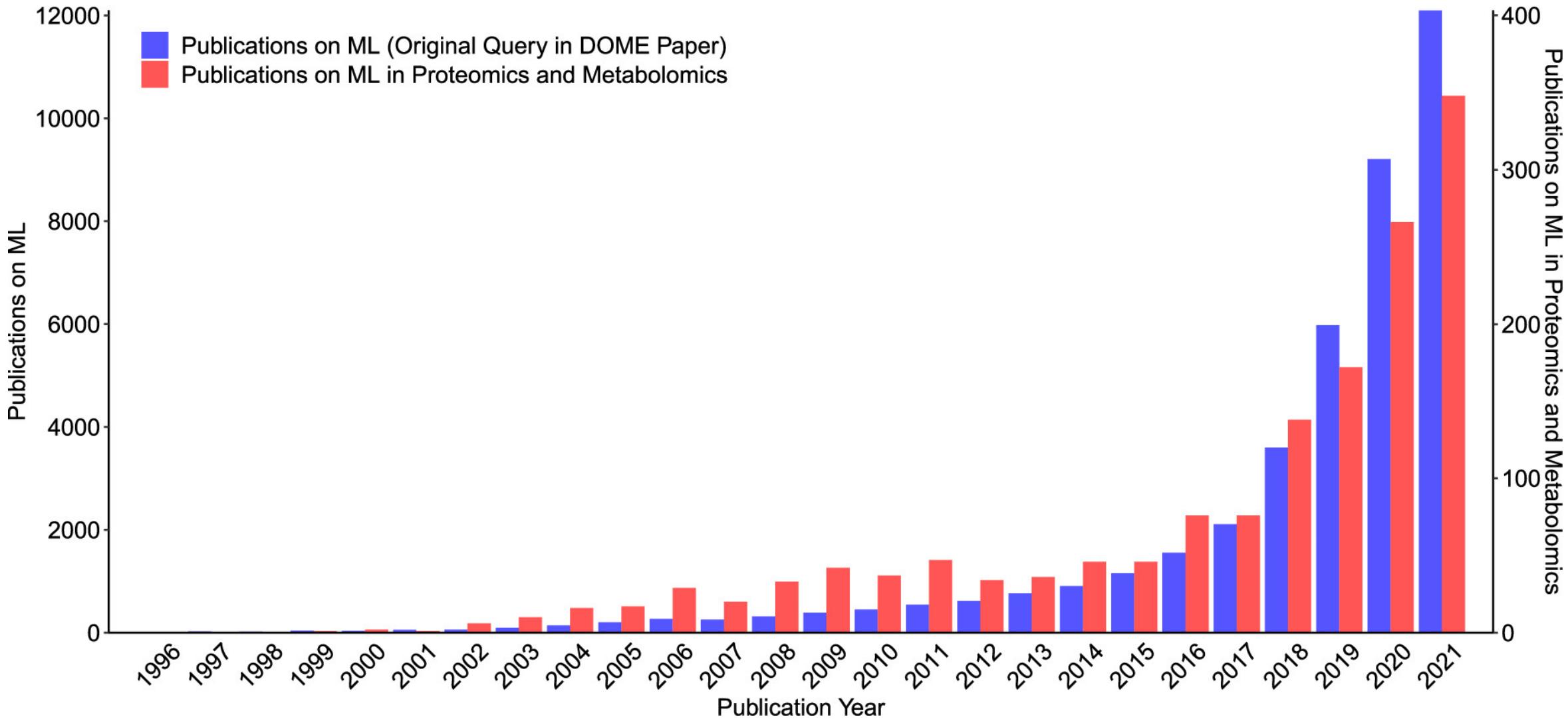
# A tour of machine learning

**Sven Degroeve**

VIB Center for Medical Biotechnology  
UGent Department of Biomolecular Medicine

- What is Machine Learning?
- Linear regression
- Logistic regression
- Hands on: MNIST model fitting
- Model regularization
- Bias / Variance
- Bagging / Boosting
- Intro to deep learning

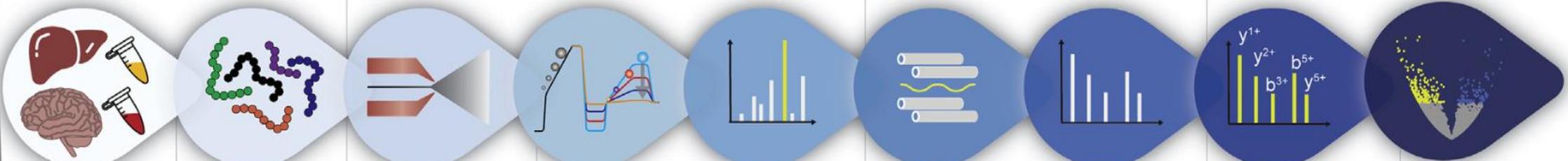
[https://github.com/sdgroeve/EuBIC2022\\_workshop\\_ML](https://github.com/sdgroeve/EuBIC2022_workshop_ML)



2006

- RTPredict, Pfeifer N...  
Kohlbacher O, 2007;
- Elute, Moruz L...  
Käll L, 2010; etc.

Sample      Digestion      Liquid Chromatography      Ion Mobility      MS1      Peptide Fragmentation      MS2      Peptide/Protein Identification      Protein Quantification



2020

- DeepDigest, Yang J...Fu Y, 2020;
- Global Targeting, Müller JB...Mann M, 2020;
- CCS Prediction, Meier F...Mann M, 2020;

- pDeep, Zhou X...Zhang Z, 2017; etc.
- Prosit, Gessulat S...Wilhelm M, 2019;
- DeepMass:Prism, Tiwary S...Cox J, 2019;
- PredFull, Liu K...Tang H, 2020;

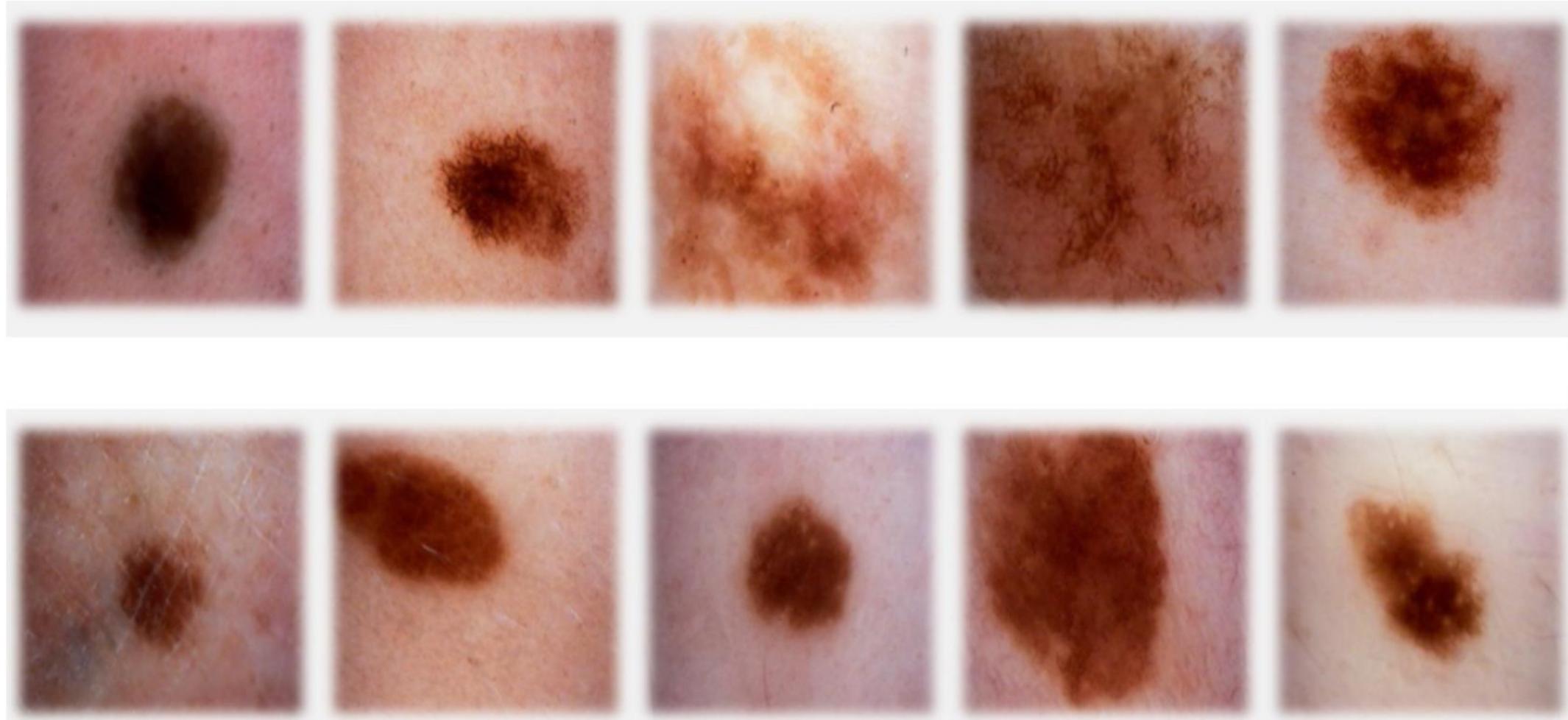
- DeepNovo, Tran NH...Li M, 2017; etc.

- DIA-NN, Demichev V...M Ralser, 2020;

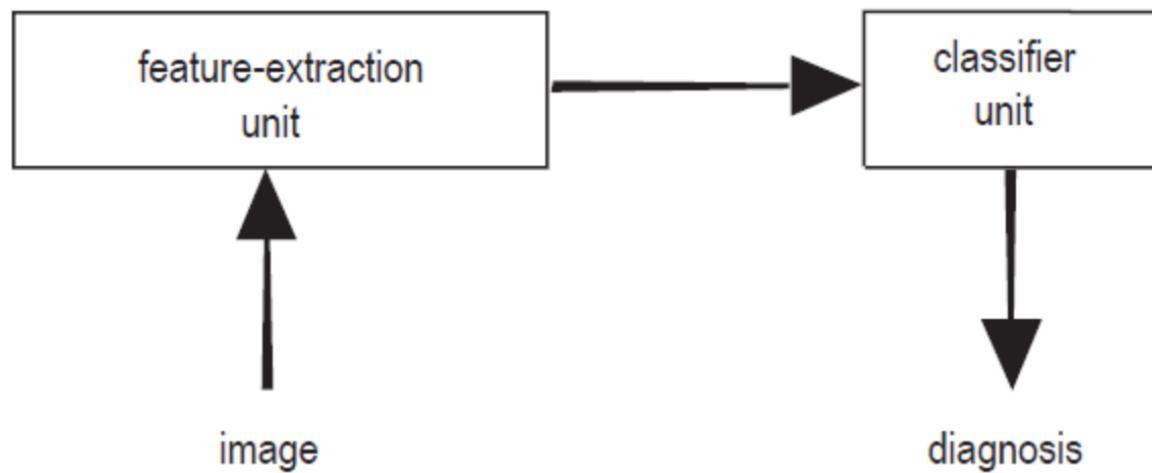
- Quandenser, The M...Käll L, 2020;

Machine Learning      Deep Learning

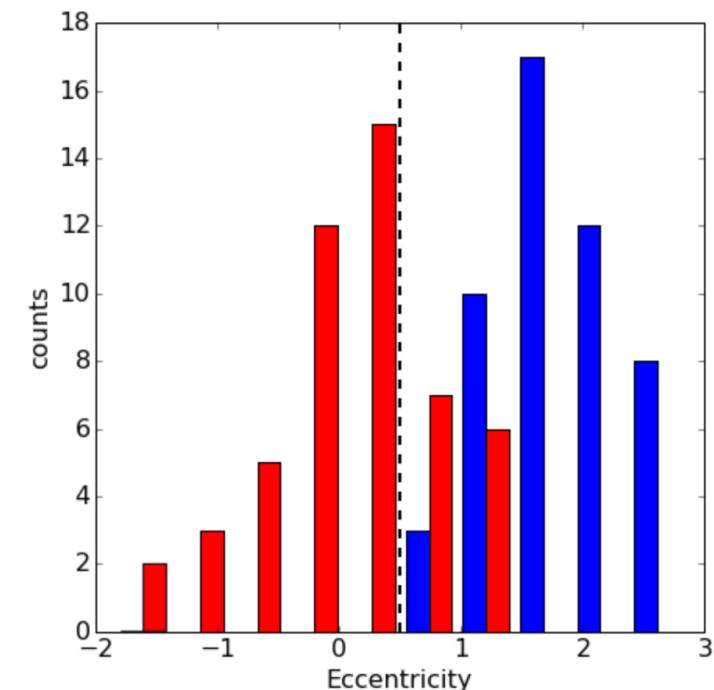
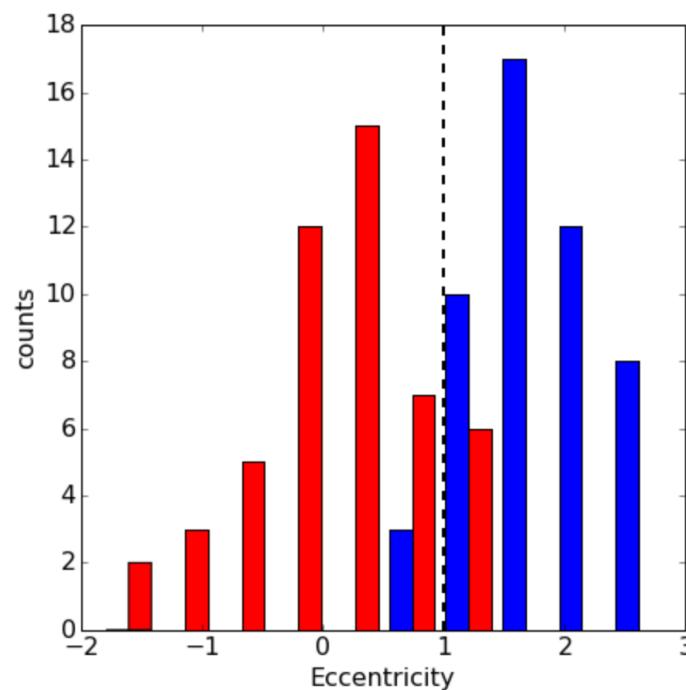
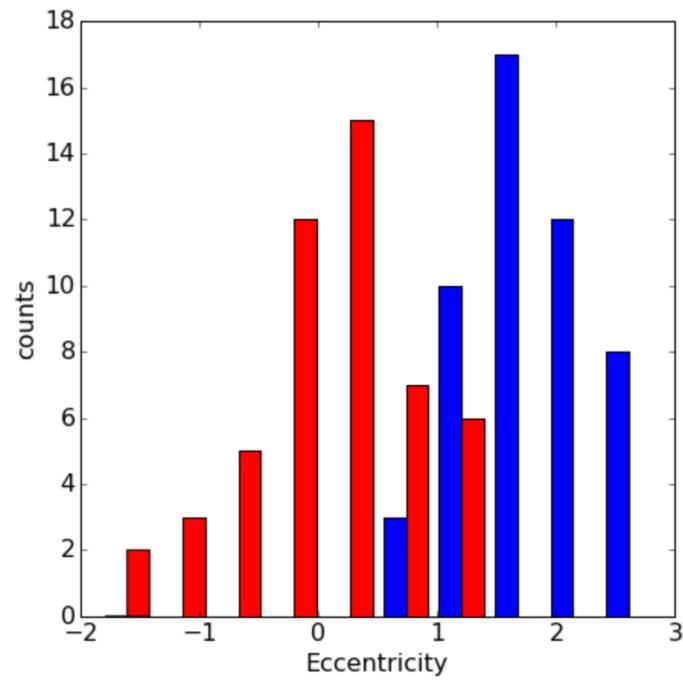
# The dataset



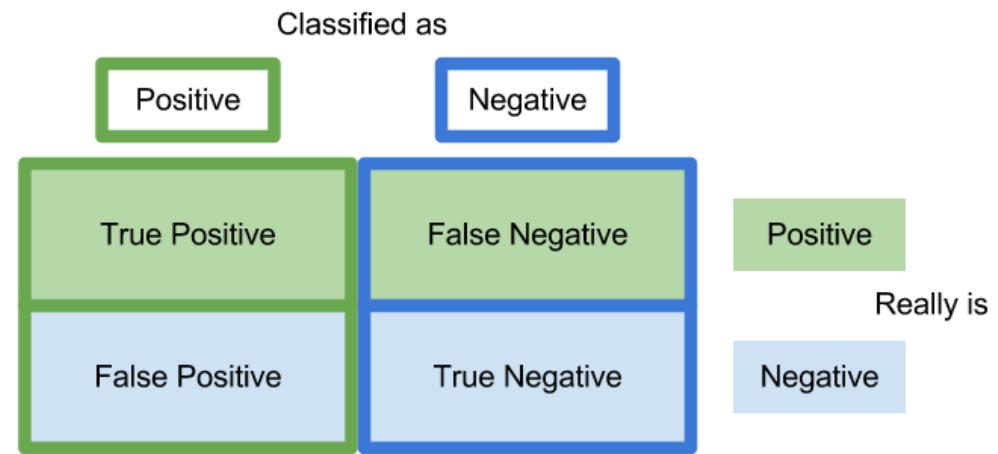
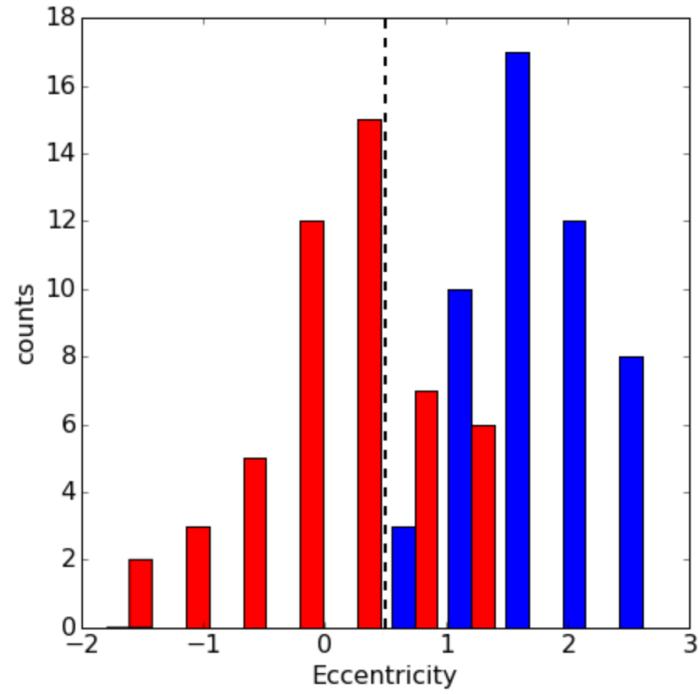
# The features



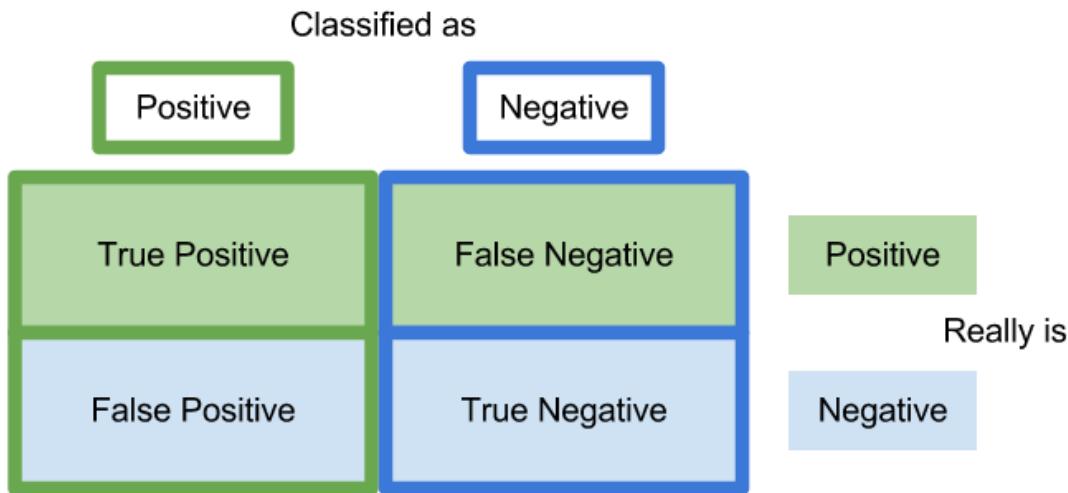
# The model



# Model evaluation

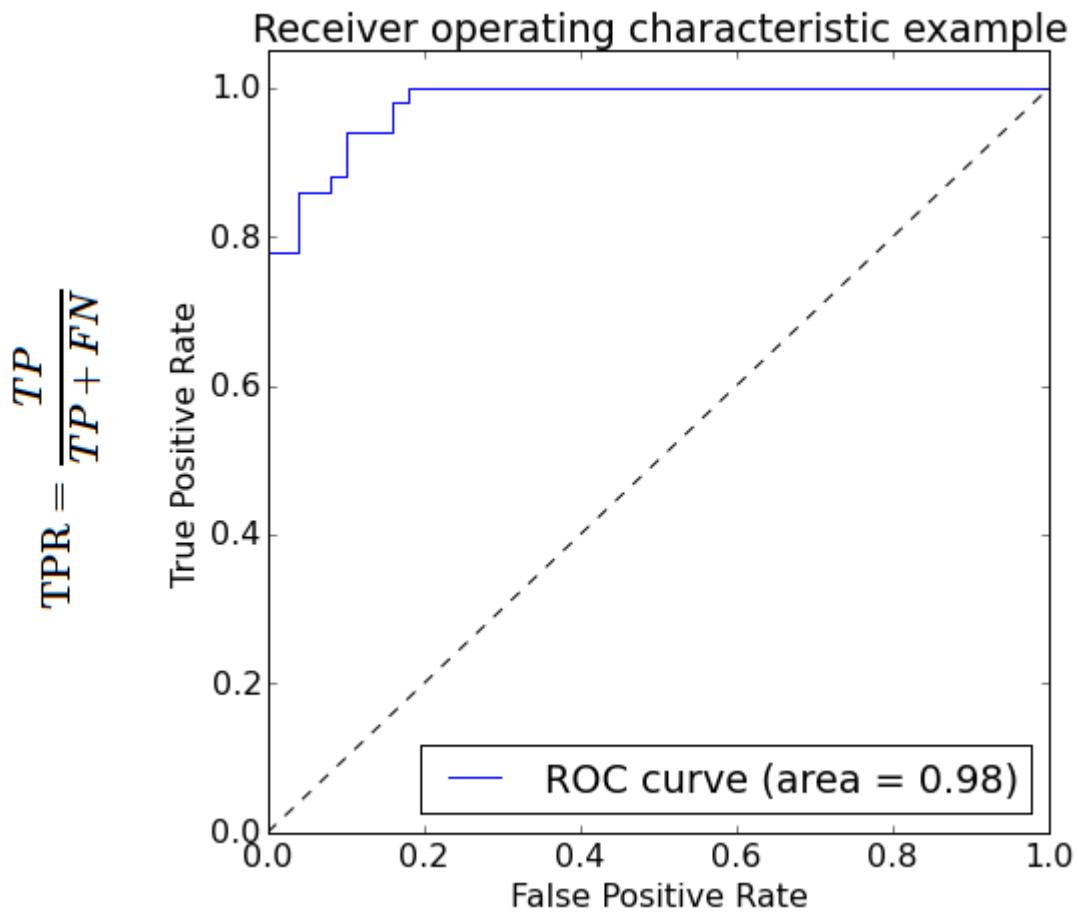


# Model evaluation

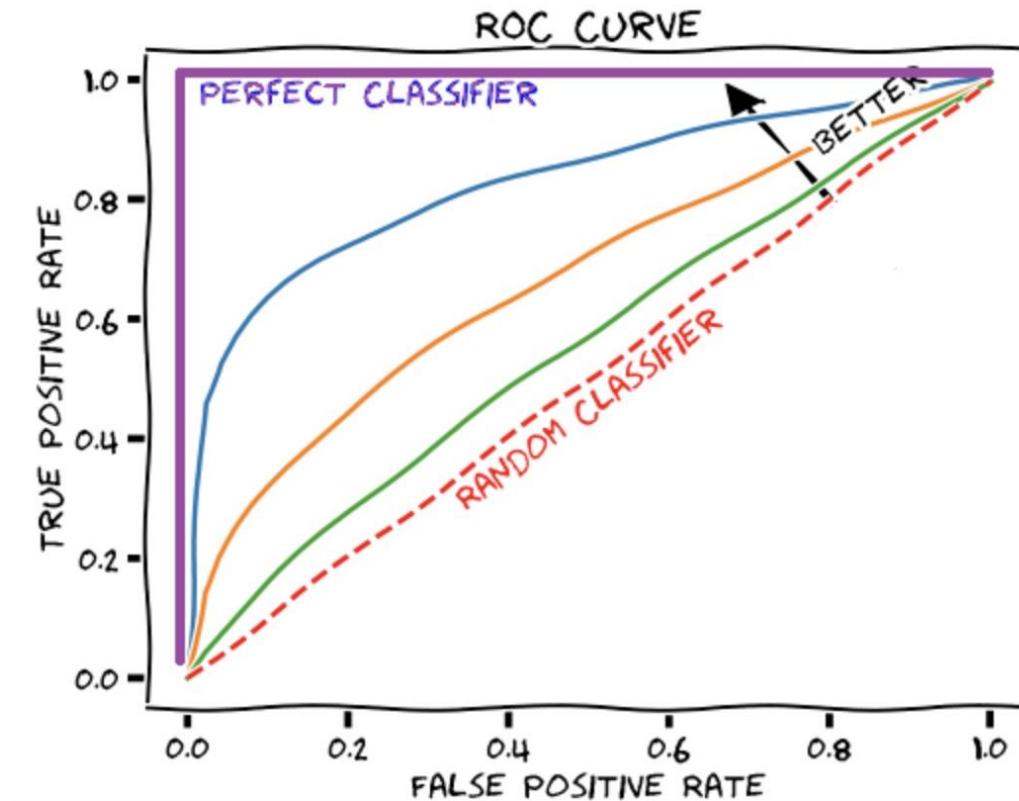


Recall Sensitivity True positive rate (TPR)	$\frac{TP}{FN + TP} = \frac{TP}{P}$
False positive rate (FPR) False alarm rate	$\frac{FP}{TN + FP} = \frac{FP}{N}$
Specificity True negative rate (TNR)	$\frac{TN}{TN + FP} = \frac{TN}{N} = 1 - FPR$
Precision	$\frac{TP}{TP + FP}$
False negative rate (FNR)	$\frac{FN}{FN + TP} = \frac{FN}{P}$
Accuracy	$\frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$

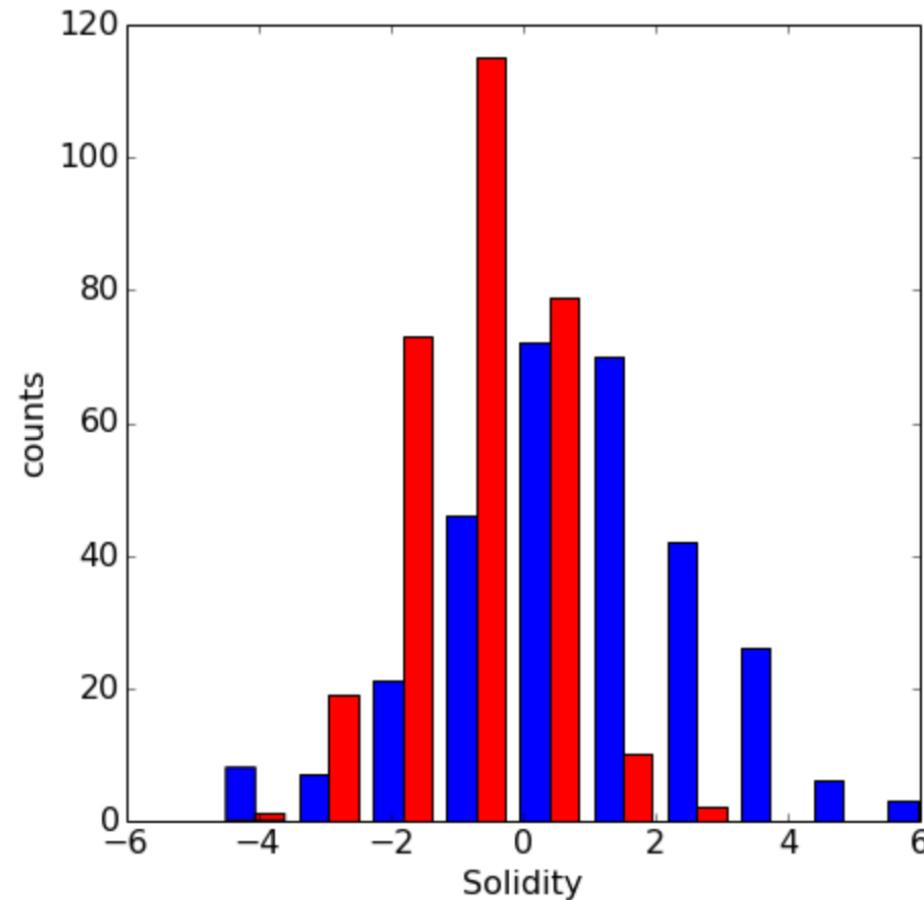
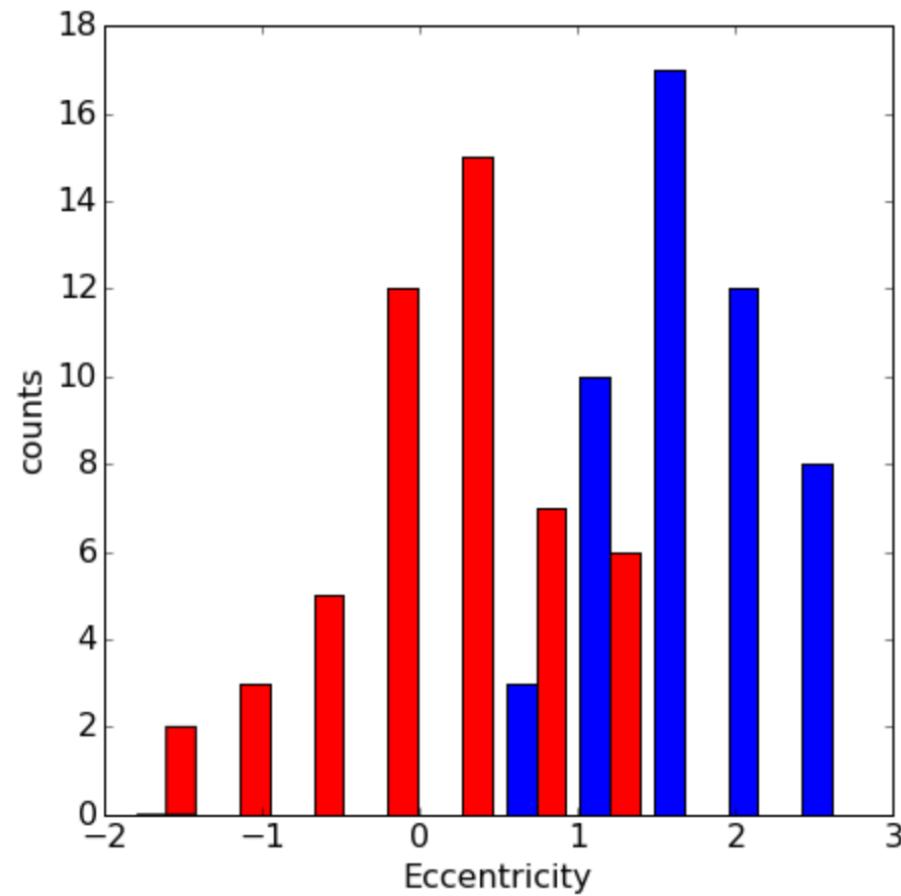
# Model evaluation



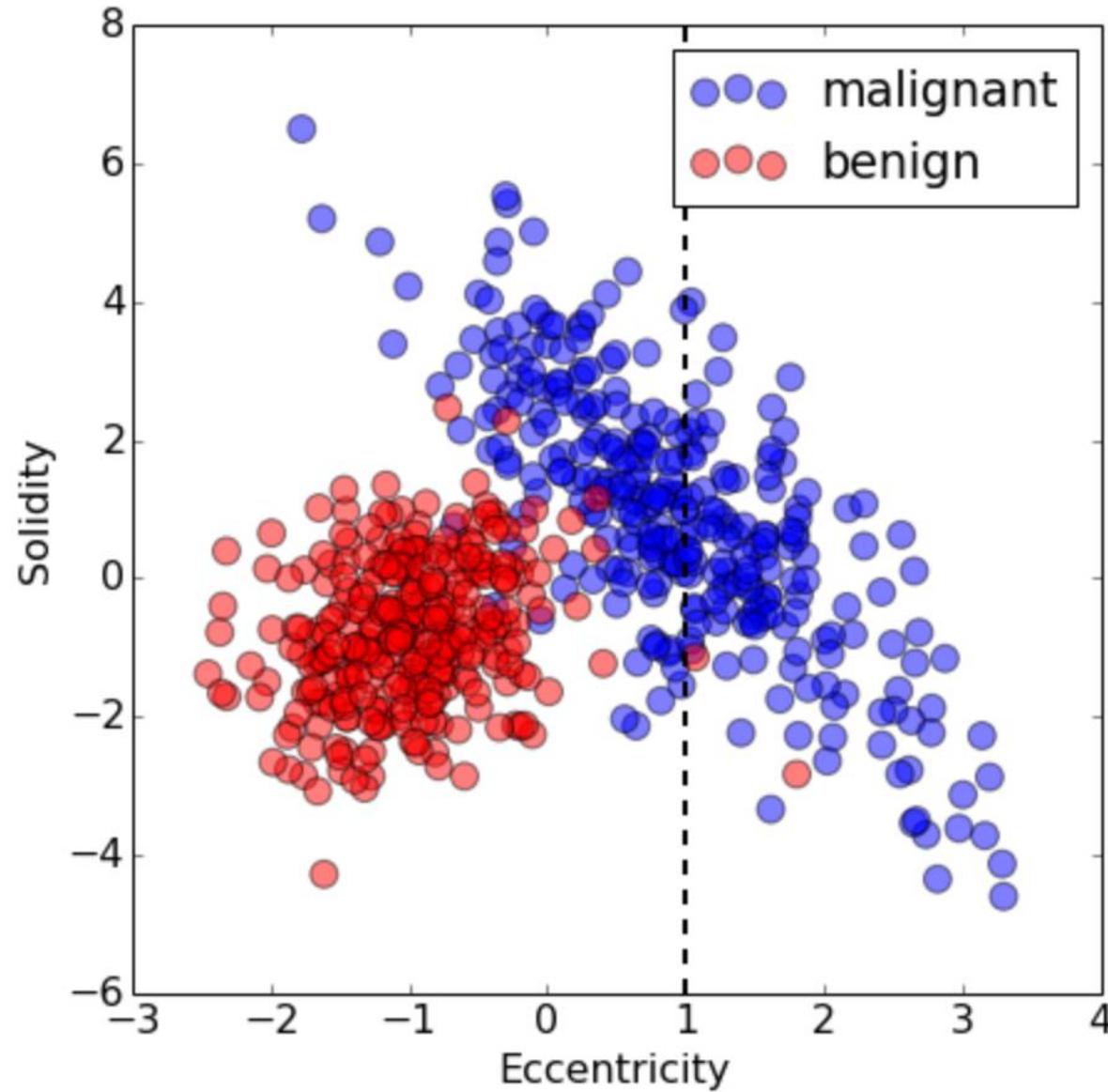
$$FPR = \frac{FP}{FP + TN}$$



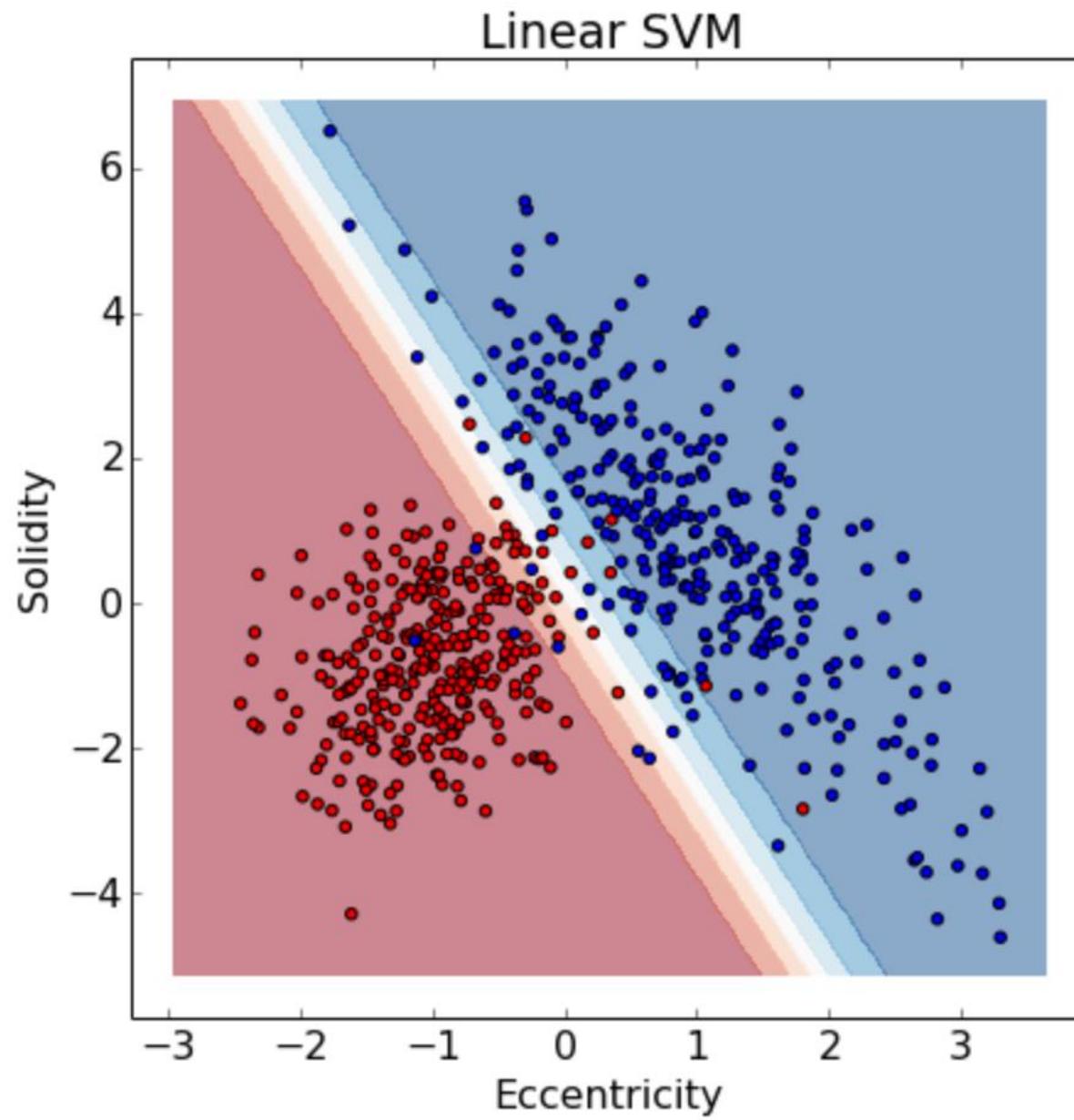
# The features



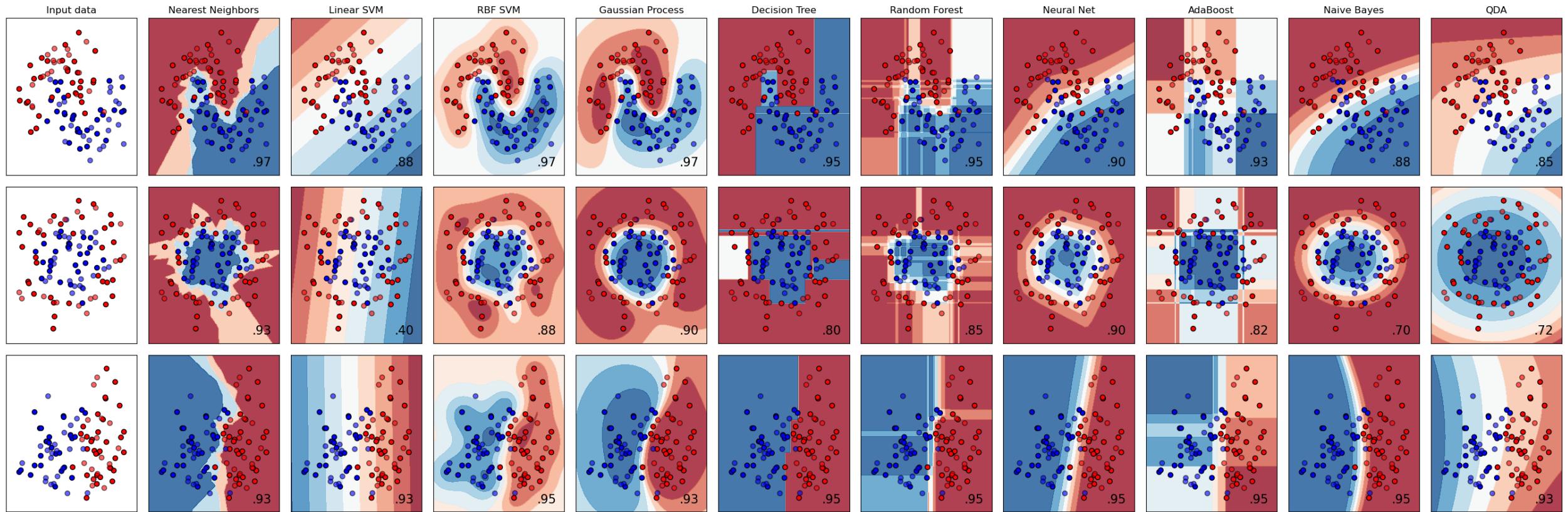
# The model



# The model



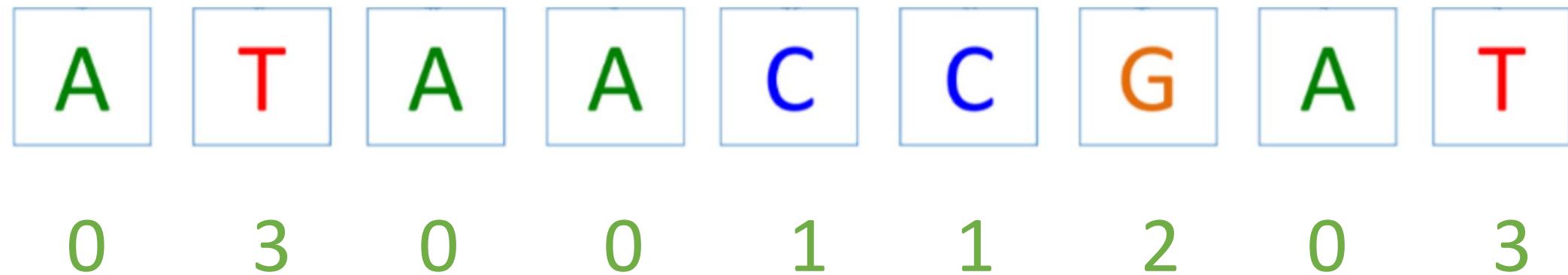
# The model



# The features

A T A A C C G A T

## The features



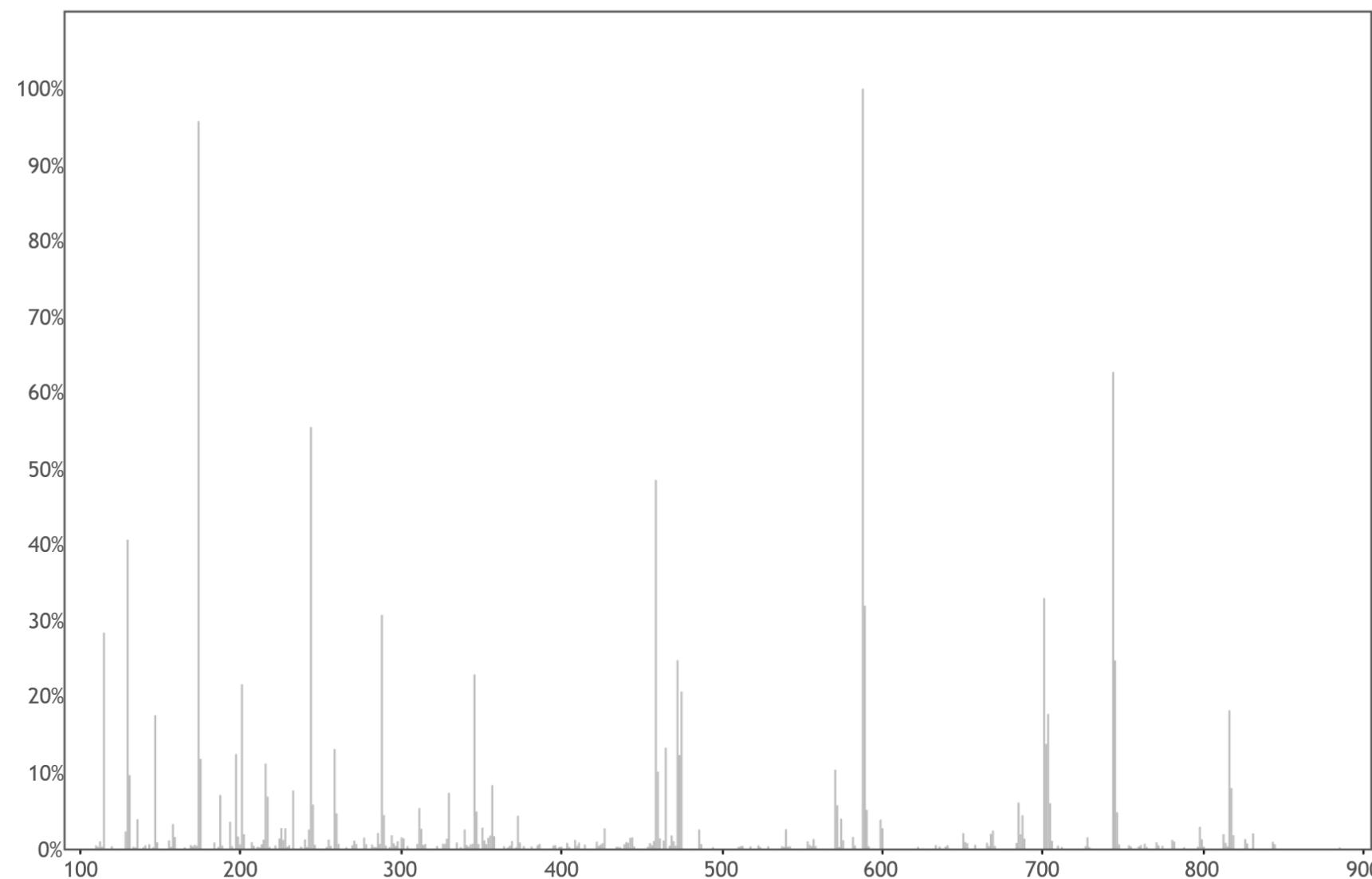
## The features

A	T	A	A	C	C	G	A	T
1	0	1	1	0	0	0	1	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1

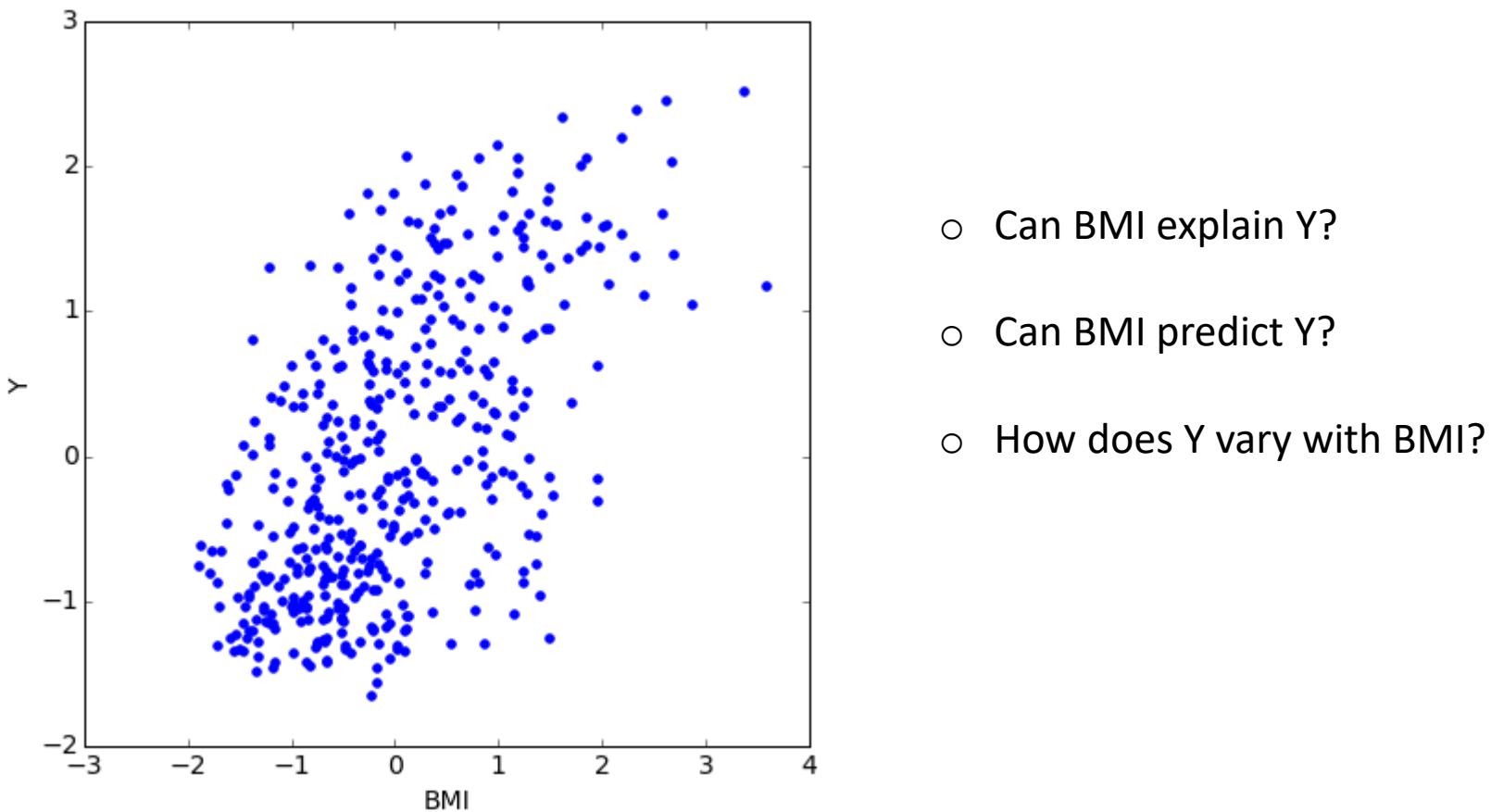
# The features

Residue	Mass	Hydrophobicity	Helicity	Basicity
A	71.0788	0.16	1.24	206.4
B	114.5962	-3.14	0.92	210.7
C	103.1388	2.5	0.79	206.2
D	115.0886	-2.49	0.89	208.6
E	129.1155	-1.5	0.85	215.6
F	147.1766	5	1.26	212.1
G	57.0519	-3.31	1.15	202.7
H	137.1411	-4.63	0.97	223.7
I	113.1594	4.41	1.29	210.8
K	128.1741	-5	0.88	221.8
L	113.1594	4.76	1.28	209.6
M	131.1926	3.23	1.22	213.3
N	114.1038	-3.79	0.94	212.8
P	97.1167	-4.92	0.57	214.4
Q	128.1307	-2.76	0.96	214.2
R	156.1875	-2.77	0.95	237
S	87.0782	-2.85	1	207.6
T	101.1051	-1.08	1.09	211.7
V	99.1326	3.02	1.27	208.7
W	186.2132	4.88	1.07	216.1
X	113.1594	4.59	1.29	210.2
Y	163.176	2	1.11	213.1
Z	128.6231	-2.13	0.91	214.9

# The features

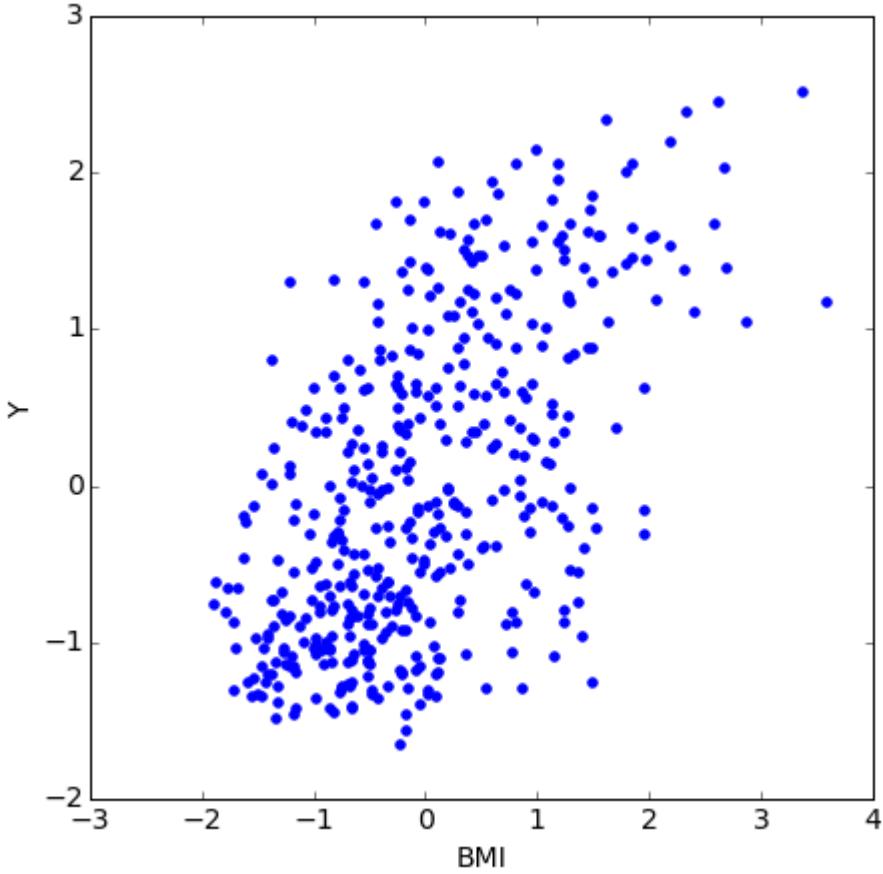


# Regression



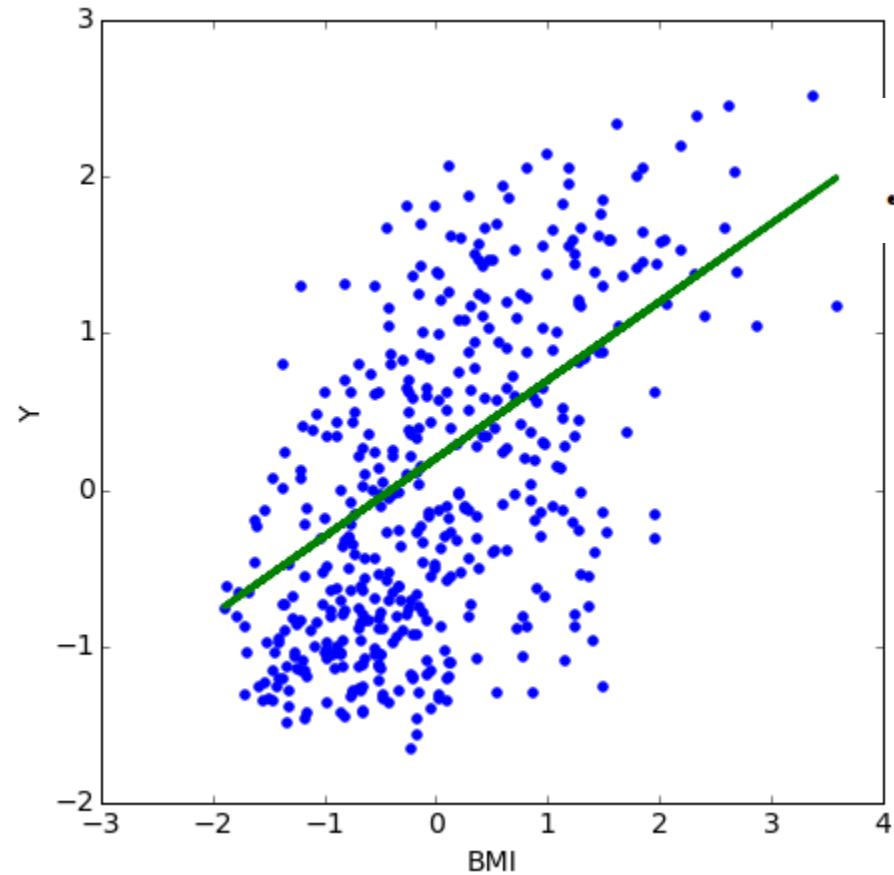
Regression is a general term for **modeling the relationship** between a **label** (a.k.a. dependent variable or target) and one or more **features** (a.k.a. the attributes, the independent or explanatory variable(s)).

# Linear regression



We need to make  
**assumptions** *linear relationship*  
about the  
**model** *linear model*  
that generated the data.

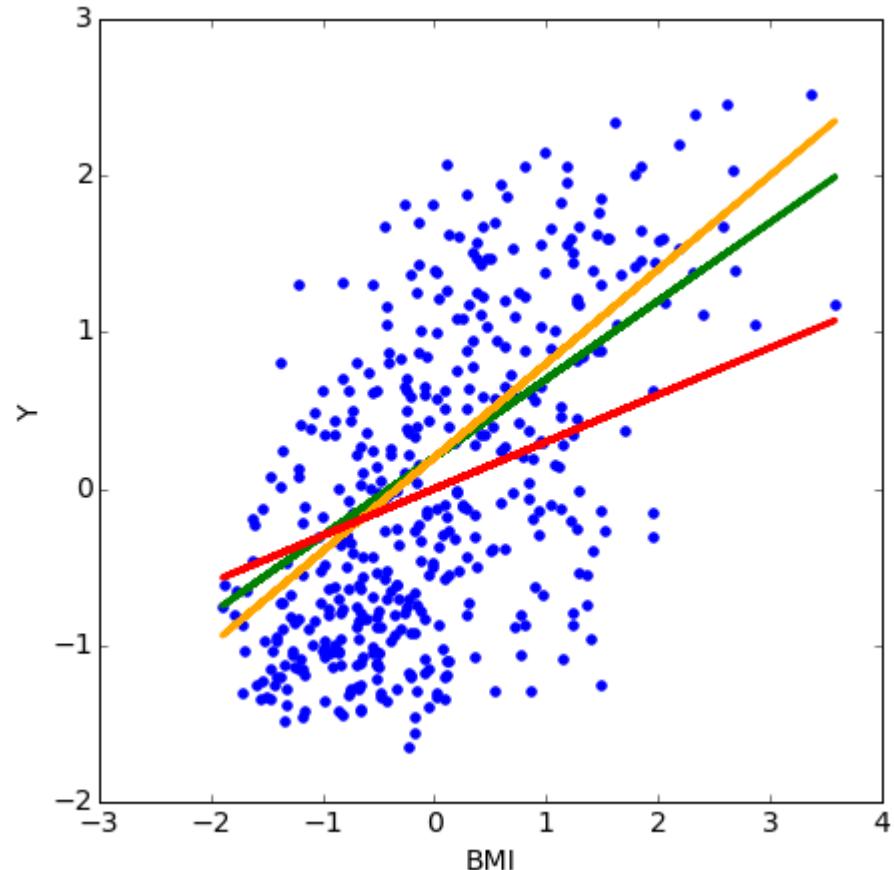
# Linear regression



$$f(x) = 0.5 \cdot x + 0.2 = ax + b$$

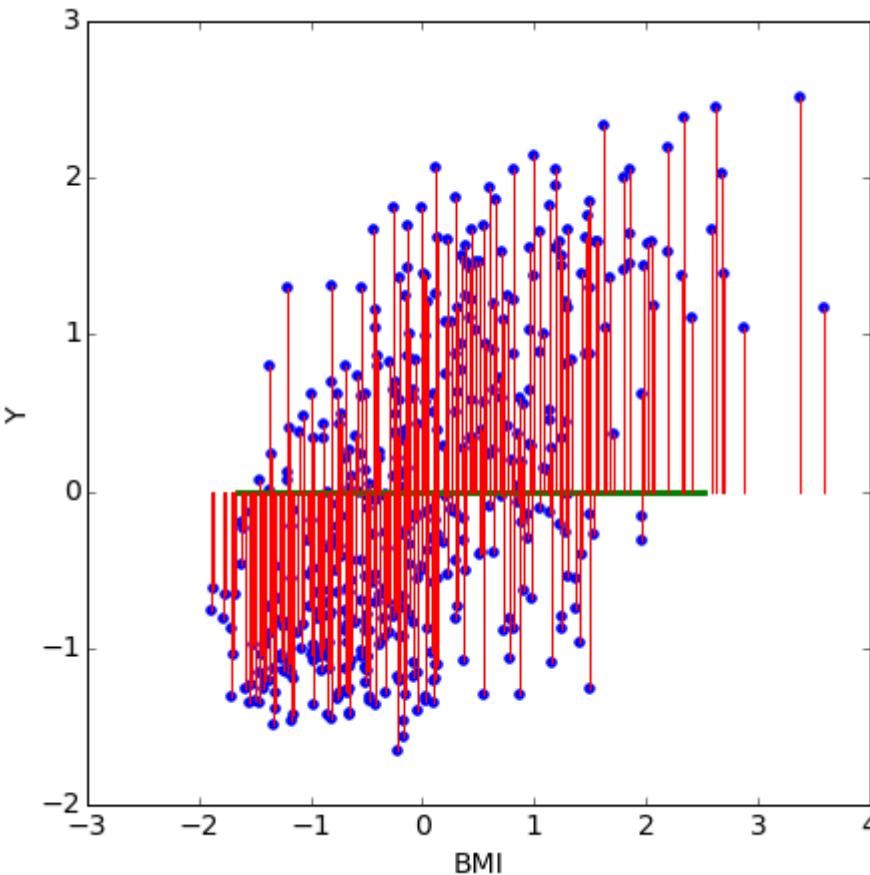
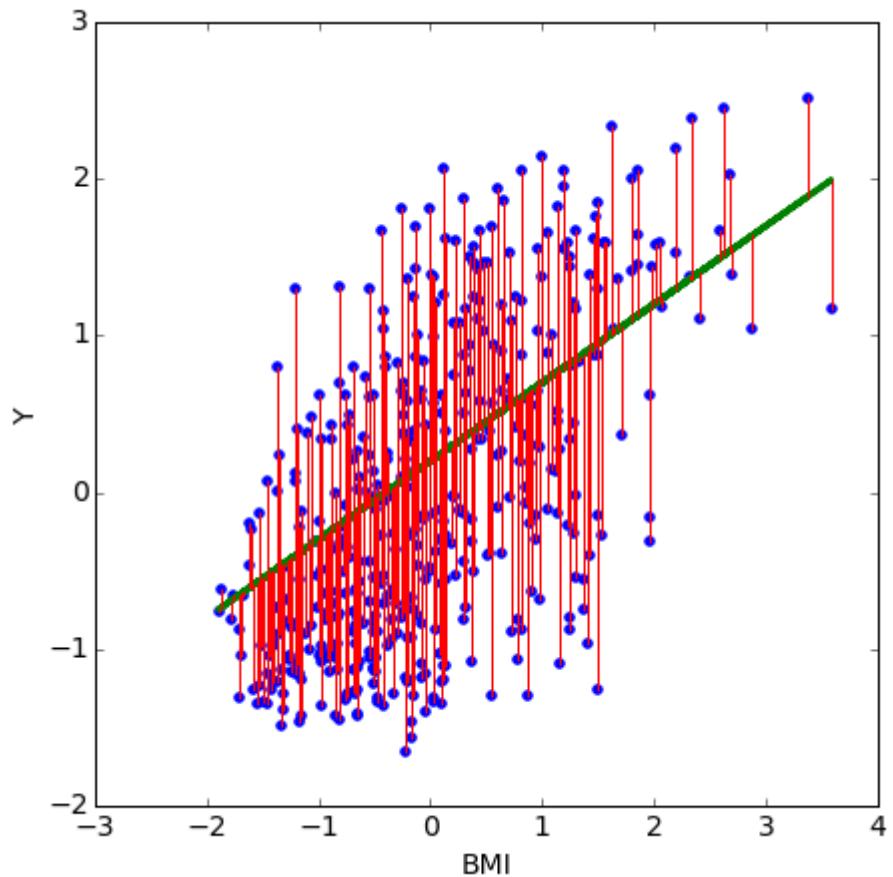
- $a$  and  $b$  are model parameters
- $a$  is the slope to the line (direction)
- $b$  is the intercept or bias (position)

# Linear regression: fitting



- What values for  $a$  and  $b$  fit the data best?
- How can we evaluate them?

# Linear regression: fitting



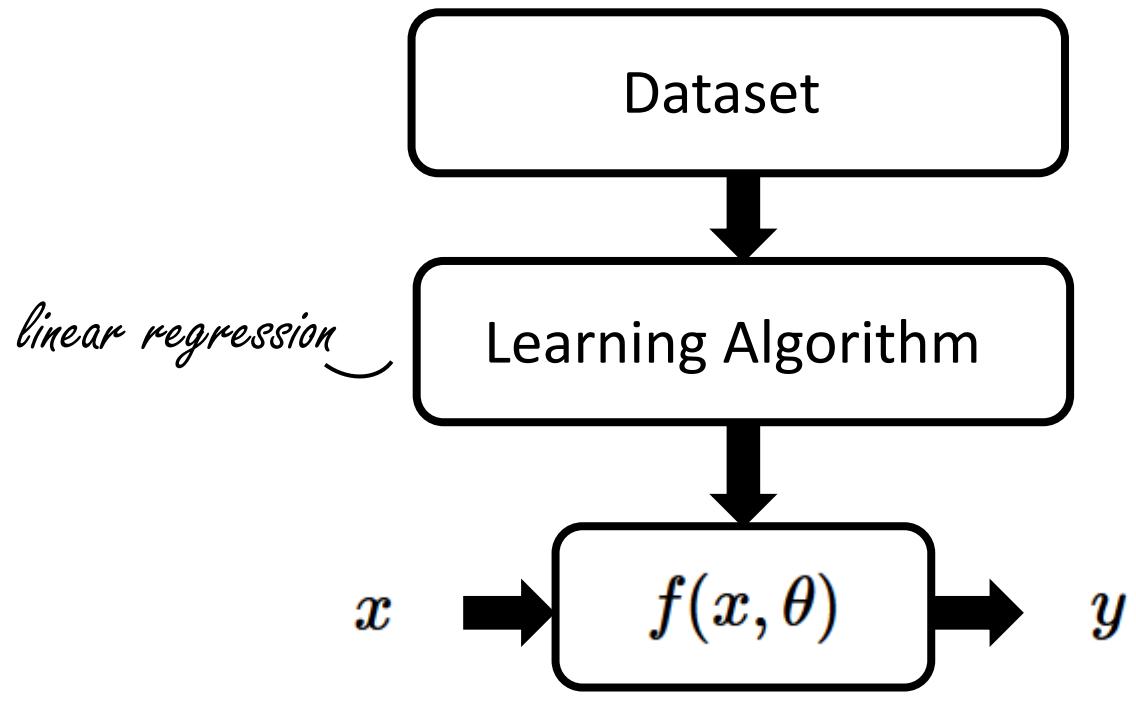
$$R^2 = 1 - \frac{SS_{error}}{SS_{total}}$$

$$R^2 = 0.296$$

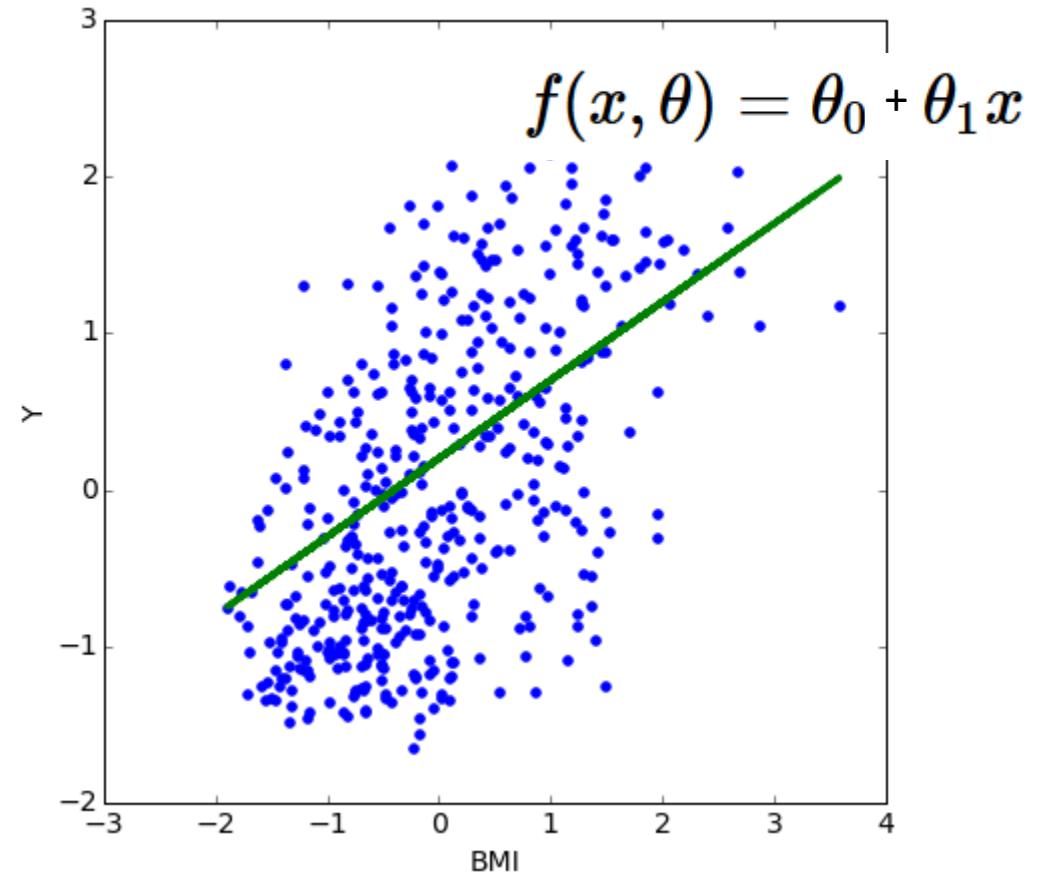
$$SS_{error} = \sum_{i=1}^n (y^{(i)} - f(x^{(i)}))^2$$

$$SS_{total} = \sum_{i=1}^n (y^{(i)} - \bar{y})^2$$

# Linear regression



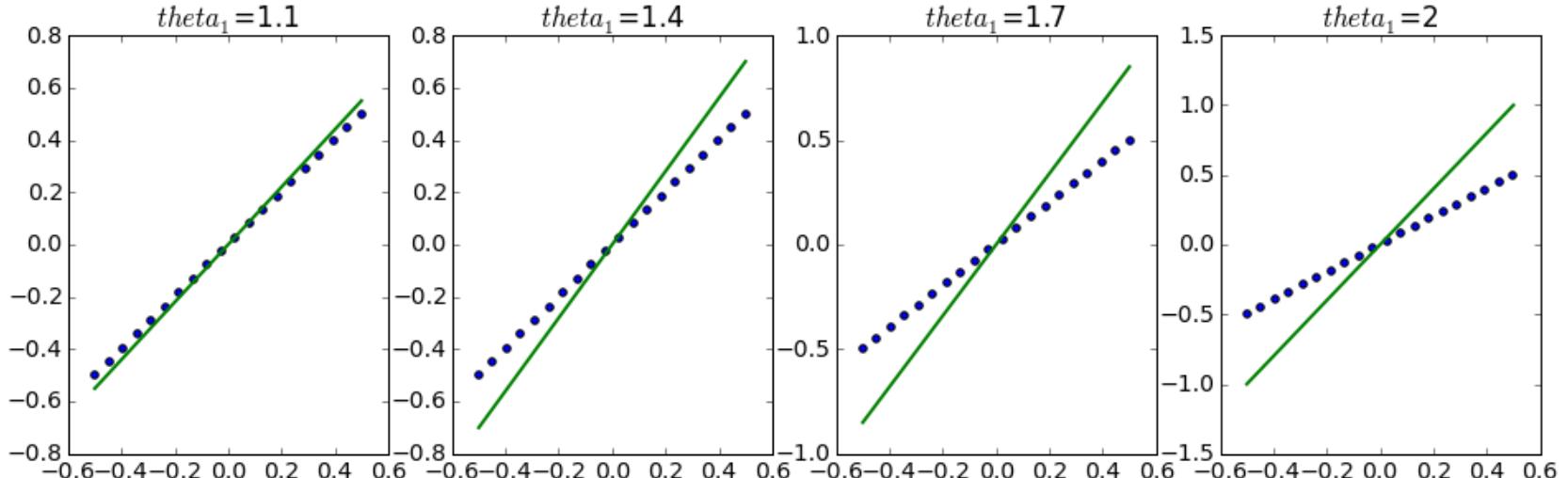
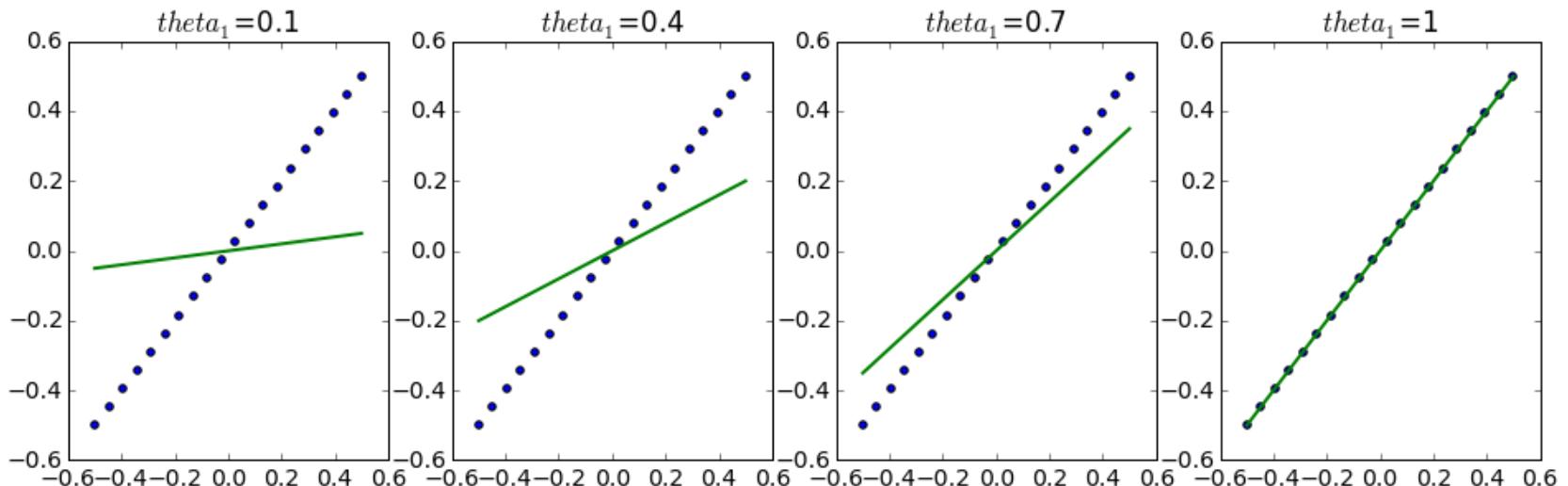
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m = \theta' x$$



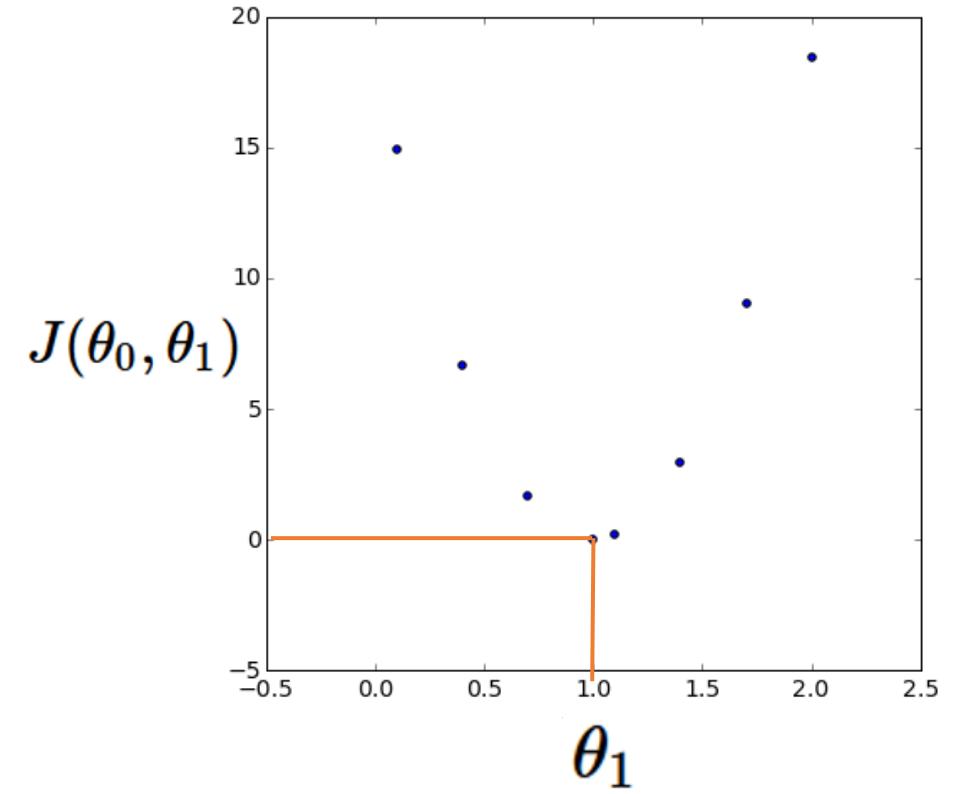
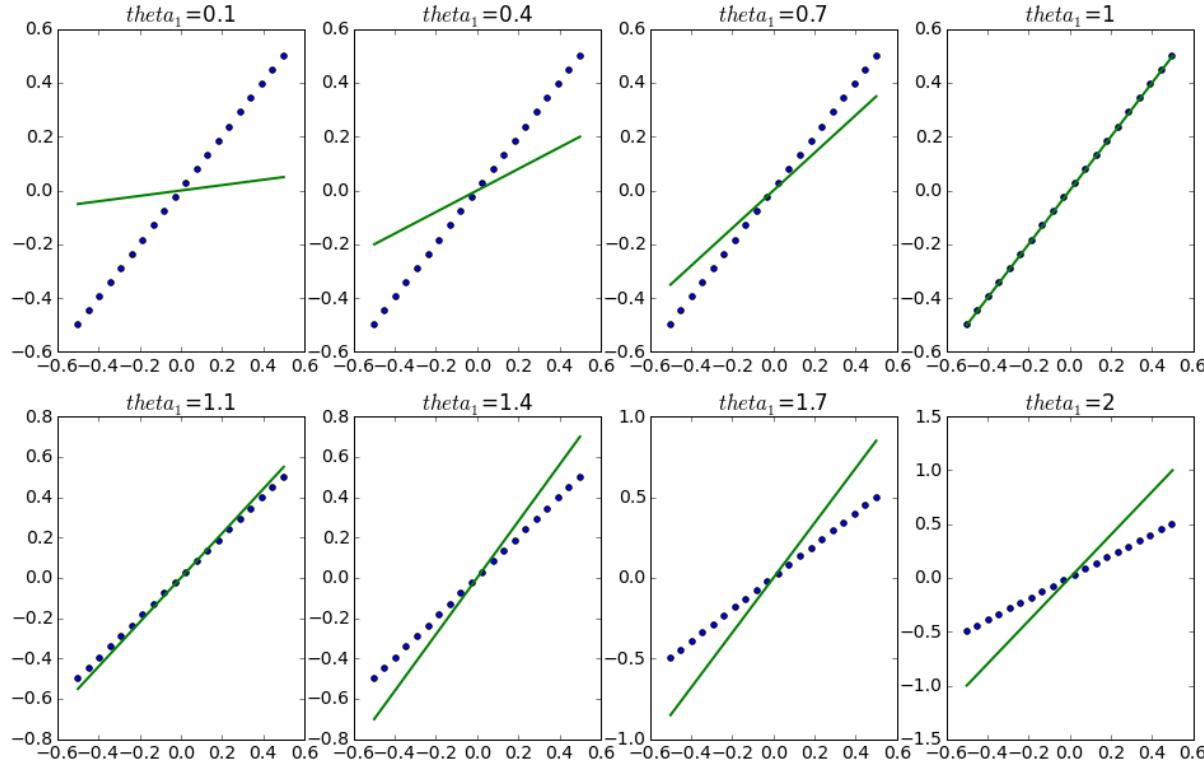
# Linear regression: cost function

$$f(x) = x$$

$$(\theta_1 = 1, \theta_0 = 0)$$



# Linear regression: cost function



$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)})^2$$

# Linear regression

Fit a linear model

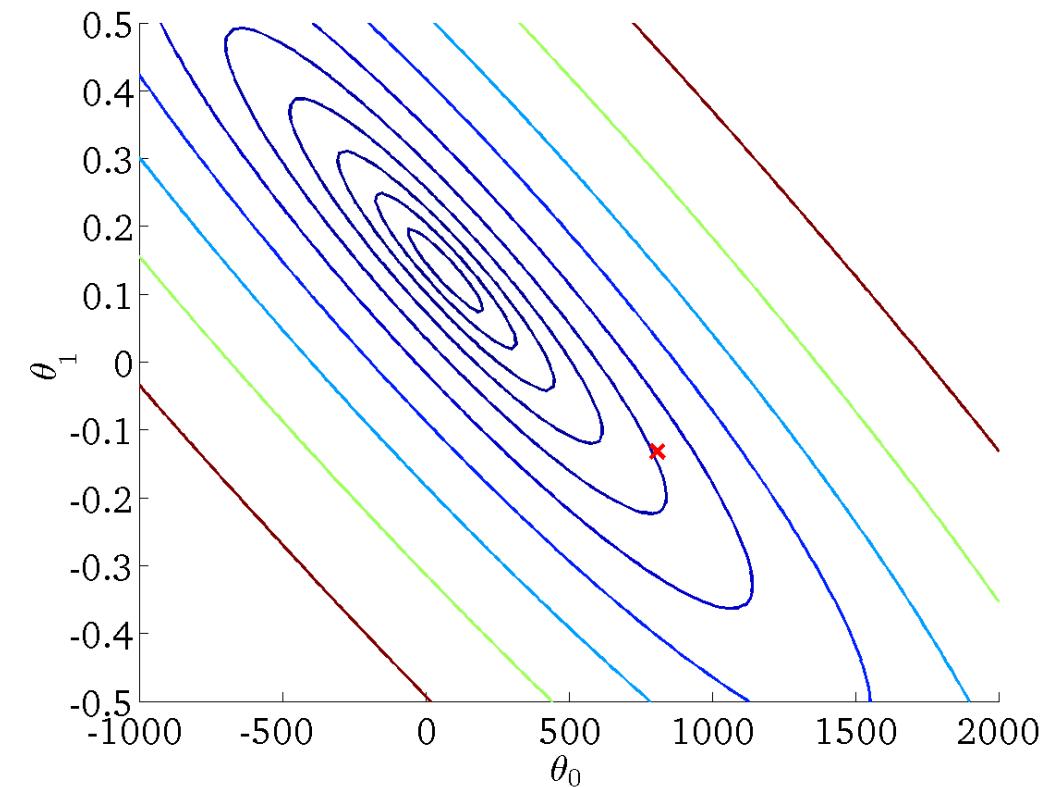
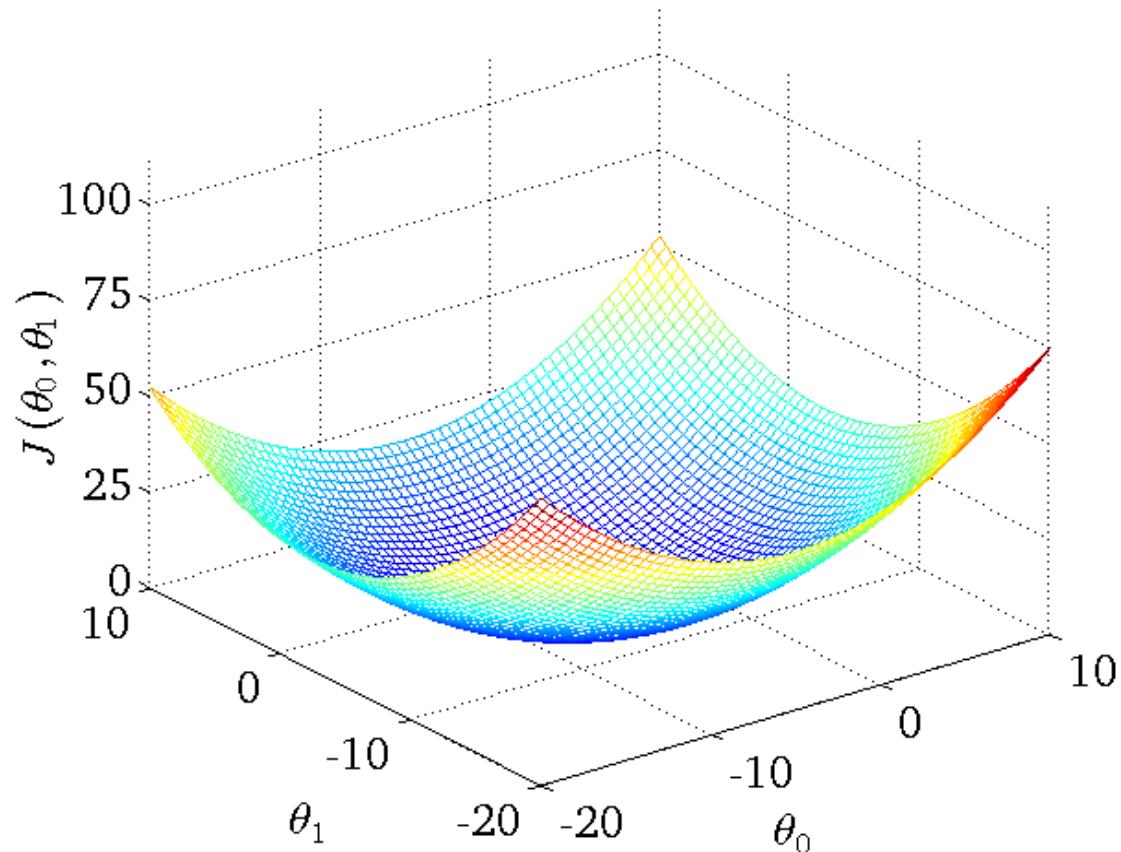
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m = \theta' x$$

to the data set such that the cost function

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)})^2$$

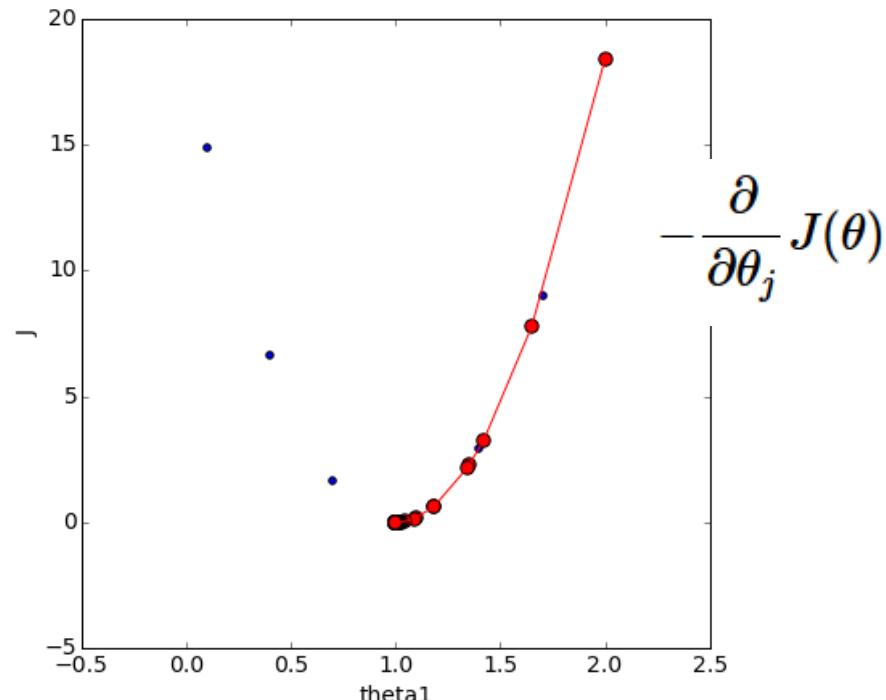
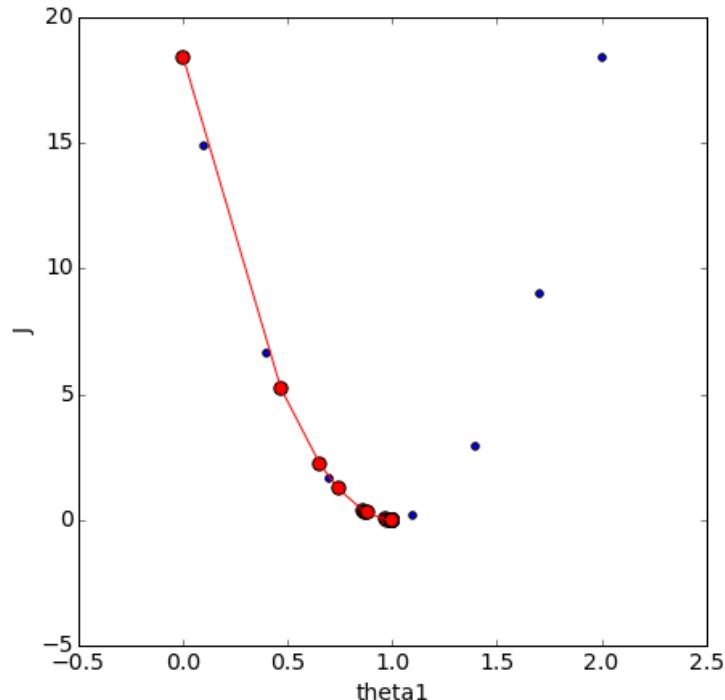
is minimal.

# Linear regression: gradient descent



# Linear regression: gradient descent

1. start with some random initialization of  $\theta$ , i.e. a randomly chosen model
2. increment or decrement the value(s) of  $\theta$  slightly such that  $J(\theta)$  is reduced
3. repeat step 2. until we observe convergence



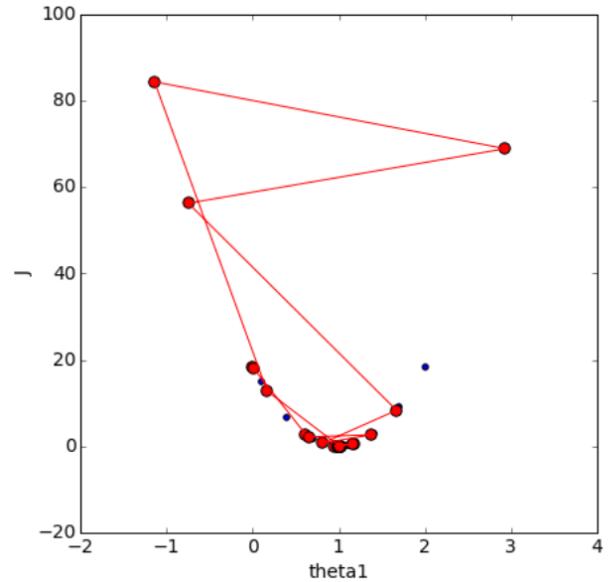
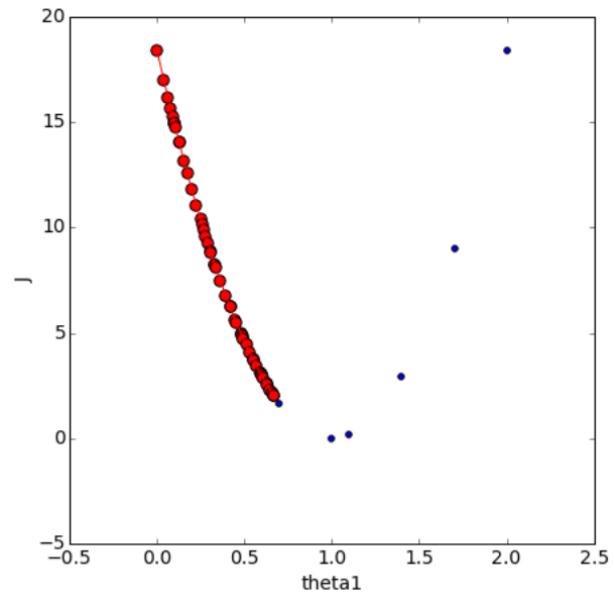
# Linear regression: gradient descent

$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

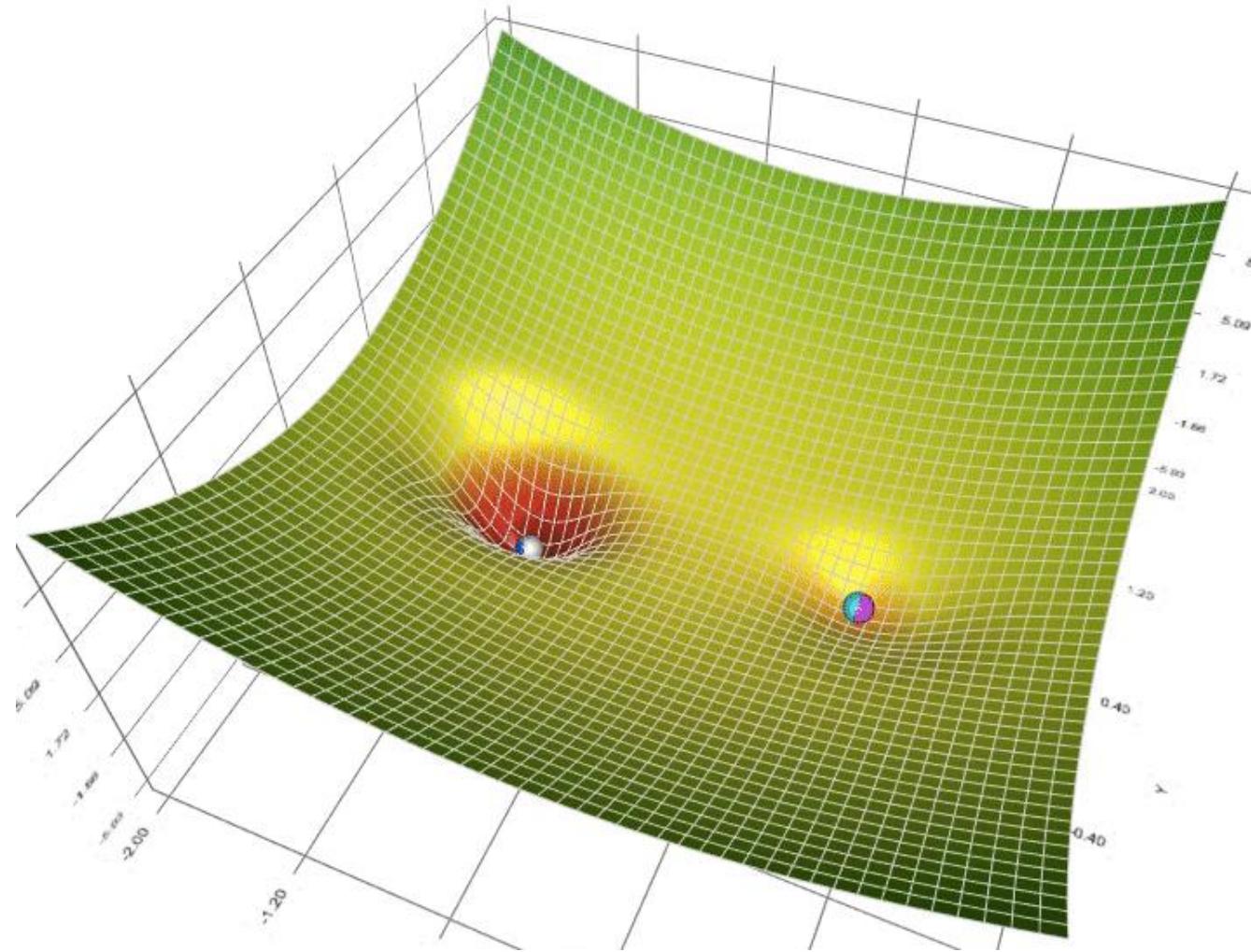
$$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

...



# Linear regression: gradient descent



# Linear regression

model:  $f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$

cost function:  $J(\theta) = \frac{1}{2n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)})^2$

Goal: minimize  $J(\theta)$

$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

Learning: 1. start with some  $\theta$   
2. change  $\theta$  to reduce  $J(\theta)$   
3. repeat 2. until convergence



$$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

...

# Data normalization

- make all features same scale
- Eccentricity [0,100], Solidity [-5,7]
- weights all features equally in their representation
- **standardization**

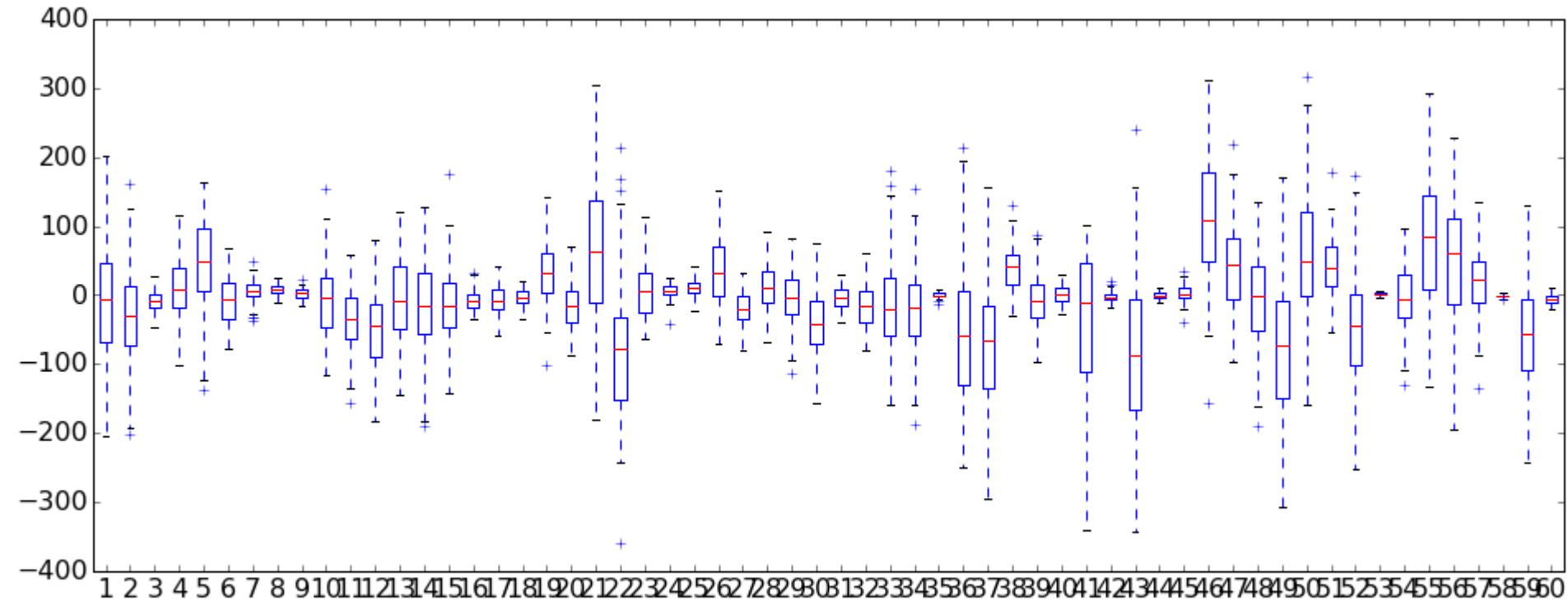
$$\mu = 0 \quad \sigma = 1$$

$$x_{norm} = \frac{x - \mu}{\sigma}$$

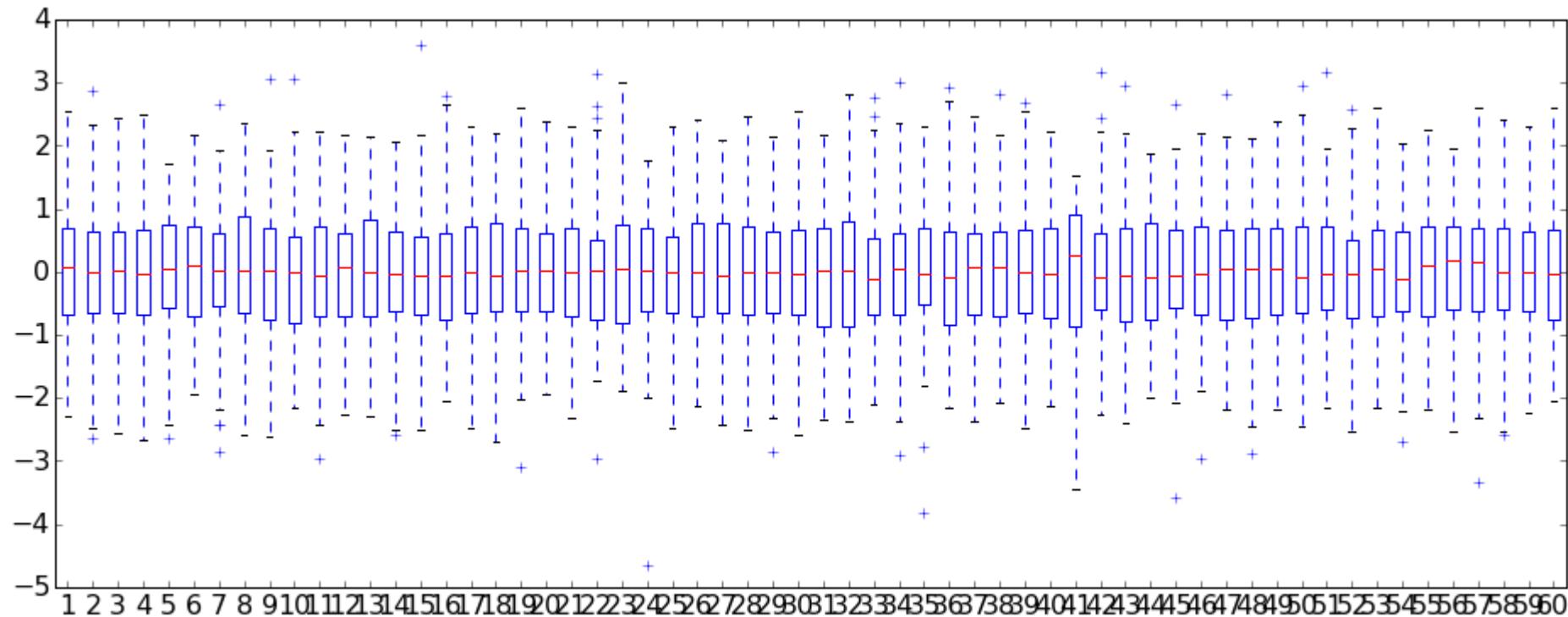
$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **min-max scaling:** scale the features to a fixed range

# Data normalization

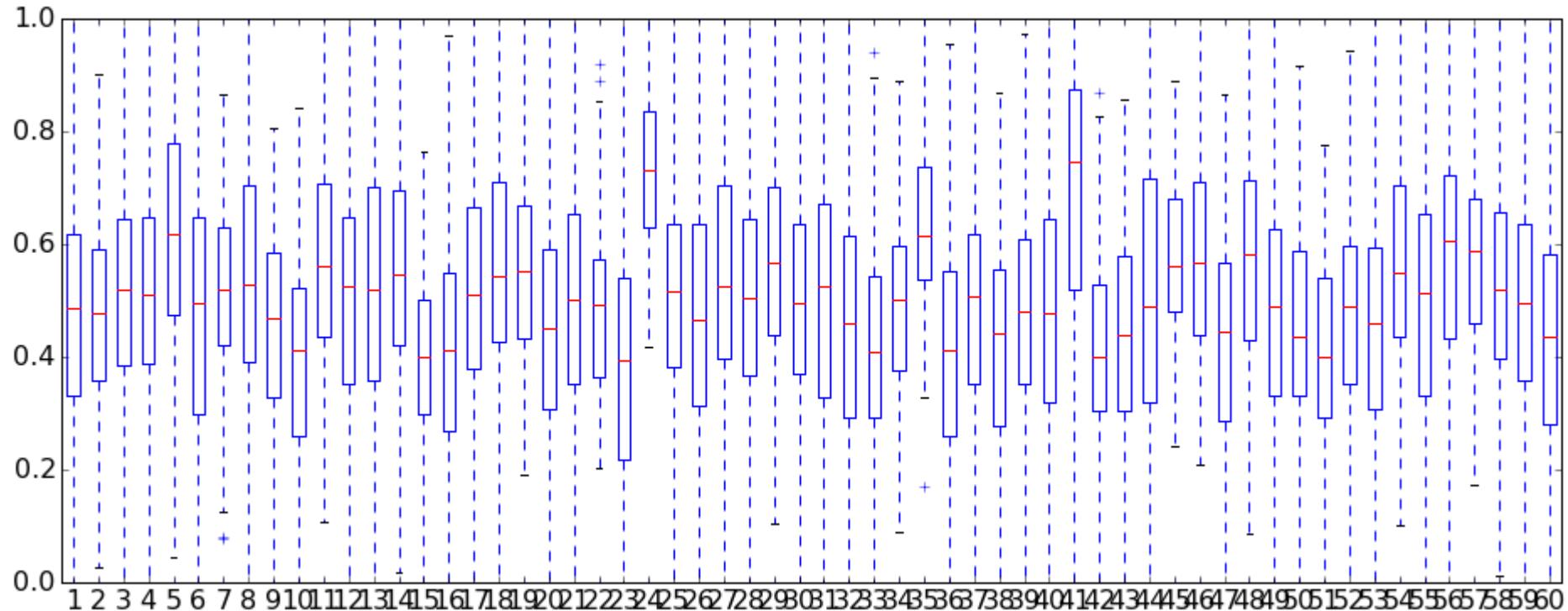


# Data normalization



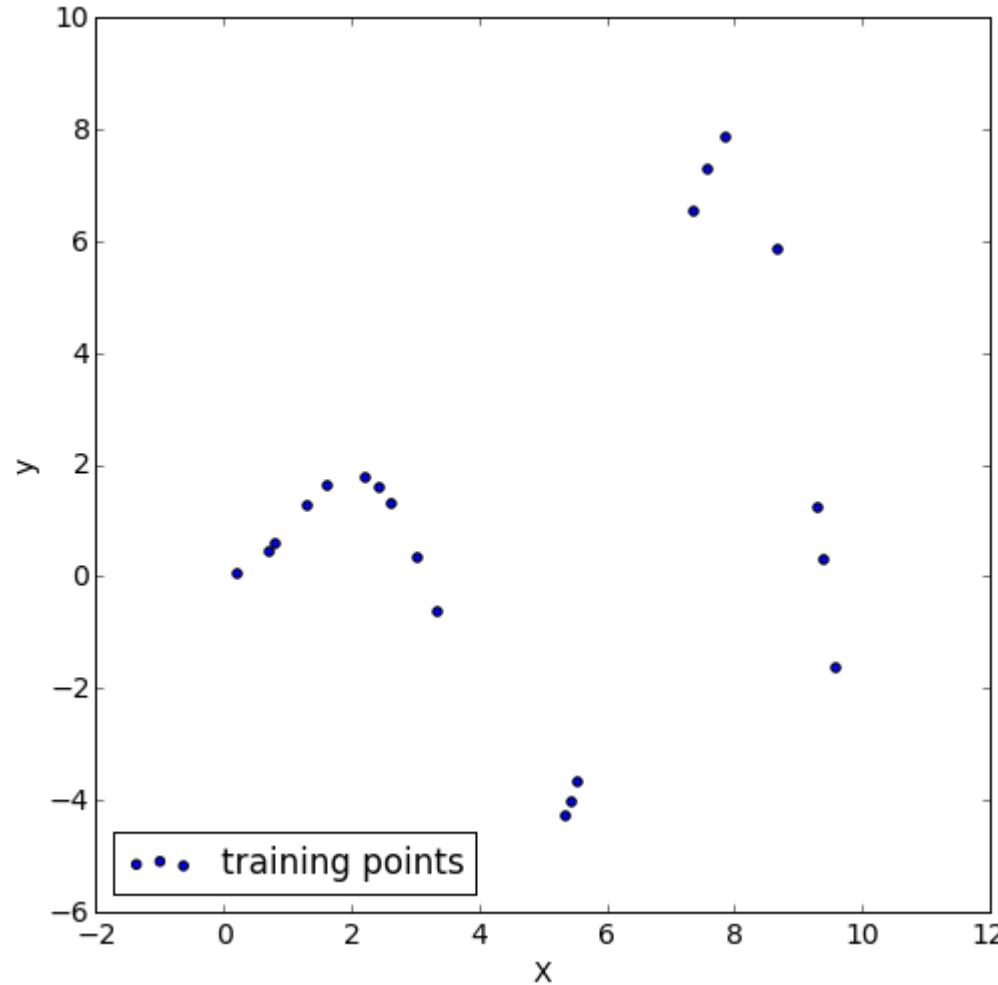
$$x_{norm} = \frac{x - \mu}{\sigma}$$

# Data normalization



$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

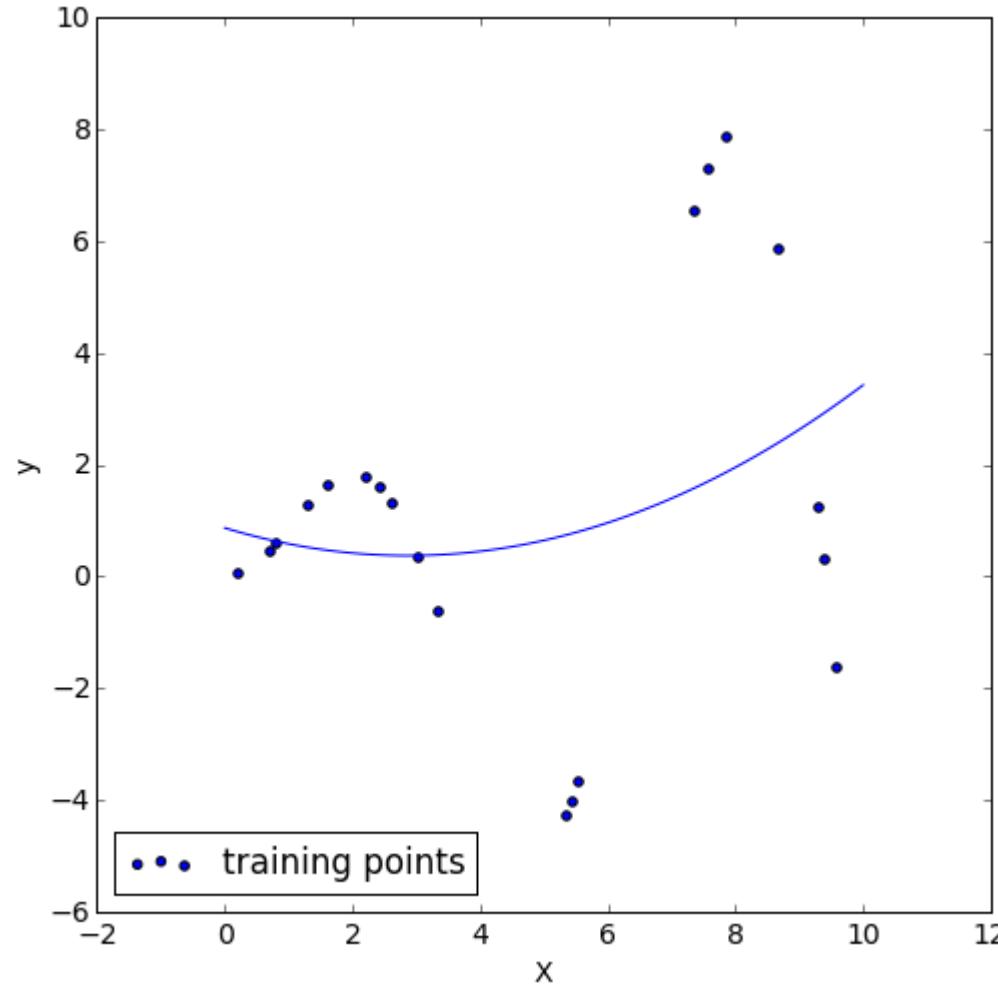
# Non-linear regression



- Does  $y$  vary linearly with  $X$ ?
- Can we fit a non-linear model? Yes
- Or we could add polynomial transformations of the features.

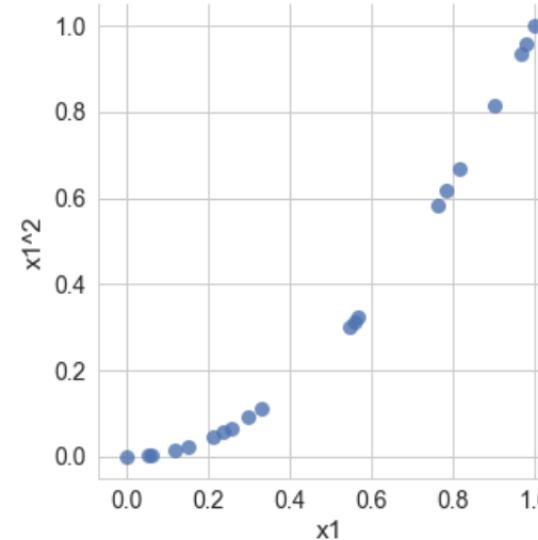
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

# Non-linear regression

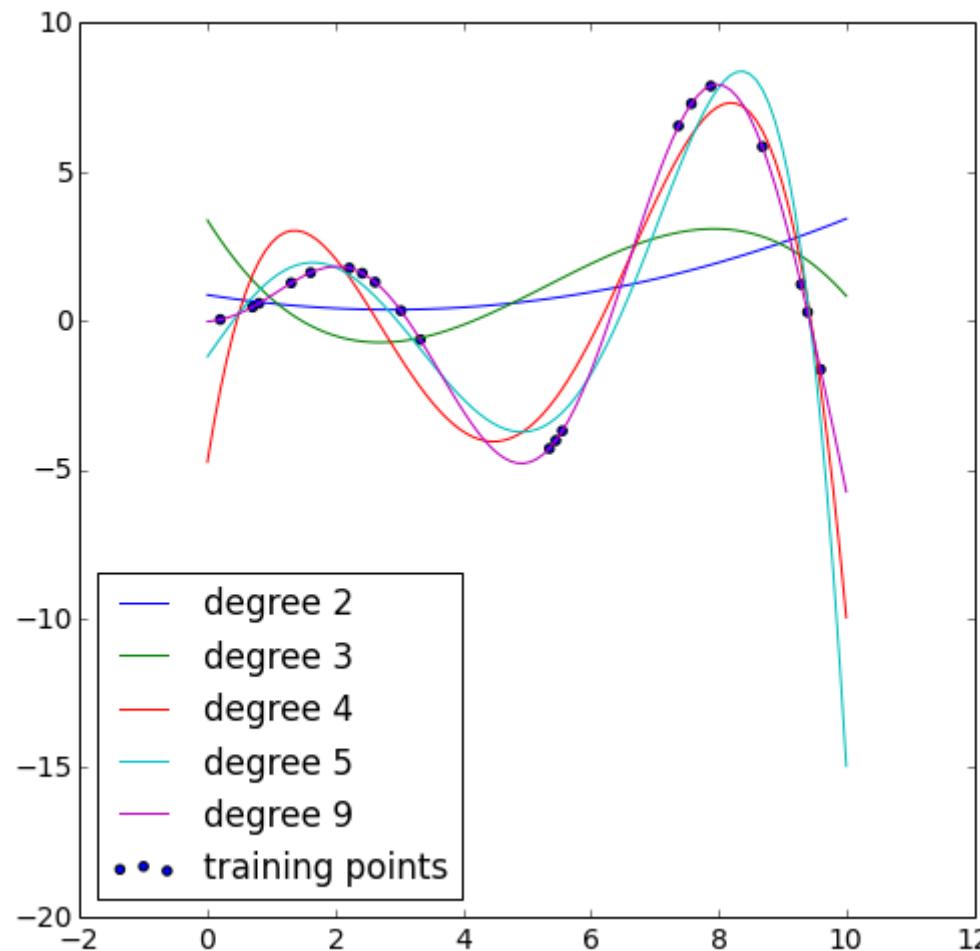


- Does  $y$  vary linearly with  $X$ ?
- Can we fit a non-linear model? Yes
- Or we could add polynomial transformations of the features.

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$



# Non-linear regression

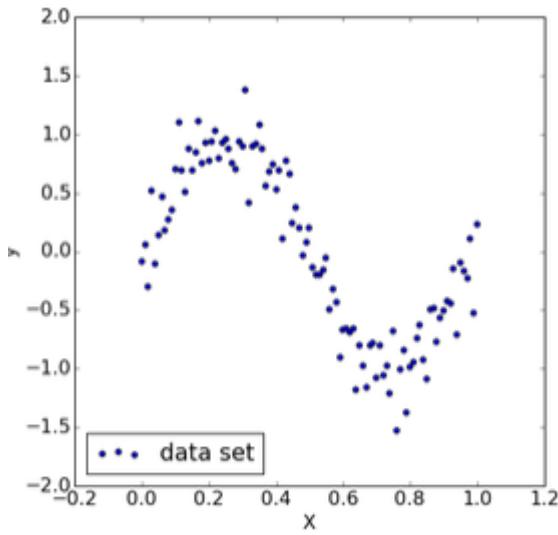


$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^4 + \theta_5 x_1^5 + \theta_6 x_1^6 + \theta_7 x_1^7 + \theta_8 x_1^8 + \theta_9 x_1^9$$

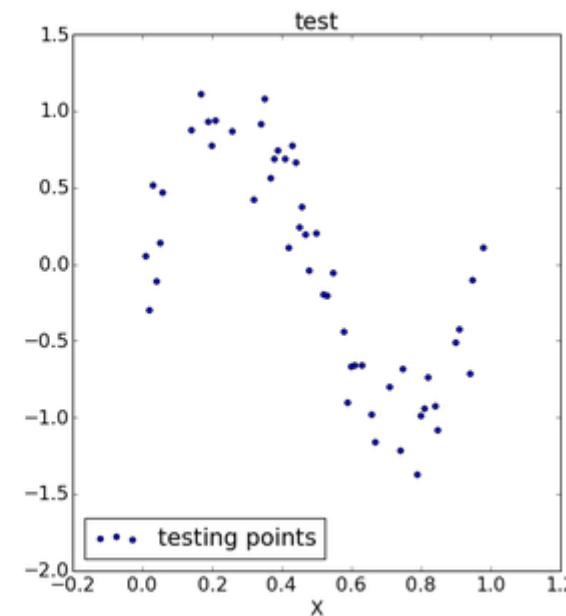
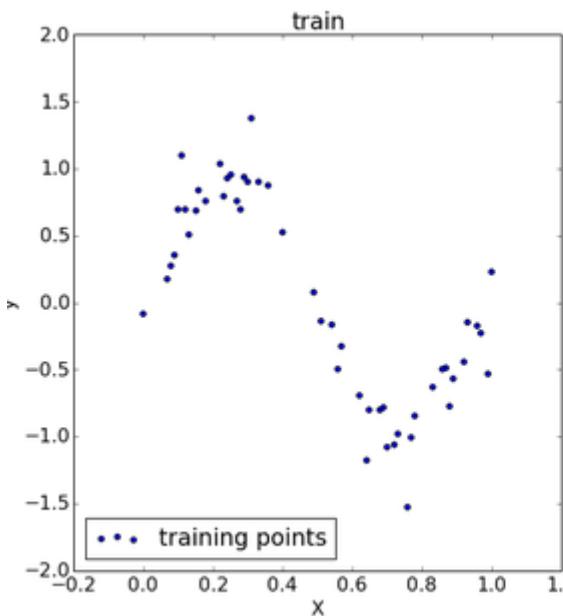
**We want to estimate the prediction  
performance of the model on  
unseen external data**

(data not seen during training)

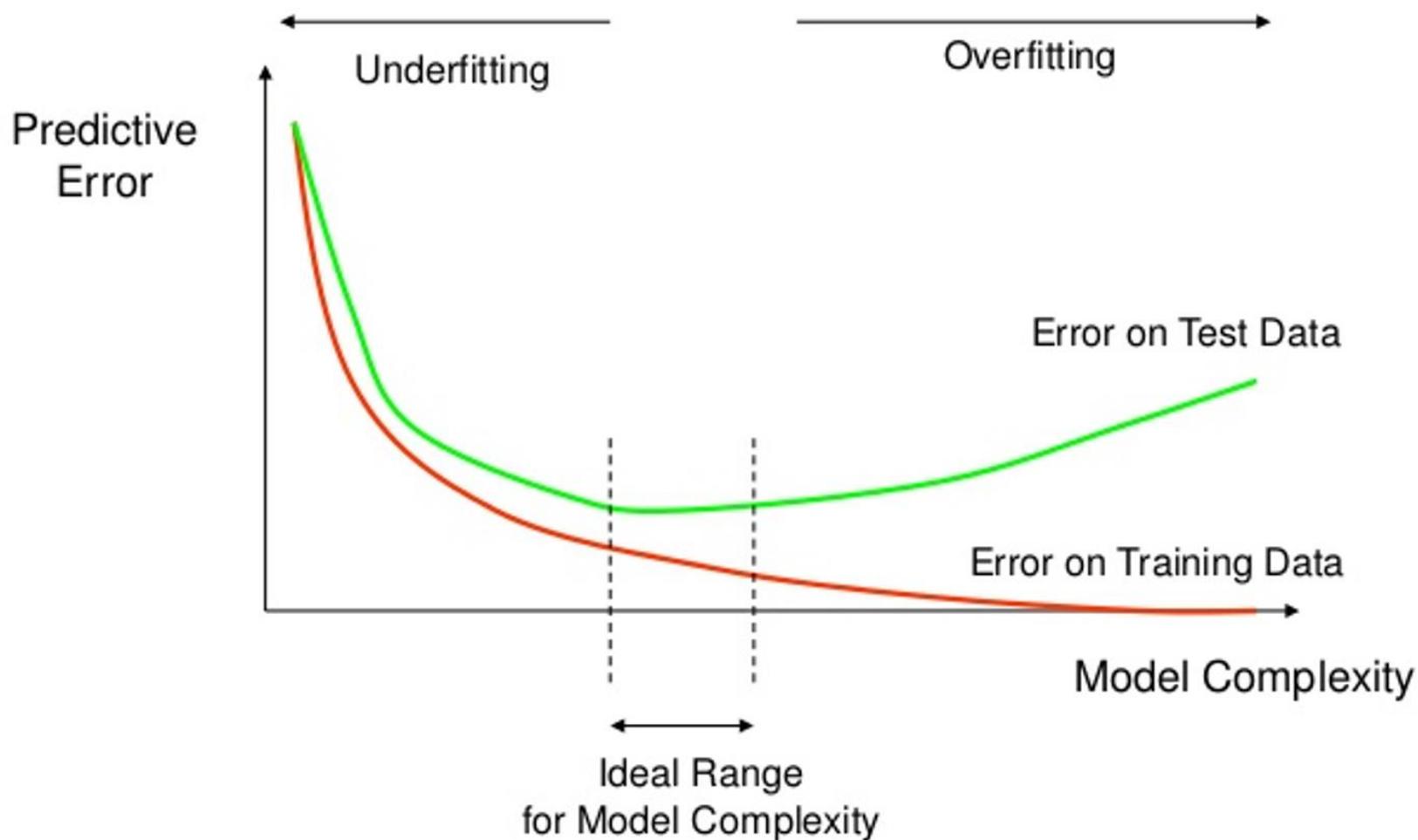
# Model evaluation: train-test split



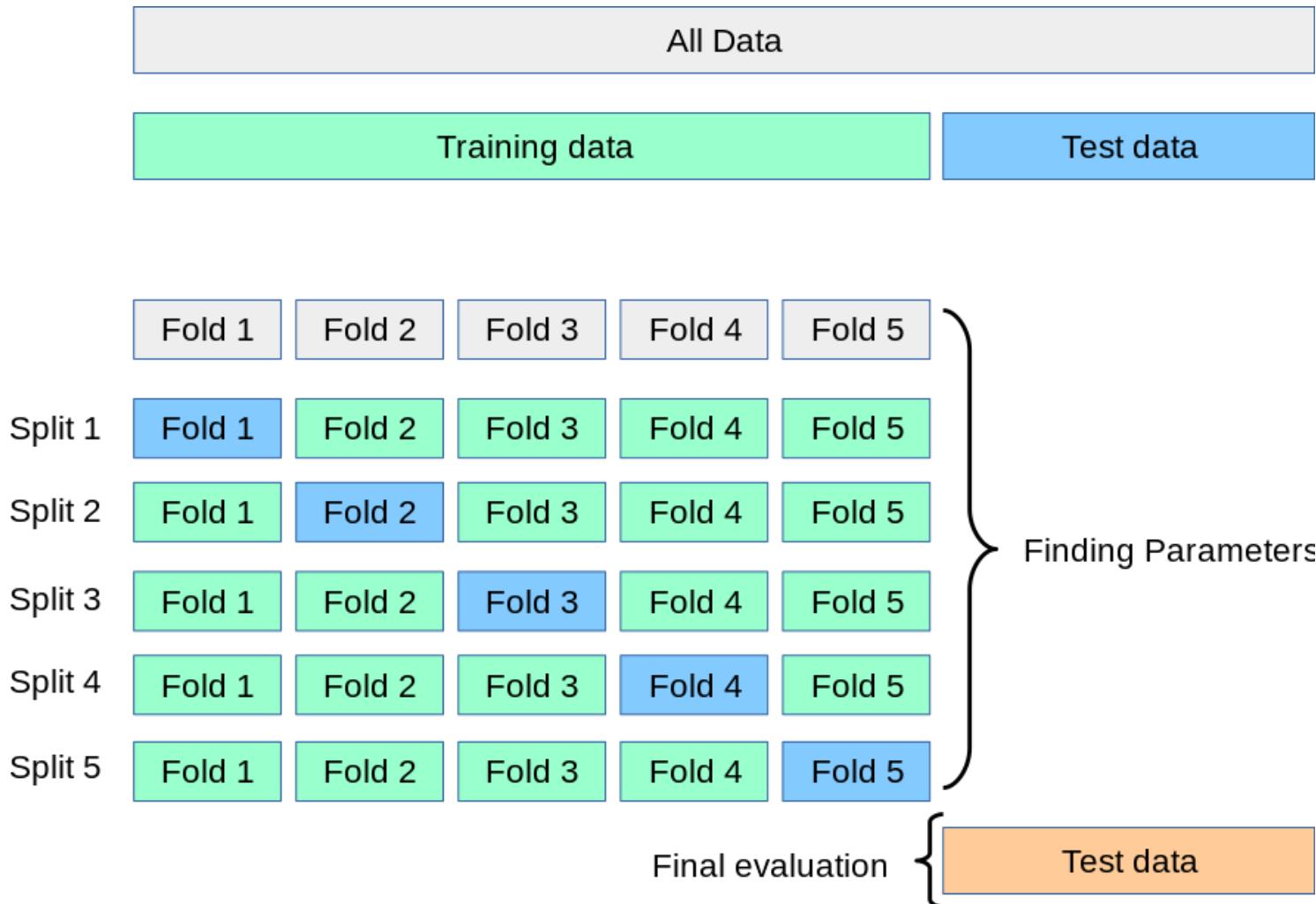
=



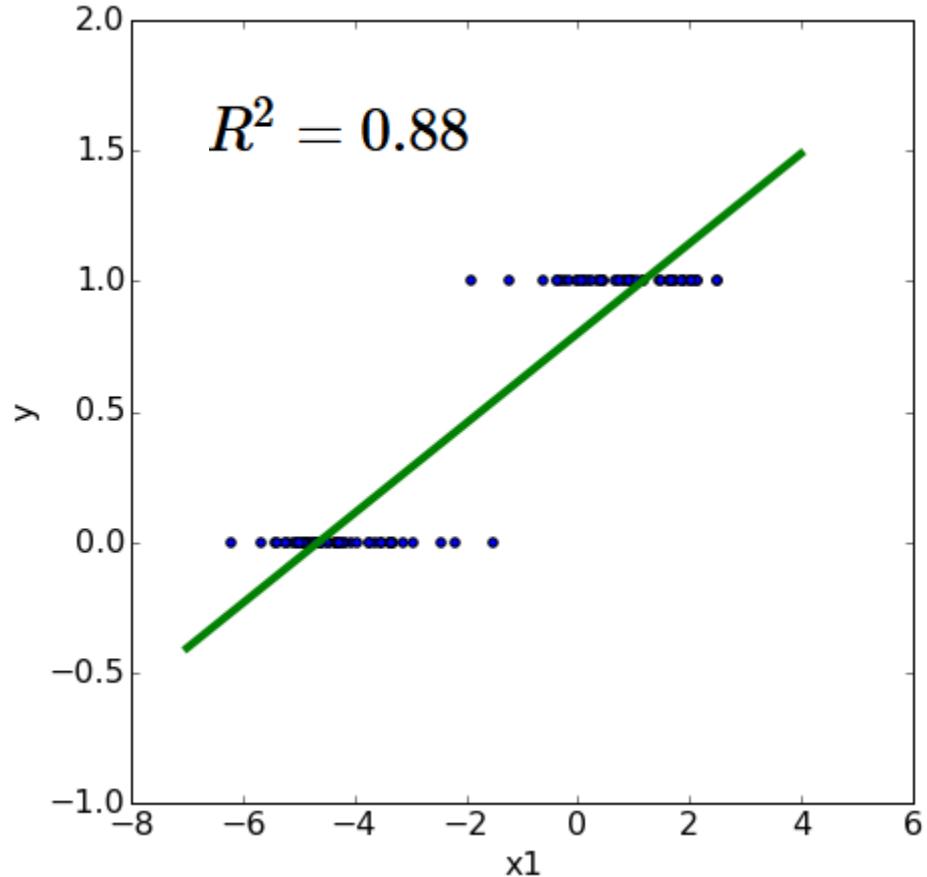
# Model selection: overfitting



# Model evaluation: cross-validation

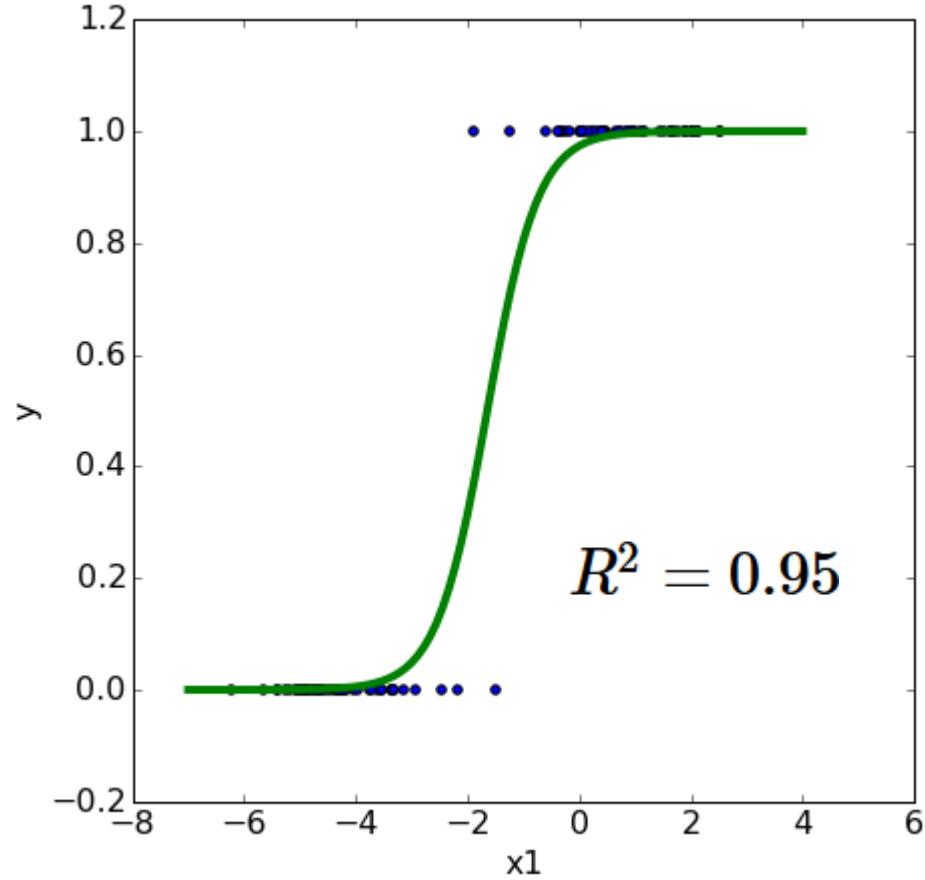


# Logistic regression



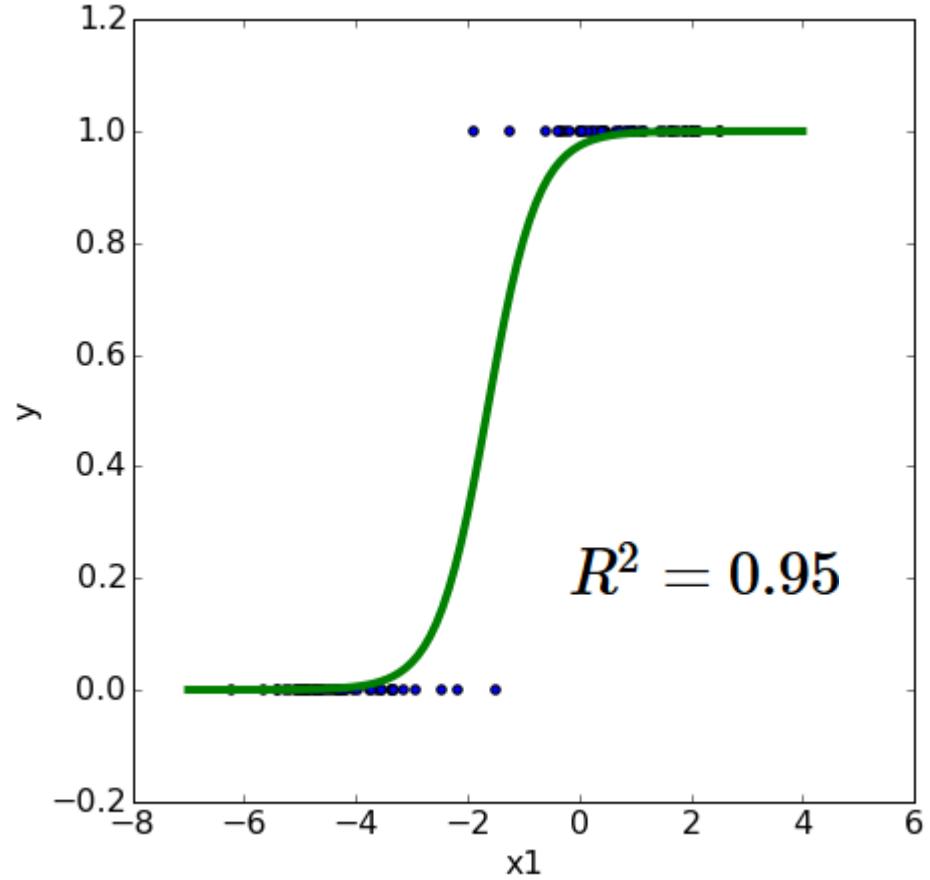
We need to make  
**assumptions** *linear relationship*  
about the  
**model** *linear model*  
that generated the data.

# Logistic regression



We need to make  
**assumptions** *linearly separable*  
about the  
**model** *logistic model*  
that generated the data.

# Logistic regression



$$f(x, \theta) = g(\theta_0 + \theta_1 x_1)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

## Logistic regression: cost function

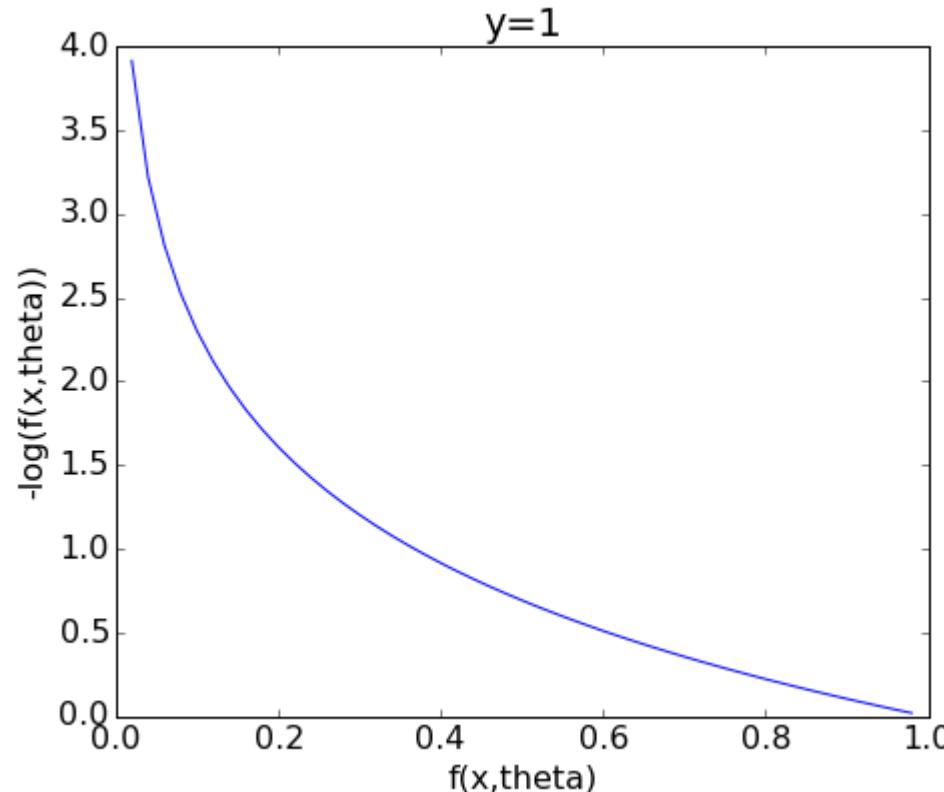
$$J(\theta) = -\left[ \frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}, \theta)) \right]$$

We know that  $y^{(i)}$  is either 0 or 1. If  $y^{(i)} = 1$  then the cost function  $J(\theta)$  is incremented by  $-\log(f(x^{(i)}, \theta))$ .

Similarly, if  $y^{(i)} = 0$  then the cost function  $J(\theta)$  is incremented by  $-\log(1 - f(x^{(i)}, \theta))$ .

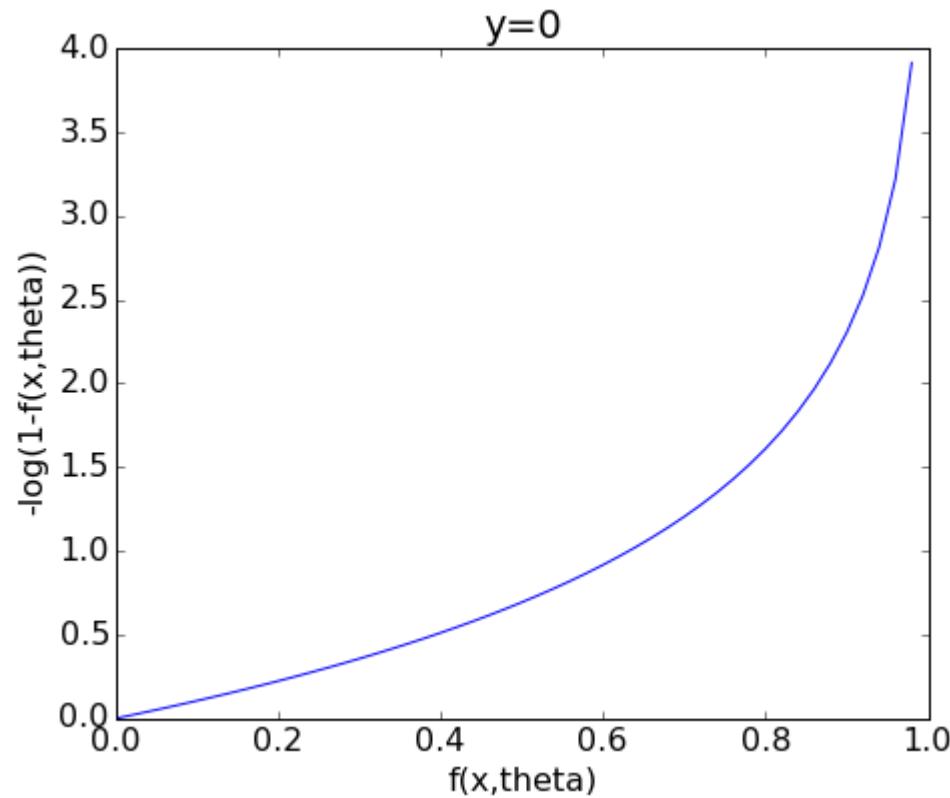
## Logistic regression: cost function

We know that  $y^{(i)}$  is either 0 or 1. If  $y^{(i)} = 1$  then the cost function  $J(\theta)$  is incremented by  $-\log(f(x^{(i)}, \theta))$ .



## Logistic regression: cost function

Similarly, if  $y^{(i)} = 0$  then the cost function  $J(\theta)$  is incremented by  
 $-\log(1 - f(x^{(i)}, \theta))$ .



# Logistic regression

Fit a logistic model

$$f(x, \theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m) = g(\theta' x)$$

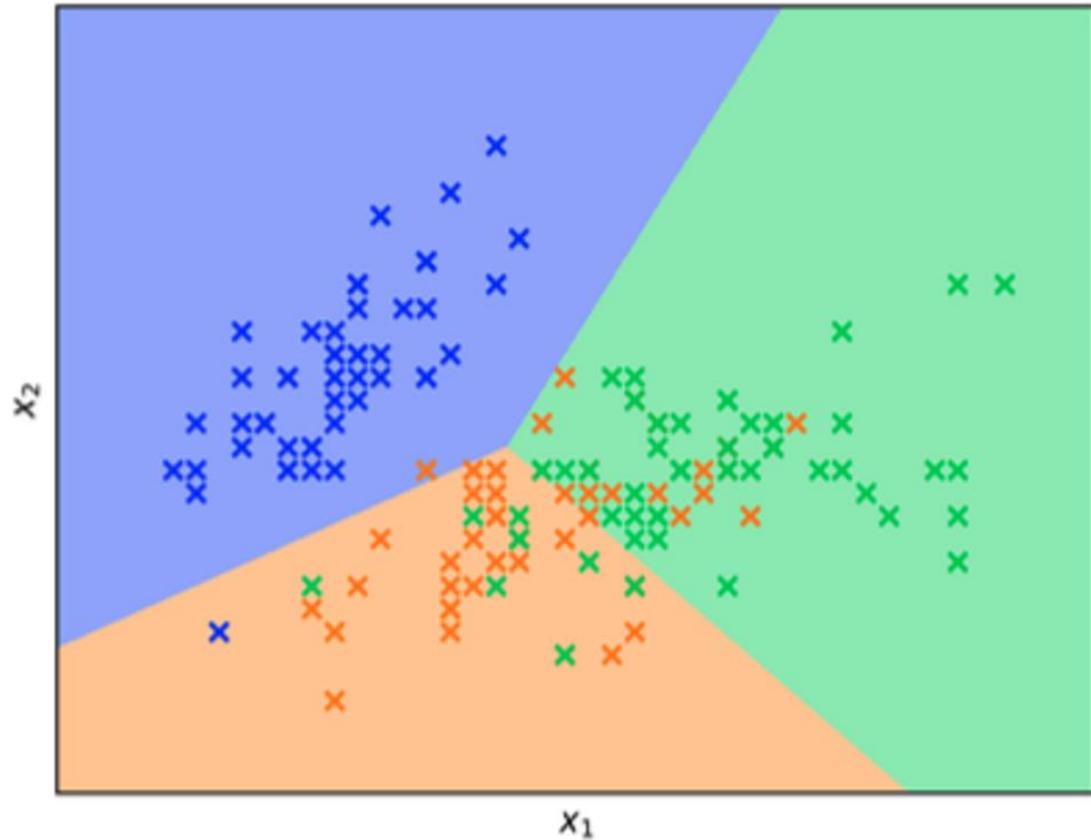
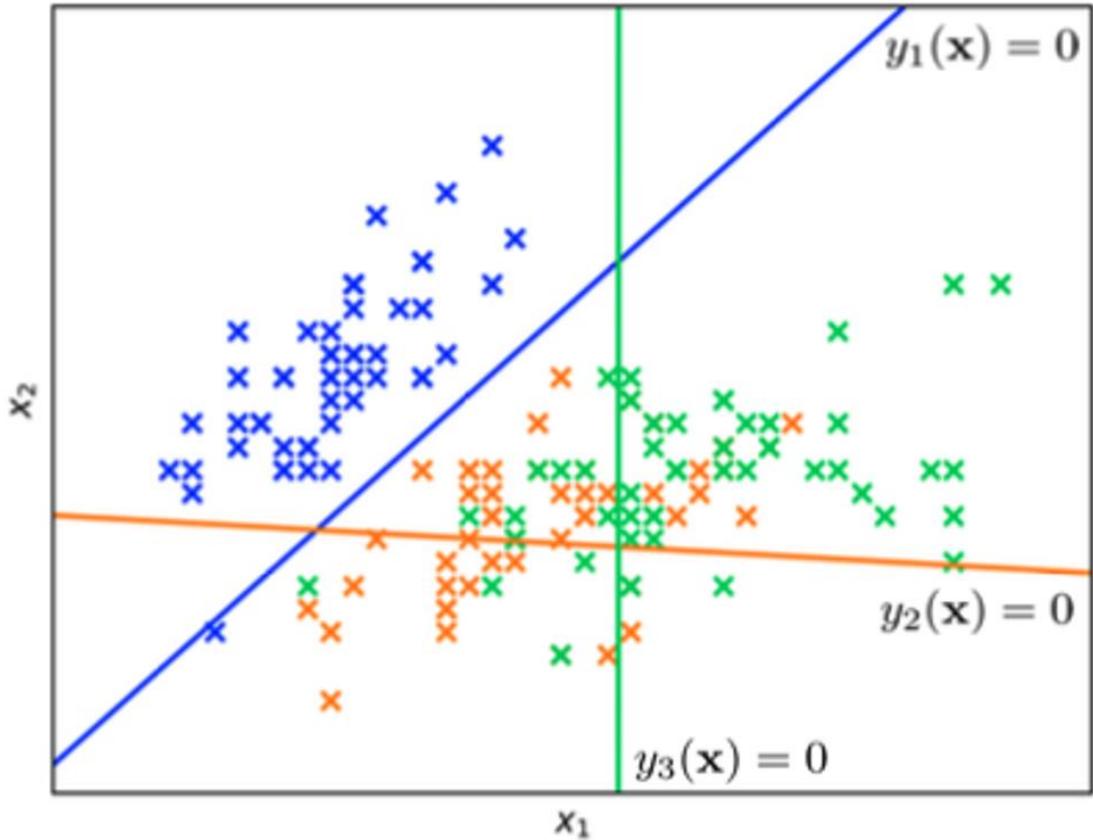
to the data set such that the cost function

$$J(\theta) = -\left[\frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}, \theta))\right]$$

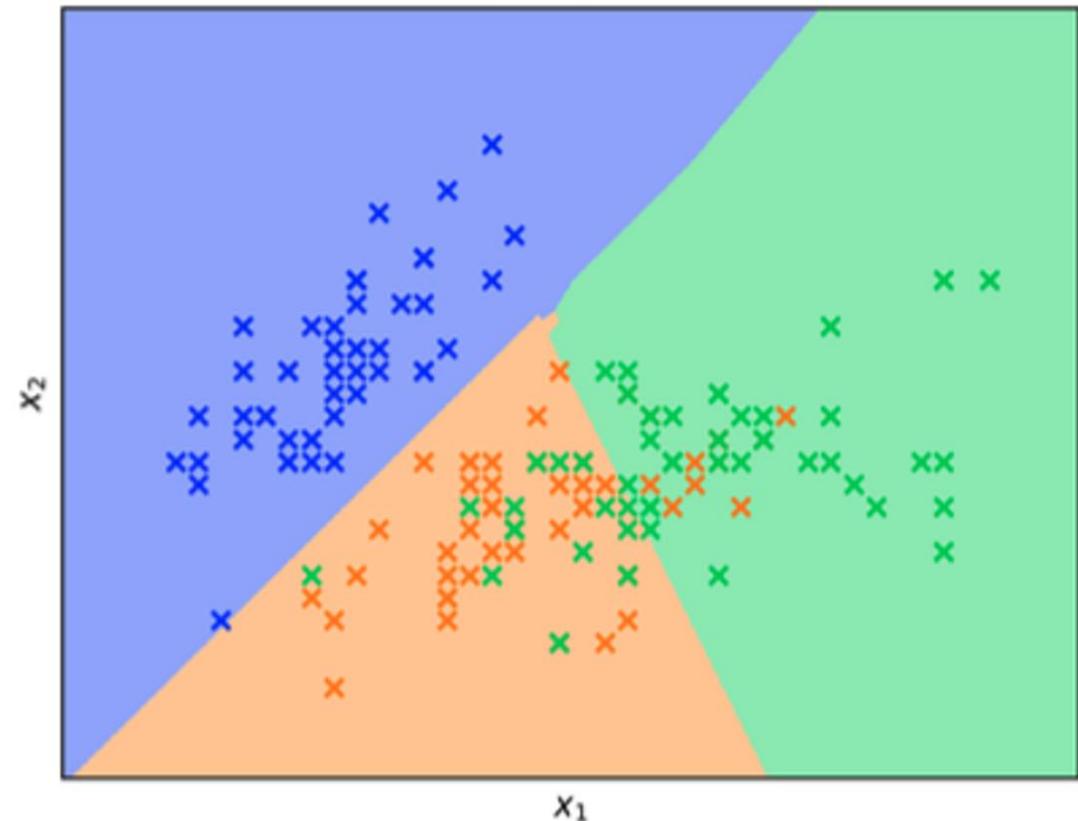
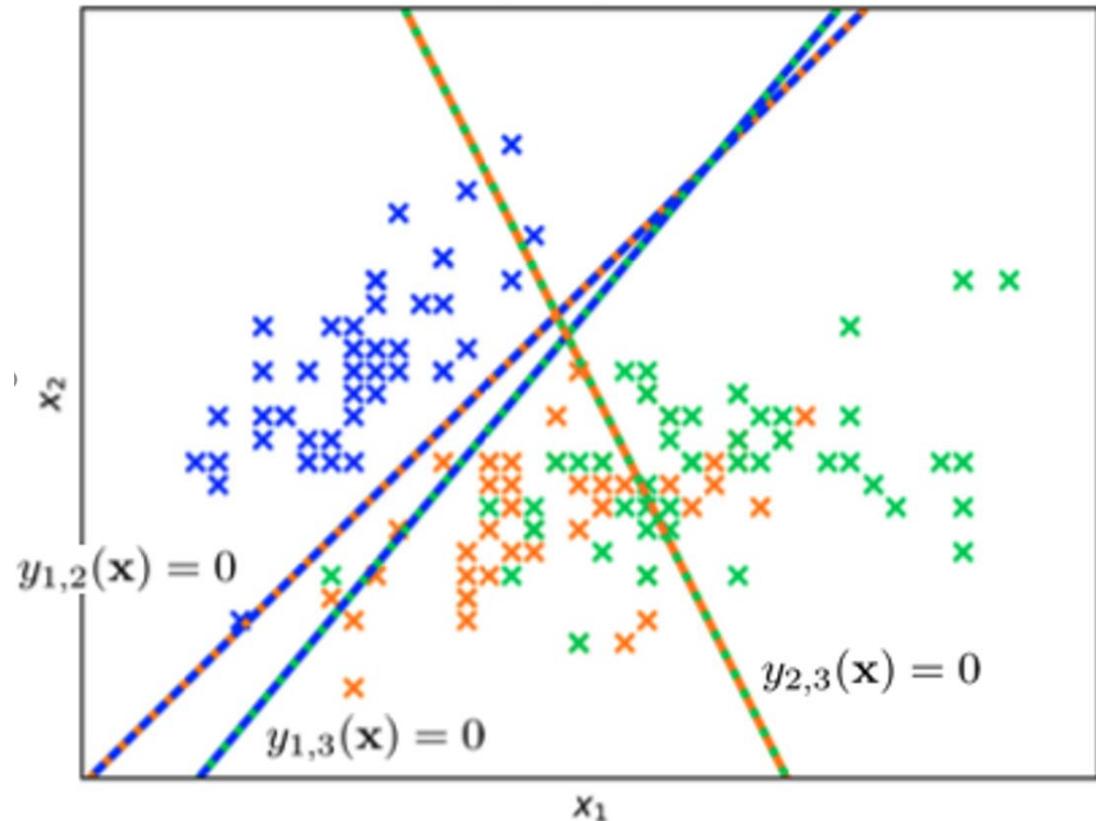
is minimal using gradient descent

$$\theta_j := \theta_j - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_j^{(i)}$$

## Muti-class classification: one vs rest



## Muti-class classification: one vs one



# Let's build a model!

## Backpropagation Applied to Handwritten Zip Code Recognition

Y. LeCun  
B. Boser  
J. S. Denker  
D. Henderson  
R. E. Howard  
W. Hubbard  
L. D. Jackel

AT&T Bell Laboratories Holmdel, NJ 07733 USA

The ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. This paper demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. This approach has been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service. A single network learns the entire recognition operation, going from the normalized image of the character to the final classification.

80322-4129 80206

40004 14310

37878 05153

~~35502~~ 75216

35460 44209

101191348572680322414186  
6359720299299722510046701  
3084111591010615406103631  
1064111030475262009979966  
8912086708557131427955460  
1018730187112993089970984  
0109707597331972015519055  
1075318255182814358090943  
17875416554603546055  
18255108503047520439401

# Let's build a model!

scikit-learn [Install](#) [User Guide](#) [API](#) [Examples](#) [Community](#) [More ▾](#)  [Go](#)

## scikit-learn

Machine Learning in Python

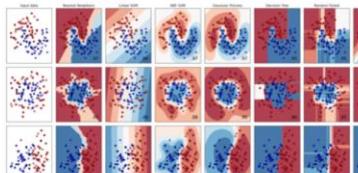
[Getting Started](#) [Release Highlights for 1.0](#) [GitHub](#)

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

### Classification

Identifying which category an object belongs to.

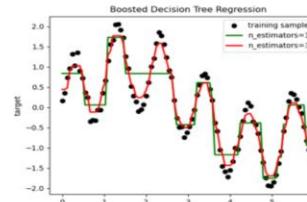
**Applications:** Spam detection, image recognition.  
**Algorithms:** SVM, nearest neighbors, random forest, and more...



### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.  
**Algorithms:** SVR, nearest neighbors, random forest, and more...



### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes  
**Algorithms:** k-Means, spectral clustering, mean-shift, and more...

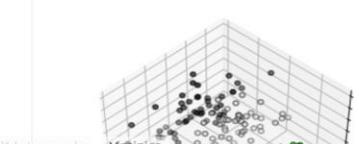


### Examples

### Dimensionality reduction

Reducing the number of random variables to consider.

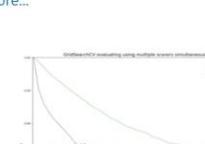
**Applications:** Visualization, Increased efficiency  
**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...



### Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning  
**Algorithms:** grid search, cross validation, metrics, and more...

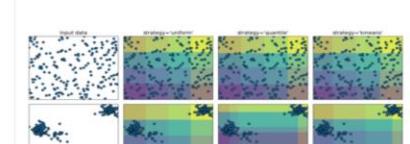


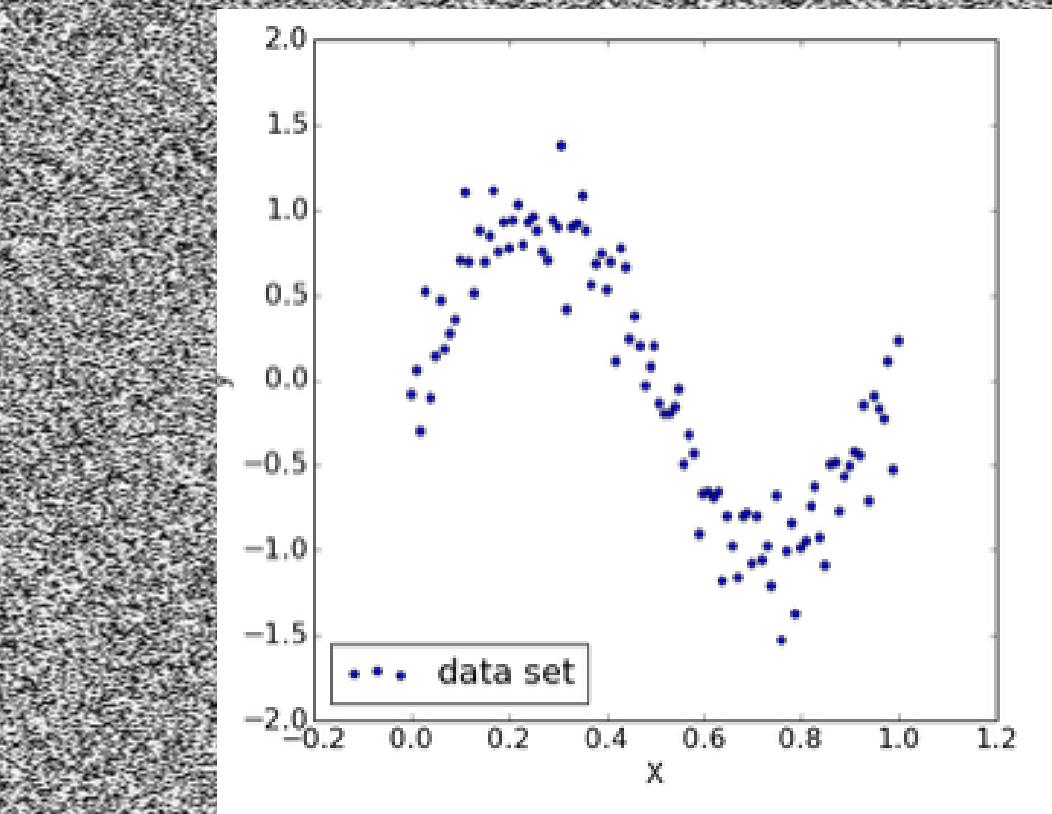
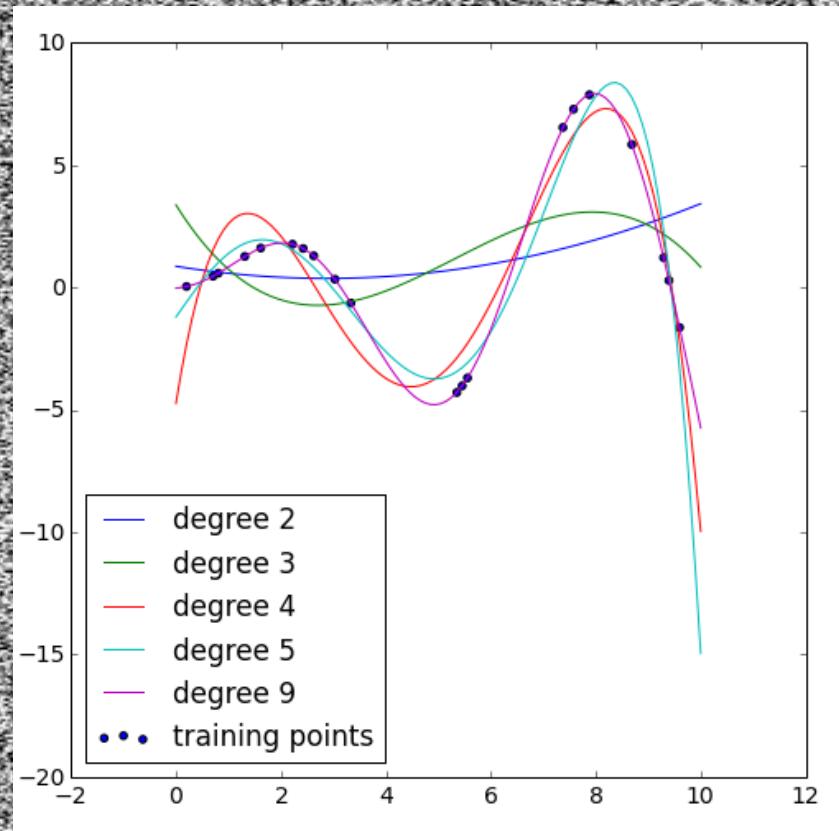
### Examples

### Preprocessing

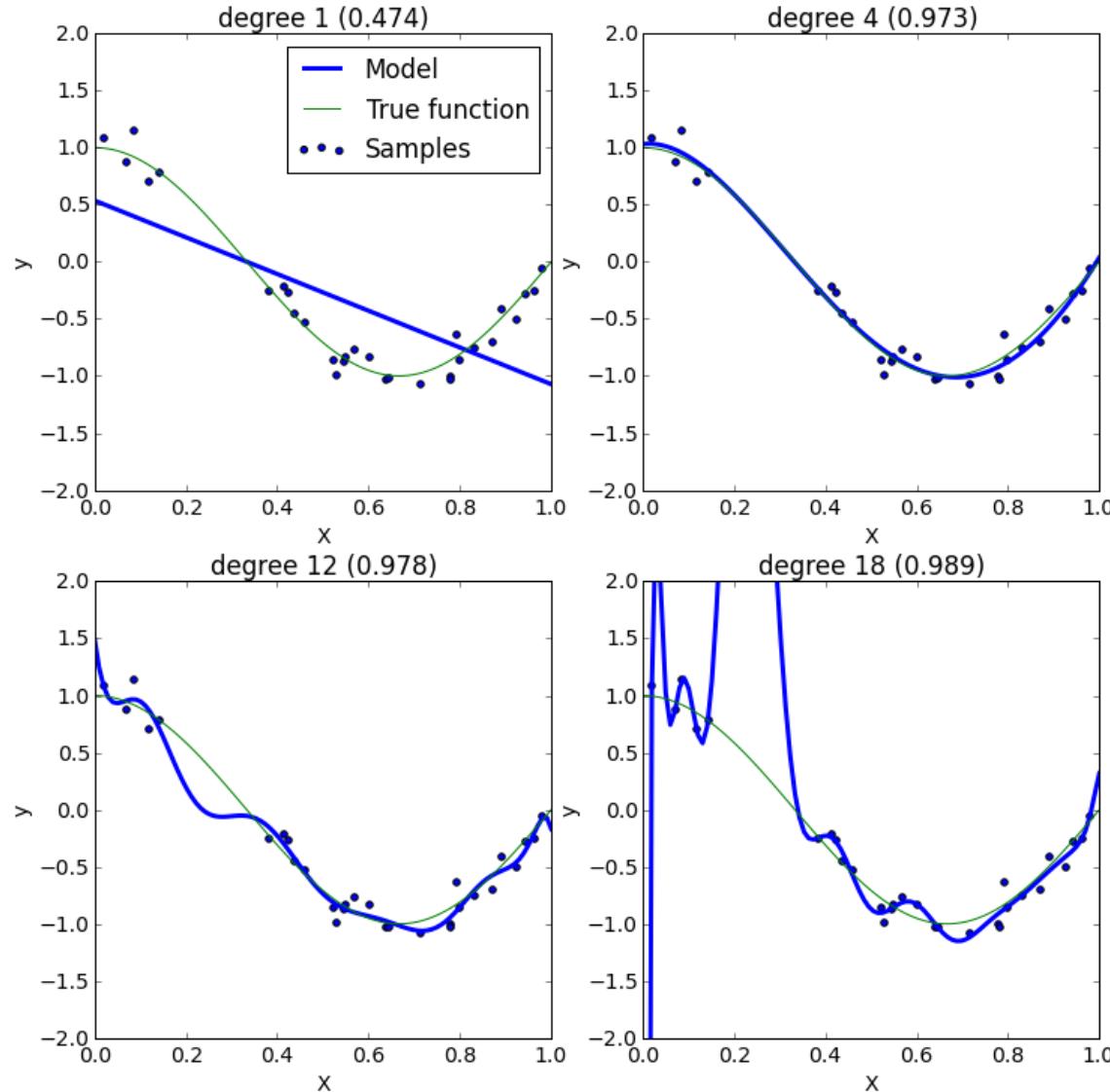
Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.  
**Algorithms:** preprocessing, feature extraction, and more...



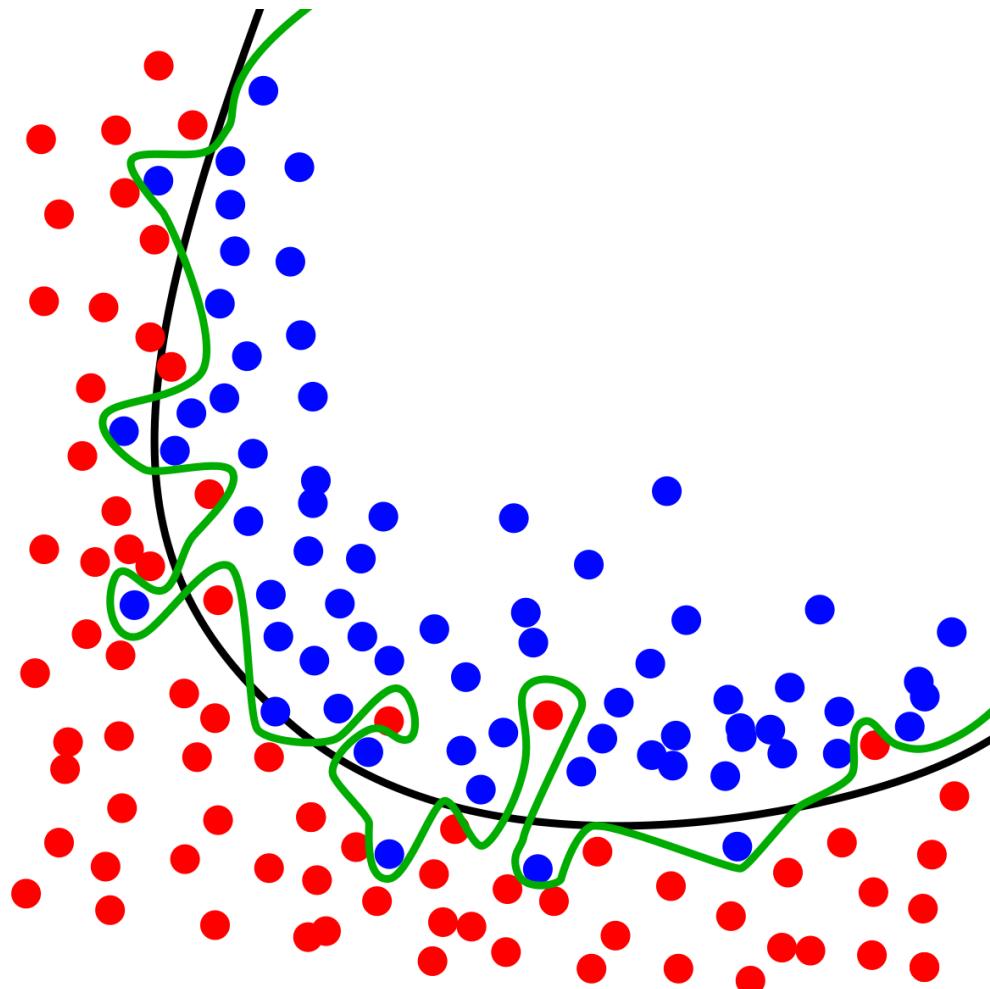


# Model regularization



- noise
- label
- feature
- feature relevance
- relatively small train sets
- don't try to fit the data perfectly
- don't try to use all features
- notice how  $R^2$  on the train set does increase with  $d$

# Model regularization



- noise
- label
- feature
  
- feature relevance
  
- relatively small train sets
  
- don't try to fit the data perfectly
  
- don't try to use all features
  
- notice how the accuracy on the train set increases with  $d$

# Regularized linear regression

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

*cost function*

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)})^2 + \lambda \sum_{j=1}^m \theta_j^2$$

*regularized cost function*

$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{n} \sum_{i=1}^n (f(x^{(i)}, \theta) - y^{(i)}) x_j^{(i)} - \frac{\lambda}{n} \theta_j$$

# Regularized logistic regression

$$f(x, \theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m)$$

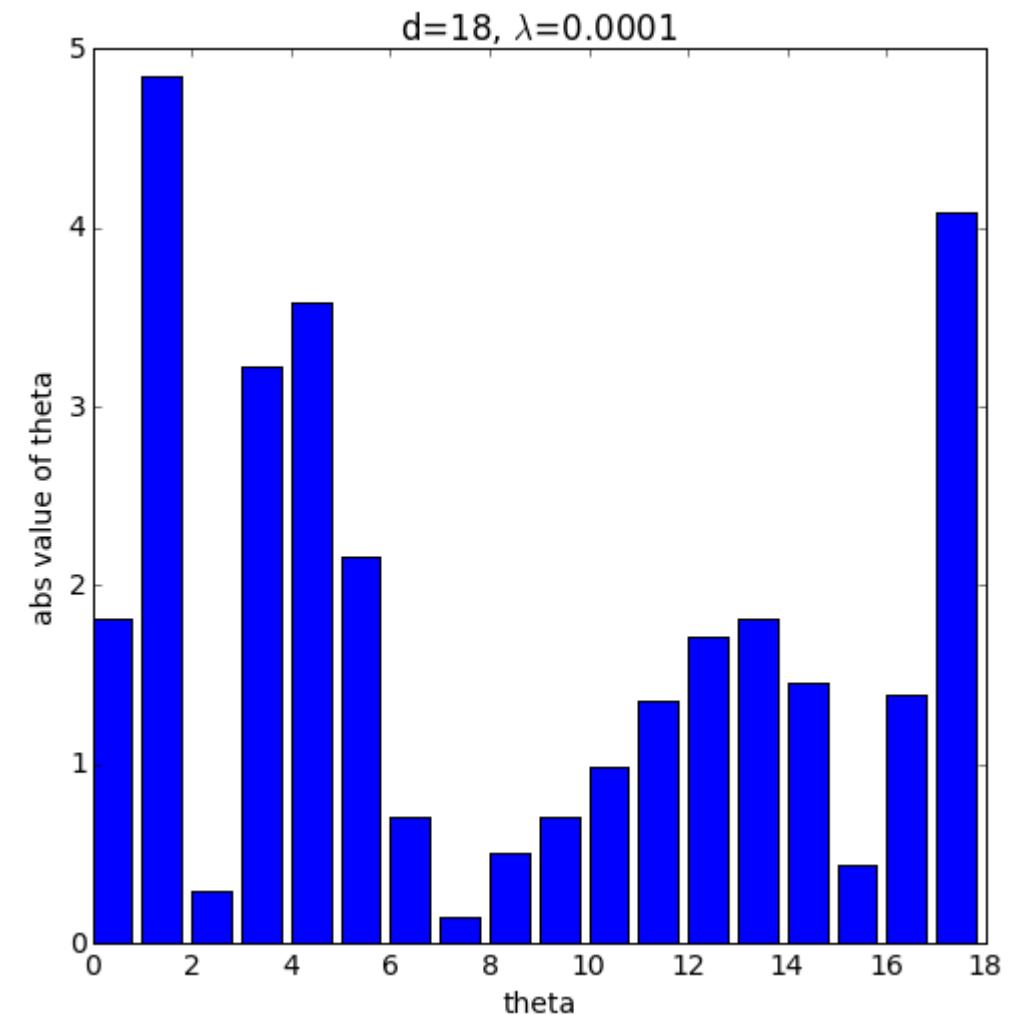
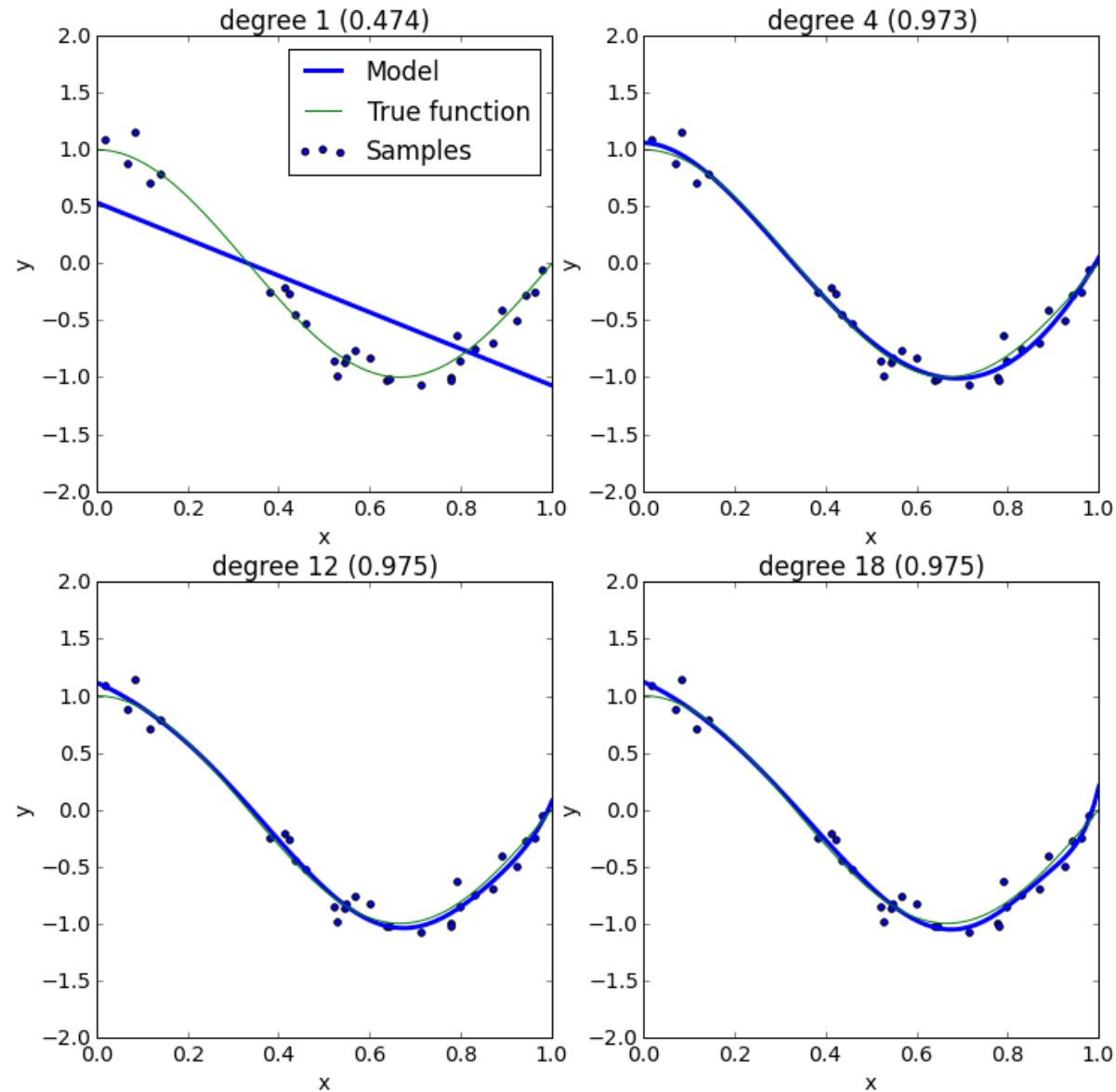
*cost function*

$$J(\theta) = -\left[ \frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}, \theta)) \right]$$

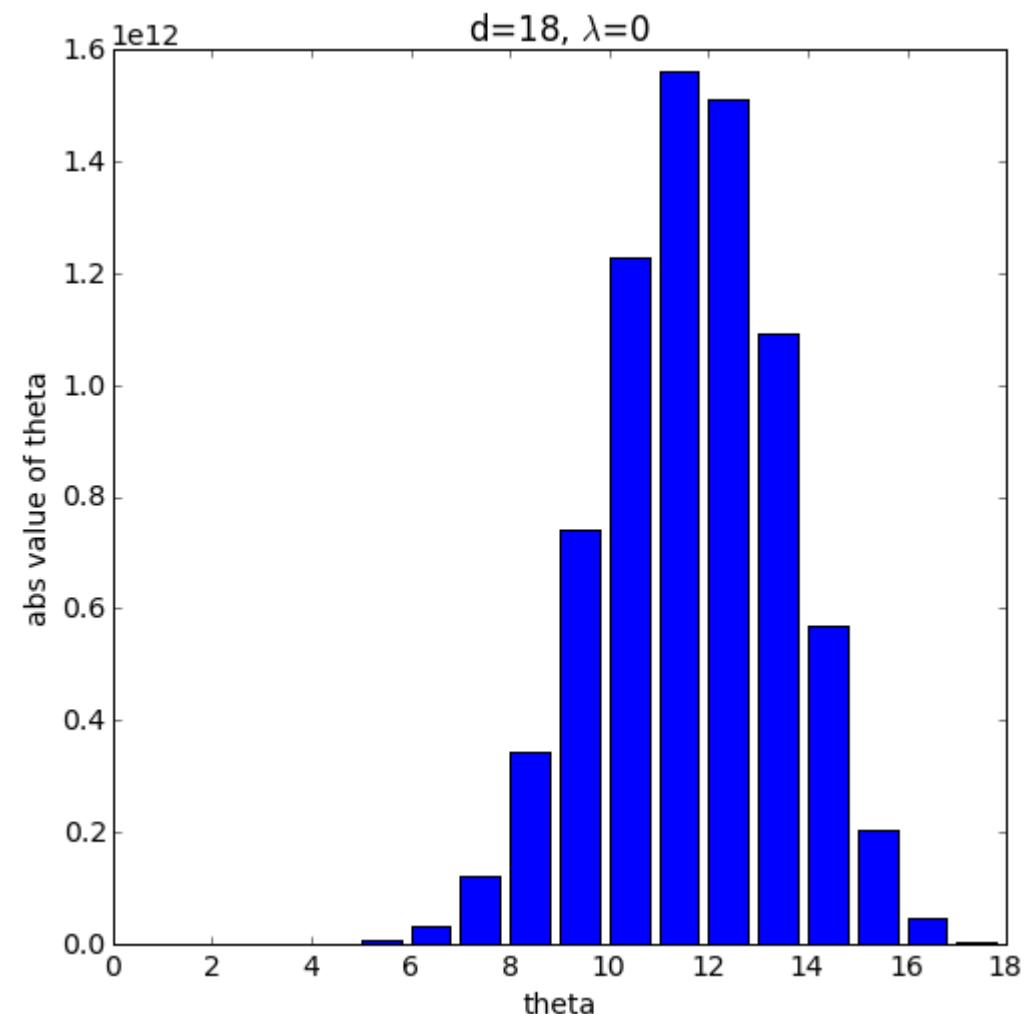
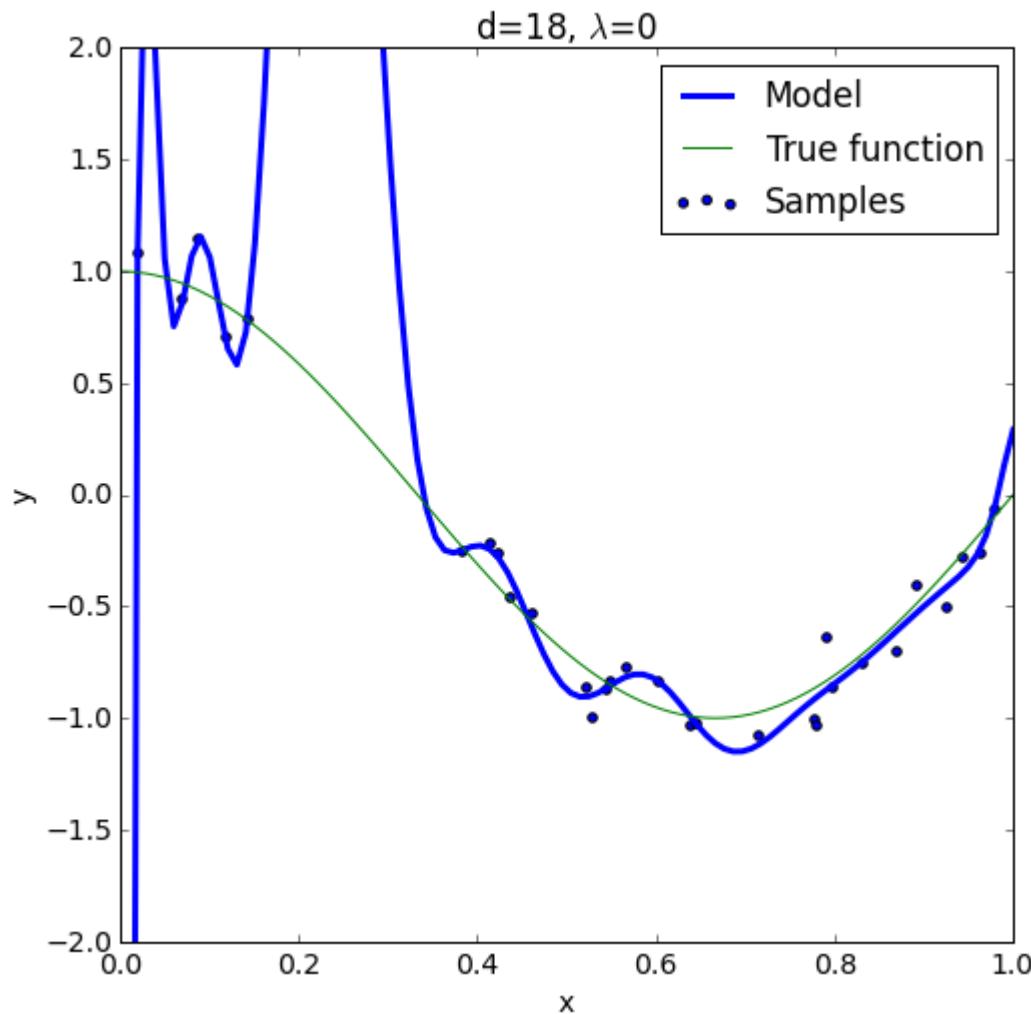
$$J(\theta) = -\left[ \frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}, \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}, \theta)) \right] + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

*regularized cost function*

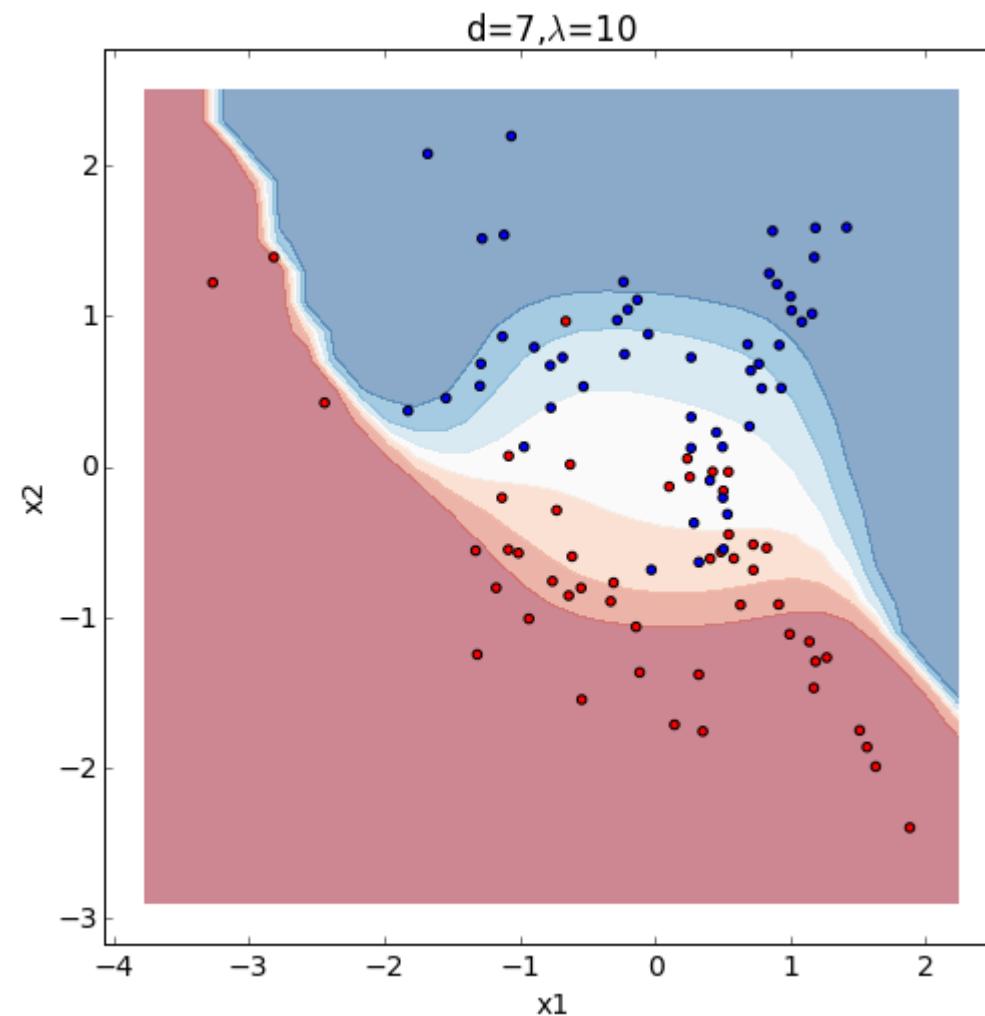
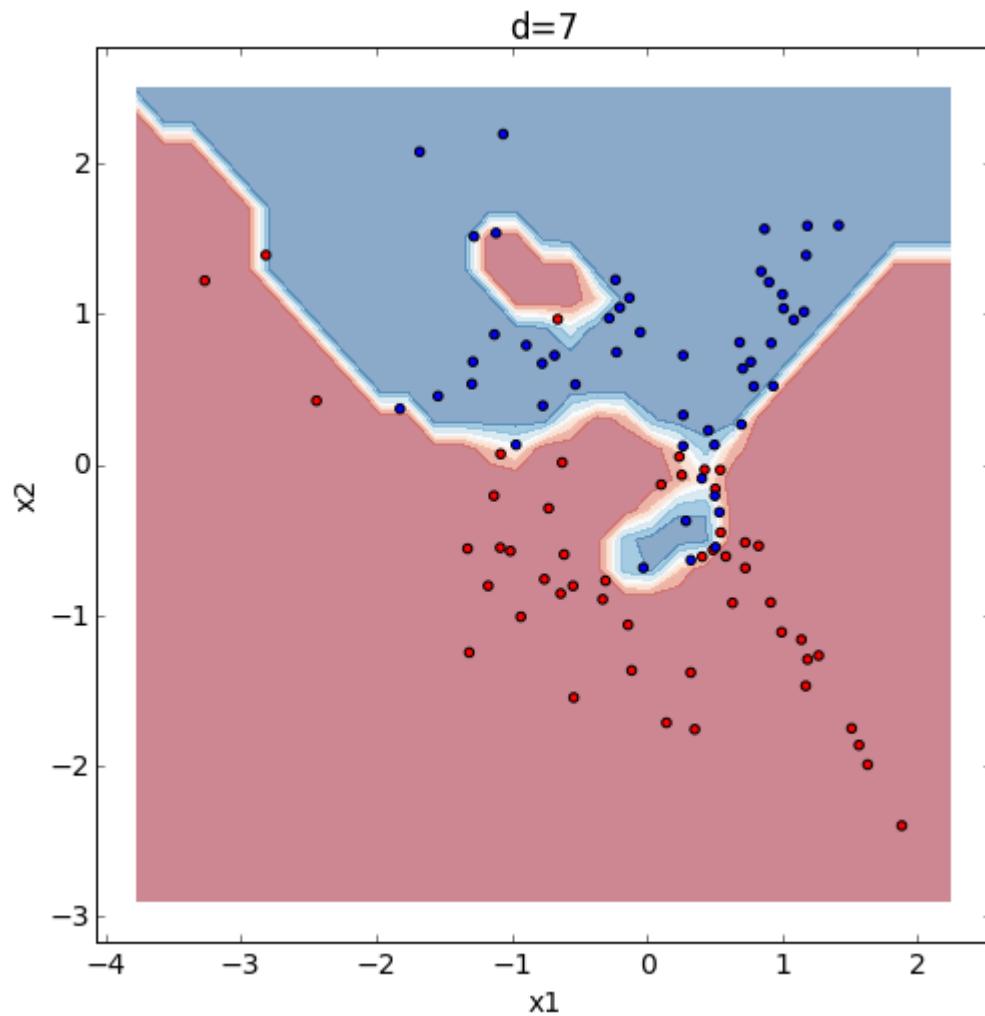
# Regularized linear regression



# Regularized linear regression

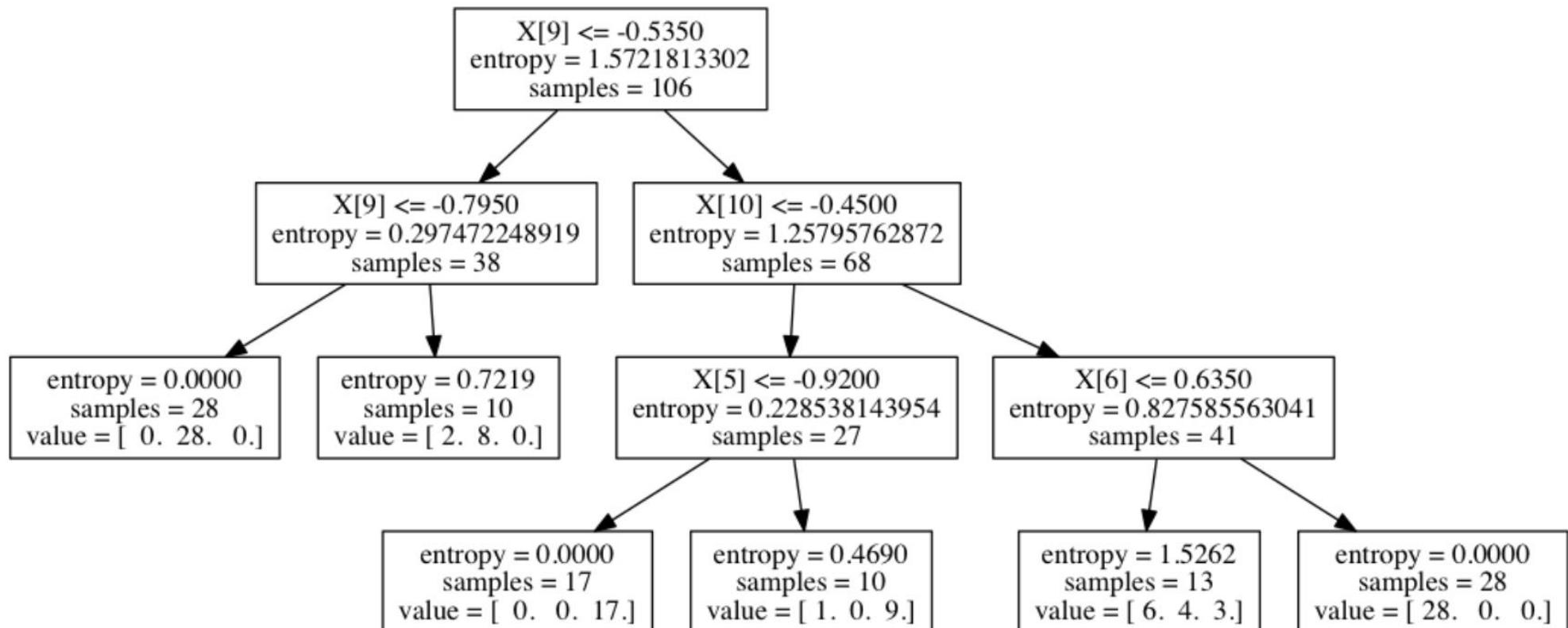


# Regularized linear regression

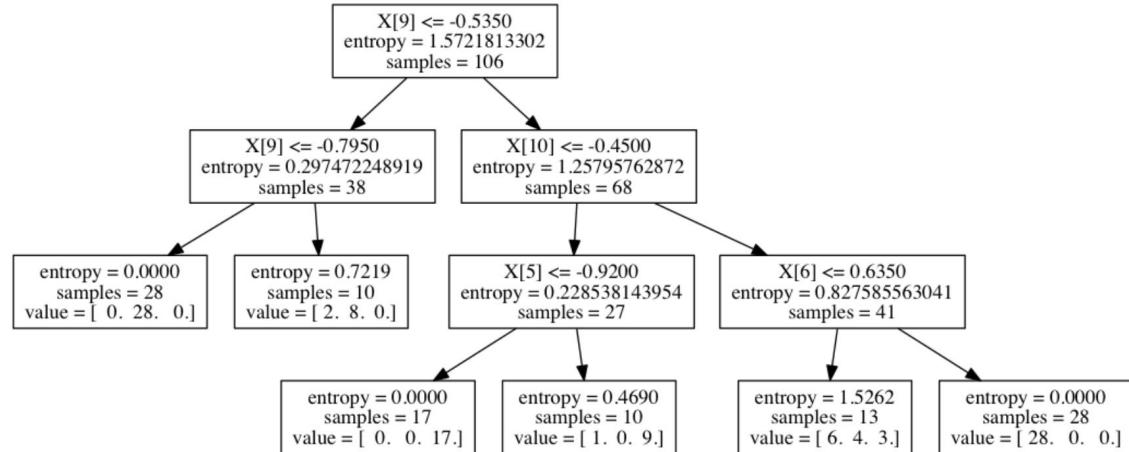


# Decision trees

$$G(D, x) = \text{Entropy}(D) - \sum_{v \in x} \frac{|D_v|}{|D|} \text{Entropy}(D_v)$$



# Decision trees



advantages:

- ease of interpretation
- handles continuous and discrete features
- invariant to monotone transformation of features
- variable selection automated
- **low bias (deep trees)**

disadvantages:

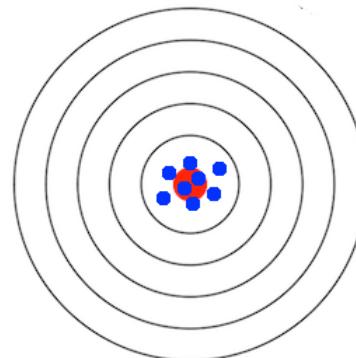
- **high variance**
- overfitting

# Bias / Variance

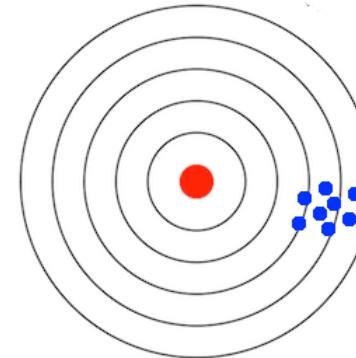
- Error due to bias
- Error due to variance
- Irreducible error

Low  
variance

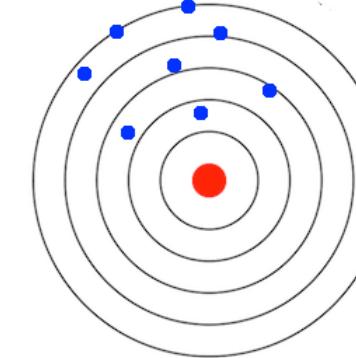
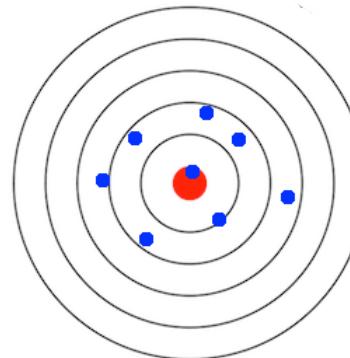
Low bias



High bias

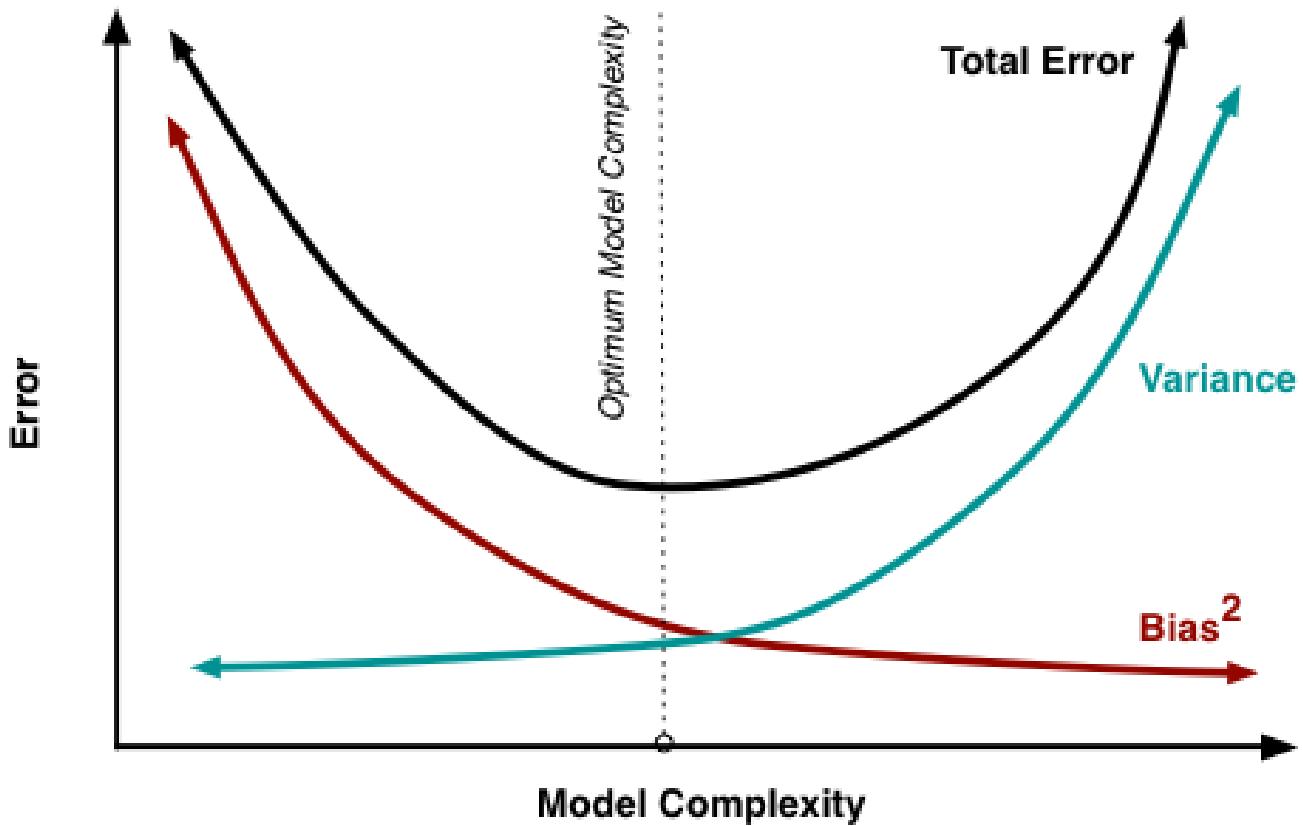


High  
variance

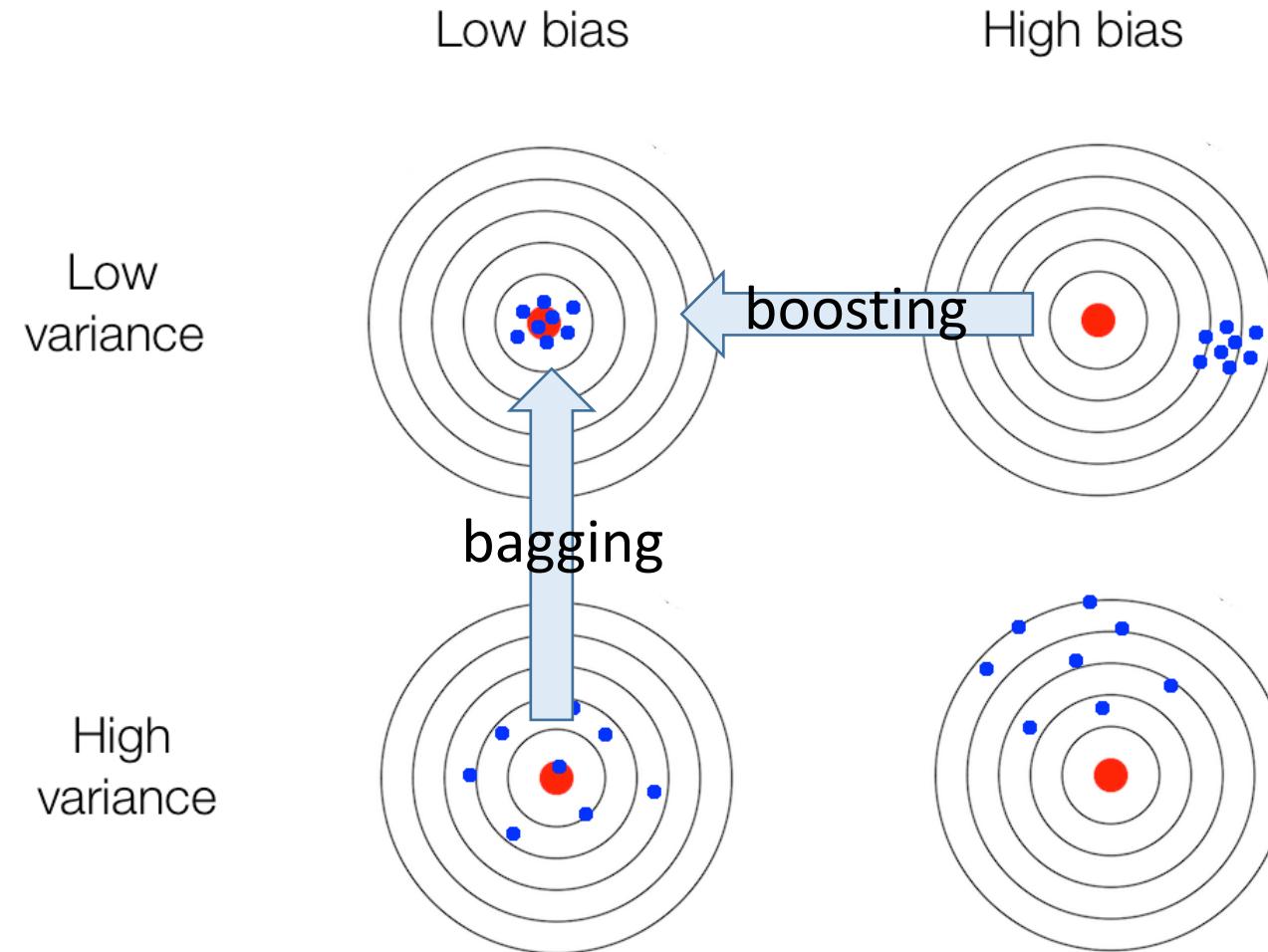


# Bias / Variance

- Error due to bias
- Error due to variance
- Irreducible error



# Bagging / Boosting



# Bagging

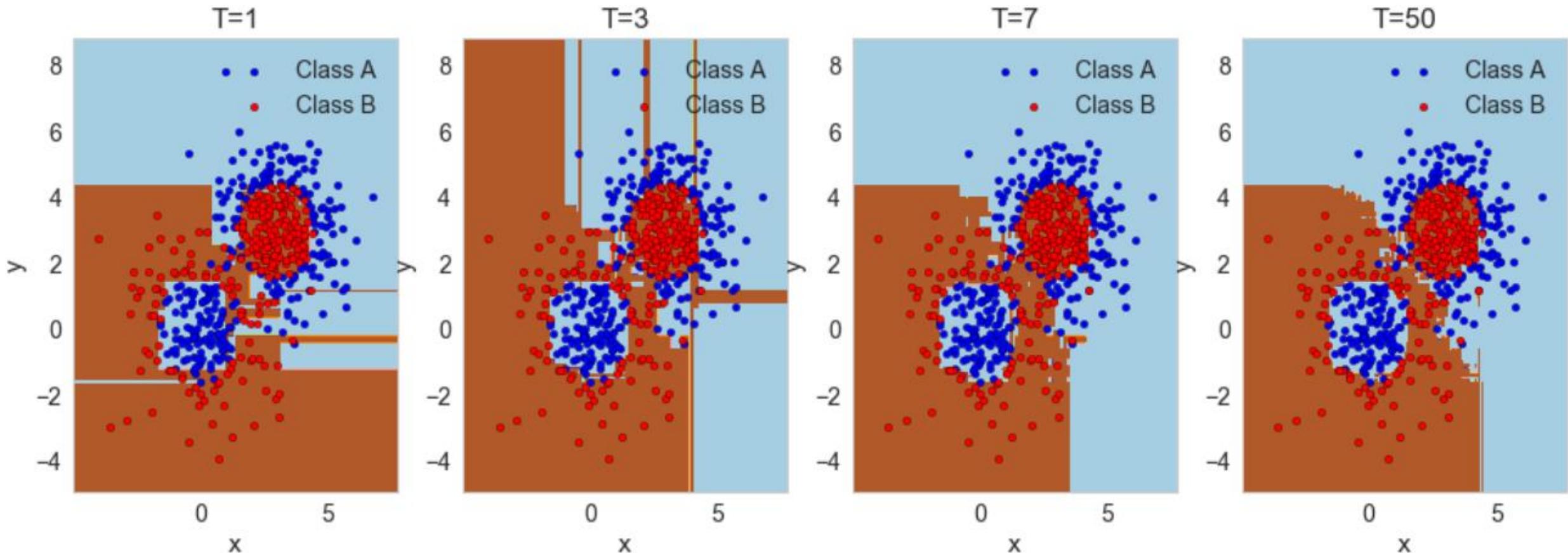
- Simulate the notion of different train set samples:
  1. sample data points from train set to create new train set
  2. fit low bias high variance model on new train set
  3. repeat steps (1) and (2)  $T$  times
  4. average the predictions of the  $T$  models

$$\hat{f}(x, \theta) = \frac{1}{T} \sum_{t=1}^T f_t(x, \theta)$$

# Bagging: Random Forest

- Train set contains  $n$  data points with  $m$  features.
- Construct  $T$  low bias high variance decision trees by following these steps:
  1. Sample  $n$  data points at random **with replacement** from the train set.
  2. At each node, select  $h \ll m$  features at random and compute the best split using only these  $h$  features.
  3. Each tree is grown to the largest extent possible. There is no pruning or early stopping.
- Step 3 ensures that the bagged models are low bias by learning deep complex decision trees.

# Bagging: Random Forest



# Boosting

- reduce the bias of a high bias low variance model
- turning an ensemble of weak learners into a strong learner
- the meta-model is additive, i.e. adaboost:

$$\hat{f}(x, \theta) = \sum_{t=1}^T \alpha_t f_t(x, \theta)$$

# Boosting: Adaboost

Initialize the weights  $D_1(i) = 1/n, i = 1, 2, \dots, n.$

$$y_i \in \{-1, +1\}$$

For  $t = 1$  to  $T$ :

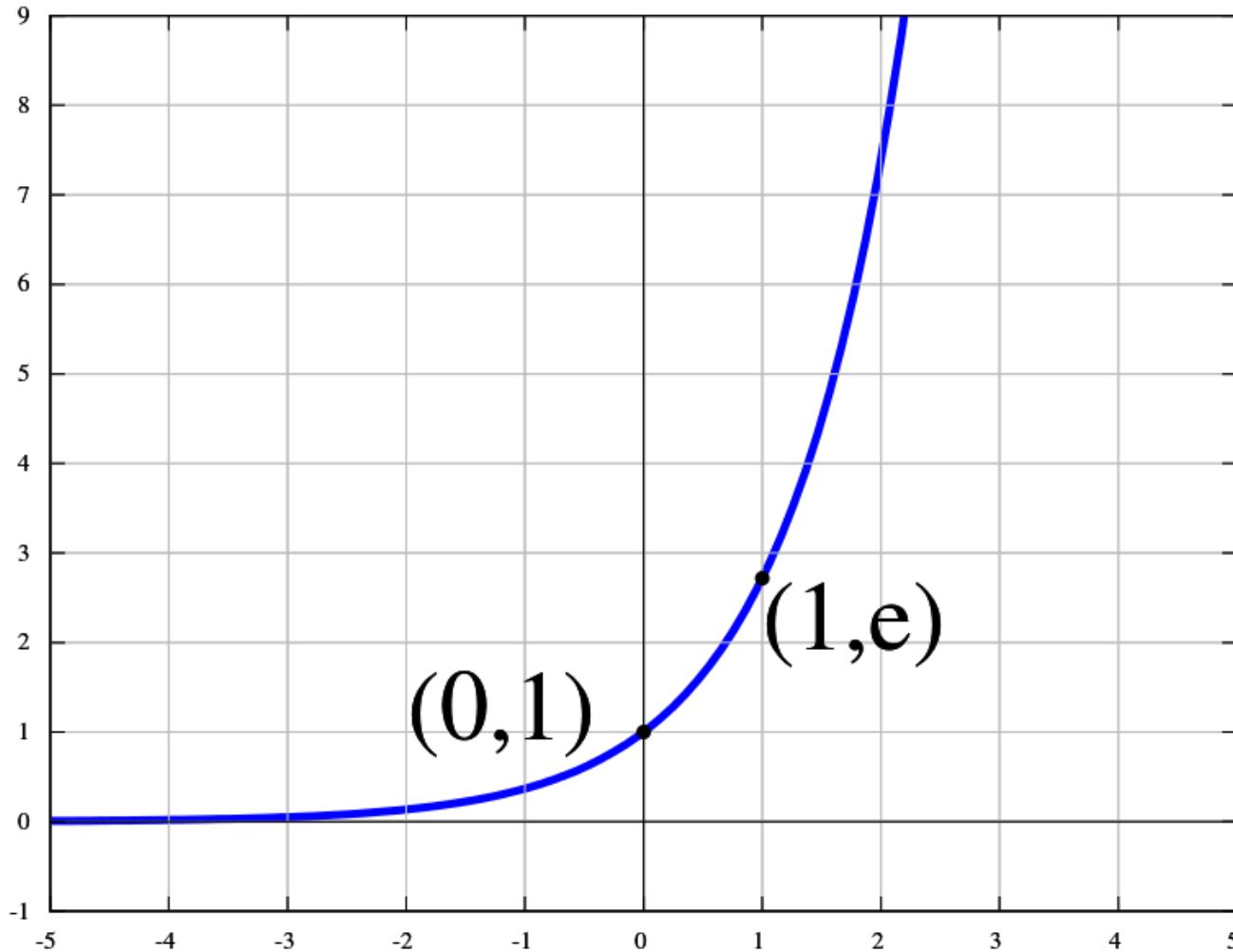
1. Fit a weak classifier  $f_t(x, \theta)$  to the trainset data using weights  $D_1(i).$
2. Set  $\alpha_t = \frac{1}{2} \ln(\frac{1 - \text{error}}{\text{error}}).$
3. Update weights:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i f_t(x_i, \theta))}{Z_t},$$

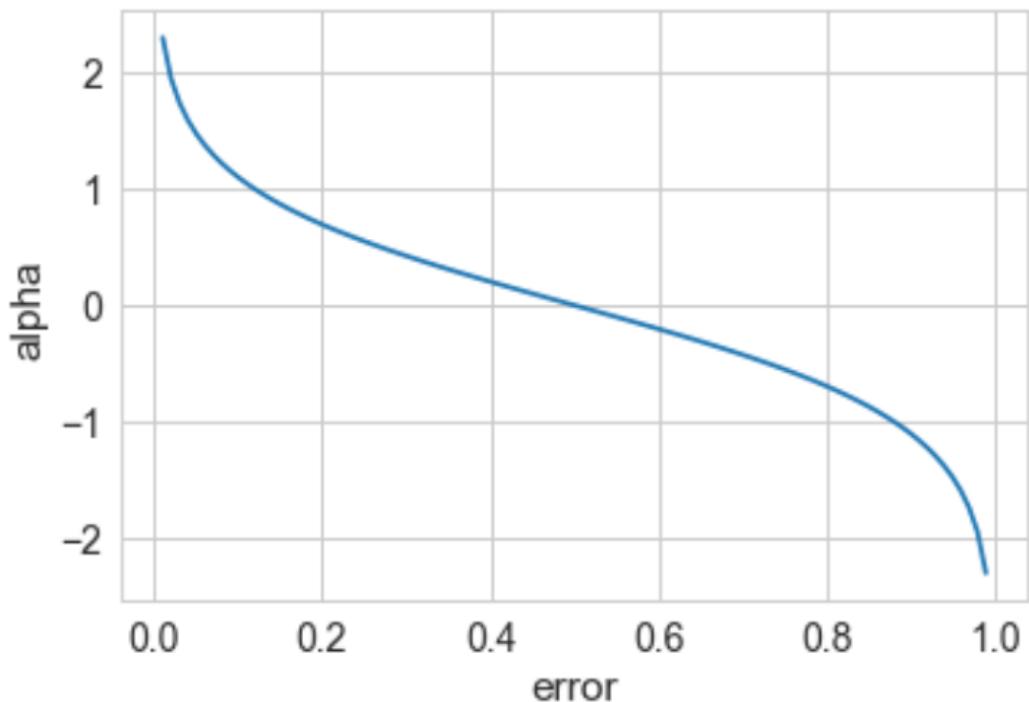
where  $Z_t$  is a normalizing factor that makes sure that  $\sum D_{t+1}(i) = 1.$

$$\hat{f}(x, \theta) = \sum_{t=1}^T \alpha_t f_t(x, \theta)$$

# Boosting: Adaboost



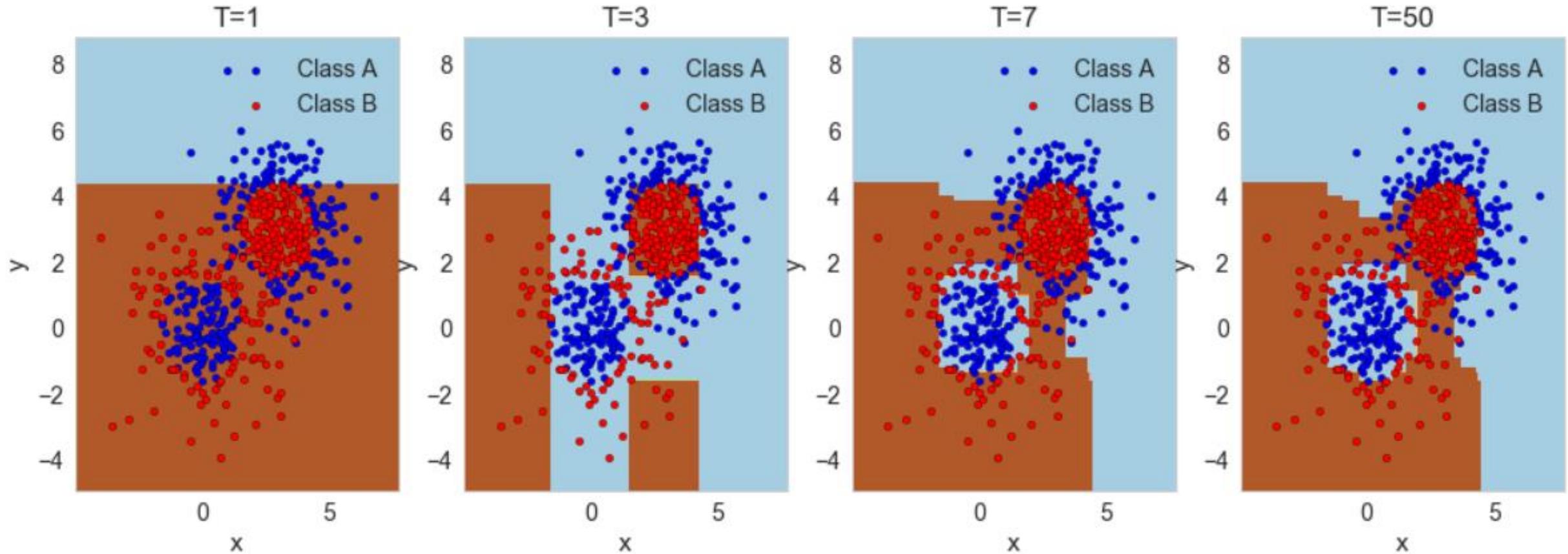
# Boosting: Adaboost



- The weight of a weak model in the boosted meta-model increases exponentially as the error approaches 0. Better models are given exponentially more weight.
- The weight is zero if the error rate is 0.5. A model with 50% accuracy is no better than random guessing, so it is ignored.
- The weight decreases exponentially as the error approaches 1. A negative weight is given to classifiers with worse than 50% accuracy.  
“Whatever that classifier says, do the opposite!”.  
This means that if a classifier has an error rate of 0.5 or higher, its weight will be negative, effectively reversing its prediction in the final ensemble.

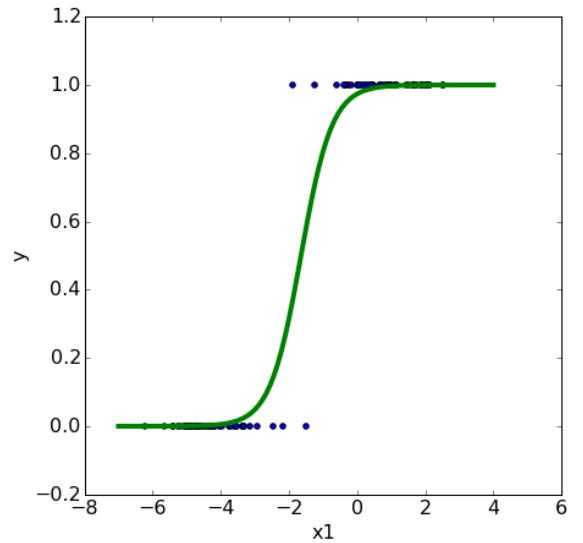
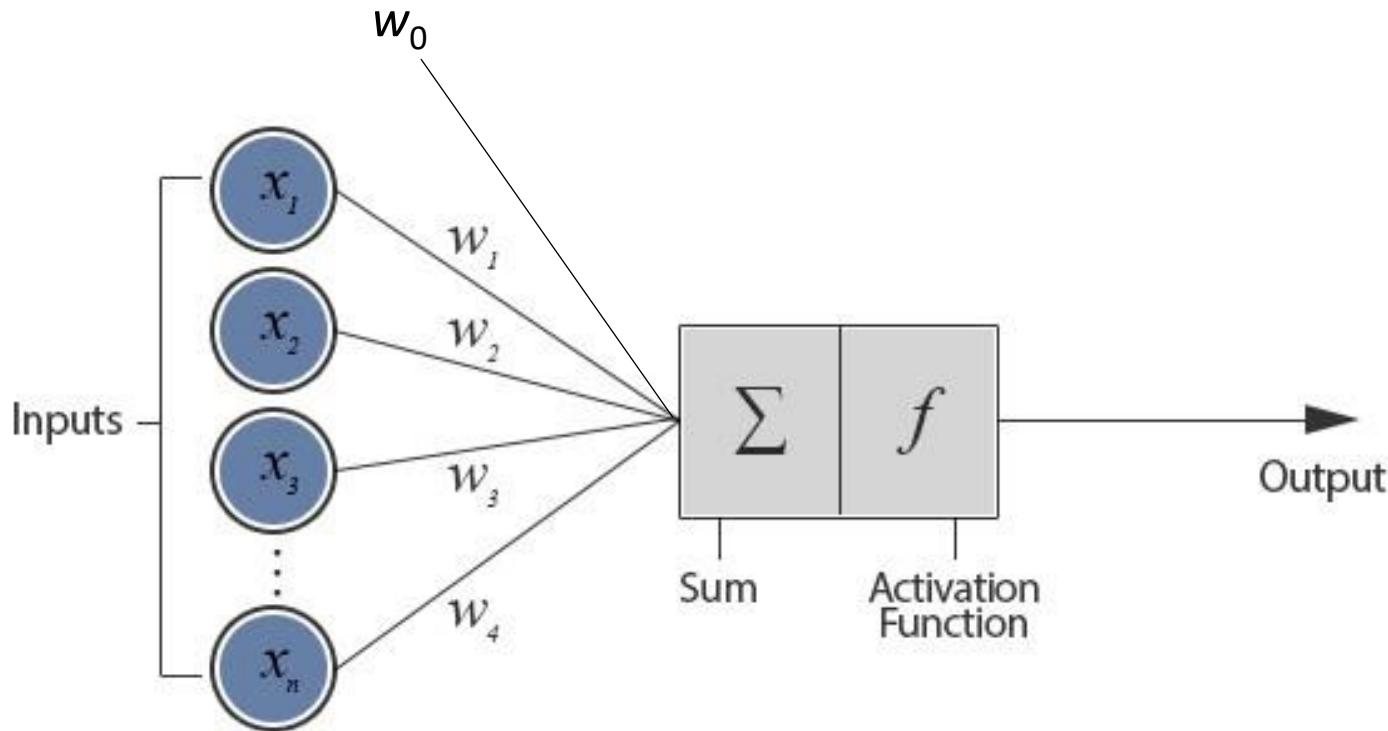
$$\hat{f}(x, \theta) = \sum_{t=1}^T \alpha_t f_t(x, \theta)$$

# Boosting: Adaboost



# Neural networks

$$f(x, \theta) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m) = g(\theta' x)$$



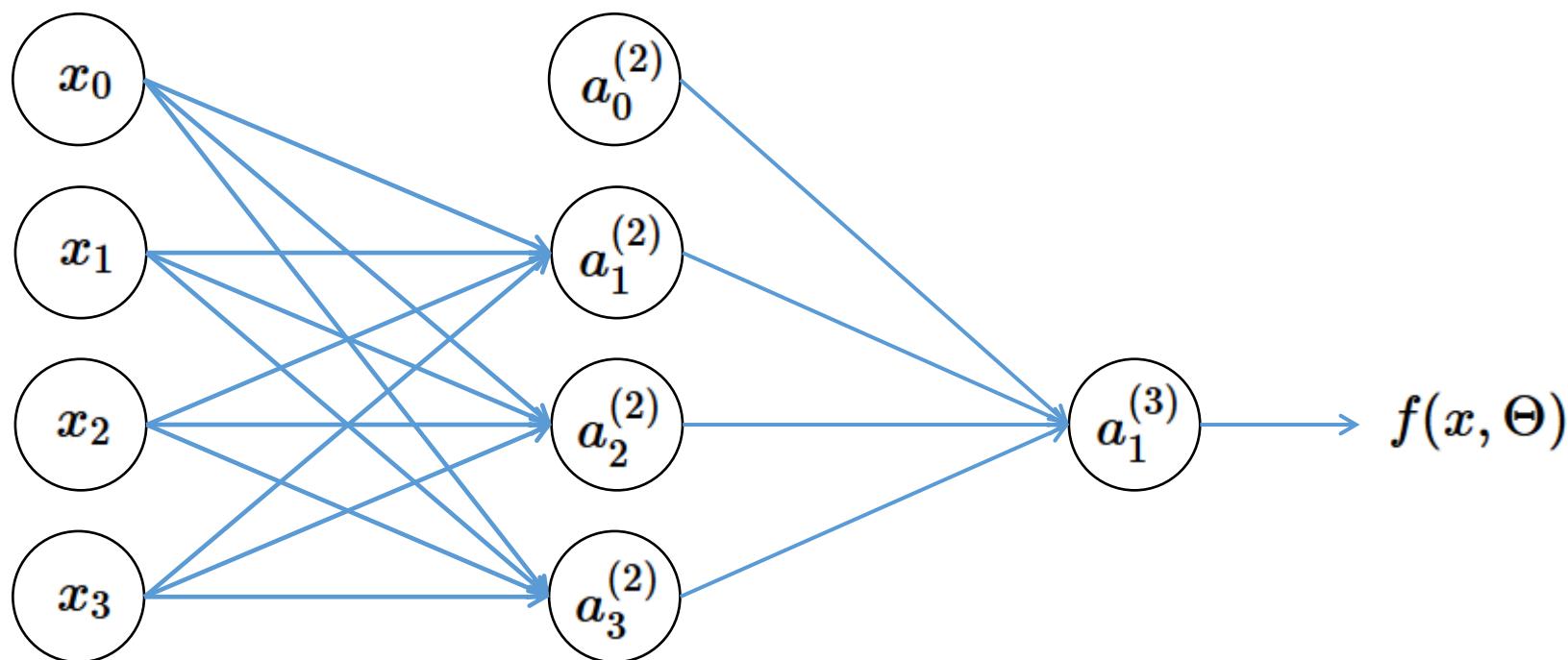
# Neural networks: model

$$f(x, \Theta) = g(\Theta_{10}^{(2)} a_0 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

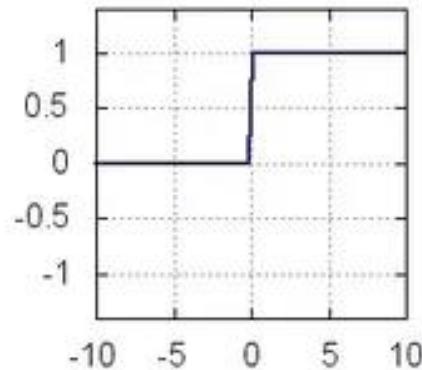
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

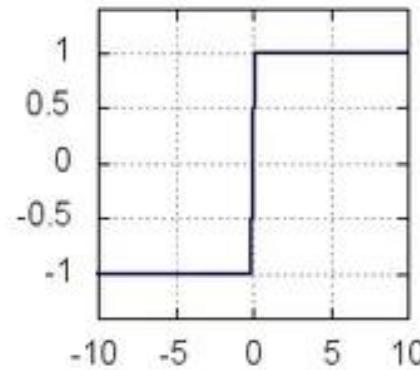


# Neural networks: activation functions

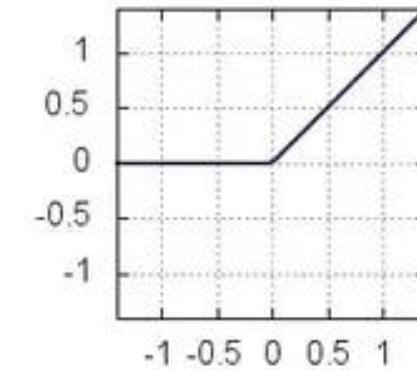
0/1 step



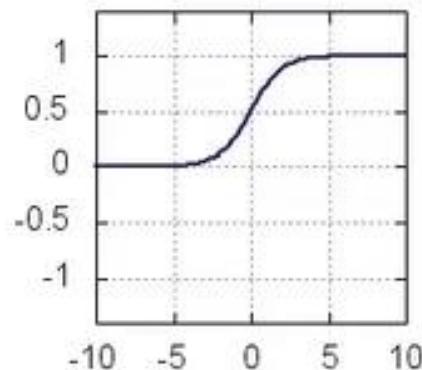
-1/+1 step



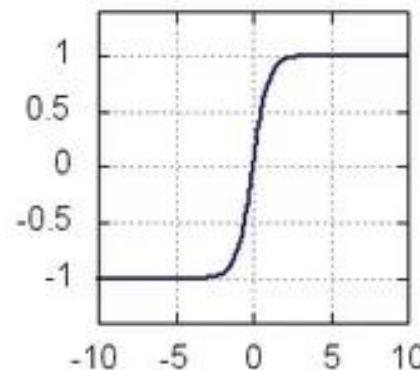
relu:  $\max(0, x)$



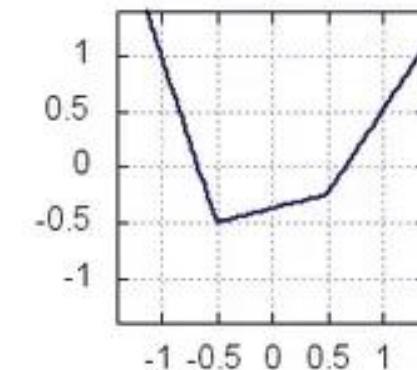
sigmoid:  $1/(1+e^{-x})$

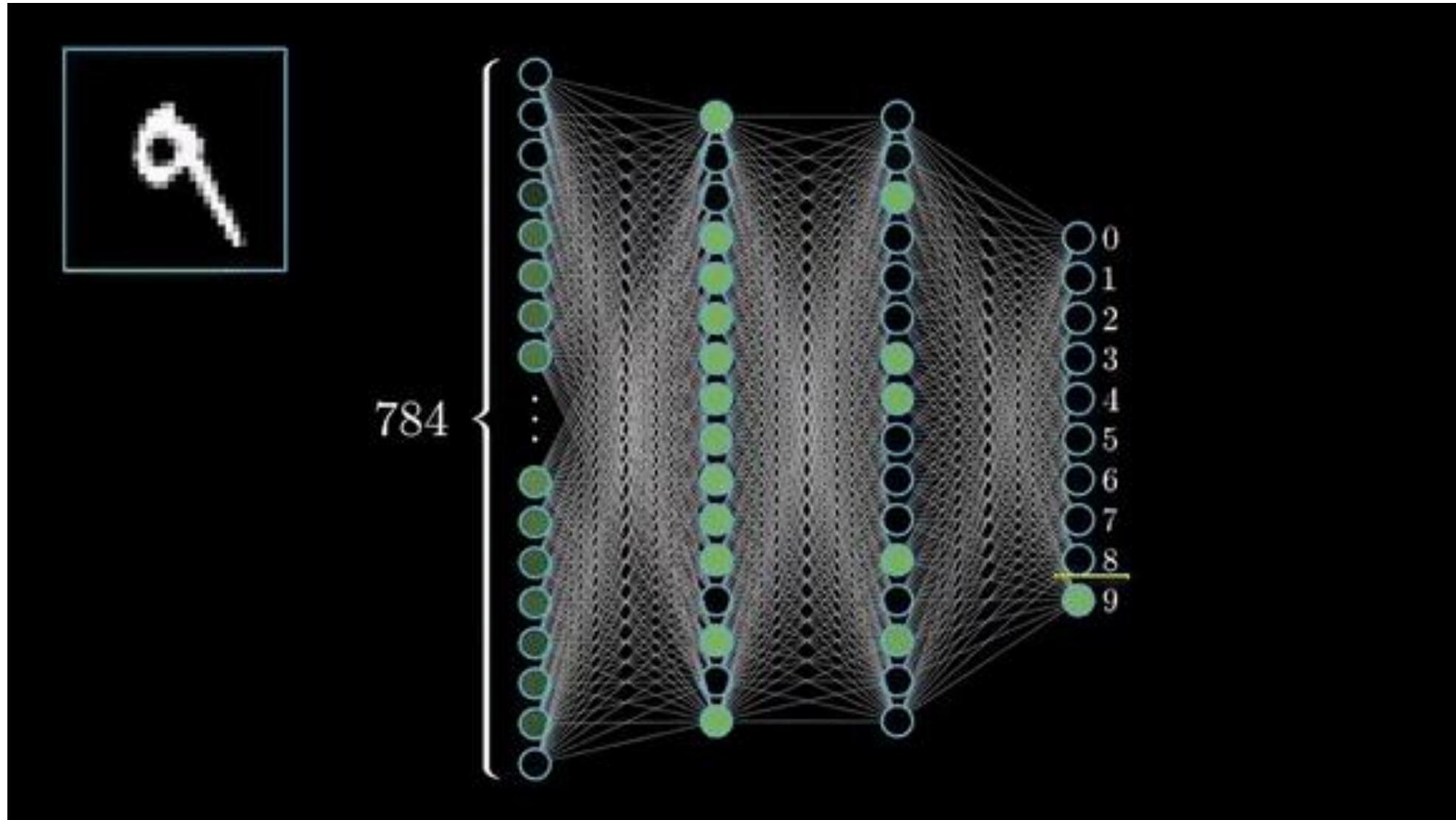


tanh:  $(e^x - e^{-x}) / (e^x + e^{-x})$



maxout





# Deep convolutional neural networks

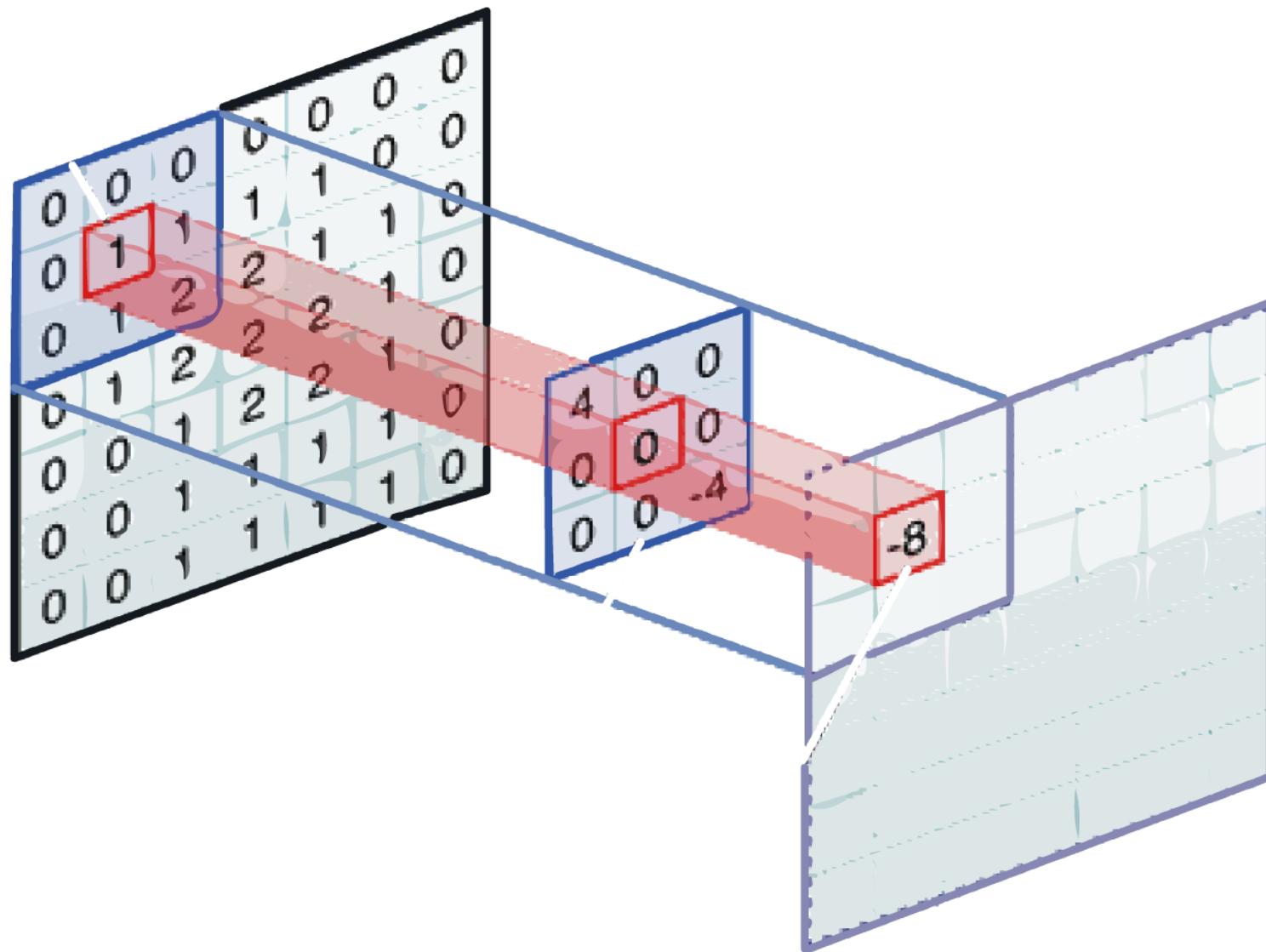


What We See

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08  
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00  
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65  
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91  
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80  
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50  
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70  
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21  
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72  
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95  
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92  
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57  
86 56 00 48 35 71 89 07 05 44 44 37 41 60 21 58 51 54 17 58  
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40  
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66  
88 36 68 87 57 62 20 72 03 46 33 67 44 55 12 32 63 93 53 69  
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36  
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16  
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54  
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

What Computers See

# Filters

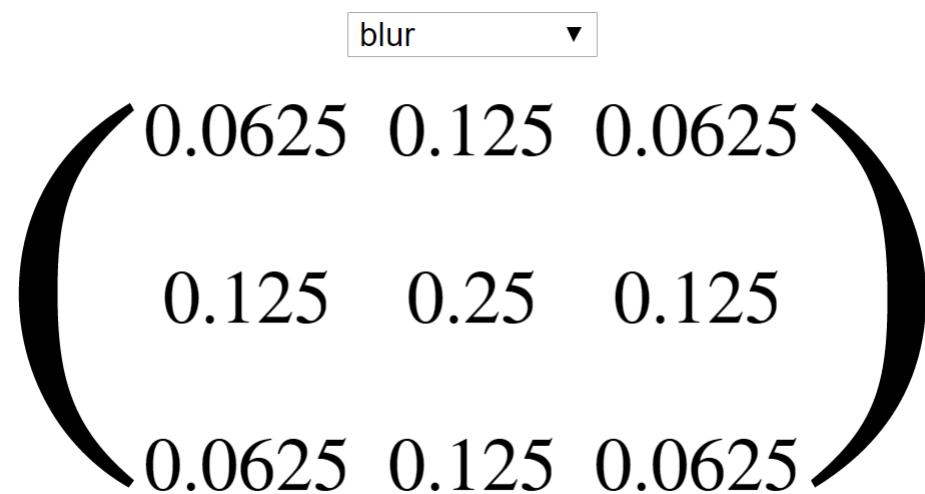


## Filters

206	205	247	248	244	253	247	245	136	151	255	256	255	255	255	255	254	254	255	255	255	255	255	255	254	255	247						
244	161	137	244	254	255	254	255	118	103	209	228	155	153	238	193	74	52	88	173	255	254	254	255	255	254	254	253	244				
192	154	75	200	249	255	255	255	119	98	84	81	35	44	89	53	44	45	43	54	140	213	253	255	255	255	245	187	188				
93	109	98	143	223	255	255	252	117	75	41	35	31	24	29	38	45	44	44	48	81	118	148	234	252	254	255	248	255	254			
87	89	107	198	238	255	255	255	104	25	34	35	29	20	25	34	32	30	32	34	53	85	100	142	231	242	247	249	255	255	255		
55	51	45	134	218	251	255	232	51	12	28	33	24	24	48	75	82	78	71	88	58	53	87	90	138	228	208	158	253	246	249	255	
79	58	58	75	234	255	255	255	118	11	27	74	99	91	108	140	162	173	173	173	172	158	157	92	48	78	187	217	208	254	222	233	255
38	43	47	52	147	255	229	56	41	81	129	145	180	189	169	172	178	179	179	179	177	177	177	172	119	31	82	209	238	255	244	249	255
40	40	33	38	90	245	171	32	85	110	139	145	151	182	171	174	178	179	182	184	187	183	173	182	71	45	167	255	254	255	254	255	
37	44	44	31	89	250	158	38	70	129	143	142	153	182	171	175	177	178	182	191	194	188	180	170	120	51	137	255	254	250	254	255	
34	45	51	64	116	237	181	53	118	138	140	143	154	164	176	178	174	177	183	188	185	185	183	178	140	88	141	254	252	225	249	255	
34	36	52	74	71	188	156	83	131	134	144	155	180	181	173	179	178	179	189	193	190	185	187	182	158	93	148	250	254	214	247	255	
32	38	52	54	159	250	128	57	129	138	138	140	151	158	166	168	171	178	180	187	188	185	185	183	180	102	138	242	255	255	254	254	
36	32	72	72	129	212	228	115	85	121	104	102	104	94	103	134	158	170	162	125	108	121	143	155	190	191	104	134	230	253	253	255	251
61	82	118	107	179	247	124	80	101	90	111	119	103	81	94	147	191	178	128	98	123	153	147	181	200	92	100	222	207	167	227	215	
144	178	187	231	210	232	170	87	115	88	78	82	83	85	88	139	192	190	135	80	53	99	141	185	201	97	79	192	245	235	248	249	
127	145	149	195	195	204	213	197	95	133	122	117	133	128	108	110	139	191	197	187	129	127	148	147	171	188	110	121	228	233	180	215	212
87	112	100	79	85	82	85	75	142	148	151	153	138	125	120	149	191	190	193	175	174	193	198	190	208	127	183	239	219	149	198	195	
83	83	109	134	129	108	39	78	132	142	155	159	139	111	124	164	195	200	188	192	191	195	200	202	200	143	217	253	249	242	238	234	
89	78	78	113	97	74	43	198	127	140	152	155	125	97	112	150	185	194	174	183	198	202	208	209	168	247	254	255	254	254	254		
72	44	83	59	48	52	49	74	127	137	148	149	132	103	78	90	134	141	168	165	199	207	204	203	218	193	238	244	251	242	238	243	
55	20	89	73	59	80	46	74	117	127	144	161	148	124	105	120	158	187	193	162	189	208	201	205	214	194	174	185	197	188	183	193	
65	49	77	89	50	68	43	81	109	127	141	147	113	100	121	145	148	169	181	178	181	201	201	205	202	174	188	169	178	183	188	184	
82	78	92	79	54	58	37	47	90	121	132	116	89	78	111	148	183	149	122	124	180	197	197	198	178	149	148	152	155	157	159	168	
104	107	122	122	123	105	79	27	33	86	111	122	120	114	114	147	175	190	198	183	101	170	200	187	185	156	148	145	139	137	141	140	145
117	124	127	133	135	105	21	28	37	88	115	121	128	128	141	142	168	202	212	153	164	188	180	188	154	148	144	149	151	151	147	144	
119	118	118	125	128	111	21	29	28	58	100	118	131	140	151	159	188	201	205	192	180	168	149	188	112	144	147	143	140	141	144	148	
117	119	125	130	139	108	18	29	44	58	70	102	133	147	168	197	212	215	210	195	177	152	133	195	57	59	128	151	145	143	142	141	
115	123	128	128	134	145	102	27	54	52	38	45	89	105	135	175	189	193	218	208	168	139	111	164	203	74	5	121	151	142	142	143	146
101	103	103	123	121	132	105	44	40	31	35	57	44	58	101	147	144	138	183	145	94	93	145	198	187	84	48	165	189	142	144	142	145
98	97	97	98	104	78	34	33	30	48	41	49	51	58	74	53	55	88	83	89	150	188	209	156	62	108	140	149	125	133	131	131	
102	102	97	88	73	35	30	23	42	50	65	41	90	80	59	51	57	82	123	157	187	205	169	82	98	151	105	101	154	135	130	129	



# Filters



# Filters

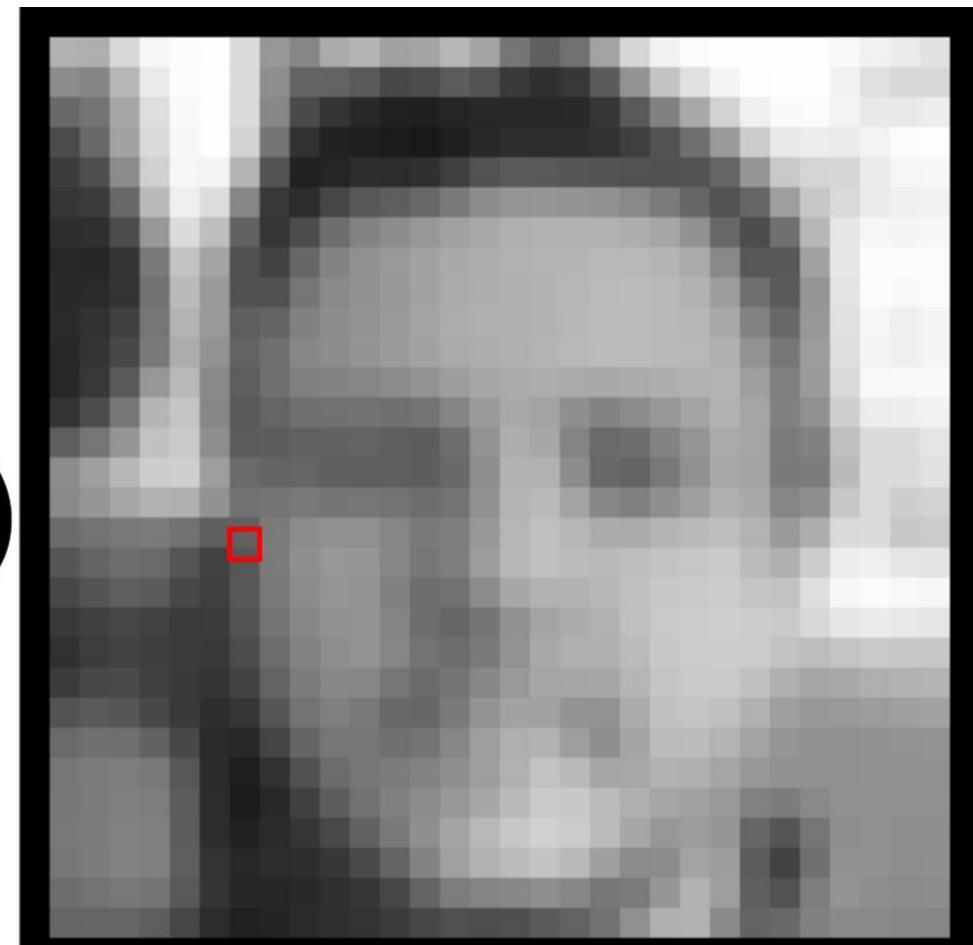


input image

$$\left( \begin{array}{c} 57 + 129 + 138 \\ \times 0.0625 \times 0.125 \times 0.0625 \\ \\ + 65 + 121 + 104 \\ \times 0.125 \times 0.25 \times 0.125 \\ \\ + 60 + 101 + 90 \\ \times 0.0625 \times 0.125 \times 0.0625 \end{array} \right) = 102$$

kernel:

blur ▾

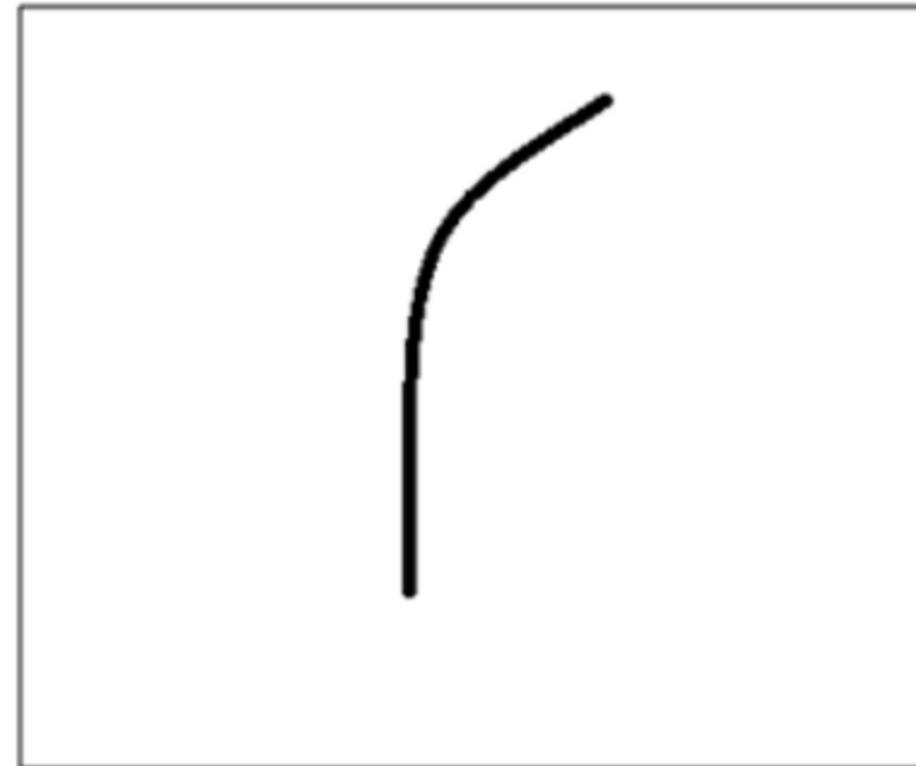


output image

# Filters

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

# Filters



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

\*

0	0	0	0	0	0	30	0
0	0	0	0	0	30	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

# Filters



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

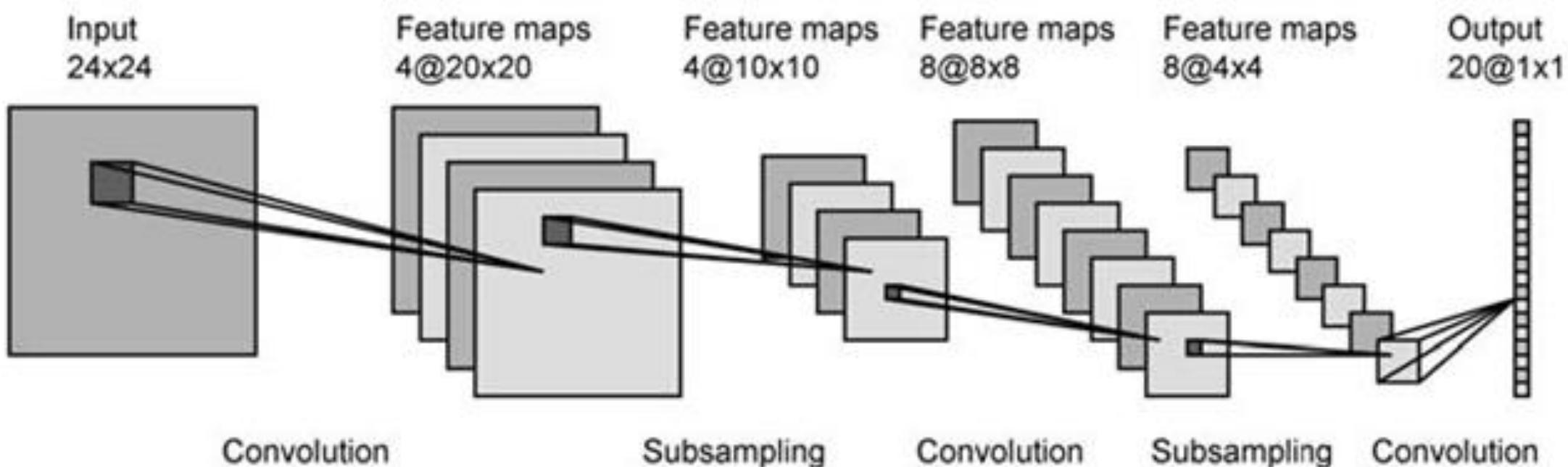
Pixel representation of receptive field

\*

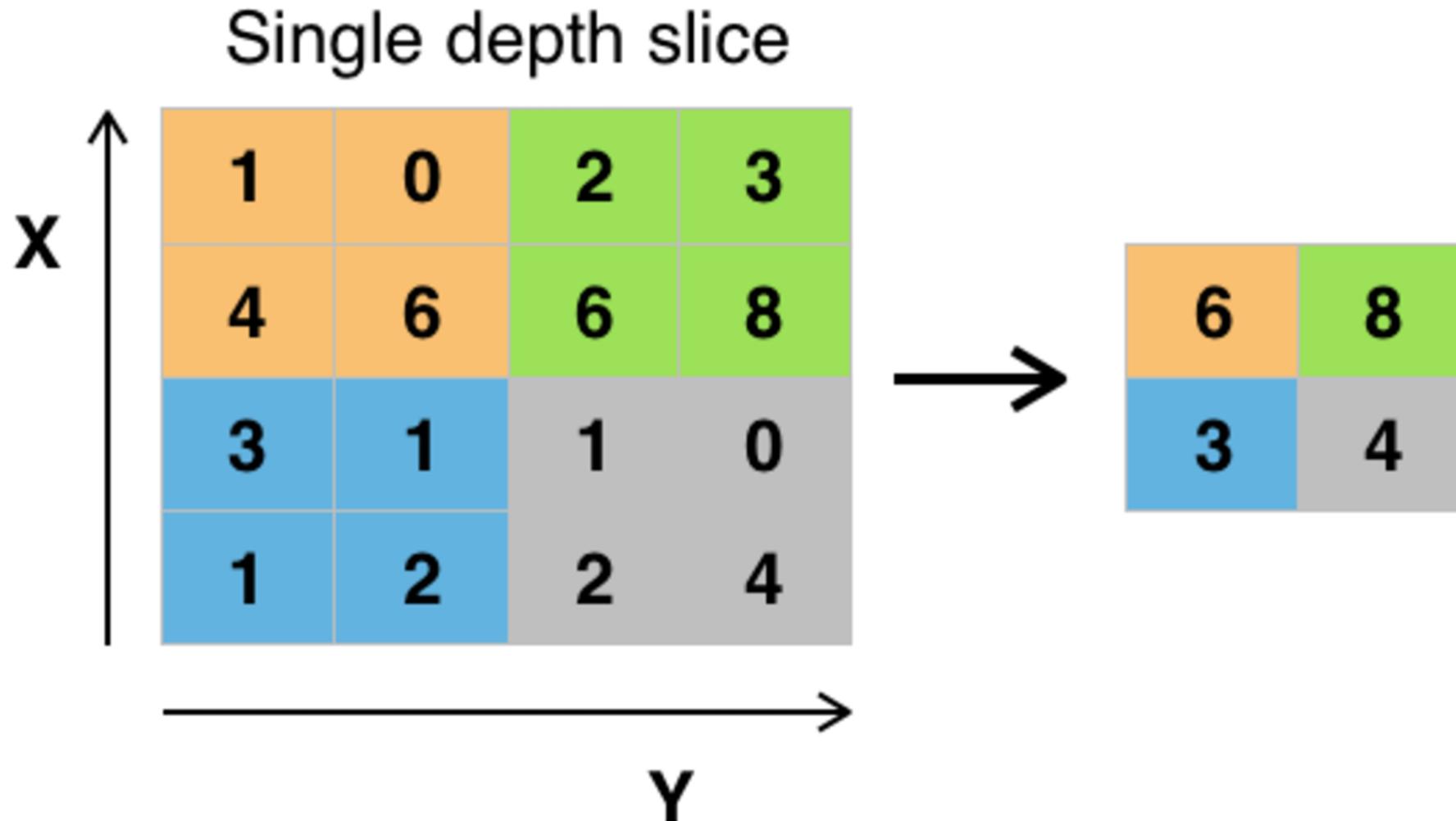
0	0	0	0	0	0	30	0
0	0	0	0	0	30	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

# Deep convolutional neural networks

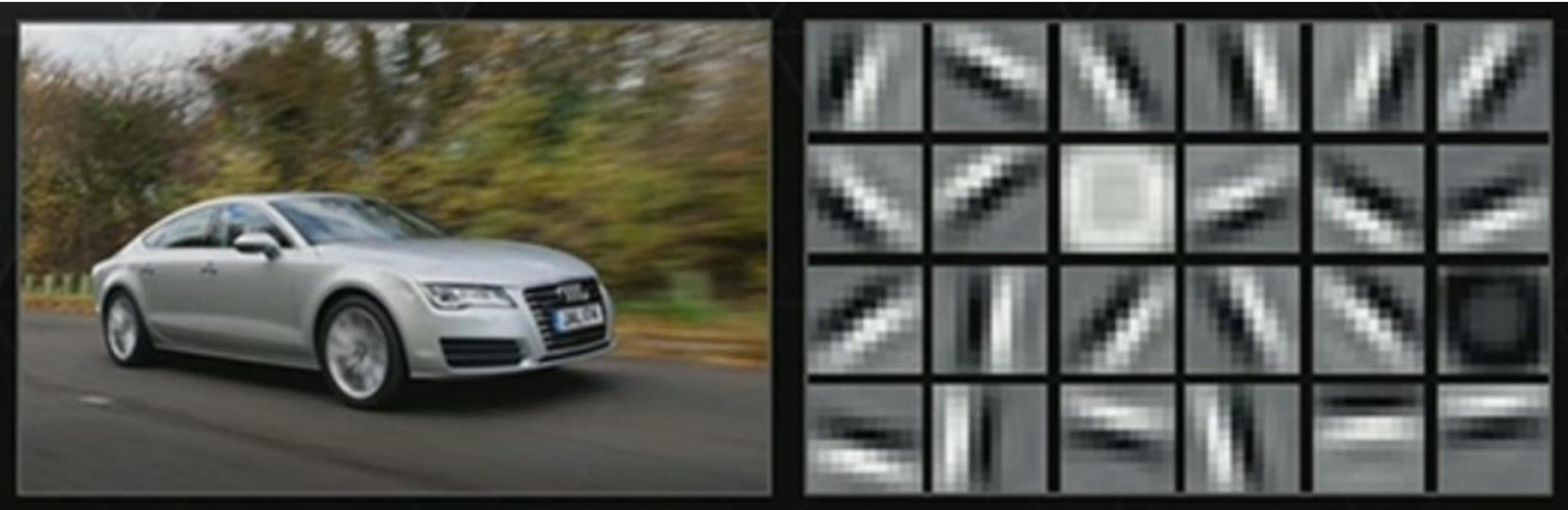


## Max pooling



Example of Maxpool with a 2x2 filter and a stride of 2

# CNN: internal representations



# CNN: internal representations



Faces



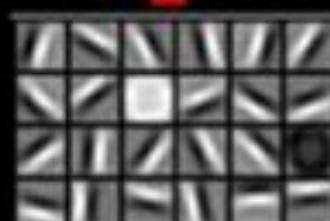
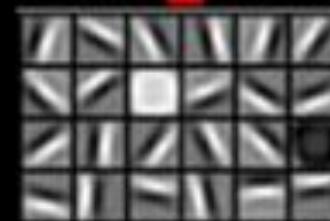
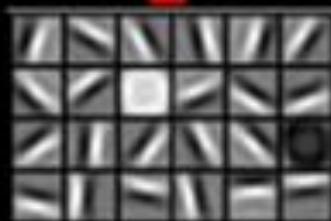
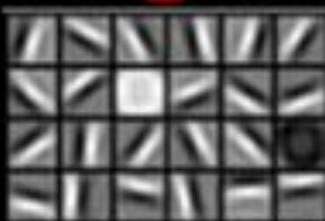
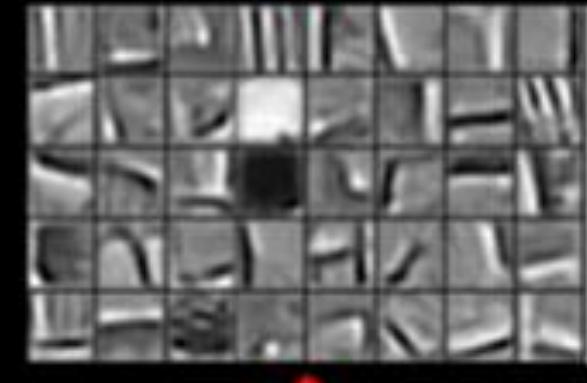
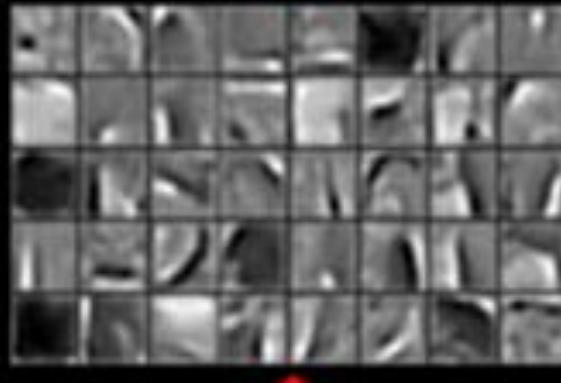
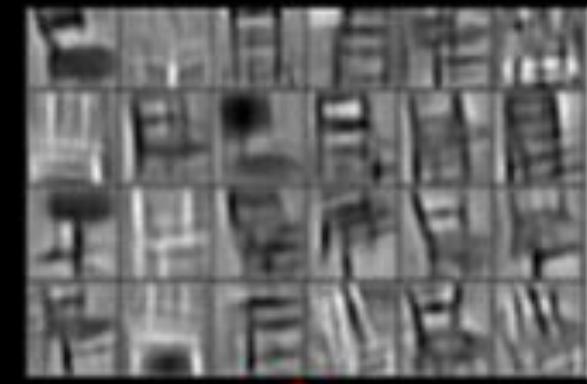
Cars



Elephants



Chairs







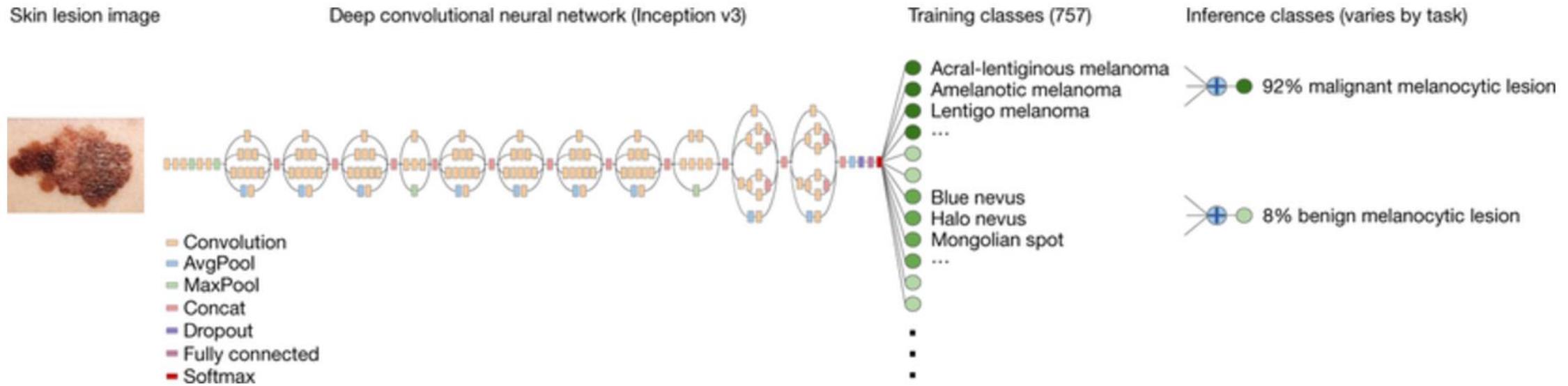
# Deep learning algorithm diagnoses skin cancer as well as seasoned dermatologists

By Jessica Hall on January 25, 2017 at 1:25 pm [16 Comments](#)

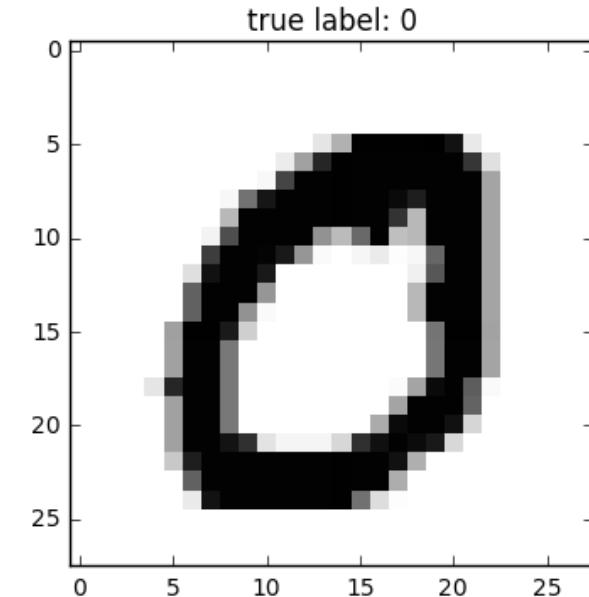
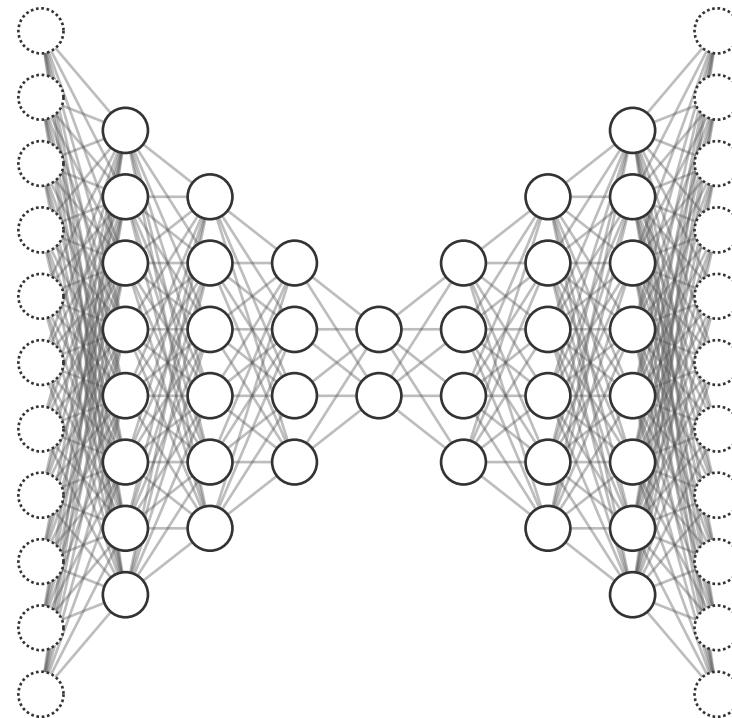
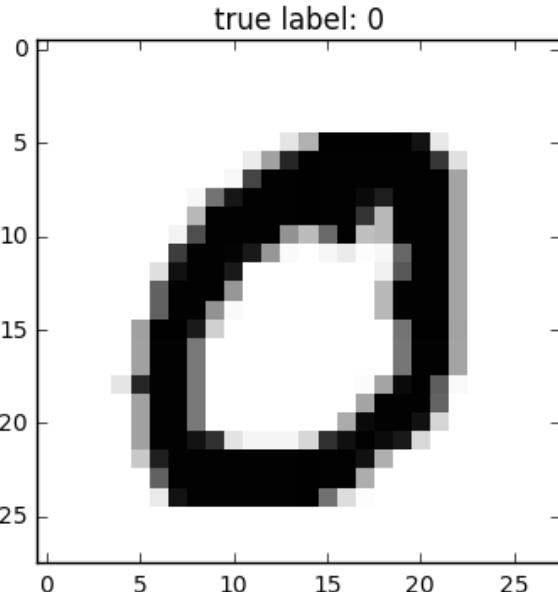
**422**  
shares



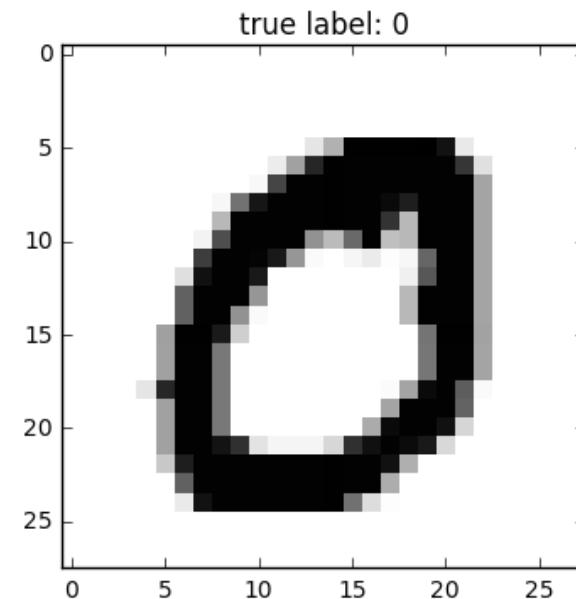
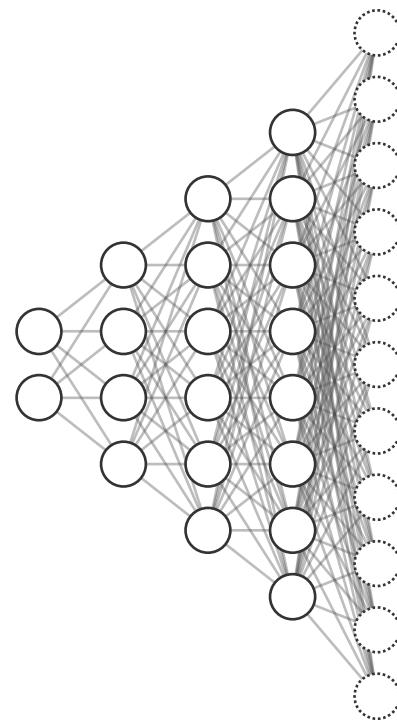
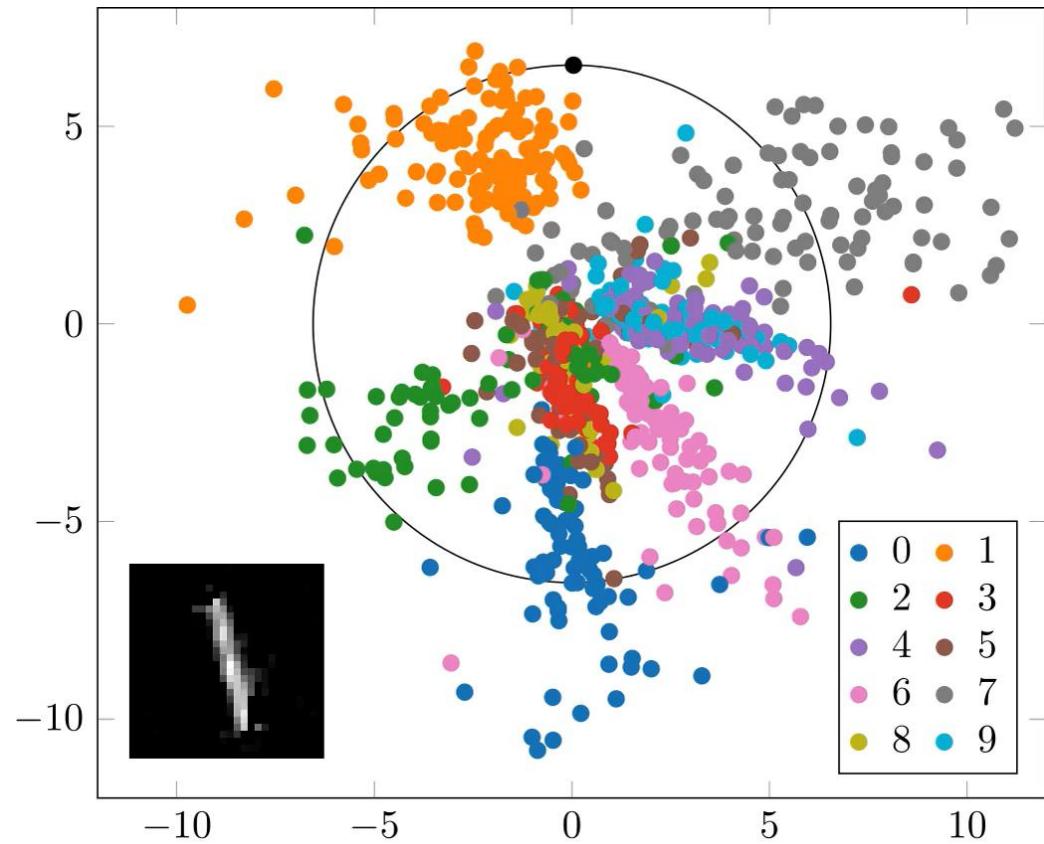
# CNN: fine-tuning



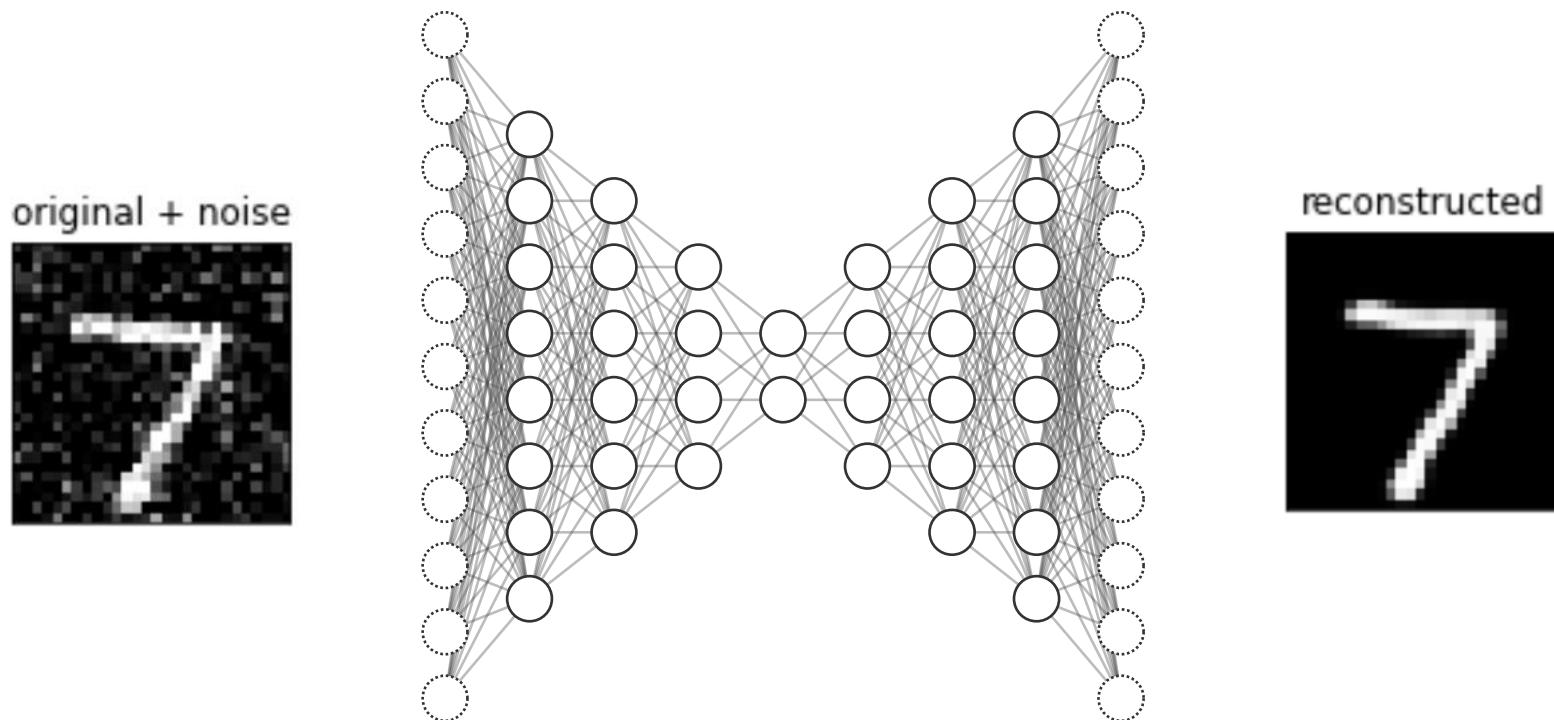
# Embeddings: auto-encoder



# Embeddings: auto-encoder



# Auto-encoder: de-noising



# Caught the ML virus?

- My course:  
<https://www.youtube.com/watch?v=xK9wHHQfGYA&list=PLXxp6nsBenSWGYnoZjCGCQ0tGdg7bI3na>
- Andrew Ng's course: <https://www.coursera.org/learn/machine-learning>
- Kaggle tutorials: <https://www.kaggle.com/learn>

