SCIENCE MEETS LIFE

A tour of machine learning

Sven Degroeve

VIB Center for Medical Biotechnology

UGent Department of Biomolecular Medicine

VIB-UGENT CENTER FOR MEDICAL BIOTECHNOLOGY

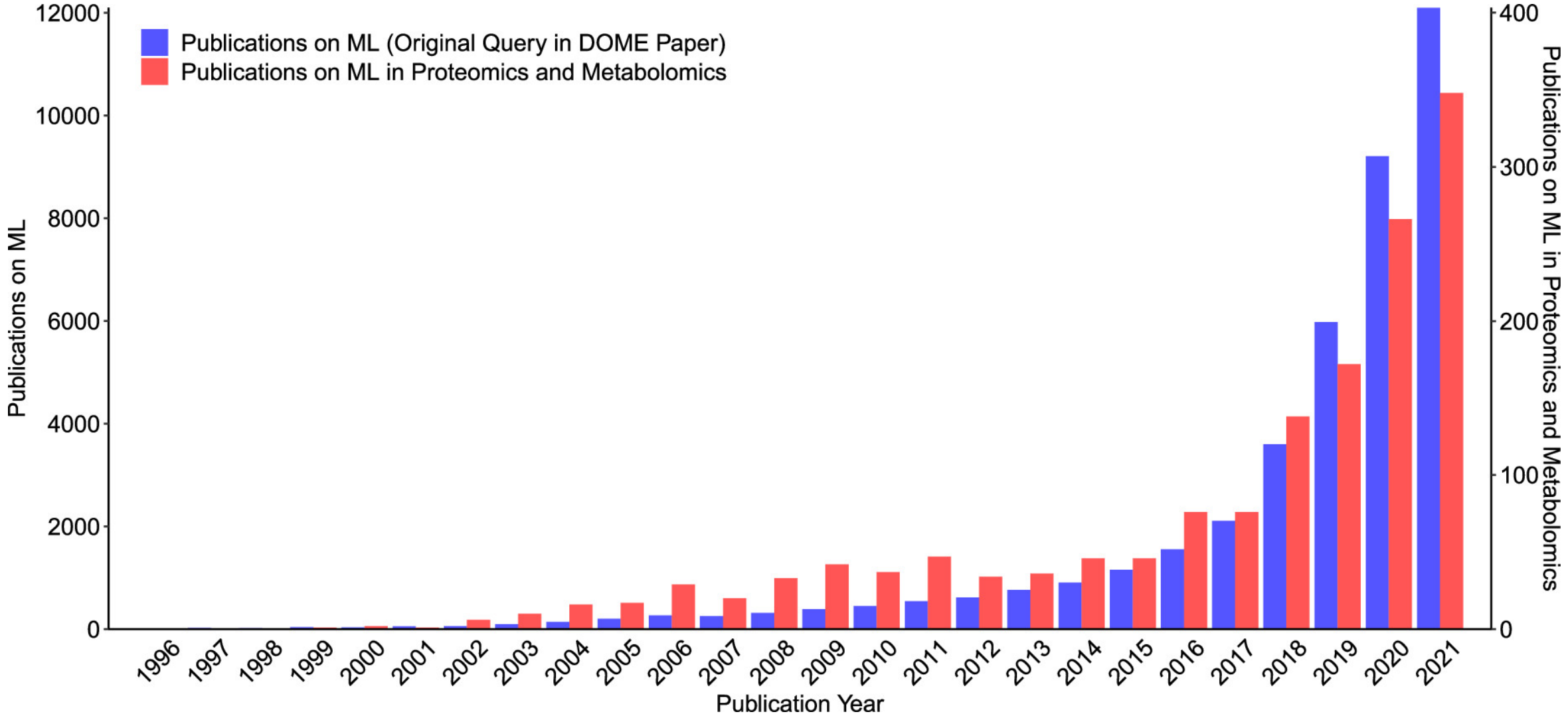
UNIVERSITEIT GENT

CompOmics

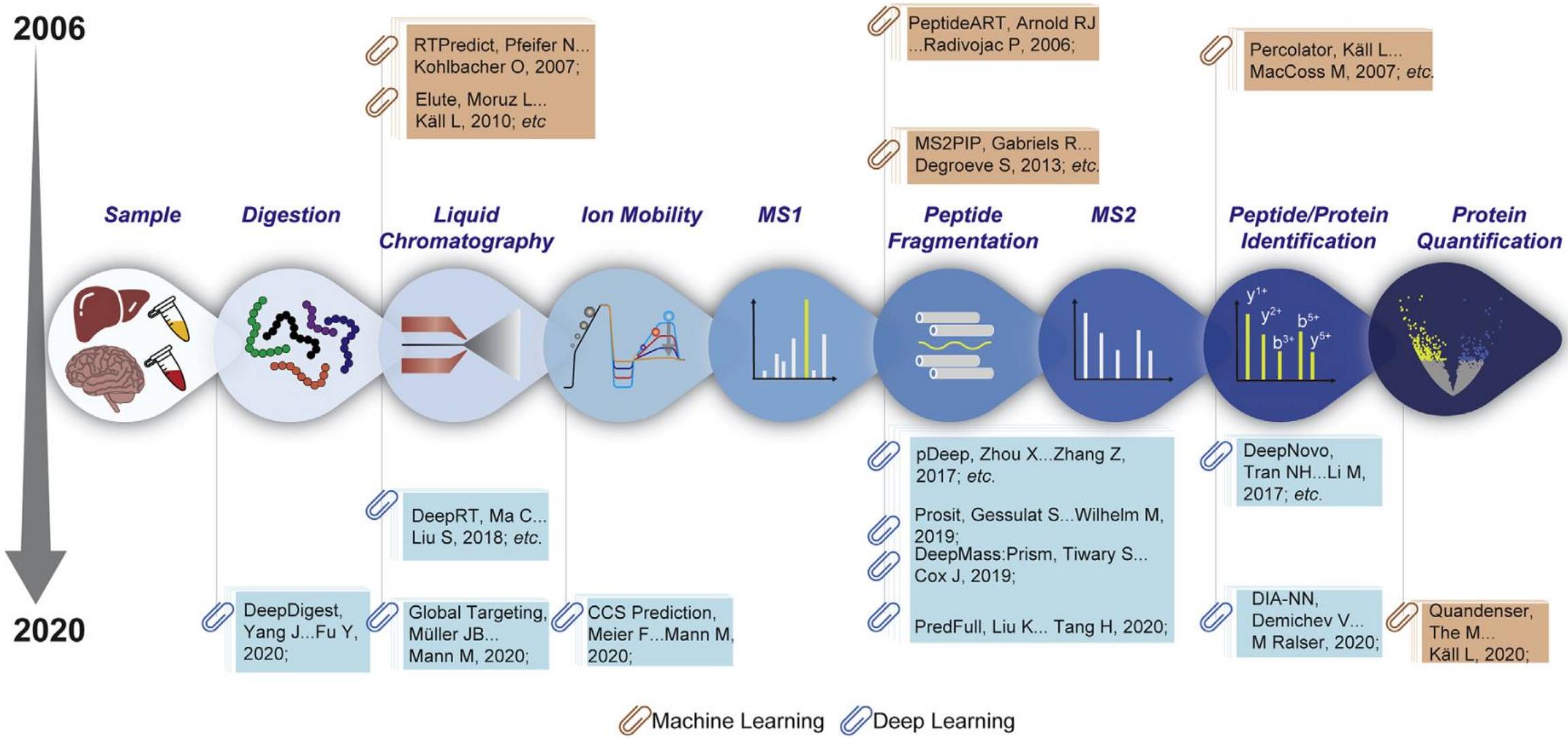
EuBIC 2022

- 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

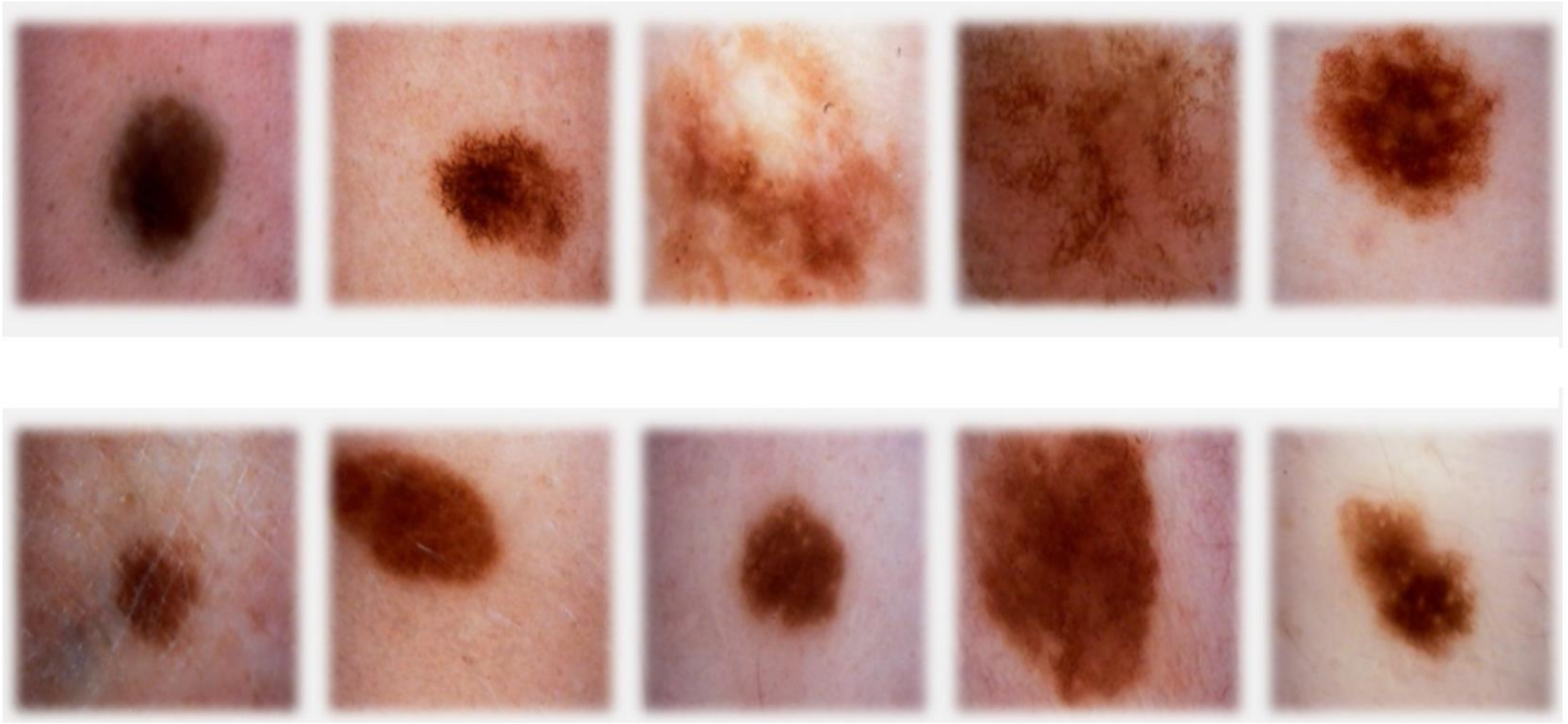


Palmblad, M. *et al.*, J. Proteome Res. (2022)



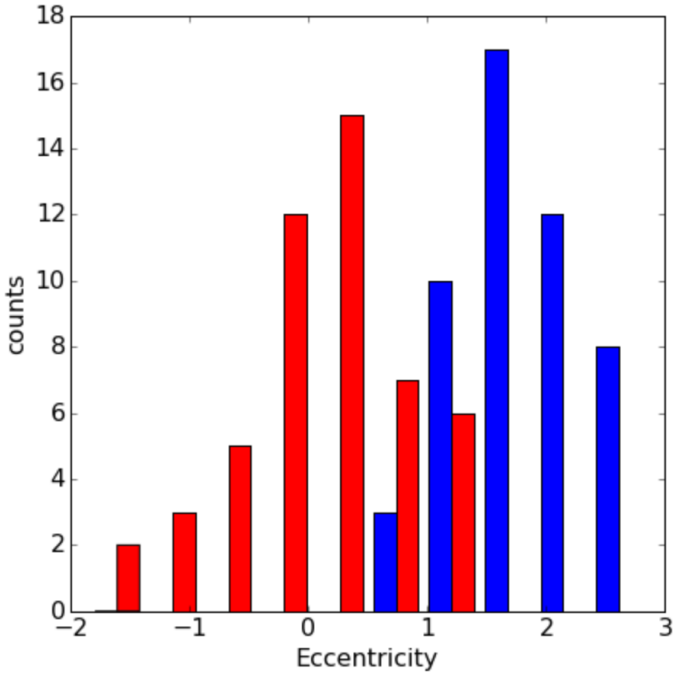
Mann, M. *et al.*, Cell Systems (2021)

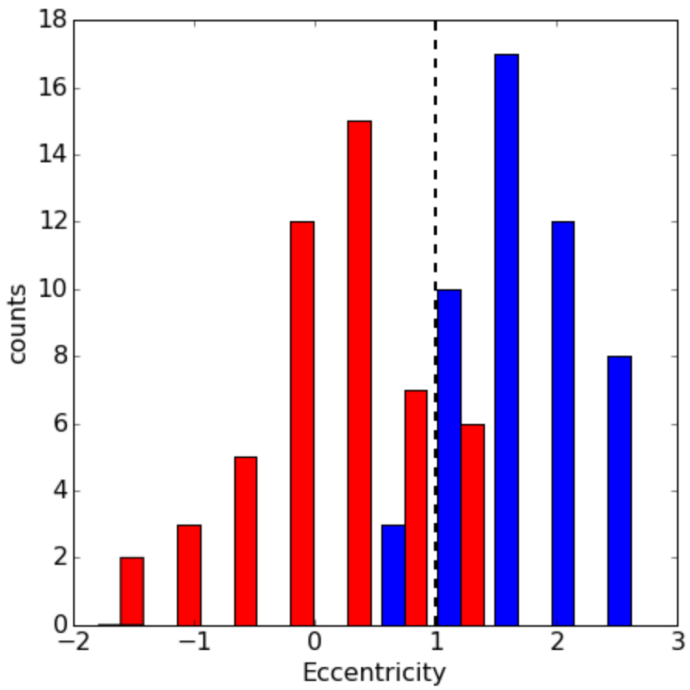
The dataset

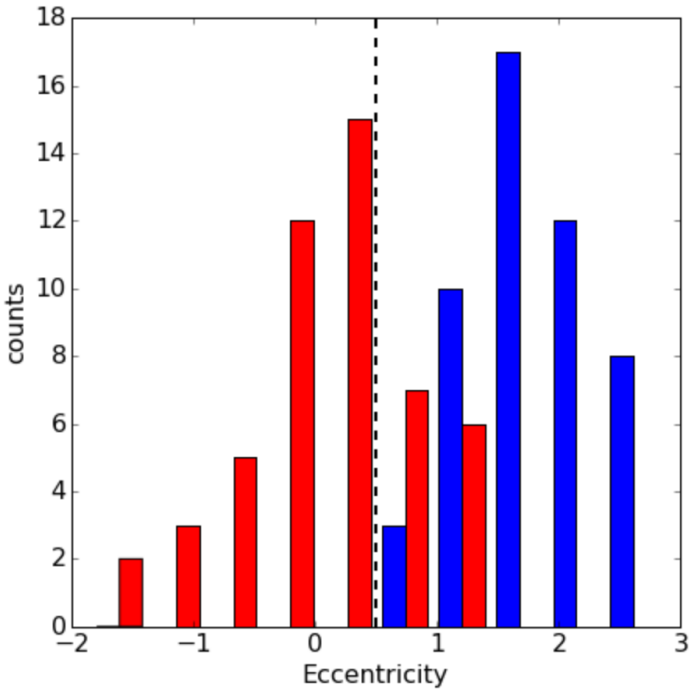


The features

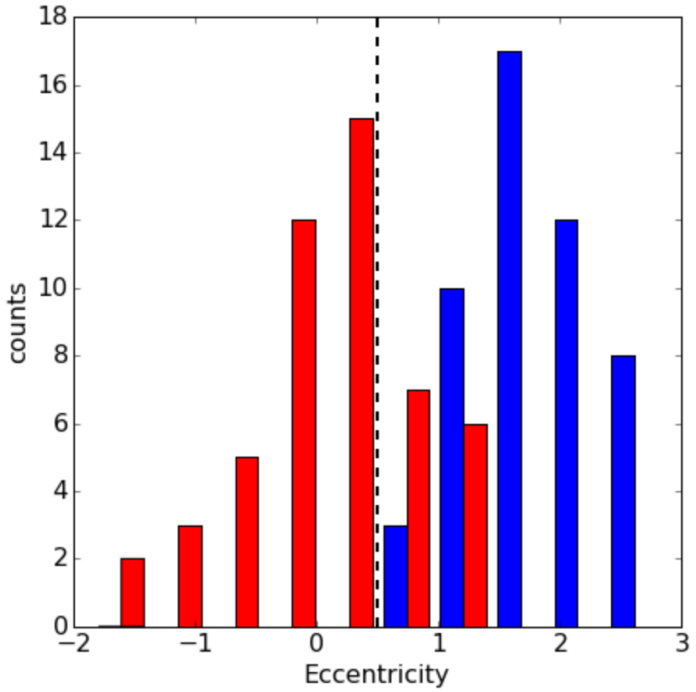
The model

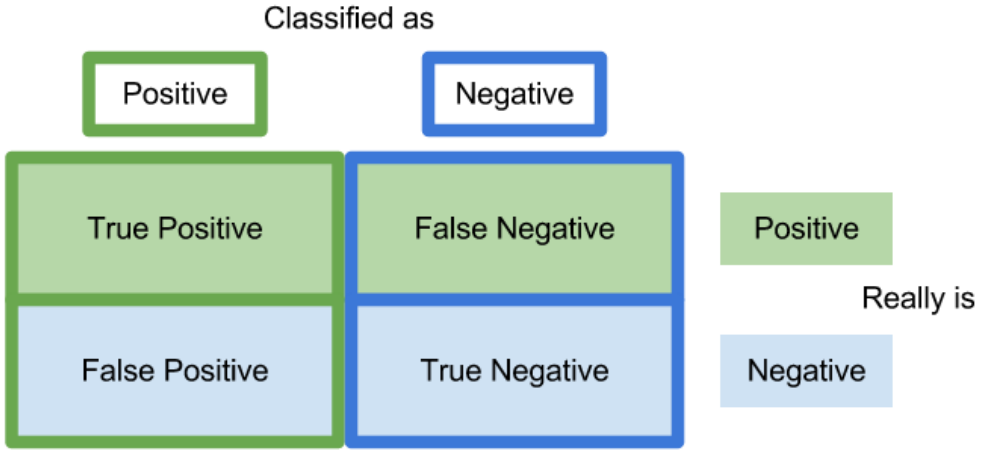

counts
Eccentricity
0
2
4
6
8
10
12
14
16
18
−2
−1
0
1
2
3


counts
Eccentricity
0
2
4
6
8
10
12
14
16
18
−2
−1
0
1
2
3

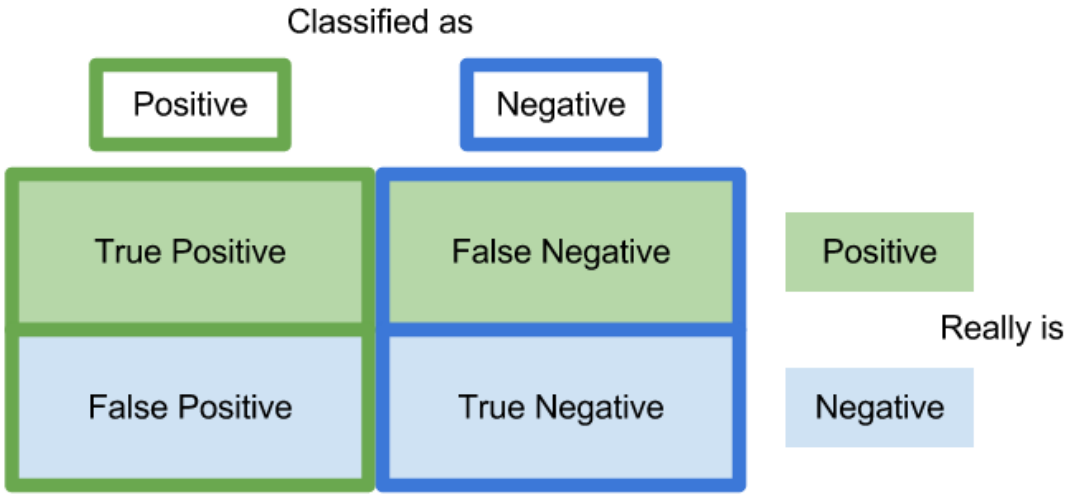

counts
Eccentricity
0
2
4
6
8
10
12
14
16
18
−2
−1
0
1
2
3

Model evaluation





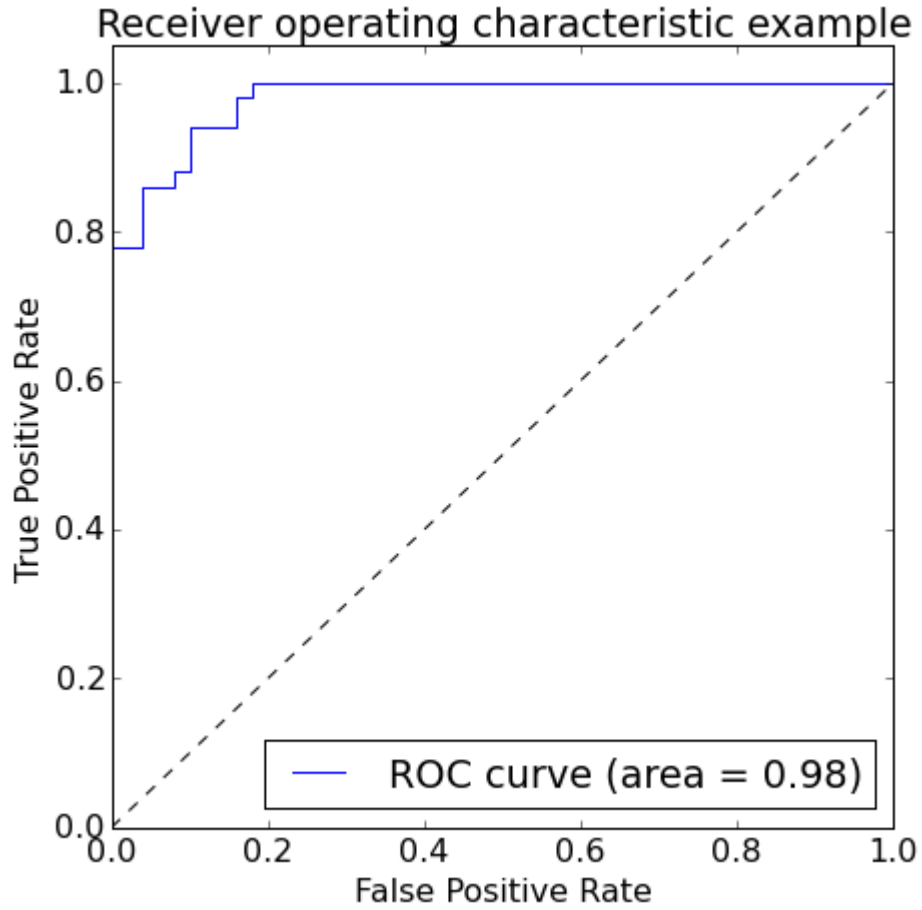
Model evaluation



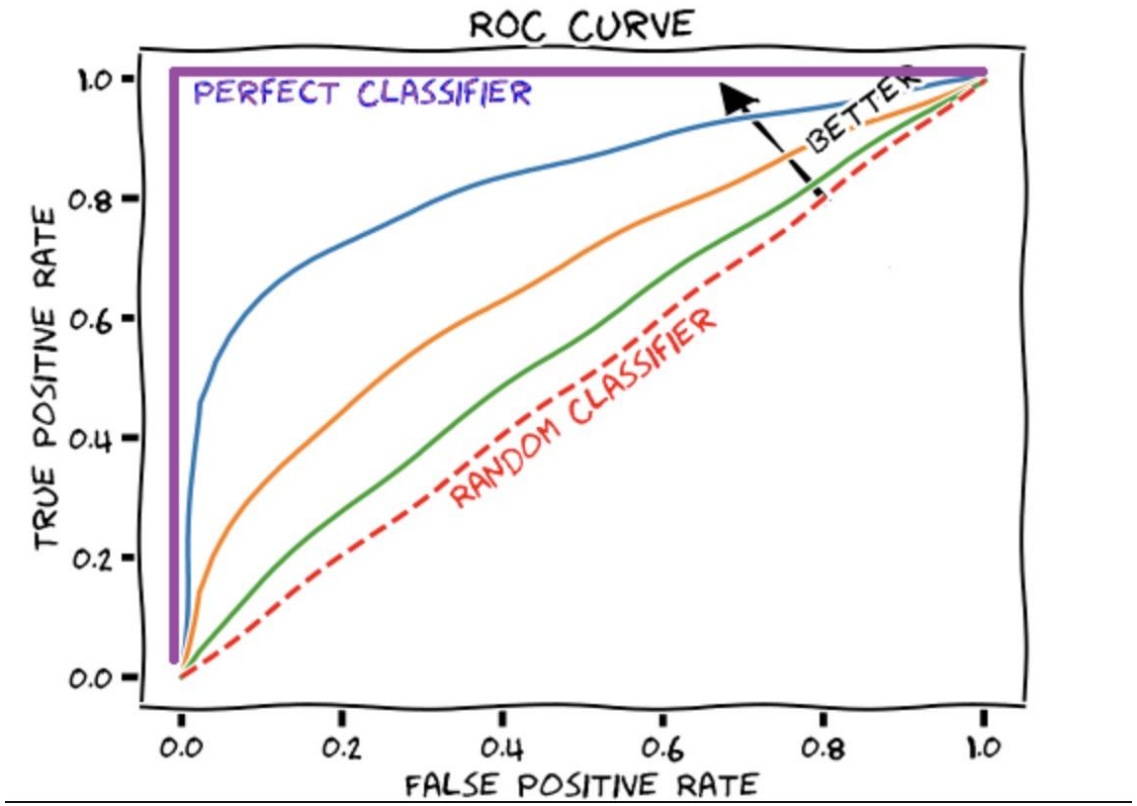
Recall Sensitivity True positive rate (TPR)	$\frac{TP}{FN+TP} = \frac{TP}{P}$
False positive rate (FPR) False alarm rate	$\frac{FP}{TN+FP} = \frac{FP}{N}$
Specificity True negative rate (TNR)	$\frac{TN}{TN+FP} = \frac{TN}{N} = 1 - FPR$
Precision	$\frac{TP}{TP+FP}$
False negative rate (FNR)	$\frac{FN}{FN+TP}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

Model evaluation

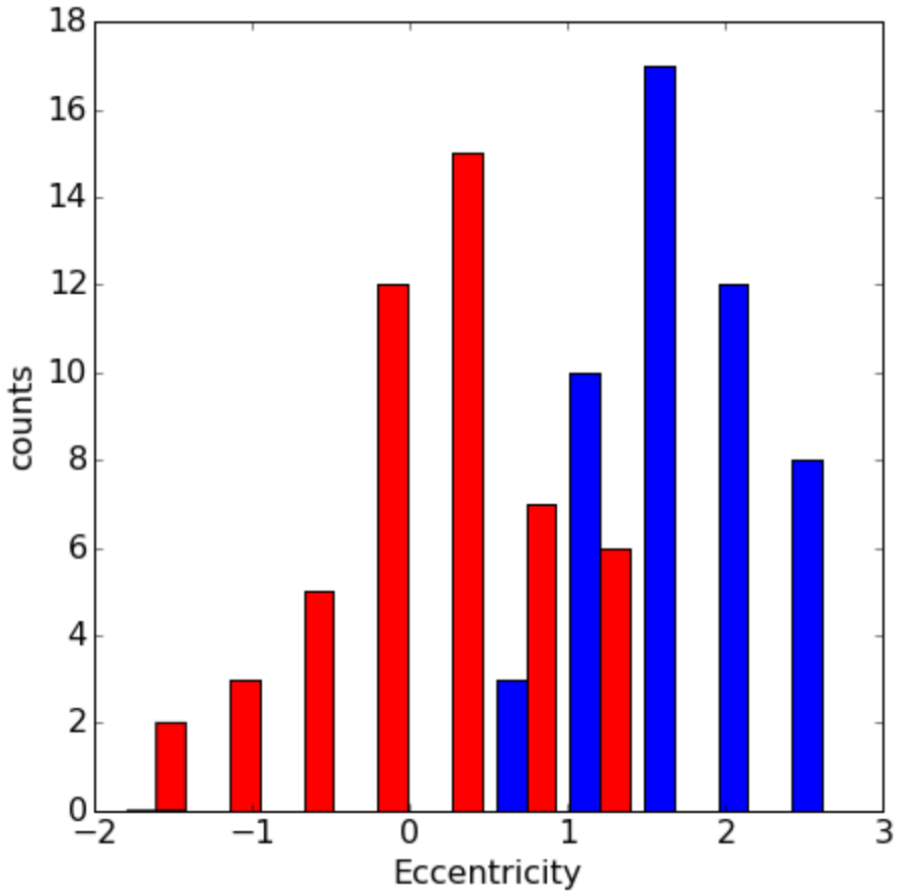
$$\mathrm{TPR} = \frac{TP}{TP + FN}$$

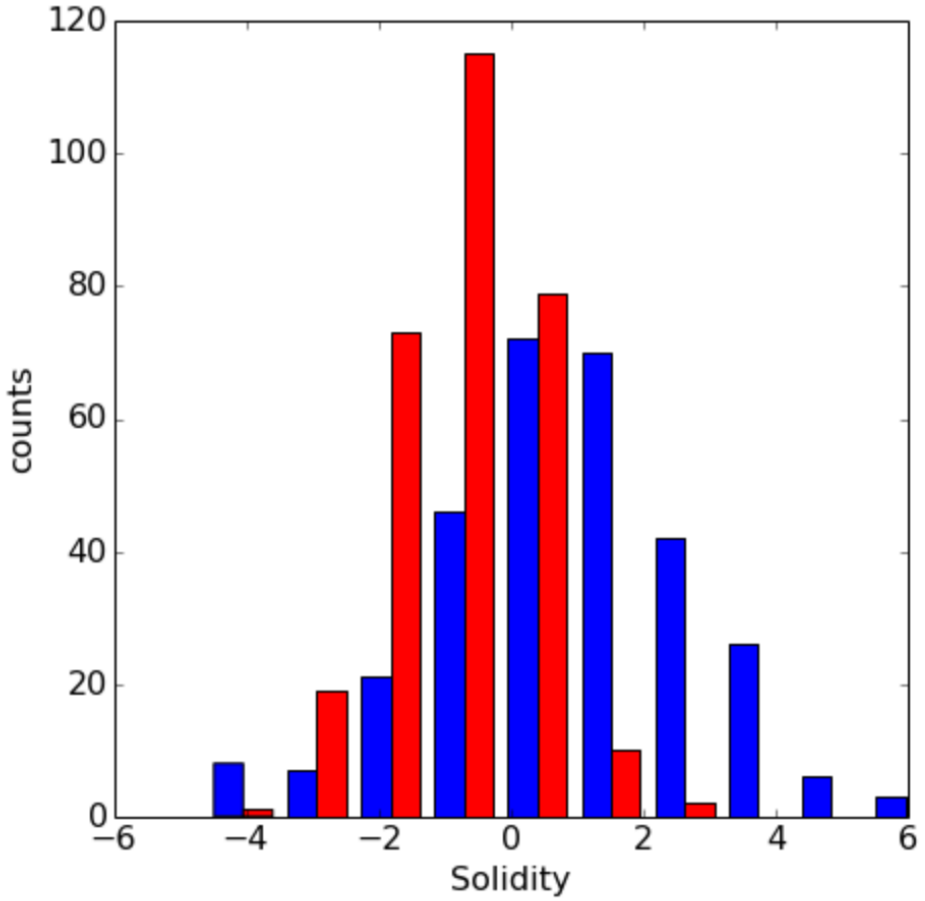


$$\mathrm{FPR} = \frac{FP}{FP + TN}$$

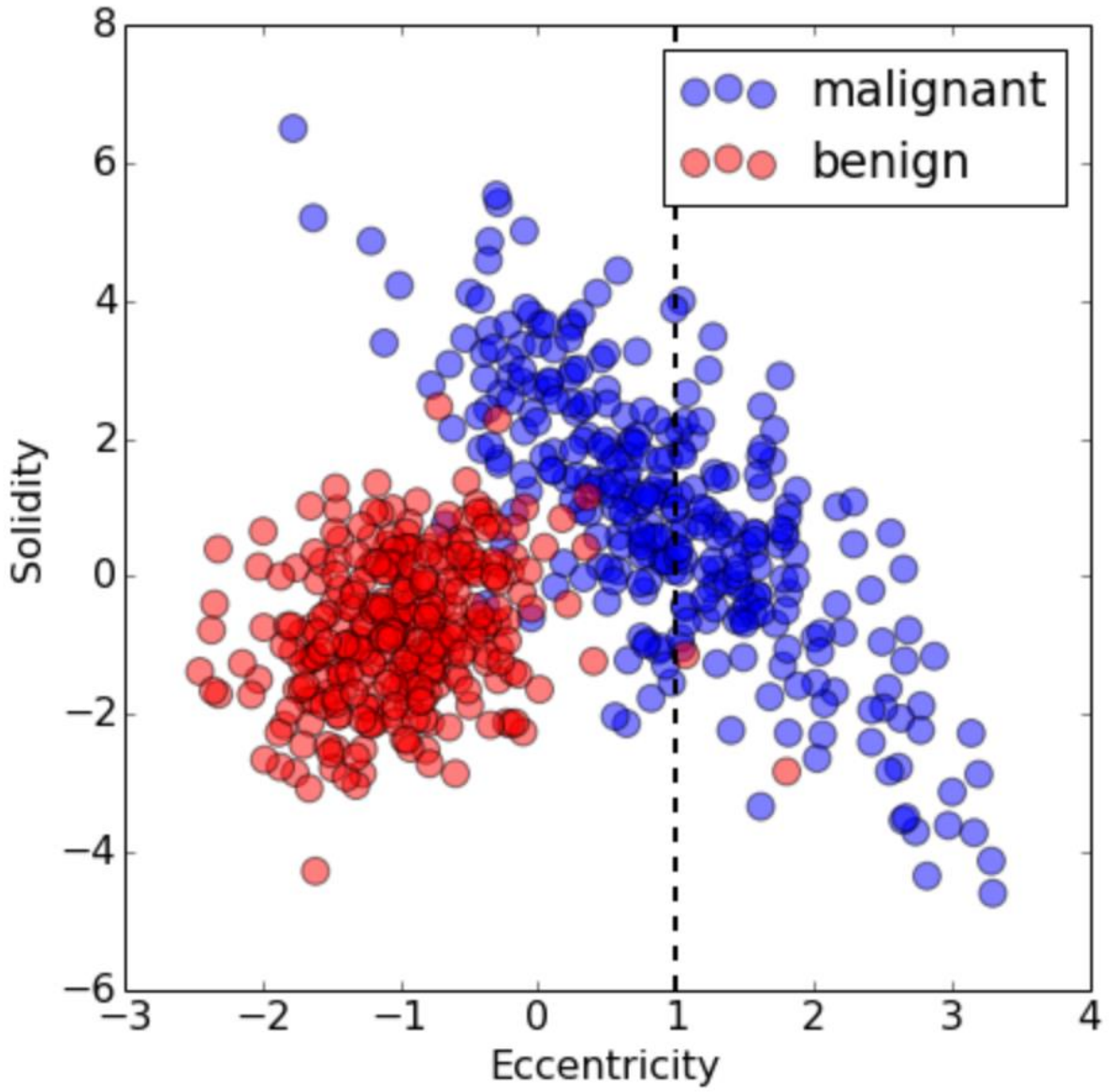


The features

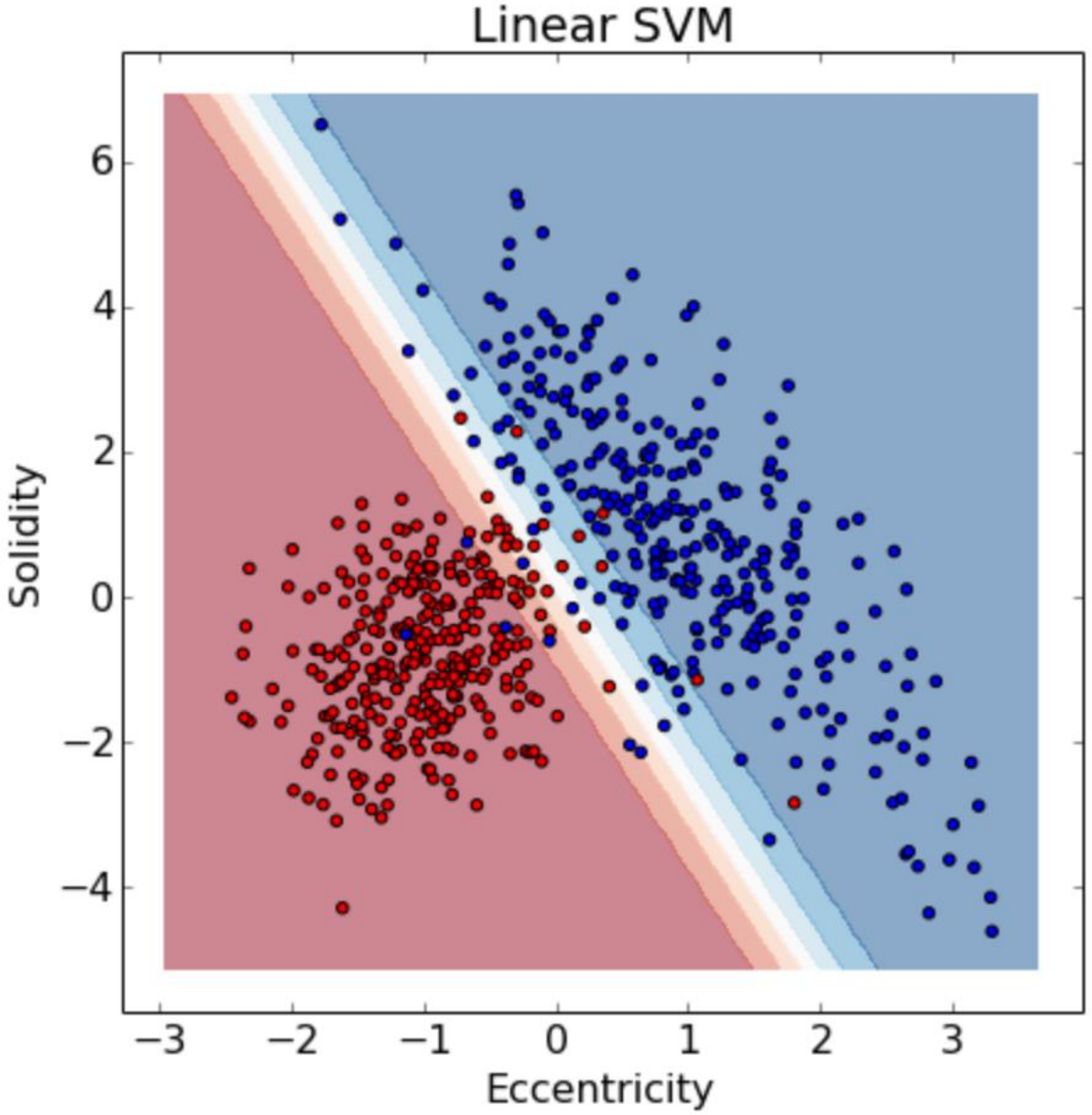




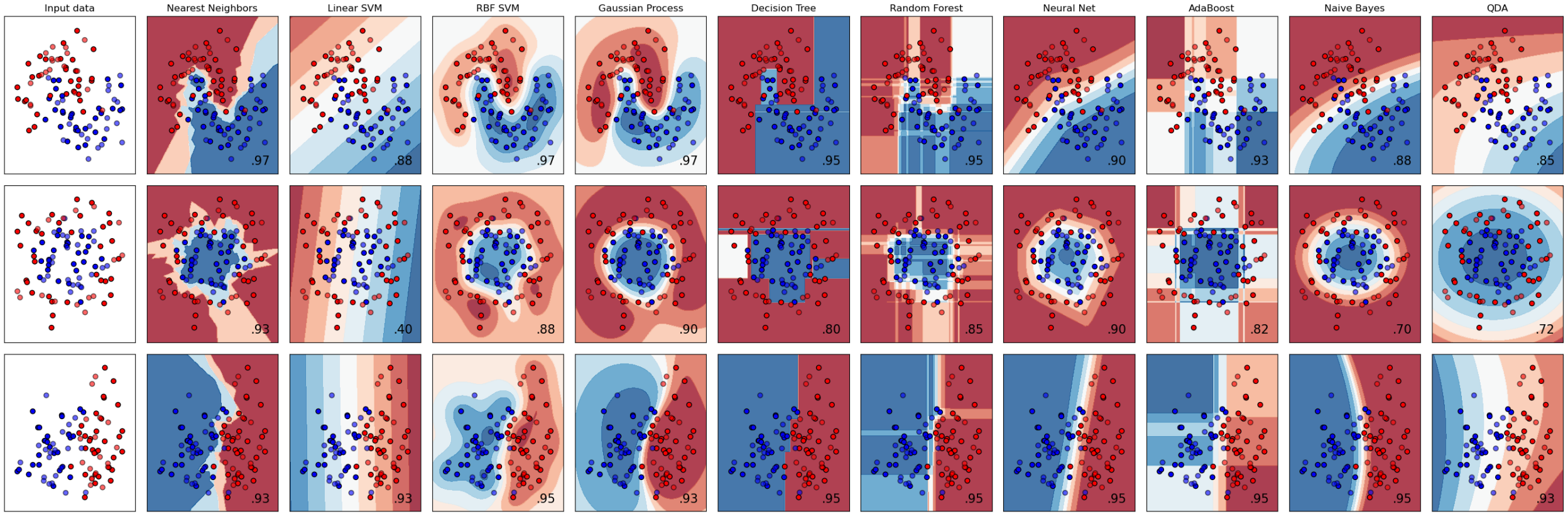
The model


malignant
benign
Solidity
Eccentricity
8
6
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3
4

The model



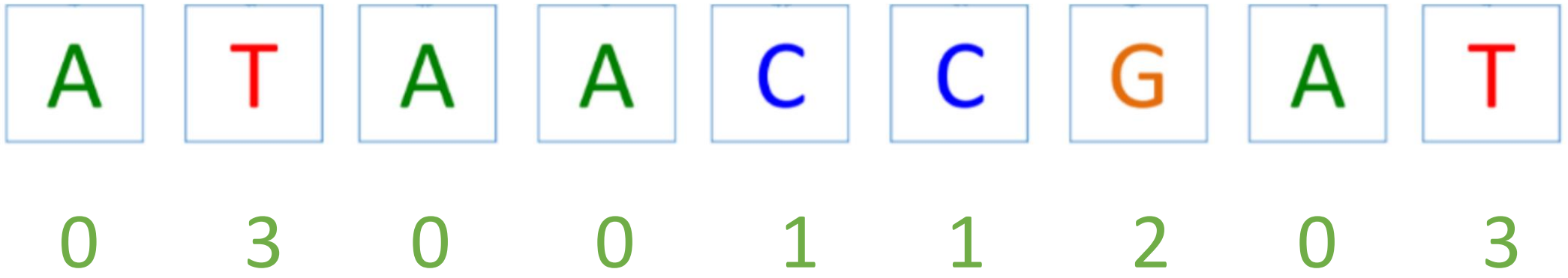
The model



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

The features

The features



The features

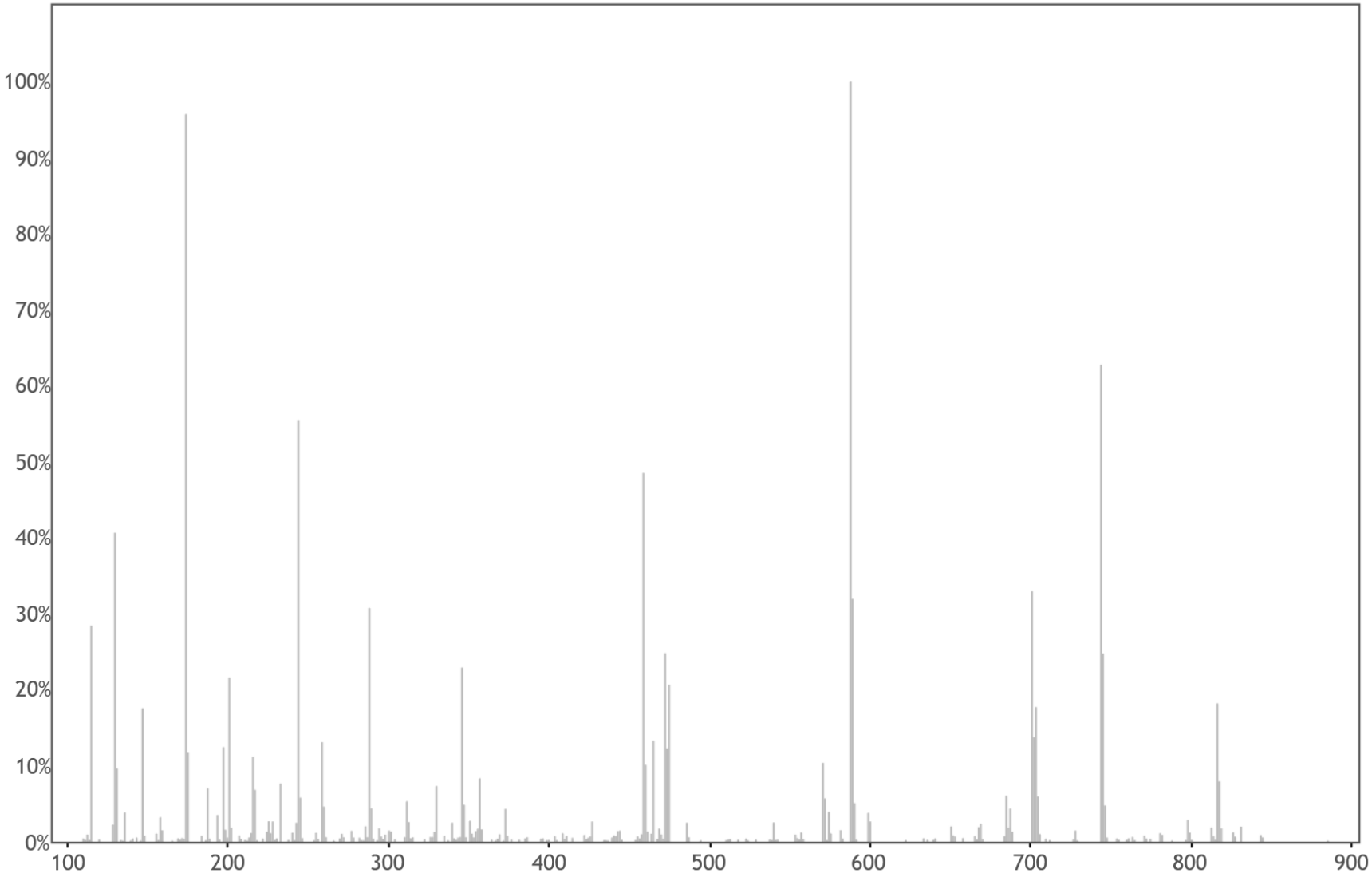
A	T	A	A	C	C	G	A	T
1	0	1	1	0	0	0	1	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1

The features

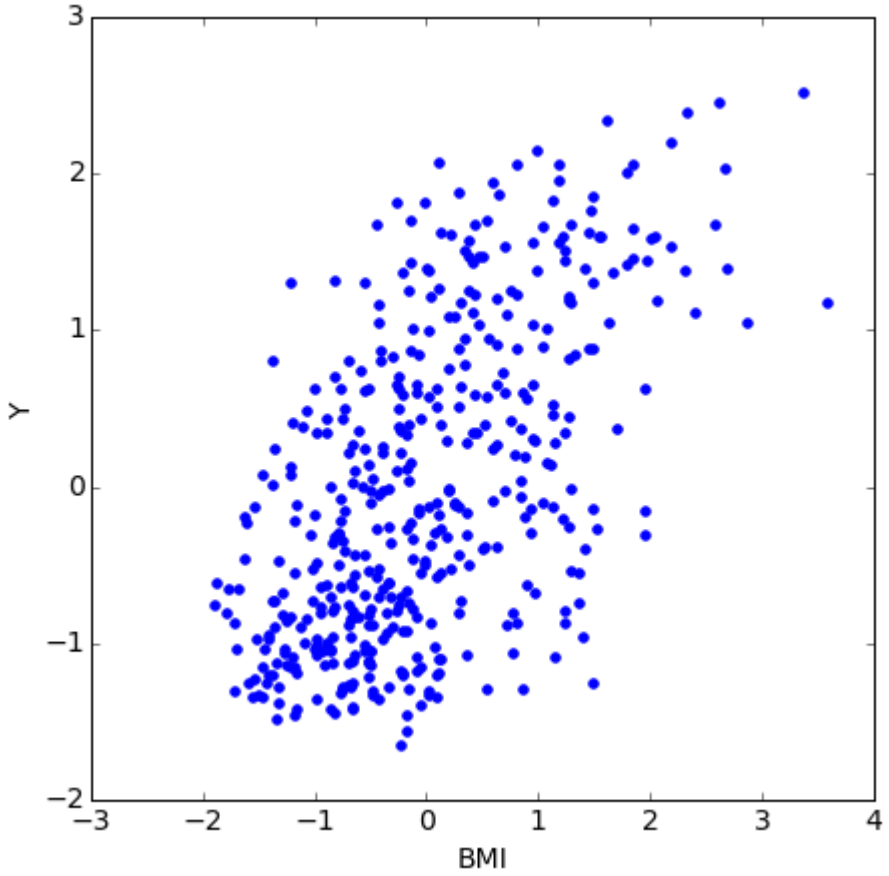
Residue	Mass	Hydrophobicity	Helicity	Basicity
A	71.0788	0.16	1.24	206.4
B	114.5962	-3.14	0.92	210.7
C	103.1388	2.5	0.79	206.2
D	115.0886	-2.49	0.89	208.6
E	129.1155	-1.5	0.85	215.6
F	147.1766	5	1.26	212.1
G	57.0519	-3.31	1.15	202.7
H	137.1411	-4.63	0.97	223.7
I	113.1594	4.41	1.29	210.8
K	128.1741	-5	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

Degroeve S. *et al*., Bioinformatics (2013)

The features



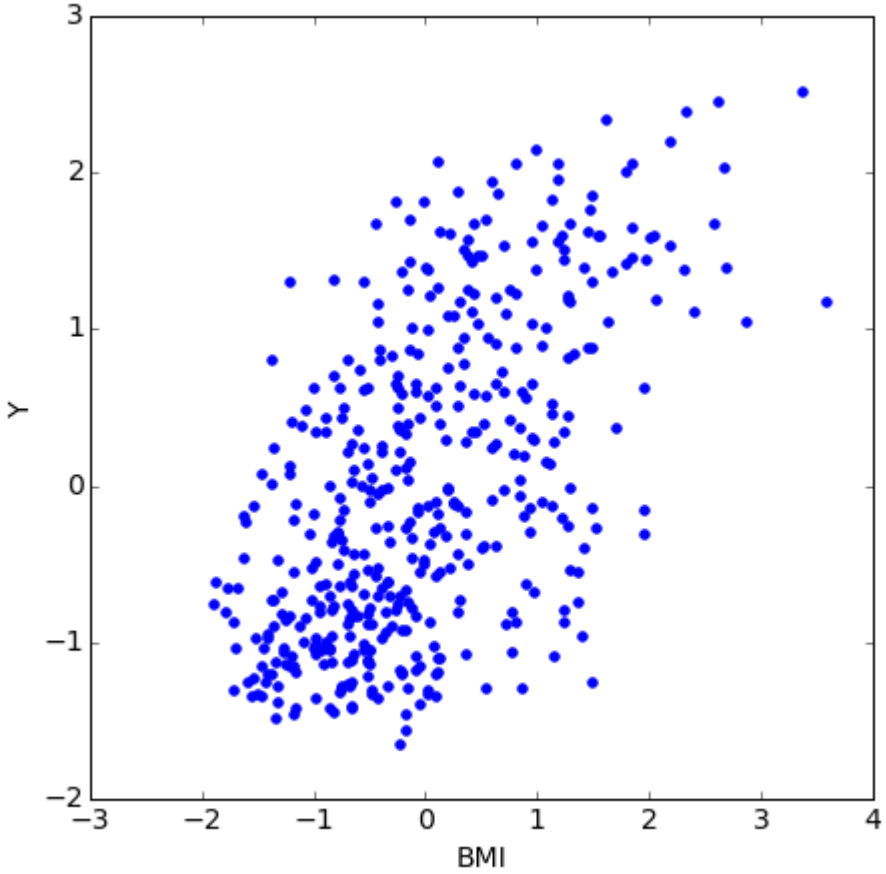
Regression



- Can BMI explain Y?
- Can BMI predict Y?
- How does Y vary with BMI?

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

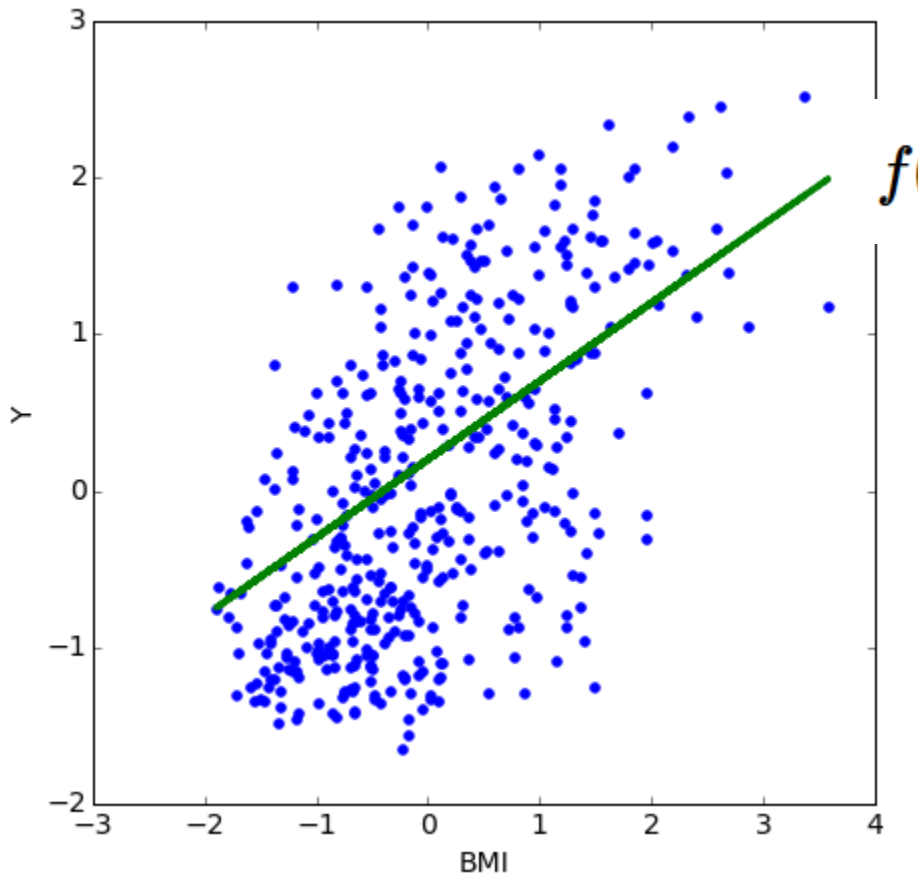
assumptions *linear relationship*

about the

model *linear model*

that generated the data.

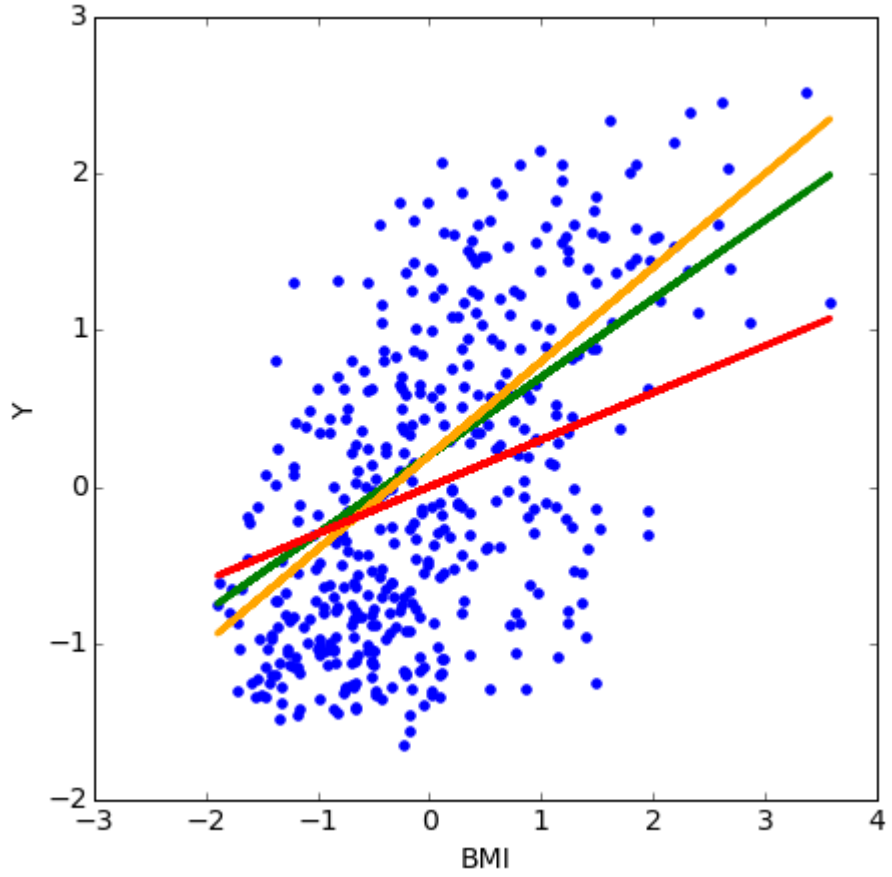
Linear regression



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

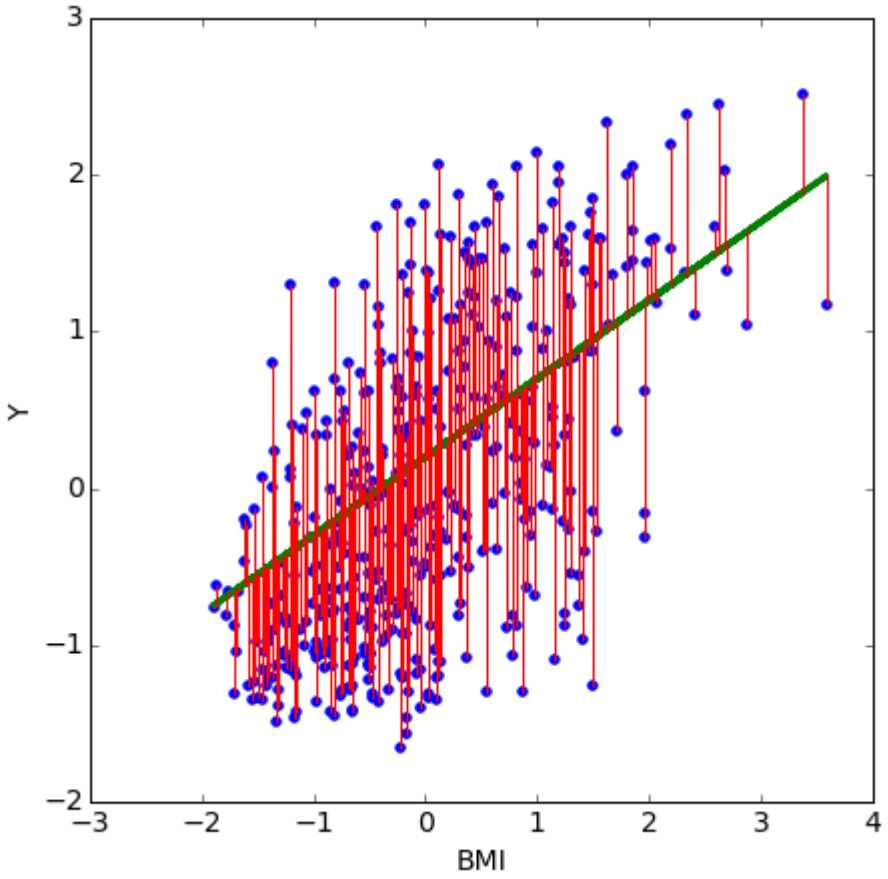
- *a* and *b* are model parameters
- *a* is the slope to the line (direction)
- *b* is the intercept or bias (position)

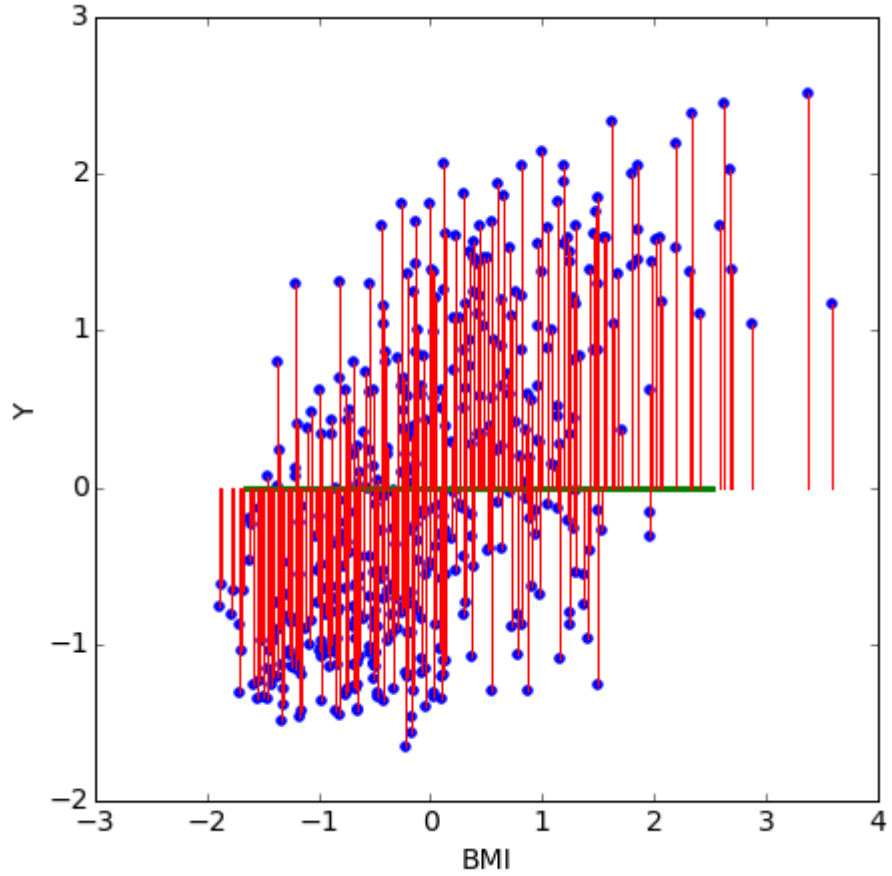
Linear regression: fitting



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

Linear regression: fitting





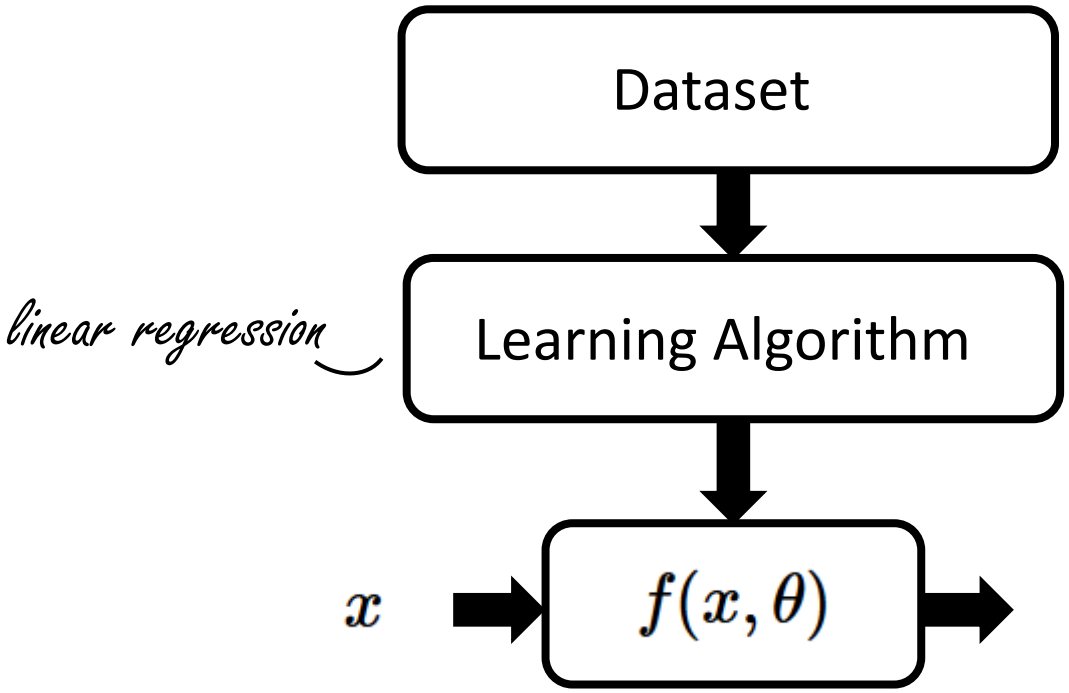
$$R^2 = 1 - \frac{SS_{error}}{SS_{total}}$$

$$R^2 = 0.296$$

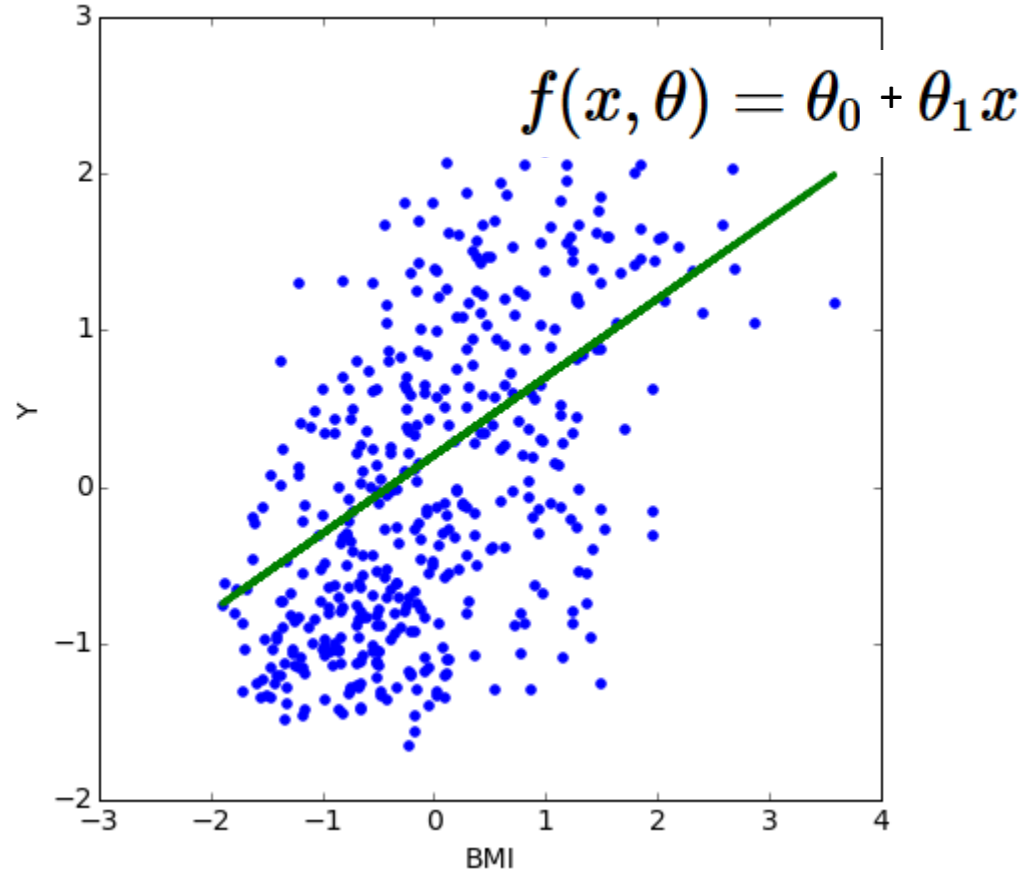
$$SS_{error} = \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}))^2$$

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

Linear regression



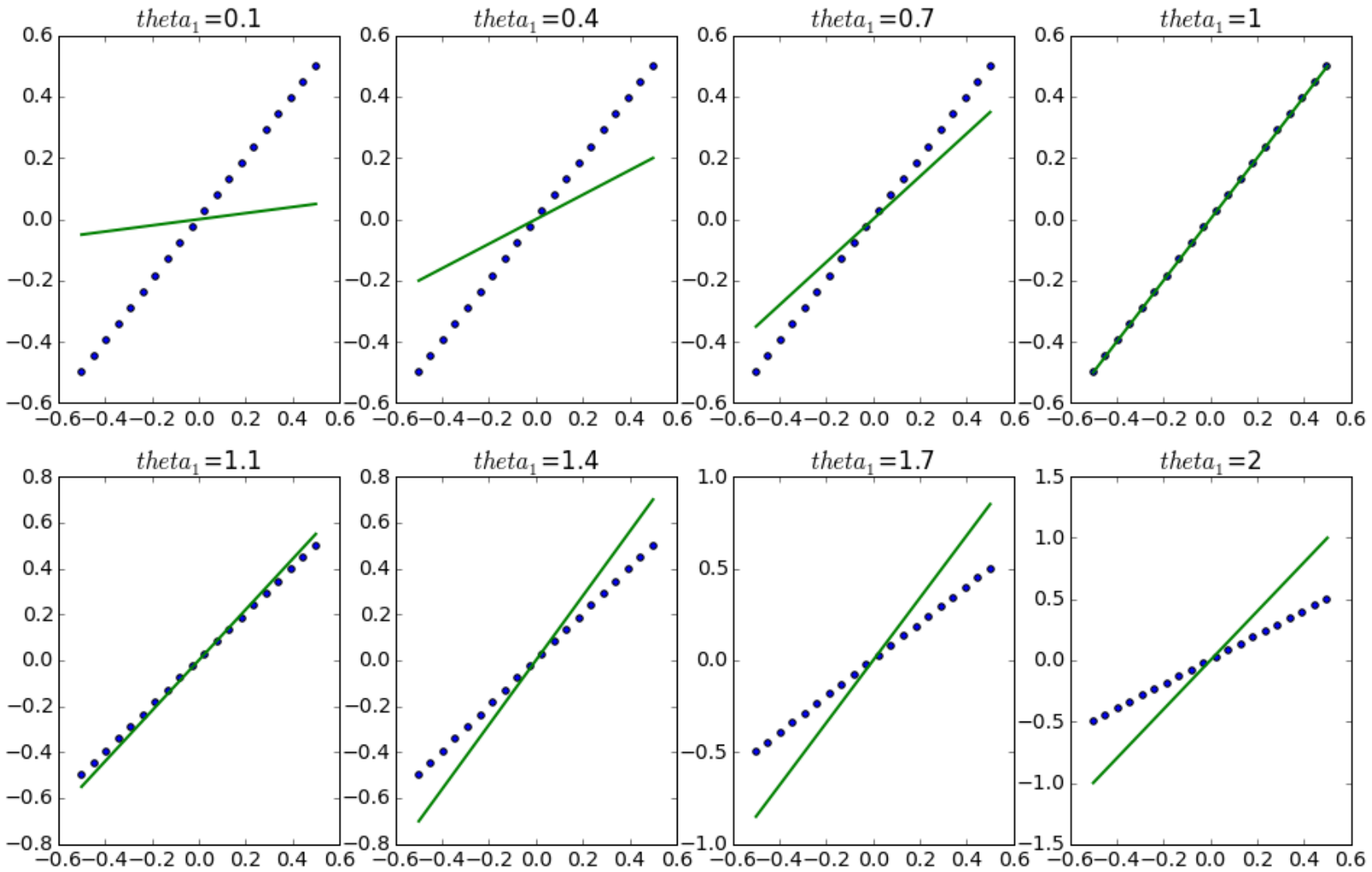
$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$



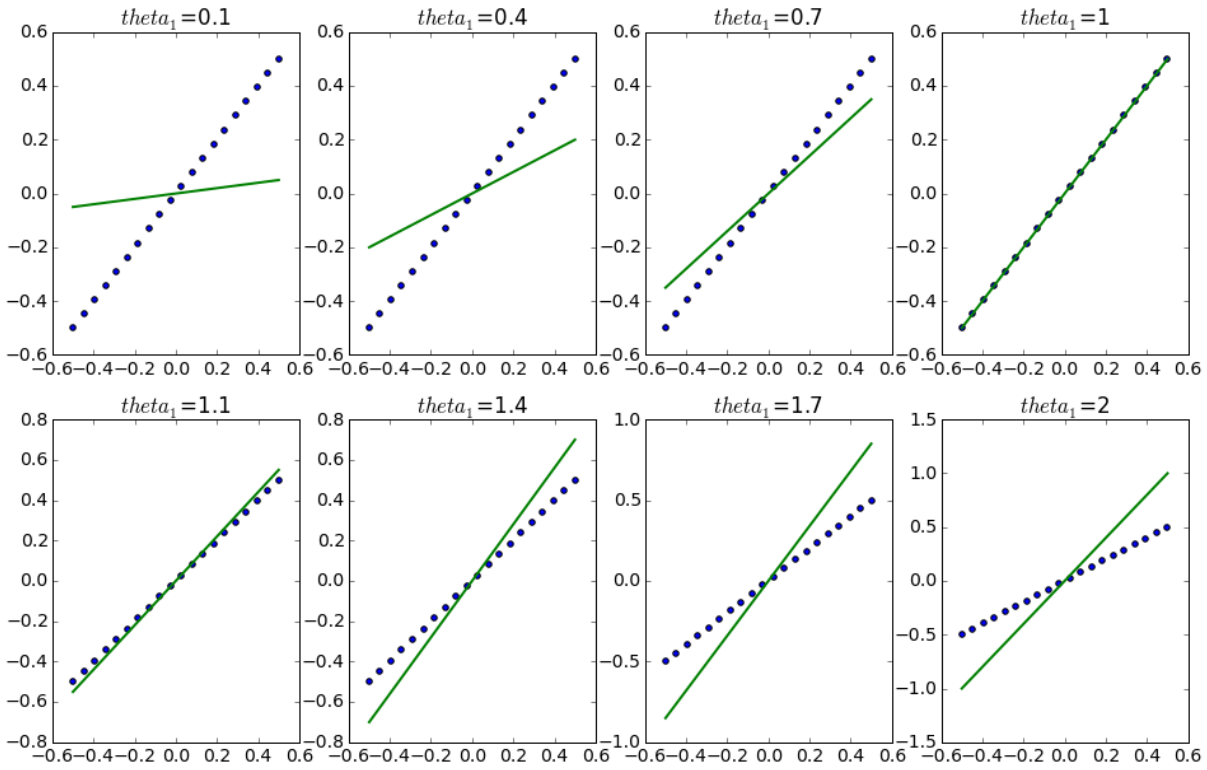
Linear regression: cost function

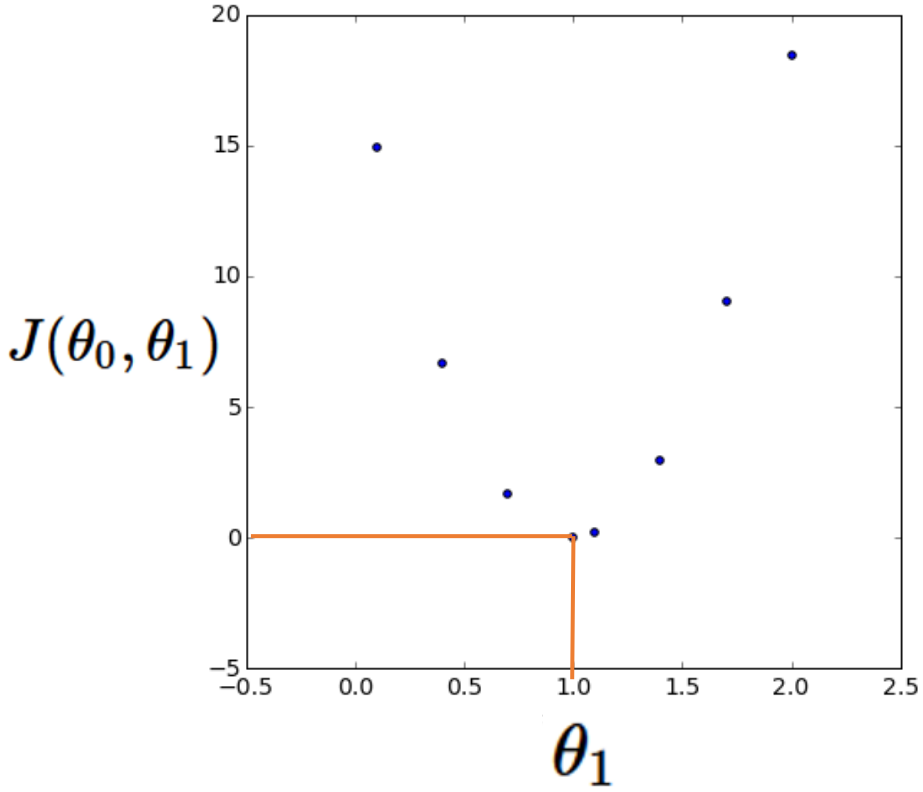
$$f(x) = x$$

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



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

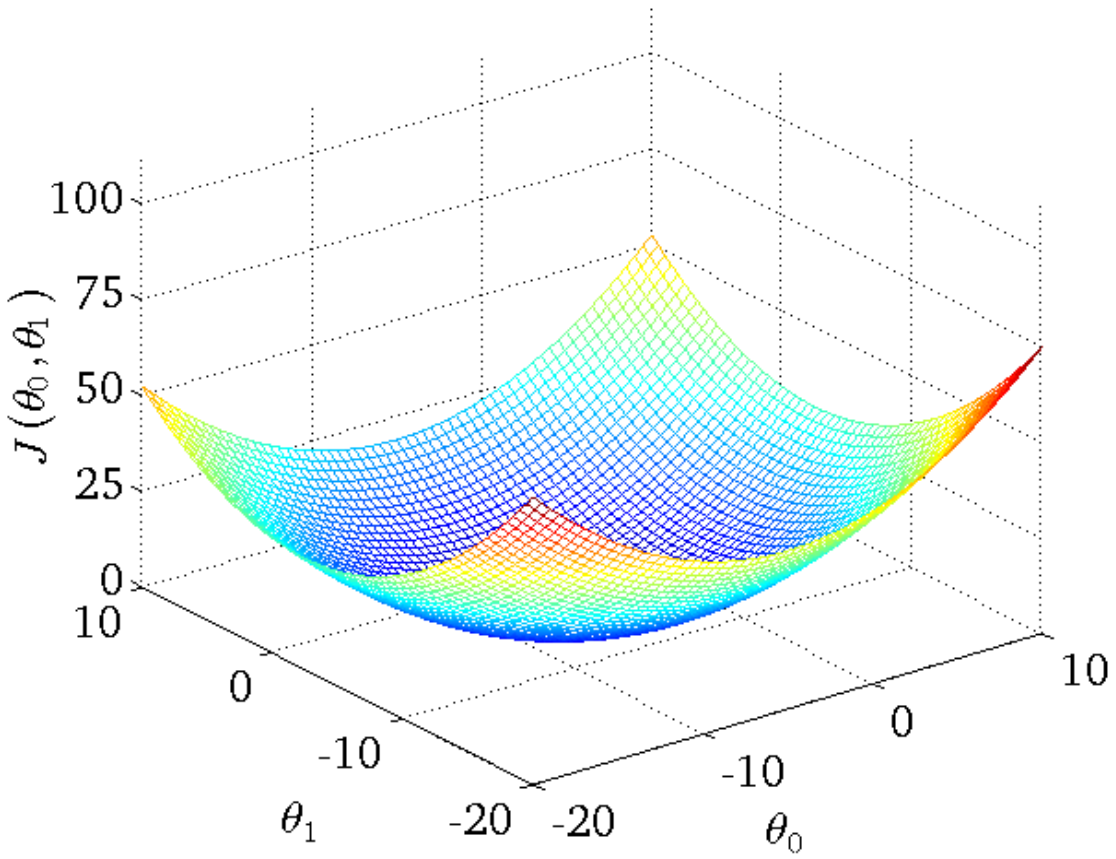
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$

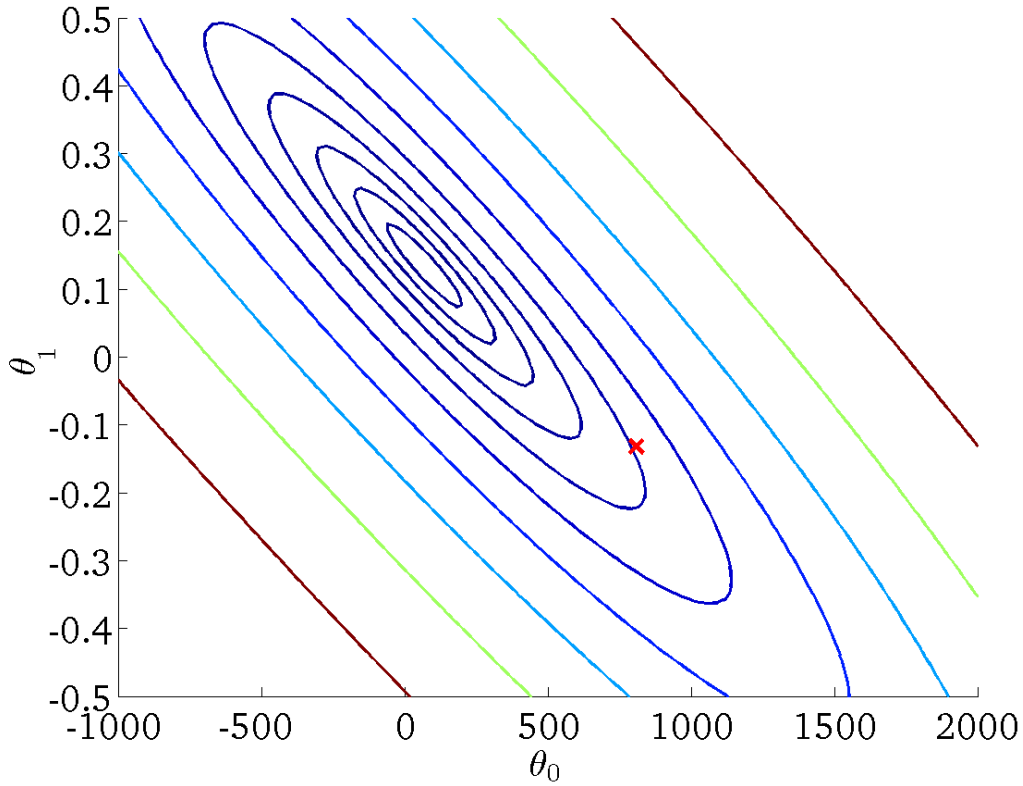
to the data set such that the cost function

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

is minimal.

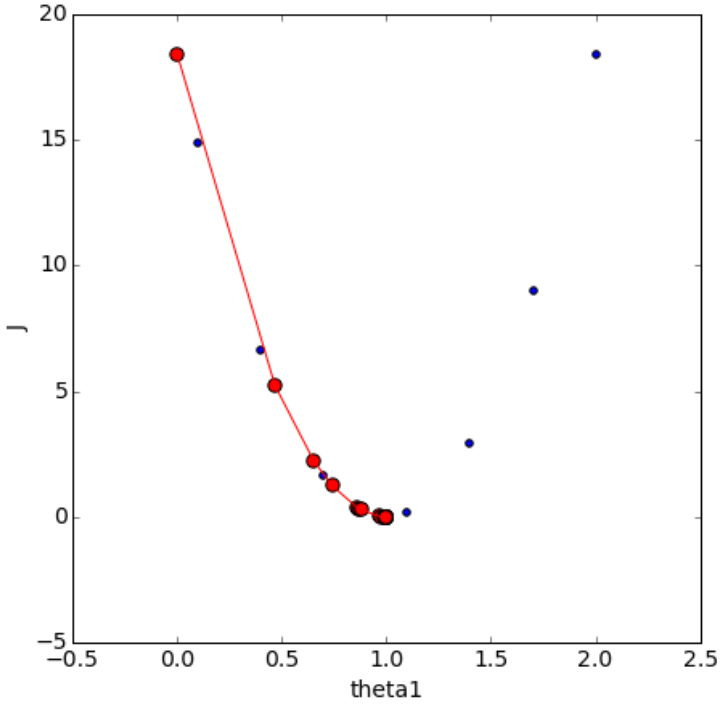
Linear regression: gradient descent

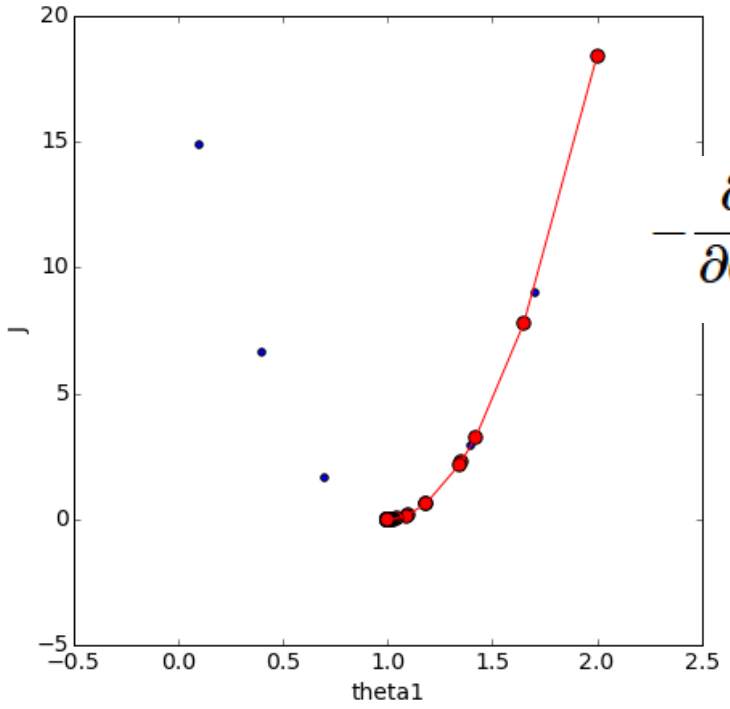




Linear regression: gradient descent

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





$$-\frac{\partial}{\partial \theta_j} J(\theta)$$

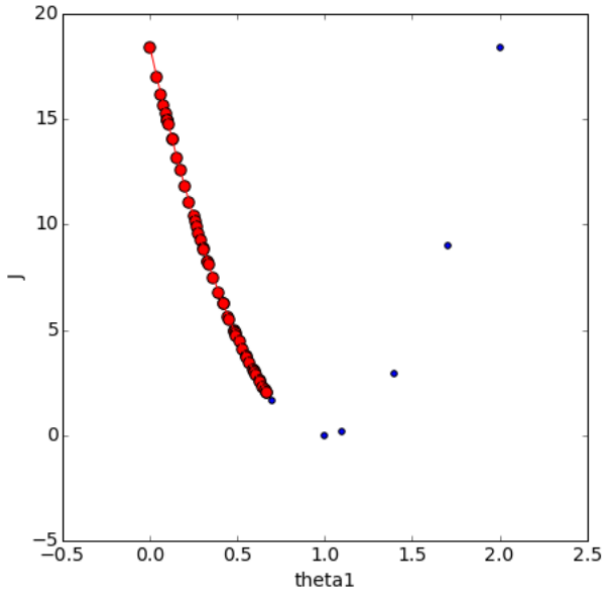
Linear regression: gradient descent

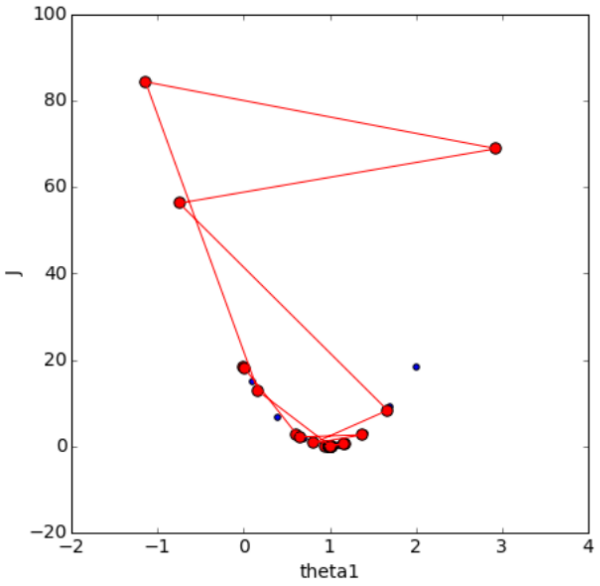
$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

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

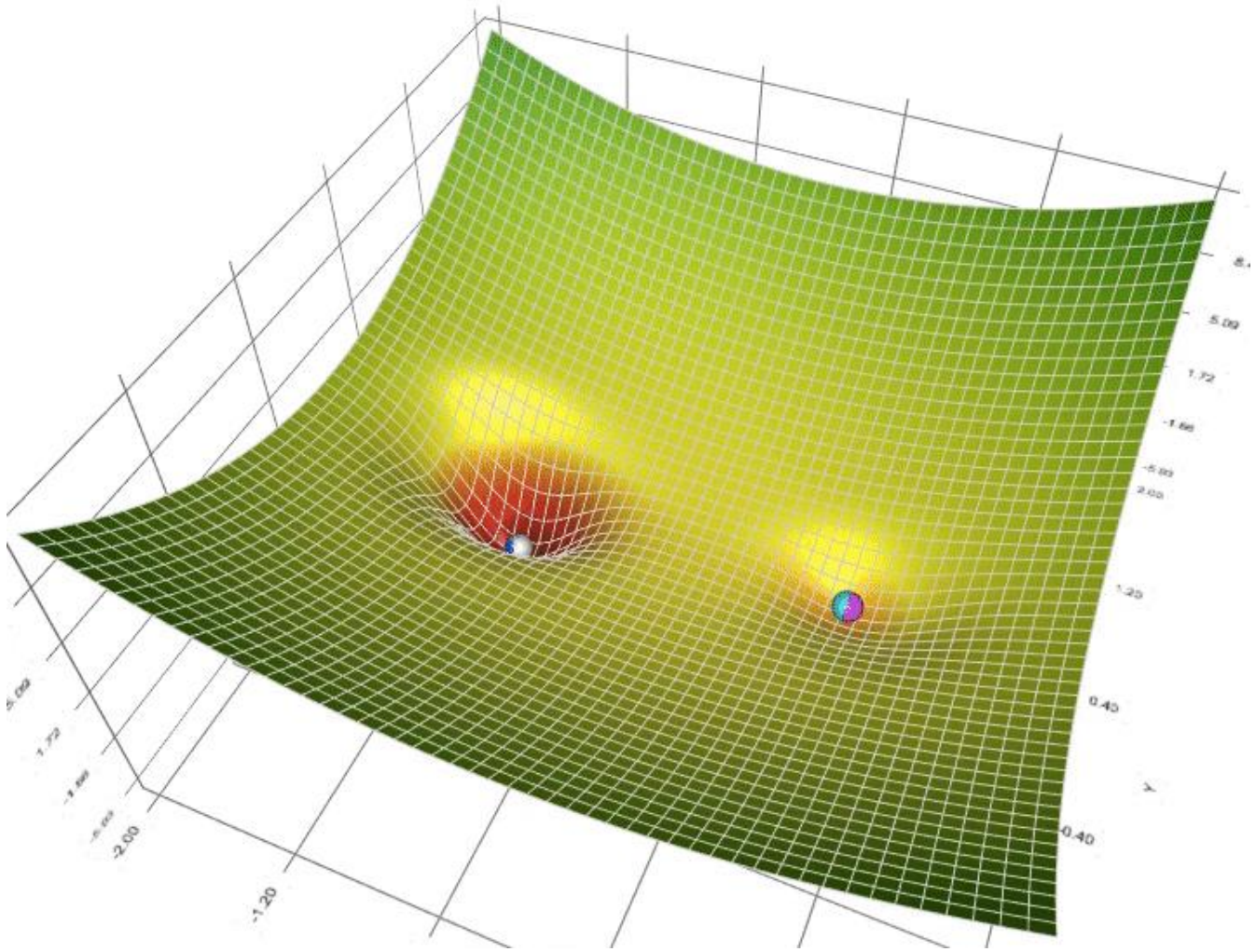
$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

$$\dots$$





Linear regression: gradient descent



Linear regression

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

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

Goal: $\underset{J(\theta)}{\text{minimize}} \quad J(\theta)$

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

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

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

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

...

Data normalization

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

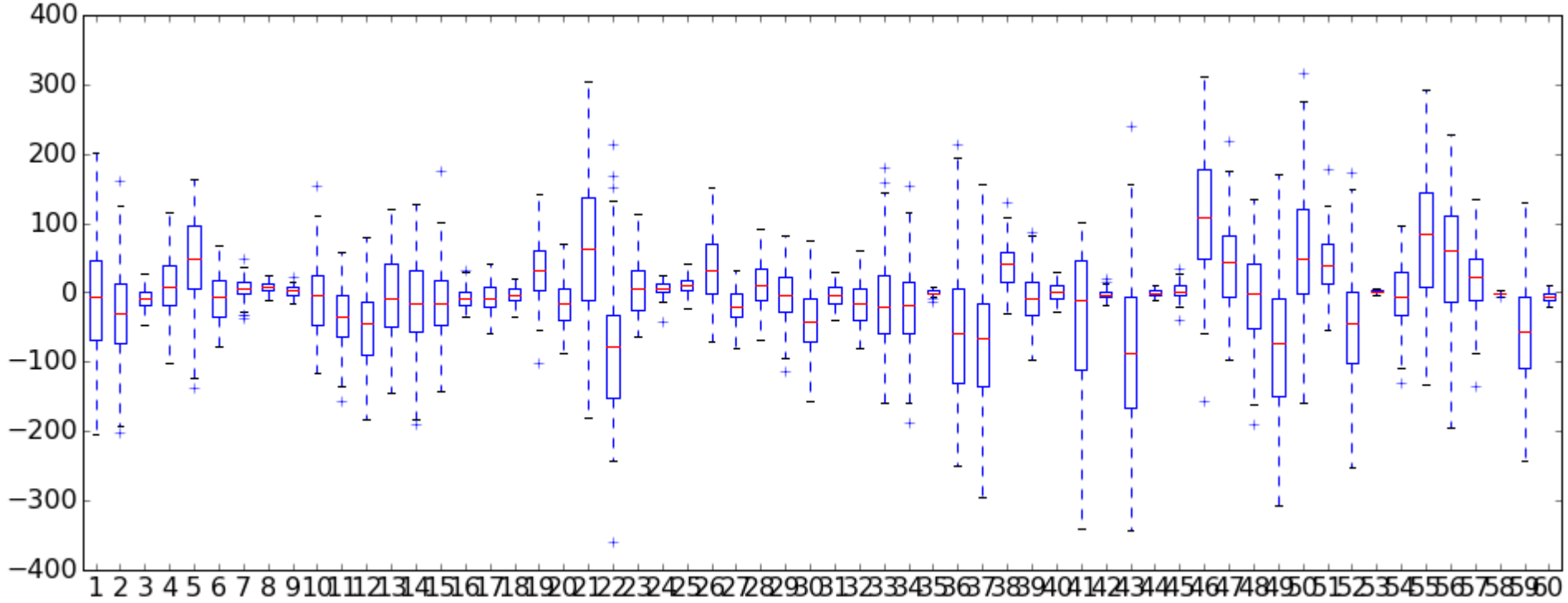
$$\mu = 0 \quad \sigma = 1$$

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

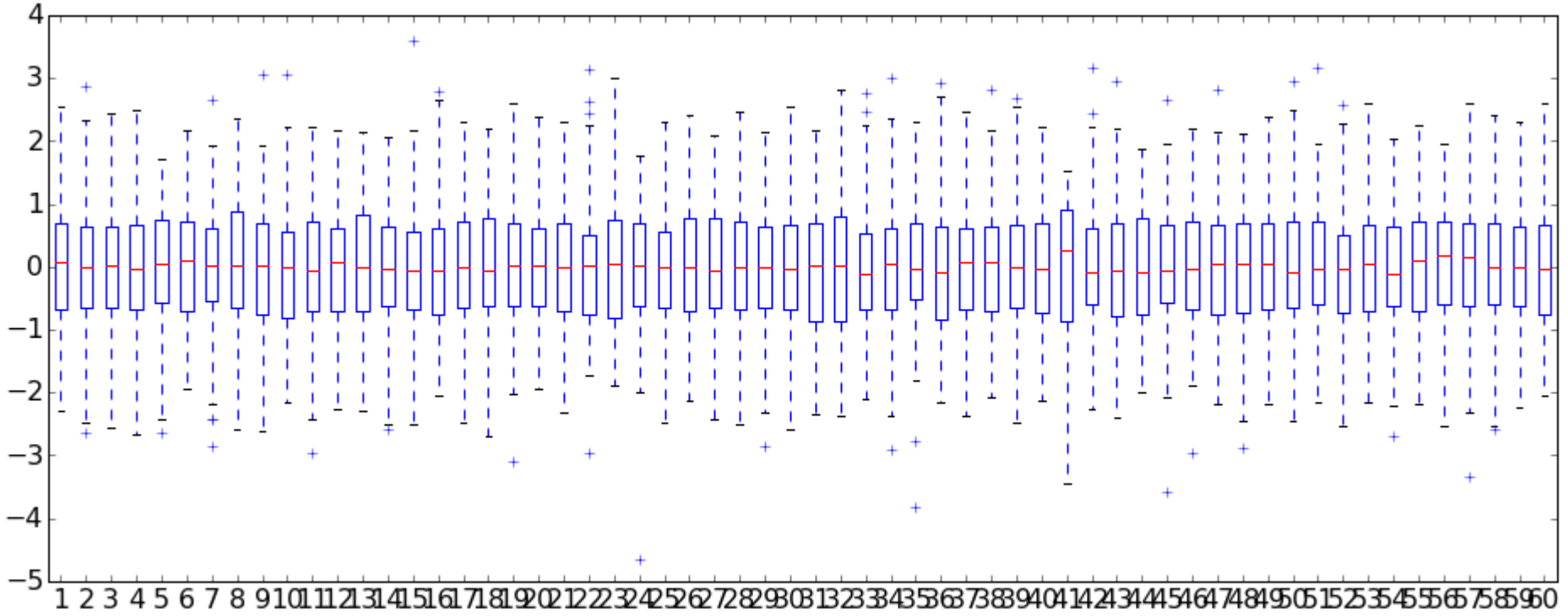
- **min-max scaling**: scale the features to a fixed range

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

Data normalization

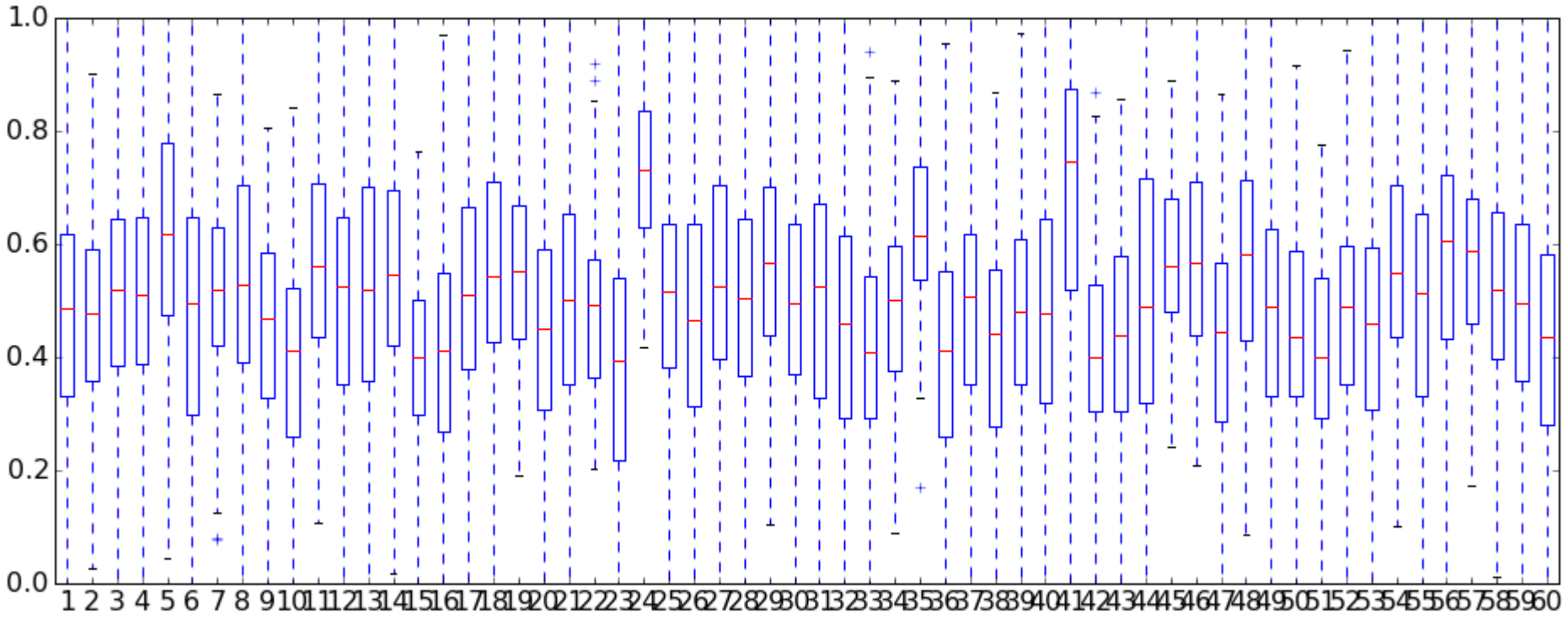


Data normalization



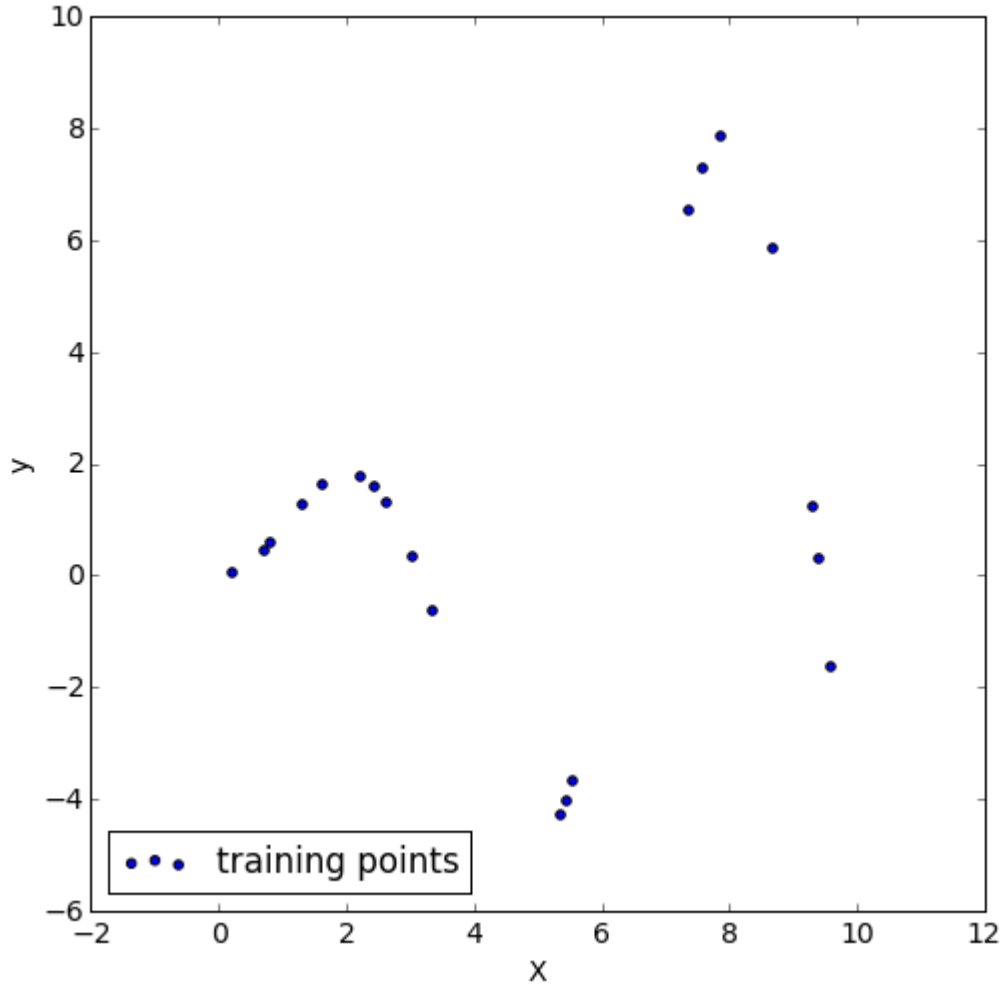
$$x_{norm} = \frac{x - \mu}{\sigma}$$

Data normalization



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

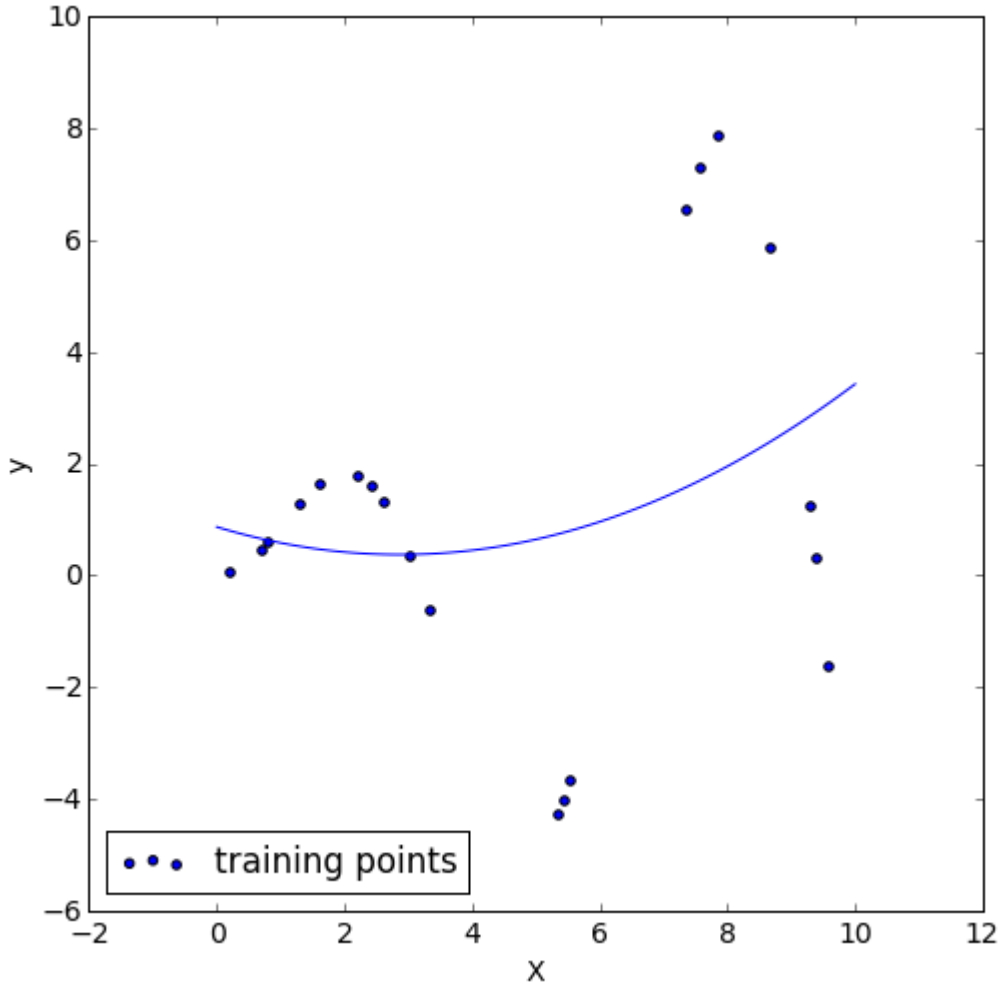
Non-linear regression



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

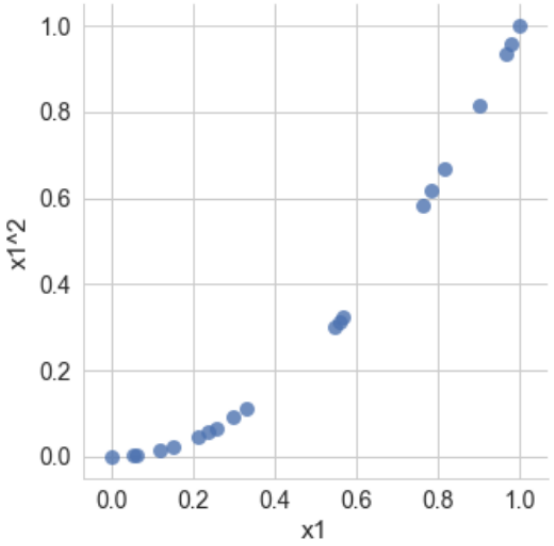
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

Non-linear regression

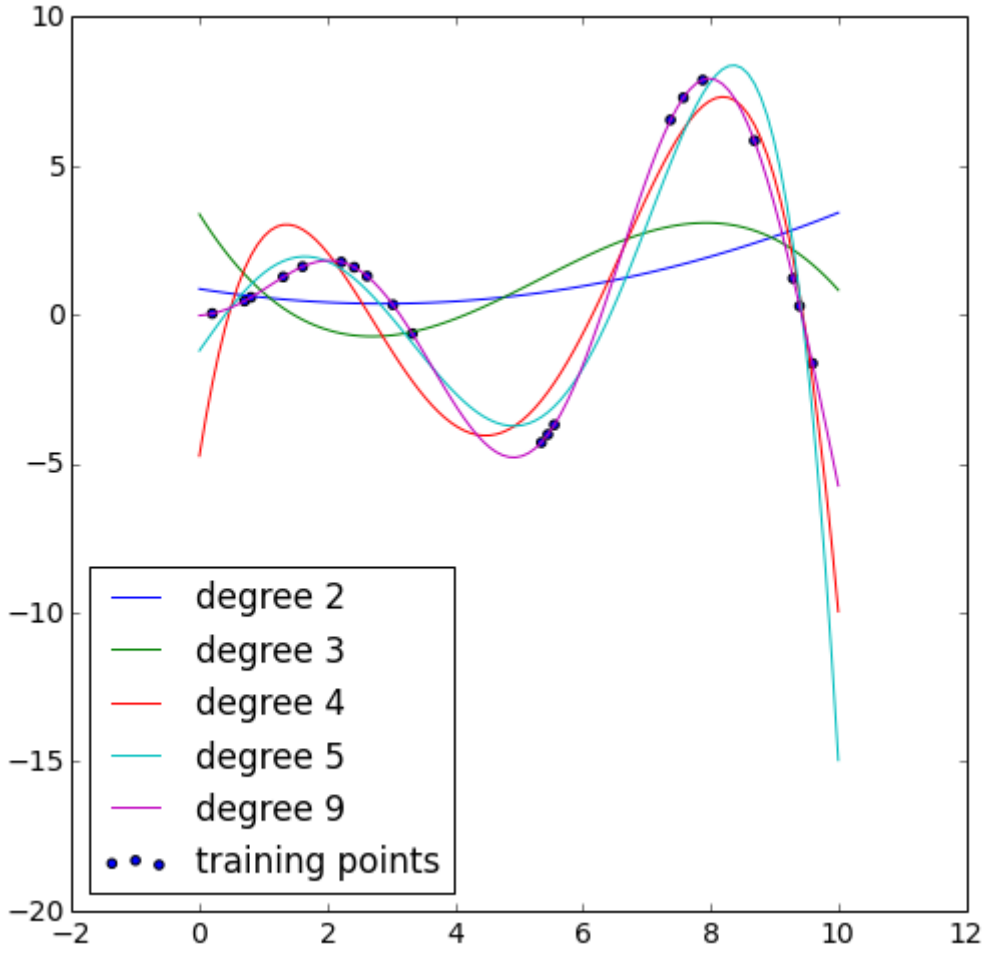


- Does *y* vary linearly with *X*?
- Can we fit a non-linear model? Yes
- Or we could add polynomial transformations of the features.

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



Non-linear regression

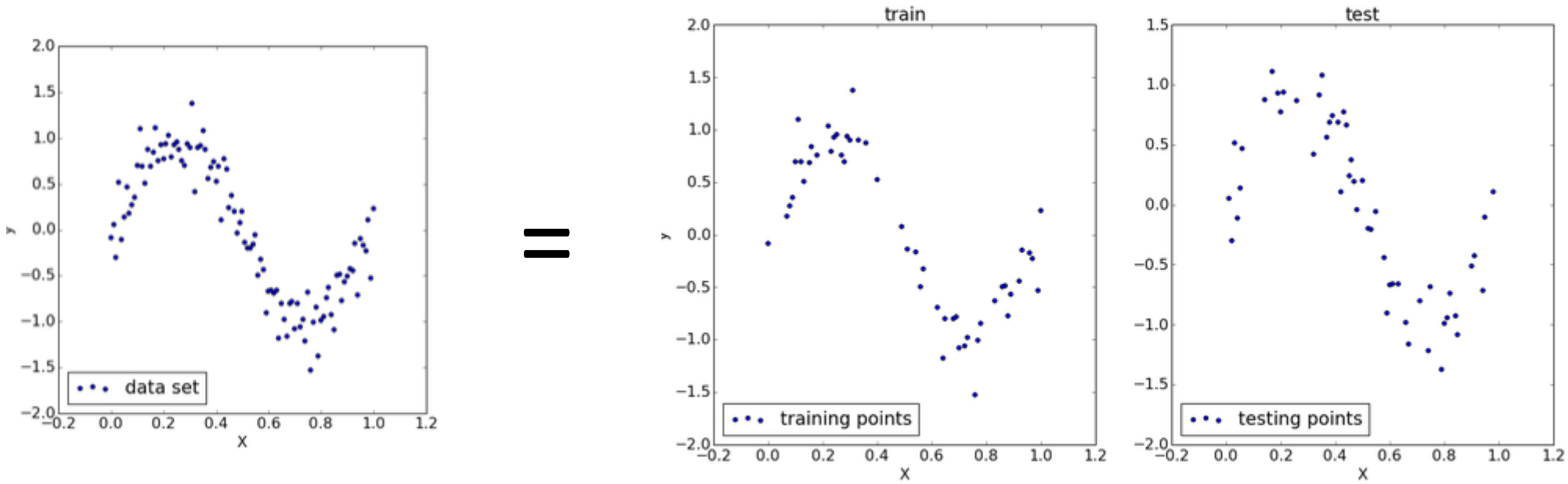


$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^4 + \theta_5 x_1^5 + \theta_6 x_1^6 + \theta_7 x_1^7 + \theta_8 x_1^8 + \theta_9 x_1^9$$

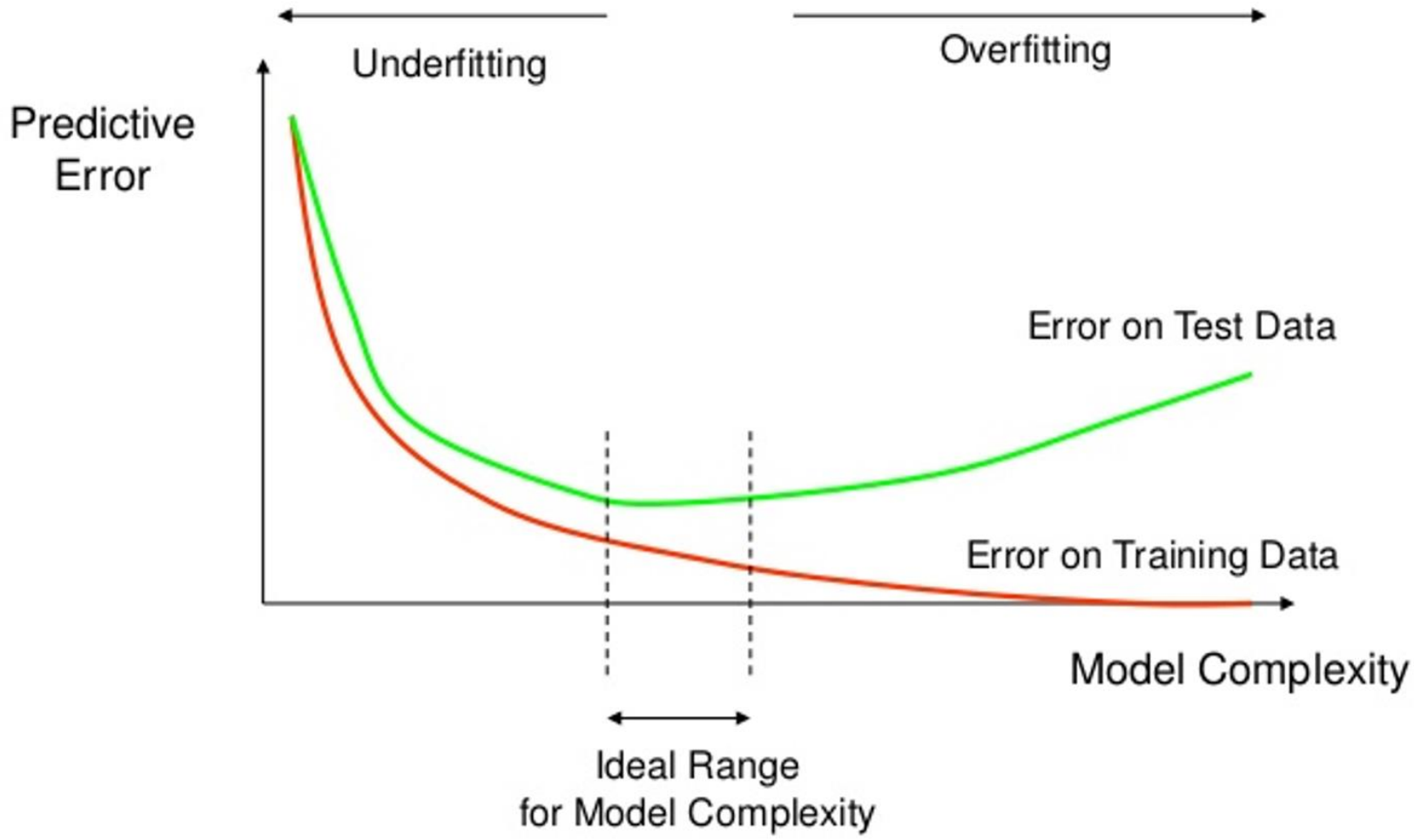
We want to estimate the prediction performance of the model on unseen external data

(data not seen during training)

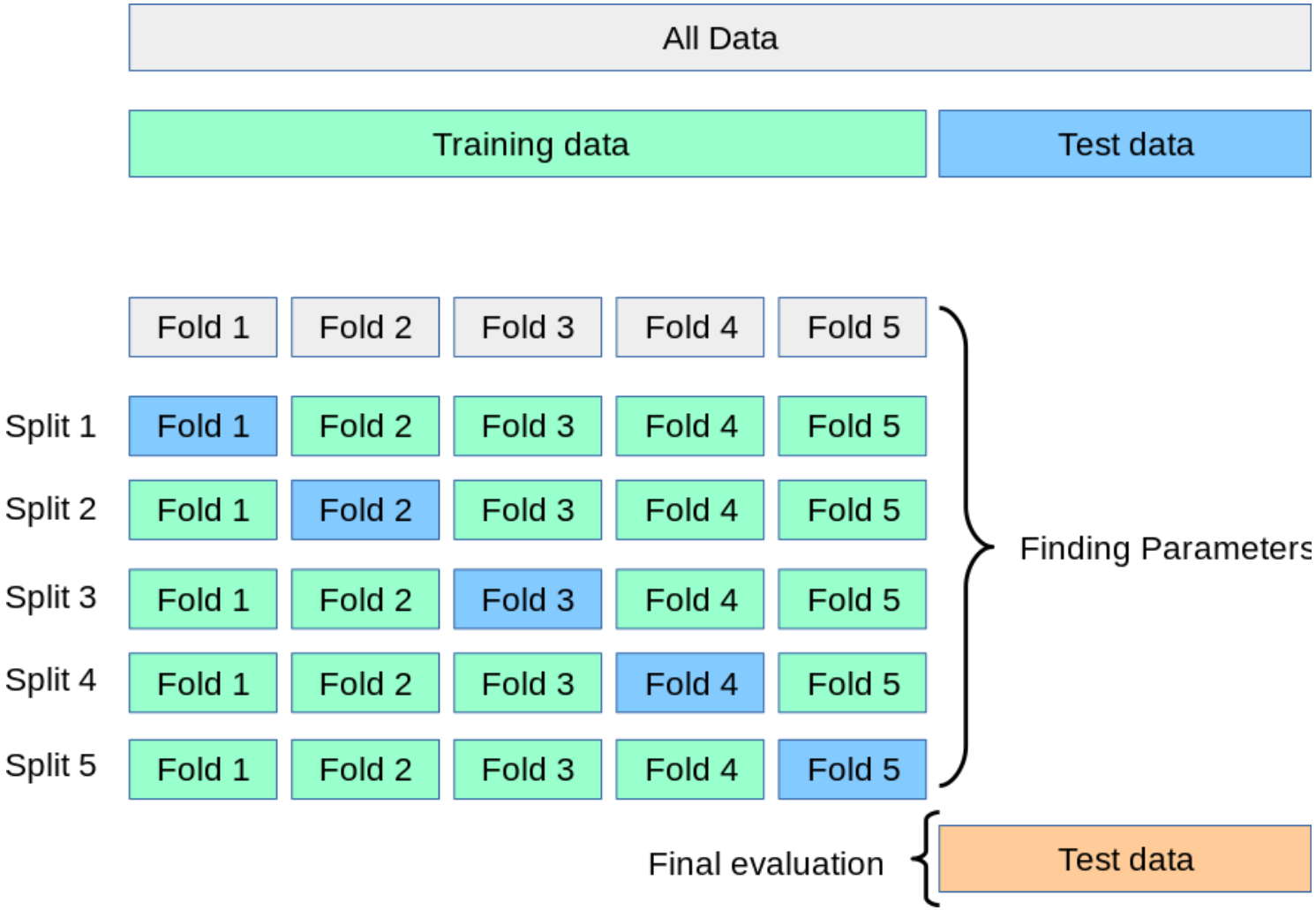
Model evaluation: train-test split



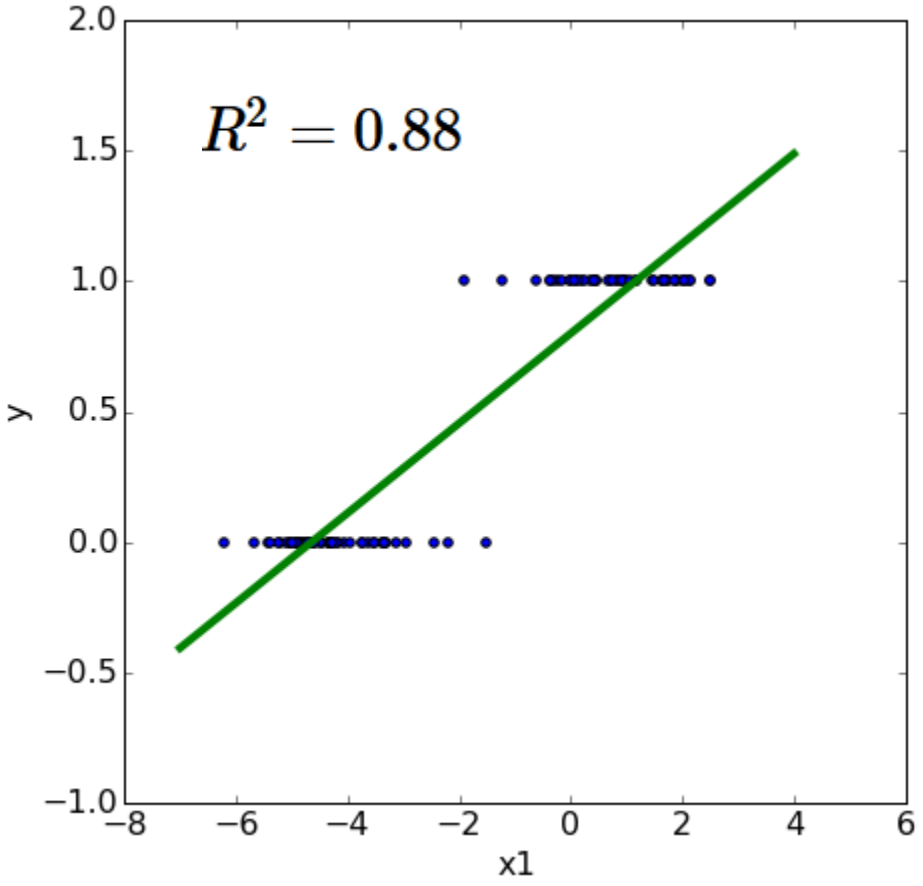
Model selection: overfitting



Model evaluation: cross-validation



Logistic regression



We need to make

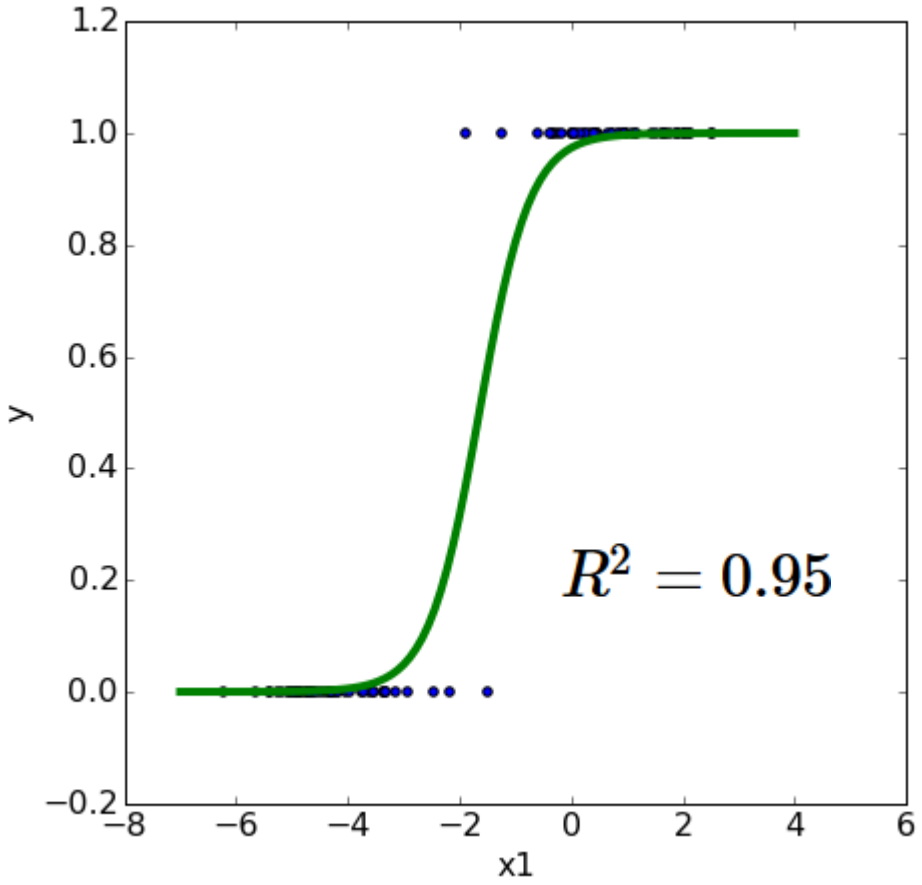
assumptions — *linear relationship*

about the

model — *linear model*

that generated the data.

Logistic regression



We need to make

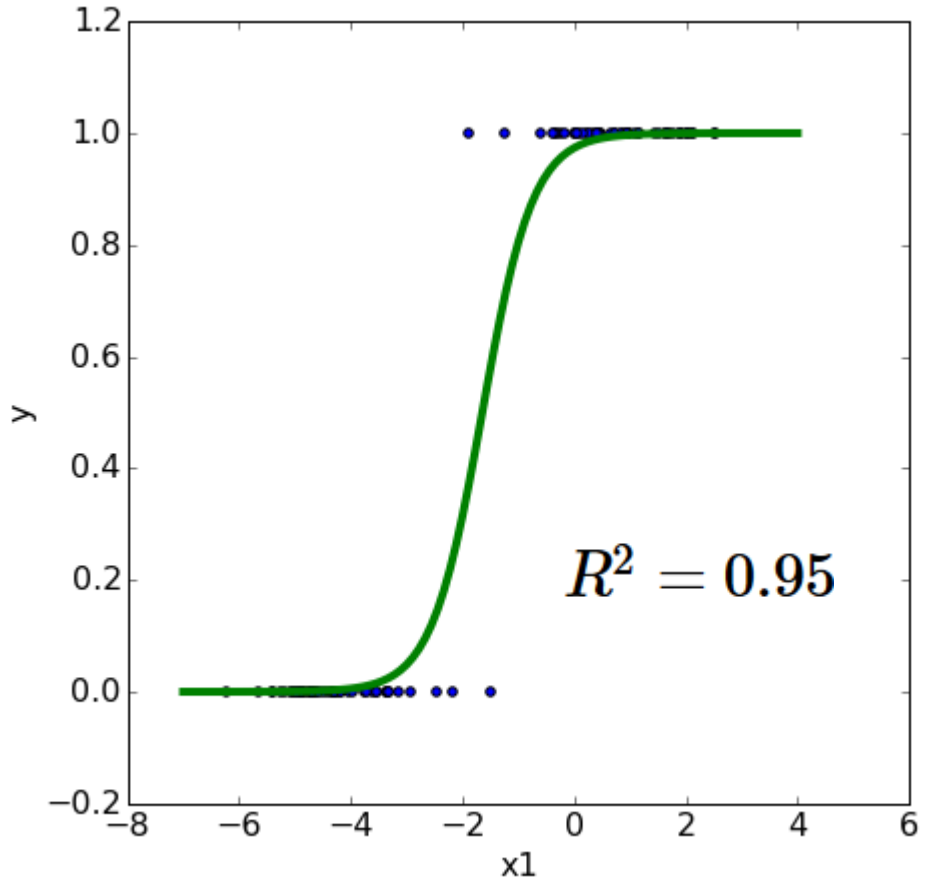
assumptions *linearly separable*

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

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

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

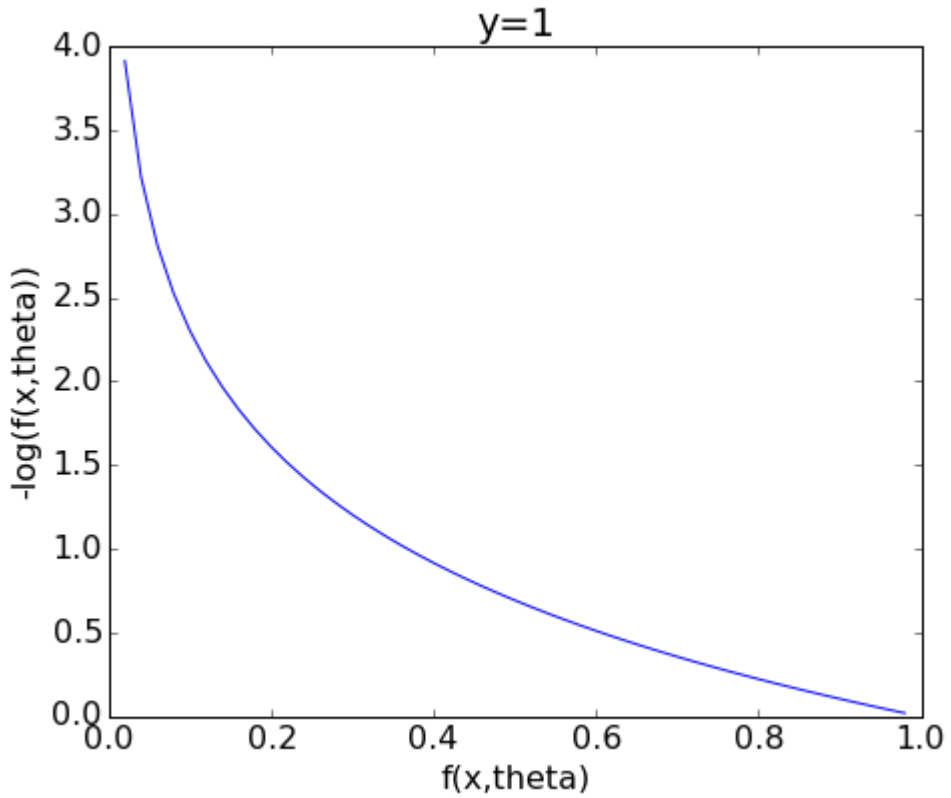
$-log(f(x^{(i)}, \theta))$.

Similarly, if $y^{(i)} = 0$ then the cost function $J(\theta)$ is incremented by

$-log(1 - f(x^{(i)}, \theta))$.

Logistic regression: cost function

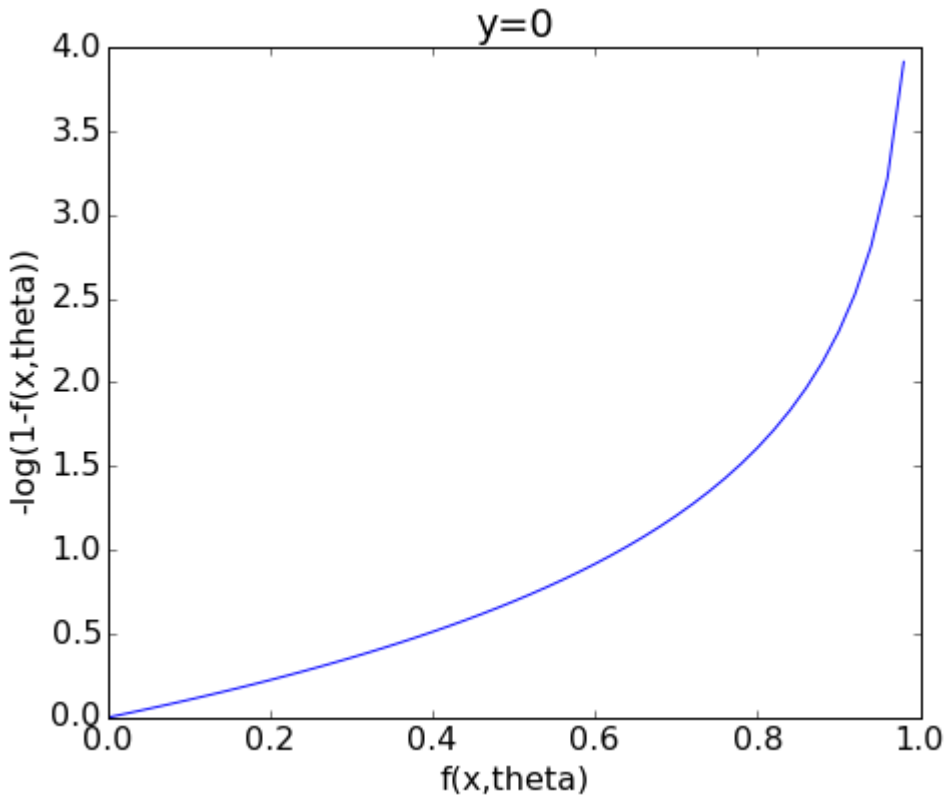
We know that $y^{(i)}$ is either 0 or 1. If $y^{(i)} = 1$ then the cost function $J(\theta)$ is incremented by $-log(f(x^{(i)}, \theta))$.



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

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

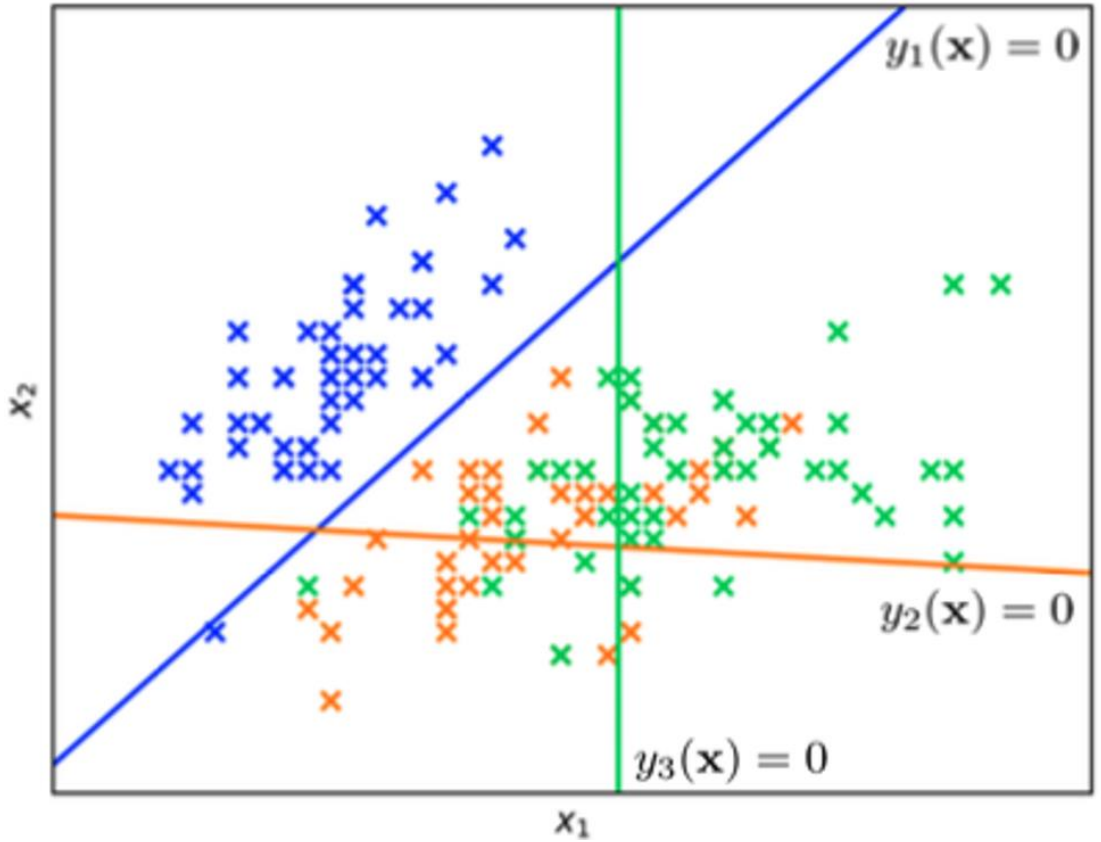
to the data set such that the cost function

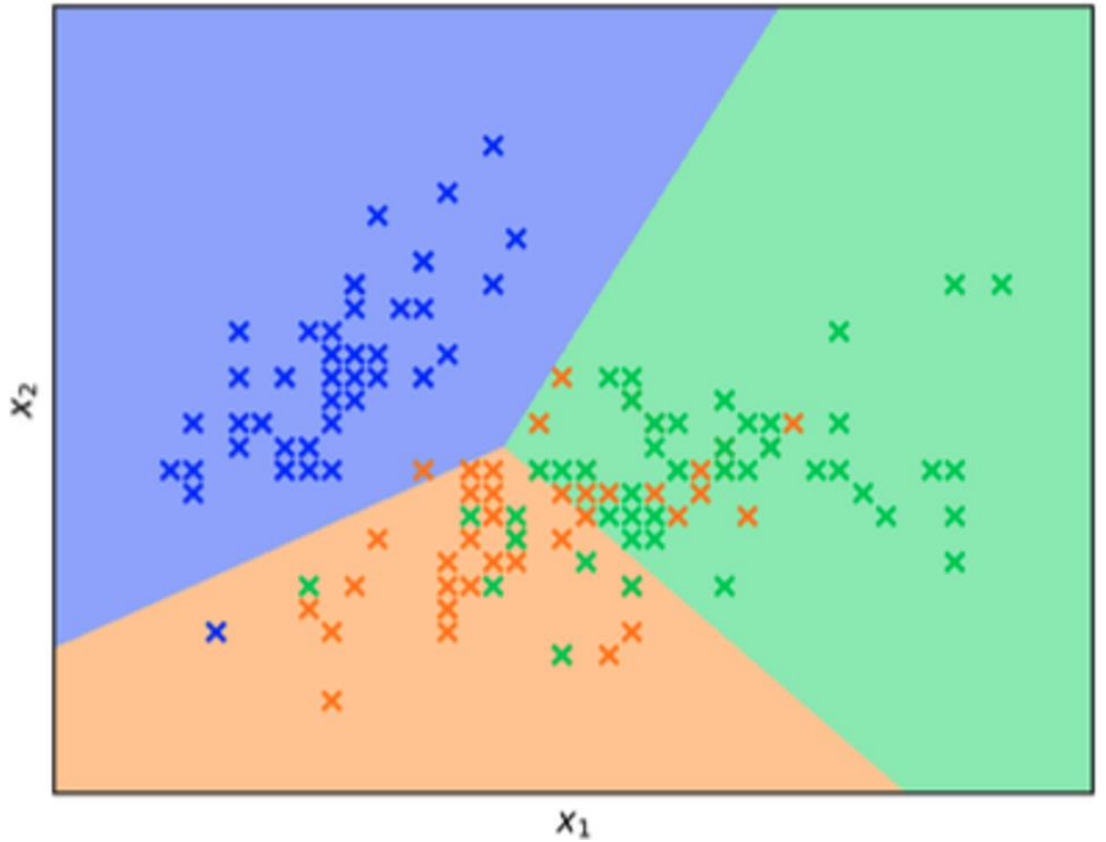
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

is minimal using gradient descent

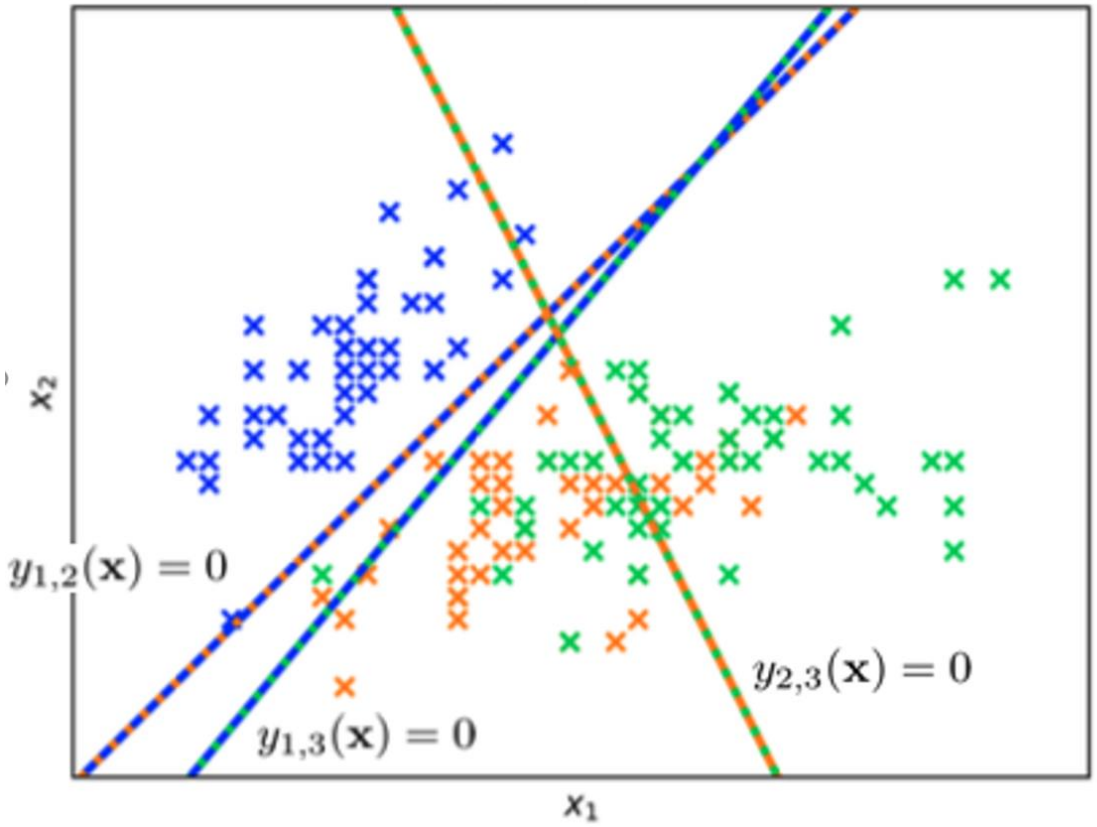
$$\theta_j := \theta_j - \alpha \frac{1}{n}\sum_{i=1}^{n} (f(x^{(i)},\theta) - y^{(i)}) x_j^{(i)}$$

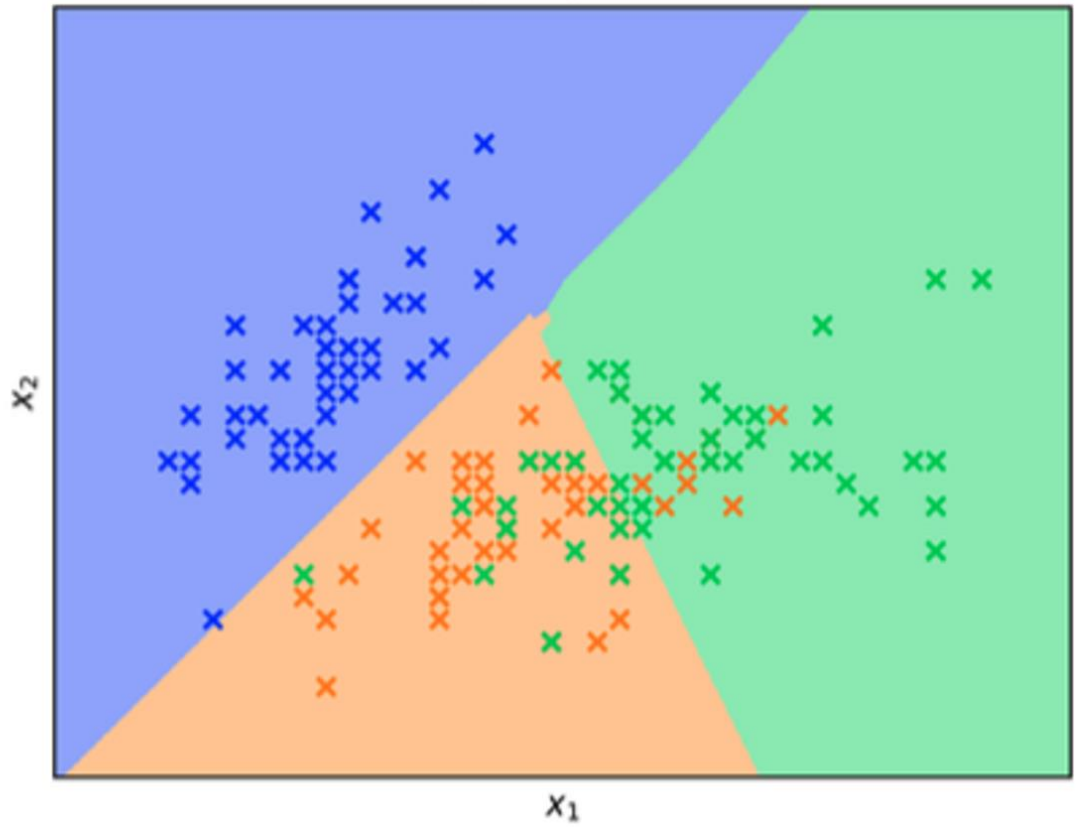
Muti-class classification: one vs rest





Muti-class classification: one vs one



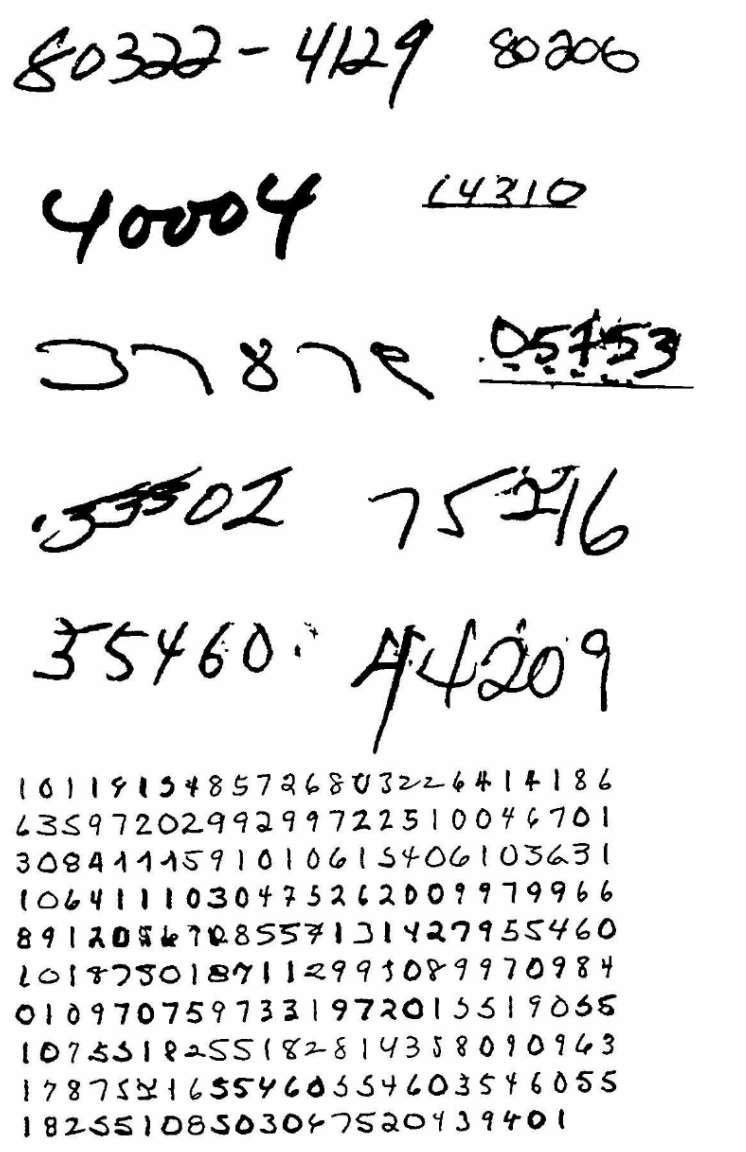


Let’s build a model!

Backpropagation Applied to Handwritten Zip Code Recognition

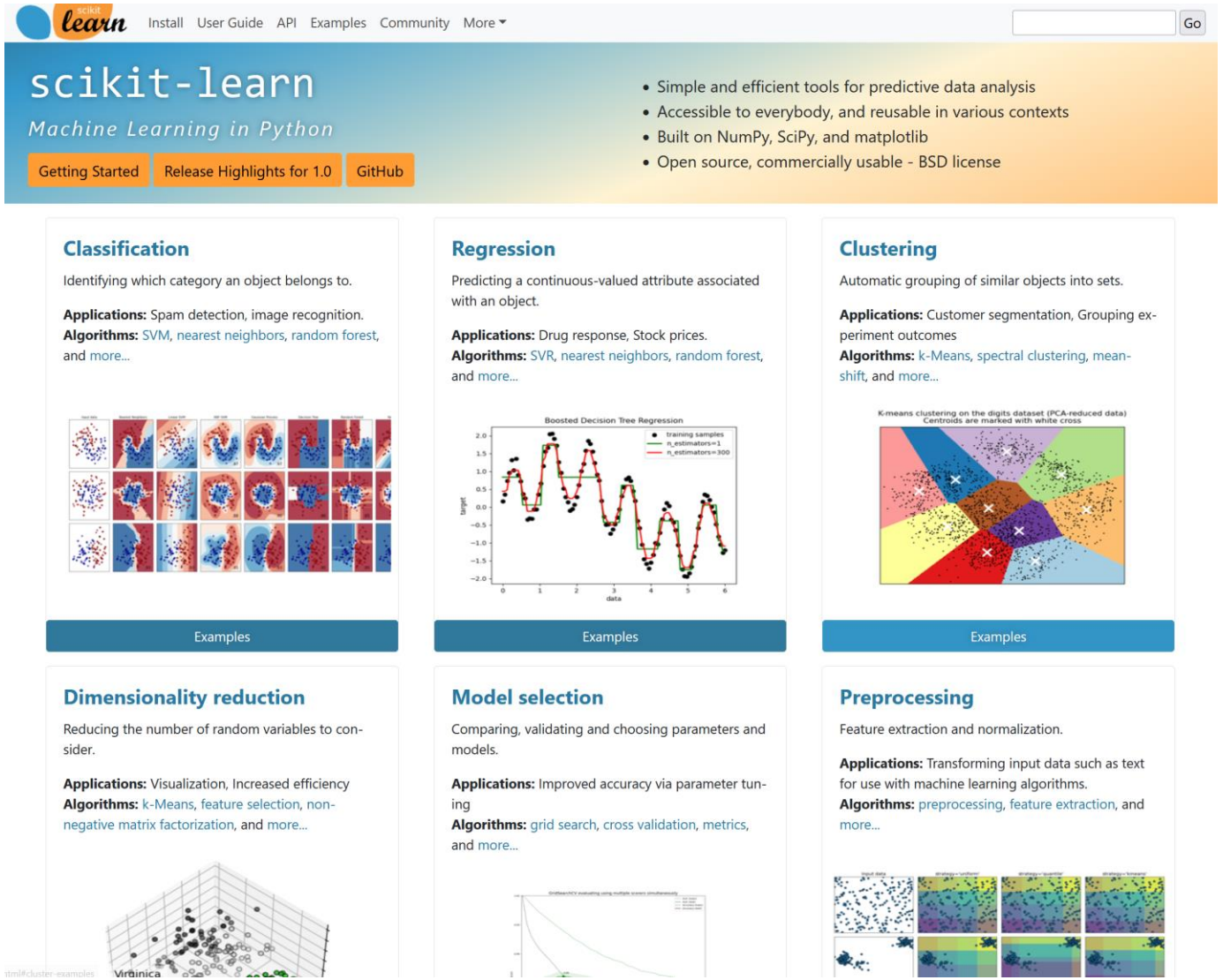
Y. LeCun
B. Boser
J. S. Denker
D. Henderson
R. E. Howard
W. Hubbard
L. D. Jackel
AT&T Bell Laboratories Holmdel, NJ 07733 USA

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.

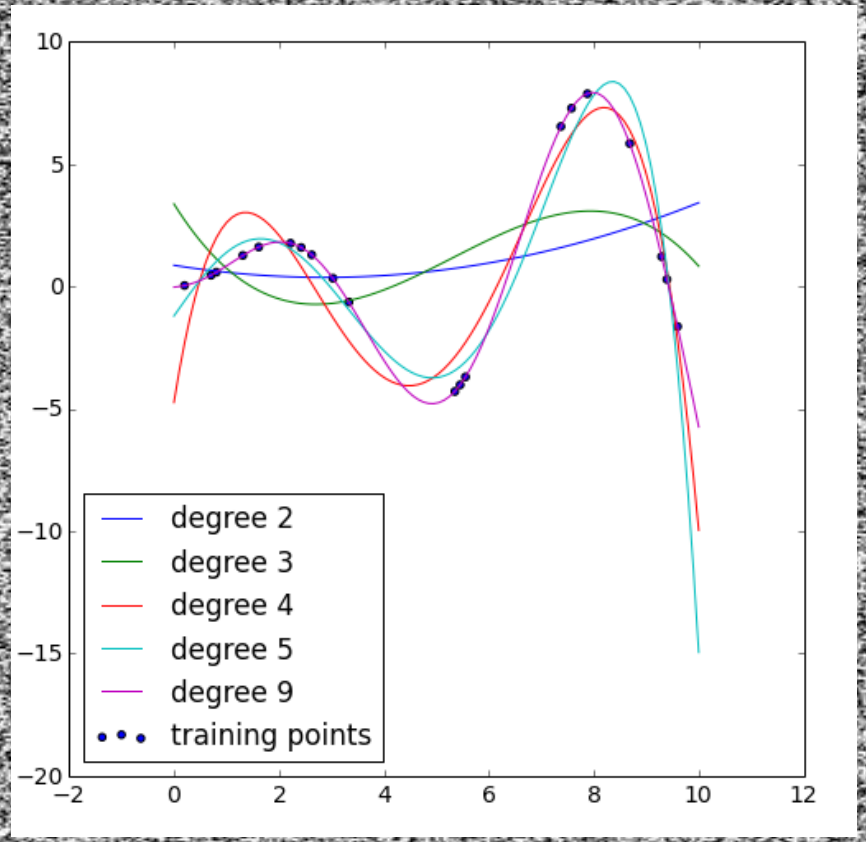


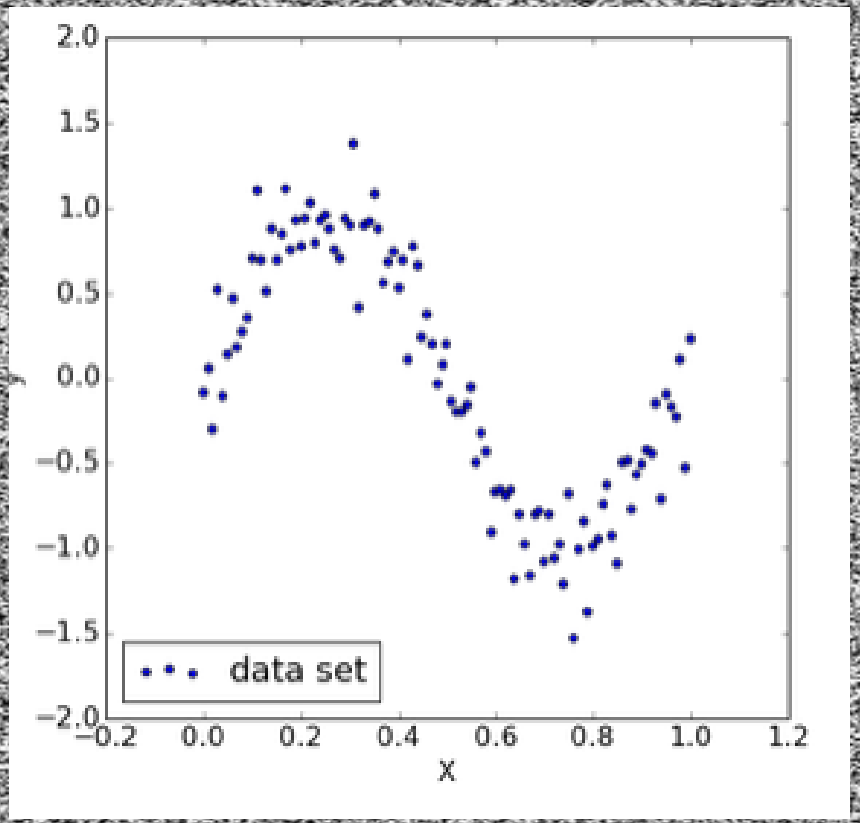
LeCun Y. *et al.*, Neural Computation (1989)

Let’s build a model!

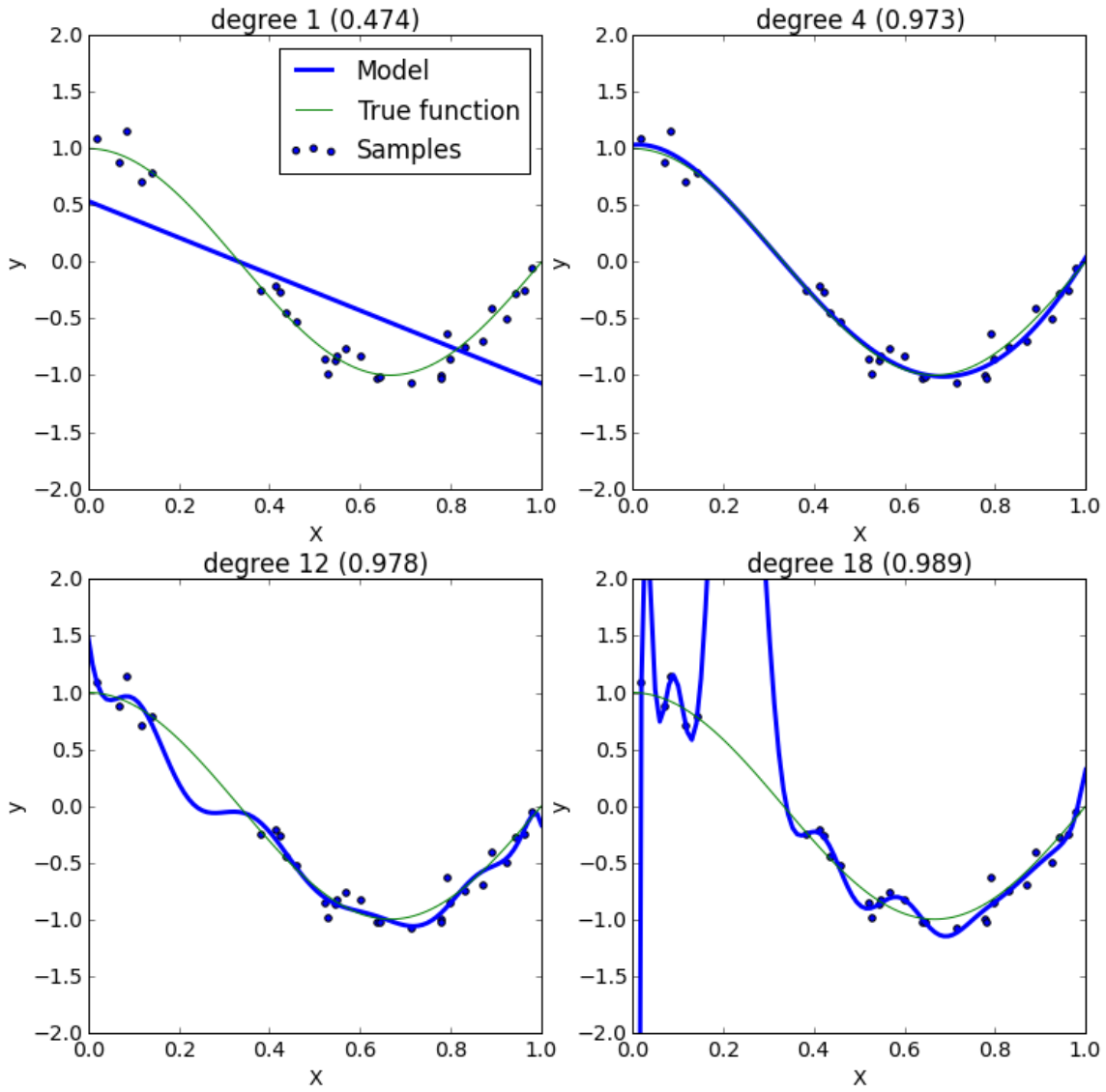


https://scikit-learn.org/stable/


degree 2
degree 3
degree 4
degree 5
degree 9
training points

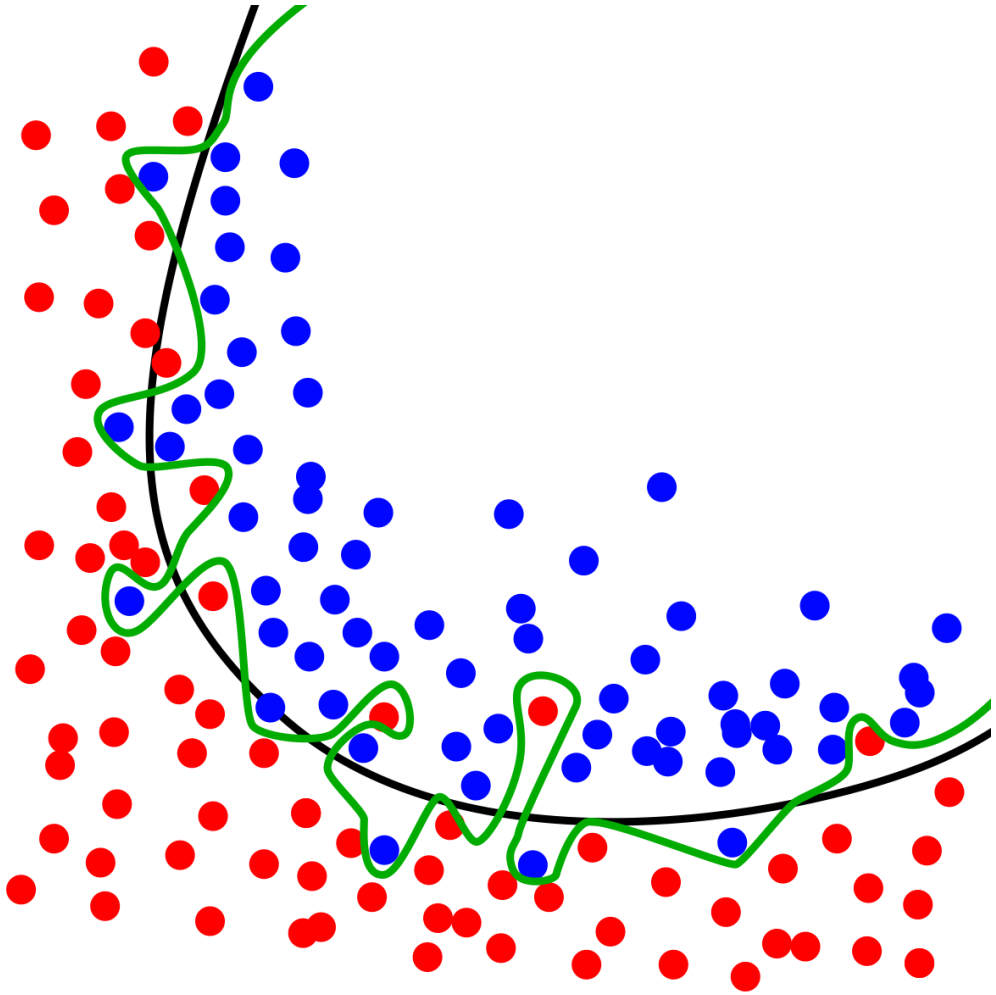

data set
X

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

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

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

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

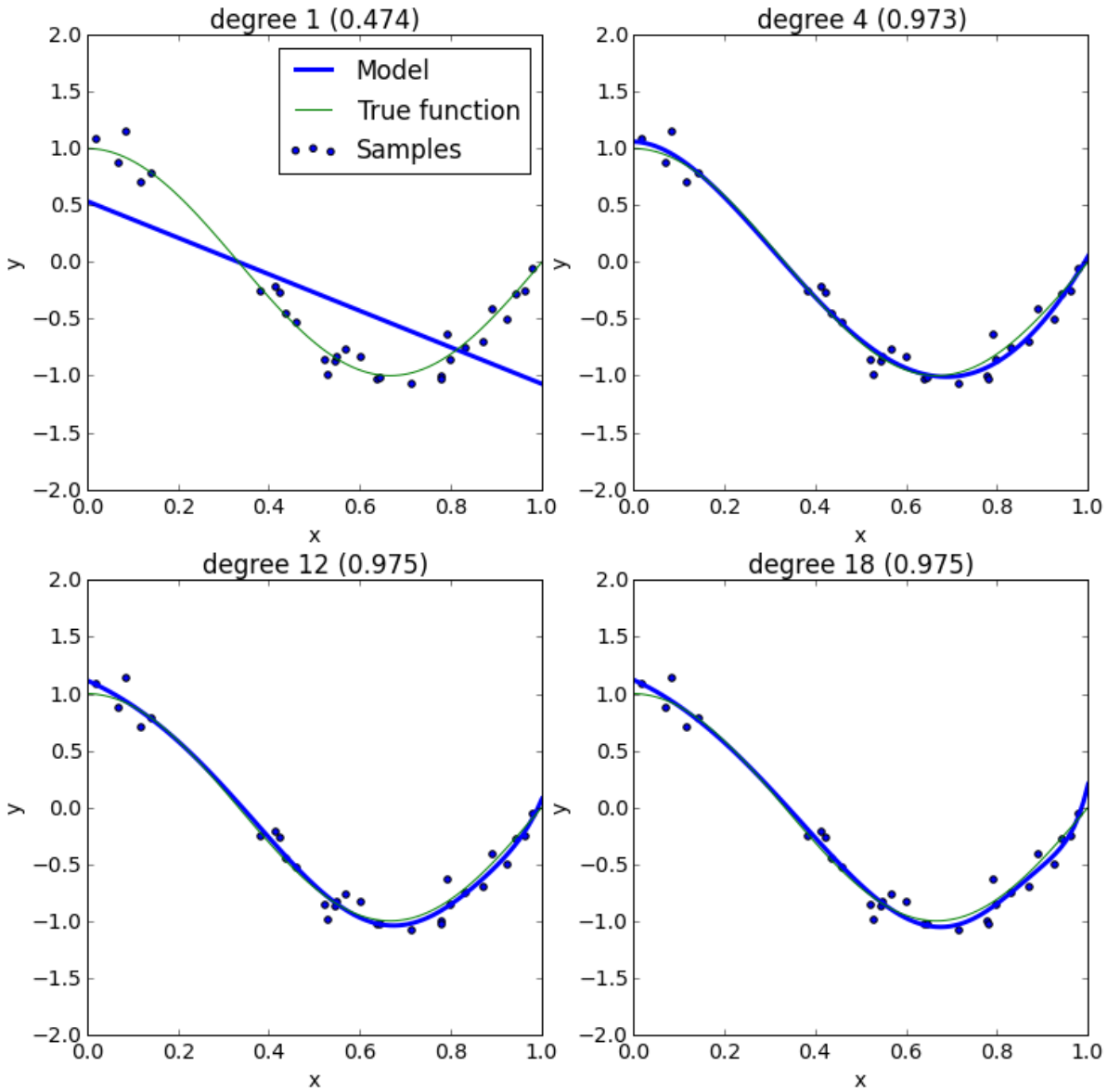
cost function

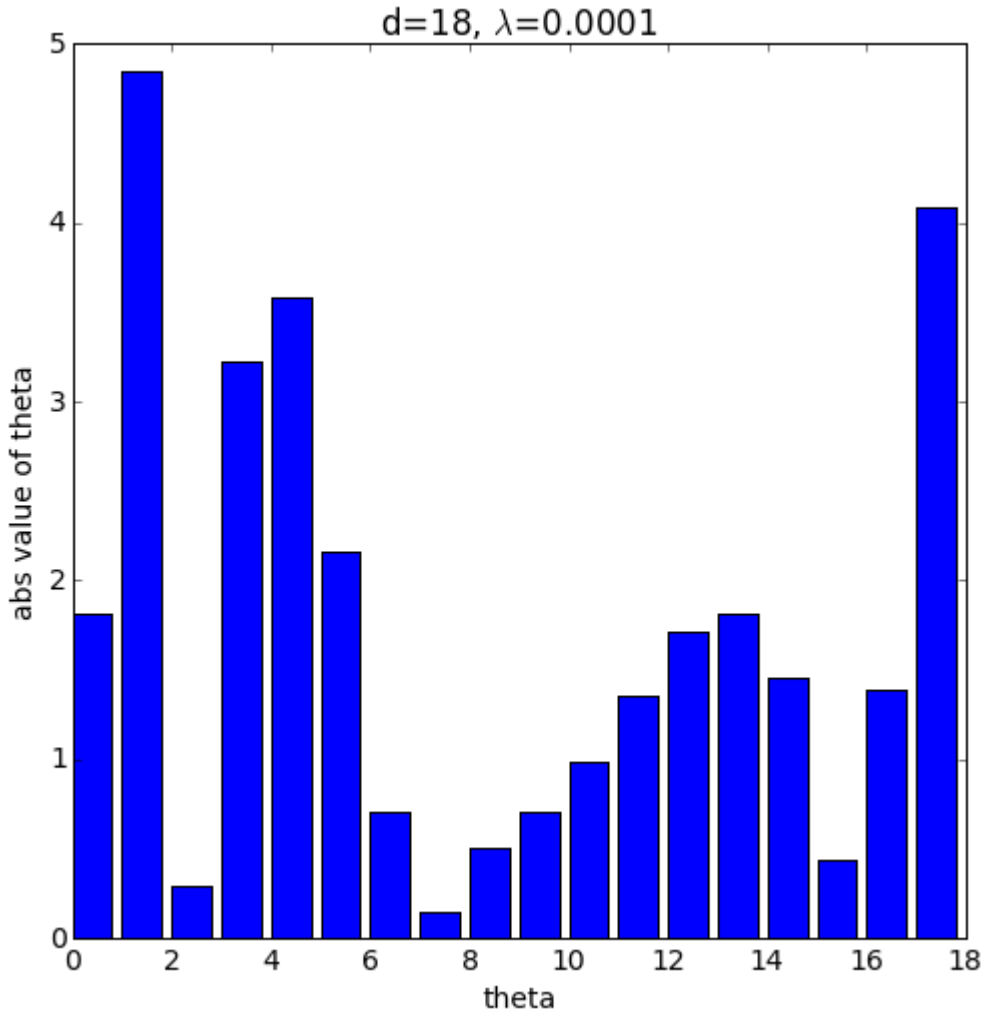
$$J(\theta) = -[\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(f(x^{(i)},\theta)) + (1 - y^{(i)}) log(1 - f(x^{(i)},\theta))]$$

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

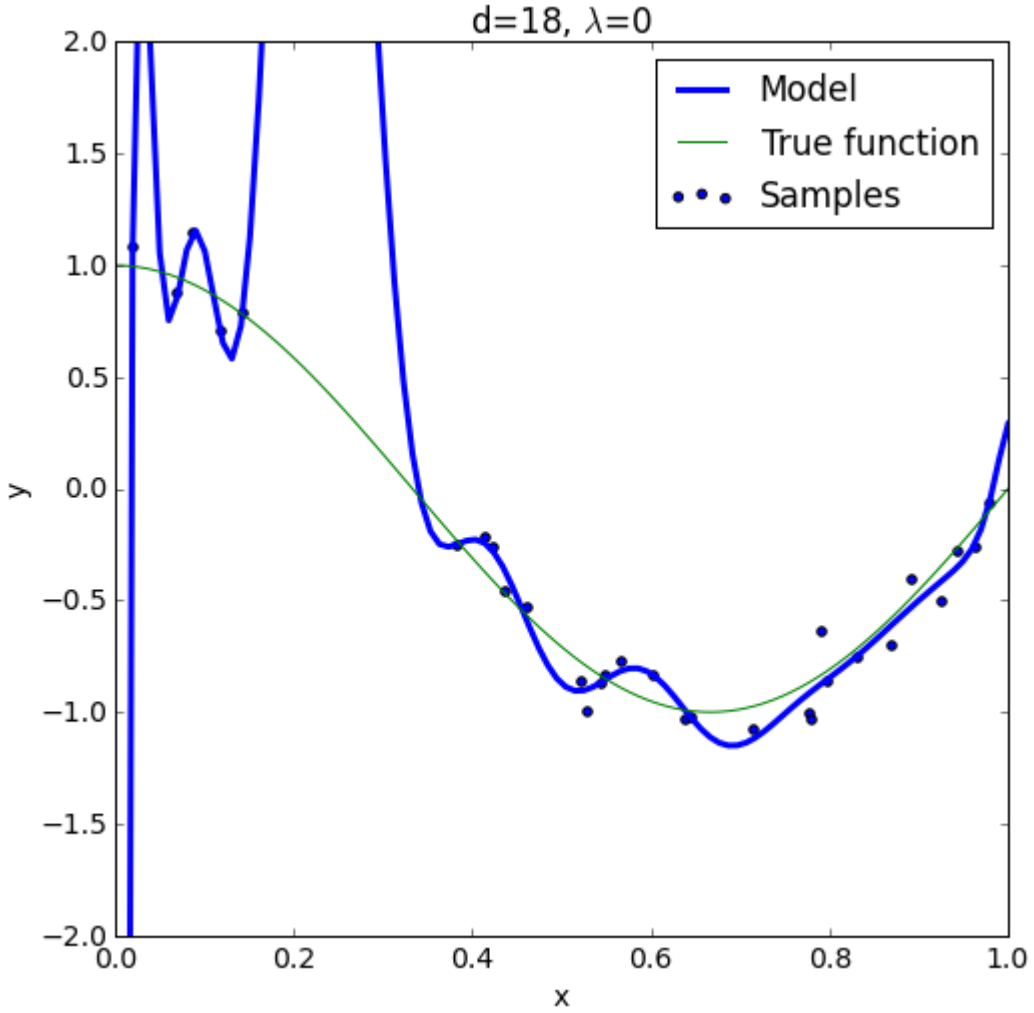
regularized cost function

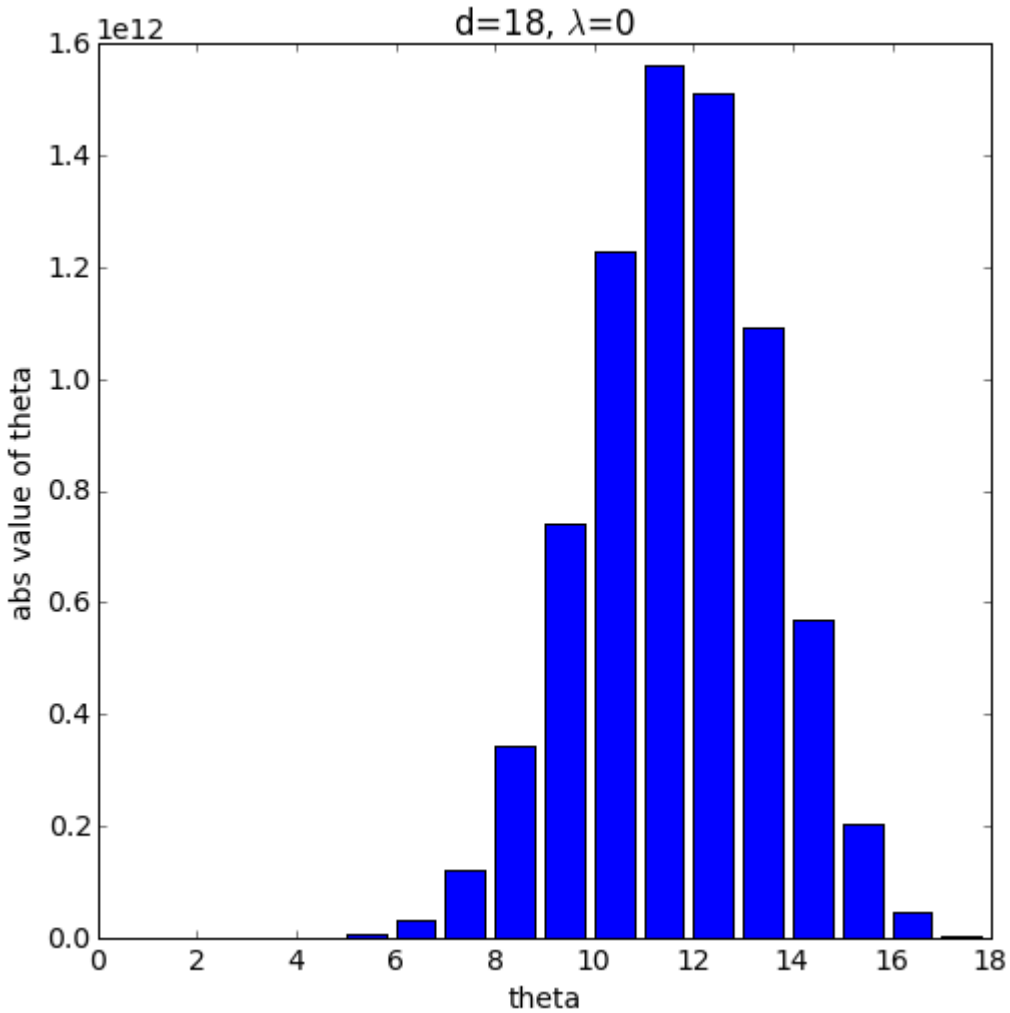
Regularized linear regression



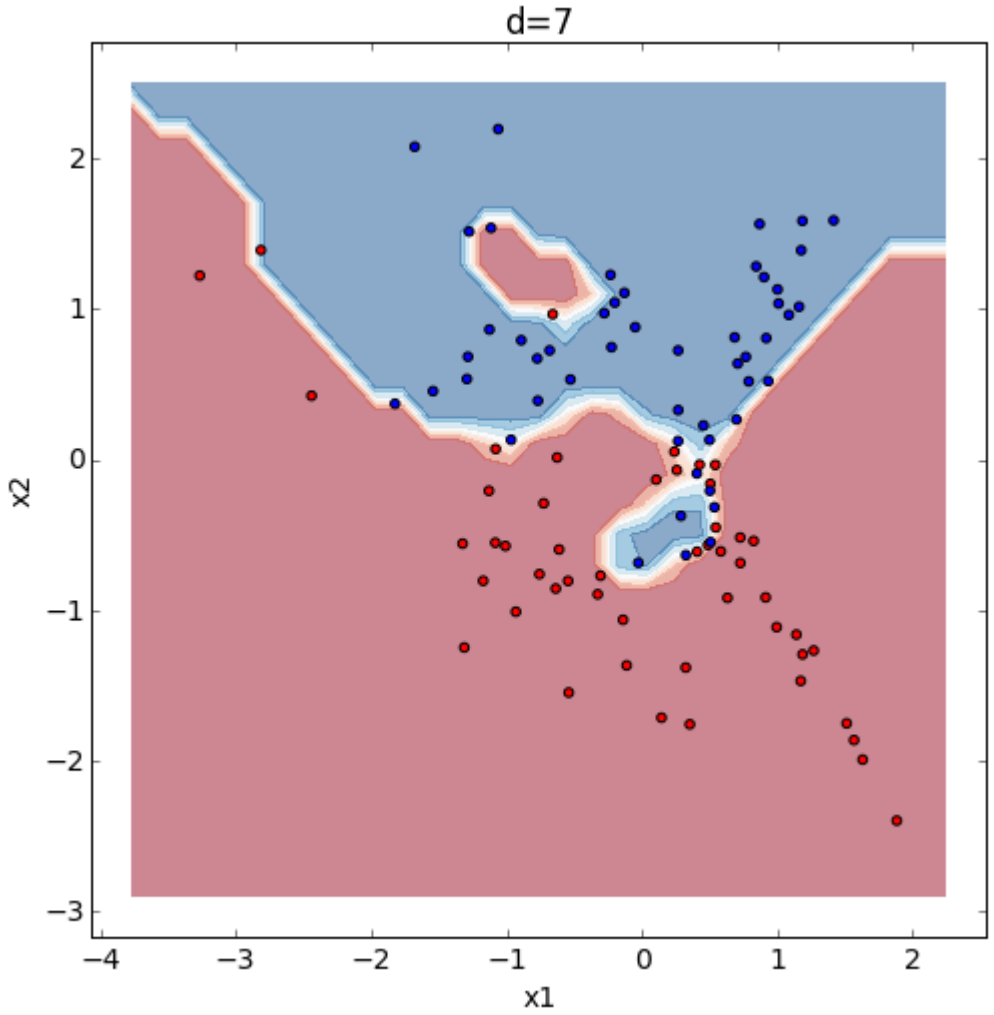


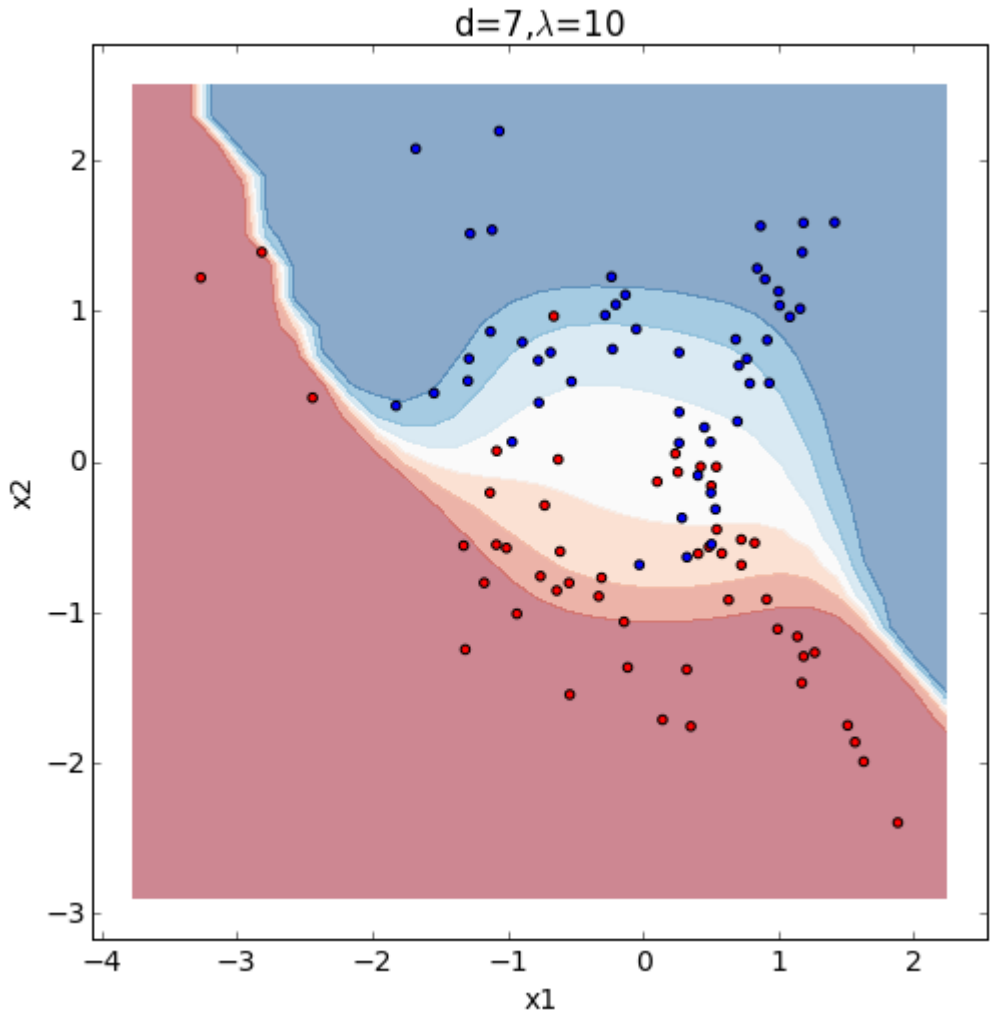
Regularized linear regression





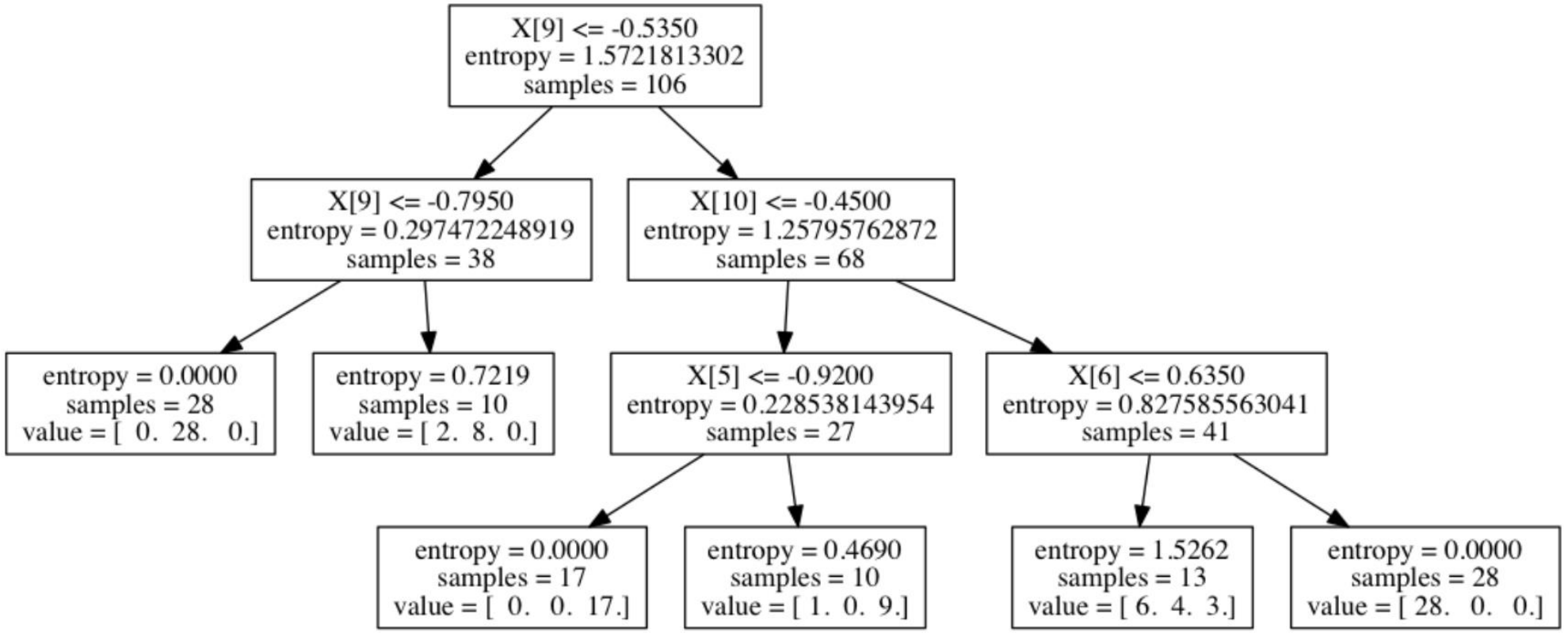
Regularized linear regression



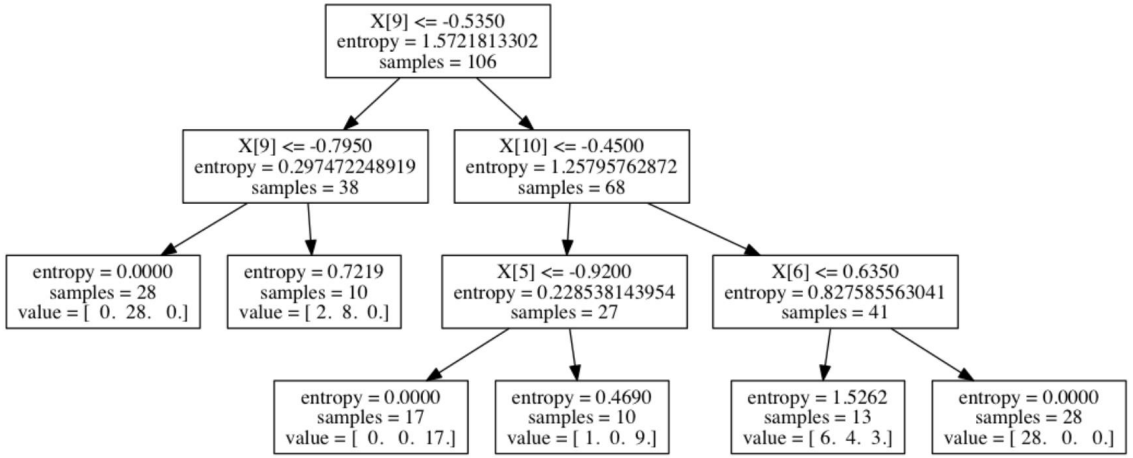


Decision trees

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



Decision trees



advantages:

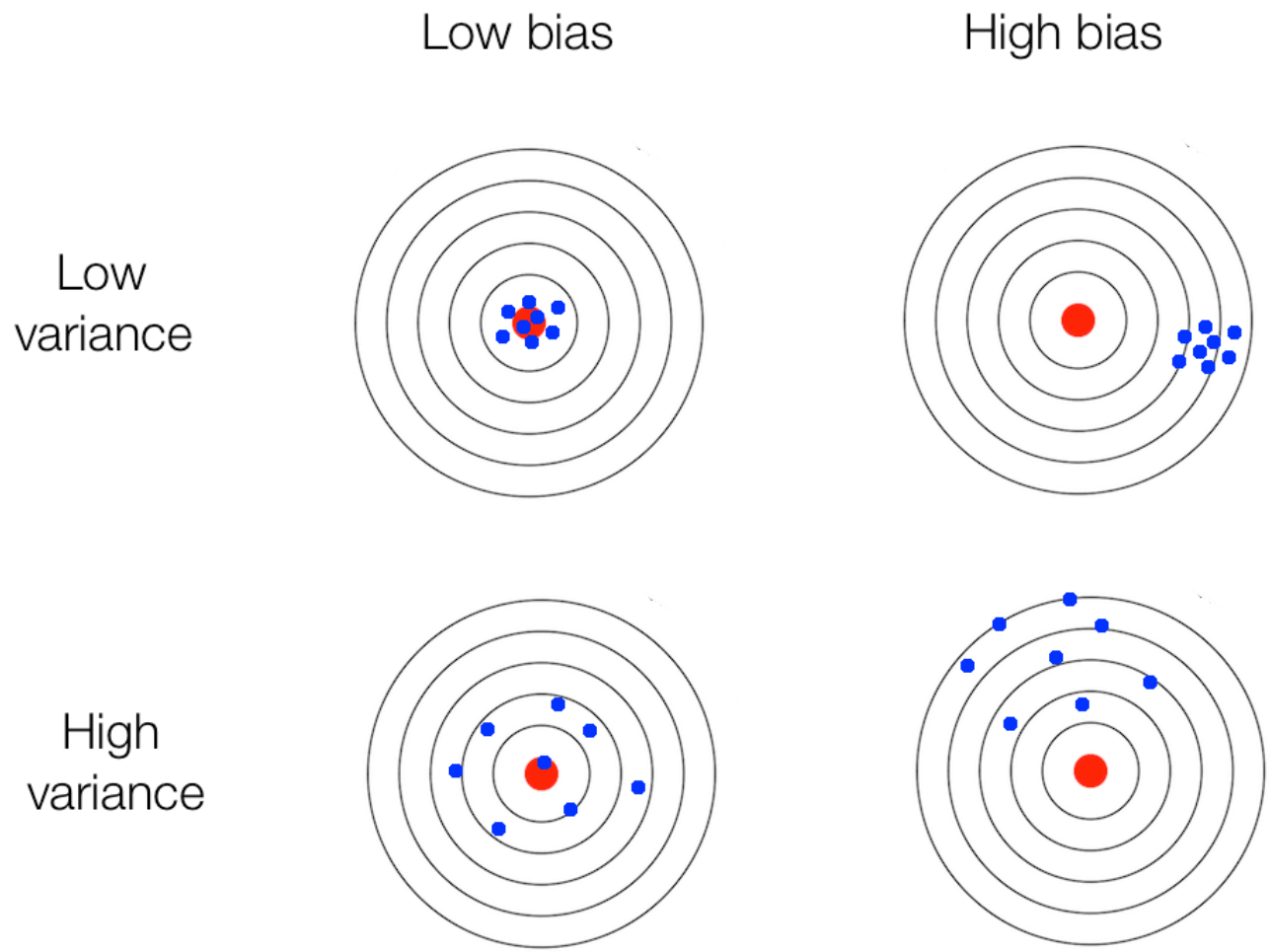
- ease of interpretation
- handles continuous and discrete features
- invariant to monotone transformation of features
- variable selection automated
- **low bias** (deep trees)

disadvantages:

- **high variance**
- overfitting

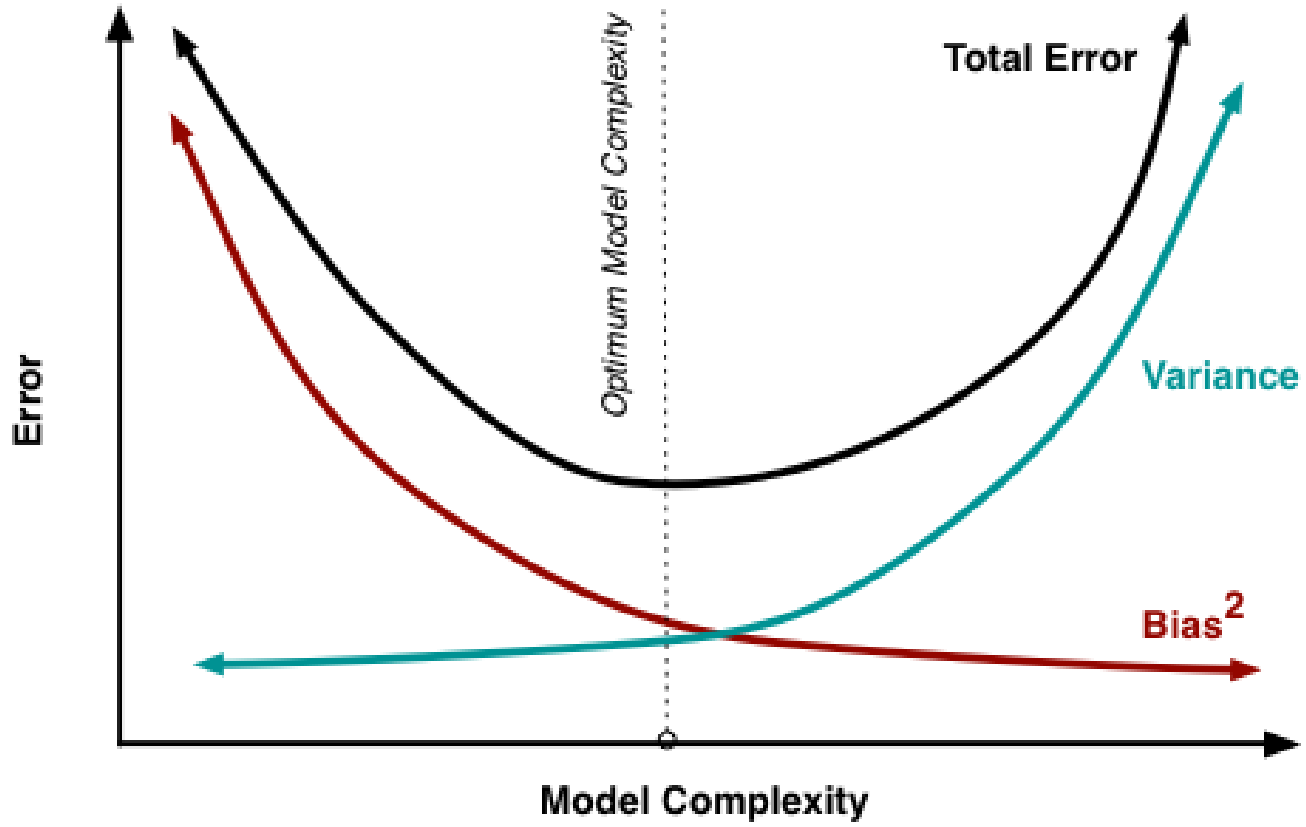
Bias / Variance

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

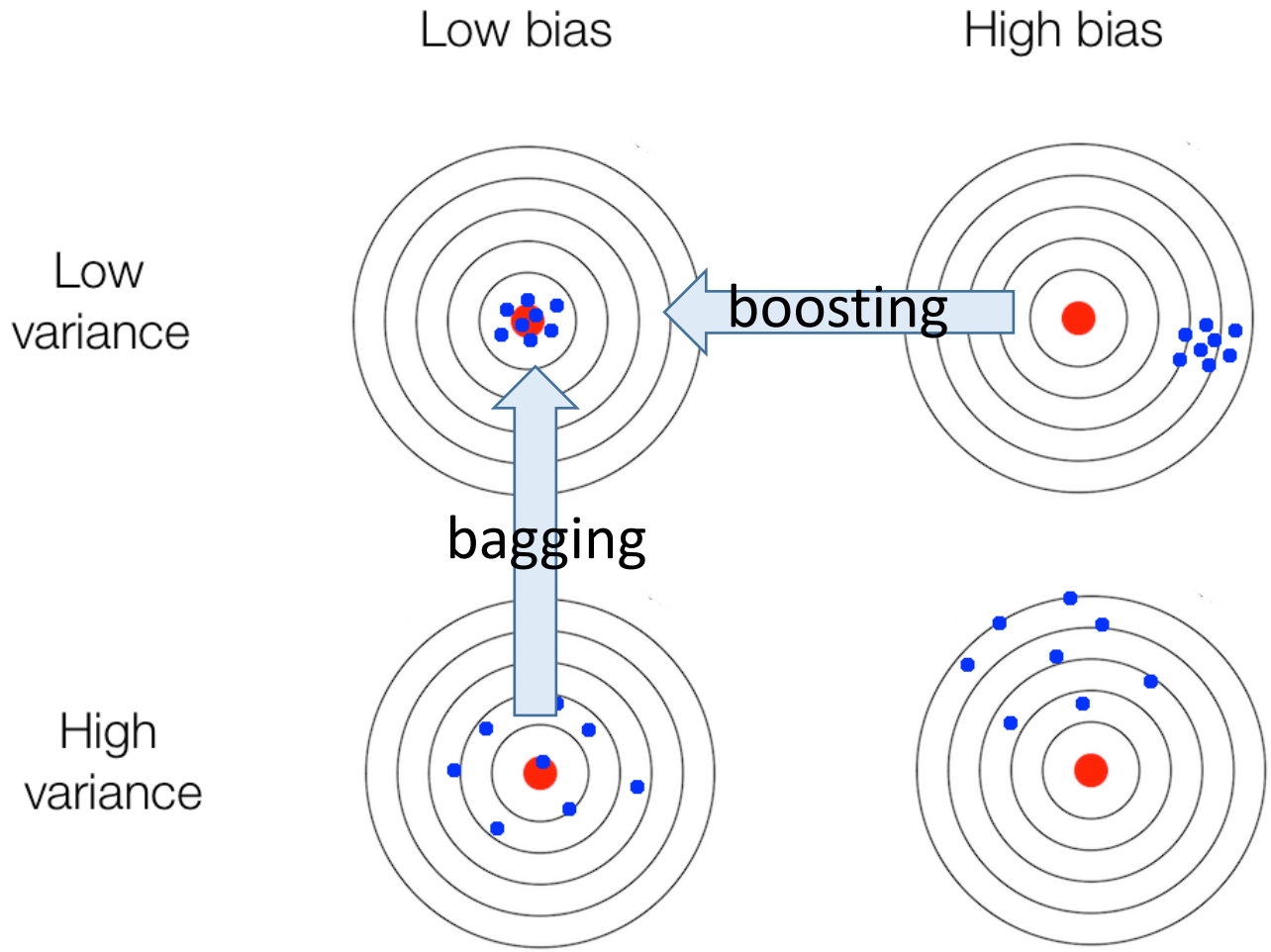


Bias / Variance

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



Bagging / Boosting



Bagging

- Simulate the notion of different train set samples:

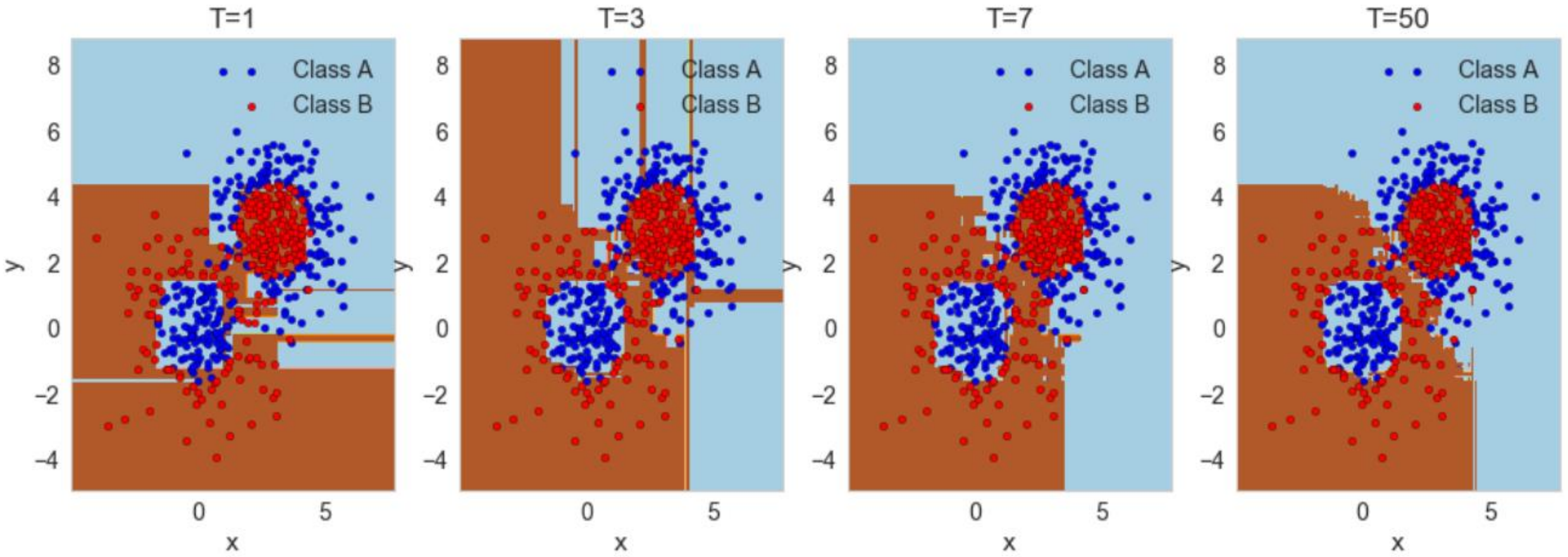
    1. sample data points from train set to create new train set
    2. fit low bias high variance model on new train set
    3. repeat steps (1) and (2) *T* times
    4. average the predictions of the *T* models

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

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 $h<<m$ 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 $D_1(i) = 1/n, i = 1, 2, \ldots, n.$

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

For $t = 1$ to T:

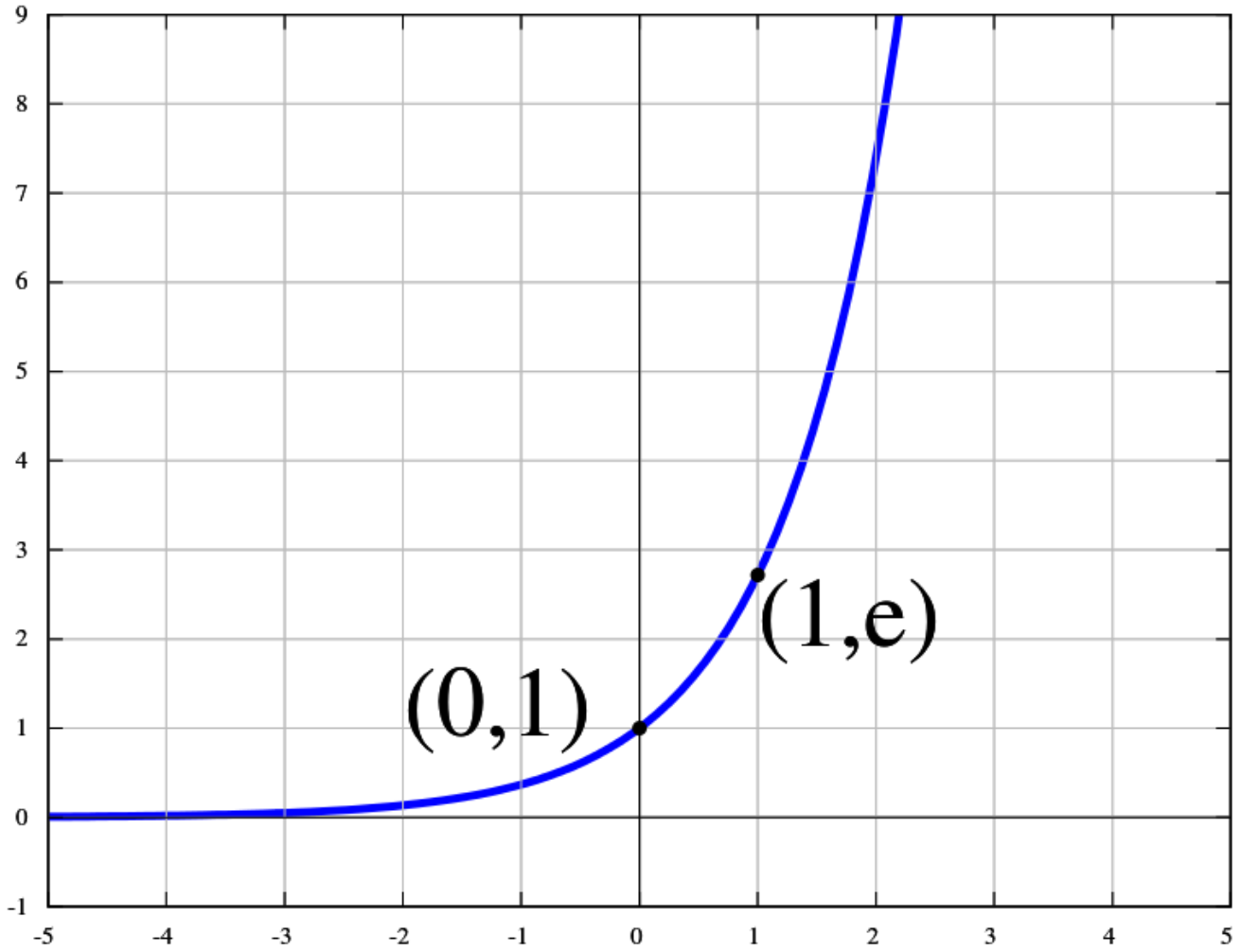
1. Fit a weak classifier $f_t(x, \theta)$ to the trainset data using weights $D_1(i)$.
2. Set $\alpha_t = \frac{1}{2} ln(\frac{1-error}{error})$.
3. Update weights:

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

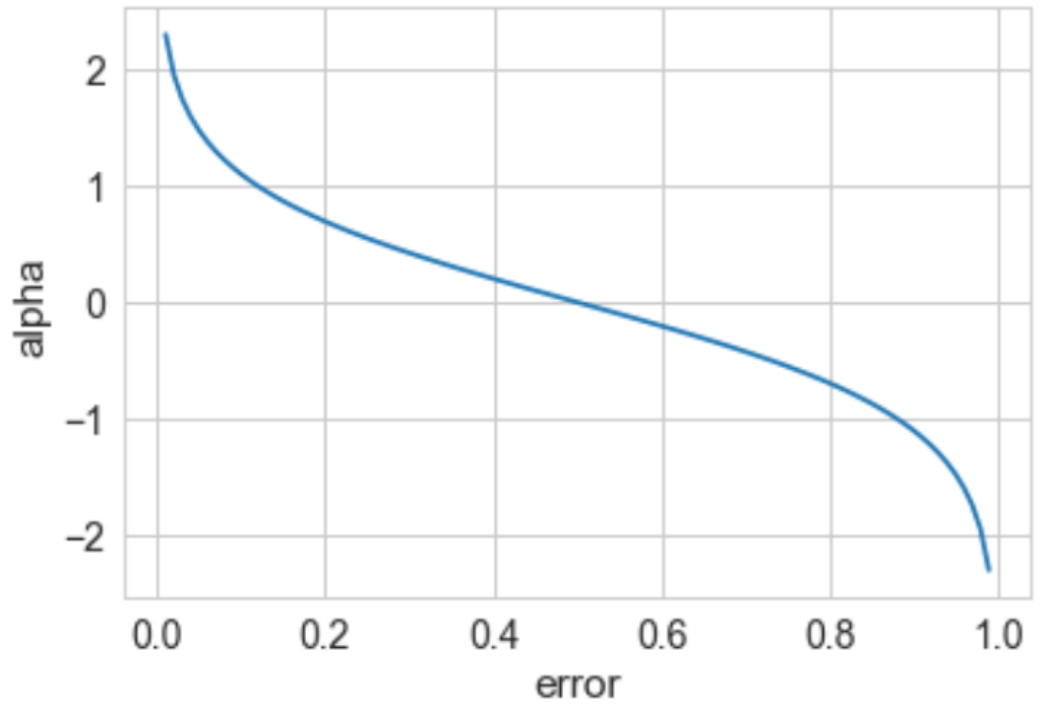
where Z_t is a normalizing factor that makes sure that $\sum D_{t+1}(i) = 1.$

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

Boosting: Adaboost



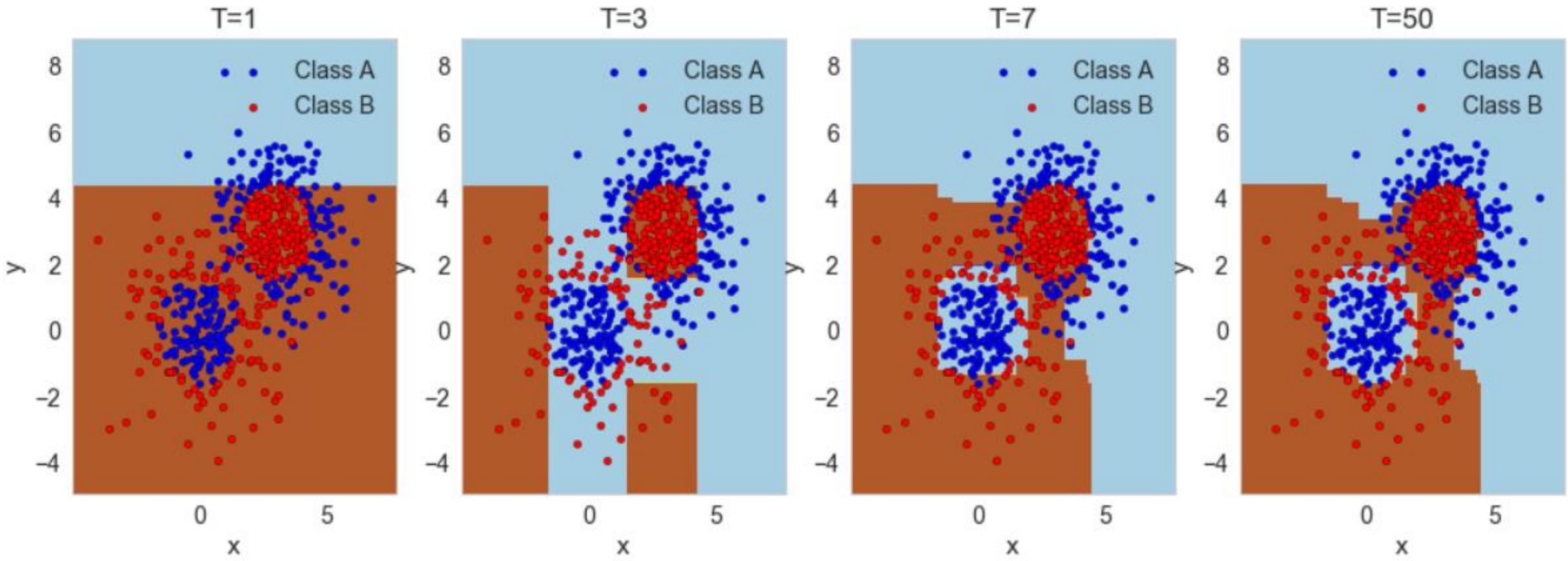
Boosting: Adaboost



- 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!”.

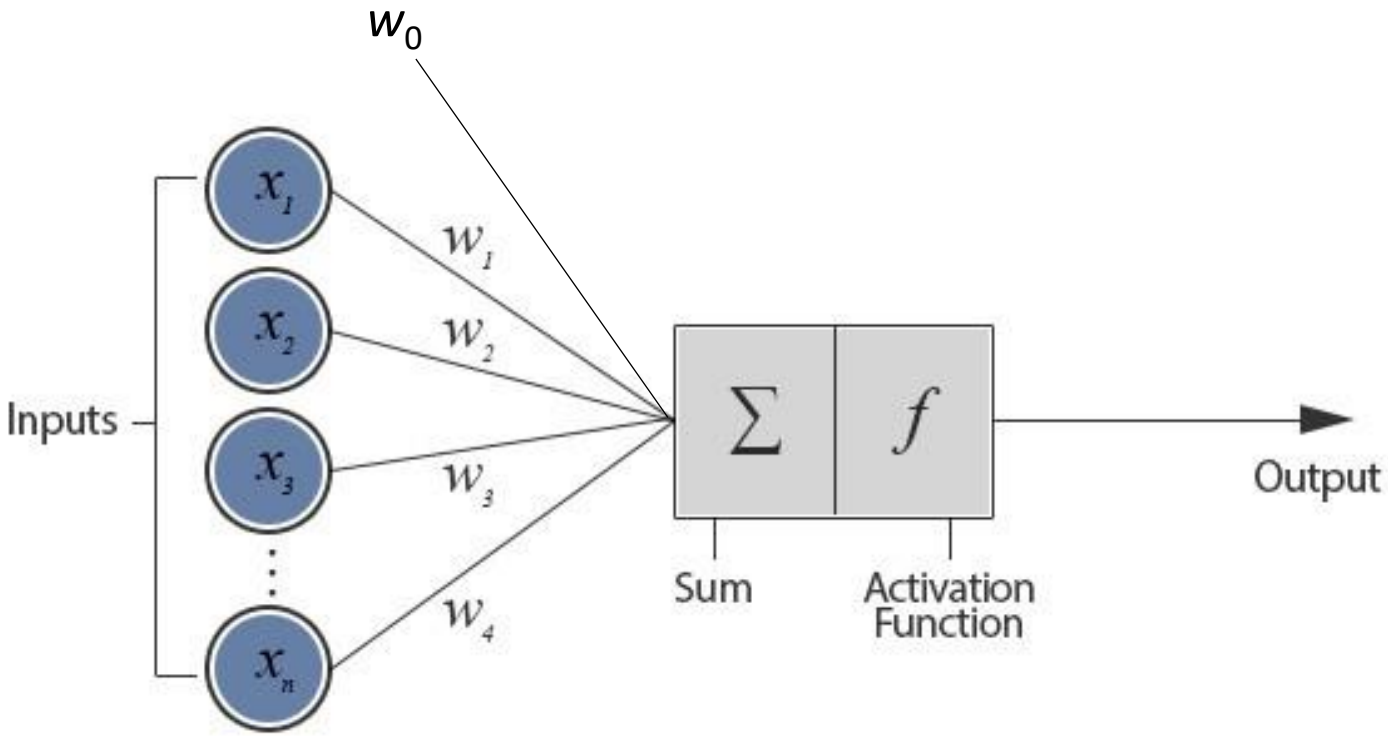
$$\hat{f}(x,\theta) = \sum_{t=1}^{T} \alpha_t f_t(x,\theta)$$

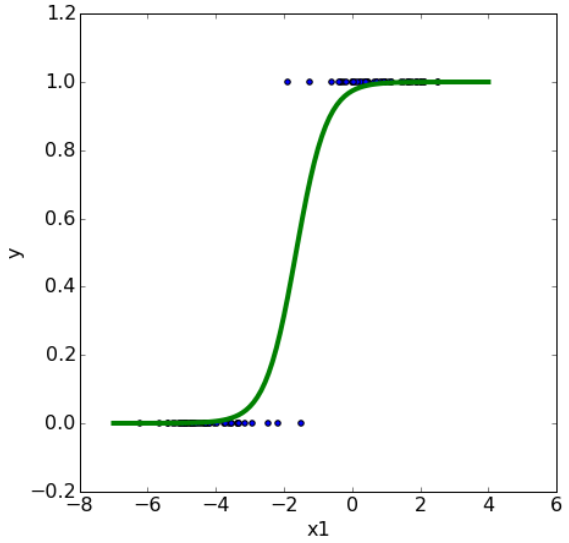
Boosting: Adaboost



Neural networks

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





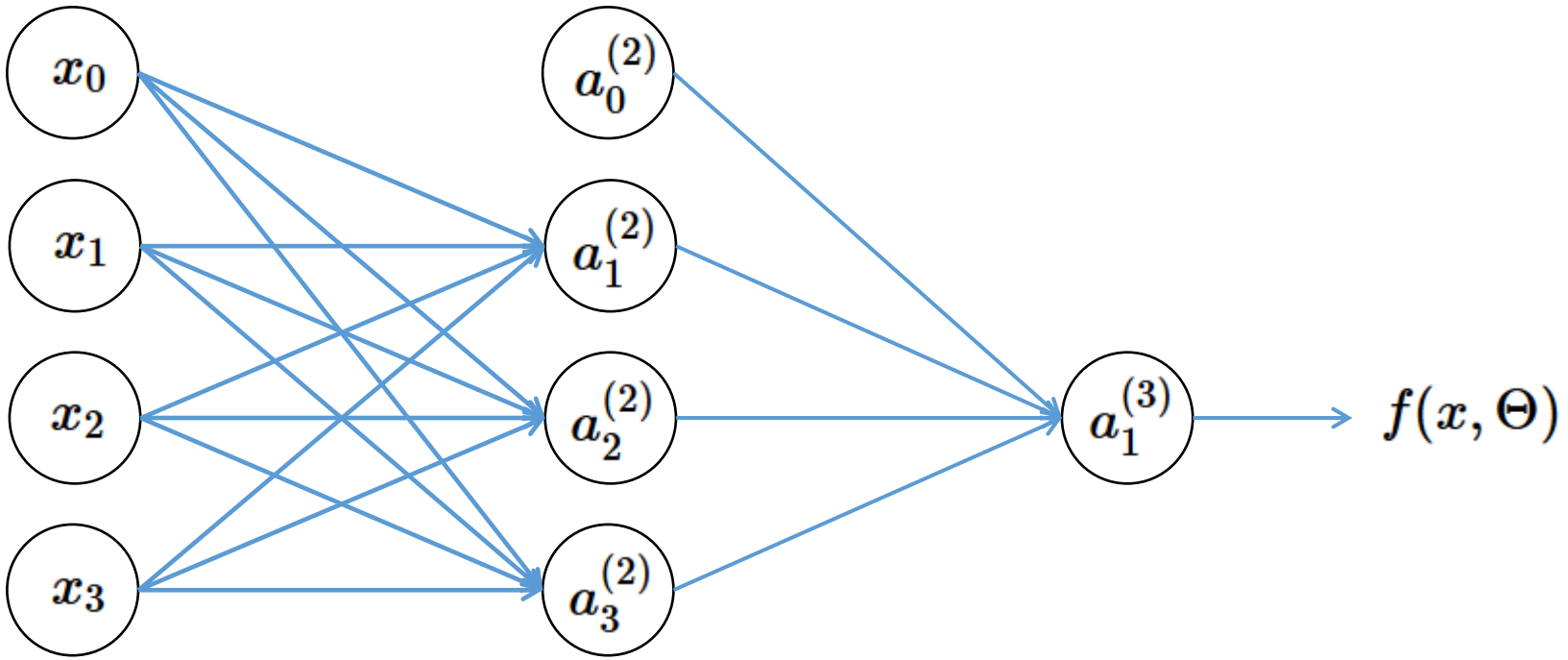
Neural networks: model

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

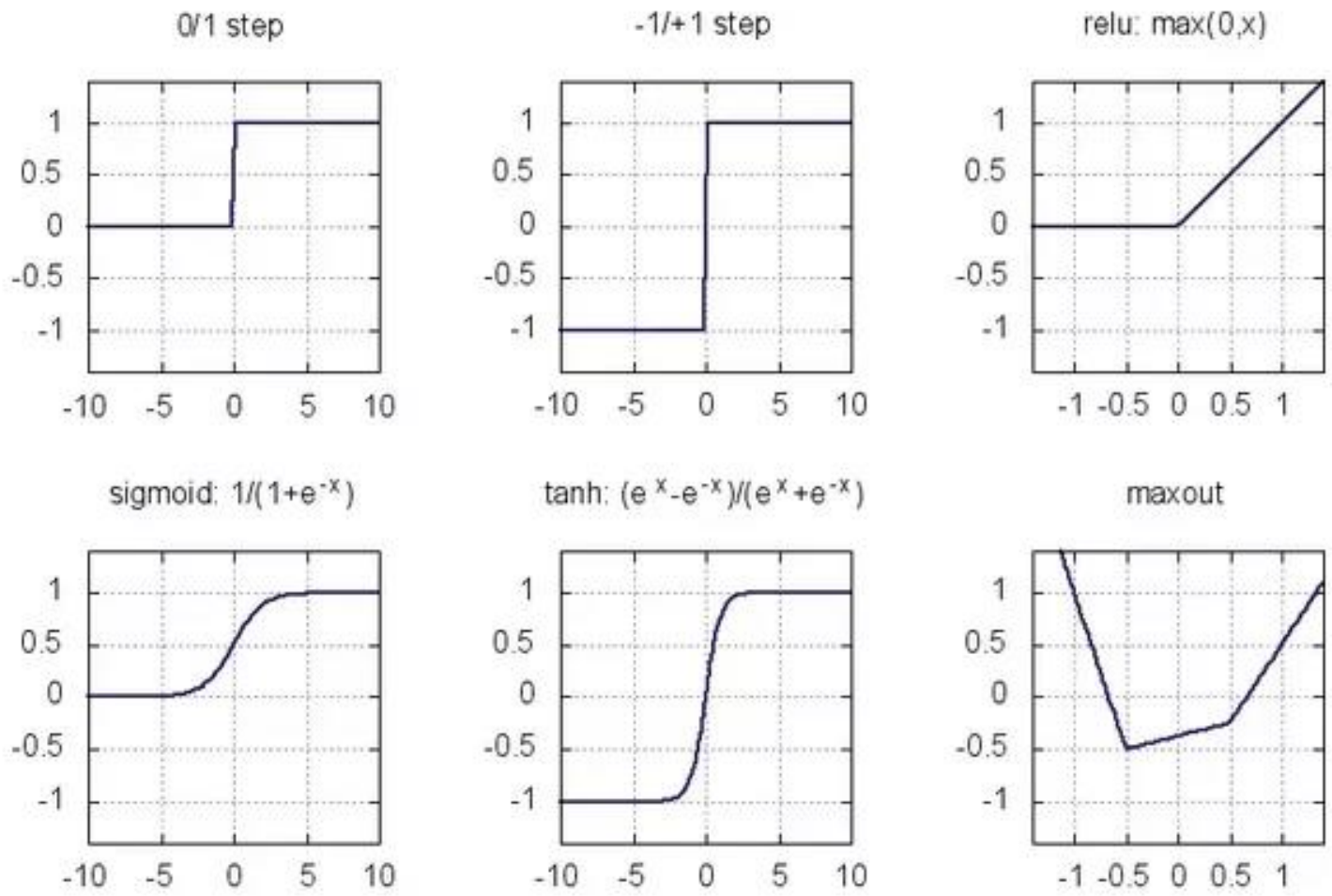
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

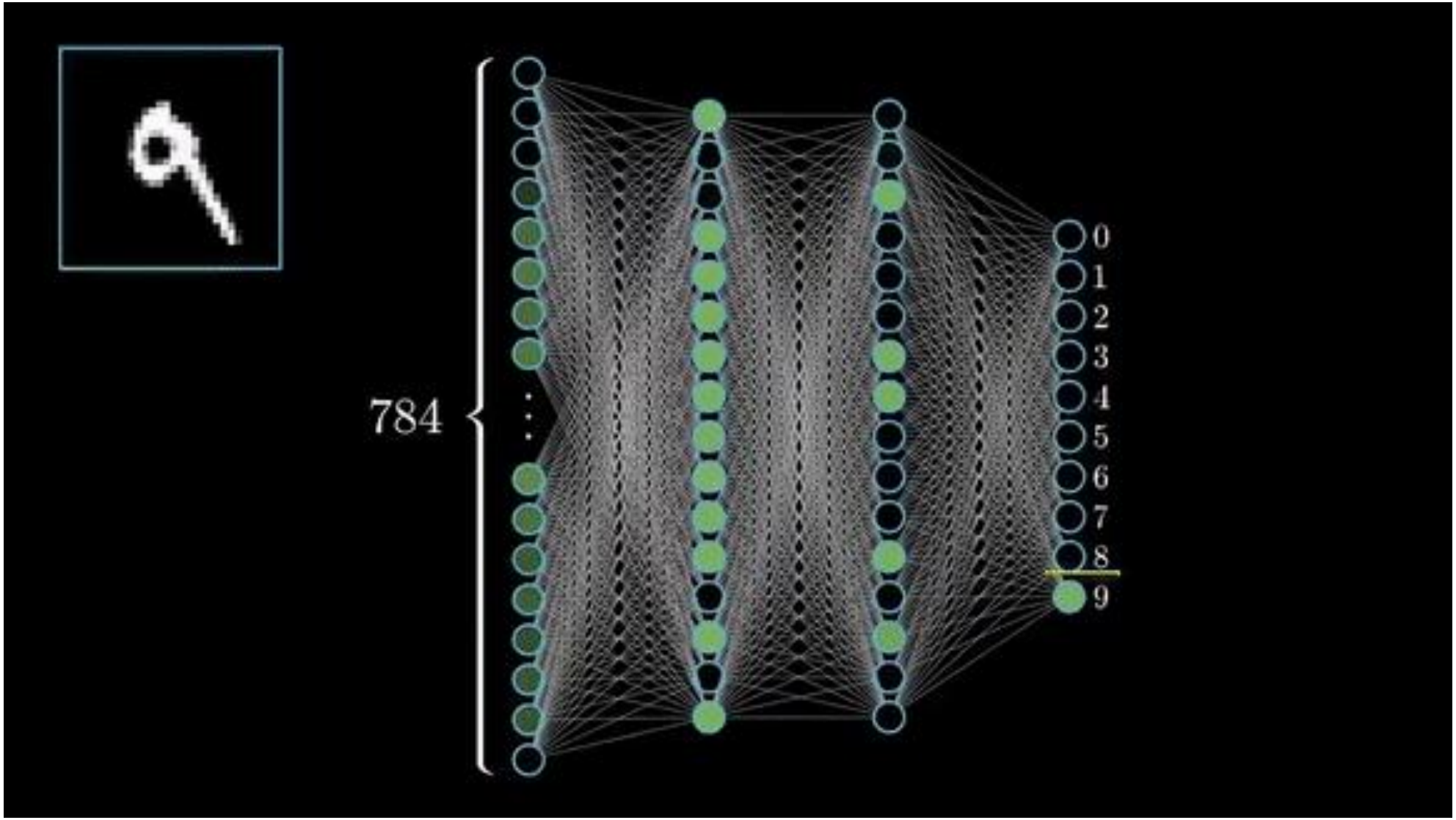
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

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



Neural networks: activation functions




784
0
1
2
3
4
5
6
7
8
9

Deep convolutional neural networks

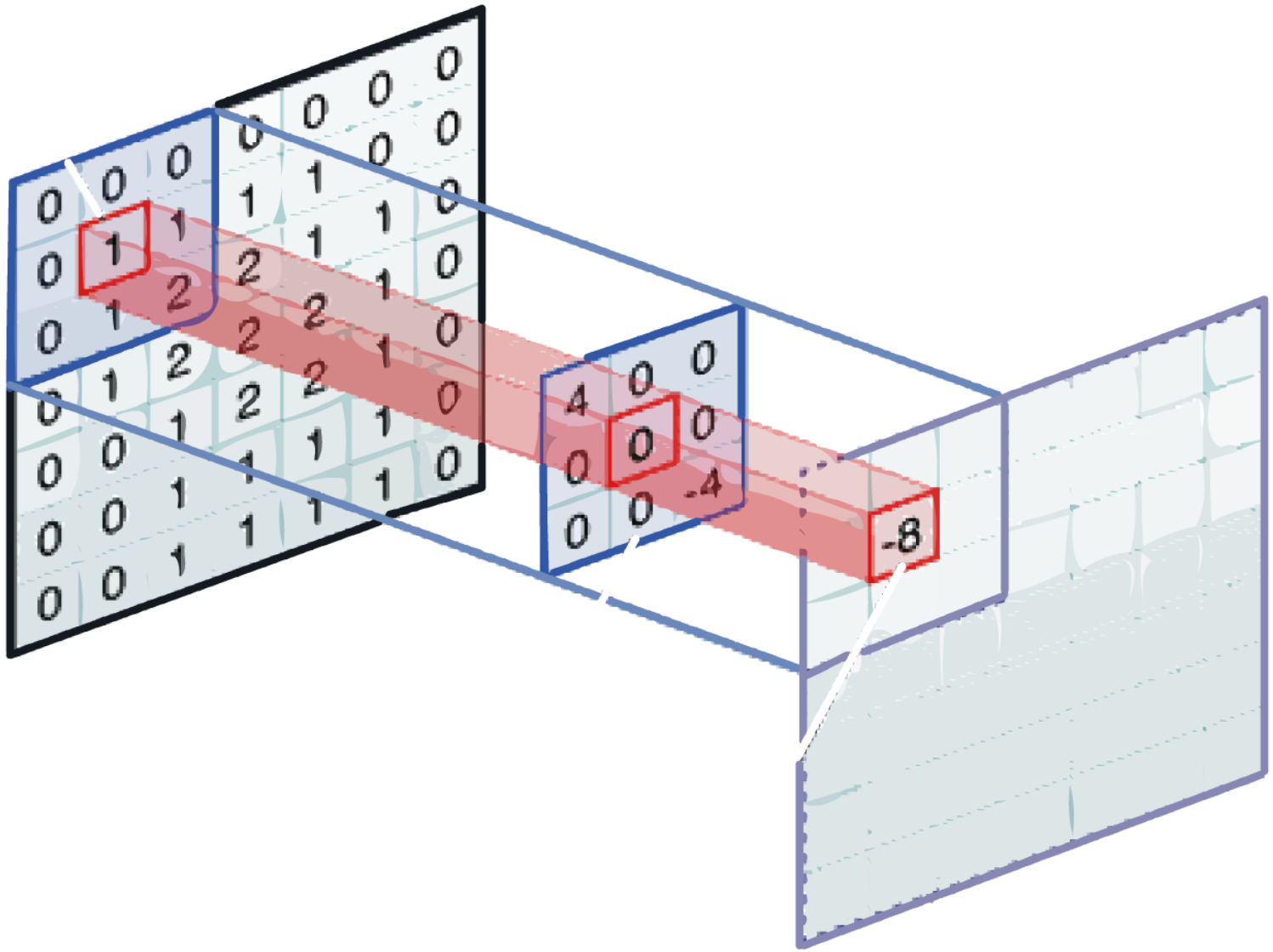
What We See

```
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 44 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 46 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

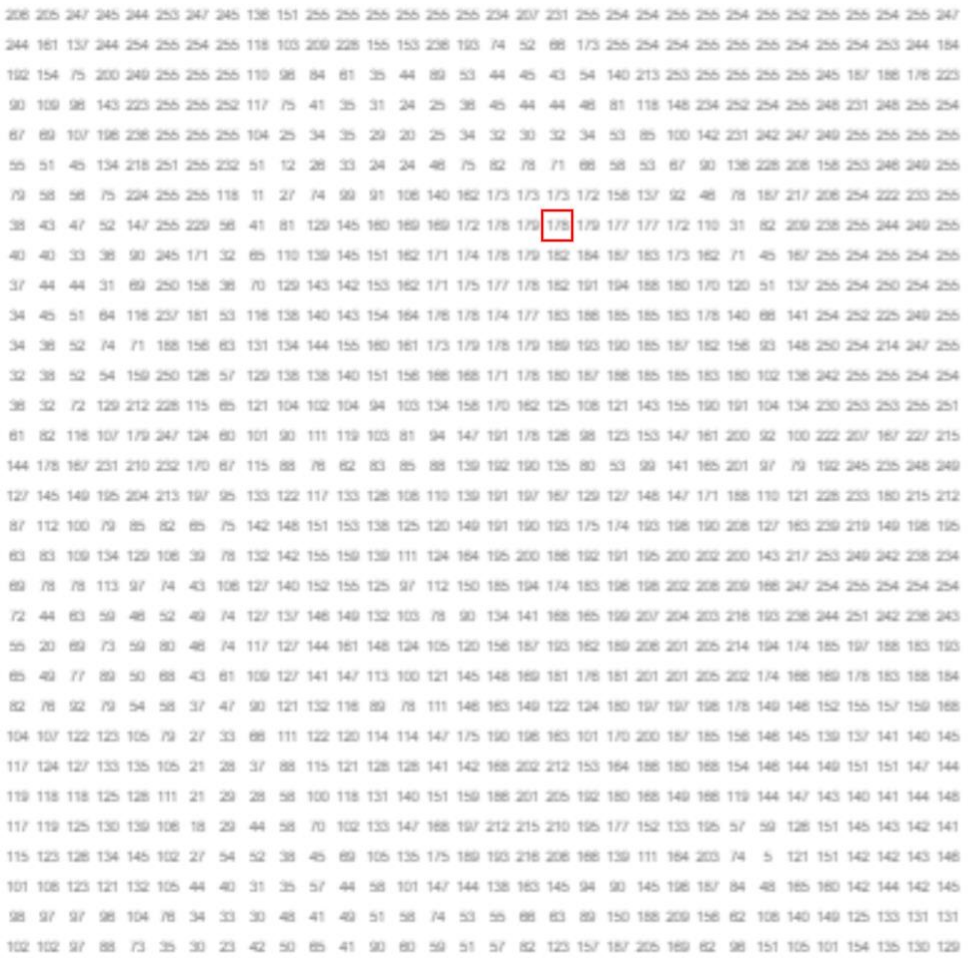
https://github.com/leriomaggio

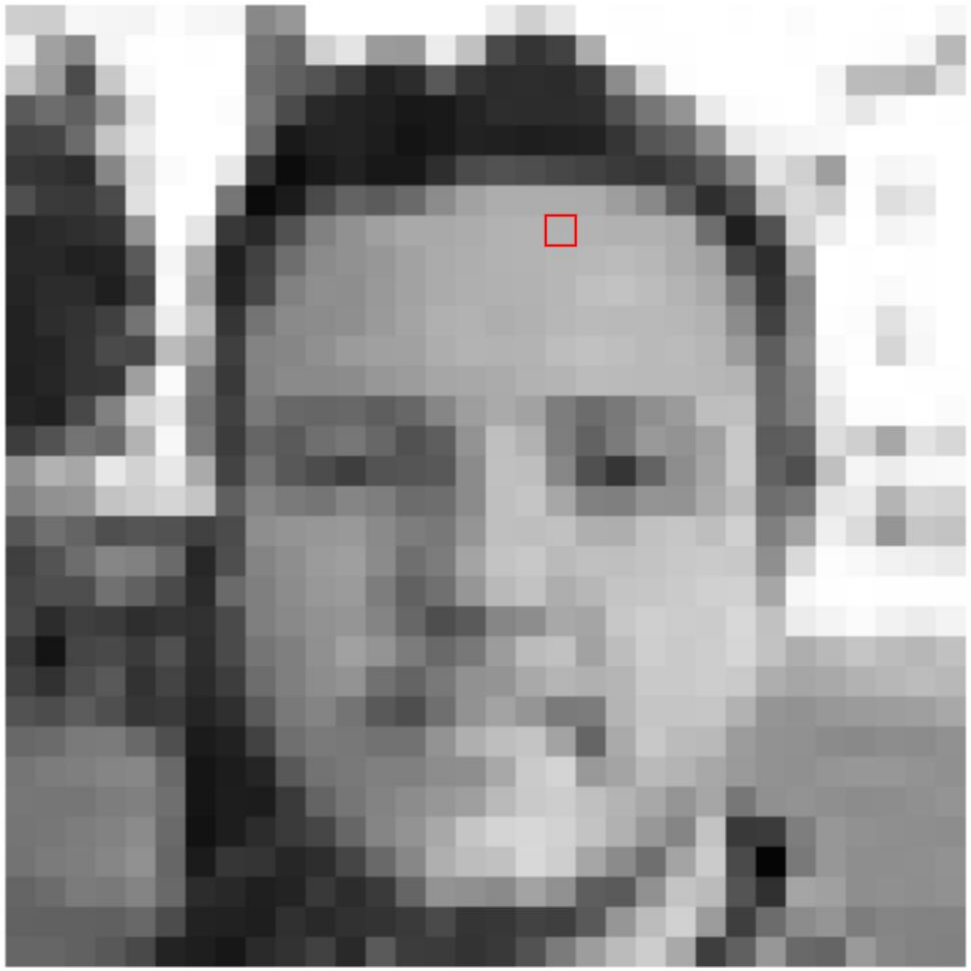
Filters



https://github.com/leriomaggio

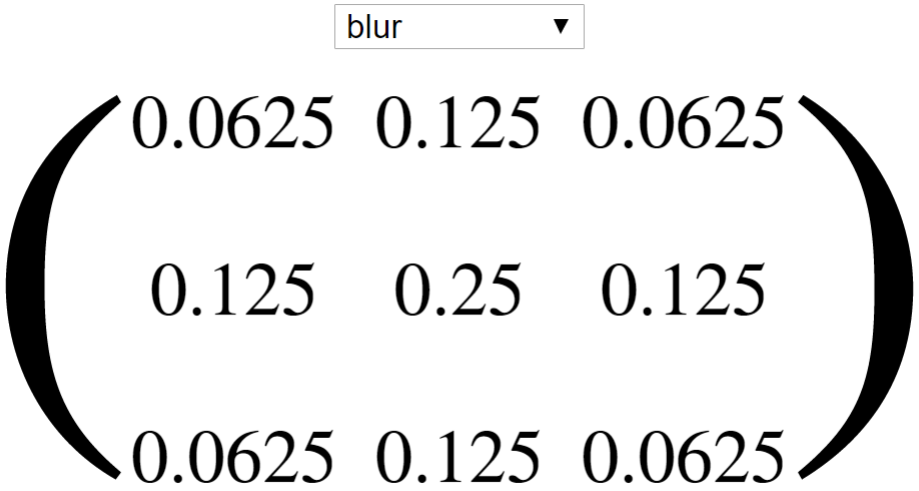
Filters





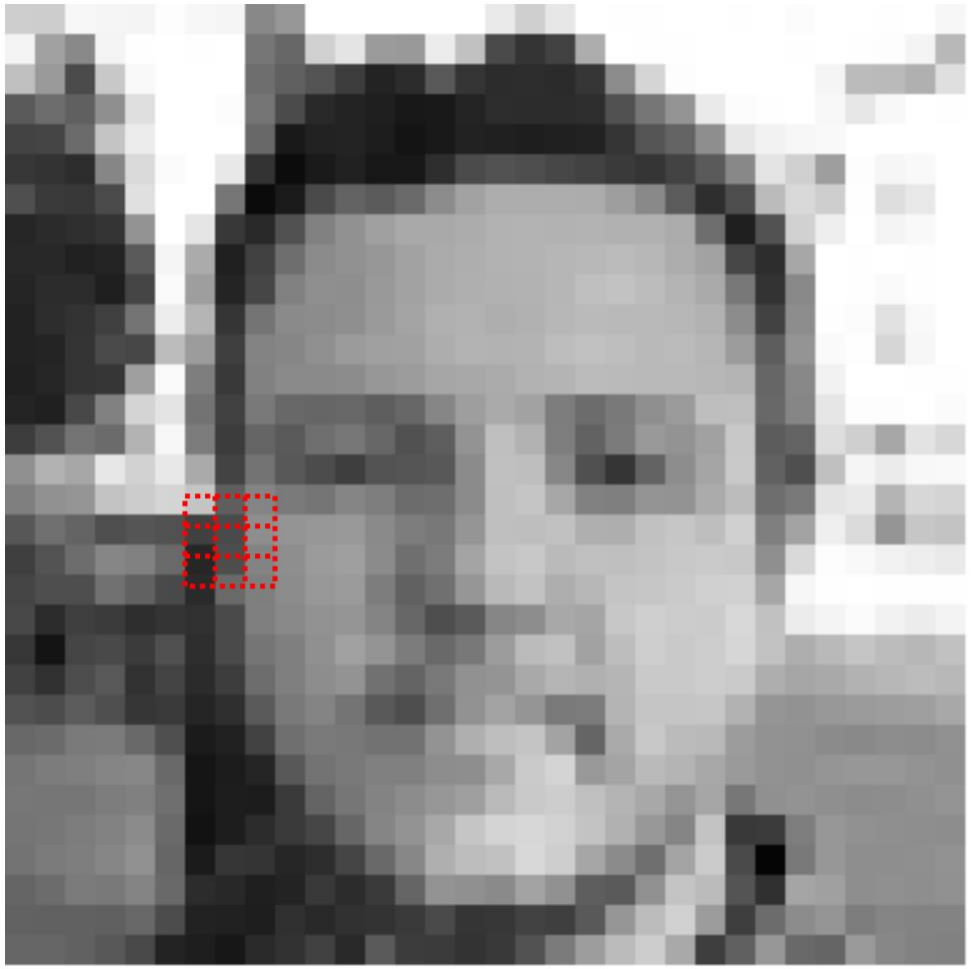
http://setosa.io/ev/image-kernels/

Filters



http://setosa.io/ev/image-kernels/

Filters

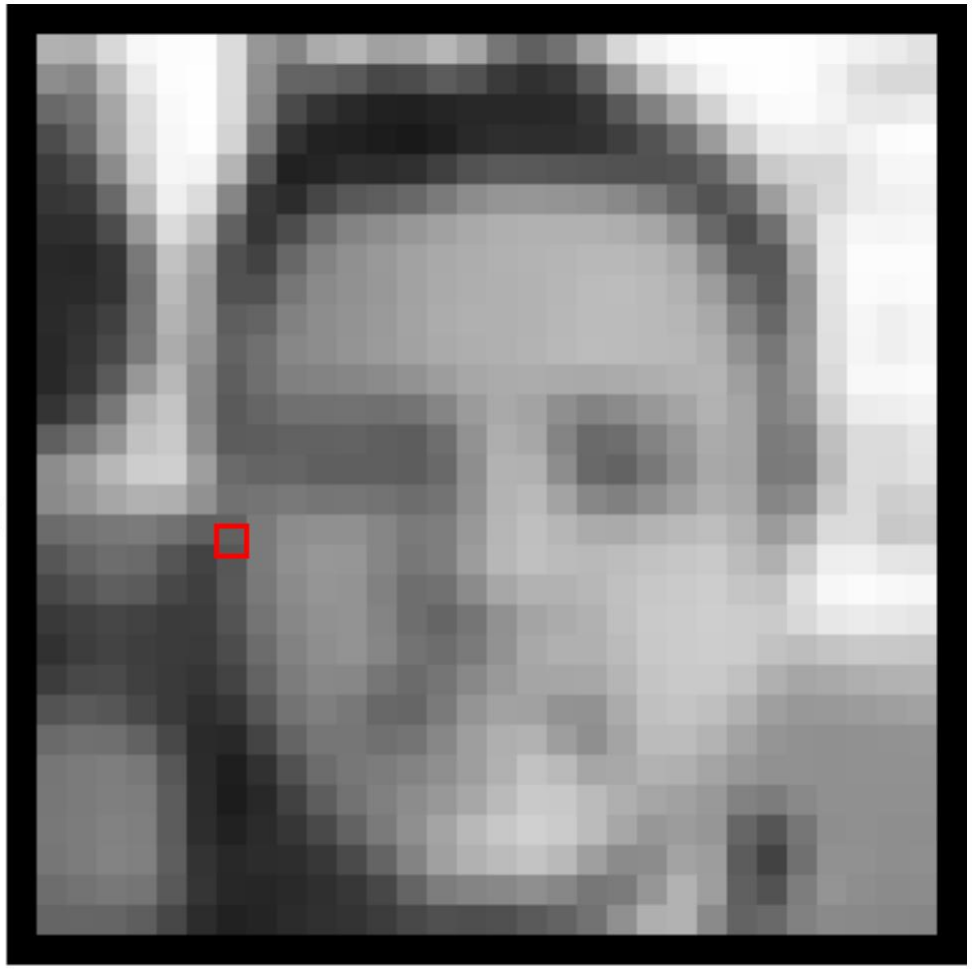


input image

$$\Big(57 \times 0.0625 + 129 \times 0.125 + 138 \times 0.0625 + 65 \times 0.125 + 121 \times 0.25 + 104 \times 0.125 + 60 \times 0.0625 + 101 \times 0.125 + 90 \times 0.0625\Big) = 102$$

kernel:

blur ▼



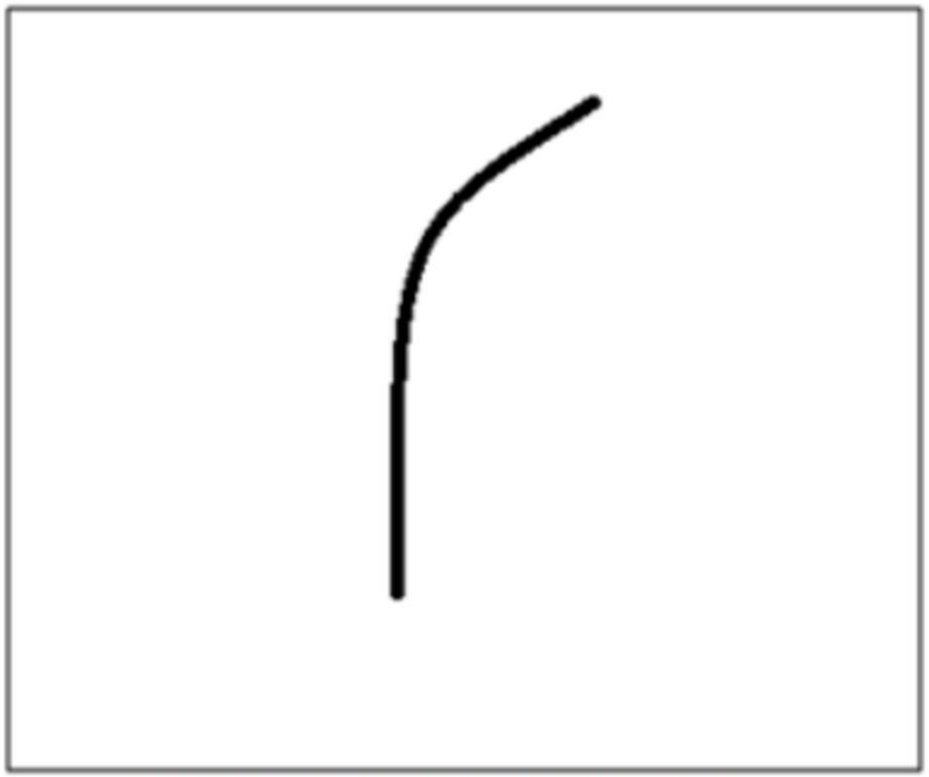
output image

http://setosa.io/ev/image-kernels/

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

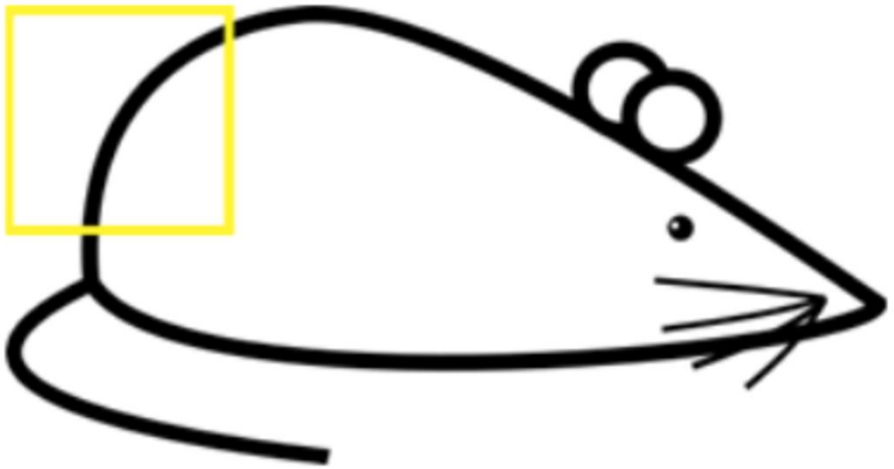


Visualization of a curve detector filter

https://github.com/leriomaggio

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	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

https://github.com/leriomaggio

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

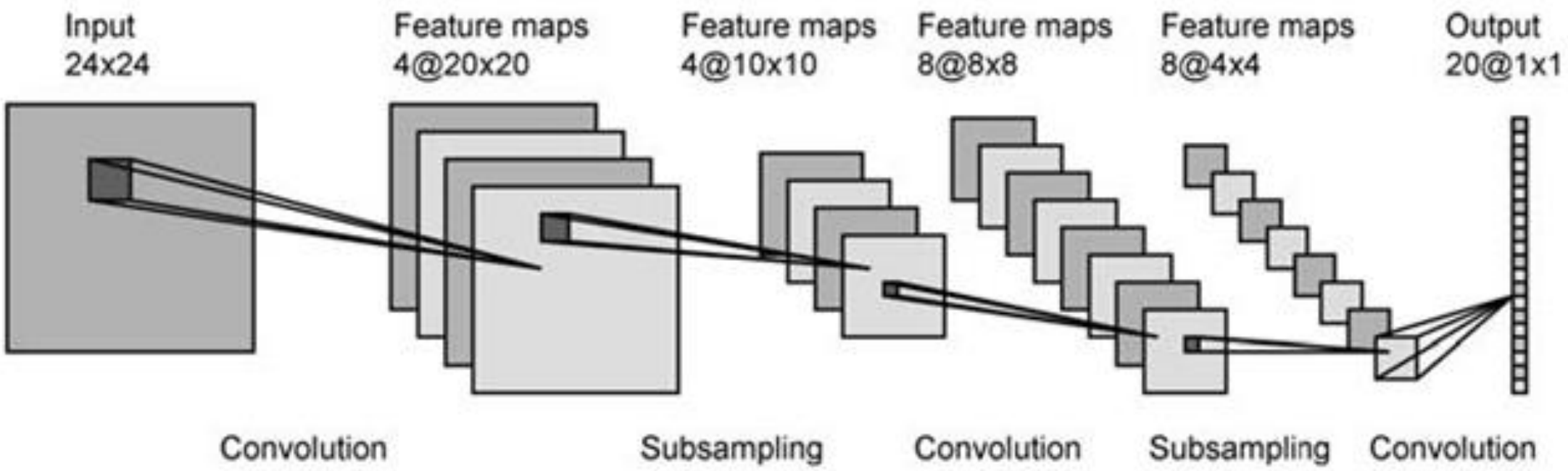
*

0	0	0	0	0	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

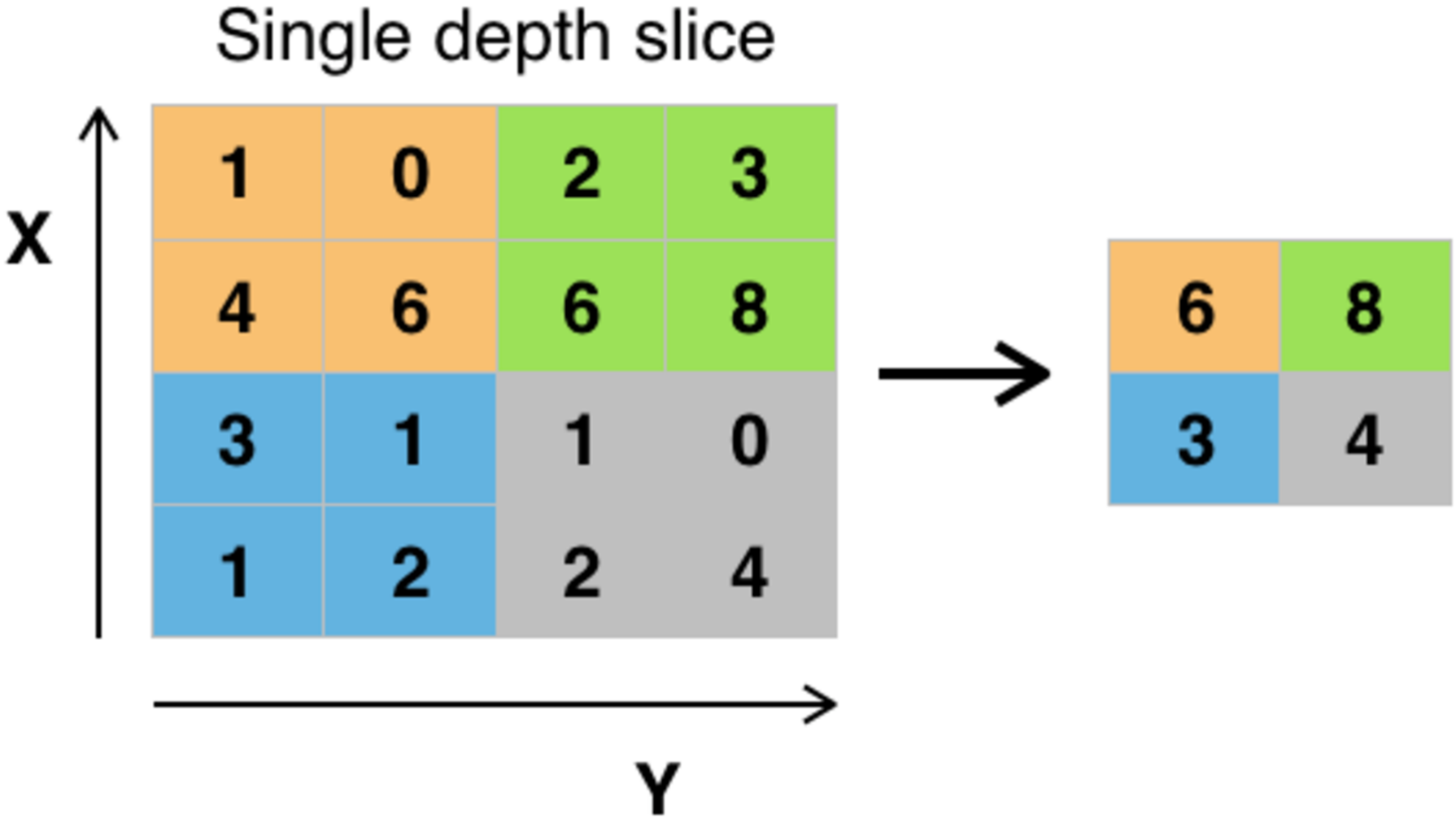
Pixel representation of filter

https://github.com/leriomaggio

Deep convolutional neural networks



Max pooling

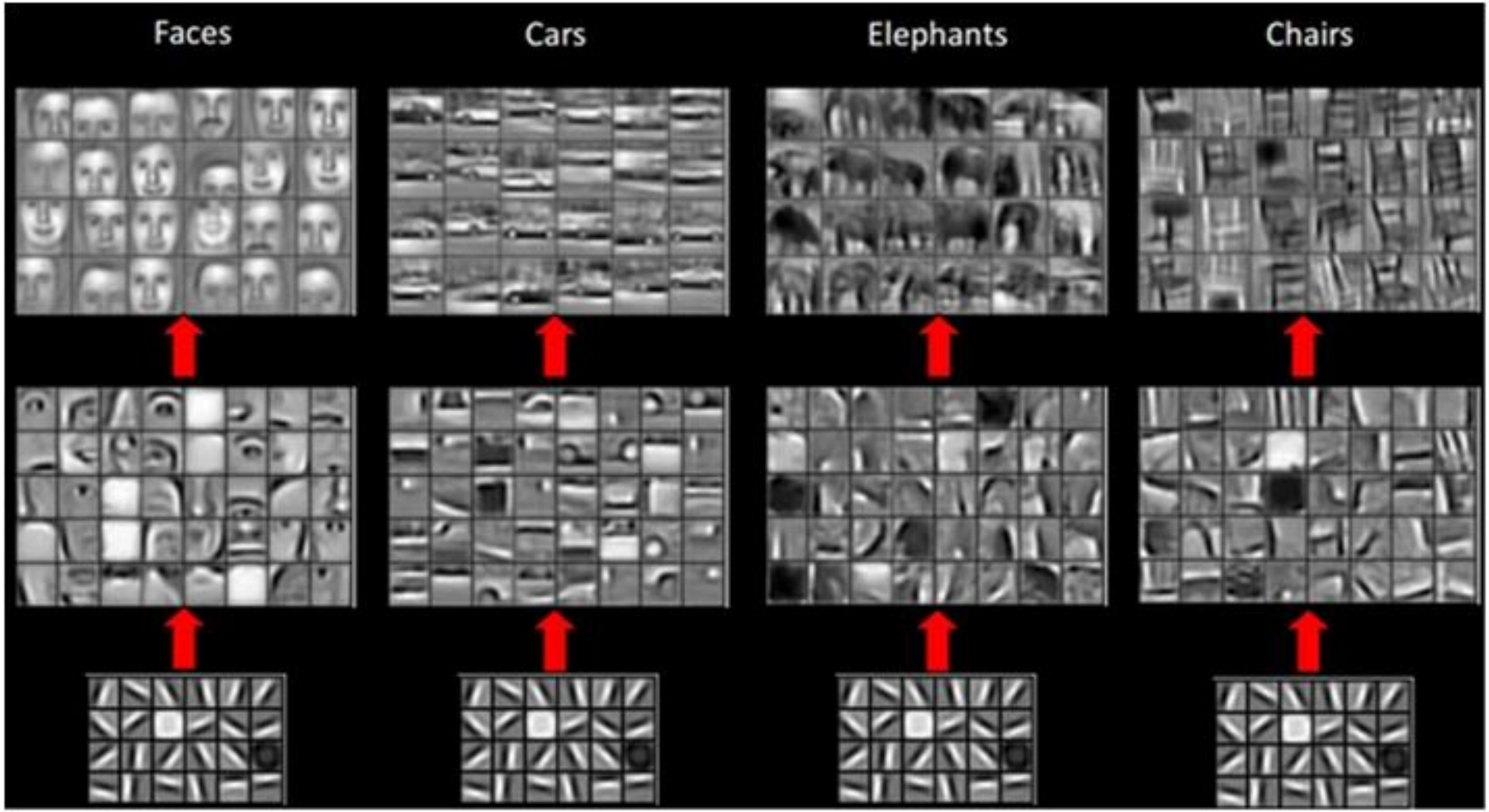


Example of Maxpool with a 2x2 filter and a stride of 2

https://github.com/leriomaggio

CNN: internal representations

CNN: internal representations

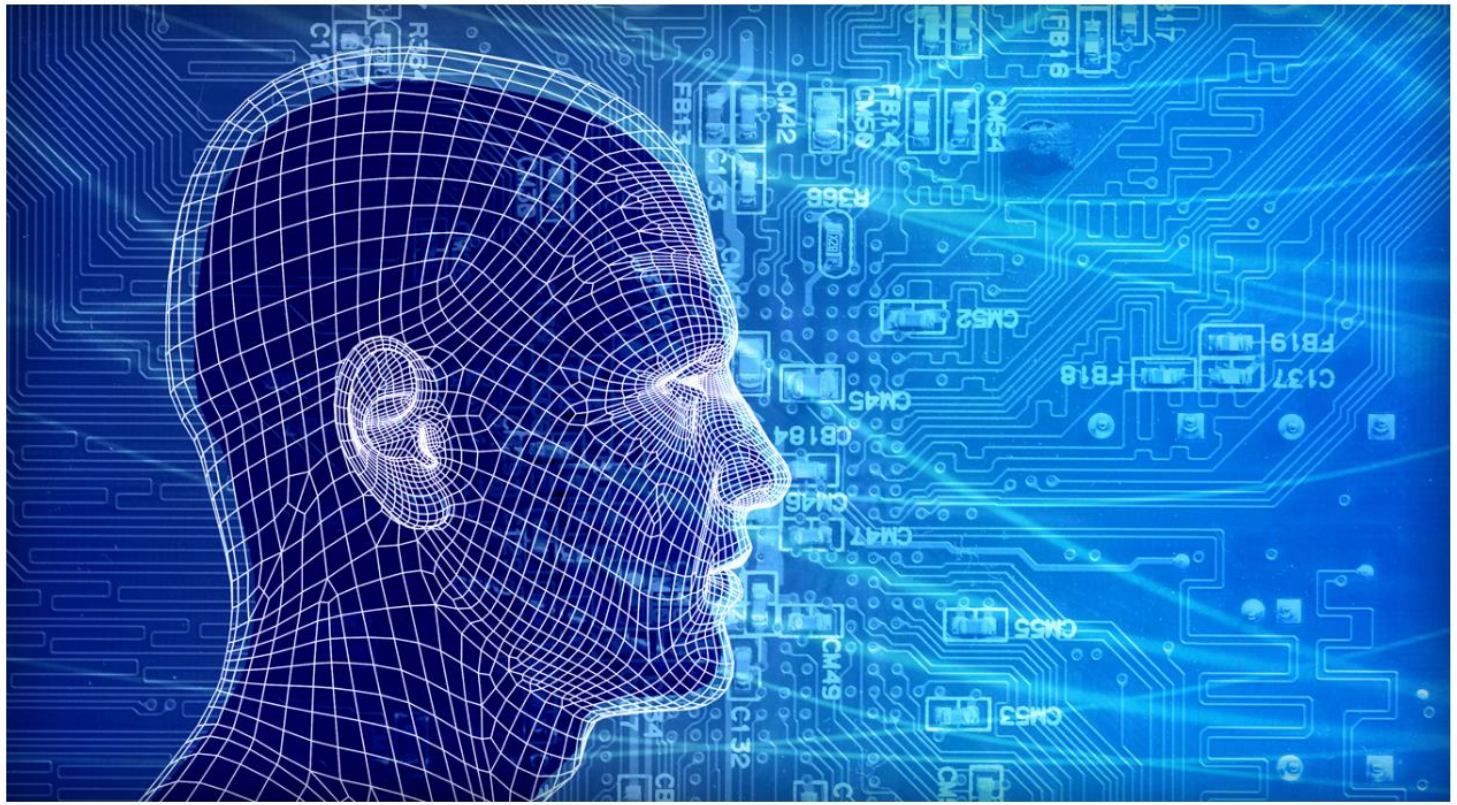

Faces
Cars
Elephants
Chairs


5 days ago

Deep learning algorithm diagnoses skin cancer as well as seasoned dermatologists

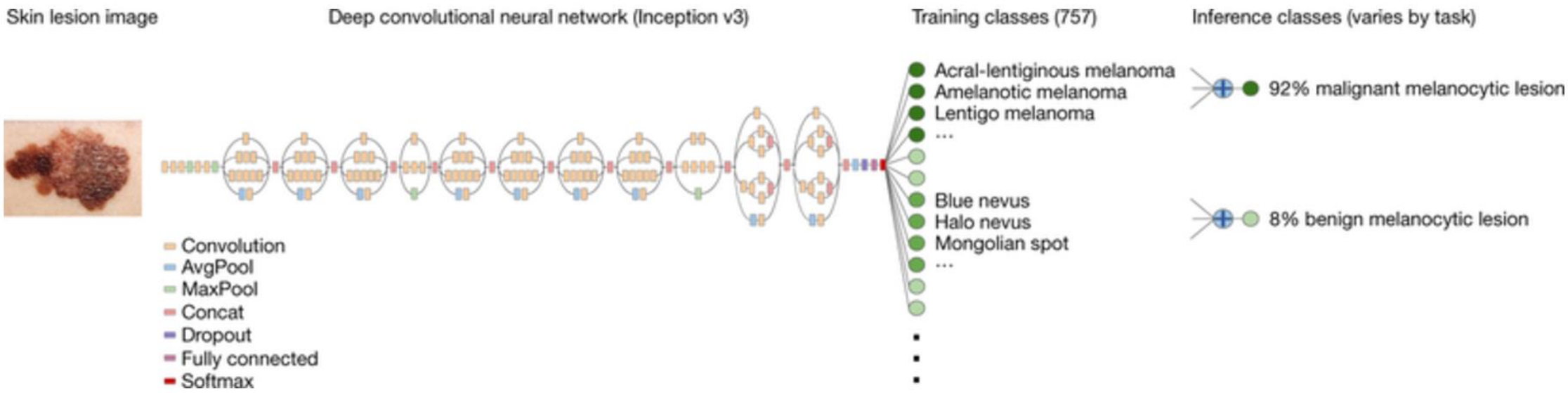
By Jessica Hall on January 25, 2017 at 1:25 pm | 16 Comments

422 shares



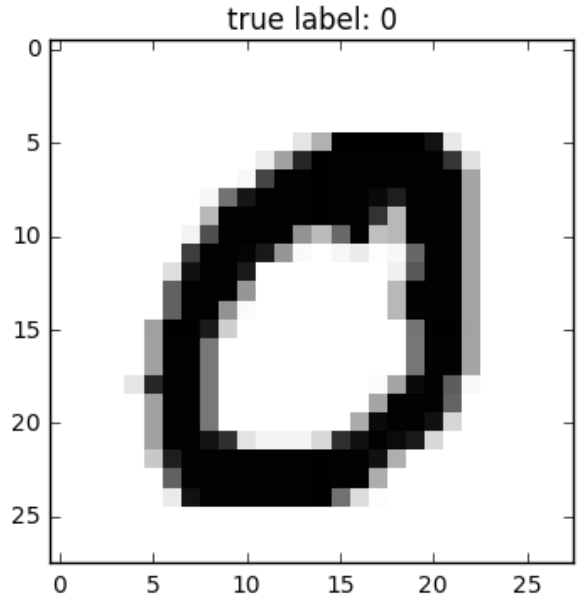
Thrun et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, January 2017 DOI: 10.1038/nature21056

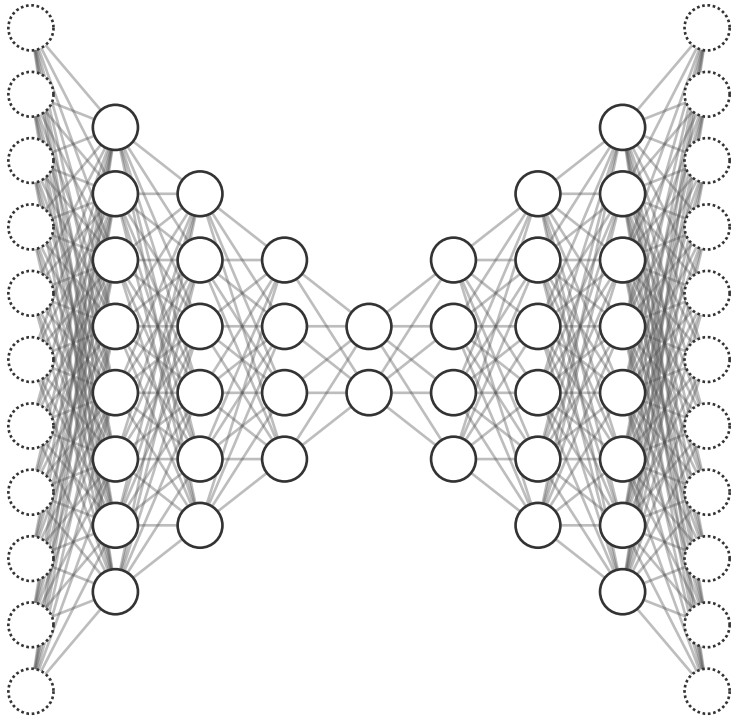
CNN: fine-tuning

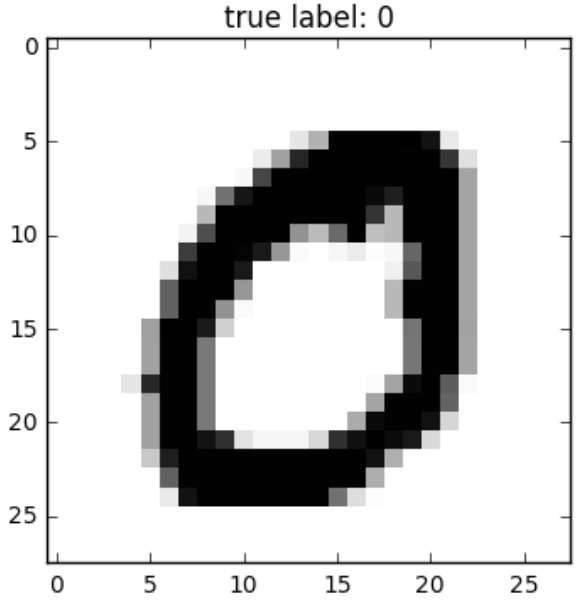


Thrun et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, January 2017 DOI: 10.1038/nature21056

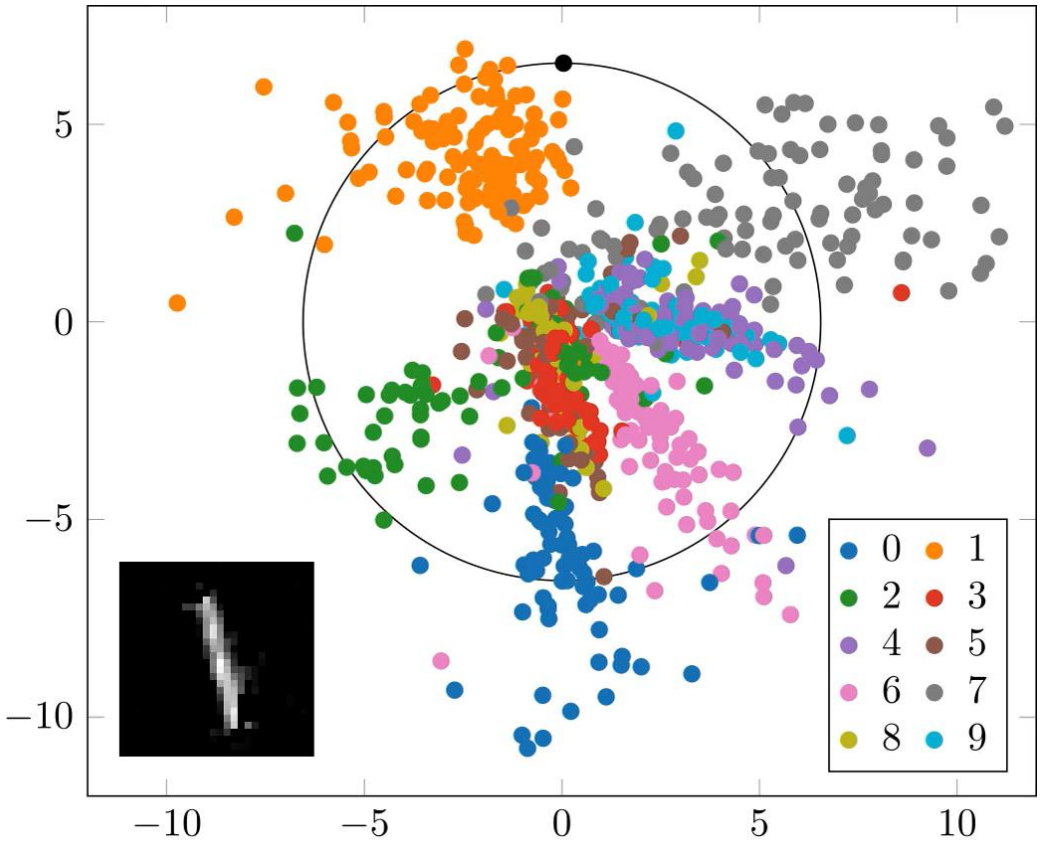
Embeddings: auto-encoder

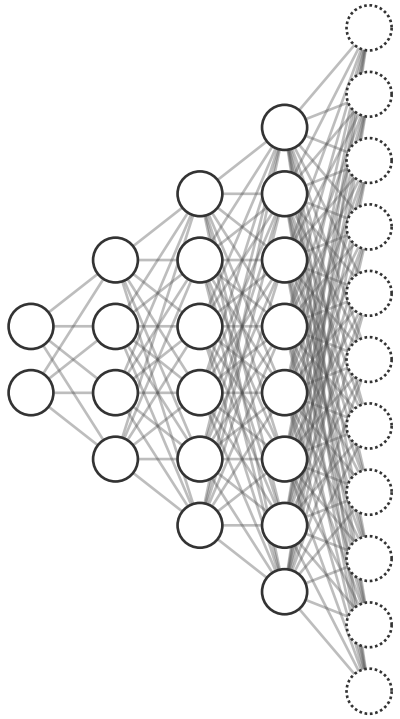


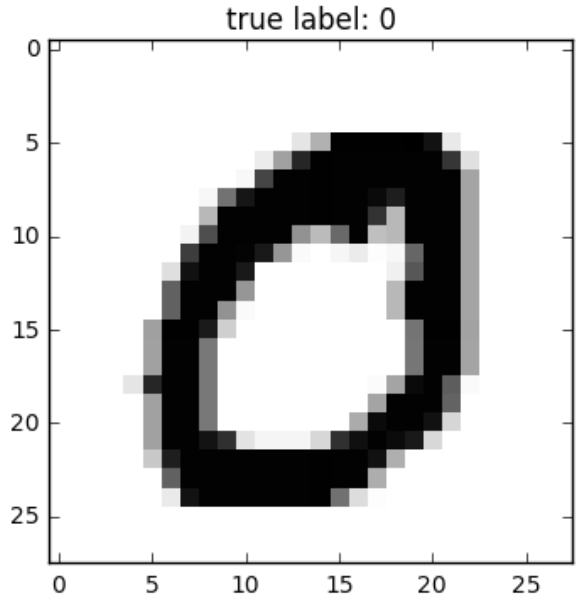




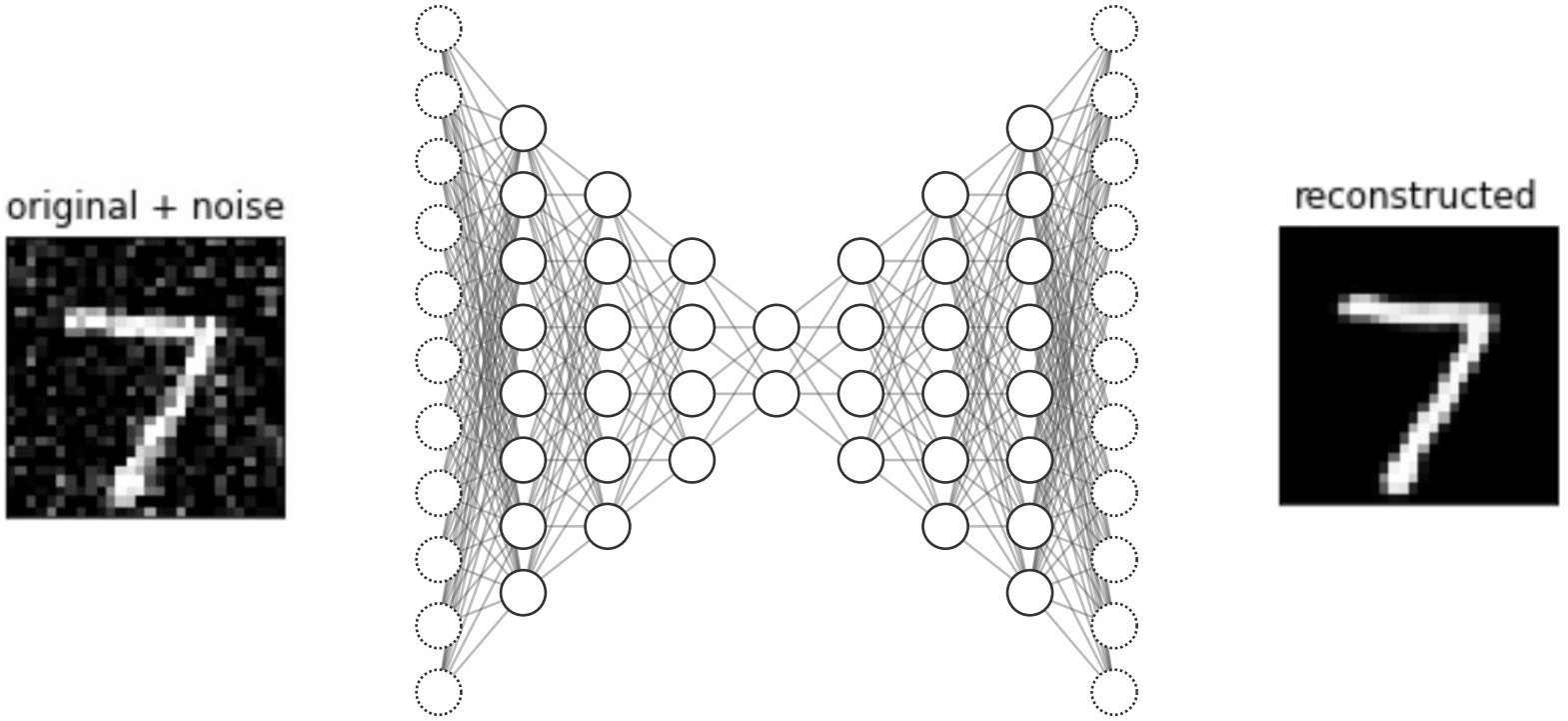
Embeddings: auto-encoder







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


VIB-UGENT CENTER
FOR MEDICAL
BIOTECHNOLOGY
GHENT
UNIVERSITY
Comp
omics