

Advanced Machine Learning

Informal syllabus

Erik Koene

November 2018

-1 The historical setting of statistical learning theory

This section, starting of at minus 1, is mostly based on VLADIMIR VAPNIK's *The Nature of Statistical Learning – for Engineering and Information Science* (1995) and more thorough *Statistical Learning Theory* (1998). This background information is interesting to set up the topic.

-1.1 Parametric inference, 1920s–1960s

By the 1920s, descriptive statistics was mostly complete: statistical laws (distribution functions) were found to describe various events of reality well. The next topic was that of **statistical inference**: when given a collection of empirical data origination from some functional dependency, infer this dependency. The two main approaches to this issue were the following:

1. The particular (parametric) inference, when the investigator knows the problem to be analyzed well enough to assume the statistical law and target function: the problem then reduces to finding the optimal parameters using the maximum likelihood method.
2. The general inference, when one does not have reliable a priori information about the statistical law underlying the problem or about the function that one would like to approximate.

The first approach (parametric inference) to inductive inference had its “golden age” between the 1930s and 1960s, in the time before computers, when simple methods were the only realizable option. The philosophy of the parametric approach assumed that:

1. *A set of functions, linear and minimal in their parameters, contains a good approximation to the desired function,*
2. *The statistical law underlying the stochastic component of most real-life problems is the normal law,*
3. *The maximum likelihood method is a good tool for estimating the parameters.*

However the wide availability of computers in the 60s showed that all three beliefs had shortcomings:

1. In realistic multidimensional problems with dozens or even hundreds of variables, the belief that one can define a reasonably small set of functions that contains a good approximation to a desired one is wrong, also through the ‘curse of dimensionality’ (R. BELLMAN).

2. Additionally, TUKEY found that real-life data often differs from classical statistical distribution functions, and one must take this into account to construct effective algorithms.
3. JAMES and STEIN showed that even for simple problems of density estimation, such as finding the mean of a $n > 2$ -dimensional Gaussian with unit covariance, the maximum likelihood method is not the best one. The estimator they suggest is uniformly better than the maximum likelihood estimator!

Thus, the beliefs upon which parametric inductive inference was based turn out to be inappropriate for many real-life problems, opening the door for other methods of statistical inference.

-1.2 Inductive inference, 1970s–2000s

ROSENBLATT introduced a computer program in 1958, based on a neurophysiological learning model called the Perceptron, which happened to generalize well to classification problems. Consequently, NOVIKOV proved a series of theorems about the Perceptron, particularly that it will converge to a solution in a finite number of iterations. (As a side-note, MINSKY & PAPERT released a book called ‘Perceptrons’ in 1969 that proved that these neurons couldn’t solve some very simple problems. This negative outlook led to a decline in interest in AI called the ‘AI winter’ – computers can’t do *logic*. However, from the ashes came ‘machine learning’, which tries to do much smaller, humbler, tasks and side-step questions of logic, and is based on **statistics and optimization**.) Thus arose the question *how, and under which constraints, can we generalize algorithms that are agnostic to the underlying probability density function?* And so, **statistical learning theory** is born!

The fundamental answer in statistical learning theory was found through *empirical risk minimization* (ERM): to generalize well to new data, we must minimize a specific *loss function* for our training data. The ERM principle yields the maximum likelihood method when we choose a logarithmic loss, but generalizes to many more varied decision rules! Machine learning is now similar to an optimization procedure, but requires you to choose the right optimization strategy!

While algorithmic advances went slow (it took until 1986 before the back-propagation technique for neural networks was discovered by LECUN), much more progress was attained in theoretical advances. VAPNIK and CHERVONENKIS developed the ERM theory to obtain minimal generalization error with only limited data. A particular insight was that we can simplify things when we want to predict values only in a *limited* range. The old parametric paradigm uses a two-stage procedure that is called **generative modeling**: at the first (induction) stage we estimate the probability density function, at the second (deduction) stage we use this function to compute values at new points of interest. The first step thus solves a problem that is more general than the one required, for it estimates the *entire* unknown function for *all* points on the domain. This is expensive and difficult to do well. When we only want to estimate the values of a function at a few points of interest, with a restricted amount of information, it is possible to estimate the unknown function in one immediate (transduction) step: **discriminative modeling**.

-1.3 Neural networks, 2010s–present

Then, with the rise of ‘big data’ and very powerful computers, multilayer perceptrons (or ‘neural networks’) started to outperform other methods developed with the ERM principle, and win many machine learning competitions! However, the methods require a lot of ‘fiddling the knobs’ to obtain good results, and the statistical community is playing catch-up to understand how to make the neural nets learn optimally and stably.

0 Prerequisites (informal introduction)

0.1 Probability

We define $\mathbb{P}(A)$ as the probability of event A to be measured from the set of *all* possible outcomes, mapping to a number within $[0, 1]$. We define $\mathbb{P}(A \cap B) = \mathbb{P}(A, B)$ as the probability of the set intersection (the common elements in A and B). Notably, for every event A there is the complement of A^c for which we have:

$$\mathbb{P}(A \cap A^c) = \mathbb{P}(\emptyset) = 0, \quad \mathbb{P}(A) + \mathbb{P}(A^c) = 1. \quad (1)$$

When probabilities are *independent* (the outcome of one event doesn't influence the likelihood of another event; they can happen in arbitrary order), it is implied that $\mathbb{P}(A, B) = \mathbb{P}(A)\mathbb{P}(B)$.

Example: Throwing dice, part 1

We'll roll two dice, and consider event A to represent that the first throw is $i \in \{1, 2, 3, 4\}$ and the second throw is $j \in \{4, 5, 6\}$. Event B is an outcome where the two dice sum to the number 7. The valid *sample space* of A and B is here tabulated in gray:

A		j						B		j					
(·, ·)		□	□	□	▣	▣	▣	+	□	□	□	▣	▣	▣	▣
i	□	(□, □)	(□, □)	(□, □)	(□, ▣)	(□, ▣)	(□, ▣)	□	2	3	4	5	6	7	7
	□	(□, □)	(□, □)	(□, □)	(□, ▣)	(□, ▣)	(□, ▣)	□	3	4	5	6	7	8	8
	▣	(▣, □)	(▣, □)	(▣, □)	(▣, ▣)	(▣, ▣)	(▣, ▣)	▣	4	5	6	7	8	9	9
	▣	(▣, □)	(▣, □)	(▣, □)	(▣, ▣)	(▣, ▣)	(▣, ▣)	▣	5	6	7	8	9	10	10
	▣	(▣, □)	(▣, □)	(▣, □)	(▣, ▣)	(▣, ▣)	(▣, ▣)	▣	6	7	8	9	10	11	11
	▣	(▣, □)	(▣, □)	(▣, □)	(▣, ▣)	(▣, ▣)	(▣, ▣)	▣	7	8	9	10	11	12	12

Throws i, j are independent, such that $\mathbb{P}(i, j) = \mathbb{P}(i)\mathbb{P}(j) = \frac{1}{6} \frac{1}{6} = \frac{1}{36}$. The probabilities are then given by summing over the number of elements in the set (the cardinalities $|A|$ and $|B|$):

$$\mathbb{P}(A) = |A| \frac{1}{36} = \frac{(4 \cdot 3)}{36} = \frac{1}{3}, \quad \mathbb{P}(A^c) = 1 - \mathbb{P}(A) = \frac{2}{3}, \quad \mathbb{P}(B) = |B| \frac{1}{6} = \frac{6}{36} = \frac{1}{6}.$$

If we want the intersection of A and B , we are left with three valid outcomes:

$$A \cap B = \{(\square, \blacksquare), (\square, \blacksquare), (\square, \blacksquare)\},$$

such that we can write:

$$\mathbb{P}(A \cap B) = \mathbb{P}(A, B) = \frac{3}{36} = \frac{1}{12} \quad \left(\neq \mathbb{P}(A)\mathbb{P}(B) \right).$$

Formally, for events A and B from the set of all possible outcomes Ω , we have:

Probability of events $A, B \in \Omega$

$$\mathbb{P}(A) \geq 0, \quad \forall A \in \Omega, \quad (2)$$

$$\mathbb{P}(\Omega) = 1, \quad (3)$$

$$A, B \text{ independent} \implies \mathbb{P}(A \cap B) = \mathbb{P}(A, B) = \mathbb{P}(A)\mathbb{P}(B), \quad \forall (A, B) \in \Omega \quad (4)$$

0.2 Conditional probability

We define $\mathbb{P}(A|B) \equiv \mathbb{P}(A, B)/\mathbb{P}(B)$ as the conditional probability of event A , given that B has already occurred. It is to be read as ‘the probability of outcome A , given outcome B ’. Everything behind the bar $|$ is the condition on A . It is one of the most fundamental concepts of statistics, but also a bit slippery and requires great care in interpretation! Particularly that $\mathbb{P}(A|B) \neq \mathbb{P}(B|A)$.

Example: Throwing dice, part 2

Continuing from the previous example, say that we are interested in the probability of $\mathbb{P}(B|A)$, thus that our dice sum up to 7, but we know that the throw ended up in the valid sample space of A . This means that our sample space is much reduced, it is only that region where A is valid:

$B A$		j		
+				
i		5	6	7
		6	7	8
		7	8	9
		8	9	10

The valid sample space of B , given A , is now reduced, and 1/4-th of its possible outcomes now generate event B ! We can find the same result in a general way using our previously found probabilities and the conditional probability rule:

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A, B)}{\mathbb{P}(A)} = \frac{\frac{1}{12}}{\frac{1}{3}} = \frac{3}{12} = \frac{1}{4}. \quad (5)$$

It is thus the *reduction of the sample space* that the probability changes – knowing that outcome A applies makes it more likely that our dice sum to 7!

We can rearrange the rule of conditional probability in three ways, which are all informative:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)}, \quad \mathbb{P}(B) = \frac{\mathbb{P}(A, B)}{\mathbb{P}(A|B)}, \quad \mathbb{P}(A, B) = \mathbb{P}(A|B)\mathbb{P}(B). \quad (6)$$

Note that for *independent* A and B , for which $\mathbb{P}(A, B) = \mathbb{P}(A)\mathbb{P}(B)$ you easily obtain:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A)\mathbb{P}(B)}{\mathbb{P}(B)} = \mathbb{P}(A). \quad (7)$$

Knowing about state B thus gives no information about state A . For example, you throw a die and flip a coin, and A tests the number of the die while B tests the side of the coin. There is then no information gained about state B when we learn about A , or vice versa.

Conditional probability of $(A, B) \in \Omega$ (definition)

$$\mathbb{P}(A|B) \equiv \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}, \quad \forall (A, B) \in \Omega. \quad (8)$$

0.3 Marginal probability

Given a set of joint probabilities $(A, B_1), (A, B_2), \dots$ with events B_n a disjoint partition (no element occurs in more than one set), we find the total chance A by summing over all sub-sets:

$$\mathbb{P}(A) = \sum_n \mathbb{P}(A, B_n). \quad (9)$$

As seen in the previous section, this may also be written in terms of conditional probabilities:

$$\mathbb{P}(A) = \sum_n \mathbb{P}(A|B_n)\mathbb{P}(B_n). \quad (10)$$

It is thus a *weighted average* of the conditional probabilities.

Example: Buying light bulbs

Assume that factory X produces 60% of all light bulbs, which work over 5000 hours in 99% of the cases, while factory Y produces 40% of all light bulbs, which work over 5000 hours in 95% of the cases. The chance that a purchased light-bulb works at least 5000 hours is then:

$$\mathbb{P}(\text{works}) = \mathbb{P}(\text{works}|F_X)\mathbb{P}(F_X) + \mathbb{P}(\text{works}|F_Y)\mathbb{P}(F_Y) = \frac{99}{100} \frac{60}{100} + \frac{95}{100} \frac{40}{100} = 97.4\%. \quad (11)$$

Indeed, the marginal probability is a weighted average of the conditional probabilities.

Example: Getting hit by cars

Assume that we know the conditional probability of being hit by a car H for three traffic light colors L , as well as the probability of observing a particular traffic light state:

			L		
			Red	Yellow	Green
$\mathbb{P}(H L)$	H	Not hit	0.99	0.9	0.2
		Hit	0.01	0.1	0.8
$\mathbb{P}(L)$			0.2	0.1	0.7

We can now compute the joint probability, $\mathbb{P}(H_p, L_n) = \mathbb{P}(H_p|L_n)\mathbb{P}(L_n)$:

		L			
$\mathbb{P}(H_p, L_n)$		Red	Yellow	Green	$\sum_n \mathbb{P}(H_p, L_n) = \mathbb{P}(H_p)$
H	Not hit	0.198	0.09	0.14	0.428
	Hit	0.002	0.01	0.56	0.572
$\sum_p \mathbb{P}(H_p, L_n) = \mathbb{P}(L_n)$		0.2	0.1	0.7	1

The bottom row and right column (the *margins* of the table!) now provide the marginal chances: $\mathbb{P}(\text{Red}) = 0.2$, $\mathbb{P}(\text{Not hit}) = 0.428$, etc., by summing over the corresponding row or column. They provide the sum of the conditional outcomes, weighed by the chance of each condition happening.

Law of total probability, when Ω is split wholly into n portions

$$\mathbb{P}(A) = \sum_n \mathbb{P}(A, B_n) = \sum_n \mathbb{P}(A|B_n)\mathbb{P}(B_n), \quad \forall A \in \Omega, \text{ and disjoint } \bigcup_n B_n = \Omega. \quad (12)$$

0.4 Bayes' rule

We may realize that intersections are identical regardless of their order ($A \cap B = B \cap A$), and find the following identical conditional probabilities:

$$\mathbb{P}(A, B) = \mathbb{P}(B, A), \quad (13)$$

$$\mathbb{P}(A|B)\mathbb{P}(B) = \mathbb{P}(B|A)\mathbb{P}(A). \quad (14)$$

Re-writing this identity provides Bayes' rule, with which we can invert conditional probabilities:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}. \quad (15)$$

Example: Drug-testing

A drug-test gives 99% true positives on drug-users and 98% true negatives on non-drug-users. Suppose that 0.5% of the population uses the drug. Then we compute the probability that a positive test really is due to a drug-user:

$$\mathbb{P}(\text{drug-user}|+) = \frac{\mathbb{P}(+|\text{drug-user})\mathbb{P}(\text{drug-user})}{\mathbb{P}(+)} = \frac{0.99 \cdot 0.005}{\mathbb{P}(+)} \quad (16)$$

We can compute $\mathbb{P}(+)$, i.e., the chance of finding a positive result regardless of whether the user actually took drugs, by taking the sum of the true and false positives. In other words, we *marginalize* over the state of the users!

$$\mathbb{P}(\text{drug-user}|+) = \frac{0.99 \cdot 0.005}{\mathbb{P}(+|\text{drug-user})\mathbb{P}(\text{user}) + \mathbb{P}(+|\text{non-drug-user})\mathbb{P}(\text{non-drug-user})}, \quad (17)$$

$$= \frac{0.99 \cdot 0.005}{0.99 \cdot 0.005 + 0.02 \cdot 0.995}, \quad (18)$$

$$= 20\%. \quad (19)$$

The test thus performs surprisingly poorly on the global population: only in 20% of the cases will a positive test result indicate that the test subject truly used the drug! But realize that for a 1000 subjects, we expect 5 users and 995 non-users. From those 995 users, $0.02 \cdot 995 \approx 20$ false positives will occur, while we will catch all five true users, $0.99 \cdot 5 \approx 5$. Out of 25 positive results, only 5 are thus genuine!

We can thus write two versions of Bayes' rule, the second one where we marginalize in the denominator over the quantity of interest.

Bayes' rule

$$\mathbb{P}(B|A) = \frac{\mathbb{P}(A|B)\mathbb{P}(B)}{\mathbb{P}(A)}, \quad \forall (A, B) \in \Omega, \quad (20)$$

$$\mathbb{P}(B_j|A) = \frac{\mathbb{P}(A|B_j)\mathbb{P}(B_j)}{\sum_n \mathbb{P}(A|B_n)\mathbb{P}(B_n)} \quad \forall A \in \Omega, \text{ and disjoint } \bigcup_n B_n = \Omega. \quad (21)$$

0.5 Indicator function

We define an indicator function $\mathbb{I}_{X=x}$, which is 1 when the statement $X = x$ is true, and 0 otherwise.













Example: Bayesian dice experiment













Two dice are thrown that sum to 9. What is the posterior distribution of the dice rolls? In other words, what dice combinations are likely, given that we know that they sum to 9?

We define event A for two dice i, j to sum to 9. We thus have the following Bayesian' rule:














$$\mathbb{P}(i, j|A) = \frac{\mathbb{P}(A|i, j)\mathbb{P}(i, j)}{\mathbb{P}(A)}. \quad (22)$$

The prior is the uniform distribution we have seen before, $\mathbb{P}(i, j) = \mathbb{P}(i)\mathbb{P}(j)$. The likelihood is expressed with an indicator function $\mathbb{P}(A|i, j) = \mathbb{I}_{i+j=9}$: after the two dice are rolled and known, they either sum to 9 or not! These two results are expressed in tabular form as:

$\mathbb{P}(i, j)$		j					
							
i		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$
		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$
		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$
		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$
		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$
		$1/36$	$1/36$	$1/36$	$1/36$	$1/36$	$1/36$

$\mathbb{P}(A i, j)$		j					
							
i		0	0	0	0	0	0
		0	0	0	0	0	0
		0	0	0	0	0	1
		0	0	0	0	1	0
		0	0	0	1	0	0
		0	0	1	0	0	0













Their point-wise multiplication leads to the following table:

$\mathbb{P}(A i, j)\mathbb{P}(i, j)$		j					
i							
		0	0	0	0	0	0
		0	0	0	0	0	0
		0	0	0	0	0	$1/36$
		0	0	0	$1/36$	0	0
		0	0	0	0	0	0
		0	0	$1/36$	0	0	0

The evidence term is found by summing over this table, giving the marginal probability:

$$\mathbb{P}(A) = \sum_{i,j} \mathbb{P}(A|i, j)\mathbb{P}(i, j) = 4/36. \quad (23)$$

The posterior is then found through Bayes' rule as:

$\mathbb{P}(i, j A)$		j					
							
i		0	0	0	0	0	0
		0	0	0	0	0	0
		0	0	0	0	0	$1/4$
		0	0	0	0	$1/4$	0
		0	0	0	$1/4$	0	0
		0	0	$1/4$	0	0	0

The posterior now says: if we know that the dice sum to 9, we are certain we threw either $(\text{3 dots}, \text{6 dots})$, $(\text{6 dots}, \text{3 dots})$, $(\text{4 dots}, \text{5 dots})$ or $(\text{5 dots}, \text{4 dots})$; all with equal probability of 25%. No other outcome is possible!

0.6 Random variables

We have, so far, used sets A and B in our probability measures, with prescribed events such as that two dice should sum up to 7. If we wanted to, conversely, get a measurement of two dice summing up to 8, we would have to define a new set C to capture those outcomes. We simplify this greatly by defining **random variables** or **stochastic variables** that use capital X to take on all valid (numerical) questions you can ask outcome space Ω . This is a very important concept!

Example: Counting heads

We flip a coin four times. We are interested in the probability of the following events:

A: exactly two of the total four flips are heads,

B: at least two of the four flips are heads,

C: the first three throws are heads.

We use a **random variable** to assign a number to each outcome in the sample space. We define two random variables here:

X: the number of heads in the four flips,

Y: the number of initial heads.

The total sample space Ω then takes on the following $2^4 = 16$ options, and we can note the random variable for each outcome:

ω				$X(\omega)$	$Y(\omega)$
1	2	3	4		
H	H	H	H	4	4
H	H	H	T	3	3
H	H	T	H	3	2
H	H	T	T	2	2
H	T	H	H	3	1
H	T	H	T	2	1
H	T	T	H	2	1
H	T	T	T	1	1
T	H	H	H	3	0
T	H	H	T	2	0
T	H	T	H	2	0
T	H	T	T	1	0
T	T	H	H	2	0
T	T	H	T	1	0
T	T	T	H	1	0
T	T	T	T	0	0

We can now answer our questions, or any variation thereof, in terms of the random variables:

$$\mathbb{P}(A) = \mathbb{P}(X = 2) = \frac{6}{16}, \quad (24)$$

$$\mathbb{P}(B) = \mathbb{P}(X > 1) = \frac{11}{16}, \quad (25)$$

$$\mathbb{P}(C) = \mathbb{P}(Y = 3) = \frac{1}{16}. \quad (26)$$

0.7 Probability mass

The random variables thus represent a *function* that maps the set of all possible outcomes into a number, $X : \Omega \rightarrow \mathbb{R}$. We could then evaluate the specific probability of a single event, which we wrote as $\mathbb{P}(X = x)$. We can extend this property and evaluate the probability for all possible outcomes: $p_X(x) = \mathbb{P}(X = x)$, called the **probability mass function**.

Example: Counting heads, continued

Continuing from the previous, we can now write down the probability mass for every specific outcome of variables X and Y , realized as x and y .

X		Y	
x	$\mathbb{P}(X = x)$	y	$\mathbb{P}(Y = y)$
0	$1/16$	0	$1/2$
1	$4/16$	1	$1/4$
2	$6/16$	2	$1/8$
3	$4/16$	3	$1/16$
4	$1/16$	4	$1/16$

We can now easily compute probabilities as $\mathbb{P}(X \geq 1) = p_X(1) + p_X(2) + p_X(3) + p_X(4) = 15/16$.

Various typical parameterized probability mass functions p_X are well-known, for example:

Typical probability mass functions

1. **Bernoulli distribution:** taking on 1 or 0 only, thus answer a yes or no question with the probability of a yes equal to p :

$$p_X(0) = 1 - p, \quad p_X(1) = p. \quad (27)$$

2. **Binomial distribution:** counts the sum of successes of n repeated Bernoulli experiments (a survey of yes/no answers, and then counting all the 'yes'es; or also $\mathbb{P}(X = x)$ in the example above for $n = 4$, $p = \frac{1}{2}$ and $k \in \{0, 1, 2, 3, 4\}$ yields $p_X(0) = 1/16, \dots$):

$$p_X(k) = \binom{n}{k} p^k (1 - p)^{n-k} \quad (28)$$

3. **Poisson's distribution:** quickly approximates a Binomial distribution for large n :

$$p_X(k) = \frac{1}{k!} \lambda^k e^{-\lambda}. \quad (29)$$

4. **Geometric distribution:** this distribution represents the number of Bernoulli experiments before a 'success' occurs (e.g., what is the probability of having 4 girls before the first boy is born?):

$$p_X(k) = p(1 - p)^{k-1}. \quad (30)$$

Probability mass function (definition)

$$p_X(x) \equiv \mathbb{P}(X = x). \quad (31)$$

0.8 Probability density

The previous section dealt with integer random variables X to represent the experimental outcome, typically counting ‘yes’ or ‘no’ answers. This breaks down for continuous outcomes where equalities $\mathbb{P}(X = x)$ don’t happen, e.g., $\mathbb{P}(\text{Temperature} = 12.0000000 \dots ^\circ\text{C}) = 0$. Therefore, we cannot evaluate similar probabilities as before. However, we can *integrate* over a **probability density** to obtain some other probabilities instead.

Example: Life expectancy, pt. 1

You can find life-expectancy tables online^a to compute the probability of dying in your n th year (column D of the table divided by 100 000). The chance of still living beyond age 100 is then given by a probability density:

$$\mathbb{P}(X > 100) = 1 - \int_0^{100} f_{\text{living til given year}}(t) dt \quad (32)$$

The answer to this is 2.1%, as confirmed by column C (divided by 100 000).

^aE.g., ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Publications/NVSR/54_14/Table01.xls

Again, some typical functions occur often:

Typical probability density functions

1. **Univariate normal distribution:** this distribution is centered on μ and has a standard deviation of σ or variance of σ^2 . The normal distribution is useful for Bayes’ rule: if the likelihood and prior are Gaussian, then so is the posterior.

$$f_X(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (33)$$

2. **Multivariate normal distribution:** we achieve a d -dimensional extension of the theory by considering a covariance matrix Σ (see further), with a d -dimensional column vector \mathbf{x} :

$$f_X(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \det \Sigma^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)}. \quad (34)$$

3. **Gamma distribution:** this distribution is parameterized by shape parameter α and rate parameter β can be written as:

$$f_X(x|\alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}, \quad (35)$$

where Γ is the gamma function.

Sometimes people write $f_X(x; a, b, \dots)$ (‘ x , parameterized by a, b, \dots ’) instead of $f_X(x|a, b, \dots)$ (‘ x , given a, b, \dots ’). One assumes parameters, the other assumes random variables; one is more accepted by mathematicians, the other by statisticians. For all practical purposes, however, they are the same!

Probability density function (definition)

$$\mathbb{P}(a \leq X \leq b) \equiv \int_a^b f_X(x) dx. \quad (36)$$

0.9 Expectation: Mean

Random variables have their means computed as the **expected value** $\mathbb{E}[X] \equiv \sum_x x \mathbb{P}(X = x)$, which is the value x weighed by the relative frequency due to $\mathbb{P}(X = x)$.

Example: Rolling dice, pt. 1

Consider a fair 6-sided die, then the expected value is defined as:

$$\mathbb{E}[X] = \sum_{n=1}^6 n \mathbb{P}(X = n) = \boxed{1} \cdot \frac{1}{6} + \boxed{2} \cdot \frac{1}{6} + \cdots + \boxed{6} \cdot \frac{1}{6} = \frac{7}{2}. \quad (37)$$

The expected value is then the average outcome if you'd roll infinitely many times!

The expected value is thus the weighted sum over all x with the probability mass function $p_X(x)$, such that more probable outcomes are better represented. Clearly, it corresponds to the *mean* or *average* value that your random variable expresses.

The expected value for the continuous case is computed using the integral: $\mathbb{E}[X] \equiv \int_{\mathbb{R}} x f_X(x) dx$.

Example: Life expectancy, pt. 2

Continuing from the previous example, we can compute the expected life expectancy of the population, using the same probabilities as we had before:

$$\mathbb{E}[X] = \int_0^{\infty} t f_{\text{living til given year}}(t) dt, \quad (38)$$

Giving as a result that the expected lifespan is 77.9 years.

Expectation / mean (definition)

$$\mathbb{E}[X] \equiv \sum_x x p_X(x), \quad (39)$$

$$\mathbb{E}[X] \equiv \int_{\mathbb{R}} x f_X(x) dx. \quad (40)$$

We note down some useful properties of the expectation operator in the box below.

Expectation, rules, for $(a, b) \in \mathbb{R}$ and $(X, Y) \in \Omega$ and f convex

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y], \quad \text{Linearity,} \quad (41)$$

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y], \quad \text{independent } X \text{ and } Y, \quad \text{Independence} \quad (42)$$

$$\mathbb{E}[g(X)] = \sum_x g(x) p_X(x), \quad \text{Law of unconscious statistician,} \quad (43)$$

$$\mathbb{E}[\mathbb{I}_A] = \mathbb{P}(A), \quad \text{Indicator expectation,} \quad (44)$$

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)], \quad \text{Jensen's inequality.} \quad (45)$$

0.10 Expectation: Variance

With the concepts and identities of the previous section, we can derive more interesting quantities. If we call the mean of our random variable $\mu = \mathbb{E}[X]$, then we can find the **variance** $\sigma \equiv \mathbb{V}(X) \equiv \mathbb{E}[(X - \mu)^2]$ as the expected squared error between the random variable and the mean.

Example: Rolling dice, pt. 2

We established the mean $\mu = \mathbb{E}(X) = \frac{7}{2}$ for throwing die. We can compute the variance with the *law of the unconscious statistician* as the following expectation:

$$\mathbb{V}(X) = \mathbb{E}[(X - \mu)^2] = \sum_{n=1}^6 (n - \mu)^2 \mathbb{P}(X = n), \quad (46)$$

$$= [1 - \frac{7}{2}]^2 \frac{1}{6} + [2 - \frac{7}{2}]^2 \frac{1}{6} + \dots + [6 - \frac{7}{2}]^2 \frac{1}{6} = \frac{35}{12}. \quad (47)$$

The following simplification is possible (note that $\mathbb{E}[X]$ is a uniform constant!):

$$\mathbb{V}(X) = \mathbb{E}[(X - \mathbb{E}(X))^2] = \mathbb{E}[X^2 - 2X\mathbb{E}[X] + \mathbb{E}[X]^2] \quad (48)$$

$$= \mathbb{E}[X^2] - 2\mathbb{E}[X]\mathbb{E}[X] + \mathbb{E}[X]^2, \quad (49)$$

$$= \mathbb{E}[X^2] - \mathbb{E}[X]. \quad (50)$$

Variance

$$\sigma(X) \equiv \mathbb{V}(X) \equiv \mathbb{E}[(X - \mathbb{E}(X))^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathbb{E}[X^2] - \mu^2. \quad (51)$$

0.11 Expectation: Covariance

We generalize the previous section to the **covariance**, which measures the association between two random variables X and Y as $\mathbb{C}(X, Y) = \mathbb{E}[(X - \mu_x)(Y - \mu_y)]$. Note that the variance is thus a special case of the covariance with itself, i.e., $\mathbb{V}(X) = \mathbb{C}(X, X)$. The covariance is positive when the two quantities are positively correlated, and negative when the quantities are negatively correlated.

Example: Correlation of two dice experiments

The equation to express Pearson's correlation coefficient is the normalized covariance:

$$\text{Cor}(X, Y) = \frac{\mathbb{C}(X, Y)}{\sqrt{\mathbb{V}(X)\mathbb{V}(Y)}}. \quad (52)$$

If we throw two dice and X counts the sum of the two dice while Y counts the product of the two dice, we can find (e.g., with MATLAB) that $\mu_x = 7$ and $\mu_y = \frac{49}{4}$, that $\mathbb{V}(X) = \frac{35}{6}$ and $\mathbb{V}(Y) = \frac{11515}{144}$. Finally, the covariance is $\mathbb{C}(X, Y) = \sum_{x,y} xy\mathbb{P}(X = x, Y = y) - \mu_x\mu_y = \frac{245}{12}$. The correlation is then $\text{Cor}(X, Y) = \frac{121}{128}$. Note that the covariance is computed with a joint probability distribution $\mathbb{P}(X = x, Y = x) \neq \mathbb{P}(X = x)\mathbb{P}(Y = y)$!

Co-variance

$$\mathbb{C}(X, Y) \equiv \mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[XY] - \mu_x\mu_y. \quad (53)$$

0.12 Expectation: Covariance matrix

We previously encountered the covariance matrix in the definition of a multivariate Gaussian. We can now fully define this **symmetric** ($\mathbb{C}(X, Y) = \mathbb{C}(Y, X)$) and **positive semi-definite** matrix as $\Sigma_{ij} = \mathbb{C}(X_i, X_j)$. The covariance matrix for 2 random variables, $X_1 = X$ and $X_2 = Y$, is:

$$\Sigma = \begin{bmatrix} \mathbb{C}(X, X) & \mathbb{C}(X, Y) \\ \mathbb{C}(Y, X) & \mathbb{C}(Y, Y) \end{bmatrix} = \begin{bmatrix} \mathbb{V}(X) & \mathbb{C}(X, Y) \\ \mathbb{C}(Y, X) & \mathbb{V}(Y) \end{bmatrix}. \quad (54)$$

MATLAB's std, var and cov functions

MATLAB is in the habit of returning the *unbiased* variance and covariance matrix through the so-called 'Bessel correction' factor $\frac{N}{N-1}$ for N the length of the vectors (e.g., X). This is relevant when you only have the sample mean and not the population mean – this is not the case in the examples we do here! Anyhow, the MATLAB built-in functions return:

$$\text{std}(X) = \sqrt{\frac{N}{N-1} \mathbb{V}(X)}, \quad (55)$$

$$\text{var}(X) = \frac{N}{N-1} \mathbb{V}(X), \quad (56)$$

$$\text{cov}(X, Y) = \frac{N}{N-1} \begin{bmatrix} \mathbb{C}(X, X) & \mathbb{C}(X, Y) \\ \mathbb{C}(Y, X) & \mathbb{C}(Y, Y) \end{bmatrix} \quad (57)$$

If you are interested in Σ as defined here, you must evaluate it as $(N-1)*\text{cov}(X, Y)/N$.

Python's np.std, np.var and np.cov functions

Python's numpy takes a less unified approach and modifies only the entire covariance matrix:

$$\text{np.std}(X) = \sqrt{\mathbb{V}(X)}, \quad (58)$$

$$\text{np.var}(X) = \mathbb{V}(X), \quad (59)$$

$$\text{np.cov}(X) = \frac{N}{N-1} \begin{bmatrix} \mathbb{C}(X, X) & \mathbb{C}(X, Y) \\ \mathbb{C}(Y, X) & \mathbb{C}(Y, Y) \end{bmatrix}. \quad (60)$$

0.13 Conditional expectations

We finalize the session on expectations by stating a conditional expectation $\mathbb{E}[X|B]$, as well as the marginalized expected value for summing up expected values.

Conditional expectation

$$\mathbb{E}_{X|B}[X] \equiv \mathbb{E}[X|B] \equiv \sum_x x \mathbb{P}(X = x|B). \quad (61)$$

Partition theorem for expectations

$$\mathbb{E}[X] = \sum_i \mathbb{E}[X|B_i] \mathbb{E}[B_i]. \quad (62)$$

0.14 I.I.D. sampling of random variables

Generally, our data are *realizations* from the possible sample space. When we repeatedly draw samples from the possible sample space, we can make a number of simple theoretical observations when we assume our samples are **i.i.d.: independent and identically distributed**:

1. Independent: *the order in which you draw the samples makes no difference*,
2. Identically distributed: *the samples are drawn from the same probability distribution*.

We already know the mathematical implication of ‘independence’, and the ‘identical distribution’ implies that we can re-use a single probability density every time.

Example: Throwing independent and identical dice

Throwing n fair die yields a probability for any unique ordered outcome $\mathcal{X} = \{x_1, \dots, x_n\}$:

$$\mathbb{P}(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = \prod_{i=1}^n \mathbb{P}(X_i = x_i) = \frac{1}{6^n}. \quad (63)$$

0.15 Sampling from a distribution & mean of iid samples

We formally denote the sampling operation using the \sim symbol such that we can, e.g., say that a random variable X is drawn from the Bernoulli distribution as: $X \sim \text{Bernoulli}(p)$.

Example: Flipping a coin

Defining X to denote if the coin shows heads, we know:

$$X \sim \text{Bernoulli}(1/2). \quad (64)$$

Defining Y to count the number n of heads before a tail occurs, we know:

$$Y \sim \text{Geometric}(1/2). \quad (65)$$

Now define $\bar{X}_n = (X_1 + X_2 + \dots + X_n)/n$ as the **sample mean**. If we assume that X_i are sampled iid, and $\mathbb{E}[X_i] = \mu$ and $\mathbb{V}[X_i] = \sigma^2$, we can then see that the expectation returns the true mean:

$$\mathbb{E}[\bar{X}_n] = \frac{1}{n} \mathbb{E}[X_1 + X_2 + \dots + X_n] = \frac{1}{n} (\mu + \mu + \dots + \mu) = \mu, \quad (66)$$

and the variance of this estimation reduces inversely with n :

$$\mathbb{V}(\bar{X}_n) = \frac{1}{n^2} \mathbb{V}(X_1 + X_2 + \dots + X_n) = \frac{1}{n^2} (\sigma^2 + \sigma^2 + \dots + \sigma^2) = \frac{\sigma^2}{n}. \quad (67)$$

The **central limit theorem** tells us that for a large enough n , we have:

$$\bar{X}_n \sim \mu + \frac{\mathcal{N}(0, \sigma^2)}{\sqrt{n}} \quad (68)$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal/Gaussian distribution with mean μ and variance σ^2 . This property is widely used in hypothesis testing, because a good number of (controlled) samples can be considered as ‘draws’ from this distribution, which we use to estimate μ and σ – and then we can see how much a new (uncontrolled) sample is located with respect to this ‘normal’!

0.16 Formal summary

We formally summarize the above section in more abstract terms. Probability is the mathematical language for quantifying uncertainty. We consider this in terms of testing outcomes through ‘experiments’. For the notation we will use set-theory, for which $i \in \{A \cap B\}$ means element i is present in both sets A and B . The expression, $i \in \{A \cup B\}$ is an element in A or B (or both). Furthermore, the expression $i \in \{\Omega \setminus A\}$ represents an element i that is in Ω but *not* in A . The notation \emptyset represents the empty set, e.g., $\Omega \setminus \Omega = \emptyset$. Finally, the cardinality of a set $|A|$ counts all the elements in the set.

Definition 1 (Sample spaces and events)

The *sample space* $\Omega = \{\omega_1, \omega_2, \dots\}$ represents all possible outcomes of the experiment. Points ω in Ω are called the *sample outcomes* or *realizations*. Exactly *one* of the outcomes actually occurs. *Events* $A \subseteq \Omega$ are subsets of Ω , thus may consist of multiple outcomes ω .

Example 1 (Tossing coins) If we toss two coins, then $\Omega = \{HH, HT, TH, TT\}$. The event that the first toss is heads is $A = \{HH, HT\}$. An actual realization may be $\omega_3 = \{TH\}$, such that A is false.

Theorem 1 (Basic axioms)

1. For any event A , $\mathbb{P}(A) \geq 0$.
2. For the entire sample space, $\mathbb{P}(\Omega) = 1$.
3. For disjoint A_1, A_2, \dots : $\mathbb{P}(\bigcup_{i=1}^{\infty} A_i) = \mathbb{P}(A_1) + \mathbb{P}(A_2) + \dots = \sum_{i=1}^{\infty} \mathbb{P}(A_i)$.
 \Rightarrow For $A^c = \Omega \setminus A$ (disjoint!), it holds that $\mathbb{P}(A) + \mathbb{P}(A^c) = 1 \iff \mathbb{P}(A^c) = 1 - \mathbb{P}(A)$.
 \Rightarrow For $A \subseteq B$ we have $\mathbb{P}(A) \leq \mathbb{P}(B)$.
 \Rightarrow For multiple A_1, A_2 we have $\mathbb{P}(A_1 \cup A_2) = \mathbb{P}(A_1) + \mathbb{P}(A_2) - \mathbb{P}(A_1 \cap A_2)$.

Definition 2 (Marginals) Given a joint probability distribution $\mathbb{P}(A, B)$ the distribution of a single variable is given by:

$$\mathbb{P}(X) = \sum_y \mathbb{P}(X, Y). \quad (69)$$

Definition 3 (Conditional probability) We define the conditional probability of X given Y :

$$\mathbb{P}(X|Y) \equiv \frac{\mathbb{P}(X, Y)}{\mathbb{P}(Y)}. \quad (70)$$

Definition 4 (Bayes’ rule) Following the definition of conditional probability we find:

$$\mathbb{P}(X|Y) = \frac{\mathbb{P}(Y|X)\mathbb{P}(X)}{\mathbb{P}(Y)}. \quad (71)$$

Definition 5 (Independence) If the state of X gives no information about Y and vice versa:

$$\mathbb{P}(X, Y) = \mathbb{P}(X)\mathbb{P}(Y). \quad (72)$$

Definition 6 (Conditional independence) If state X and Y are independent of each other, provided that we know the set of Z . Then we obtain:

$$\mathbb{P}(X, Y|Z) = \mathbb{P}(X|Z)\mathbb{P}(Y|Z). \quad (73)$$

Definition 7 (Random variable (rv)) Any countable outcome of an experiment is a random variable, which can take on various numerical values. Often abbreviated as *rv*.

Definition 8 (Probability mass function) We assess the behaviour of a random variable for all the possible discrete states it can take on:

$$p_X(x) = \mathbb{P}(X = x). \quad (74)$$

Definition 9 (Probability density) If the states are not discrete but continuous, we may integrate over a range of probability density:

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f_X(x) dx. \quad (75)$$

We use the integrable probability function f_X which is related to the cumulative distribution function F_X through the following two relations:

$$F_X(x) = \int_{-\infty}^x f_X(u) du, \quad f_X = \frac{d}{dx} F_X(x). \quad (76)$$

Definition 10 (Expectation) The expectation operator takes the sum over all possible values weighted by their associated probabilities for discrete or continuous data:

$$\mathbb{E}[X] = \sum_x x \mathbb{P}(X = x), \quad \mathbb{E}[X] = \int_{\mathbb{R}} x f_X(x) dx. \quad (77)$$

Definition 11 (Moments of expectation) The three important relations of expectation are the mean:

$$\mathbb{E}[X] = \mu_x, \quad (78)$$

the variance:

$$\mathbb{V}(X) = \mathbb{E}[(X - \mu_x)^2] = \mathbb{E}[X^2] - \mu_x^2, \quad (79)$$

and the covariance:

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mu_x)(Y - \mu_y)] = \mathbb{E}[XY] - \mu_x \mu_y. \quad (80)$$

Definition 12 (Indicator function) We define the indicator function as:

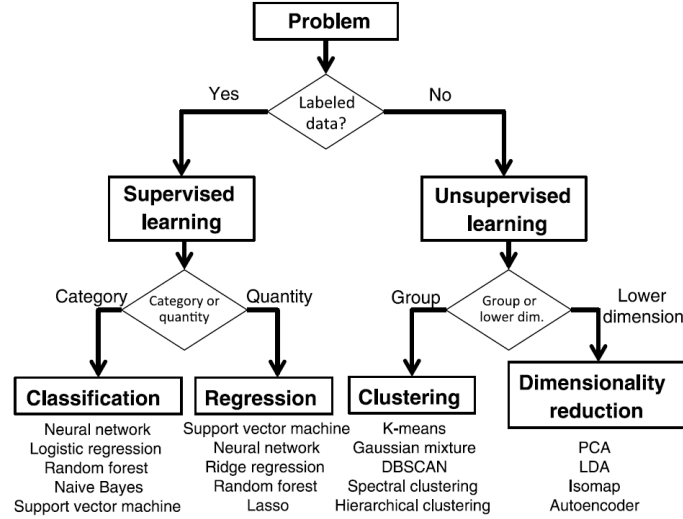
$$\mathbb{I}_A = \begin{cases} 1 & \text{if } A \text{ is true,} \\ 0 & \text{otherwise,} \end{cases} \quad (81)$$

which selects only the elements of relevance as, e.g.:

$$\mathbb{E}[\mathbb{I}_{X=x_0}] = \sum_x \mathbb{I}_{X=x_0} \mathbb{P}(X = x) = \mathbb{P}(X = x_0). \quad (82)$$

1 Machine learning 101: Measuring accuracy

Computer science used to be about writing a program that turns inputs into outputs; but now is about finding a program *given* the inputs and outputs, in particular about optimally traversing the space of possible programs efficiently.



1.1 The formalized set-up

Imagine a data-set with samples and their label:

$$\mathcal{D} = \left\{ (\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N) \right\} \subseteq \mathcal{X} \times \mathcal{Y} \quad (83)$$

where \vec{x}_i is the i -th **feature vector** from space \mathcal{X} , and y_i the corresponding **label** from space \mathcal{Y} . (\mathcal{Y} could represent k classes $\{0, 1, \dots, k\}$ when we do a classification problem, or all real numbers \mathbb{R} for a regression problem). We assume that the pairs are drawn i.i.d. from some distribution,

$$(\vec{x}_i, y_i) \sim P, \quad (84)$$

but this distribution is unknown to us!

Then we require a program (or ‘hypothesis’), referred to as $h(x)$ or $f(x)$ that maps \vec{x}_i into y_i . We assume that the hypothesis h is drawn from its own class $h \in \mathcal{C}$. Examples of the space \mathcal{C} are decision trees, linear classifiers, artificial neural networks, support vector machines, \dots . The optimal algorithm is a problem-dependent feature. It often pays off to try a large number of methods (the ‘**No Free Lunch theorem**’ states that there is no single optimal problem-independent algorithm).

No Free Lunch theorem

The No Free Lunch (NFL) theorem states that there is no single algorithm that works optimally on *all* problems. The message of NFL is that it tells us choosing an appropriate algorithm requires making assumptions about the kinds of target functions the algorithm is being used for. With no assumptions, no ‘meta-algorithm’ performs better than random choice! In our case, it thus means that we should try a variety of models to see which model works optimally for our problem; or think very carefully which algorithm best fits the problem at hand!

1.2 Loss functions & generalization

To be able to distinguish between different hypotheses h , we require a scoring function that tells us ‘how well did my hypothesis perform for my data?’. This is provided by a **loss function** L , which tells us how *badly* we predict a given set of samples.

Loss-functions

1. **0-1 loss**. This loss function counts the number of wrong predictions:

$$L(h, \mathcal{D}) = \sum_{i=1}^N \mathbb{I}_{h(\vec{x}_i) \neq y_i}. \quad (85)$$

2. **Squared loss**. For regression, it is more interesting to see how far off the true answer you are, which is typically measured with a squared loss function, weighing large errors extra:

$$L(h, \mathcal{D}) = \sum_{i=1}^N |h(\vec{x}_i) - y_i|^2. \quad (86)$$

3. **Absolute loss**. Rather than the L^2 -norm of the squared loss, we can consider an L^1 -norm:

$$L(h, \mathcal{D}) = \sum_{i=1}^N |h(\vec{x}_i) - y_i|. \quad (87)$$

In this setting, large errors will be less amplified – you don’t want the squared loss measure if large errors or statistical flukes are present in the training data (when big outliers are squared, they dominate the loss function’s score fully, even if you do well on the rest!).

We realize that loss-functions are **non-negative**, and the optimal solution lies at $L(h, \mathcal{D}) = 0$. However, note the following. If we train our data on a set of training data $\mathcal{Z}^{\text{train}} \subset \mathcal{D}$, and have the following simple hypothesis that just memorizes the input data:

$$h^\bullet(\vec{x}) = \begin{cases} y_i & \text{if } \vec{x} = \vec{x}_i, \\ 0 & \text{otherwise,} \end{cases} \quad (88)$$

it will have a loss-function $L(h^\bullet, \mathcal{Z}^{\text{train}}) = 0$, yet it will be terrible on any new inputs \vec{x} not in the training data! Memorization would thus give a perfect loss function, but would not generalize at all! This is the problem that we must address: how do we design machine learning algorithms that generalize to *all* samples drawn from P , not just the training samples!

Example: Generalization error for generals

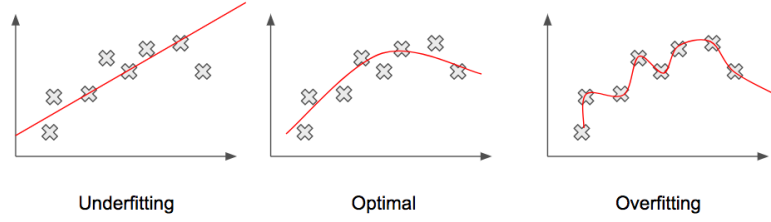
KILIAN WEINBERGER tells the humorous story of an image classifier that separated military from civilian vehicles, working really well on all collected data. But then it didn’t work in practice at all! What happened was that all the military vehicles were photographed during the day, while the civilian vehicles were photographed much later in the day. Thus, the classifier simply learned that bright pictures were military vehicles, and the dark pictures were civilians! Clearly, the samples and reality were not drawn from a similar distribution P !

1.3 Risk functions & generalization

So, we want to obtain a hypothesis h that maps all relevant data inputs \mathcal{X} to solutions \mathcal{Y} . However, h can only be trained on a subset $\mathcal{Z}^{\text{train}} \subset \mathcal{D}$, thus the difficulty lies in *generalization*: how can we be sure that our hypothesis fits not only the training data but also new unseen data.

Example: Underfitting and overfitting

The goal in machine learning is to find a function that works well on *all* relevant inputs, while only having a limited number of (noisy) training data available. Therefore, our optimal solution lies somewhere between an underfitting and overfitting solution:



Assume that $\mathcal{Z}^{\text{train}}$ is drawn i.i.d. from the distribution P . Then choose a loss function $L(h, \mathcal{D})$. We then define an **Expected risk function** that expresses the expected loss for *all* data \mathcal{D} :

$$R(h, \mathcal{D}) = \mathbb{E}[L(h, \mathcal{D})]. \quad (89)$$

As an example, with the 0-1 loss-function we would obtain:

$$R(h, \mathcal{D}) = \mathbb{E}[\mathbb{I}_{\hat{y}_{\text{predicted}} \neq y_{\text{true}}}], \quad (90)$$

However, we can only compute the **Empirical risk function** (written with \hat{R}) based on the N training data $\mathcal{Z}^{\text{train}}$, which simply equals the loss function equally weighted for all available data:

$$\hat{R}(h, \mathcal{Z}^{\text{train}}) = \frac{1}{N} L(h, \mathcal{Z}^{\text{train}}). \quad (91)$$

As an example, with the 0-1 loss-function we would obtain:

$$\hat{R}(h, \mathcal{Z}^{\text{train}}) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{h(x_i) \neq y_i}. \quad (92)$$

If we just choose the function h that minimizes the *empirical risk function*, we're prone to obtain a hypothesis that is overfitting or just memorizes all the inputs. Often it holds that the optimal model $\hat{h} \in \arg \min_h \hat{R}(h, \mathcal{Z}^{\text{train}})$ based on the training data overestimates the performance on all the data:

$$\hat{R}(\hat{h}, \mathcal{Z}^{\text{train}}) < R(\hat{h}, \mathcal{D}). \quad (93)$$

Expected and Empirical Risk Functions

$$R(h, \mathcal{D}) = \mathbb{E}(L(h, \mathcal{D})), \quad (94)$$

$$\hat{R}(h, \mathcal{Z}^{\text{train}}) = \frac{1}{N} L(h, \mathcal{Z}^{\text{train}}). \quad (95)$$

1.4 Generalization error & holdout strategy

Clearly, with only knowledge of the training data, it appears impossible to gain an understanding of the expected risk. However, an estimation of the risk can be made by splitting the available data in two. We keep a $N - M$ instances in the training data-set $\mathcal{Z}^{\text{train}} = (\mathcal{X}^{\text{train}} \times \mathcal{Y}^{\text{train}})$, and retain the remaining M instances in a test data-set $\mathcal{Z}^{\text{test}} = (\mathcal{X}^{\text{test}} \times \mathcal{Y}^{\text{test}})$ which is held-out and only used to validate the results of the trained model, to compute the **test error**, for example with the 0-1 loss:

$$\hat{R}(\hat{h}, \mathcal{Z}^{\text{test}}) = \frac{1}{M} \sum_{m=1}^M \mathbb{I}_{\hat{h}(x_m) \neq y_m}. \quad (96)$$

The test data-set should **never** be used within the training algorithm! The test error then gives an idea of the level of underfitting or overfitting, as it gives an estimate of the generalization error. Ideally, the model is tested only once on the test data – for if one keeps tweaking the model to improve the score on the test data-set we are, in a complicated way, fitting our model to the test data-set as well! Therefore, it often also holds that the trained hypothesis $\hat{h} \in \arg \min \hat{R}(h, \mathcal{Z}^{\text{train}})$ still overestimates the expected risk, but is better already.

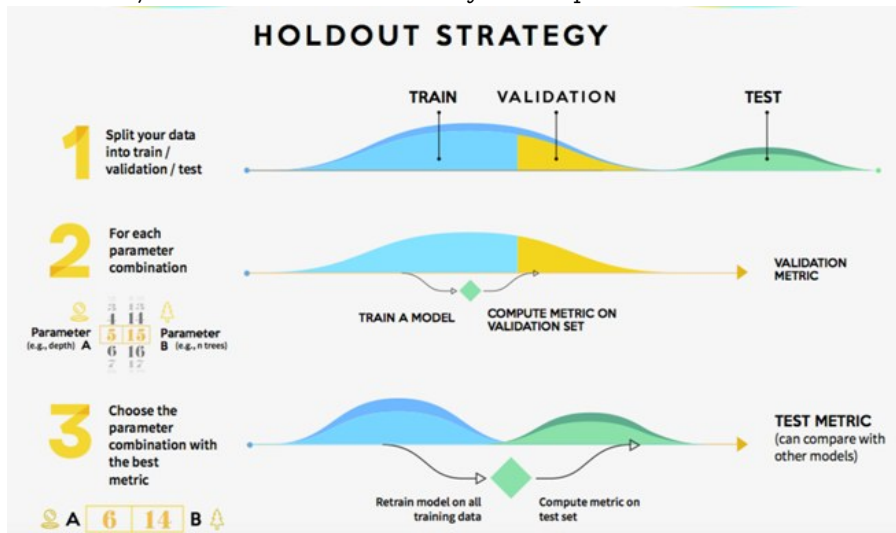
$$\hat{R}(\hat{h}, \mathcal{Z}^{\text{test}}) < R(\hat{h}, \mathcal{D}). \quad (97)$$

Holdout strategy: Training, validation and test data

We established that if we choose the optimal hypothesis for our training data:

$$\hat{h} \in \arg \min_h \hat{R}(h, \mathcal{Z}^{\text{train}}), \quad (98)$$

we likely overfit and have a poor test-score, $\hat{R}(\hat{h}, \mathcal{Z}^{\text{test}})$. Then how to fine-tune our model? Well – we create *another* split of the data: the **validation data**. This is equally held out during training like the test data, but is used to score various models immediately after the training; and the set of model ‘hyper-parameters’ that optimize the **holdout score** become a good approximation of which model is most appropriate. Retraining on *all* data with those optimal parameters then gives the final model, for which the test error may be computed.



2 Machine learning 102: Classification

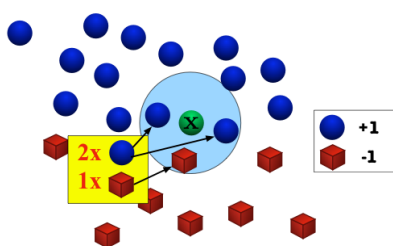
Before further statistical notions, an overview of the machine learning methods seems appropriate.

2.1 k-Nearest neighbours

The first method worth looking at is the (k -)nearest neighbour algorithm. The assumption is simple: similar inputs must have similar outputs. The the k-NN algorithm, for any test input \vec{x} , assigns it the most common label y amongst its k most similar training inputs.

Example: An illustration in 2-D

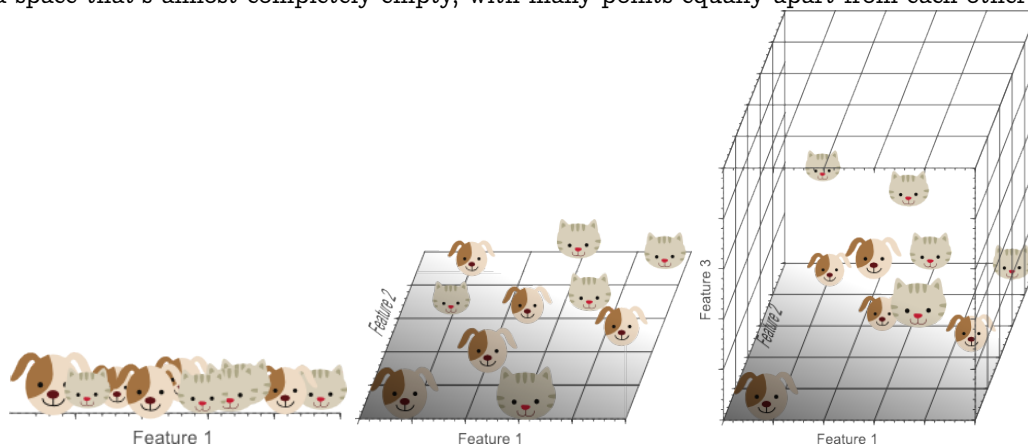
We have a training data-set $\mathcal{Z}^{\text{train}}$ consisting of blue (+1) and red (-1) points. A new point \vec{x} is compared against the nearest $k = 3$ points, and results in a majority predicting the label blue (+1), which we may assign to \vec{x} correspondingly:



The positive aspect of the k-NN is its simplicity and meaningful notion of similarity learning. The downsides are that its slow and suffers from the *curse of dimensionality*.

The curse of cats and dogs and dimensionality

Putting 100 samples on a unit line $x \in [0, 1]$ means that any random new point lies close to many other points – it's busy! However, placing 100 samples in a d -dimensional space $\vec{x} \in [0, 1]^d$ results in a space that's almost completely empty, with many points equally apart from each other!



A small 256×256 black-and-white image already has already 65 536 unique pixel *features* – thus suffering greatly from the curse of dimensionality. It is thus not very useful for ‘big data’, because the algorithm becomes slow and the whole concept of closeness starts to break down!

2.2 Separation by hyperplane

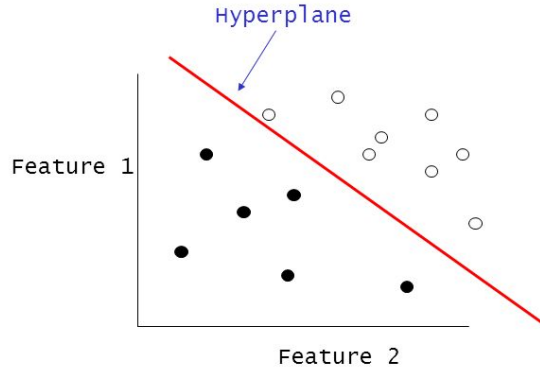
The second perspective is to assume that we can separate our data using a linear ‘hyperplane’. For example, the line that describes a linear plane is given by:

$$w_0 + w_1 \text{Feature 1} + w_2 \text{Feature 2} = 0 \implies \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 \\ \text{Feature 1} \\ \text{Feature 2} \end{bmatrix} = 0. \quad (99)$$

Our classification task thus really comes down to finding the appropriate weights \vec{w} .

Example: An illustration in 2-D

Assuming the presence of 2 features, and 2 classes (black and white), we observe that the following data is easily classified by realizing that either class lies on only one side of the classifier line:



We can describe the red line with a simple equation:

$$\text{Feature 1} = \text{offset} - \text{slope} \times \text{Feature 2}, \quad (100)$$

which could also be parameterized as:

$$\begin{bmatrix} -\text{offset} & 1 & \text{slope} \end{bmatrix} \begin{bmatrix} 1 \\ \text{Feature 1} \\ \text{Feature 2} \end{bmatrix} = 0. \quad (101)$$

We choose **normalized coefficients** $\vec{x}^T = [1 \text{ Feature 1 Feature 2}]$, such that we can write the entire hyperplane of Eq. (99) with the following efficient notation:

$$\vec{w}^T \vec{x} = 0, \quad (102)$$

for which the classifier is defined in the following way:

$$h(\vec{x}_i) = \begin{cases} \vec{w}^T \vec{x}_i \geq 0 & \text{white class,} \\ \vec{w}^T \vec{x}_i < 0 & \text{black class.} \end{cases} \quad (103)$$

Note that the concept easily generalizes to higher dimensions which are, perhaps, more difficult to imagine. One simply has the homogenized $\vec{x} \in \mathcal{X}^{d+1}$ for d features, and correspondingly $d + 1$ weights in \vec{w} as well. This describes a plane for 3-D data, etc.

2.2.1 Perceptron

The perceptron (ROSENBLATT, 1957) is the first machine-learning algorithm that could learn to find a hyperplane based on the given data. We assume binary classes, positioned on the positive or negative side of the hyperplane:

$$\mathcal{Y} = \{-1, +1\}. \quad (104)$$

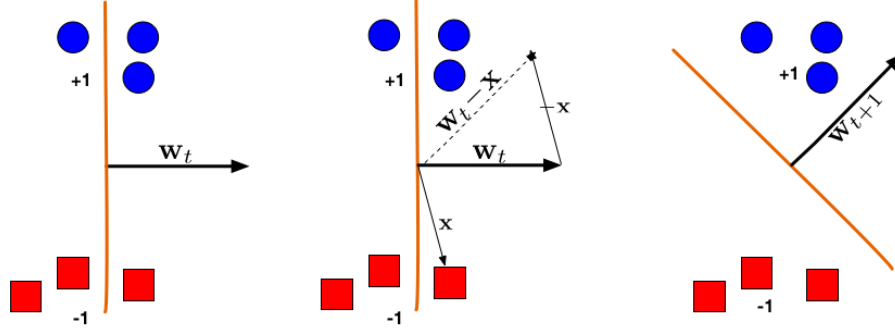
The perceptron then initializes with a random hyperplane parameterization \vec{w} , takes a misclassified sample (\vec{x}_i, y_i) (found when the sample has: $y_i \vec{w}^T \vec{x}_i \leq 0$, because when our hyperplane is correct, we have $++ \geq 0$ or $-- \geq 0$) and we add the required correction to the hyperplane in the following way:

$$\vec{w}_{\text{updated}} = \vec{w} + y_i \vec{x}_i. \quad (105)$$

Iterating these updates always converges to a separating hyperplane, if the data is linearly separable!

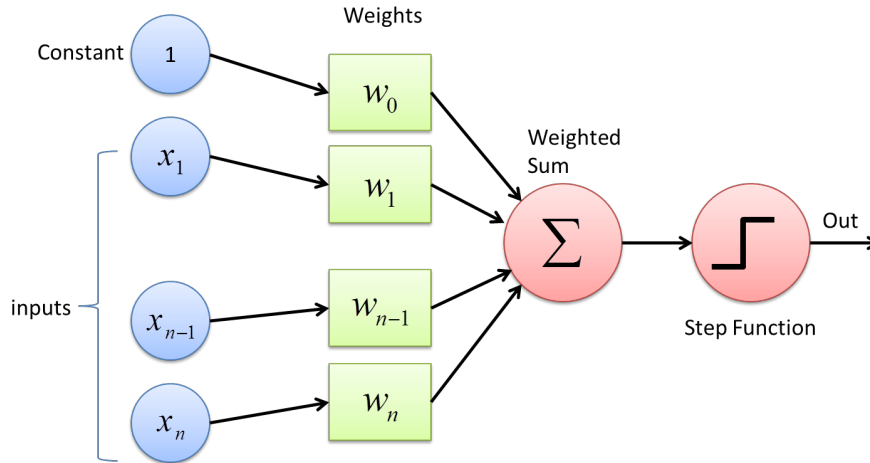
Example: Graphical example in 2-D

Geometrically, the misclassified red box gives an update $y_i \vec{x}_i = -\vec{x}_i$, which really just means that we rotate the separating line \vec{w} such that it doesn't misclassify \vec{x}_i anymore:



Illustrating neural networks as graphs

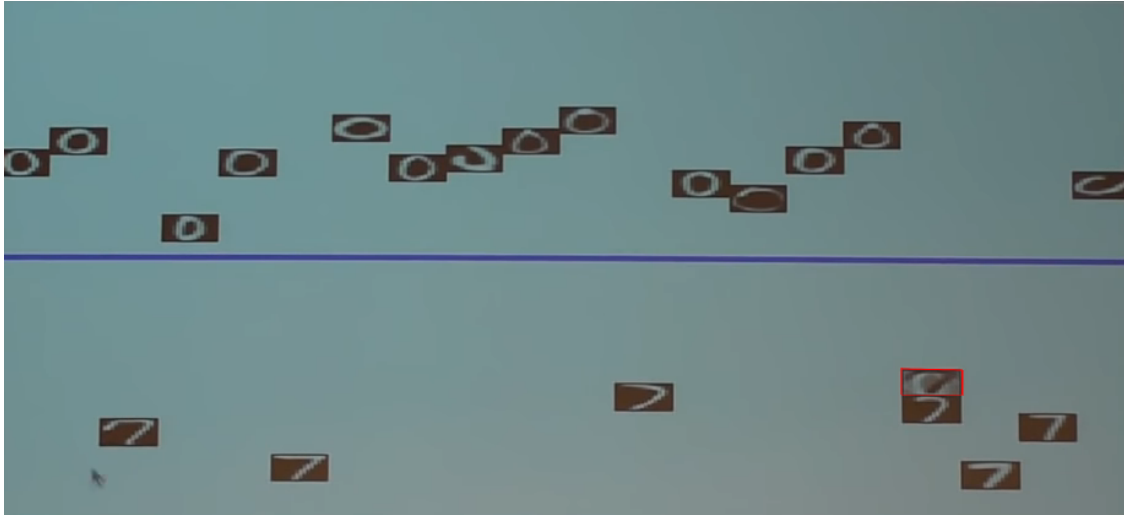
The resulting classifier is sometimes illustrated with a graph as follows:



Note that the figure really just expresses this mathematical operation: $\text{sign}(\vec{w}^T \vec{x})$. For a given homogenized feature vector \vec{x} and weights \vec{w} , we compute the 'weighted sum' (the scalar product $\vec{w}^T \vec{x}$), and a step function is used to see whether the outcome is positive or negative ($\text{sign}(\vec{w}^T \vec{x})$).

Example: Handwritten number recognition

The method appeared to work rather well for recognition of handwriting also, considering every pixel as an entry of the vector \vec{x} , which multiplied with the weights \vec{w} separate a 0 from a 7:



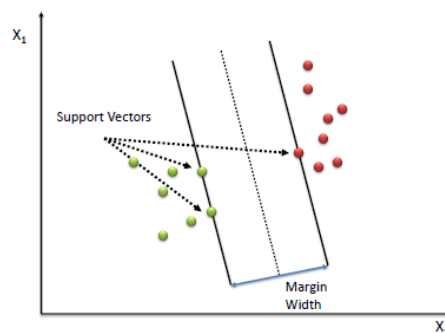
The weights are illustrated in red outline, on top of the '7' fourth of the right. You see that you can discern a white 0 and a black 7 in the weight. Thus multiplying a white zero image with the white '0' weight, and summing over the result, gives a positive sum; while multiplying a white 7 with the black '7' weight sums to a negative result. Thus: the two data are separated by their score after summation between image and weight!

2.2.2 Support vector machines (SVM)

The perceptron algorithm always converges, but its result is different every time, and there is no guarantee that it finds the best separating hyperplane. The support vector machine (VAPNIK & CHERVONENKIS, 1963) tries to obtain the separating hyperplane that *maximizes* the distance ('margin') between the two clusters. This method makes sure we always find the same classifier based on the data, and furthermore generalize as well as possible!

Support vectors

In practice, it means that that our separator is defined by just 3 data-points: 2 from one class and 1 from the other. These three points are called the 'support vectors'. The line exactly in-between the parallel lines through the support vectors gives the separator with the largest possible margin:



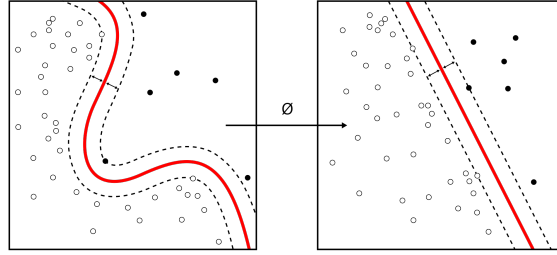
2.2.3 The kernel trick

The ability to make linear separation boundaries only is rather limiting. But extending the previous methods to other separation boundaries is difficult – the methods are so inherently based on ideas of linearity. The solution that came about is the ‘Kernel method’, by which the data is transformed into another space, after which the data becomes linearly separable. Rather than finding the hyperplane $\vec{w}^T \vec{x}$ we require the optimal hyperplane for the transformed data:

$$\vec{w}^T \phi(\vec{x}) = 0. \quad (106)$$

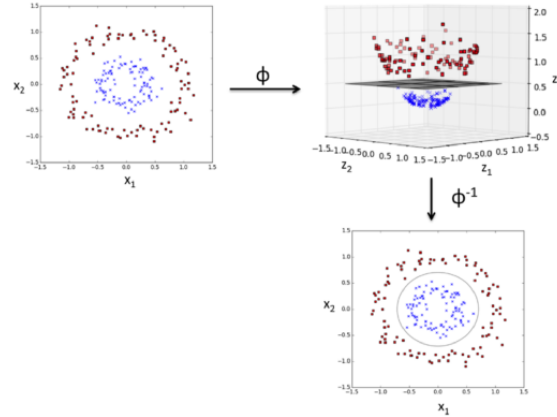
Example: Non-linear transformation (1)

We may simply transform the data, after which they become linearly separable (on the right):



Example: Non-linear transformation (2)

Sometimes, the data is better separable in dimensions higher than the original input space:



The crucial portion of the kernel trick is that we tend require the scalar product $\vec{x}^T \vec{x}$ to find the optimal support vector machine solution. The transformed version is:

$$\phi(\vec{x})^T \phi(\vec{x}) \in \mathbb{R}. \quad (107)$$

Note that ϕ may transform \vec{x} into many dimensions, such that the above scalar product is very expensive to evaluate! However, if we find a closed-form solution to compute this scalar product, we have a *kernel* that never actually has to expand to higher dimensions:

$$K(\vec{x}) = \phi(\vec{x})^T \phi(\vec{x}). \quad (108)$$

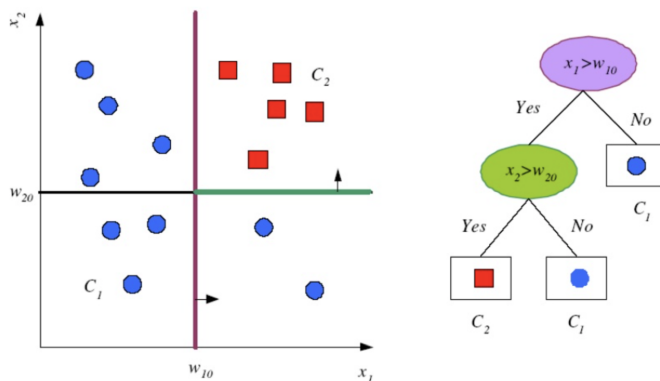
This allows us to obtain the scalar-product immediately through $K(\vec{x})$! Effectively, as long as you choose an appropriate kernel (Gaussian radial basis functions are the most popular one, which transform into an *infinite*-dimensional space!) you can now also fit non-linear clusters extremely rapidly with your linear algorithm.

2.3 Decision Trees & Ensemble models

Why use only *one* hyperplane? This is the idea behind *decision trees*, which can be trained and evaluated rapidly, which makes them interesting in practice! We obtain a classifier which goes through a binary tree, testing at every node whether some feature is above or below a given value.

Example: Decision tree

The figure below shows a decision tree (on the right) that classifies the data (on the left) perfectly.

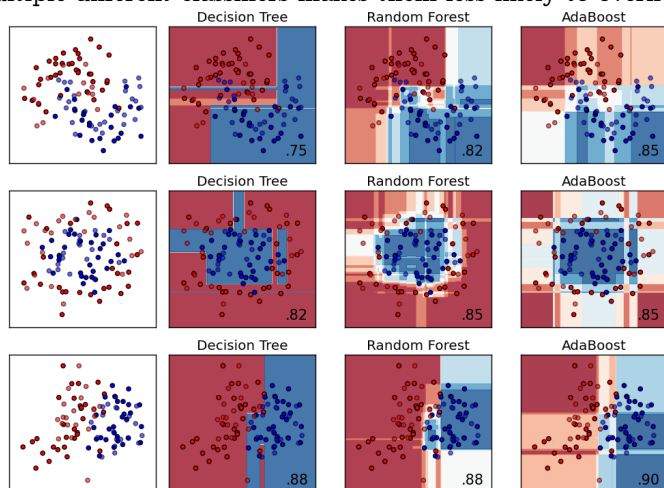


Rather than using on a single decision tree, it often pays off to use **ensemble methods**. In such a case we often train multiple decision trees, and classify by a (weighted) majority vote:

$$y_i = \text{sign} \left(\alpha_1 h_1(\vec{x}_i) + \alpha_2 h_2(\vec{x}_i) + \dots \right). \quad (109)$$

Example: Ensemble methods

In the example below, you see a Random Forest (training many deep decision trees) and AdaBoost (training many one-level decision trees) classifying data, with the right-bottom showing the accuracy (higher is better). These ensemble methods outperform the simple decision tree, because pooling multiple different classifiers makes them less likely to overfit.



2.4 Probabilistic models

The final perspective takes a probabilistic approach. Assume that the data $\mathcal{Z}^{\text{train}}$ can in some way be explained by a distribution parameterized with θ_j (one for each class j). The machine learning problem now reduces to finding the appropriate θ_j that explains the data. Compared to the previous approaches, we now actually obtain the probability of any new point belonging to either of the classes!

Really, all three methods below are just approximations of the Bayes' solution, $\mathbb{P}(\theta|\mathcal{Z}^{\text{train}}) = \mathbb{P}(\mathcal{Z}^{\text{train}}|\theta_j)\mathbb{P}(\theta_j)/\mathbb{P}(\mathcal{Z}^{\text{train}})$.

2.4.1 Maximum Likelihood Estimation (MLE)

If we assume a uniform likelihood of $\mathbb{P}(\theta_j)$, we maximize the likelihood of the data given the model:

$$\theta_j = \arg \max_{\theta} \frac{\mathbb{P}(\mathcal{Z}^{\text{train}}|\theta_j)\mathbb{P}(\theta_j)}{\mathbb{P}(\mathcal{Z}^{\text{train}})} \propto \arg \max_{\theta} \mathbb{P}(\mathcal{Z}^{\text{train}}|\theta_j), \quad (110)$$

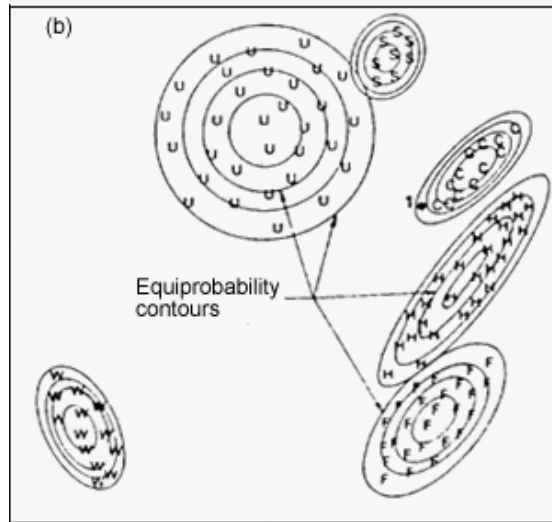
we drop the evidence (denominator) because it is unaffected by θ_j ! We assume that the data can be explained by a distribution (usually a Gaussian), we must simply find the best parameters of the distribution. New points are assigned to the most likely class based on the established distribution:

$$y_i = \arg \max_j \mathbb{P}(\vec{x}_i|\hat{\theta}_j). \quad (111)$$

Finding the optimal $\hat{\theta}_j$ can be done efficiently for standard distributions. For example, for the Gaussian the MLE parameters are established from the data with mean $\hat{\mu}_j = \sum_j \vec{x}_j/J$ and variance $\hat{\Sigma}_j = \sum_j (\vec{x}_j - \mu_j)(\vec{x}_j - \mu_j)^T/J$, where J is the number of samples in class j .

Example: MLE of Gaussians

After a fitting procedure, we then end up with Gaussians in our feature space with Gaussian parameters $\theta_j = \{\mu_j, \Sigma_j\}$ for 6 different classes all with different centers of mass and variance in the feature space:



2.4.2 Maximum a posteriori (MAP)

Rather than only optimizing the likelihood as in MLE, we can compute the $\arg \max_{\theta_j}$ for the full Bayesian posterior:

$$\mathbb{P}(\theta_j | \mathcal{Z}^{\text{train}}) = \frac{\mathbb{P}(\mathcal{Z}^{\text{train}} | \theta_j) \mathbb{P}(\theta_j)}{\mathbb{P}(\mathcal{Z}^{\text{train}})} \propto \mathbb{P}(\mathcal{Z}^{\text{train}} | \theta_j) \mathbb{P}(\theta_j), \quad (112)$$

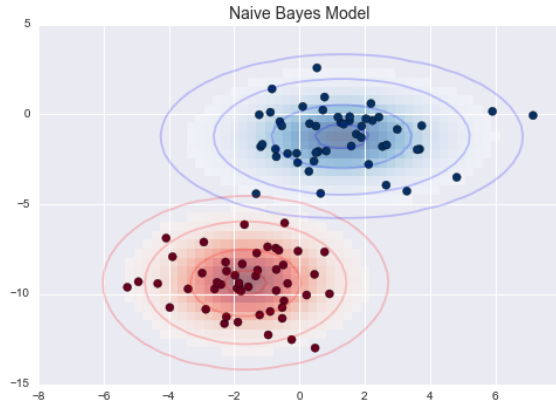
Note that the resulting form is thus the MLE objective weighed by the prior $\mathbb{P}(\theta_j)$. This prior is independent of the data and we can consider it to be a form of **regularization**: when the parameter θ_j deviates much from what we believe is reasonable, we need a lot of evidence to convince us it's right. The MAP thus reduces the MLE if we assume a uniform prior for the parameters θ_j !

2.4.3 Naive Bayes Classifier

The Naive Bayesian classifier assumes that all features are independent. This simplifies the method (no joint probabilities are required anymore) and is furthermore rather robust in practice! Classifying a new point to a cluster is then given by taking the $\arg \max_j (\mathbb{P}(\mathcal{Z}^{\text{train}}, \hat{\theta}_j))$, i.e., which class scores highest, given the model!

Example: Naive Bayes model

You can see it in this plot because the Gaussians are aligned with the features (i.e., grid), rather than allowed to rotate as we saw on the previous page.



MLE, Naive Bayes & MAP

$$\text{MLE: } \arg \max_{\vec{\theta}} \mathbb{P}(\mathcal{Z}^{\text{train}} | \vec{\theta}) \propto \arg \max_{\vec{\theta}} \prod_{i=1}^N \mathbb{P}(y_i | x_i, x_{i+1}, \dots, x_n, \vec{\theta}), \quad (113)$$

$$\text{MAP: } \arg \max_{\vec{\theta}} \mathbb{P}(\vec{\theta} | \mathcal{Z}^{\text{train}}) \propto \arg \max_{\vec{\theta}} \prod_{i=1}^N \mathbb{P}(y_i | x_i, x_{i+1}, \dots, x_n, \vec{\theta}) \mathbb{P}(\vec{\theta}), \quad (114)$$

$$\text{Naive Bayes: } \arg \max_{\vec{\theta}} \mathbb{P}(\mathcal{Z}^{\text{train}} | \vec{\theta}) \propto \arg \max_{\vec{\theta}} \prod_{i=1}^N \mathbb{P}(y_i | x_i, \vec{\theta}) \mathbb{P}(\vec{\theta}). \quad (115)$$

3 Machine learning 103: Regression

We'll do a quick run-down of the three typical regression perspectives/algorithms.

3.1 Linear regression

From the above section with hyperplanes, we may remember that we can put our features in homogenized coordinates $\vec{x} = [1 \text{ Feature 1 Feature 2 } \dots]$, to be able to write any linear regression solution (2-D line, 3-D plane, ...) as:

$$y = \vec{w}^T \vec{x} = \vec{x}^T \vec{w}. \quad (116)$$

Our challenge thus lies in finding the weights \vec{w} . Note that, as with the classification, we can use the Kernel trick to work with a non-linear transformations of our features $\phi(\vec{x})$, such that the non-linear data still allow a linear regression model.

Ordinary least squares

Creating vector $\vec{y} = [y_1 \ y_2 \ \dots \ y_N]^T$ and matrix $\mathbf{X} = [\vec{x}_1 \ \vec{x}_2 \ \dots \ \vec{x}_N]^T$, we can write down the solution for all inputs \vec{X} :

$$\vec{y} = \mathbf{X} \vec{w}. \quad (117)$$

We find the optimal least-squares solution by minimizing the residual sum of squares:

$$\hat{\vec{w}} = \arg \min_{\vec{w}} \|\vec{y} - \mathbf{X} \vec{w}\|_2^2. \quad (118)$$

The optimal solution is well-known to be:

$$\hat{\vec{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y}. \quad (119)$$

We then predict any new point (x_i, y_i) with the least-squares regression model:

$$y_i = \vec{x}_i^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y} = ((\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y})^T \vec{x}_i. \quad (120)$$

Regularization models

If we think we know more about $\hat{\vec{w}}$, i.e., we have a prior, we can regularize the inversion. For example, desiring little energy in the least-squares solution vector $\hat{\vec{w}}$:

$$\text{Ridge regression: } \hat{\vec{w}} = \arg \min_{\vec{w}} \|\vec{y} - \mathbf{X} \vec{w}\|_2^2 + \lambda \|\vec{w}\|_2^2, \quad (121)$$

$$\text{LASSO: } \hat{\vec{w}} = \arg \min_{\vec{w}} \|\vec{y} - \mathbf{X} \vec{w}\|_2^2 + \lambda \|\vec{w}\|_1. \quad (122)$$

The ridge regression solution has an explicit solution:

$$\hat{\vec{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \vec{y}, \quad (123)$$

but no explicit solution exists for LASSO – there we require more computationally intensive optimization techniques to find a solution. The weights from the LASSO optimization usually result in only a few non-zero coefficients, thus are sparse. Both regularization strategies tend to make the models much more stable to compute, and ensure that the solution is not overfitting.

3.2 Bayesian regression

With Bayesian regression we remain close to linear regression, but quantify the uncertainties in the approximation. We assume that our solution may contain errors, for example normally distributed:

$$y = \vec{x}^T \vec{w} + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(0, \sigma^2). \quad (124)$$

The probabilistic model thus assumes that y is drawn from an, e.g., normal distribution:

$$y \sim \mathcal{N}(\vec{x}^T \vec{w}, \sigma^2), \quad (125)$$

that is, the ‘mean’ of every drawn y is given by the linear regression model, but it is polluted with a variance σ^2 . So, how do we do this exactly? It turns out that we typically end up with similar solutions as ridge regression or LASSO; but also obtain the variance in its variables. For example, assuming \vec{w} has a variance of $\lambda \mathbf{I}$, we end up with the following ridge regression models being drawn:

$$\vec{w} \sim \mathcal{N}(\vec{w} | \underbrace{(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \vec{y}}_{\mu_w}, \underbrace{(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1}}_{\Sigma_w}), \quad (126)$$

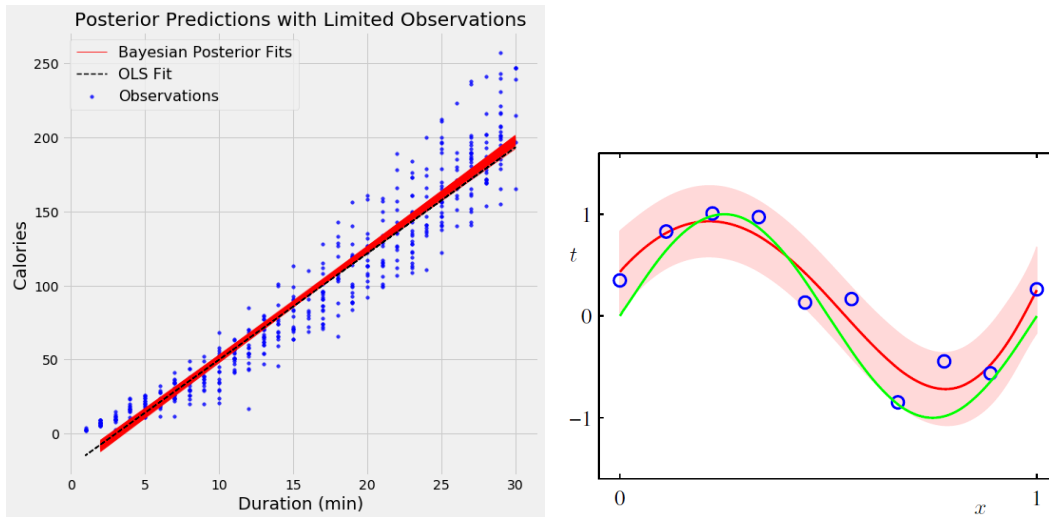
where μ_w is the average ridge regression model, and Σ_w its variance. Thus, based on the data and our prior, we also find out how likely it is to arrive at our model!

Using the Kernel trick we can, furthermore, stray away from strictly linear regression, assuming a transform is applied to the features such that linear regression is possible:

$$y = \vec{w}^T \phi(\vec{x}) + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(0, \sigma^2). \quad (127)$$

Example: Ordinary least squares vs. Bayesian regression

Bayes regression thus provides a *range* of possible linear fits based on the data – not only the single ordinary least squares (OLS) solution! On the left-hand side we see a linear model, on the right-hand side we use the kernelized Bayes’ regression. In both situations, you see a *range* of solutions shaded, usually representing one or two standard deviations from the mean. The line in its center is the least squares solution. We thus find a range of possible model realizations, based on the data!



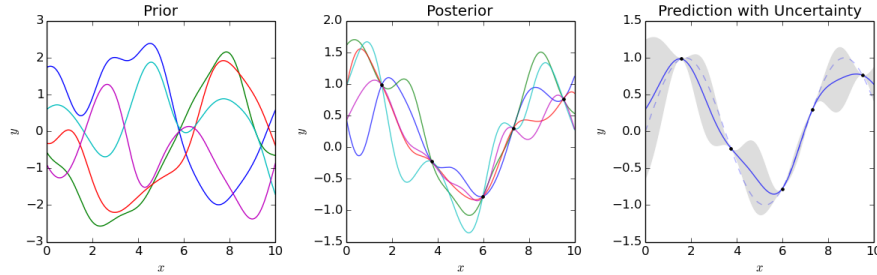
3.3 Gaussian process regression

For the Gaussian process, we take a *very* different perspective at the problem, more akin to nearest neighbours. We assume that $\mathcal{Z}^{\text{train}}$ can be explained by *all* the possible models in the universe that fit our current data-points. The only prior we choose is how far away we are willing to extrapolate information from known data-points to unknown data-points.

We thus make no choice for a linear model \vec{w} or transformation ϕ . We simply assume all possible lines that we can draw are possible models, and we let the data constrain the type of models that actually work for our data!

Example: Gaussian process regression

In the figure below, we see 5 of the infinite number of *priors* (all possible models in the universe, only assuming that data is correlated in x and y directions to a certain point, such that the lines are smooth), then some data comes in and we see 5 draws from the infinite number of *posterior* models (all possible models in the universe from the prior that cross through the known data at $x = \{2, 4, 6, 7, 9\}$), which we can also graph in the way that is done in the right-most picture that shows the standard-deviation of the infinite number of models:



The only required user input is thus the *covariance* or *kernel* that expresses how well we can extrapolate away from given points, thus how smooth the priors are. Check out the demos: <https://youtu.be/R-NUdqxKjos?t=2829> and <https://youtu.be/BzHJ57QCdVo?t=945>. From the right-hand side figure it is clear where we should get new samples (in the most uncertain place, where the grey area is largest) to learn most new information – this is powerful, it tells us where we are more or less uncertain, and which areas we should put effort in when we, e.g., want to find the global minimum!

In other words, we assume that our ‘known’ values y^{train} , as well as our latest new value y_N are drawn from a multivariate Gaussian distribution:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \sim N \left(\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \begin{bmatrix} k_{1,1} + \sigma^2 & k_{1,2} & \dots & k_{1,N} \\ k_{2,1} & k_{2,2} + \sigma^2 & \dots & k_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ k_{N,1} & k_{N,2} & \dots & k_{N,N} + \sigma^2 \end{bmatrix} \right), \quad (128)$$

where $k_{i,j}$ represents the similarity between points x_i, x_j – this encodes how far away points extrapolate

olate. Predictions for new points are then made based only on the previously established data:

$$y_{\text{new}} \sim \mathcal{N}(\mu_*, \sigma_*^2) \quad (129)$$

$$\mu_* = \mathbf{k}^T \mathbf{C}_n^{-1} \bar{y}_{\text{so far}}, \quad \sigma_*^2 = c - \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{k}, \quad (130)$$

$$\mathbf{C}_n = \mathbf{K} + \sigma_{\text{assumed noise}}^2 \mathbf{I}, \quad c = k(x_{\text{new}}, x_{\text{new}}) + \sigma_{\text{assumed noise}}^2, \quad (131)$$

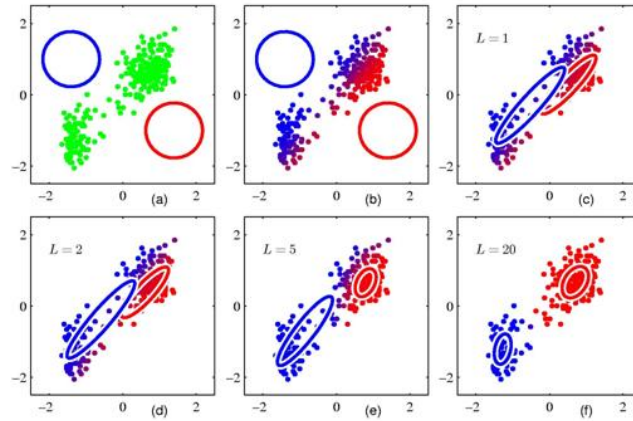
$$\mathbf{k} = k(x_{\text{new}}, \mathbf{X}_{\text{so far}}), \quad \mathbf{K}_{ij} = k(x_{i,\text{so far}}, x_{j,\text{so far}}). \quad (132)$$

4 Machine Learning 104: Density Estimation

I'll keep this short because there is not much to say. But if the data is unstructured, i.e., there are no available y outcomes and we merely wonder about *where* clusters are, *how many* clusters there are, or try to come up with ways to *generate new samples* from an approximate estimate of the true data density P . Principally, we discussed **four** methods, though not in any great detail.

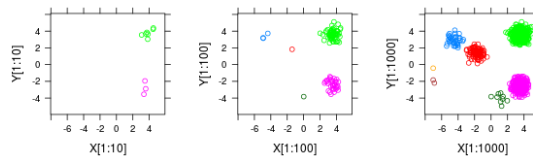
Expectation maximization

In this method we assign each point the most likely cluster given a user-defined number of randomly initialized distributions; and then we modify the distribution to optimally fit the given points. Iterating this process converges to a separation of the data in clusters:



Latent Dirichlet Process

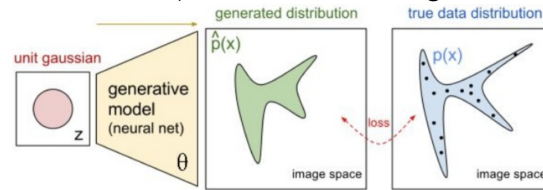
What if the number of clusters is unclear, or you don't want to commit to a single one? The latent Dirichlet Process allows you to assign an arbitrary amount of clusters, based on how much data is available – which is why it relies on the Dirichlet distribution. Points that are close to each other end up in the same cluster, but when new points lie further away, we simply introduce a new cluster. From left to right we add more, introducing new clusters as needed:



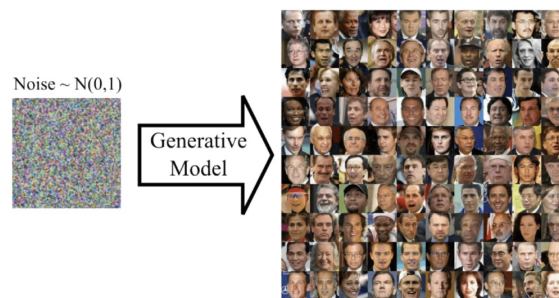
See: <https://www.youtube.com/watch?v=6k7IzONQ0zM> on the Dirichlet distribution, with slides at <https://www.slideshare.net/g33ktalk/machine-learning-meetup-12182013>.

Generative Adversarial Network

This method plays a game with two neural nets: one generates random samples, the other classifies whether the sample could belong to the original data distribution. After many games, the generator net can fool the discriminator, and create convincing new data based on a limited input!

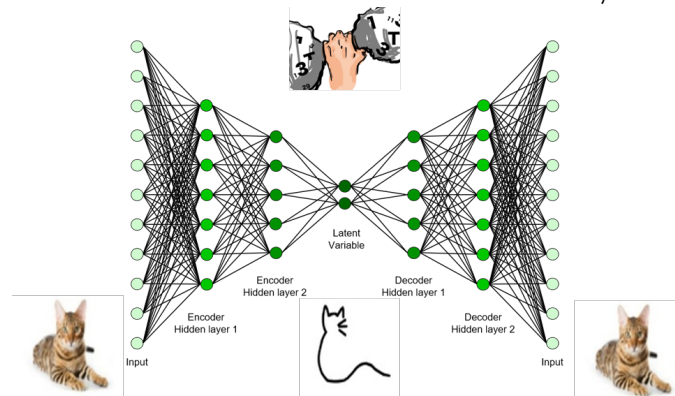


For example, this GAN can generate convincing new faces at random!

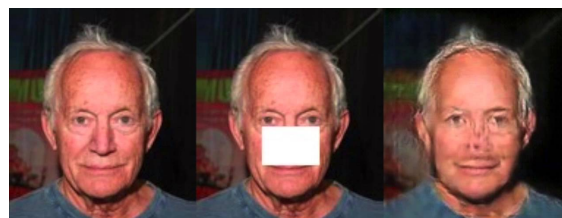


Autoencoders

The autoencoder is a network with identical input and output, but with a small center layer such that the network *must* learn the low-dimensional features of the data, a bit like PCA:



This method makes it for example possible to fill in data from incomplete pictures, though the results are usually slightly unsettling:



5 Machine learning 201: Quantifying Accuracy

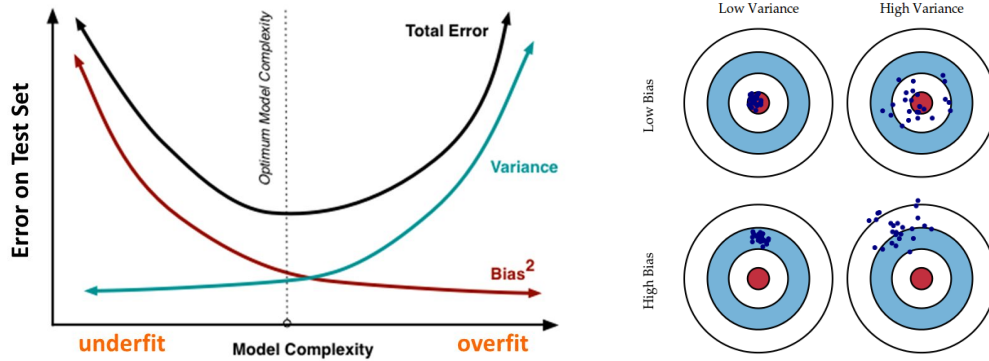
We now study an important aspect of the statistical nature of learning: the bias-variance trade-off.

5.1 Bias-variance trade-off of expected risk

First, what is it? The **bias-variance trade-off** decomposes the **test-error** into a sum of bias, noise and variance. Generally, reducing the bias increases the variance and vice versa, hence, a trade-off. It is a defining insight that separates people just playing around with algorithms from data-scientists! See <http://scott.fortmann-roe.com/docs/BiasVariance.html>.

Example: Typical bias-variance trade-off

Bias skews the results away from the true distribution (thus represents underfitting); variance expresses how much predictions change from draw to draw (thus represents overfitting). An optimum must be found where both components are both small, at a trade-off point.



Setting of the bias-variance trade-off

We assume that the N training data are drawn i.i.d. from:

$$\mathcal{Z}^{\text{train}} \sim P^N, \quad (133)$$

and an additional new test-point (x, y) is drawn from the same distribution:

$$(x, y) \sim P. \quad (134)$$

We will assume a classification problem, as it is easier to derive the trade-off with. But the concept applies also to a regression problem.

We then define three expectations:

$$\bar{y}(x) = \mathbb{E}_{y|x}[Y], \quad \text{Expected label,} \quad (135)$$

$$\bar{h}(x) = \mathbb{E}_{\mathcal{Z}}[h_{\mathcal{Z}}(x)], \quad \text{Expected classifier,} \quad (136)$$

$$E = \mathbb{E}_{\mathcal{Z}, x, y}[(h_{\mathcal{Z}}(x) - y(x))^2], \quad \text{Expected test-error.} \quad (137)$$

That is: $\bar{y}(x)$ is the average expected label for, while $\bar{h}(x)$ represents the average expected classifier based on the data. E , finally, is the expectation of the test-error.

We now decompose the expected test-error for the new data-point (x, y) , after training h with $\mathcal{Z}^{\text{train}}$:

$$E = \mathbb{E}_{\mathcal{Z}, x, y}[(h_{\mathcal{Z}}(x) - \bar{h} + \bar{h} - y)^2] = \mathbb{E}_{\mathcal{Z}, x, y}[\underbrace{((h_{\mathcal{Z}}(x) - \bar{h}) + (\bar{h} - y))^2}_{\text{add 0}}], \quad (138)$$

$$= \mathbb{E}_{\mathcal{Z}, x, y}[\underbrace{(h_{\mathcal{Z}}(x) - \bar{h})^2}_{a^2}] + \mathbb{E}_{\mathcal{Z}, x, y}[\underbrace{(\bar{h} - y)^2}_{b^2}] + \mathbb{E}_{\mathcal{Z}, x, y}[\underbrace{2(h_{\mathcal{Z}}(x) - \bar{h})(\bar{h} - y)}_{+2ab}], \quad (139)$$

will become 0...
Independent expectations w.r.t \mathcal{Z}

$$= \underbrace{\mathbb{E}_{\mathcal{Z}, x, y}[(h_{\mathcal{Z}}(x) - \bar{h})^2] + \mathbb{E}_{\mathcal{Z}, x, y}[(\bar{h} - y)^2]}_{\text{keep the same for now...}} + 2\mathbb{E}_{x, y} \left[\underbrace{\mathbb{E}_{\mathcal{Z}}[(h_{\mathcal{Z}}(x) - \bar{h})] \mathbb{E}_{\mathcal{Z}}[(\bar{h} - y)]}_{\substack{\mathbb{E}_{\mathcal{Z}}[h_{\mathcal{Z}}(x)] = \bar{h}, \\ \mathbb{E}_{\mathcal{Z}}[h_{\mathcal{Z}}(x) - \bar{h}] = 0}} \right], \quad (140)$$

$\rightarrow 0$

$$= \underbrace{\mathbb{E}_{\mathcal{Z}, x}[(h_{\mathcal{Z}}(x) - \bar{h})^2] + \mathbb{E}_{x, y}[(\bar{h} - y)^2]}_{\text{remove independent conditions}}. \quad (141)$$

We now do the exact same thing as above: insert \bar{y} in the equation and get rid of the mixed term:

$$E = \underbrace{\mathbb{E}_{\mathcal{Z}, x}[(h_{\mathcal{Z}} - \bar{h})^2]}_{\text{keep untouched}} + \mathbb{E}_{x, y}[\underbrace{((\bar{h} - \bar{y}) + (\bar{y} - y))^2}_{\text{add 0}}] \quad (142)$$

$$= \mathbb{E}_{\mathcal{Z}, x}[(h_{\mathcal{Z}} - \bar{h})^2] + \underbrace{\mathbb{E}_{x, y}[(\bar{h} - \bar{y})^2] + \mathbb{E}_{x, y}[(\bar{y} - y)^2] + 2\mathbb{E}_{x, y}[(\bar{h} - \bar{y})(\bar{y} - y)]}_{\text{complete the square...}} \quad (143)$$

will become 0...

$$= \mathbb{E}_{\mathcal{Z}, x}[(h_{\mathcal{Z}} - \bar{h})^2] + \mathbb{E}_{x, y}[(\bar{h} - \bar{y})^2] + \mathbb{E}_{x, y}[(\bar{y} - y)^2] + 2 \underbrace{\mathbb{E}_{x, y}[\mathbb{E}_{y|x}[(\bar{h} - \bar{y})] \mathbb{E}_{y|x}[(\bar{y} - y)]]}_{\substack{= \mathbb{E}_{x, y} \\ \mathbb{E}_{y|x}[y(x)] = \bar{y}, \\ \mathbb{E}_{y|x}[\bar{y} - y(x)] = 0}}, \quad (144)$$

$$= \underbrace{\mathbb{E}_{\mathcal{Z}, x}[(h_{\mathcal{Z}}(x) - \bar{h}(x))^2]}_{\text{variance}} + \underbrace{\mathbb{E}_{x, y}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{bias}^2} + \underbrace{\mathbb{E}_{x, y}[(\bar{y}(x) - y(x))^2]}_{\text{noise}^2}. \quad (145)$$

Bias-variance parts

We have now achieved the decomposition, and see that the test-error is composed of three parts:

- **Variance:** Captures how much the classifier changes if you train on a different training set. How “over-specialized” is the classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?
- **Bias:** What is the inherent error that you obtain from the classifier even with infinite training data? This is due to the classifier being “biased” to a particular kind of solution (e.g. a linear classifier, when the actual data is quadratic, has a large bias!). In other words, bias is inherent to the model assumption, and leads to underfitting.
- **Noise:** How big is the data-intrinsic noise? This error measures ambiguity due to the data distribution and feature representation. You can not beat this, it is an aspect of the data. However, finding a better feature representation can improve upon this result!!!

5.2 Rao-Cramer lower bound

We can show that for an unbiased estimator $\mathbb{E}[\theta] = \hat{\theta}$ (i.e., the expected estimator is the true one!), the variance actually has a lower bound. You wouldn't quickly be asked to derive this on an exam, but it's interesting and important to know that there are exact limits to our knowledge.

Rao-Cramer (variance) lower bound

That variance for an unbiased estimator is:

$$\mathbb{V}[\hat{\theta}] \geq \frac{1}{\mathbb{E} \left[\left(\frac{\partial \log \mathbb{P}(x|\theta)}{\partial \theta} \right)^2 \right]} \stackrel{\text{generally}}{=} \frac{1}{\mathbb{E} \left[\frac{\partial^2 \log \mathbb{P}(x|\theta)}{\partial \theta^2} \right]}, \quad (146)$$

where the right-hand sides are the inverse Fisher information I^{-1} . For a 'peaky' distribution with one clear optimal $\hat{\theta}$, the average derivative $\mathbb{E}[\mathbb{P}']$ is large, and, hence, the right-hand side is small. Thus, a focused pdf gives a small lower bound. But it cannot be any smaller than that! See also: <https://www.youtube.com/watch?v=i0JiSddCXMM>.

Example: Rao-Cramer bound for Gaussians

Say we are given an unbiased estimator of the mean μ of a Gaussian. What is the Rao-Cramer lower bound based on N i.i.d. drawn samples from a distribution $\mathcal{N}(x|\mu, \sigma^2)$? Well:

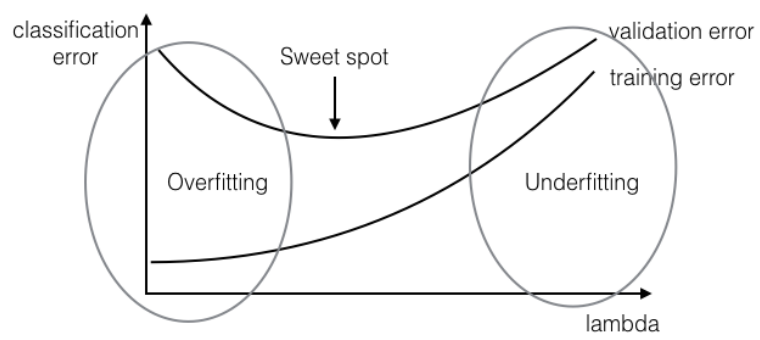
$$\mathbb{V}[\hat{\mu}] \geq - \frac{1}{\mathbb{E} \left[\frac{\partial^2 \log \mathcal{N}(x_1, \dots, x_n | \mu)}{\partial \mu^2} \right]} \stackrel{\text{i.i.d.}}{=} - \sum_{i=1}^N \frac{1}{\mathbb{E} \left[\frac{\partial^2 \log \mathcal{N}(x_i | \mu)}{\partial \mu^2} \right]} = \sum_{i=1}^N \frac{1}{\mathbb{E} \left[\frac{\partial^2 \frac{(x_i - \mu)^2}{2\sigma^2}}{\partial \mu^2} \right]} = \frac{\sigma^2}{N}. \quad (147)$$

Thus, the variance of finding the true μ is at least N/σ^2 . Thus, the smaller the spread in the original samples, or the greater number of draws, the smaller our error in estimating of μ will be. This makes a lot of intuitive sense! If you draw one sample, you expect to be off μ by exactly σ^2 , which is exactly what $\mathcal{N}(x|\mu, \sigma^2)$ means to begin with!

5.3 Regularization & Model selection

Data-based complexity measures: cross-validation, bootstrap, jackknife: data-driven. Model-based complexity measures: Neyman Pearson test, AIC, BIC.

To-do.

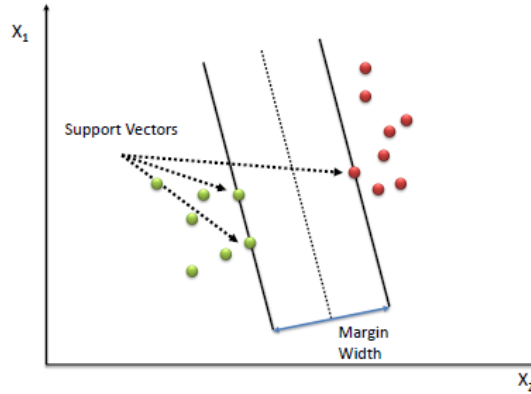


6 Machine learning 202: Perceptrons and SVM's for classification

6.1 Perceptron learning

6.2 Hard SVM learning

The goal in designing the Support Vector Machine is to find the **maximum separating hyper-plane** for our (perfectly separated) binary clusters. To remind you, the figure again (actually, the perpendicular separation between the two clusters is measured from the middle plane, the separation between the two is $2 \cdot \text{margin}$, unlike what the figure may suggest):



So we now state that we want to maximize a margin while (for homogeneous coordinates),

$$y\vec{w}^T \vec{x} \geq m, \quad (148)$$

for the cases:

$$y = \begin{cases} -1 & \vec{w}^T \vec{x} \leq m, \\ +1 & \vec{w}^T \vec{x} > m. \end{cases} \quad (149)$$

We can measure the margin the following way. Assuming it exists, then there is at least one support vector \vec{x}^+ and one support vector \vec{x}^- corresponding to the two clusters. Then we know that we can take their difference $(\vec{x}^+ - \vec{x}^-)$ and project it in the unit-vector direction of \vec{w} . From the figure we specified that the total separation is $2 \cdot \text{margin}$, thus:

$$2 \cdot \text{margin} = \frac{\vec{w}^T}{|\vec{w}|} (\vec{x}^+ - \vec{x}^-) = \frac{2m}{|\vec{w}|}. \quad (150)$$

Note that the margin is thus given by $m/|\vec{w}|$, we can vary the two parameters inversely proportional and find an infinite number of solutions with the same margin... So, let's choose $m = 1$, such that we get the following optimization problem:

Hard-margin SVM optimization (primal)

$$\text{Maximize } 1/|\vec{w}|, \text{ i.e., minimize } |\vec{w}|: \quad \frac{1}{2} \vec{w}^T \vec{w}, \quad (151)$$

$$\text{Subject to: } y\vec{w}^T \vec{x} \geq 1. \quad (152)$$

Solving SVM with Lagrangian

We can convert the SVM problem to a dual problem of the Lagrangian, simply putting the minimization object and constraints in one function, penalizing missing the constraints with factors α :

$$L(\vec{w}, \alpha) = \frac{1}{2} \vec{w}^T \vec{w} - \sum_{n=1}^N \alpha_n [y_n \vec{w}^T \vec{x}_n - 1]. \quad (153)$$

We then set its partial derivatives for \vec{w} to 0:

$$\frac{\partial L(\vec{w}, \alpha)}{\partial \vec{w}} = \vec{w} - \sum_{n=1}^N \alpha_n y_n \vec{x}_n \equiv 0 \iff \vec{w} = \sum_{n=1}^N \alpha_n y_n \vec{x}_n. \quad (154)$$

Substituting into eq. (153) provides:

$$L(\vec{w}, \alpha) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j - \sum_{n=1}^N \sum_{i=1}^N \alpha_n \alpha_i y_n y_i \vec{x}_n^T \vec{x}_i + \sum_{n=1}^N \alpha_n, \quad (155)$$

which we can simplify to:

$$L(\vec{w}, \alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j. \quad (156)$$

This gives rise to the dual problem for SVM:

SVM optimization (dual problem)

$$\text{Minimize: } L(\vec{w}, \alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j, \quad (157)$$

$$\text{Subject to: } \forall_i, \alpha_i \geq 0, \quad (158)$$

$$\sum_{n=1}^N \alpha_n y_n = 0, \quad (159)$$

Where the final constraint comes out due to the homogenized constraint $(\vec{x})_1 = 1$ for which we, apparently, don't want to minimize the L^2 norm. Thus the actual Lagrangian formulation had to be:

$$L(\vec{w}, \alpha) = \frac{1}{2} \vec{w}^T \vec{w} - \sum_{n=1}^N \alpha_n [y_n (\vec{w}^T \vec{x} + b) - 1]. \quad (160)$$

We then, furthermore, know that the α_i values are only required at the support vector locations which will give us the answer to the primal problem (**Kuhn-Tucker Conditions**), i.e., the coefficients α_i are an incredibly sparse set; and we compute \vec{w} from eq. (154).

The advantageous feature of this formulation is that we have the inner product $\vec{x}_i^T \vec{x}_j$, such that a data-transformation $\phi(\vec{x}_i)^T \phi(\vec{x}_j)$ can be replaced by a more efficient transformation $k(\vec{x}_i, \vec{x}_j)$ that computes this inner product for us, thus a non-linear **kernelized SVM** is easily found:

$$L(\vec{w}, \alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k(\vec{x}_i, \vec{x}_j). \quad (161)$$

6.3 Soft-margin SVM

Variations on the theme are then easily made. We can allow some ‘slack’ in the classification, such that:

$$y_i(\vec{w}^T \vec{x} + w_0) \geq m(1 - \xi_i), \quad \xi_i \geq 0, \quad (162)$$

where of course we want $\xi_i = 0$ for most points. In other words, we want to penalize those occurrences of ξ_i as much as possible. A useful parameterization of the problem is found to be:

Soft-margin SVM optimization (primal)	
Minimize $ \vec{w} $:	$\frac{1}{2} \vec{w}^T \vec{w} + C \sum_{n=1}^N \xi_n,$ (163)
Subject to:	$y(\vec{w}^T \vec{x} + w_0) \geq 1 - \xi_i,$ (164)
	$\forall_i, \xi_i \geq 0.$ (165)

The solution happens to have the same Lagrangian L , just different secondary constraints!

7 Lecture 3: Density Estimation in Regression; Parametric Models

We start with parametric density estimation: we assume a parametric form of the density (e.g., Gaussian, ...) and estimate the location and width of parameters such as mean and variance. Then we do the maximum likelihood estimation, i.e., for which parameter do we optimally fit the data:

Maximum likelihood model parameter

$$\hat{\theta}_{ML} \in \arg \max_{\theta} \mathbb{P}(\mathcal{X}^{\text{train}} | \theta). \quad (166)$$

The method of finding this $\hat{\theta}$ is to consider it to consider $\mathbb{P}(X|\theta)$ a continuous function of θ , for which the maxima has a 0 derivative, which we must simply find:

Maximum likelihood procedure

$$\hat{\theta}_{ML} \Rightarrow \nabla_{\theta} \log \left(\mathbb{P}(\mathcal{X}^{\text{train}} | \hat{\theta}_{ML}) \right) = 0. \quad (167)$$

Example: Mean of Gaussian

It is helpful to go through derivations like this one, at least once. The Gaussian for univariate data is $\mathcal{N}(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. The maximum likelihood for μ based on N samples is the found through:

$$0 \equiv \frac{\partial}{\partial \mu} \left(\sum_{i=1}^N \left(-\frac{1}{2} \log(2\pi\sigma^2) - \frac{(x_i - \mu)^2}{2\sigma^2} \right) \right), \quad (168)$$

$$\equiv \frac{\partial}{\partial \mu} \left(\sum_{i=1}^N \left(-\frac{(x_i - \mu)^2}{2\sigma^2} \right) \right), \quad (169)$$

$$\equiv \sum_{i=1}^N \frac{(x_i - \mu)}{\sigma^2} = \left(\sum_{i=1}^N x_i \right) - N\mu \iff \hat{\mu} = \frac{1}{N} \sum_{i=1}^N x_i. \quad (170)$$

This aligns with our understanding of the expected value based on samples, $\mathbb{E}(x_{i=1,\dots,N})$.

7.1 Model selection 1: K-fold cross-validation

So we establish that there really are two tasks to carry out in machine learning: model fitting and model selection. Leaving the model fitting for later sections, we first focus on model selection methods. The **K-Fold cross-validation** method uses the hold-out strategy from the previous section, but does so K times, *splitting the data into K non-overlapping portions*. With N samples in the full training data, we then compute K times an intermediate validation score for a chosen model \hat{f} :

$$\hat{R}_k(\hat{f}, \mathcal{X}^{\text{train}} \setminus \mathcal{X}^{\text{validate}_k}, \mathcal{Y}^{\text{train}} \setminus \mathcal{Y}^{\text{validate}_k}) = \frac{1}{(1 - \frac{1}{K})N} \sum_{n=1}^{(1 - \frac{1}{K})N} \mathbb{I}_{\hat{f}(x_n) \neq y_n}. \quad (171)$$

We then sum up all K validation scores to obtain the final error measure:

$$\hat{R}(\hat{f}, \mathcal{X}^{\text{train}}, \mathcal{Y}^{\text{train}}) = \frac{1}{K} \sum_{k=1}^K \hat{R}_k(\hat{f}, \mathcal{X}^{\text{train}} \setminus \mathcal{X}^{\text{validate}_k}, \mathcal{Y}^{\text{train}} \setminus \mathcal{Y}^{\text{validate}_k}) \quad (172)$$

For $K = N$ this equals to ‘leave-one out cross-validation’ (LOOCV), but this can take a long time. However, it is unbiased. In practice, $K \in [5, 10]$ is a typical choice.

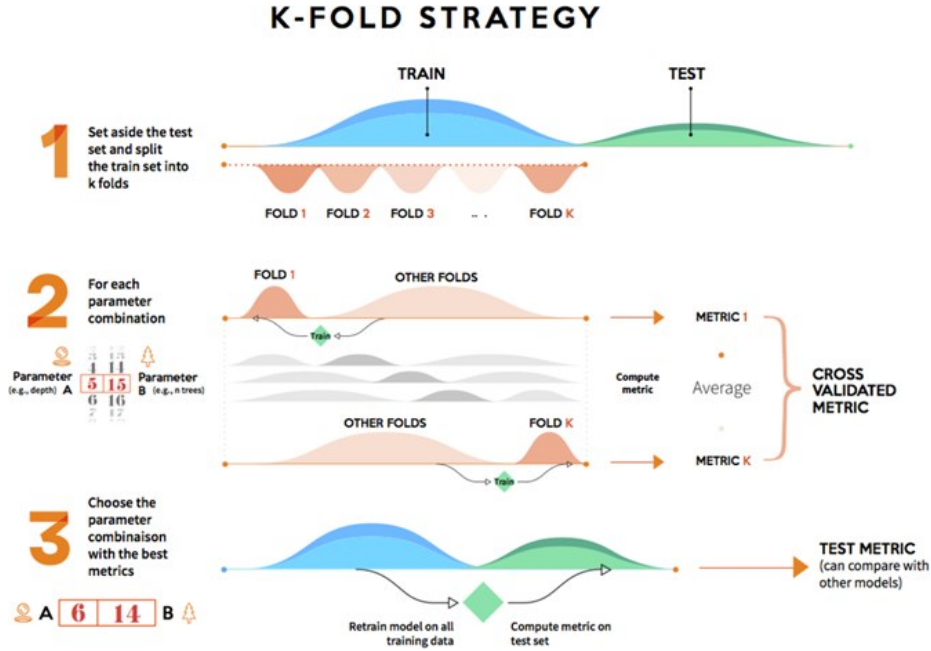


Figure 1: A graphical example of K -fold cross-validation. The training set is split into K folds; then for each model-parameter combination the model is trained in the $K - 1$ sets and scored on the K remaining fold. The average score is used as the cross-validated metric. The optimal model-parameter combination can then be used to train the final model.

7.2 Bootstrap

.632 bootstrap

7.3 Jack-knife

8 Regression

8.1 Regression

8.1.1 The setting

We, abstractly, consider an object space \mathcal{O} with data-space for d -dimensional data leading to a single output, sorted in tuples: $\mathcal{Z} = \{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R} : i \leq i \leq n\}$. We use a model with output Y and input $X = (X_0, X_1, \dots, X_d)$ features with $X_0 = 1^1$ and ϵ the noise with expected value $\mathbb{E}[\epsilon] = 0$:

$$\mathbf{Y} = f(\mathbf{X}, \theta) + \epsilon. \quad (173)$$

In the **Bayesian view**, \mathbf{X} , \mathbf{Y} and θ are random variables, which brings with it useful properties I think. In the view of **parametric statistics**, the form of $\mathbb{P}(\mathbf{X}, \mathbf{Y}|\theta)$ is given, we want to estimate θ for the likelihood $\mathbb{P}(\mathbf{Y}|\mathbf{X})$. In **non-parametric statistics** we sample \mathbf{X} and \mathbf{Y} to estimate the likelihood $\mathbb{P}(X, Y)$. In **statistical learning theory** we directly minimize the empirical risk function $(\arg \min_{f \in \mathcal{C}} \hat{R}(f, \mathcal{Z}^{\text{train}}))$ without estimating the likelihood.

8.1.2 The frequentist vs Bayesian view

Bayes' rule (preferred by the teacher over Bayes' theorem):

$$\mathbb{P}(\text{model}|\text{data}) = \frac{\mathbb{P}(\text{data}|\text{model})\mathbb{P}(\text{model})}{\mathbb{P}(\text{data})} = \text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}. \quad (174)$$

This is in contrast with Frequentism. However, these guys did come up with the **maximum likelihood** method, Fisher information, sampling theory and hypothesis testing.

8.1.3 Ex: Regression with maximum likelihood

This is extra, from Bishop's 'Pattern Recognition & Machine Learning'. We will work systematically in the below.

1. **The model.** We assume that we can write the model as a polynomial function:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j. \quad (175)$$

2. **One data distribution.** We assume that we can write the one data-point t given an x , using a normal distribution with inverse variance β^{-1} (giving the precision) and a mean given by the equation for y above:

$$\mathbb{P}(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1}). \quad (176)$$

3. **The likelihood function.** We can now write down the likelihood of the data given the model. Assuming that the data was drawn iid we simply get the product of the individual probability distributions:

$$\mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}). \quad (177)$$

¹For linear regression we assume $y = \mathbf{a}^T \mathbf{x} + b$, which we could also write as $y = \mathbf{a}^T \mathbf{x}$ when $x_0 = 1$ and $a_0 = b$.

4. **The log likelihood function.** Taking the logarithm of the function above:

$$\log \mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \log \prod_{n=1}^N \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}), \quad (178)$$

$$= \sum_{n=1}^N \log (\mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1})), \quad (179)$$

$$= \sum_{n=1}^N \log \left(\frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{-\frac{\beta}{2}(t_n - y(x_n, \mathbf{w}))^2} \right), \quad (180)$$

$$= \sum_{n=1}^N \left[\log(\sqrt{\beta}) - \log(\sqrt{2\pi}) + \log \left(e^{-\frac{\beta}{2}(t_n - y(x_n, \mathbf{w}))^2} \right) \right], \quad (181)$$

$$= \sum_{n=1}^N \left[\frac{1}{2} \log(\beta) - \frac{1}{2} \log(2\pi) + \left(-\frac{\beta}{2}(t_n - y(x_n, \mathbf{w}))^2 \right) \right], \quad (182)$$

$$= \frac{N}{2} \left(\log(\beta) - \log(2\pi) \right) + \sum_{n=1}^N \left(-\frac{\beta}{2}(t_n - y(x_n, \mathbf{w}))^2 \right) \quad (183)$$

5. **The maximum log-likelihood.** Taking the derivative w.r.t. \mathbf{w} , and setting this to zero, should give the optimal model:

$$\frac{\partial \log \mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)}{\partial \mathbf{w}} = -\frac{\beta}{2} \sum_{n=1}^N (t_n - y(x_n, \mathbf{w}))^2, \quad (184)$$

$$= -\frac{\beta}{2} \sum_{n=1}^N \sum_{j=1}^M (t_n - w_j x_n^j)^2, \quad (185)$$

$$= 0 \iff \text{least-squares solution } \mathbf{w}_{\text{ML}}. \quad (186)$$

Similarly, we can derive the optimal precision (inverse standard-deviation) of the likelihood function, β_{ML} :

$$\frac{\partial \log \mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)}{\partial \beta} = \frac{N}{2\beta} - \sum_{n=1}^N \left(\frac{1}{2}(t_n - y(x_n, \mathbf{w}_{\text{ML}}))^2 \right), \quad (187)$$

$$= 0 \iff \frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N (t_n - y(x_n, \mathbf{w}_{\text{ML}}))^2. \quad (188)$$

6. **Predict new data.** We can no make predictions for new values of x , in terms of the predictive distribution rather than a point-estimate that we usually get:

$$\mathbb{P}(t|x, \mathbf{w}_{\text{ML}}, \beta_{\text{ML}}) = \mathcal{N}(t|y(x, \mathbf{w}_{\text{ML}}), \beta_{\text{ML}}^{-1}). \quad (189)$$

7. **Introducing a prior for \mathbf{w} .** If we assume that the coefficients \mathbf{w} are spread with a Gaussian distribution:

$$\mathbb{P}(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}) \quad (190)$$

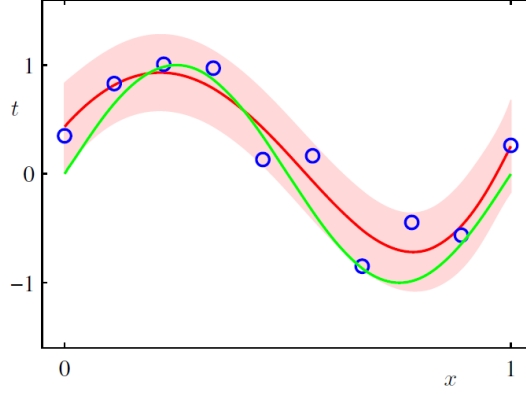


Figure 2: A Bayesian version of regression with mean (red) and ± 1 standard deviation (shaded), given a limited number of data-points, and some expectations about the noise levels.

We can now make a comment about the *posterior* distribution! Indeed, we know that:

$$\mathbb{P}(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto \mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)\mathbb{P}(\mathbf{w}|\alpha). \quad (191)$$

Taking the logarithm of the RHS of the equation we find:

$$\log(\mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)\mathbb{P}(\mathbf{w}|\alpha)) = \quad (192)$$

$$\frac{N}{2} \left(\log(\beta) - \log(2\pi) \right) + \sum_{n=1}^N \left(-\frac{\beta}{2} (t_n - y(x_n, \mathbf{w}))^2 \right) + \frac{M+1}{2} \log \left(\frac{\alpha}{2\pi} \right) - \frac{\alpha}{2} \mathbf{w}^T \mathbf{w}. \quad (193)$$

The extremum is then found through the derivative to \mathbf{w} :

$$\frac{\partial -\log(\mathbb{P}(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)\mathbb{P}(\mathbf{w}|\alpha))}{\partial \mathbf{w}} = \sum_{n=1}^N \left(\frac{\beta}{2} (t_n - y(x_n, \mathbf{w}))^2 \right) + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w}, \quad (194)$$

$$= 0 \iff \text{the ridge-regression solution with } \lambda = 2 \frac{\alpha}{\beta}. \quad (195)$$

8. **Predicting values given the posterior distribution.** We can ‘simply’ marginalize the known quantities now to introduce new t^* for x^* :

$$\mathbb{P}(t^*|x^*, \mathbf{x}, \mathbf{t}) = \int \mathbb{P}(t^*|x^*, \mathbf{w})\mathbb{P}(\mathbf{w}|\mathbf{x}, \mathbf{t}) d\mathbf{w} \quad (196)$$

The solution to this problem is a normal distribution itself with a mean and variance. The solution to this can be found with matrix calculus. Basically, the mean is our usual solution to the problem, and the variance is the information gained: an uncertainty boundary (or ± 1 standard deviation boundary around the mean).