UNIVERSITÄT BONN

Machine Learning Basics

Elena Trunz

February 17, 2025

Visual Computing Group, University of Bonn

Overview

Machine Learning Algorithms

Image Classification

Nearest Neighbor Classification

Linear Regression as ML Algorithm

Generalization

Machine Learning Algorithms

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

What is machine learning?

Machine learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically.

What do we mean by learning?

"A computer program is said to learn from *experience* E with respect to some class of *tasks* T and *performance measure* P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997[1])

[1] Mitchell, T. M. (1997). Machine Learning. McGraw-Hill, New York.

Experience E

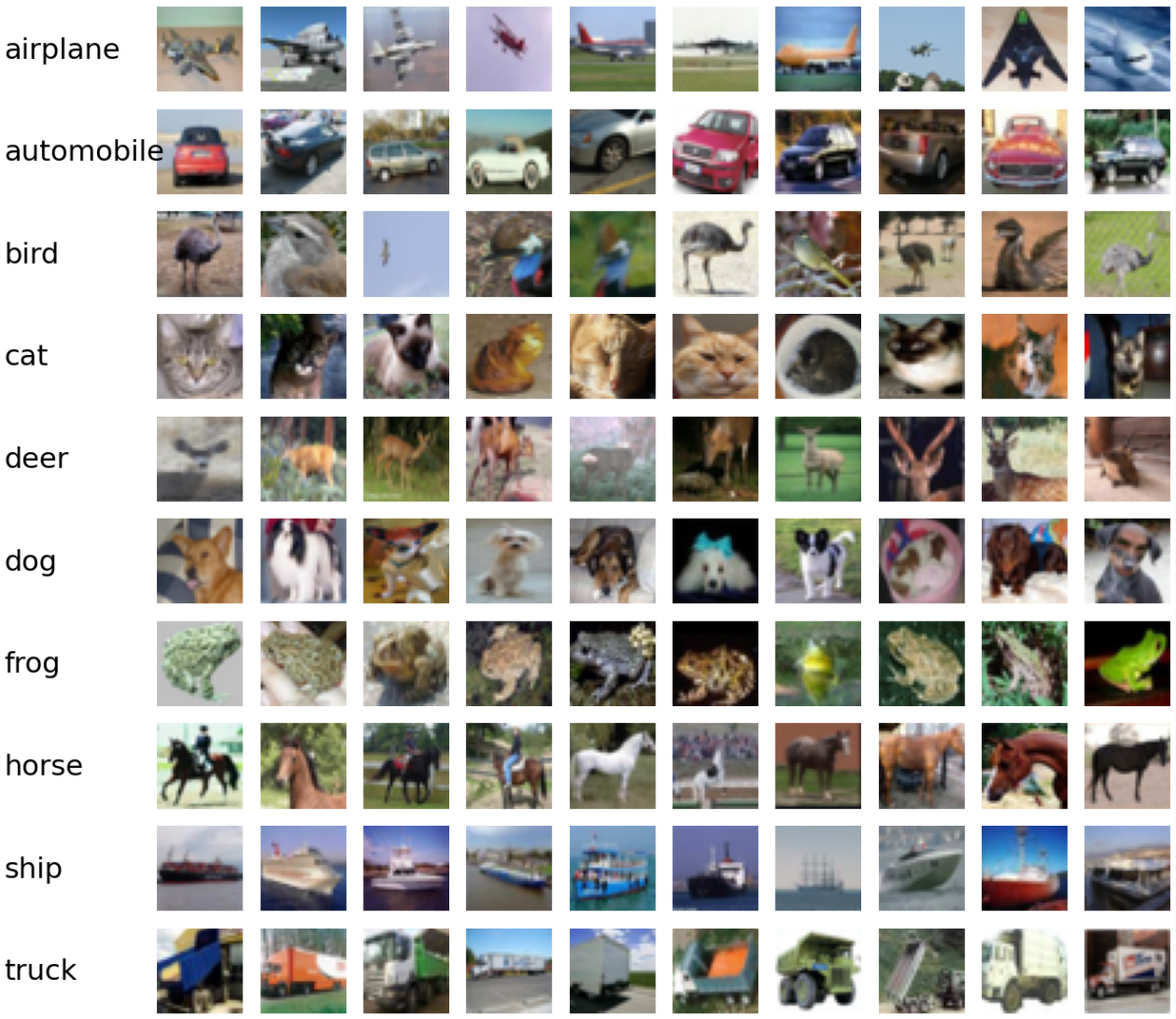
The experience E is the information that the algorithm can use during learning:

- *dataset*
- aka (training) data

Dataset contains *examples* (*data points*): collection of *features* that have been quantitatively measured from some object or event

- m-dimensional data point: $\mathbf{x} \in \mathbb{R}^m$
- each entry x_i is another feature

Example dataset: CIFAR10



10 classes

50,000 training images

10,000 testing images

Alex Krizhevsky "Learning Multiple Layers of Features from Tiny Images", 2009.

Supervised vs. unsupervised learning

Supervised learning:

- Each data point **x** comes with an associated *label* or *target* value **y**
- The algorithm is learned by trying to match the targets, or predict **y** from **x**

Unsupervised learning:

- Only data points **x**, no *labels*
- The goal is to uncover the inherent structure within the data

Machine learning tasks

Tasks specify how a machine learning system should process *examples*. Common tasks include

- **classification**: determine category of input i.e. $f : \mathbb{R}^m \to \{1, ..., K\}$
- **regression**: predict numerical value(s) for some input, i.e. $f : \mathbb{R}^m \to \mathbb{R}^p$
- **density estimation**: learn probability density (if $\mathbf{x}$ is continuous) or probability mass (if $\mathbf{x}$ is discrete) function $p_{\text{model}} : \mathbb{R}^m \to \mathbb{R}$ on the space that the examples were drawn from
- **dimensionality reduction**: find a compact, lower-dimensional representation of high-dimensional data $\mathbf{x} \in \mathbb{R}^D$, which is often easier to analyze than the original data

More machine learning tasks

- **imputation of missing values** or **hole filling**: predict values of missing entries x_i of $\mathbf{x} \in \mathbb{R}^m$
- **structured prediction**: vector output with well-defined relationships between the elements
- **synthesis**: generation of new examples that are similar to the training data
- **denoising**: get *clean* example $\mathbf{x} \in \mathbb{R}^m$ from *corrupted* example $\tilde{\mathbf{x}} \in \mathbb{R}^m$

Performance measure P

To evaluate a machine learning algorithm, we must design a quantitative measure of its performance.

Performance measure is task-spesific:

- Classification and imputation of missing inputs:
  - *Accuracy*: proportion of examples for which the model produces the correct output
- Density estimation:
  - Average log-probability the model assigns to examples

Generalization

We want to evaluate performance of our system on data that it has *not yet seen*.

This estimates how well the system will *generalize*, in comparison to performance on already seen data used during the learning process.

- Unseen data makes up our *test set* or test data
- Already seen data is the *training set* or training data

During training we want to reduce the *training error* – this is simply an optimization problem.

What distinguishes learning from optimization is the interest in also reducing the *test error* (*generalization error*).

Loss function

A *loss function* is a function that measures the difference (or loss) between a predicted label and a true label

It is closely related to performance measure:

- Loss function is used *during* the training to *train* the model
- Performance measure is used *after* the training to *evaluate* the model

Sometimes both can be represented by the same function (as it is in the case of regression), but for some other tasks (e.g. classification) we have different measures (e.g. cross-entropy vs. accuracy).

Image Classification

Classification task

Given:

- Input examples
- Corresponding labels

Find: Classifier that predicts a class from the input.

Example: given an x-ray image, predict if the person is sick.

Image classification

$\longrightarrow$ cat

Semantic gap

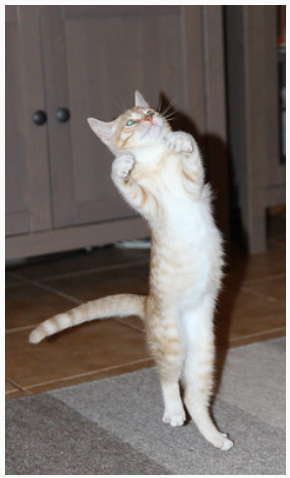
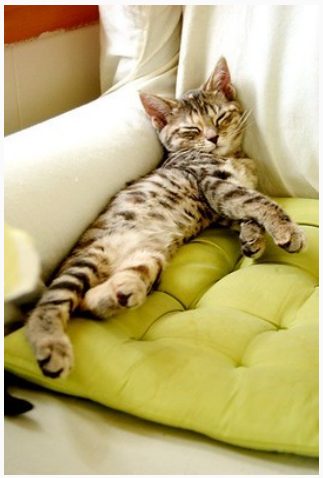
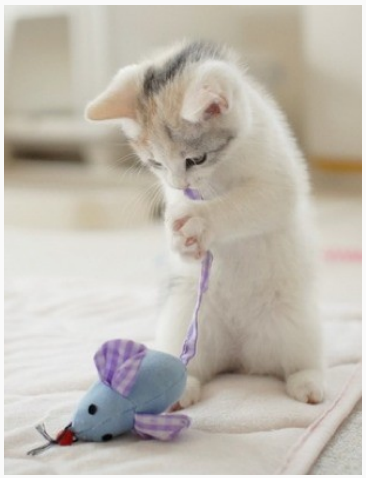
What we see

```
[[206  84 145   1  24  61 124  70 186 103  82 114 255 203 169]
 [147 250 234  12 245  90  72 193  21 100 181 247 161 240 110]
 [ 91  68  20 176 107 176 193 254 245 109 186 232 184 246 145]
 [  6  72 199 103  85  64   7 189 254  75  52 137 190 223 240]
 [ 64  98 182  71 255  36 212  19 185  60  66 148 105 177 198]
 [ 32 214  18  81  66 134 229 238  34  43  33  59  67 229  88]
 [ 75 231  24  73 167 252 151  14  38 166 137 150  71  29   4]
 [150 121  24 175 255  69 104 105 240 182 104 194  58 137  82]
 [233  74 152 138  83 142  58 150  24 110 109  75 224 236 221]
 [113  70 194 104  37 181 117  97 240  43 197 109 221 119 142]
 [ 68  51 170 173  25  68 154 152  46 106  72  54 115  89  46]
 [222 101 166 211 102 238 165  54  51 252   4  94 128 161   0]
 [ 75 180 254 150 144  53 197 152  66 108  57  32 245 190  93]
 [ 11  32 193 251 225 199 147 105  52  91  32 207  47  80 112]
 [115  20 114  74 212  57 134  92 195 193 167 177 241 131 153]
 [ 49 117 252  34 102 116 222  83 205 119  38 194 253 164  71]
 [ 59 159 178  52   5 148 225   1 136 184  75  22  43  39 107]
 [142   8  38 104 184 249 186 133 190 229 197 119 144 105 235]
 [ 51 157  72  11 215 159  79  11 129  59  23 125  21   7 207]
 [ 18 223  10  32 153 137   6 201  16   7 155 136   9  26 197]
 [184  63 124 159 173  99  83   5  36  36 186 105  38  21  34]
 [189 106  15  64 146 215  87  17  68 148 130  99 202 210 252]
 [149 144 228 159  65  63 164 179  78 199  92 135  91 231 242]
 [141  96 169  31  18  59 243 190  12 154 227  43 128  98 133]
 [205  80 149 171 116  87  62 231 142 193  95  56 239 145  67]
 [ 31 106 144 102 235 215 157 225 169 196  64 219 123  50  58]
 [180  44 126  18  94  64  35 234 196  95 107  87  95  97   7]
 [ 84  84 247 104 188 106  99 140  58  40 171  92 111  77 135]
 [220 163 200 177 156 226  95  21 152 176   2   1  96  90  12]
 [ 67 101 141  37 163 159 212 212  48 220  65 116 188  89  34]]
```

What the computer sees

- An image is just a grid of numbers between 0 and 255
- E.g. this RGB image: $300 \times 400 \times 3$

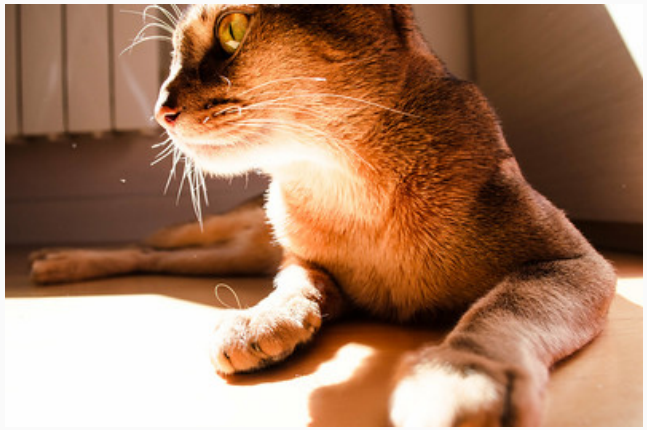
Challenges: deformation


Challenges: occlusion



Challenges: illumination


Challenges: background clutter

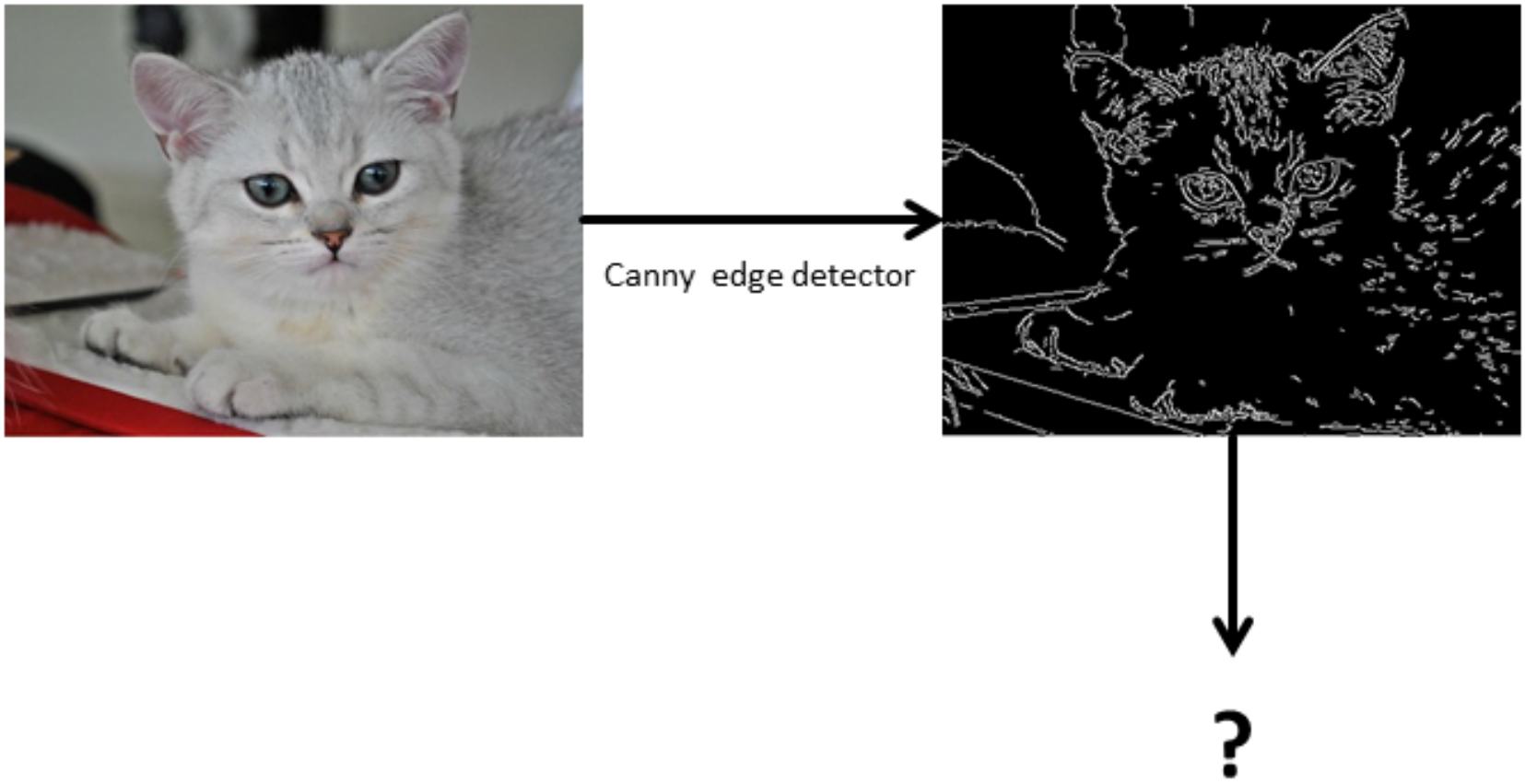
Challenges: intraclass variation

A rule-based image classifier

```
def classify_image():
    # do something
    return class_label
```

- There are many rule-based algorithms for e.g. sorting a list of numbers
- No obvious way to hard-code the algorithm for recognizing a cat or other classes

Attempts have been made



Data-driven classification

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(train_images, train_labels):
    # machine learning
    return model

def predict(model, test_images):
    # use model to predict labels
    return test_labels
```

Nearest Neighbor Classification

First classifier: Nearest Neighbor

```
def train(train_images, train_labels):
    # memorize all data and labels
    return model

def predict(model, test_images):
    # predict label of most similar
    # training example
    return test_labels
```

What is similarity?

- The real meaning of similarity is a philosophical question
- We will take a more pragmatic approach

Definition: Let U be the universe of possible objects. The *distance* (*dissimilarity*, *metric*) on U is a function $d : U \times U \to \mathbb{R}$ that satisfies certain conditions (axioms).

Intuition behind axioms of a distance function

$$d(a, b) = 0 \Leftrightarrow a = b \qquad \textit{Identity of indiscernibles}$$

Otherwise there are objects that differ from each other, but we can't distinguish them.

$$d(a, b) = d(b, a) \qquad \textit{Symmetry}$$

Otherwise we could claim "Mary looks like Jenny, but Jenny bears no resemblance to Mary."

$$d(a, c) \leq d(a, b) + d(b, c) \qquad \textit{Triangle inequality}$$

Otherwise we could claim "Mary looks like Jenny and Sarah, yet Jenny bears no resemblance to Sarah."

L1 (Manhattan) distance

One common similarity measure is the L1 distance:

$$d_1(a, b) = \sum_{i=1}^{m} |a_i - b_i|$$

image a

4	56	13	21
55	78	7	32
96	10	33	12
87	14	2	43

$-$

image b

56	3	6	15
71	42	8	11
30	98	77	12
40	82	15	22

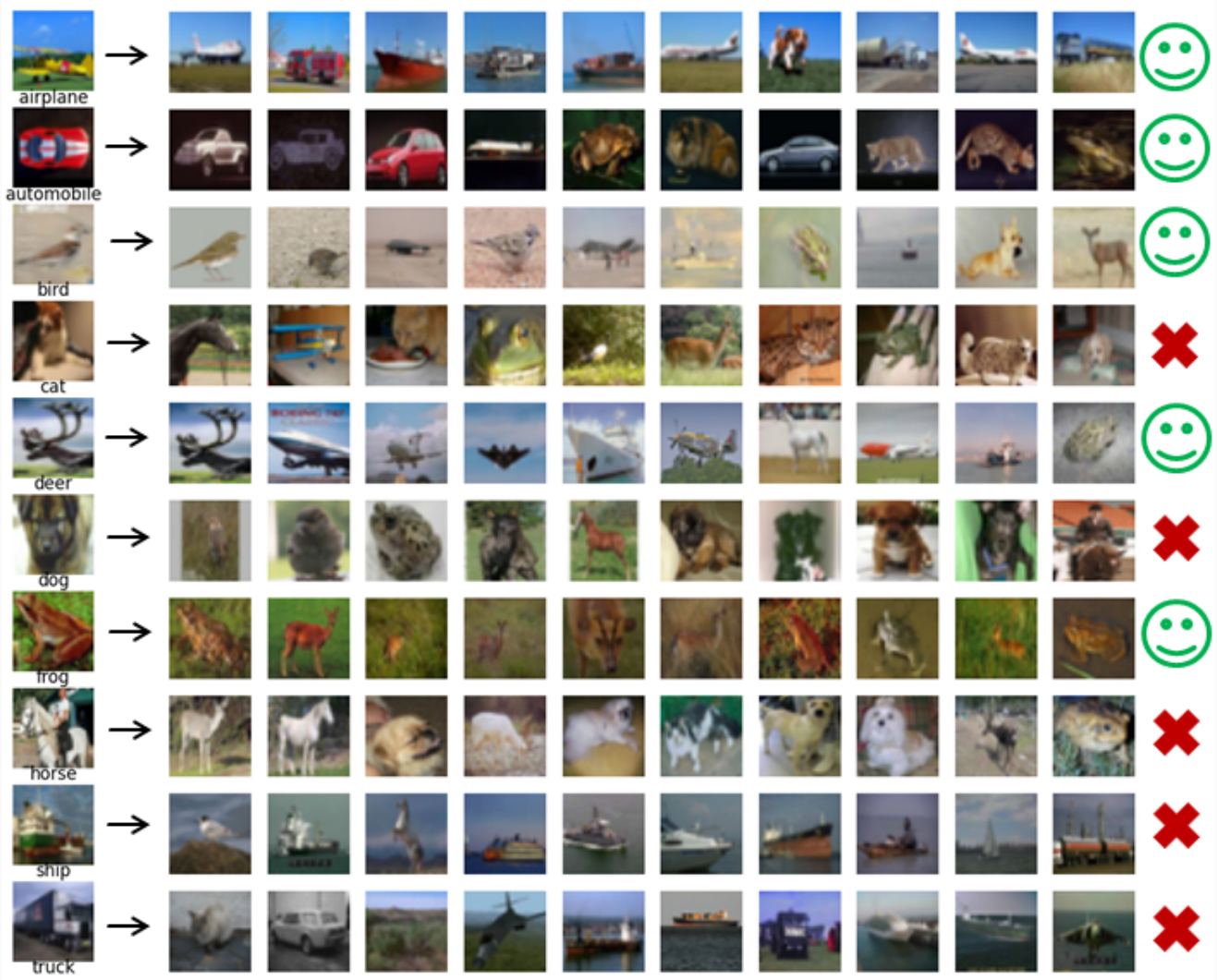
$=$

pixel-wise absolute value differences

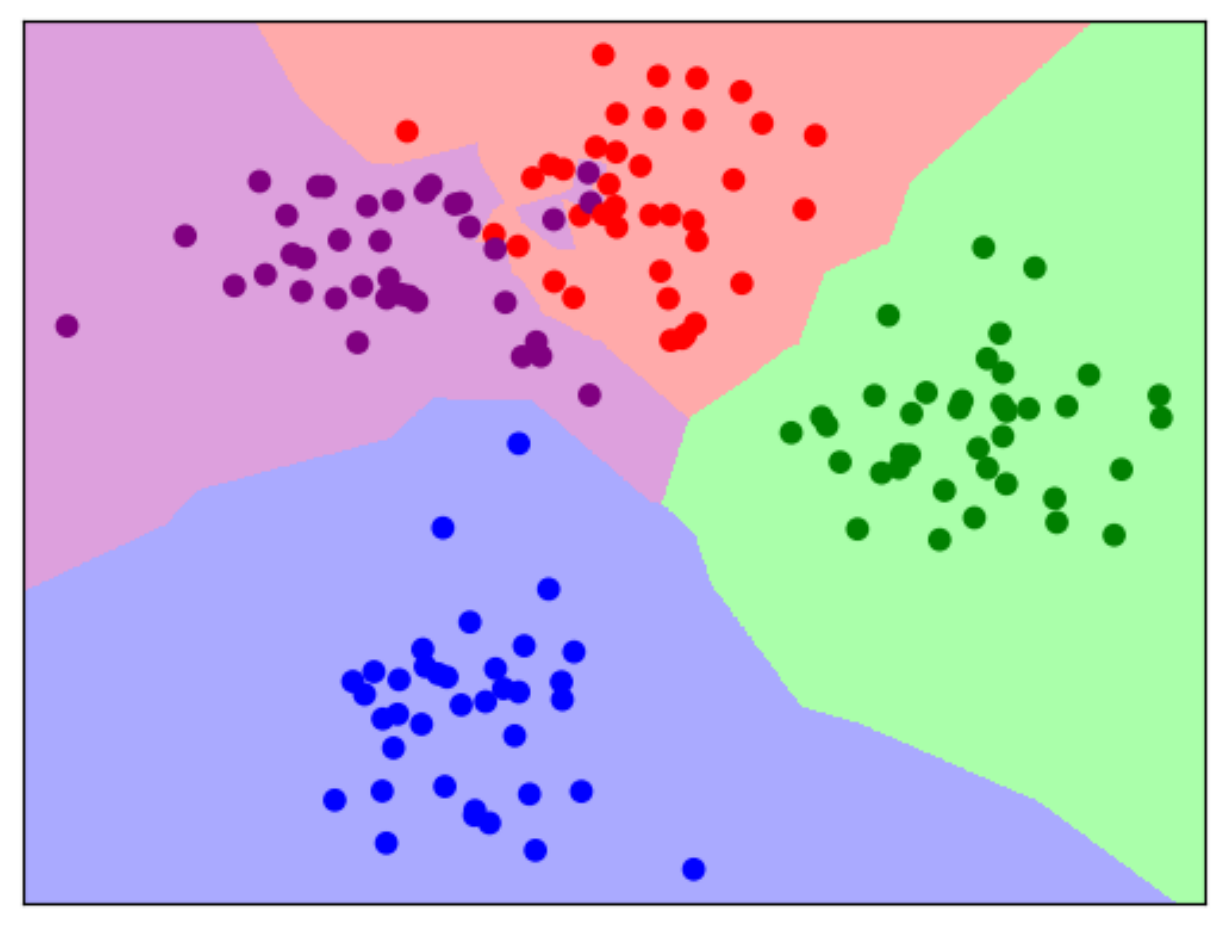
52	53	7	6
16	36	1	21
66	88	44	0
47	68	13	21

$\xrightarrow{\text{sum}}$ 539

NN results on CIFAR10

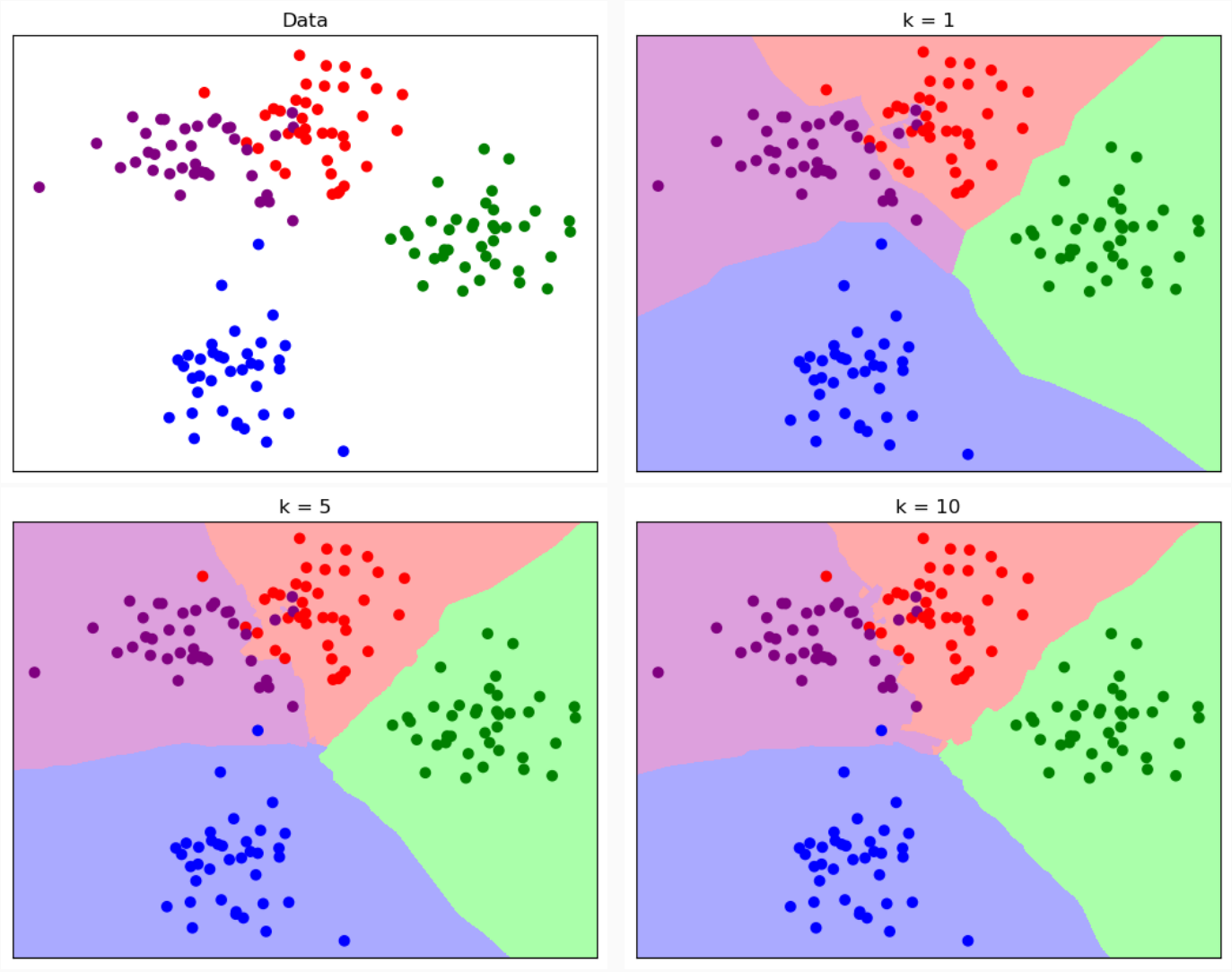


Decision regions of NN

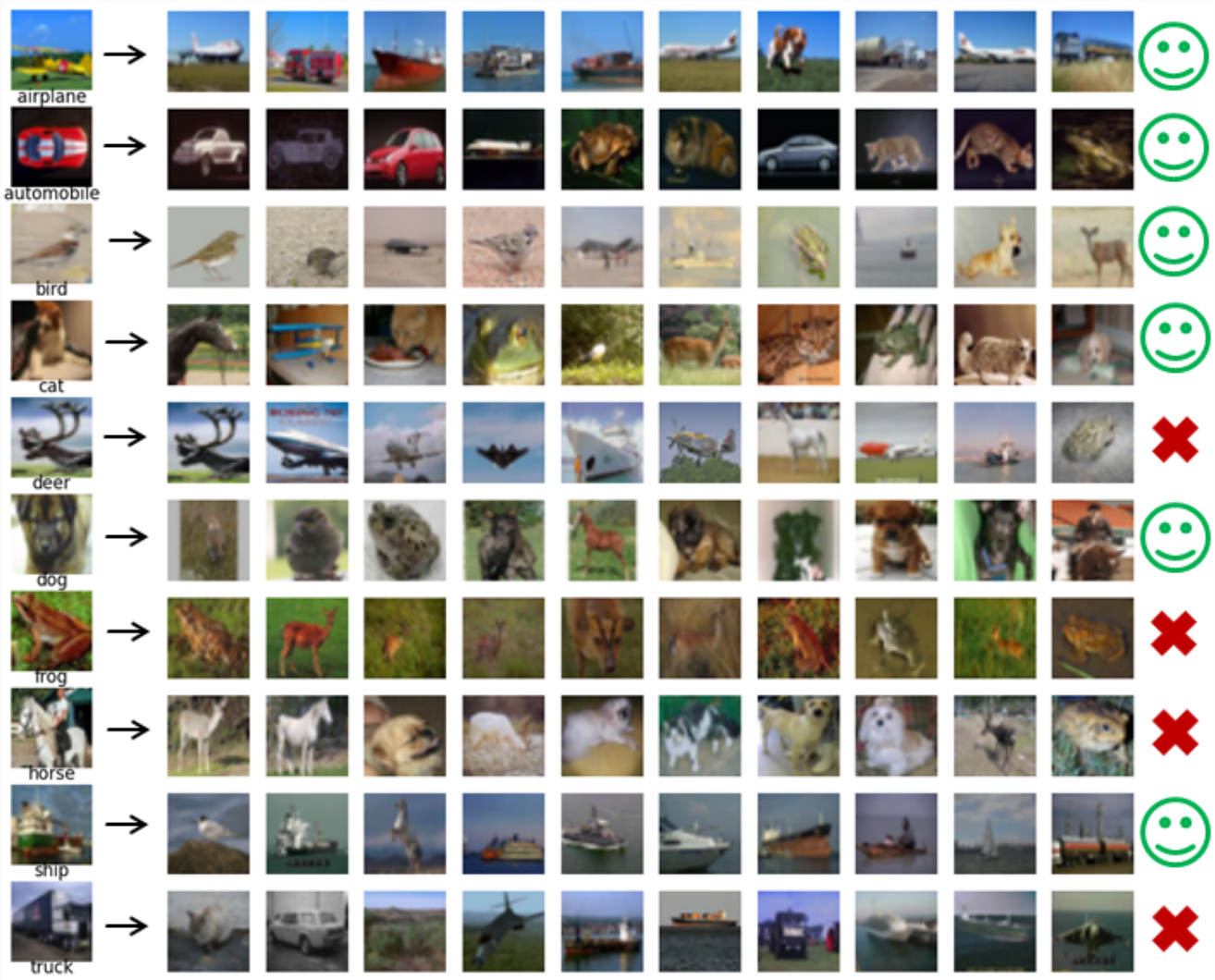


k-Nearest Neighbors classifier

Instead of copying label from nearest neighbor, take *majority vote* from k closest points:



10-NN results on CIFAR10



L2 (Euclidean) distance

Another common similarity measure is the *Euclidean distance*.

The Euclidean distance between two m-dimensional points is the length of a line segment between these points

$$d_2(a, b) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}.$$

Choosing k and distance metric

What is the best distance?

What is the best value of k?

Very problem-dependent!

Try them all out and see what works best

Linear Regression as ML Algorithm

Linear regression: experience

Recall the regression problem: the goal is to build a system that

- takes a vector $\mathbf{x} \in \mathbb{R}^m$ as input and
- predicts the value of a scalar $y \in \mathbb{R}$ as its output.

Our experience E are the input examples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and the corresponding target values $\mathbf{y} \in \mathbb{R}^N$.

Thus, the dataset consists of N example pairs
$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Linear regression: task

Let $\hat{y}$ be the value that our model predicts y should take on. We define the output to be

$$\hat{y} = \mathbf{w}^\top \mathbf{x}$$

where $\mathbf{w}$ is a vector of model *parameters*.

Parameters are values that control the behavior of the system.

$\rightarrow$ Our task T: predict y from $\mathbf{x}$ by outputting $\hat{y} = \mathbf{w}^\top \mathbf{x}$.

Linear regression: performance measure

Suppose we have M example inputs together with the correct target values in the test dataset.

One way of measuring the performance of the model for a regression task is to compute the *mean squared error* (MSE):

$$\begin{aligned} MSE_{test} &= \frac{1}{M} \sum_i (\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)})_i^2 \\ &= \frac{1}{M} \|\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)}\|_2^2 \end{aligned}$$

Linear regression: ML algorithm

To make a machine learning algorithm, we need to design an algorithm that will improve the parameters $\mathbf{w}$ in a way that reduces MSE_{test} when the algorithm is allowed to gain experience by observing the training set $(\mathbf{X}^{(train)}, \mathbf{y}^{(train)})$.

One way to do it is to minimize MSE_{train}. We can simply solve for where the gradient of MSE_{train} is $\mathbf{0}$:

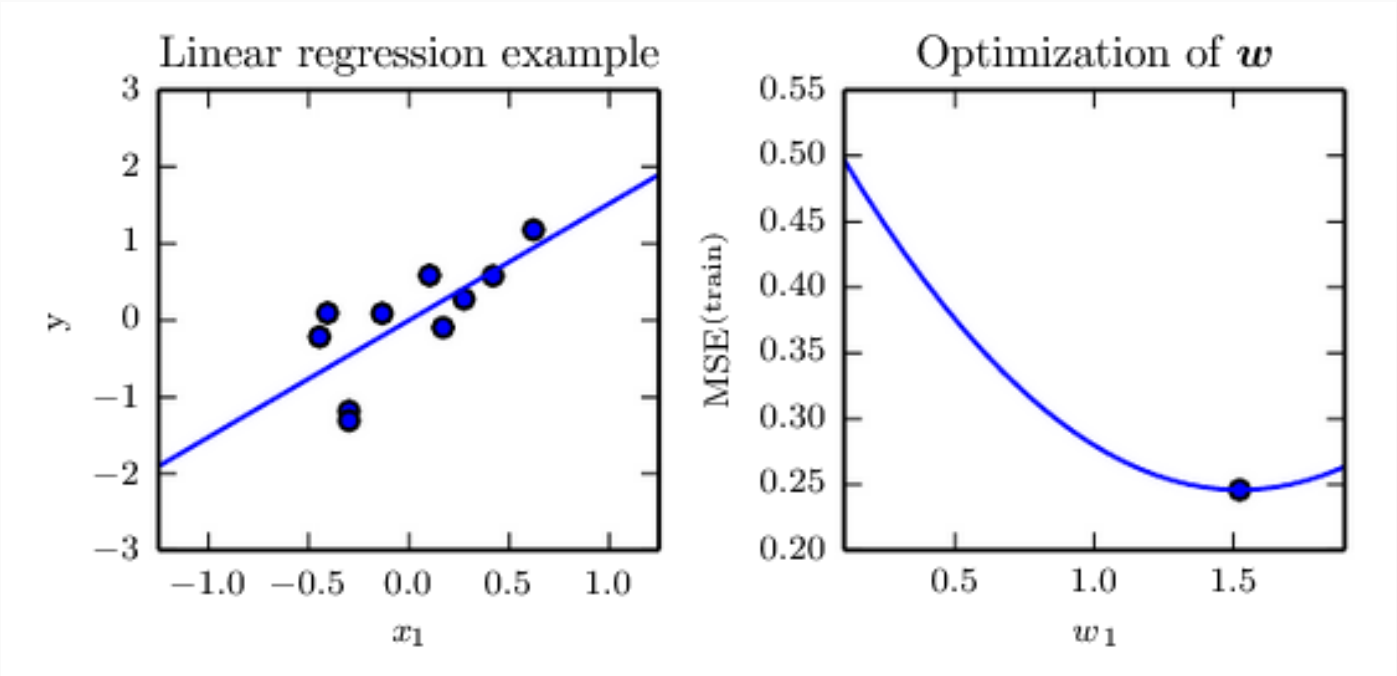
$$\nabla_{\mathbf{w}} MSE_{train} = \mathbf{0}$$

$$\nabla_{\mathbf{w}} \frac{1}{M} \|\hat{\mathbf{y}}^{(train)} - \mathbf{y}^{(train)}\|_2^2 = \mathbf{0}$$

$$\frac{1}{M} \nabla_{\mathbf{w}} \|\mathbf{X}^{(train)}\mathbf{w} - \mathbf{y}^{(train)}\|_2^2 = \mathbf{0}$$

$$\Rightarrow \mathbf{w} = (\mathbf{X}^{(train)\top}\mathbf{X}^{(train)})^{-1}\mathbf{X}^{(train)\top}\mathbf{y}^{(train)}$$

Linear regression: result



Training sets consists of ten examples, each containing one feature. So there is only one parameter to learn, w_1 (= slope).

Image was taken from [GBC16]

Linear regression: notes

Usually linear regression refers to a model with an additional parameter – an intercept term b:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

This means that the plot of the model's predictions need not pass though the origin.

We can continue to use the model $\hat{y} = \mathbf{w}^\top \mathbf{x}$, by implicitly including b:

- Augment $\mathbf{x}$ with an extra entry that is always set to 1
- The weight corresponding to the extra 1 entry plays the role of the bias parameter b

Generalization

Underfitting and overfitting

How well a machine learning algorithm will perform, depends on its ability to

1. Make the training error small
2. Make the gap between training and test error small

Algorithms which *underfit*, cannot satisfy the first criteria.

Algorithms which *overfit*, can satisfy the first criteria but not the second.

Model Capacity

Underfitting and overfitting can be controlled by adjusting the *model capacity* in the learning algorithm.

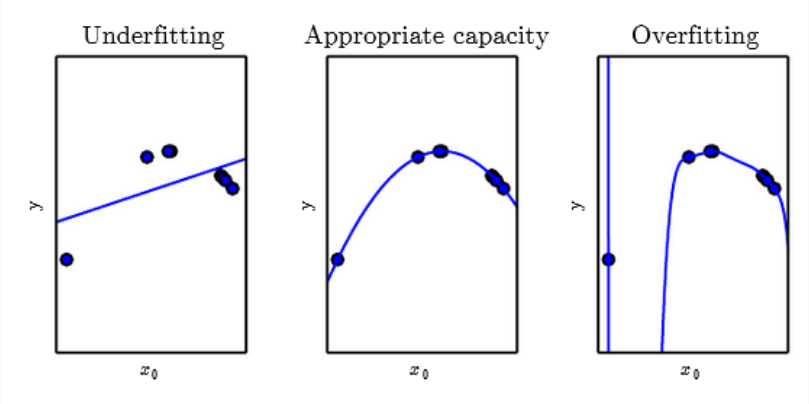
A model's capacity is its ability to fit a wide variety of functions.

Models with (too) low capacity are not rich or expressive enough to capture the training data and hence underfit.

Models with (too) high capacity simply memorize the training data and will therefore overfit.

Model Capacity (2)

- Capacity can be controlled by the choice of the *hypothesis space*. This space determines the set of functions that the learning algorithm can select as being the solution.
- The linear regression algorithm has the set of all linear functions of its input as its hypothesis space.
- We can increase the model's capacity by generalizing linear regression to include polynomials in its hypothesis space.

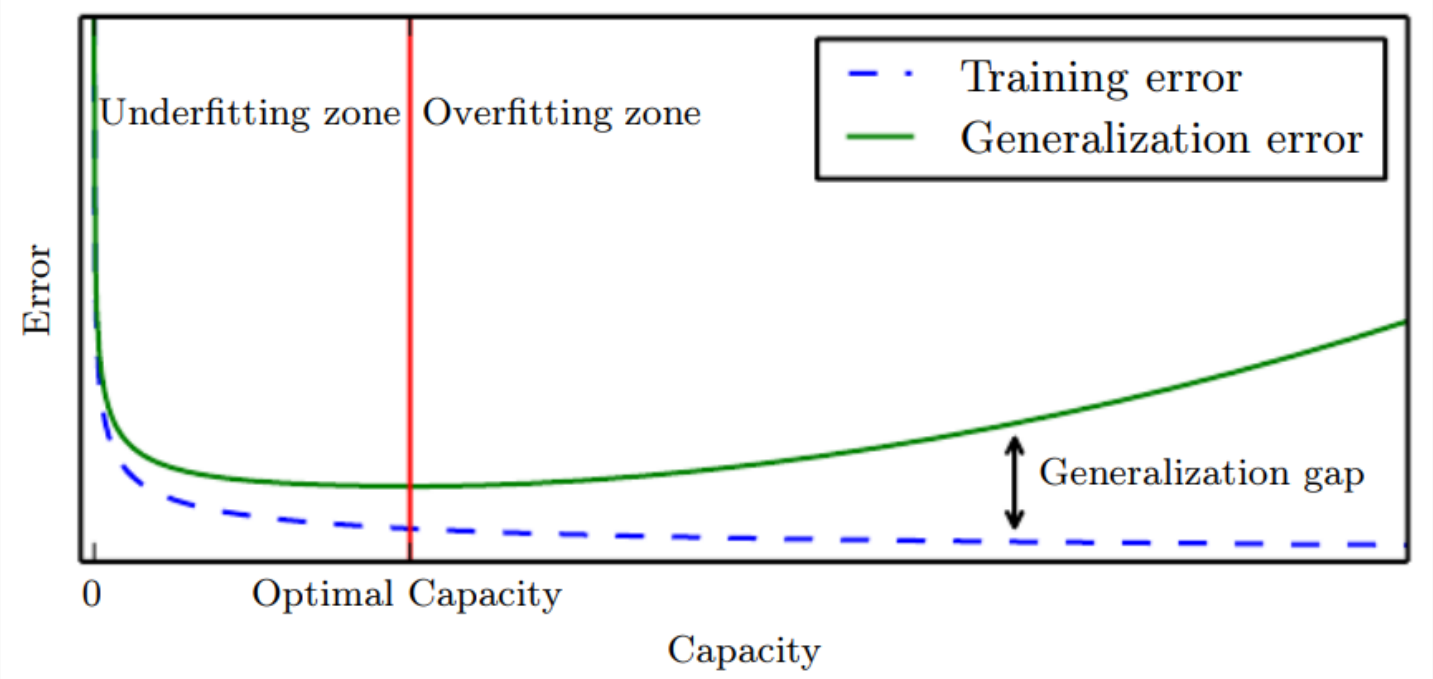


Capacity here is determined by the polynomial degree.

Image was taken from [GBC16]

Capacity vs. error

Depending on the model capacity training and test error behave differently:



Typical relationship between capacity and error.

Image was taken from [GBC16]

Regularization

Another way to control underfitting and overfitting is to drastically increase the capacity and use a *regularizer* so as to not overfit.

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

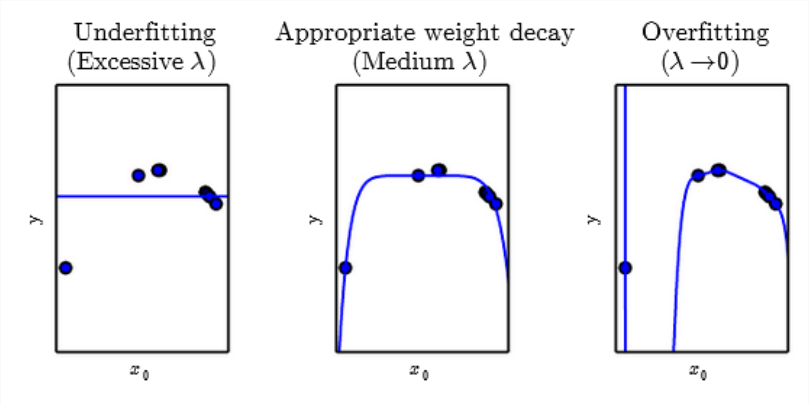
We can regularize a model by giving a learning algorithm a preference for one solution over another in its hypothesis space.

Regularization: weight decay

- We can modify the objective function (loss function) for linear regression to include *weight decay*:

$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^\top \mathbf{w}$$

- Regularizer penalizes weights that have bigger squared L^2 norm. λ controls the strength of our preference for smaller weights.



We fit a high-degree polynomial regression model and use weight decay against overfitting.

Image was taken from [GBC16]

Hyperparameters

The parameters $\mathbf{w}$ of the regression example are determined (optimized) during the training. Such parameters are also called *weights*.

Most learning algorithms have some free parameters that are not determined by the learning algorithm, but are set in advance to control the algorithm's behavior.

Such free parameters are called *hyperparameters*.

In the previous regression example the polynomial degree and λ are two hyperparameters.

Hyperparameter estimation

We cannot use our test set to make any choices about the model, including its hyperparameters.

To determine the hyperparameters, we need a *validation set* of examples that the training algorithm does not observe.

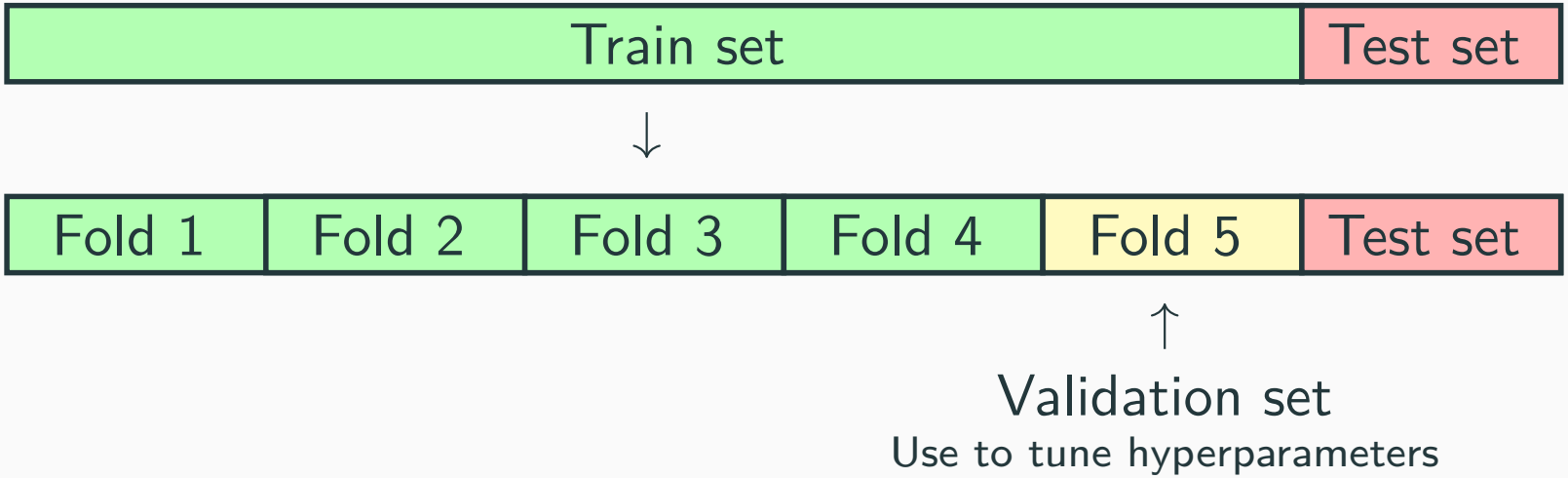
We always construct the validation set from the training data:

- Split the training data into two disjoint subsets
- One of the subsets is used to learn the parameters
- The other subset is used to estimate the generalization error during or after training and to guide the selection of the hyperparameters

After the training and all hyperparameter optimization is complete, we can estimate the generalizaiton error on the test set.

Cross-validation

If the training set is small and the split would result in the validation set being too small, cross-validation can help:



- Loop through all folds to be the validation fold
- Other folds build the train set
- Average performance values

What do we mean by learning? - revised

Learning: Based on training data, finding a model with parameters $\boldsymbol{w}$ that generalizes well.

In machine learning models are often called *estimators* or *hypotheses*.

References

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.