

Machine Learning

Kinds of learning
Mathematical detour. Vectors
Nearest Neighbours algorithms
Conclusion

Chapter 2: Introduction to machine learning and Nearest Neighbours

Volodymyr Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS5920 Machine Learning
Last edited: September 17, 2022

Volodymyr Vovk

Chapter 2

1

Plan

- 1 Kinds of learning
- 2 Mathematical detour. Vectors
- 3 Nearest Neighbours algorithms
- 4 Conclusion

Supervised learning (1)

Typical machine-learning problems (such as the problem of recognizing irises) have a similar structure:

- They deal with **samples** (or **objects** or **cases** or **instances** or **unlabelled examples** or **inputs**) x .
- The task is to provide a **label** (or **target** or **outcome** or **response** or **output variable**) y for a sample x .
- We learn from a set of **labelled samples** (or **observations** or **labelled examples**), which are pairs (x, y) consisting of a sample x and its label y .
- **Remark:** in statistics, “sample” usually means a set of observations, so be careful when talking to statisticians.

Main type of machine learning

Supervised learning (2)

The handwritten digits recognition problem:

- A sample is a scanned image of a symbol.
- A label belongs to the set $\{0, 1, 2, \dots, 9\}$.

The problem of **supervised learning** consists of providing labels for new (test) samples.

Supervised learning (3)

- If the set of possible labels in supervised learning is finite, the problem is called **classification** (and then labels are sometimes referred to as **classes**).
 - **Binary classification**: two possible labels; for example: differentiating 0s from the other digits.
 - **Multi-class classification**: more than two (but finitely many) possible labels; for example: recognizing digits from the set $\{0, 1, 2, \dots, 9\}$.
- If the set of possible labels in supervised learning is infinite (usually the set \mathbb{R} of real numbers or its uncountable subset), the problem is called **regression**.
 - Example: determining the price of a house from its description.
- (Binary) classification and regression are our main problems in this module.

Classification is just classifying the label for a sample

Binary:

Multi-class: for a finite number of labels, more than 2.

Regression: for infinite number of labels.

Classification vs Regression: classification is when there are a finite number of labels, regression is regressing to the point from an infinite spectrum until we reach our guess. or just when there is an infinite label set.

Supervised learning (4)

- We are usually interested in the case where each sample is a vector (say, a NumPy array).
- Components of samples are **features** (also known as **attributes**).
- Features can be **discrete** (with countably, usually finitely, many possible values) or **continuous** (taking values in the real line \mathbb{R} or its uncountable subset, such as $[0, \infty)$).

Two kinds of supervised learning

- Supervised learning can proceed according to two protocols: **batch** or **online**.
- In **batch** learning we are given a **training set** of labelled samples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and we need to work out labels for the samples from a test set $x_1^*, x_2^*, \dots, x_m^*$.
- Most of this module is about batch learning.
- **Pedantic remark:** a training set is not a set since it can contain several copies of the same labelled sample (such objects are called **bags**, or **multisets**). In this module it will usually be regarded as a sequence.

batch is also most popular, we focus on it in this course

test set is what we try to figure out, x^*

Online learning

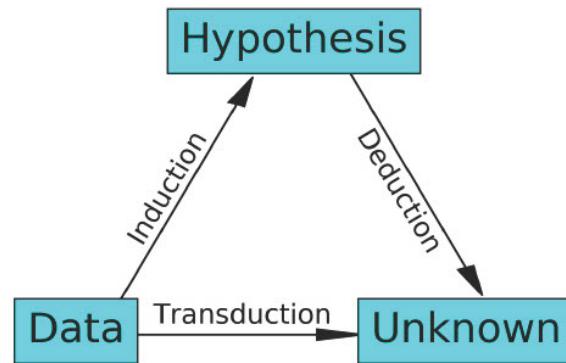
- In **online** (supervised) learning we are given samples and their labels as follows:
 - we see x_1
 - we work out the predicted label for x_1
 - we see the true label y_1 for x_1
 - we see x_2
 - we work out the predicted label for x_2
 - we see the true label y_2 for x_2
 - etc.
- Examples: predicting the weather or stock prices.

inefficient because you cant stop the training and start over

Exploration and exploitation

- In batch learning, there are two separate stages:
 - the **training** (or **exploration**) stage, when we analyze the training set (and possibly find a hypothesis describing it)
 - the **exploitation** stage, when we apply the hypothesis to the test data.
- In online learning, exploration and exploitation are intertwined.

Induction vs transduction (1)



Induction vs transduction (2)

- Sometimes we do not create a hypothesis.
- **Induction**: based on our experience (dataset), we arrive at a general hypothesis (or **model**) which tells us something about the unseen data.
- **Transduction**: we avoid a general hypothesis and deal with each instance (or batch) of new data individually.
- The difference can be subtle (e.g., computational); we will see examples later.
- The words “induction” and “transduction” are used in many other senses (e.g., mathematical induction and transduction in biology), but in this module we ignore those.

Induction vs transduction (3)

“Induction” is ancient and “transduction” is new:

- For a move from particular to universal, Aristotle in the 300s BCE used the Greek word *epagoge* ($\epsilon\pi\alpha\gamma\omega\gamma\eta$), which Cicero translated into the Latin word *inductio*.
- There was a deep study of induction by David Hume in 1740. His conclusion: induction has no rational, let alone logical, basis, but is a custom of the mind.
- New light on induction was shed by Karl Popper starting from 1934 (although he did not like the term “induction”): we formulate hypotheses freely, and some of them will survive the clash with reality.
- The term “transduction” is due to Vladimir Vapnik, one of the founders of machine learning.

Induction vs transduction (4)

- Deep philosophical discussions of induction are somewhat irrelevant in machine learning.
- Whereas real world is unrestricted, mainstream machine learning makes the **IID assumption**:
the labelled samples (x_i, y_i) are assumed to be generated independently from the same probability measure.
- IID = independent identically distributed

assume data is IID

Unsupervised learning

- **Unsupervised learning** is concerned with analyzing data without labels, e.g.:
 - finding out the structure of the data (does it live in a low-dimensional manifold? can we simplify it by a simple transformation?)
 - clustering, i.e., finding **clusters** (groups of similar examples) in data.
- Some uses of unsupervised learning:
 - company clustering its customers as a first stage of a marketing campaign
 - outlier detection (such as recording mistakes).

eg finding the structure of data

Plan

- 1 Kinds of learning
- 2 Mathematical detour. Vectors
- 3 Nearest Neighbours algorithms
- 4 Conclusion

Vectors

- A p -dimensional vector is an array of p numbers

$$v = (v_1, v_2, \dots, v_p).$$

- \mathbb{R}^p is the set of all p -dimensional vectors.
 $\mathbb{R}^p = \mathbb{R} \times \mathbb{R} \times \dots \times \mathbb{R}$ (p times)
- Sometimes we consider **row** vectors and sometimes **column** vectors.

v will be a vector of the sample, where all the elements v_1 to v_p are the features of that sample x .

Basic operations

- Vectors are added component-wise (element-by-element):

- if $u = (u_1, u_2, \dots, u_p)$ and $v = (v_1, v_2, \dots, v_p)$, then

$$u + v = (u_1 + v_1, u_2 + v_2, \dots, u_p + v_p).$$

- A vector can be multiplied by a scalar:

- if $v = (v_1, v_2, \dots, v_p)$ and c is a number, then

$$cv = (cv_1, cv_2, \dots, cv_p).$$

Dot product

- The dot product of vectors $u = (u_1, u_2, \dots, u_p)$ and $v = (v_1, v_2, \dots, v_p)$ is

$$u \cdot v = u_1 v_1 + u_2 v_2 + \cdots + u_p v_p = \sum_{j=1}^p u_j v_j.$$

- Other notation: $\langle u, v \rangle$.
- u and v are **orthogonal** if $u \cdot v = 0$.

Norm

- The **norm** of a vector v is

$$\|v\| = \sqrt{v \cdot v}.$$

- If $v = 0 = (0, 0, \dots, 0)$ then $\|v\| = 0$.
- And if $\|v\| = 0$ then $v = 0 = (0, 0, \dots, 0)$.

Vectors and points; distance

- A vector $v = (v_1, v_2, \dots, v_p)$ can be thought of as
 - a point
 - an arrow ("vector") from 0 to (v_1, v_2, \dots, v_p)
- The **distance** between two points u and v is expressed via the corresponding vectors by

$$d(u, v) = \|u - v\|.$$

- It is a common (and acceptable in most cases) abuse of terminology to identify vectors and points.

A Python-style notation for vectors (1)

- In mathematics the components of a p -dimensional vector v are usually denoted v_1, \dots, v_p (as we do in this section):

$$v = (v_1, \dots, v_p).$$

- In computer science, the Python-style notation is sometimes used:

$$v = (v[0], \dots, v[p - 1]).$$

A Python-style notation for vectors (2)

- In this notation the formula for dot product becomes

$$\begin{aligned} u \cdot v &= u[0]v[0] + u[1]v[1] + \cdots + u[p-1]v[p-1] \\ &= \sum_{j=0}^{p-1} u[j]v[j]. \end{aligned}$$

- Conceptually this shift in indexing from $1, \dots, p$ to $0, \dots, p-1$ is trivial, but in practice it can be confusing.

Plan

- 1 Kinds of learning
- 2 Mathematical detour. Vectors
- 3 **Nearest Neighbours algorithms**
- 4 Conclusion

Nearest Neighbour algorithm

- **Nearest Neighbour (NN)** is a simple algorithm for classification or regression.
- Suppose we are given a training set $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
- We need to predict the label for a test sample x^* .
- The algorithm:
 - Search for the training sample that is nearest the test sample x^* .
 - Predict that the label of the new sample is the same as of this nearest training sample.
- It works for both classification and regression!

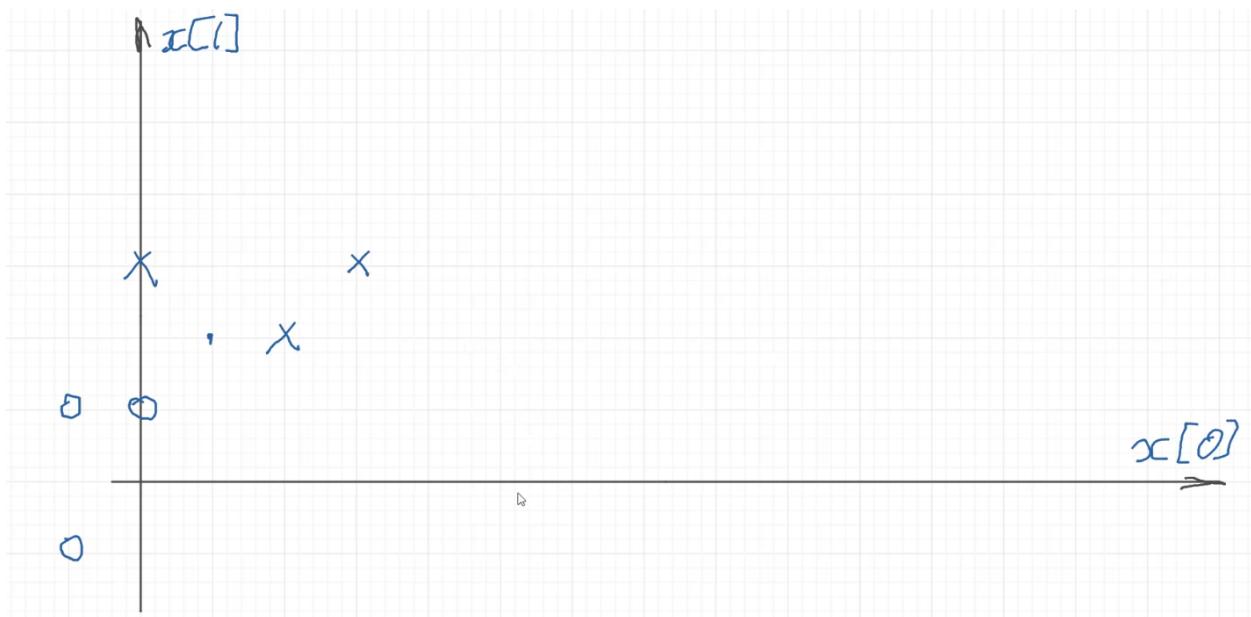
Classification

- In binary classification problems, the label space is often taken to be $\{-1, +1\}$ (another convention is $\{0, 1\}$).
- Sometimes we call the samples with label $+1$ **positive** and the samples with label -1 **negative**.

Example (1)

- Training set:
 - positive samples: $(0, 3), (2, 2), (3, 3)$
 - negative samples: $(-1, 1), (-1, -1), (0, 1)$.
- Test sample: $(1, 2)$.
- Let us calculate the distance from the new sample to each training sample.

Let us draw these points on a graph.



the small point is our test sample, we find its nearest neighbor with euclidian distance

Kinds of learning
 Mathematical detour. Vectors
Nearest Neighbours algorithms
 Conclusion

1-NN algorithm
 KNN algorithm

Example (2)

Training sample	Label	Euclidean distance
(0, 3)	+1	1.414
(2, 2)	+1	1
(3, 3)	+1	2.236
(-1, 1)	-1	2.236
(-1, -1)	-1	3.506
(0, 1)	-1	1.414

- (2, 2) is the nearest sample and it is positive.
- We predict that our new sample is positive too.

Transduction

- This is our first example of transduction.
- We do not formulate any hypothesis; we simply output a prediction on the test sample.
- There is no model.

IMPORTANT

This is transduction

K Nearest Neighbours

- The ***K* Nearest Neighbours (KNN)** algorithm is an enhancement of simple Nearest Neighbour.
- The algorithm for classification:
 - find the K nearest neighbours to the new sample
 - take a vote between them to decide on the best label for the new sample.
- The algorithm for regression:
 - find the K nearest neighbours to the new sample
 - predict with the average of their labels.

classification vs regression.

Classification : We vote

Regression: We take average

Example

- Take the same training set as before (slides 26–27).
- Let us find the 3 Nearest Neighbours classification for the test sample $(1, 2)$.
- The nearest 3 training samples and their labels are:
 - $((2, 2), +1)$
 - $((0, 3), +1)$
 - $((0, 1), -1)$.
- Two of these are positive and one is negative.
- The vote gives us a positive classification for the test sample $(1, 2)$.

Training sample	Label	Euclidean distance
$(0, 3)$	+1	1.414
$(2, 2)$	+1	1
$(3, 3)$	+1	2.236
$(-1, 1)$	-1	2.236
$(-1, -1)$	-1	3.506
$(0, 1)$	-1	1.414

So 3 nearest neighbors are $0, 3 \quad 2, 2 \quad$ and $0, 1$

As this is classification, we vote for +1

Computational aspects

- Computing one distance takes time $O(p)$, where p is the dimension of the samples (number of numeric features).
- For each sample from the test set we need to compute n distances, where n is the size of the training set.
- The total time required per each test sample: $O(np)$.
 - K is usually a small constant.

Plan

- 1 Kinds of learning
- 2 Mathematical detour. Vectors
- 3 Nearest Neighbours algorithms
- 4 Conclusion

General distances

- So far we have discussed Nearest Neighbours based on Euclidean distance.
- But we can use any distance instead.
- Recognition of hand-written images: one of the best methods is 1NN based on the “tangent distance” (not in this module).
- Later in the module: kernel-based distances in a feature space.
 - Kernels are a rich source of useful metrics: to each kernel corresponds a metric.

Some other uses of Nearest Neighbours

- Recognizing satellite images.
- Recognizing patterns in electrocardiography.
- Text classification (e.g., answering questions such as “is this Web article about politics?”)
 - $K = 3$ and $K = 5$ are common choices
 - but sometimes much larger values (between 50 and 100) are also used.
- More examples: the lab and Assignment 1.

Literature



M17: Chapter 1

Nearest Neighbours in general



H09: Sections 13.3–13.5

Nearest Neighbours in general

Section 13.3.3: tangent distance (not in this module)



J21: Section 2.2.3 (Nearest Neighbours classification);

Section 3.5 (Nearest Neighbours regression)

Chapter 3: Conformal prediction

Vladimir Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS4920/CS5920 Machine Learning

Last edited: September 29, 2020

Plan

- 1 Assumptions of machine learning
- 2 Conformal prediction
- 3 Conformal prediction based of Nearest Neighbour
- 4 Validity and efficiency of conformal predictors

skipping the history, ML assume that IID is data and in stats we assume parameters like Gaussian.

Plan

- ① Assumptions of machine learning
- ② Conformal prediction
- ③ Conformal prediction based of Nearest Neighbour
- ④ Validity and efficiency of conformal predictors

Prediction with confidence

- We want **guaranteed validity** (such as guaranteed probability of error).
- Commonplace in statistics (confidence intervals, prediction intervals).
- Statisticians could do it because of their strong assumptions (such as Gaussianity).
- Relatively recent in machine learning (conformal prediction).
- New notation: **Y** is the set of all possible labels (**label space**).

In conformal prediction, we want to achieve guaranteed validity.

We want to see a probability of error, like in stats/VaR we have confidence level.

Conformal prediction (1)

- Idea of conformal prediction: given a training set and a test sample, try in turn each potential label for the test sample.
- For each postulated label, we look at how plausible the extended training set is (under the IID assumption).
- We can make a confident prediction if all but one completion looks implausible.
- To evaluate the implausibility of the extended training set we use the statistical notion of a **p-value** (to be defined).

For Each **Postulated Label** :

So we look at how plausible the **extended training set** is:

This is the TRAINING SET + TEST sample with posutalted label.

So to predict under IID, we need to see how plausible, we will use p-values.

Basically how strange a extended training set is.

Conformal prediction (2)

- The first step is to define a **conformity measure**.
- This is a function that maps any finite sequence of labelled samples

$$z_1, \dots, z_m$$

to the corresponding **conformity scores**

$$\alpha_1, \dots, \alpha_m$$

and is required to be **equivariant**: if we permute z_1, \dots, z_m , the corresponding $\alpha_1, \dots, \alpha_m$ will be permuted in the same way.

To define a conformal predictor, we need a conformal measure.

Conformity measure is a way of evaluating how strange an observation looks

How strange a labeled sample looks

for z_1 to z_m (m number of labeled samples):

we assign each one a conformity score alpha. (real number)

equivariant, equivariance.

Conformal prediction (3)

- An equivalent way to express equivariance: α_i should be computable from z_i and the bag $\{z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_m\}$ (remember that a **bag** is a set with multiple copies of the same element allowed; also called a multiset).
- The intuition behind α_i : how well z_i conforms to the rest of the dataset.
- If α_i is small, we say that z_i is non-conforming, or **strange**.

if alpha (conformity score) is small, then it is strange, non-conforming, doesn't fit.

Conformal prediction (4)

This is how conformal predictors work on a training set z_1, \dots, z_n and test sample x^* :

- For each possible label $y \in \mathbf{Y}$ for x^* , compute the **p-value**

$$p(y) = \frac{\#\{i = 1, \dots, n+1 \mid \alpha_i^y \leq \alpha_{n+1}^y\}}{n+1},$$

where $\alpha_1^y, \dots, \alpha_n^y, \alpha_{n+1}^y$ are the conformity scores corresponding to $z_1, \dots, z_n, (x^*, y)$.

- If we are given a **significance level** $\epsilon > 0$ (our target probability of error), we can compute the corresponding **prediction set**

$$\Gamma^\epsilon = \{y \in \mathbf{Y} \mid p(y) > \epsilon\}.$$

Our training set z_1 to z_n (labeled samples) and we have a test sample x^* (no label, lets predict it)

So what we do is:

For each possible label y (element of \mathbf{Y}) on x^* :

Compute p-value from formula above.

Numerator: This is just the rank: Calculated by - number of conformity scores that are less than x^* conformity score.

so if $\alpha_i \leq \alpha_{x^*}$:

$$\text{rank} = \text{rank} + 1$$

when considering other labels y from \mathbf{Y} , all the alphas can change so we need to recompute them.

I believe his meaning of strange for p-values and conformity scores is different or just doesn't make sense. UPDATE: he mixes up smallest and largest when talking about alphas in the video ... **When alpha is the biggest, it is the least strange(most conforming).**

So in our formula:

$$\alpha_i \leq \text{bigest_alpha}$$

Is true for all so the rank is added by 1 everytime making the rank equal to denominator.

More examples / special cases below:

Special cases

To understand the formula for $p(y)$ on the previous slide, consider special cases:

- If (x^*, y) is the strangest labelled sample among $z_1, \dots, z_n, (x^*, y)$ (which it might well be if y is a wrong label), then $p(y) = 1/(n+1)$.
 - $1/(n+1)$ is the smallest possible value for $p(y)$
- If (x^*, y) is the second strangest labelled sample among $z_1, \dots, z_n, (x^*, y)$, then $p(y) = 2/(n+1)$.
- If (x^*, y) is the most conforming labelled sample among $z_1, \dots, z_n, (x^*, y)$, then $p(y) = 1$.

Useful terminology

- I will sometimes refer to the numerator in

$$p(y) = \frac{\#\{i = 1, \dots, n+1 \mid \alpha_i^y \leq \alpha_{n+1}^y\}}{n+1}$$

as the **rank** of α_{n+1}^y in the sequence (or bag) $\alpha_1^y, \dots, \alpha_{n+1}^y$.

- So the p-value $p(y)$ is defined as the rank of α_{n+1}^y over $n+1$.
- Roughly, the **rank** of α_{n+1}^y is k if α_{n+1}^y is the k th smallest element in $\alpha_1^y, \dots, \alpha_{n+1}^y$.

Validity and efficiency of conformal prediction

- Conformal predictors satisfy the following property of **validity** automatically: $y^* \notin \Gamma^\epsilon$ (the predictor makes a mistake) with probability at most ϵ (provided the labelled samples are IID).
- The property is easy to achieve (set $\Gamma^\epsilon = \mathbf{Y}$). We also want **efficiency**: in addition to validity, the prediction set should be small.

How to treat a tie?

pessimistic, make the rank as high as possible.

Plan

- ① Assumptions of machine learning
- ② Conformal prediction
- ③ Conformal prediction based of Nearest Neighbour
- ④ Validity and efficiency of conformal predictors

Assumptions of machine learning Conformal prediction Conformal prediction based of Nearest Neighbour Validity and efficiency of conformal predictors	Nearest Neighbour classification Nonconformity measures Nearest Neighbour regression Conformal prediction in an open world
--	---

Suitable conformity measures for 1-Nearest Neighbour

- The distance to the nearest sample of a different class.
- One over the distance to the nearest sample of the same class:

$$\frac{1}{\text{the distance to the nearest sample of the same class}}.$$

- Or we can combine the two ideas: the distance to the nearest sample of a different class divided by the distance to the nearest sample of the same class:

$$\frac{\text{the distance to the nearest sample of a different class}}{\text{the distance to the nearest sample of the same class}}.$$

sample is strange if it is far away from its class

2nd bullet point is one method to measure strangeness - conformity measure

or we can combine the two ideas from 1st,2nd bullet point.

Example for 3rd point:

Highly conforming sample IF :

Numerator is large - far frome different class

Denominator small - close to a sample of same class.

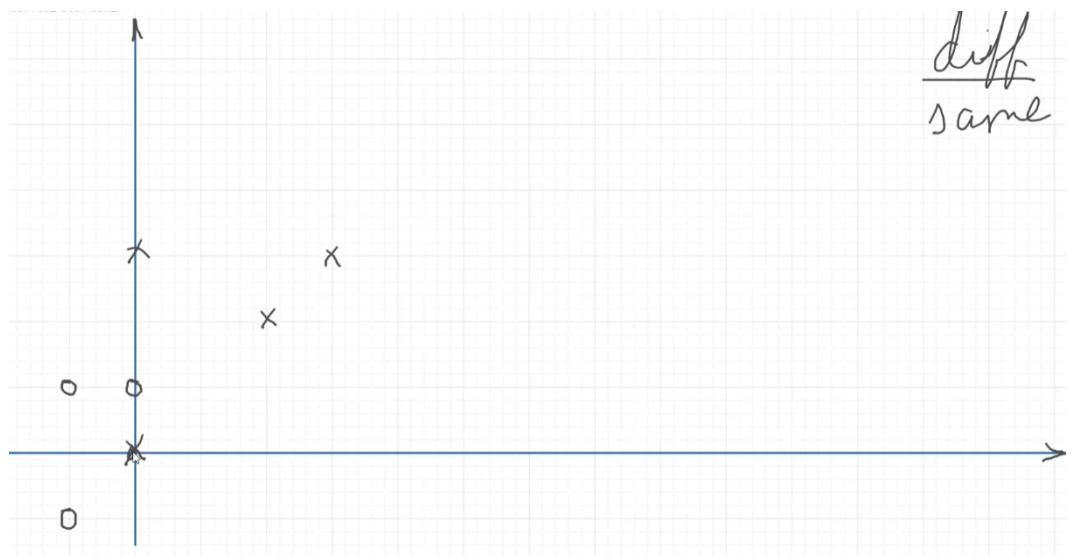
which in turn gives a high conformity score, so high likeliness

Assumptions of machine learning Conformal prediction Conformal prediction based of Nearest Neighbour Validity and efficiency of conformal predictors	Nearest Neighbour classification Nonconformity measures Nearest Neighbour regression Conformal prediction in an open world
<h2>Example (1)</h2>	

- Remember the training set in Chapter 2:
 - positive samples: $(0, 3), (2, 2), (3, 3)$
 - negative samples: $(-1, 1), (-1, -1), (0, 1)$.
- But now the test sample is $(0, 0)$, in the middle of the negative samples.
- What are the two p-values?
- As conformity measure, use the distance to the nearest sample of a different class divided by the distance to the nearest sample of the same class.

3rd video on week intro to conf predic

Graphing:



First using label of +1 for test sample (this looks strange)

Now with this example lets calculate conformity measures for the above scenario

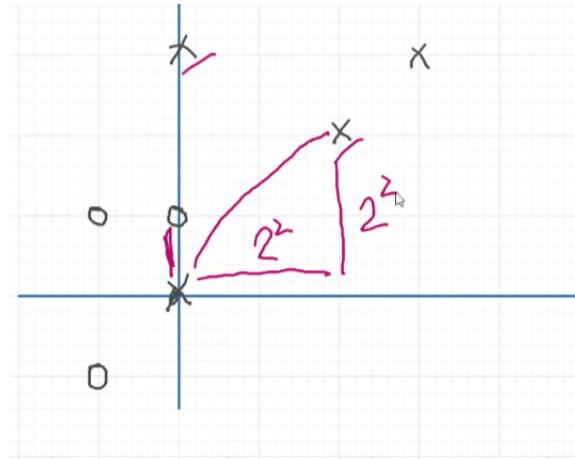
Example (2)

First assume the label of $(0, 0)$ is $+1$.

Sample	Label	Conformity score
$(0, 3)$	$+1$	$2/\sqrt{4+1} \approx 0.894$
$(2, 2)$	$+1$	$\sqrt{4+1}/\sqrt{1+1} \approx 1.581$
$(3, 3)$	$+1$	$\sqrt{9+4}/\sqrt{1+1} \approx 2.550$
$(-1, 1)$	-1	$\sqrt{1+1}/1 \approx 1.414$
$(-1, -1)$	-1	$\sqrt{1+1}/2 \approx 0.707$
$(0, 1)$	-1	$1/1 = 1$
$(0, 0)$	$+1 (?)$	$1/\sqrt{4+4} \approx 0.354$

The test sample is the strangest, and the p-value is $1/7 \approx 0.143$.

Now to calculate conformity scores, just go through NN algo like we showed 2/3 slides above. Lets show below how to calculate conformity score for $(0,0)$ aka x^* :



So we see:

$$\text{closest diff class / closest same class} == 1/\sqrt{4+4}$$

And now to find the p-value from above slide:

we first rank the conformity score for **postulated label**

0.354 is the smallest so only satisfies $\alpha_i \leq \alpha_{\text{postulated}}$, once. therefore rank=1

Now rank/n+1 = 1/7

Assumptions of machine learning
Conformal prediction
Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

Example (3)

Next assume the label of $(0, 0)$ is -1 .

Sample	Label	Conformity score
$(0, 3)$	+1	$2/\sqrt{4+1} \approx 0.894$
$(2, 2)$	+1	$\sqrt{4+1}/\sqrt{1+1} \approx 1.581$
$(3, 3)$	+1	$\sqrt{9+4}/\sqrt{1+1} \approx 2.550$
$(-1, 1)$	-1	$\sqrt{4+1}/1 \approx 2.236$
$(-1, -1)$	-1	$\sqrt{16+1}/\sqrt{2} \approx 2.915$
$(0, 1)$	-1	$2/1 = 2$
$(0, 0)$	-1 (?)	$\sqrt{4+4}/1 \approx 2.828$

The test sample is the second most conforming; its rank is 6, and the p-value is $6/7 \approx 0.857$.

less strange, higher conformity score

rank of test sample is 6

for all conformity scores

$\alpha_i \leq 2.828$ is satisfied 6 out of 7 times (2.915 is bigger)

$p \text{ val} = 6/7$ (from previous formula)

+1 label : pvalue is 13%

-1 label : pvalue is 86%

-1 is a better label

p values are not probabilities, dont need to sum to 1

Assumptions of machine learning
Conformal prediction
Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

Example (4)

- Notice: changing the postulated label changes plenty of conformity scores, not just one.
- The p-values are 0.143 (for +1) and 0.857 (for -1).
- We can predict -1, but our prediction does not achieve statistical significance (5%, as discussed below; for that, we would need at least 19 labelled samples in the training set).

Exercises for home

- Compute the two p-values taking as conformity score the distance to the nearest sample of a different class. [Answer:](#) 0.286 (for +1) and 0.714 (for -1).
- Compute the two p-values taking as conformity score one over the distance to the nearest sample of the same class. [Answer:](#) 0.143 (for +1) and 1 (for -1).
- What is your conclusion (if any) about the efficiency of the three conformity measures?

first bullet point: just ignore denominator of the diff/same scoring technique

Exercise for now

- The training set has only one feature:
 - positive samples: 0 and 1
 - negative samples: 10 and 11.
- The test sample is 12 (which seems to be in the negative area).
- What are the two p-values?
- As conformity measure, take the distance to the nearest sample of a different class.

4th video of intro to conf pred

SHOW WHAT TO DO WHEN EQUAL RANKS

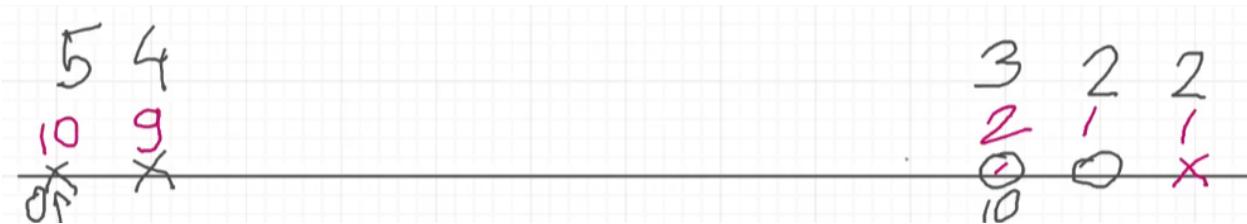
IF two conformity scores have the same value, use the larger of those two.

Conformity score: distance to nearest of different class:



We see two with conformity score 1.

Rank should be 1 and 2, but we will give them both rank 2. (from smallest to biggest)



in the above: top row is rank

bottom row is conformity score (diff)

So lets see how this works:

Postulated label == x

p-value:

for all alpha: $\alpha_i \leq \alpha_{\text{postulated}}$

we see on the two right most satisfy the above so p-value = 2/5

Nonconformity measures (1)

- Formally, nonconformity measures are defined in the same way as conformity measures.
- But their interpretation is different: α_i measures how **strange** (rather than how **conforming**) z_i is.
- The formula for computing p-values on slide 10, becomes

$$p(y) = \frac{\#\{i = 1, \dots, n+1 \mid \alpha_i^y \geq \alpha_{n+1}^y\}}{n+1}$$

(the only difference is that \leq becomes \geq).

opposite of conformity

keep p value but rank is counted opposite, from the other side

How to use formula:

go over Example(1) above

Nonconformity measures (2)

- In principle, it does not matter whether you use nonconformity measures (the Royal Holloway convention) or conformity measures (the Carnegie Mellon convention).
- Instead of using nonconformity scores α_i , you can instead use conformity scores $-\alpha_i$ (or $1/\alpha_i$ if α_i are positive).
- But in application to regression nonconformity scores are often more convenient.

Exercise for now

- Consider the same training set as before (slide 21):
 - positive samples: 0 and 1
 - negative samples: 10 and 11.
- The test sample is still 12.
- What are the two p-values?
- But now we use a nonconformity measure, namely: the distance to the nearest sample of the same class.

Now using a nonconformity measure, distance to the nearest sample of same class.



We can see nonconformity scores seem much more manageable, most are 1, **now to rank.**

nonconformity scores = alpha_1 → alpha_n+1

alpha_n+1 = postulated label sample

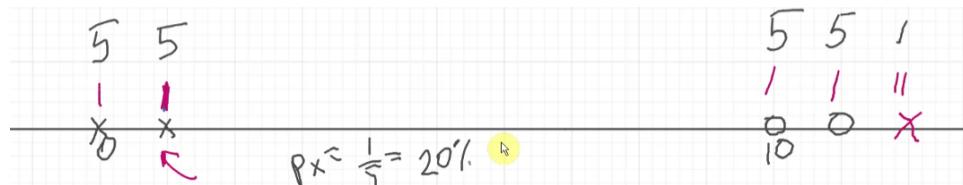
for all nonconformity scores:

alpha_i ≥ alpha_n+1

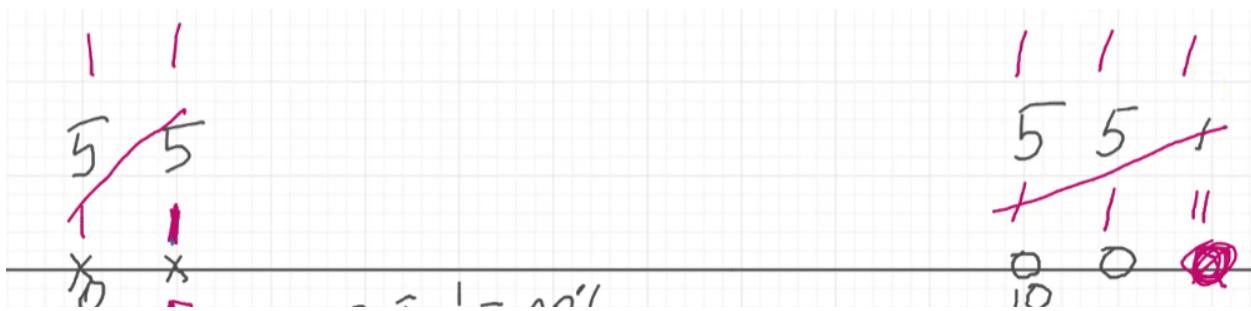
This is true once so rank 1.

p-value = 1/5

Using the other method where we rank instead of checking inequality:



NOW lets do the other postulated label



Everything is close to same class so all have nonconformity scores of 1.

Now to rank:

For all nonconformity scores:

$\alpha_i \geq \alpha_{\text{postulated}}$

We satisfy the above 5 times so rank 5

p-value = 5/5

**So far we have discussed classification,
Now for Regression**

Two families of nonconformity measures

- Perhaps the most popular class of nonconformity measures in the case of regression is

$$\alpha_i = |y_i - \hat{y}_i|,$$

where \hat{y}_i is a prediction for y_i . Advantage: mathematical simplicity; facilitates efficient computations.

- Another popular class of nonconformity measures in the case of regression is

$$\alpha_i = |y_i - \hat{y}_i| / \sigma_i,$$

where \hat{y}_i is a prediction for y_i and $\sigma_i > 0$ is an estimate of its accuracy. Advantage: the size of the prediction set is more adaptive.

Now it is a bit different to define a nonconfromity score:

given a bag of labeled samples: $(x_1, y_1) \dots (x_m, y_m)$: we need to find their nonconformity scores of these samples:

How do we define it? Given the whole set we find a prediction rule:

We can say our i th sample conforms well if the true label is close to the predicted label.

$$y_i - \hat{y}_i$$

where \hat{y}_i is the predicted label

THINK ABOUT IT: if the difference is small, its close to the real thing, its conforming

And vice versa for if the difference is big

2nd bullet point is an alternative which we will not use in the course

Example:

set of points

one point looks strange

how to compute non conformity score?

Fit a straight line through the main cloud of points

define s as the non conformity score of sample.

$y_i(\hat{y})$ is predicted value

Assumptions of machine learning
Conformal prediction

Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

A nonconformity measure based on Nearest Neighbour

In this chapter we consider an element of the first family: the nonconformity scores

$$\alpha_1, \dots, \alpha_m$$

of labelled samples

$$(x_1, y_1), \dots, (x_m, y_m)$$

are defined by $\alpha_i = |y_i - \hat{y}_i|$, where \hat{y}_i is the label of the nearest neighbour of x_i among $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_m$.

y_i is the label of nearest neighbor of x_i (transduction)

Assumptions of machine learning
Conformal prediction
Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

Difficulty

- The main difficulty for regression problems is that there are infinitely many potential labels to consider.
- One possible solution is to consider a dense finite grid of potential labels, and then for each possible label in the grid compute its p-value.
- Occasionally another solution is possible: we can derive a formula (or a very efficient algorithm) for the prediction set. This is the case for K Nearest Neighbours, Least Squares regression, Ridge Regression, and Lasso (the last three algorithms will be discussed in later chapters).

Example of querying specific labels

- Consider the training set

$$(x_1, y_1) = (2, 0), \quad (x_2, y_2) = (1.2, 2), \\ (x_3, y_3) = (1, 1), \quad (x_4, y_4) = (0, 2)$$

consisting of four labelled samples.

- Find the p-values for the test labelled samples $(x, y) = (4, 0)$ and $(x, y) = (2.5, 1)$.

4 observations in training set.

Lets see how to comute p-values

Solution for $(x, y) = (4, 0)$

Sample	Label	Label of the NN	NS
2	0	2	$ 0 - 2 = 2$
1.2	2	1	$ 2 - 1 = 1$
1	1	2	$ 1 - 2 = 1$
0	2	1	$ 2 - 1 = 1$
4	0 (?)	0	$ 0 - 0 = 0$

Here NN stands for “Nearest Neighbour” (in the extended training set) and NS stands for “nonconformity score”. The test sample is the least strange, and the p-value is 1.

We are using **Nearest Neighbor as our nonconformity score**, so we first find the nearest neighbor of each sample, for example nearest for $x,y(2,0)$ is $x,y(1.2, 2)$ so the label is 2.

nearest for $x,y(1.2, 2)$ is (1, 1) so label for NN is 1.

Now to compute the **nonconformity score**

NS - nonconformity score

$NS = \text{label of sample} - \text{predicted label}$ ($\alpha_i = y_i - \hat{y}_i$)

$$\alpha_i = |y_i - \hat{y}_i|,$$

predicted label is found by doing 1NN, one nearest neighbour.

For each sample predict NS

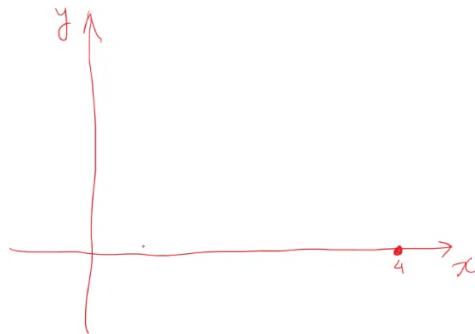
start with sample 4:

$$(x, y) = (4, 0)$$

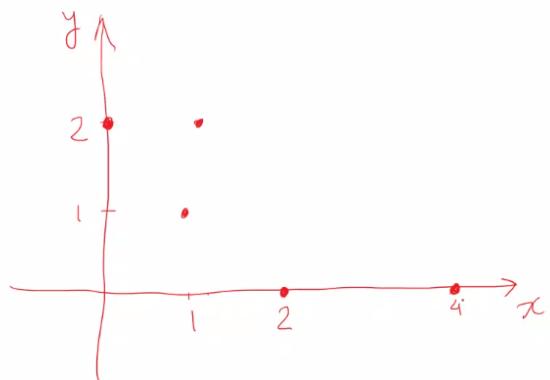
postulated label is

nearest neighbor is (2,0) so its NN label is 0

and $NS = 0 - 0 = 0$



now putting all other samples on graph



sample 4, has p value = 1

Now lets try for (2.5, 1)

postulated label = 1

we find NS

all NS are the same so rank is 5

p-value = 5/5

In these examples, we were given postulated labels, what if we dont have these?

Using a grid (1)

- In regression problems, Γ^ϵ is typically an interval, $\Gamma^\epsilon = [a, b]$.
- A crude way to compute it is to choose a large interval $[A, B]$ (containing all interesting values for the label y) and choose a dense grid in it: $\{A, A + \text{step}, A + 2\text{step}, \dots, B\}$ for a small $\text{step} > 0$.
- Go over all y in the grid and output the range $[a, b]$ of y for which $p(y) > \epsilon$.

epsilon is some threshold

So what we can do is go over all y (labels) in the grid and we output from $[a, b]$ where $p(y) > \text{epsilon}$

Using a grid (2)

This is a possible snippet of Python code, where

- p is a function computing $p(y)$ given y ,
- `arange(A,B,step)` is (in NumPy) $\{A, A + \text{step}, A + 2\text{step}, \dots, B\}$ (not including B),
- `epsilon` is ϵ ,
- and `NaN` is an undefined value (“not a number”).

```
a = NaN
b = NaN
for y in arange(A,B,step):
    if p(y) > epsilon:
        b = y
    if a == NaN:
        a = y
```

choose big set $[A, B]$ that has all possible sample labels for example

then use the big set to find sub set $[a, b]$ to find prediction interval

A shortcut

- Consider the same training set

$$\begin{aligned}(x_1, y_1) &= (2, 0), & (x_2, y_2) &= (1.2, 2), \\ (x_3, y_3) &= (1, 1), & (x_4, y_4) &= (0, 2)\end{aligned}$$

consisting of four labelled samples.

- The test sample is $x^* = 4$. What is the prediction set at the significance level 20%?

Same set as before.

test $x^* = 4$:

epsilon = 20%

We can choose the smallest acceptable significance level by seeing what the smallest possible p-value is: which is $1/5$ aka $1/n+1$

calculating like before below:

Solution for $x^* = 4$

Sample	Label	Label of the NN	NS
2	0	2	$ 0 - 2 = 2$
1.2	2	1	$ 2 - 1 = 1$
1	1	2	$ 1 - 2 = 1$
0	2	1	$ 2 - 1 = 1$
4	y	0	$ y - 0 = y $

The p-value is 20% (or less) if the test labelled sample is the strangest. In other words: if $|y| > 2$. The prediction set at 20%:

$$\Gamma^{20\%} = [-2, 2].$$

It contains 0, as we already know (see slide 29).

for test sample we write label as y :

y can be any number/label

So when do we get a p-value of 20% or less?

if the test labelled sample is the strangest 20%

the nonconformity score should be the largest

to be the largest it is more than 2, because 2 is the other largest nonconformity score, so 3 and above is what we need for 20% p-value

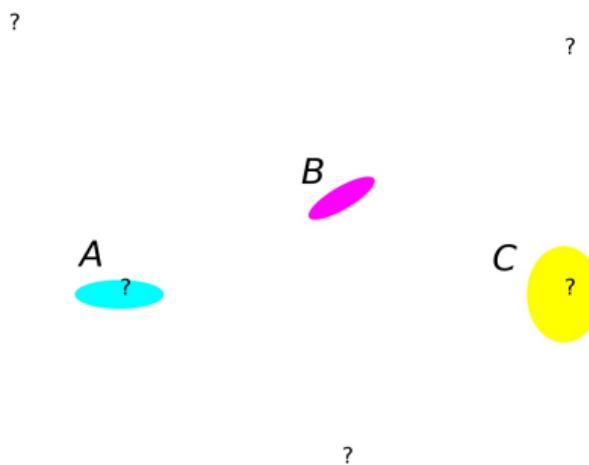
WHY IS IT -2 to 2 ??????????????????

because when y is bigger than 2 or smaller than -2, $|y|$ makes the pvalue smaller than 20%

Conformal prediction for anomaly detection

- It is possible that no postulated label for the test sample is plausible (all p-values are small). What does it mean?
- Consider three families of computer viruses as shown on the next slide (three compact clouds) and think what happens if the new virus is far from any of the clouds.
- The slide after that: a numeric illustration.
- For simplicity, take the distance to the same class as nonconformity measure.
- We previously have known every possible label, eg we know if it will be male or female
 - But what if we don't know?

Three families and several test samples



example: computer virus:

each PC is represented as a point in the plane. and some measurements about points of how infected or not.

training set consists of 3 clouds A B C

family A virus, B and C etc.

If new PC arrives we want to know which virus it has, maybe family C ...

If new PC is near C then it will probably be C family of virus

p-value of A = << 1

p-value of B = << 1

p-value of C = </< 1 (p of c is big!)

Or

if PC has all values small maybe its part of a different family ?!

Assumptions of machine learning
Conformal prediction
Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

Exercise (1)

- Consider the following training set with three classes (such as virus families) A, B, and C:
 - 0 and 1 are labelled A;
 - 5 and 6 are labelled B;
 - 10 and 11 are labelled C.
- Using the distance to the same class as nonconformity measure, compute the p-values for 0.5, 5.5, and 3.
 - Answer for 0.5: $p_A = 1$, $p_B = 1/7$, $p_C = 1/7$.

Two points, 0 and 1 (A and A)

and also points 5, 6 label (B and B)

and point 10 and 11 label (C and C)



We are asked to compute test samples:

0.5 test sample:

our prediction:

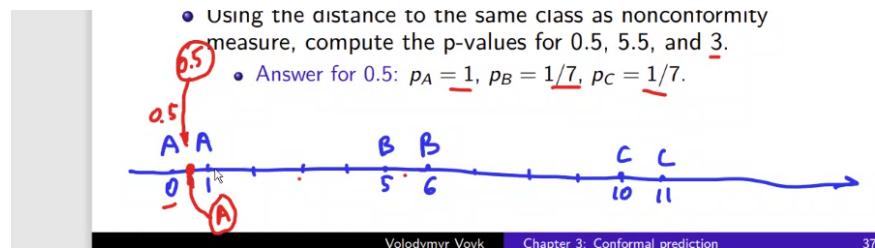
Distance to same class as non conformity measure

so nearest distance is 0.5 so looks like A

p_A is 1, p_B is small and p_C is small so its likely A ...

3 test sample:

- To find p-value with label A for 0.5: (p_A)

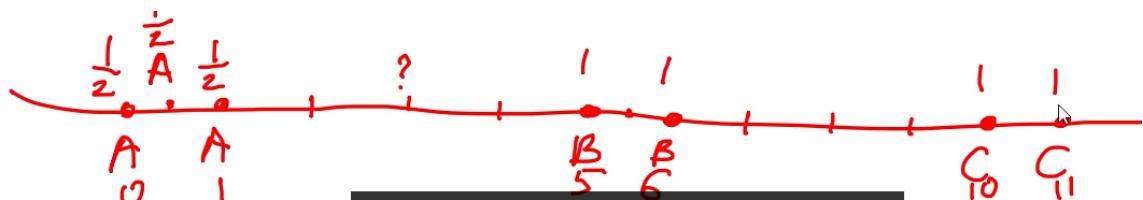


this is same for 0 and 1

0, 1, 0.5 all have NN score : 0.5

5, 6, 10, 11 all have NN score : 1

like below:

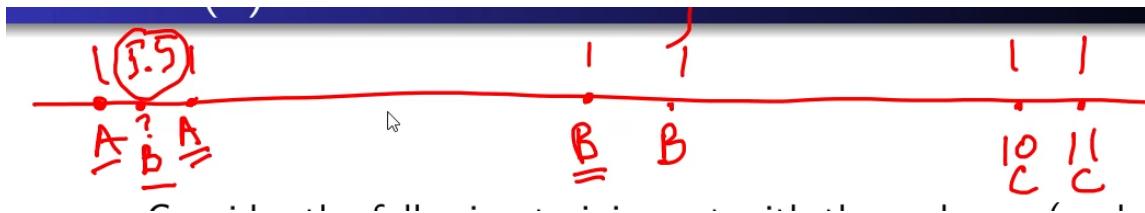


- So points [0, 0.5, 1] have a rank of 7

- $\alpha_i \geq \alpha_{\text{postulated}}$ (NS of 0.5 for sample(0.5))
 - this is true 7 times

so p-value = $7/6+1 = 1$

- To find p-value with label B for 0.5: (p_B)



- Nonconformity scores are shown above, with a label of B. 0.5 has a NS of 5.5
 - This looks strange
- It is the largest NS so naturally it has the rank 1, nothing but itself satisfies $\alpha_i \geq \alpha_{\text{postulated}}(5.5)$
 - 5.5 is the biggest number
- p-value
 - $1/7$

Imao this was me at the start of term

non vs normal just changes the way you do ranking !!!!

conformity score, higher score means larger rank

non conformity, higher score lower rank !!!!!!!!!!!!!!!

On-line Lecture 14/10/22

Assignment:

talk

Assumptions of machine learning
Conformal prediction
Conformal prediction based of Nearest Neighbour
Validity and efficiency of conformal predictors

Nearest Neighbour classification
Nonconformity measures
Nearest Neighbour regression
Conformal prediction in an open world

Exercise (2)

- What should be our conclusions?
- In the case of the test samples 0.5 and 5.5 we can make a confident prediction: the families are A and B , respectively.
- For the test sample 3, all p-values are low. It looks as if we have a new family (or, otherwise, the test sample is an unusual representative of an old family).

old family meaning that it should belong to A for example but A has just changed a lot over time.

Assumptions of machine learning
 Conformal prediction
Conformal prediction based of Nearest Neighbour
 Validity and efficiency of conformal predictors

Nearest Neighbour classification
 Nonconformity measures
 Nearest Neighbour regression
 Conformal prediction in an open world

Confidence and credibility (1)

- Let p_A , p_B , and p_C be the three p-values for a test sample.
- We can make a confident prediction if all p-values apart from one are very small.
- We can summarize our prediction as follows:
 - the **point prediction** is the label with the largest p-value (A for the test sample 0.5 on slide 37);
 - our **confidence** is one minus the second largest p-value (6/7 for the test sample 0.5);
 - the **credibility** is the largest p-value (1 for the test sample 0.5).

if p_A is big and p_B and p_C are small then we can see it should be p_A
 our example from above:

$$p_A = 1$$

$$p_B = 1/7$$

$$p_C = 1/7$$

Different methods:

- **Point Prediction:** Give the label to the largest pValue
- **Confidence level:** take second largest pVal, if it is small we are confident (obvs 1-pval will be big hopefully)
 - $1 - 1/7 = 6/7$
- **Credibility** is largest pVal: $7/7 = 1$, our example looks not suspicious, we have high

Confidence and credibility (2)

- Therefore, we can make a confident prediction if the confidence is high (close to 1) and the credibility is not low.
 - If the credibility is very low: are we witnessing a new class?
 - But it is not a good idea to measure the performance of your conformal predictor by, say, the average confidence on the test set. We will see a much better way in the next section.
-
- a new class:
 - if we ignore 2nd and 3rd ... etc pValues

its better to use average false p value. later we see

Proof of validity for $\epsilon = 1/(n+1)$

- What is the probability that $y^* \notin \Gamma^{1/(n+1)}$? (Cf. slide 13.)
- Notice that $y^* \notin \Gamma^{1/(n+1)}$ means that $z^* = (x^*, y^*)$ is the strangest labelled sample in the set z_1, \dots, z_n, z^* .
- By the IID assumption, all permutations of z_1, \dots, z_n, z^* (and the corresponding permutations of $\alpha_1, \dots, \alpha_n, \alpha_{n+1}$) have the same probability.
- So the probability that the smallest conformity score in the bag $\{\alpha_1, \dots, \alpha_{n+1}\}$ will be the last one is exactly $1/(n+1)$ (provided there is only one smallest element in the bag; otherwise the probability of error will be less, 0).

Why conformal predictors are valid:

- Epsilon is our target probability of error.
 - Why is epsilon the probability of error (or an upperbound):
 - What is the probability that $y^* \notin \Gamma^{1/(n+1)}$?
- This is saying: What is the probability that our prediction (y^*) fails to cover the true label.

$$x^* \quad y^*$$

- This is our test sample, What is the chance y^* is not in the predicton set?

For validity:

prediction set: gamma epsilon

Proof of validity for $\epsilon = 1/(n+1)$

Γ^ϵ

This is our theoretical lower bound. So we suppose epsilon = above.

Our definition of **Prediction Set** :

$$\Gamma(y) = \{y \mid p(y) > \epsilon\}$$

- Is when $p\text{-value}(y) > \epsilon$
 - So we get all the y labels when above is true in our prediction set.
- y is not in the prediction set when $p\text{-value}$ of y is small:
 - just opposite of the above inequality.

$$p(y) \leq \epsilon$$

OR

3) $p \leq \frac{1}{n+1}$

we get p value of $1/n+1$ when it is the last one ranked, the probability of this occurring is $1/n+1$

There are $n+1$ possible positions

Watch second last week3 video 5:00 for explanation with 1 and 2 items in a bag

probability error at most is $1/n+1$, when all conformity scores are different

I think these 2 lines above kind of sum up the proof?

Assumptions of machine learning Conformal prediction Conformal prediction based of Nearest Neighbour Validity and efficiency of conformal predictors	Argument for validity Statistical significance Main property of validity (*) How to measure efficiency
Exercises for home	

- **Exercise 1:** Show that the probability of $y^* \notin \Gamma^{2/(n+1)}$ does not exceed $2/(n + 1)$.
- **Exercise 2 (optional):** Show that the probability of $y^* \notin \Gamma^{k/(n+1)}$ does not exceed $k/(n + 1)$, for any $k \in \{1, 2, \dots, n\}$.

Statistical significance

- In statistics, p-values are used for testing statistical hypotheses.
- If we obtain a p-value $\leq 5\%$, the result is **statistically significant**.
- If we obtain a p-value $\leq 1\%$, the result is **highly statistically significant**.
- In conformal prediction, we are testing the IID assumption.
- The standard statistical conventions call for paying particular attention to $\Gamma^{5\%}$ and $\Gamma^{1\%}$.

Nested prediction sets

But it's best to look at what happens at more than two significance levels. For example: we can call

- $\Gamma^{20\%}$ casual prediction,
- $\Gamma^{5\%}$ confident prediction,
- $\Gamma^{1\%}$ highly confident prediction.



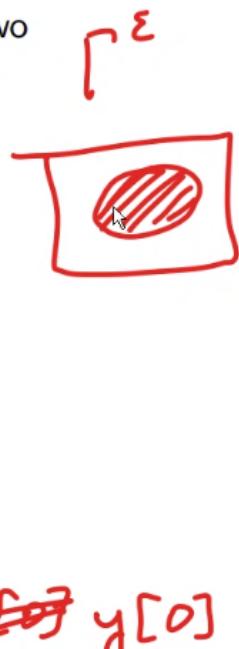
example: prediction where is tank.

lightest gray is the 1%, we are 1% sure that the tank is somewhere in the lightest gray.

Nested prediction sets

But it's best to look at what happens at more than two significance levels. For example: we can call

- $\Gamma^{20\%}$ casual prediction,
- $\Gamma^{5\%}$ confident prediction,
- $\Gamma^{1\%}$ highly confident prediction.



gamma epsilon is saying its somewhere in the highlighted area

casual predictiton is the small black area

confident predictiton is the light grey

light shaded is highly confident region

Randomized p-values

- In theoretical (never, or almost never, experimental) work people usually use **randomized p-values**

$$p(y, \tau) = \frac{\#\{i : \alpha_i^y < \alpha_{n+1}^y\} + \tau \#\{i : \alpha_i^y = \alpha_{n+1}^y\}}{n+1},$$

where $i = 1, \dots, n+1$ and $\tau \in [0, 1]$ is chosen independently from the uniform distribution on $[0, 1]$.

- Notice: $p(y, \tau) \leq p(y, 1) = p(y)$; now $p(y, \tau) < 1/(n+1)$ is possible.
- Using randomized p-values can only make our prediction sets smaller (and so validity will be more difficult to achieve).

If you want your probability of error to be exactly epsilon, we randomise.

- When you compute randomised p-values:
 - have to be careful about ties.
 - from the normal formula: $\alpha_i \geq \alpha_{\text{postulated}}$, we compute the rank of our augmented training set.

BUT:

- Here when comparing **conformity score of test sample** to the **conformity scores of training set**

OR

Comparing to the conformity score of the **test sample its self**

- We have to be careful of equalities
 - we multiply by **Tao**

Tao is a random number generator number between 0-1

- In the standard definition of p-values, we had $\alpha_i \leq \alpha_{\text{postulated}}$ (conformal prediction algorithm)
 - Its like saying $Tao = 1$

So now with this extra Tao, we will get smaller p-values and the prediction sets will become smaller.

So now we can get p-values less than $1/n+1$

if training set is big, then the right side of denominator will be a small set (typical size 1)

Assumptions of machine learning Conformal prediction Conformal prediction based on Nearest Neighbour Validity and efficiency of conformal predictors	Argument for validity Statistical significance Main property of validity (*) How to measure efficiency
Prediction in the online mode	

Let P be a probability measure on \mathbf{Z} (the labelled samples) and U be the uniform probability measure on $[0, 1]$.

Protocol

ONLINE MODE OF PREDICTION

```

    generate a labelled sample  $z_1 = (x_1, y_1) \sim P$ 
    for  $n = 1, 2, \dots$  do
        generate a new labelled sample  $z_{n+1} = (x_{n+1}, y_{n+1}) \sim P$ 
        independently
        generate a new random number  $\tau_n \sim U$  independently
        compute  $p(y, \tau_n)$  for each potential label  $y$  for  $x_{n+1}$  as test
        sample from  $z_1, \dots, z_n$  as training set
        set  $p_n = p(y_{n+1}, \tau_n)$ 
    end for
  
```

- Online mode works like this:
 - Before making prediction, Generate a labelled sample $z_1 = (x_1, y_1)$ generated from P
 - For infinite loop
 - generate a new labeled sample z_{n+1} and try to predict it (from P I think)
 - generate new random number Tao (0 to 1)
 - compute $p(y, tao)$ for each potential label of our test sample. All the labels are coming from training set
 - So now we have p-values for all possible test labels
 - then we can form a prediction set from it
 - we also record the p-value of the true label (y_{n+1}) and p_n is the resulting value.

The property of validity

Theorem

In the online mode of prediction, the consecutive p-values p_1, p_2, \dots are independent and distributed uniformly on $[0, 1]$.

- In particular, conformal predictors make errors at different steps independently with probability ϵ .
- **Optional remark 1:** the independence allows us to use the law of large numbers (so that the percentage of errors over the first n steps is close to ϵ with high probability when n is large).
- **Optional remark 2:** the proof of the theorem uses a backward argument.

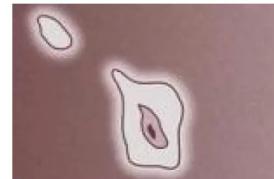
Average false p-value

How do we measure the efficiency of conformal predictors?

- In the case of regression, we could look at the area of the nested prediction sets such as those given on slide 45 (details omitted).
- In this subsection we only discuss the case of classification.
- The **average false p-value**: the average of the p-values for all postulated labels in the test set except for the true labels.

So here we are talking about efficiency instead of validity above.

- $\Gamma^{20\%}$ causal prediction,
- $\Gamma^{5\%}$ confident prediction,
- $\Gamma^{1\%}$ highly confident prediction.



We can say the 1% is efficient

Anyway idk why he said that: back to the slide

- Suppose we have test sample
 - for each possible label, compute the p-value
 - for the true label the p-value is not under our control (it is uniformly distributed)

- but the rest of the p-values should be as small as possible, as we want to be able to detect if a label is false.

Assumptions of machine learning Conformal prediction Conformal prediction based of Nearest Neighbour Validity and efficiency of conformal predictors	Argument for validity Statistical significance Main property of validity (*) How to measure efficiency
Exercise	

- Suppose we have obtained the following p-values when applying a conformal predictor to a test set of size 4 and with the label space $\mathbf{Y} = \{A, B, C\}$:

True label	p_A	p_B	p_C
B	0.05	0.3	0.05
A	0.7	0.02	0.08
B	0.04	0.4	0.06
C	0.01	0.09	1

- Find the average false p-value. [Answer:](#) 0.05.
- This is the way you are asked to measure the efficiency of your conformal predictor in Assignment 1.

here we have three labels: A B C

how to compute false pVal: definiton is 2nd slide above.

- false just means pVal for a label which is not the true label.
- How it works:
 - for each test sample:
 - we have true label(above)
 - for first one (B)

- we see p_B is bigger so we expect it to be a B
- For 2nd
 - we see p_A is biggest but lets compute average false p val
 - 1:00:00 from recording

True label	p_A	p_B	p_C
B	0.05	0.3	0.05
A	0.7	0.02	0.08
B	0.04	0.4	0.06
C	0.01	0.09	0.1

← ↑ → //

Here we get rid of the ones with the high label pVal and then we take the avg of the remaining numbers which is 0.05 answer like in the slide above!!!

Week 4 Lecture Thursday:

Assignment help:

Code for 1NN:

i - range of test set

for i = 1, ..., k x^{*i}

 for j = 1, .. , n : x_j

 if $|x_j - x^{*i}| < \text{distance}$

 distance = $|x_j - x^{*i}|$

$y = y_i$ update label of nearest

 distance from x^{*i} to x_j

Chapter 4: General principles of machine learning

Volodymyr Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS5920 Machine Learning
Last edited: October 20, 2022

General principles of machine learning

Plan

1 Underfitting and overfitting

Generalization, overfitting, and underfitting

- In supervised learning, we want to build a model on the training data and then be able to make accurate predictions on new, unseen data that has the same characteristics as the training set that we used.
 - If a model is able to make accurate predictions on unseen data, we say it is able to **generalize** from the training set to the test set.
 - We want to build a model that is able to generalize as accurately as possible.
-
- Supervised learning: goal is to predict an unknown label
 - try to build a model using training data and make accurate predictions on test data

the two obstacles are overfitting and underfitting

overfitting more dangerous for naive people.

Example data (selling boats)

Table 2-1. Example data about customers

Age	Number of cars owned	Owns house	Number of children	Marital status	Owns a dog	Bought a boat
66	1	yes	2	widowed	no	yes
52	2	yes	3	married	no	yes
22	0	no	0	married	yes	no
25	1	no	1	single	no	no
44	0	no	2	divorced	yes	no
39	1	yes	2	married	yes	no
26	1	no	2	single	no	no
40	3	yes	1	married	yes	no
53	2	yes	2	divorced	no	yes
64	2	yes	3	divorced	no	no
58	2	yes	2	married	yes	yes
33	1	no	1	single	no	no

- Small dataset, makes it easy to overfit
 - if big its harder to overfit, prediction model has to be complicated to overfit.

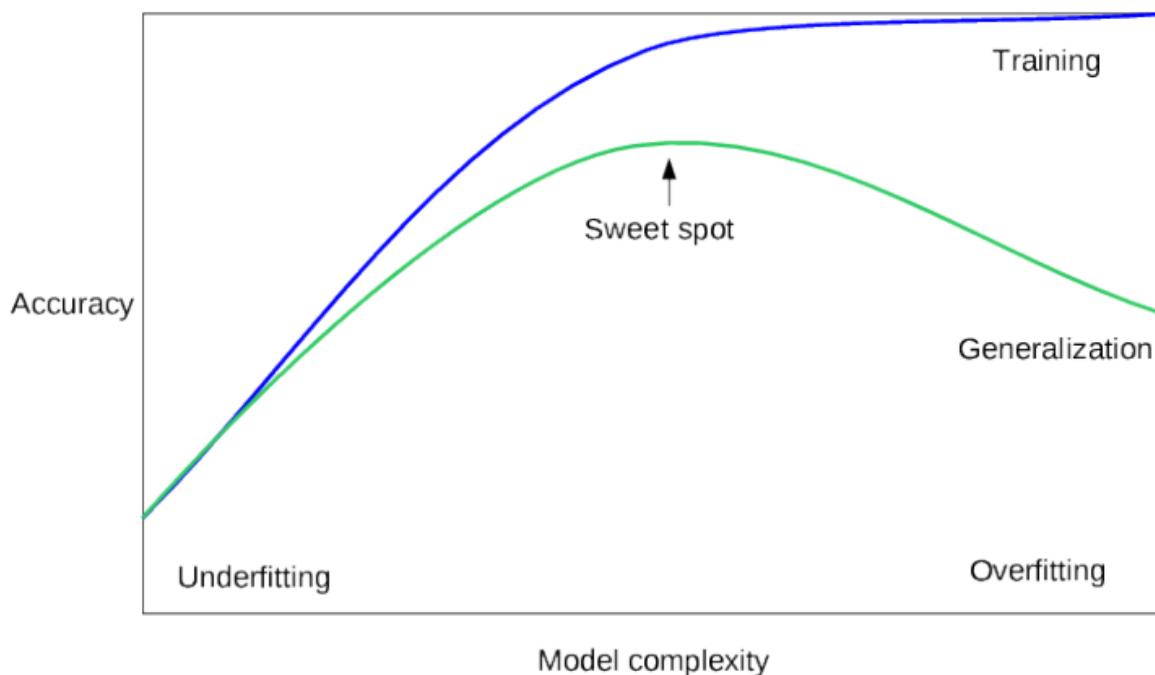
To show how easy it is to overfit: look at next slides:

Different models

- “If the customer is older than 45, and has less than 3 children or is not divorced, then they want to buy a boat.” This rule is 100 percent accurate! But unlikely to perform well on the new data. **Overfitting**.
- No age appears twice in the data, so we could say people who are 66, 52, 53, or 58 years old want to buy a boat, while all others don’t. **Overfitting**.
- First rule is too complicated... fits data too well.

How to avoid this: next

Looking for a balance



- For overfitting:
 - the **Model Complexity is important**
 - not easy to define
 - If too complex we are overfitting
 - What happens in many cases, suppose you fit different models to your data (training set) and you make it model more complicated.
 - you try to choose one model that fits it better so you keep making it more complicated, it becomes easier to fit the data well.

From the above anecdote, we see the training set accuracy will be monotonic, trying too hard to get high accuracy. But the test set accuracy (green line).

Here Generalisation is just the test set accuracy for an infinitely large test set

detecting sweet spot is hard, we only see the blue line initially.

assume data of both sets are from same distribution

error rate is opposite of green line

Underfitting and overfitting
K Nearest Neighbours
Learning curves

Examples
Balancing
Two sample datasets

Training vs test accuracy

- The blue curve on the previous slide: the accuracy on the training set (**training accuracy**).
- The green curve: the accuracy on a very large test set (**generalization accuracy**).
- The training accuracy is typically a monotonic (increasing) function of the model complexity.
- For the test accuracy, there is a “sweet spot” where it attains its maximum.
- If, in the case of classification, we use the error rate instead of the accuracy (one minus the error rate), we get a U-shaped curve.
- The green curve is an **inverted U**.

For classification we like to invert the lines above

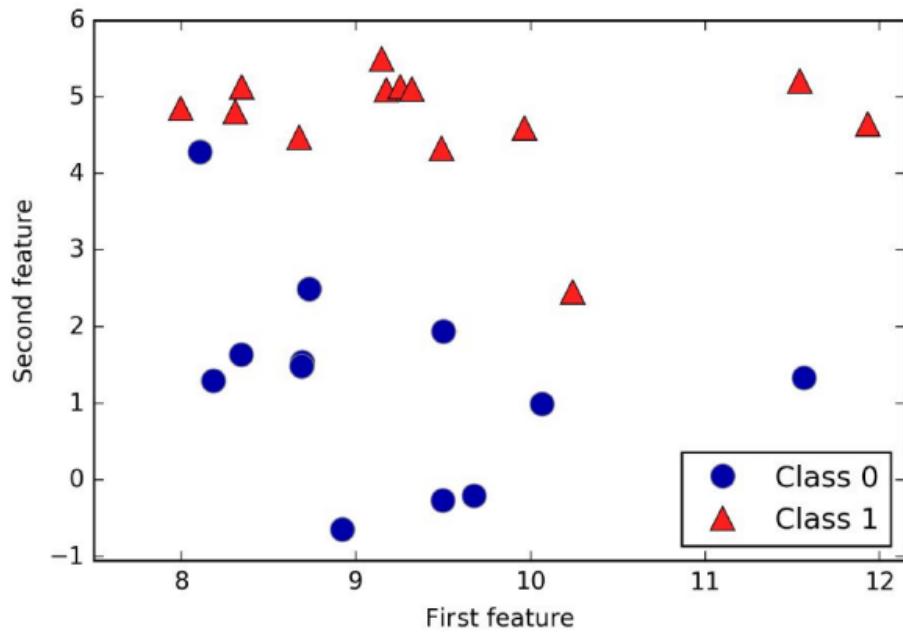
General tendency

- The optimal model complexity tends to grow with the size of the dataset.
- In this module, we focus on datasets of fixed sizes. But in the real world, you often have the ability to decide how much data to collect, which might be more beneficial than tweaking and tuning your model.

General Rules

- optimal model complexity tends to grow with size of dataset
 - huge training set, can afford huge models

forge dataset (classification)



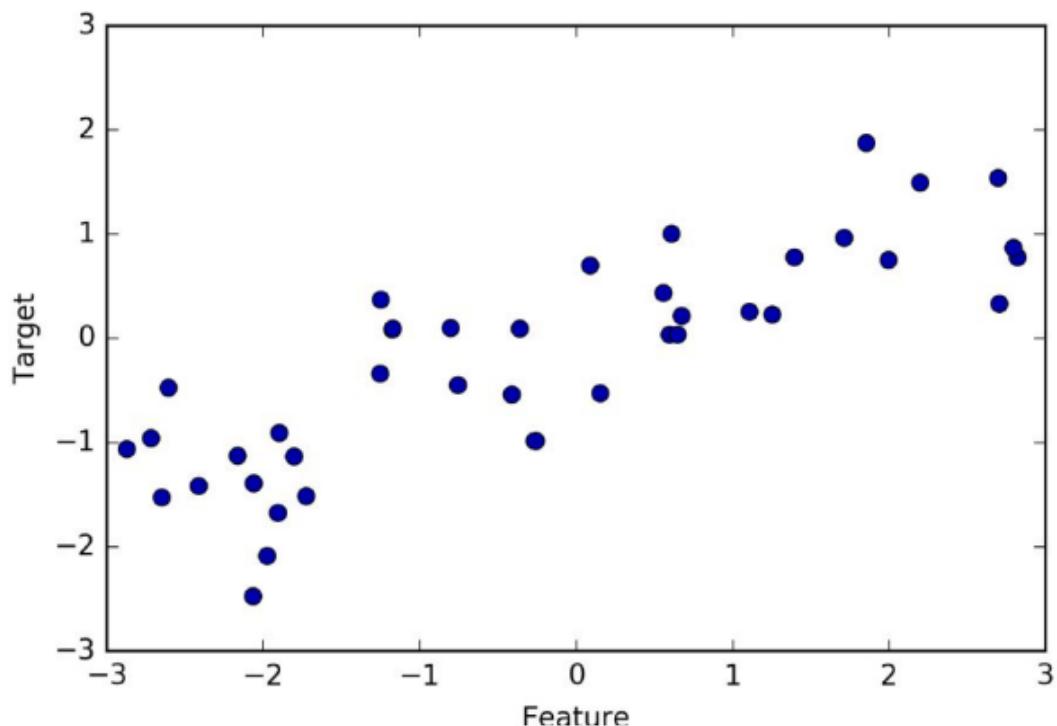
binary classification

both axis are our features

label is the shape: circle, triangle

We will also have a regression dataset: below:

wave dataset (regression)



target is the label

regression only has one feature, 1D dataset

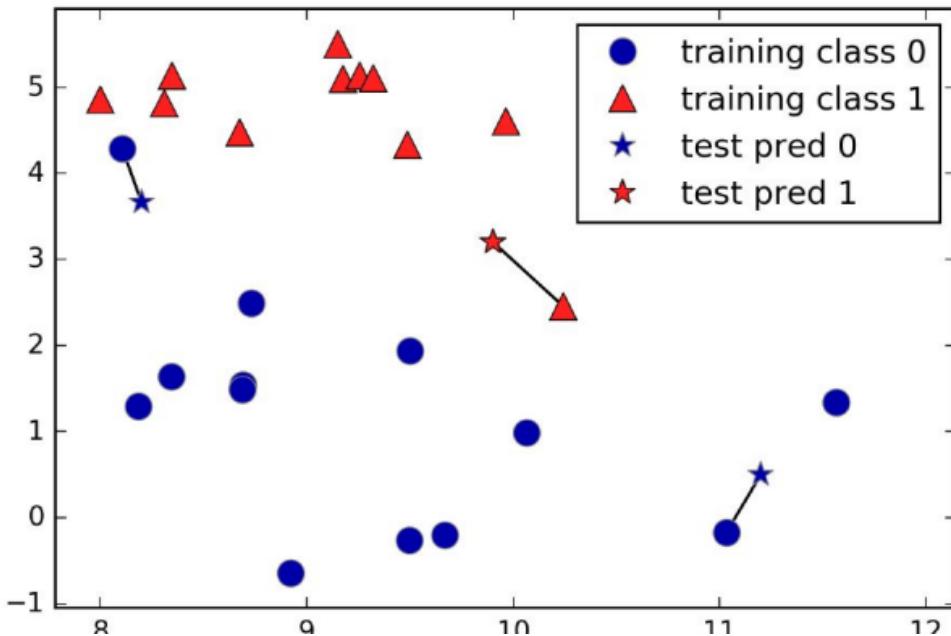
Plan

1 Underfitting and overfitting

2 K Nearest Neighbours

3 Learning curves

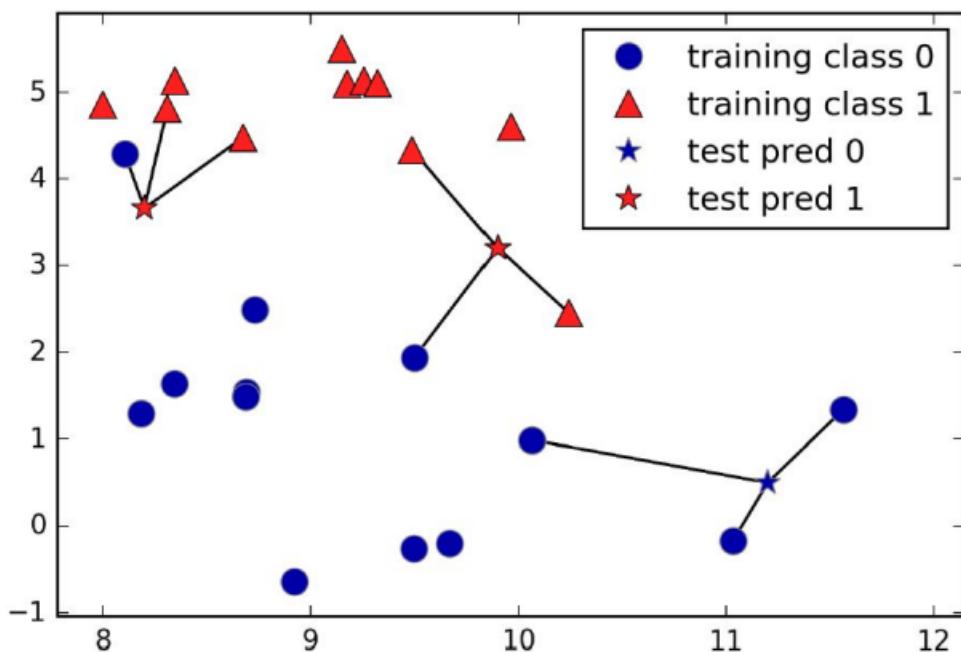
1NN classification on forge (3 test samples)



- Stars are test sample.
 - lets predict their symbol/color
- Suppose we have test_pred_0 as our first test:
 - We have NN as Circle
 - our prediction is circle
 - do the same for test_pred_1
 - its triangle

for 3NN:

3NN classification on forge (3 test samples)



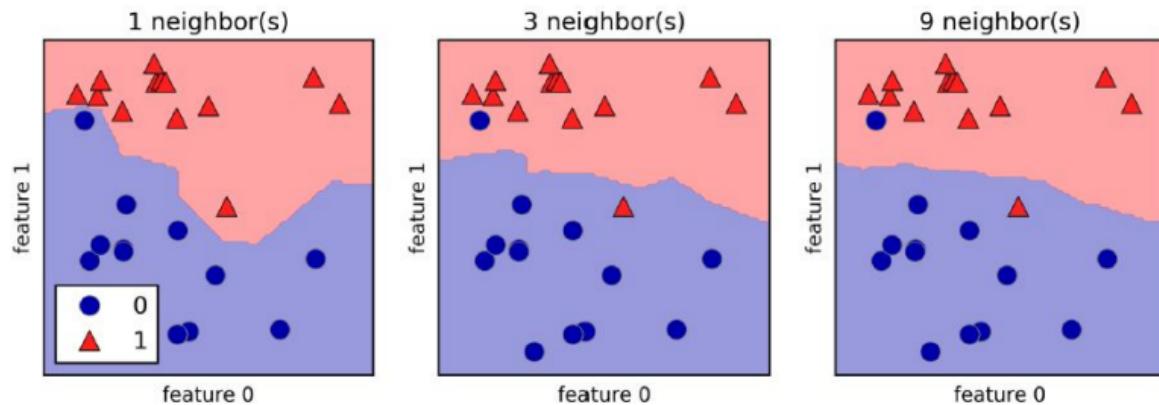
1NN and 3NN on the forge dataset

- The previous two slides show the way the labels of test samples (indicated by stars) are computed, in the case of 1 Nearest Neighbour (1NN) and 3 Nearest Neighbours (3NN).
- The following slide shows the **decision boundary** for 1NN, 3NN, and 9NN and the **forge** dataset giving the prediction for all possible test points in the plane. We colour the plane according to the class that would be assigned to a point in this region. This lets us view the decision boundary, which is the divide between where the algorithm assigns class 0 versus where it assigns class 1.
- The slide after that shows the accuracy as function of the number K of nearest neighbours.

if K is big, complexity is small

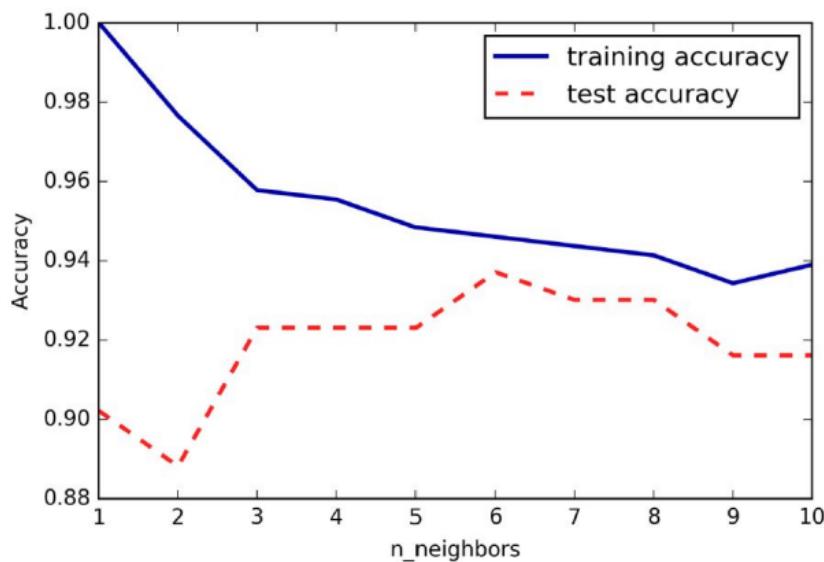
Complexity is lower when it looks more regular

Classification on forge: decision boundary



- A counterintuitive idea:
 - The lower neighbors gives a more complex line
 - 1NN is a complex model
 - 9NN is less complex, fairly simple

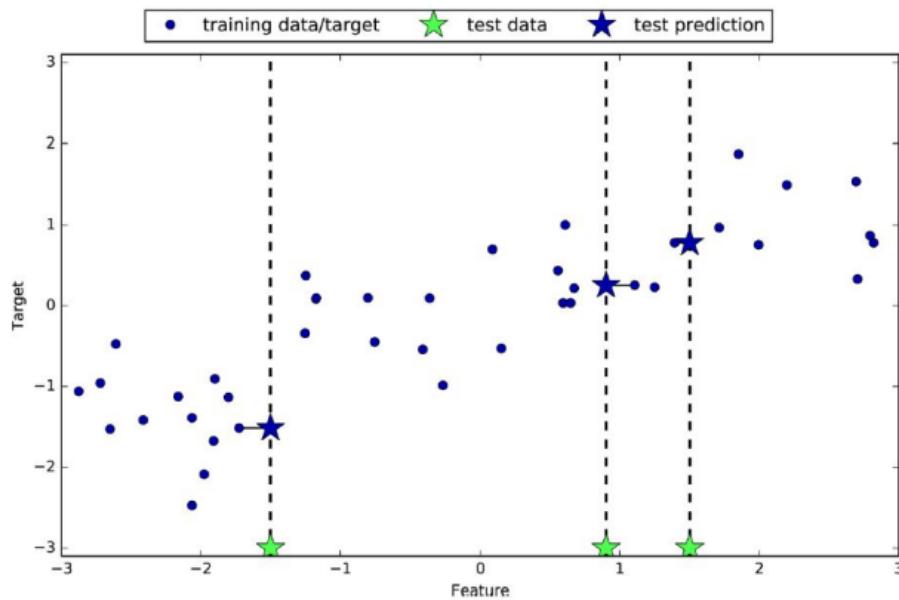
Classification on forge: accuracy as function of K



test accuracy is inverted U, almost

- Watch 24min vid from week4 to get better understanding
- We also have the graph from right to left, the above graph was left to right where the blue line started from small and then went big, this is because more neighbors is less complex (counterintuitive thought from earlier)

1NN regression on forge (3 test samples)



- Now we only measure horizontal distance for NN, as we are only in 1D.
 - Each point, is (feature, label) (x, y)
- Test data is a star, starts with unknown label
 - To find its prediction, find nearest sample in its training set.
 - Horizontal line is distance and prediction its height.

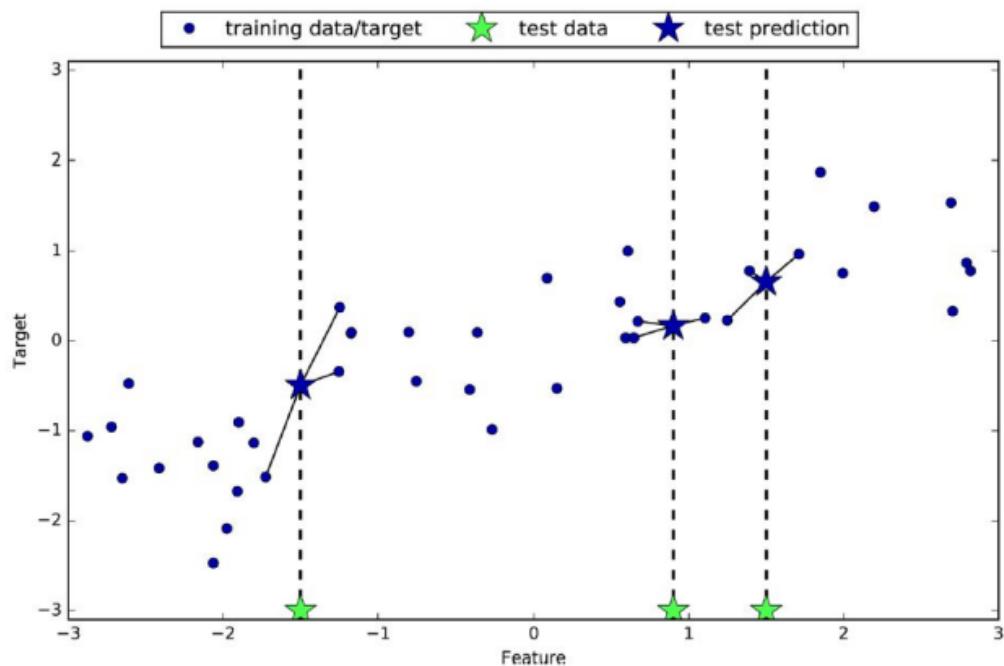
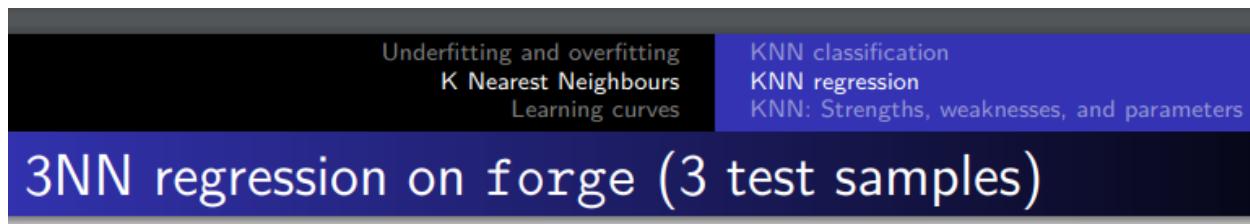
to find nearest neighbor:

find nearest to unlabeled sample, only care about horizontal distance

give same height as labeled nearest neighbour

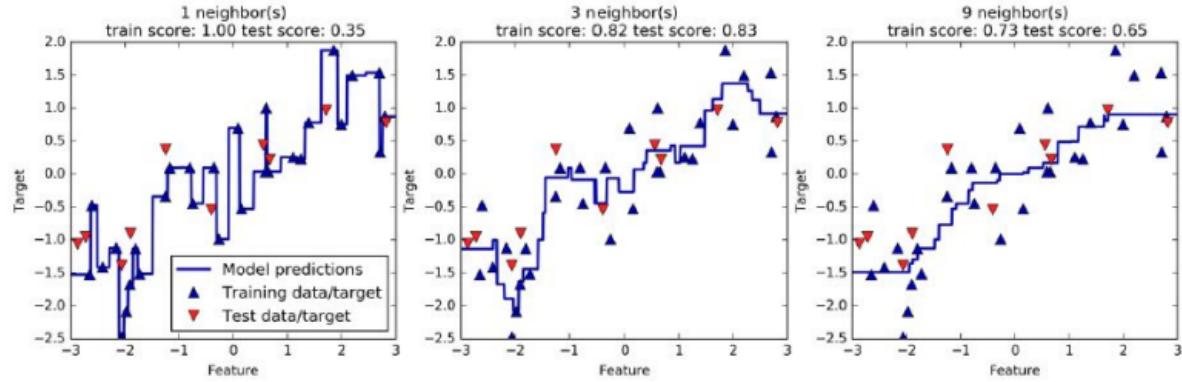
for 1 dimensional feature just use horizontal distance ... not Euclidian.

For 3NN



- Find 3 nearest points to horizontal line
 - **Prediction is the avg of the 3 points.**

Regression on forge: what it looks like



In scikit-learn, score means R^2 in the case of regression (to be discussed in the next chapter, but 1 is the highest value and means “ideal”).

- The left model is Complex (1NN)
- The right is simple (9NN)
- To calculate the **model predictions blue line**
 - we consider every point (on x axis) in turn and find the NN
 - **NOTICE about OVERFITTING**
 - We see the model prediction passes through all training points, BUT we don't want this.
 - We want to regularise

Terminology

- You have seen the function `KNeighborsClassifier` in `scikit-learn`; it's a KNN model for classification.
- Its analogue for regression is `KNeighborsRegressor`.
- In general, models designed for classification problems are referred to as **classifiers** and those designed for regression problems are referred to as **regressors**.
- `scikit-learn` also refers to models as **estimators**.

Parameters

- In principle, there are two important parameters to the K Nearest Neighbours method (both for classification and regression): the number of neighbours and how you measure distance between data points (samples).
- In practice, using a small number of neighbours like three or five often works well, but you should certainly adjust this parameter.
- Choosing the right distance measure is difficult. By default, Euclidean distance is used. which works well in many settings.

Parameters for KNN:

- K - number of neighbors
- d - distance measure

Strengths and weaknesses

- ++ One of the strengths of KNN is that the model is easy to understand, and often gives reasonable performance without a lot of adjustments. It's a good baseline method to try before considering more advanced techniques.
- Building the Nearest Neighbours model is usually very fast, but when your training set is very large (either in number of features or in number of samples) prediction can be slow.
- When using the KNN algorithm, it's important to preprocess your data (see Chapter 6 of this module).
- This approach often does not perform well on datasets with many features (hundreds or more), and it does particularly badly with datasets where most features are 0 most of the time (**sparse** datasets).

advantages/disadvantages

- For preprocessing:
 - if we have two features like petal lenght and petal width
 - Both should be measured in cm or both in inches etc.
- Curse of dimensionality, this is often the case where we have hundreds of features.

Last fundamental point of ML : Learning Curves

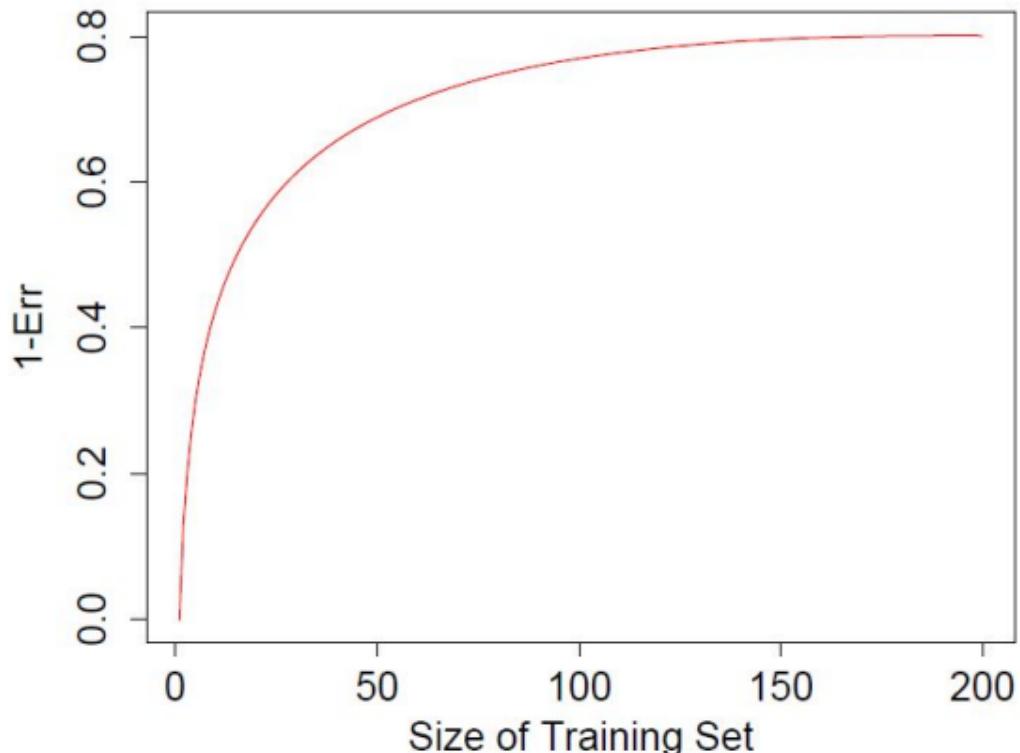
Underfitting and overfitting
K Nearest Neighbours
Learning curves

Learning curves
Cross-validation
Cross-validation and conformal prediction

Plan

- 1 Underfitting and overfitting
- 2 K Nearest Neighbours
- 3 Learning curves

Hypothetical learning curve (accuracy vs n)



- as training set is larger accuracy becomes better.
 - there is a saturation point.
 - sometimes large training sets can destroy algorithm
- We will use the idea of learning curves in the process of **cross-validation**

Learning curves

- Another fundamental notion in machine learning is that of learning curves.
- Previous slide: a hypothetical **learning curve** for a classifier on a given task, which is a plot of the accuracy ($1 - \text{Err}$, where Err is the error rate) versus the size of the training set n . Imagine the accuracy is measured on a large (or even infinite) test set.
- The performance of the classifier improves as the training set size increases to 100 samples; increasing the number further to 200 brings only a small benefit.
- A typical use of learning curves is in understanding cross-validation.

Cross-validation (1)

- Cross-validation is a method of evaluating generalization performance.
- In cross-validation, the data is split randomly and multiple models are trained.
- In *K-fold cross-validation*, where K is a user-specified number (usually 5 or 10), the data is first partitioned into K parts of (approximately) equal size, called *folds*.
- Next, a sequence of models is trained.

given a training set: want to estimate accuracy

No test set, etc

So we need to model test set inside training set

primitive way is to split training data into two parts, smaller training set and a test set

Division is usually random

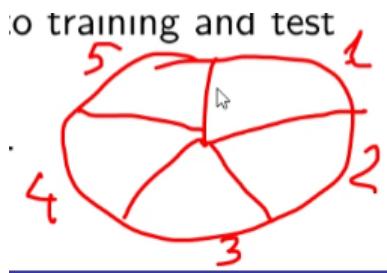
Cross validation is smarter...

shown below

Cross-validation (2)

Say, in 5-fold cross-validation:

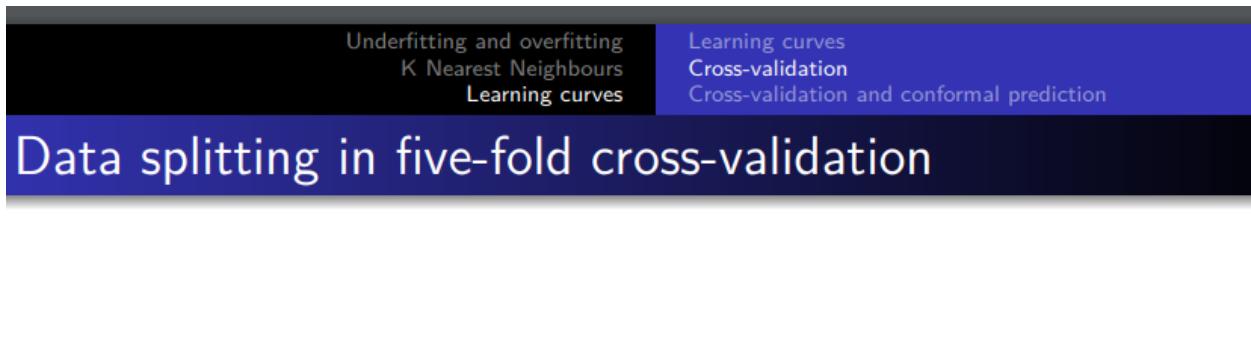
- The first model is trained using the first fold as the test set, and the remaining folds (2–5) are used as the training set.
- The model is built using the data in folds 2–5, and then the accuracy is evaluated on fold 1.
- Then another model is built, this time using fold 2 as the test set and the data in folds 1, 3, 4, and 5 as the training set.
- This process is repeated using folds 3, 4, and 5 as test sets.
- For each of these five splits of the data into training and test sets, we compute the accuracy.
- The process is illustrated on the next slide.



- split into 5 sets of data:
 - Train data on 4 folds, then:
 - take last slot of data and compute accuracy on it (as test set)
- do the same for second slot of data, and for the rest ...
 - so that each slot is at one point a test set
 - Each iteration is one fold, as we see in next slide

We get a better representation on the Next Slide

Then we get 5 numbers, so the avg is one number estimate using cross validation



- In the end, we have collected five accuracy values.
- To obtain an overall estimate of generalization performance, average the five accuracy values.

Cross-validation in scikit-learn

This is how it's done in scikit-learn:

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
iris = load_iris()
knn = KNeighborsClassifier(n_neighbors=1)
cross_val_score(knn, iris.data, iris.target)

array([0.967, 0.967, 0.933, 0.933, 1.    ])
```

By default, scikit-learn performs 5-fold cross-validation, but you can use, say,

```
cross_val_score(knn, iris.data, iris.target, cv=10)
```

for 10-fold cross-validation.

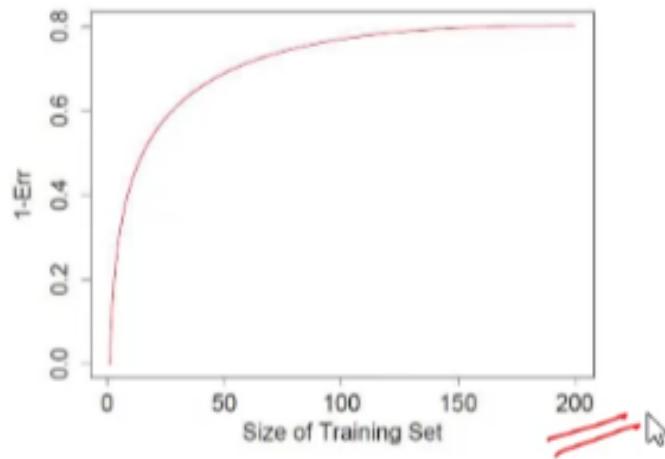
- We can see 5 cross val scores, so by default it is 5 folds of data.

Cross-validation and learning curves (1)

- Remember the learning curve on slide 24.
 - Suppose we are given a training set, train a classifier on it, and estimate its generalization performance using 5-fold cross-validation on this training set (this is all data we have!).
 - How accurate is our estimate?
-
- Suppose learning curve is like the one on slide 24 (earlier)
 - Do we have any bias?
 - It tends to be downwards

Lets recall:

Hypothetical learning curve (accuracy vs n)



- All these points tie to the next 2 slides and the one above!
- Doing 5 fold Cross Validation
 - 4/5 is our training set (160) if training set == 200
 - 1/5 is our test set (40)
 - So therefore our accuracy will be a bit smaller, at size =160
 - This is fine in the case above
 - **BUT** what if our training set is only of size 50 samples
 - Normally is good accuracy
 - BUT using CV we get sample size of 40,
 - WITH a significantly lower accuracy

SO we like to use CV when we are at the right side of the learning curve.

Cross-validation and learning curves (2)

- If our training set has 200 labelled samples, 5-fold cross-validation estimates the performance of our classifier over training sets of size 160, which is virtually the same as the performance for training set size 200.
- Thus cross-validation does not suffer from much bias.
- However if the training set has 50 labelled samples, 5-fold cross-validation estimates the performance of our classifier over training sets of size 40, and from the figure that is an underestimate of $1 - \text{Err}$.
- Hence, as an estimate of $1 - \text{Err}$, cross-validation is biased downwards.

Bias vs variance

- To summarize the previous slide: if the learning curve has a considerable slope at the given training set size, K -fold cross-validation will overestimate the true prediction error.
- The extreme version of cross-validation is where $K = n$ (each sample is in its own fold); this is known as **leave-one-out cross-validation**.
- For leave-one-out cross-validation the bias is typically negligible, but now **variance** raises its head: you are likely to get a very different result on a different dataset of the same size.
- Overall, 5- or 10-fold cross-validation is usually recommended as a good compromise.
- Extreme case of CV:
 - Where $K = n$ (sample size)
 - So each sample has its own fold
 - We take out each sample in turn, train on remaining samples and then test the classifier on that remaining sample and record whether we make an error or not
 - The final avg of CV generalisation accuracy is the percentage of correct predictions.
 - This is called leave-on-out-C-V
 - The variance will be significant,

- if we take another set of data taken from the same probability distribution we are likely to get very different estimates of accuracy

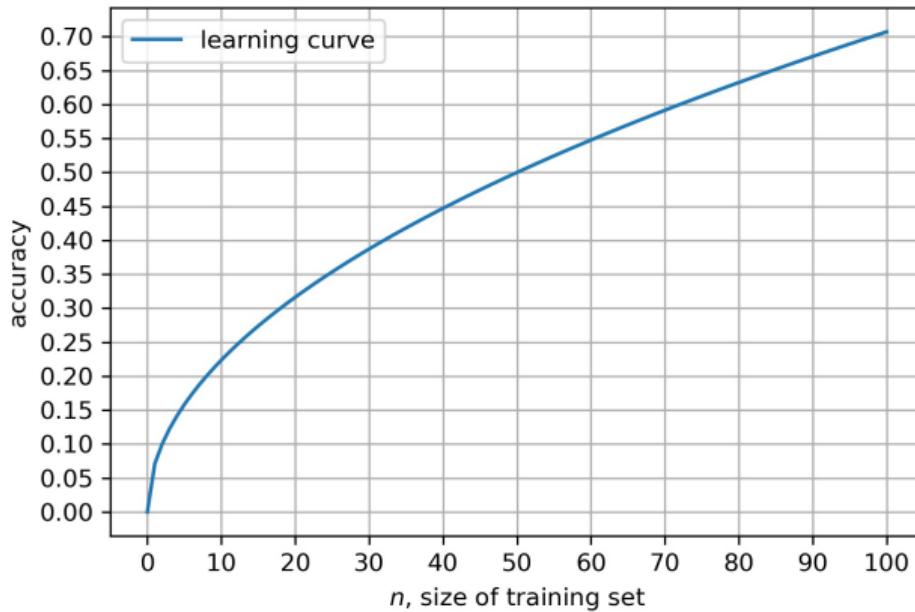
We need to find a balance between variance and bias

Underfitting and overfitting
K Nearest Neighbours
Learning curves

Learning curves
Cross-validation
Cross-validation and conformal prediction

Exercise

Estimate the downward bias of 5-fold cross-validation for a training set of size $n = 100$ and this learning curve:



- With 100 samples:
 - accuracy = 71%
 - Lets do 5 fold CV:
 - Each fold is size 20

- 80 size training set
 - Accuracy = 63%
- Now lets measure our downward bias:
- $0.71 - 0.63 = 0.08$

Underfitting and overfitting
K Nearest Neighbours
Learning curves

Learning curves
Cross-validation
Cross-validation and conformal prediction

Solution

- The accuracy for the full training set is about 0.71.
- The accuracy for training sets of size $\frac{K-1}{K}n = 80$ is about 0.63.
- Therefore, the downward bias is about $0.71 - 0.63 = 0.08$.

Values from graph above

Cross-validation (CV) and conformal prediction (CP)

- CV and CP have similar goals.
- They answer the question “How confident can we be in our prediction?”
- These are some differences:
 - CV evaluates a prediction algorithm *en masse*, whereas CP tries to say something about a given test sample.
 - Unlike CP, CV does not have any simple properties of validity.
 - CV might be easier to interpret; it is still the standard method.

Cross validation vs Conformal Prediction.

- For conformal prediction
 - Might be better in the case when you are the test sample and a doctor is trying to evaluate you. CP is better for specific test sample
- CV is better for evaluating a general algorithm

New Chapter

Chapter 5: Linear regression

Volodymyr Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS5920 Machine Learning

Last edited: October 21, 2022

Simple linear regression

- Suppose we have only one feature ($p = 1$): each sample is a real number x .
- The linear regression model is

$$\hat{y} = wx + b,$$

where \hat{y} is the prediction for the label, and w (the **slope**) and b (the **intercept**) are parameters.

- w is the effect on \hat{y} of a one unit increase in x .
- p is number of features
 - So each sample is just one number (1D example)
- y_{hat} (prediction for sample x) = $wx + b$

Multiple linear regression

- If we have p features, the linear regression model is

$$\begin{aligned}\hat{y} &= w \cdot x + b \\ &= w[0]x[0] + w[1]x[1] + \cdots + w[p-1]x[p-1] + b,\end{aligned}$$

where $x[j]$ is the $(j+1)$ st feature.

- $w[j]$ is interpreted as the effect on \hat{y} of a one unit increase in $x[j]$, holding all other features fixed.
- We will discuss three ways of estimating the coefficients $w[j]$: Least Squares, Ridge Regression, and Lasso.
- Once the parameters are estimated, we can use them for prediction.

• ^^^^^

- Now these parameters w and b are estimated with:

- Least Squares
- Ridge Regression
- Lasso

1. Once you make these estimates of coefficients:

- a. You take test sample x^*
 - i. You take all its features and plug them into the equation
 - ii. Then you get prediction of the test sample.

Linear regression vs Nearest Neighbours

- Linear regression is an example of a **parametric** approach: it assumes a linear functional form for the prediction $\hat{y} = f(x)$. (For example, f is determined by two parameters in the case of simple linear regression.)
- Nearest Neighbours regression: does not assume a parametric form for $\hat{y} = f(x)$, and so is a **non-parametric** method.

Opposite ends of spectrum from ML:

- Linear Regression:
 - Parametric approach:
 - We assume prediction $y_{\text{hat}} = f(x)$
- Nearest Neighbor:
 - Non parametric approach

Parametric methods: advantages and disadvantages

Advantages:

- They are often easy to fit, because we need to estimate only a small number of parameters.
- In many cases (including linear regression), the coefficients have simple interpretations and can be easily estimated.

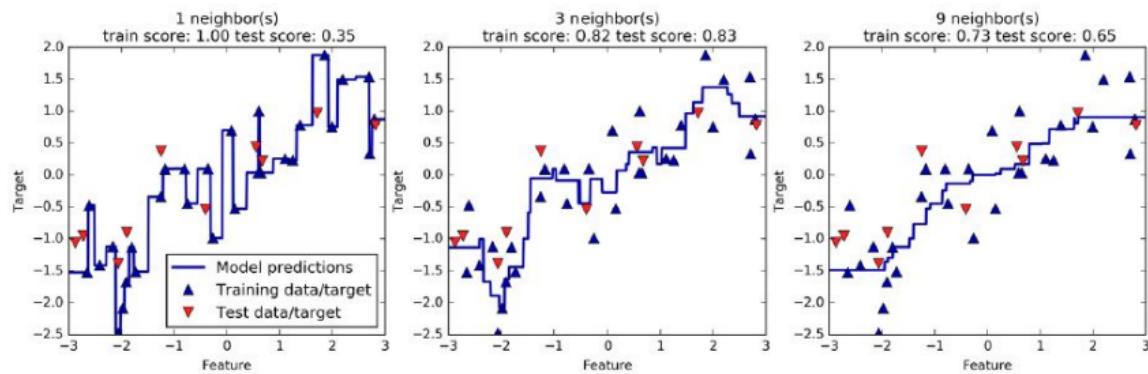
Disadvantage:

- They make strong assumptions about the form of $\hat{y} = f(x)$. If the specified functional form is far from the truth, the parametric method will perform poorly.

Non-parametric methods

- Non-parametric methods provide an alternative and more flexible approach.
- As we can see on the following slide (from Chapter 4):
 - The flexibility of K Nearest Neighbours diminishes as K grows.
 - The 1 Nearest Neighbour algorithm is extremely flexible and overfits.

Regression on forge (Chapter 4)



here we dont assume a linear function would fit (nearest neighbor approach)

The two extremes

- The 1 Nearest Neighbour method is the most flexible method studied in machine learning.
 - Often overfits.
 - We are often interested in non-parametric methods that are less flexible.
- Linear regression is the most inflexible method studied in machine learning.
 - Often underfits.
 - We are often interested in parametric methods that are more flexible.

Linear regression (simple one) so inflexible, literally can't move the line

Estimating the regression parameters by Least Squares

This is a classical approach to regression (going back to Gauss and Legendre, 1805!).

- The parameters are estimated using the **Least Squares** approach: we choose w and b to minimize

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad \text{where } \hat{y}_i = w \cdot x_i + b.$$

- RSS stands for **residual sum of squares** (and $y_i - \hat{y}_i$ are sometimes referred to as **residuals**).
- The optimal w and b are usually denoted \hat{w} and \hat{b} , respectively. The components of \hat{w} are $\hat{w}[0], \hat{w}[1], \dots, \hat{w}[p-1]$.

- Try all W and B
 - Using these we can compute Residual Sum of Squares (RSS)
 - Now using (**yi - yi_hat**)
 - for each label **yi** in training set
 - **yi_hat** is prediction using fixed **w** and **b**
 - we are basically finding how bad prediction is, how wrong we get it.
 - square it to get rid of sign and then sum it
1. Do this for all **w** and **b**:
 - a. and find the best
 - b. those for which RSS is as small as possible

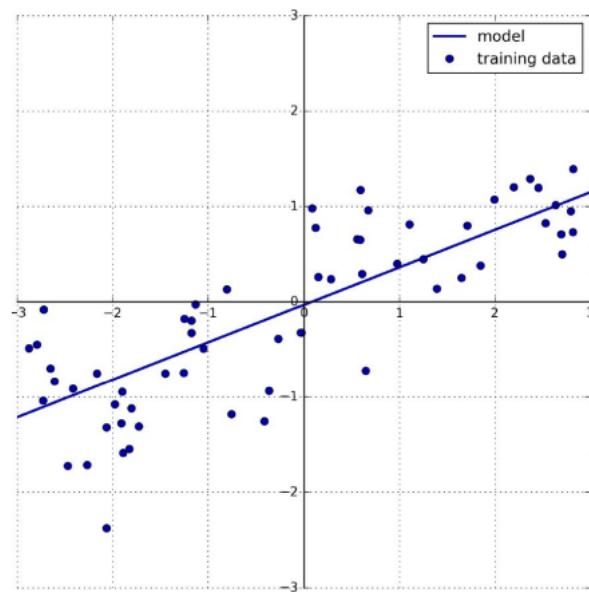
optimal w and b are $w_{\hat{}}$, $b_{\hat{}}$

- RSS $\longrightarrow \min$

Linear regression
Ridge Regression
Lasso
Discussion

Introduction
Optimization problem for Least Squares
Feature engineering
Measuring the quality of a linear model

Predictions on wave



- This is our model, RSS is minimum here.
- Lose lots of structure in data
- Maybe better line is a parabola?
- How do we get something better?
 - Lets use **feature engineering!**

Following are old notes: skip to:

”””

SKIP TO HERE

”””

The screenshot shows a navigation bar for a machine learning course. On the left, there is a black sidebar with the text "Linear regression", "Ridge Regression", "Lasso", and "Discussion". To the right of this is a blue sidebar with the text "Introduction", "Optimization problem for Least Squares", "Feature engineering", and "Measuring the quality of a linear model". Below these bars, the main content area has a dark blue header with the text "Multiple linear regression".

- If we have p features, the linear regression model is

$$\begin{aligned}\hat{y} &= w \cdot x + b \\ &= w[0]x[0] + w[1]x[1] + \cdots + w[p-1]x[p-1] + b,\end{aligned}$$

where $x[j]$ is the $(j+1)$ st feature.

- $w[j]$ is interpreted as the effect on \hat{y} of a one unit increase in $x[j]$, holding all other features fixed.
- We will discuss three ways of estimating the coefficients $w[j]$: Least Squares, Ridge Regression, and Lasso.
- Once the parameters are estimated, we can use them for prediction.

when $p > 1$ above.

so now we have scalar product instead of multiplication

$x[0]$ is the first feature

We need to estimate W and B

to estimate W is **least squares**, **ridge regression** and **lasso**

After estimating coefficients:

So we take x^* (test sample), take its features and plug it into place of $x[0]$, $x[1]$, $x[2]$

.....

The screenshot shows a dark-themed user interface for a machine learning course. At the top, there is a horizontal navigation bar with several items: 'Linear regression' (in white), 'Ridge Regression', 'Lasso', and 'Discussion' (all in light gray). To the right of this bar is a blue sidebar containing the following text: 'Introduction', 'Optimization problem for Least Squares', 'Feature engineering', and 'Measuring the quality of a linear model'. Below the navigation bar, the main title 'Linear regression vs Nearest Neighbours' is displayed in large, white, sans-serif font.

- Linear regression is an example of a **parametric** approach: it assumes a linear functional form for the prediction $\hat{y} = f(x)$. (For example, f is determined by two parameters in the case of simple linear regression.)
- Nearest Neighbours regression: does not assume a parametric form for $\hat{y} = f(x)$, and so is a **non-parametric** method.

They are opposite ends of machine learning

LinearRegression is parametric

Parametric methods: advantages and disadvantages

Advantages:

- They are often easy to fit, because we need to estimate only a small number of parameters.
- In many cases (including linear regression), the coefficients have simple interpretations and can be easily estimated.

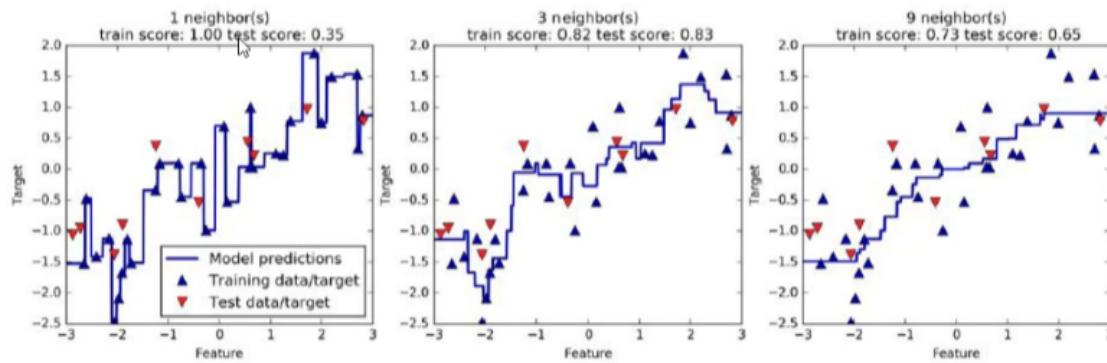
Disadvantage:

- They make strong assumptions about the form of $\hat{y} = f(x)$. If the specified functional form is far from the truth, the parametric method will perform poorly.

Non-parametric methods

- Non-parametric methods provide an alternative and more flexible approach.
- As we can see on the following slide (from Chapter 4):
 - The flexibility of K Nearest Neighbours diminishes as K grows.
 - The 1 Nearest Neighbour algorithm is extremely flexible and overfits.

Regression on forge (Chapter 4)



we can see how the blue line is clearly jagged and overfitted for 1NN

But for 9 is more regular

The two extremes

- The 1 Nearest Neighbour method is the most flexible method studied in machine learning.
 - Often overfits.
 - We are often interested in non-parametric methods that are less flexible.
- Linear regression is the most inflexible method studied in machine learning.
 - Often underfits.
 - We are often interested in parametric methods that are more flexible.

How do we estimate parameters?

Estimating the regression parameters by Least Squares

This is a classical approach to regression (going back to Gauss and Legendre, 1805!).

- The parameters are estimated using the **Least Squares** approach: we choose w and b to minimize

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad \text{where } \hat{y}_i = w \cdot x_i + b.$$

- RSS stands for **residual sum of squares** (and $y_i - \hat{y}_i$ are sometimes referred to as **residuals**).
- The optimal w and b are usually denoted \hat{w} and \hat{b} , respectively. The components of \hat{w} are $\hat{w}[0], \hat{w}[1], \dots, \hat{w}[p-1]$.

Least squares.

estimate for w and b is “learnt” from training set

RSS:

Try all possible W and B

For fixed values of W and B , vector and scalar respectively

we can compute RSS

RSS is defined as:

For each label y_i , in training set:

find residuals $(y_i - y_i_\hat{})$

where $y_i_\hat{}$ is above with fixed W and B

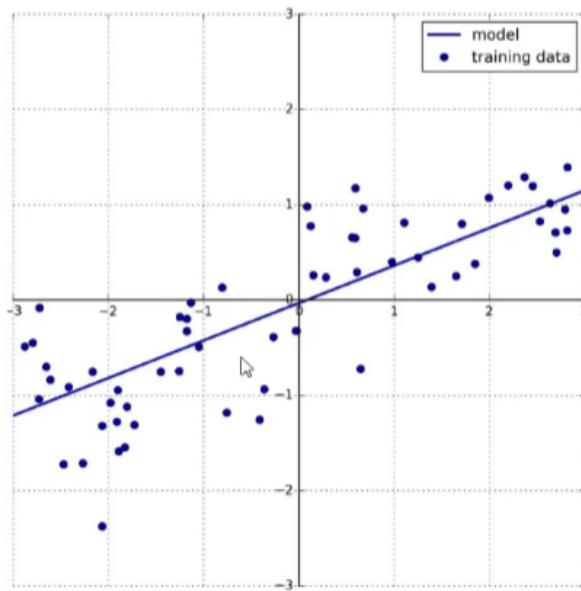
Now do this for ALL W and B and choose best W and B ($w_{\hat{}}$ and $b_{\hat{}}$)

residuals is just how wrong you are, smaller is better

Linear regression
Ridge Regression
Lasso
Discussion

Introduction
Optimization problem for Least Squares
Feature engineering
Measuring the quality of a linear model

Predictions on wave



Example

quite a crude approximation, lets do something better

SKIP TO HERE

The screenshot shows a Jupyter Notebook interface. At the top, there's a dark header bar with the text "SKIP TO HERE". Below it is a light gray header bar containing the title "Feature engineering (1)". Underneath these, there's a dark blue sidebar on the left with the following menu items:

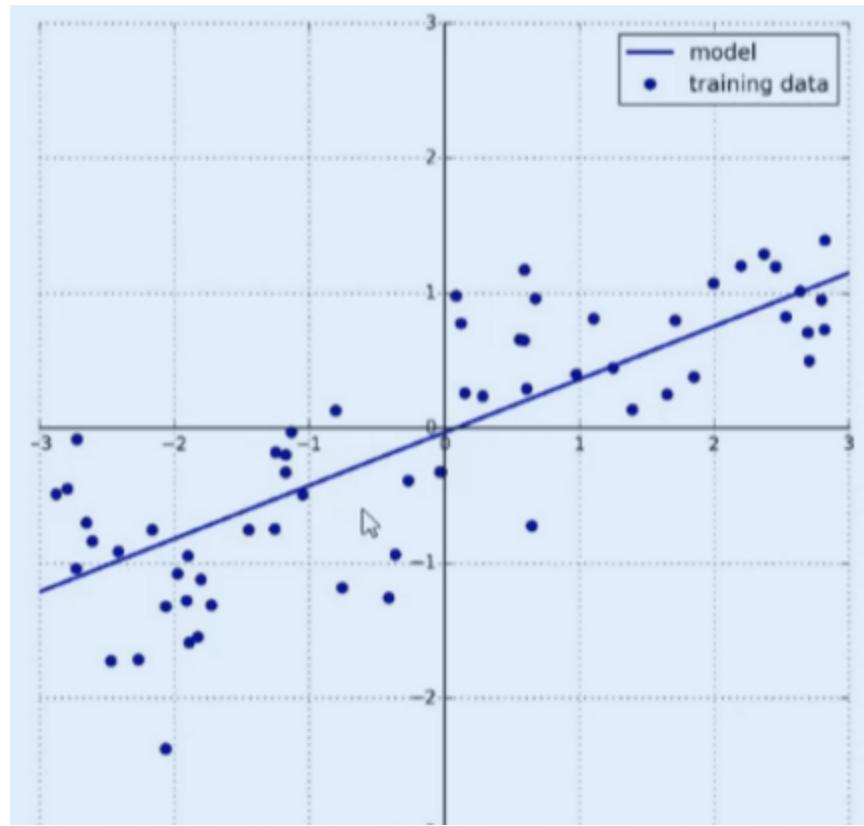
- Linear regression
- Ridge Regression
- Lasso
- Discussion

To the right of the sidebar, the main content area has a dark blue header with the following text:

- Introduction
- Optimization problem for Least Squares
- Feature engineering**
- Measuring the quality of a linear model

A red arrow points from the "Feature engineering" section in the sidebar to the corresponding section in the main content area.

- If you compare the predictions made by the straight line with those made by the K Nearest Neighbors, using a straight line to make predictions seems very restrictive; all the fine details of the data are lost.
- But for datasets with many features, linear models can be very powerful!
- And we can add features to our training set at will. You will see (or have seen) examples in Lab 5 (for the Boston Housing and wave datasets).
- Including derived features is called **feature engineering**.



- In above example:
 - We had just one feature x , and label on y axis.
 - But we can make more features, how do we do this?

This will maybe give us a parabola like below:

Feature engineering (2)

- Suppose that you believe that a parabola $\hat{y} = \alpha x^2 + \beta x + b$ would be a better fit for your data than a straight line (e.g., the wave dataset looks curving down if you go left to right).
- To fit a parabola, you add another feature to your data set, x^2 .
- Fitting a linear function in the new features gives you $\hat{y} = w[0]x^2 + w[1]x + b$, a parabola!
- You can try it in Lab 5.

We have added the feature xSqrld:



- Straight line wouldn't fit well for above data, so let's add feature x^2

x	x^2	y
0	0	0
1	1	2
2	4	5

- We have a bigger dataset now

- So now fit a linear function to new dataset
 - and we get the above equation. Parabola (y_{hat})

How to measure quality of a linear model, R^2

R^2 (1)

- Remember the residual sum of squares:

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

- The no-data analogue is the **total sum of squares**

$$\text{TSS} := \sum_{i=1}^n (y_i - \bar{y})^2,$$

where \bar{y} is the average label,

$$\bar{y} := \frac{1}{n} \sum_{i=1}^n y_i.$$

- R^2 between 0 and 1, it shows how well model performs to training set.

RSS we just look at residuals

TSS - total sum of squares.

average all labels.

We aren't using x (data) just y (labels), no samples.

Instead of predicted labels using the samples, we use \bar{y}

The screenshot shows a Jupyter Notebook interface. The title of the cell is "R² (2)". The top navigation bar has tabs for "Linear regression", "Ridge Regression", "Lasso", and "Discussion". The right side of the navigation bar is blue and contains links for "Introduction", "Optimization problem for Least Squares", "Feature engineering", and "Measuring the quality of a linear model".

- R^2 : the percentage of variability in the label that is explained by the data,
$$R^2 = \frac{TSS - RSS}{TSS} \in [0, 1].$$
- R^2 measures the performance on the training set only. A good (i.e., close to 1) R^2 is compatible with poor performance on new data (because of the possibility of overfitting).
- R^2 is the variability in the label.
 - It is the ratio
 - Numerator: Shows how using our data, improves our performance
 - From using averages (for prediction) to using our actual predictions
 - 0 means RSS is almost TSS so R^2 0 is bad, no variation between the average and what we predict. we will be underfitting
 - **BUT ALSO CLOSE TO 1:**

- means you might not predict well
- because on new data performance might be bad, lots of overfitting

This is why we have the notion of Test R^2

Linear regression
Ridge Regression
Lasso
Discussion
Introduction
Optimization problem for Least Squares
Feature engineering
Measuring the quality of a linear model

Test R^2 (1)

- scikit-learn uses the notion of test R^2 .
- Given a training set, find the estimates $\hat{w}[0], \hat{w}[1], \dots, \hat{w}[p - 1], \hat{b}$ of the parameters.
- For each test sample x_i^* (with true label y_i^* , not used at this step) compute the prediction

$$\begin{aligned}\hat{y}_i^* &= \hat{w} \cdot x_i^* + \hat{b} \\ &= \hat{w}[0]x_i^*[0] + \hat{w}[1]x_i^*[1] + \dots + \hat{w}[p - 1]x_i^*[p - 1] + \hat{b}.\end{aligned}$$

- Compute the test RSS

$$\text{RSS} = \sum_{i=1}^m (y_i^* - \hat{y}_i^*)^2,$$

where m is the size of the test set.

- w_hat and b_hat are our predictions
- Given a test sample (x^*, y^*)
 - For each x^*
 - Compute prediction $y_{\text{hat}}^* = w_{\text{hat}} * x^* + b_{\text{hat}}$

- Compute test RSS
 - comparing y_i^* with $y_i^*_{\text{prediction}}$

This is test RSS because we are doing it for the test set

To define Test R^2

Test R^2 (2)

- Compute the **test TSS**

$$\text{TSS} := \sum_{i=1}^m (y_i^* - \bar{y}^*)^2,$$

where \bar{y}^* is the average label in the test set,

$$\bar{y}^* := \frac{1}{m} \sum_{i=1}^m y_i^*.$$

- Compute the **test R^2** using the same formula,

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} \in (-\infty, 1].$$

-
- We need some analoge of the total
 - Doing this all for the test set y^*

- It is unusual here that it can take negative values (-infinity) :
 - This is because TSS has some element of data snooping/overfitting
 - From ChatGPT, Vovk explanation in next slide
 -

One way that computing the R-squared value can lead to data snooping is if the data used to compute the value is not independent from the data used to train the model. For example, if the same dataset is used both to train the model and to compute the R-squared value, this can lead to an overly optimistic estimate of the model's performance, as the model has essentially been trained to fit the specific patterns and relationships present in that dataset.

To avoid data snooping when computing the R-squared value, it is important to use a separate, independent dataset for this purpose, and to ensure that the data used to compute the value is not used in any way to train the model. Additionally, techniques such as cross-validation can be used to further ensure that the model is able to generalize well to new data.

Test R^2 (3)

- Notice that now R^2 can be negative!
- And notice that the test TSS has an element of data snooping in it; a slightly better definition would use \bar{y} (the average label of the training set) instead of \bar{y}^* . (But the difference is usually tiny.)
- The same definition of test (and training) R^2 can be used for any linear model (such as Ridge Regression or Lasso discussed later in this chapter),
- and even for any regression model (except that \hat{y}_i^* will be defined differently).

For TSS there is data snooping because to compute the neutral label (y_{hat}^*) we are using our test labels, which we are not supposed to do.

avg test label (y_{hat}^*) shouldnt be available to us

- Compute the **test TSS**

$$\text{TSS} := \sum_{i=1}^m (y_i^* - \bar{y}^*)^2,$$

So sometimes it is better to use y_{hat} (avg label) instead of y_{hat}^* (avg test label)

Exercise

- Consider the model $\hat{y} = 2x + 3$ (found from a training set using simple linear regression).
- What is the test R^2 of this model on the following test set?

$$(x_1^*, y_1^*) = (0, 1)$$
$$(x_2^*, y_2^*) = (2, 5).$$

(Answer: 0.)

- Give an example of a test set for which $R^2 < 0$.

suppose given training set:

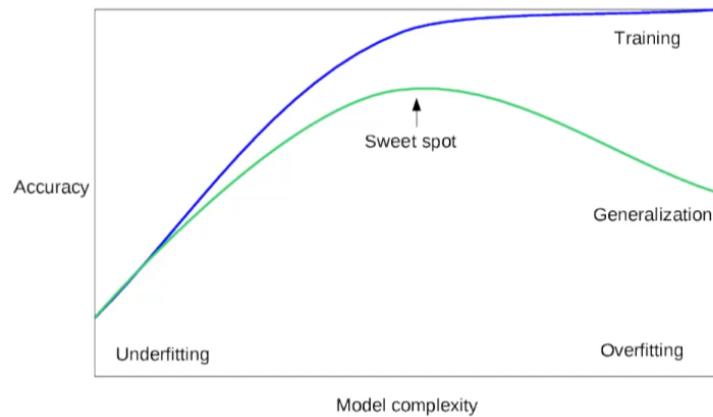
- We find a model, simple linear regression with feature x
 - suppose model is $y_{\text{hat}} = 2x + 3$
 - we can use this to predict test labels.
 - test set: x_1^*, y_1^* AND x_2^*, y_2^*

Ridge Regression

Regularization

- In Ridge Regression the coefficients (w) are chosen not only so that they predict well on the training data, but to make their magnitude as small as possible.
 - Intuitively, this means each feature should have as little effect on the outcome as possible (which translates to having a small slope), while still predicting well.
 - This is an example of **regularization**, a standard way to avoid overfitting.
 - Adding regularization means moving left in the figure of Chapter 4 (reproduced on the next slide).
-
- We are trying to strike the right balance.
 - Coefficients to:
 - Do a good job of prediction
 - And Coefficients to be as small as possible

So basically:



We are trying to move left

It would seem good idea to climb on as high as possible, but that's not optimal, so we are trying to move left. in the hope of meeting the sweetspot.

Ridge Regression (1)

- The Least Squares procedure estimates w and b by minimizing

$$\text{RSS} = \sum_{i=1}^n (y_i - w \cdot x_i - b)^2.$$

- Ridge Regression is similar, except that the parameters are estimated by minimizing

$$\text{RSS} + \alpha \|w\|^2 = \text{RSS} + \alpha \sum_{j=0}^{p-1} w[j]^2,$$

where $\alpha \geq 0$ is the regularization parameter (or tuning parameter).

- We are trading off two criteria:
 - fitting the data well
 - moving the parameter estimates towards zero

This is what it looks like to minimise mathematically.

This is For Least Square

- We are trying to choose our parameters w and b :
 - To minimise RSS

$$\text{RSS} = \sum_{i=1}^n (y_i - w \cdot x_i - b)^2. \rightarrow \min$$

$y_i - w \cdot x_i - b$

$$y_i \hat{=} (w * x_i + b)$$

- We just substitute in above to $(y_i - y_i \hat{)}$

In **RidgeRegression**, instead of just minimising RSS we are trying to improve performance on training set by minimising this combination:

$$\text{RSS} + \alpha \|w\|^2 :$$

So definition of $\|w\|$ is just sum of w array squared

$$\text{RSS} + \alpha \|w\|^2 = \text{RSS} + \alpha \sum_{j=0}^{p-1} w[j]^2,$$

alpha is a tuning parameter. j goes from 0 to p-1 where p is number of features. w is coefficient of the features.

$$+ \alpha \sum_{j=0}^{p-1} w[j]^2, \quad LS$$

$\alpha = \infty : w = 0$

- The above semi-circled defines how big we want w to be, if alpha is infinite, w goes to 0

REASON FOR $\alpha = \infty, w = 0$

- We are minimising RSS, so obviously if alpha is big, we need to minimise RSS by making w smaller

Ridge Regression (2)

- When $\alpha = 0$: Ridge Regression coincides with Least Squares.
- As $\alpha \rightarrow \infty$, the Ridge Regression coefficient estimates approach zero.
- Selecting a good value for α is critical.
- Notice: the **shrinkage penalty** $\alpha \sum_{j=0}^{p-1} w[j]^2$ is not applied to the intercept b !
 - we want to shrink the estimated association of each features with the label
 - we do not want to shrink the estimated intercept

Cross Validation to choose alfa

Sometimes called L2 Regularisation

Terminology

- A fancier name for the Euclidean norm

$$\sqrt{\sum_{j=0}^{p-1} w[j]^2}$$

is the *L₂ norm*.

- In which case $\|w\|$ may be written as $\|w\|_2$, and we may be said to be using *L₂ regularization*.

Lasso - Modification of RidgeRegression

Lasso (1)

- We can modify Ridge Regression in such a way that it achieves “model selection” as by-product.
- Ridge Regression typically includes all p features in the final model.
- The shrinkage penalty $\alpha \sum_j w[j]^2$ shrinks the coefficients towards zero, but does not set them exactly to zero (unless $\alpha = \infty$).
- May not be a problem for prediction accuracy, but may complicate model interpretation when p is large.
- Lasso will set many coefficients $w[j]$ to zero.

Lasso is for Model Selection

- In Ridge Regression typically, none of the coefficients in front of the features = 0
- But Lasso aims to make some 0.

Lasso (2)

- The **Lasso** coefficients \hat{w} and \hat{b} minimize the quantity

$$\text{RSS} + \alpha \sum_{j=0}^{p-1} |w[j]|.$$

- The predictions are computed as before: the prediction for a test sample x^* is

$$\hat{w} \cdot x^* + \hat{b} = \hat{w}[0]x^*[0] + \cdots + \hat{w}[p-1]x^*[p-1] + \hat{b},$$

where $x^*[j]$ are the features.

- Now we are minimising the top equation again
 - Ridge regression we had to square (that is **the only difference**)

Still same: linear regression method

$$\hat{y} = \hat{w} \cdot x^* + \hat{b} = \hat{w}[0]x^*[0] + \cdots + \hat{w}[p-1]x^*[p-1] + \hat{b},$$

Only difference is the expression we use to find w and b.

Lasso (3)

- The only difference from Ridge Regression is that the L_2 shrinkage penalty $\alpha \sum_j w[j]^2$ has been replaced by the L_1 shrinkage penalty $\alpha \sum_j |w[j]|$.
- The L_1 norm of a coefficient vector $w = (w[0], \dots, w[p-1])$ is defined by

$$\|w\|_1 = \sum_{j=0}^{p-1} |w[j]|.$$

Remark: Lasso = least absolute shrinkage and selection operator.

L1 norm as opposed to L2 norm

Lasso and model selection

- As Ridge Regression, the Lasso shrinks the coefficient estimates towards zero.
- However, using the L_1 shrinkage penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the regularization parameter α is sufficiently large.
- So the Lasso performs model selection (yielding **sparse models**, i.e., models that involve only some of the features).

Sparse models

But why is this the case?

The case $p = 2$ (*)

It helps intuition to consider the case $p = 2$ (2 features):

- An L_2 -ball $\|w\|_2 \leq r$ looks like a 2D ball (= circle) around 0.
- But an L_1 -ball $\|w\|_1 \leq r$ looks like a diamond around 0!
- The “RSS-ball” $\text{RSS}(w) \leq r$ is an ellipse around \hat{w}^{LS} .
- If we draw the L_2 -ball and RSS-ball containing \hat{w}^{RR} on their surfaces, these two balls will touch.
- If we draw the L_1 -ball and RSS-ball containing \hat{w}^L on their surfaces, they will also touch.
- Here \hat{w}^{LS} , \hat{w}^{RR} , and \hat{w}^L mean the vector parameter w selected by Least Squares, Ridge Regression, and Lasso, respectively.

Lets first remind of L1 and L2 norm:

The L1 norm is the sum of the absolute values of the vector.

$$\text{L1 norm} = |x_1| + |x_2| + \dots + |x_n|$$

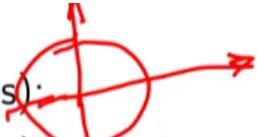
The L2 norm, also known as the Euclidean norm, is the square root of the sum of the squares of the vector. It is defined as:

$$\text{L2 norm} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

L2 ball is the set of all Ws such that L2 norm **doesnt exceed R where r is some constant.**

It helps intuition to consider the case $p = 2$ (2 features):

- An L_2 -ball $\|w\|_2 \leq r$ looks like a 2D ball (= circle) around 0.
- W_2 looks like this ball(circle) because any 2D vector, computing the L2 norm is just like pythagorus:
 - For example: vector (3, 4)
 - This vector has length 5 from the origin: $\sqrt{3^2 + 4^2} = 5$
 - so taking as many points as we can with length 5, we get a circle,
 - **IE this length 5 is the constant R, r.**



but what does W_1 look like?

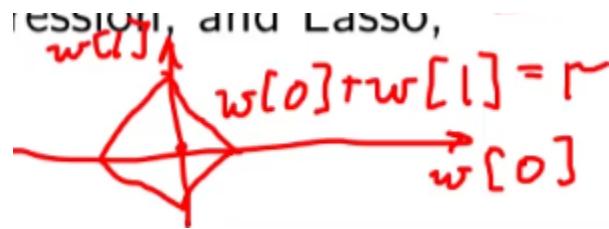
$$\|w\|_1 = r \quad |w[0]| + |w[1]| \leq r$$

- $\|w\|_1 \leq r$
 - Where w can look like: $|w[0]| + |w[1]| \leq r$
 - first componenet in absolute value + second comp in abs val
- The boundary is where $\|w\|_1 = r$



At the 2nd quadrant we have $w[0] = w[1]$

- so its just a straight line, because:
- this is the equation of the boundary where $[0] + [1] = r$
 - if x axis becomes smaller, y axis needs to be bigger to = r.



So we get this slightly awkward looking diamond

This is the L1 Ball

- Now the **RSS Ball**
 - For each value of the parameters we have some value of RSS
 - We are minimising a combination of RSS (L1 norm or L2 norm in this)
 - So we are interested in RSS of $(w) \leq r$

$$\text{RSS}(w) \leq r$$

- And remember: RSS is quadratic function:

$$\text{RSS} = \sum_{i=1}^n (y_i - w \cdot x_i - b)^2.$$

- So we get an ellipse.
- Our best RSS is achieved at $w_{\text{hat}}^{\text{LS}}$ (best weight vector in least squares)
- But the best RR weight, $w_{\text{hat}}^{\text{RR}}$, if we draw L2 and RSS balls they will touch at this w .
- Similarly for Lasso, w_{hat}^L , the balls touch at that point

But Why Is It Where The Points Meet?

ChatGPT:

The optimal weight vector is the one that minimizes the RSS while also satisfying the constraints of the L1 norm. This means that the weight vector must lie on the boundary of the L1 ball, where the L1 ball is the set of points in weight space that satisfy the constraints of the L1 norm.

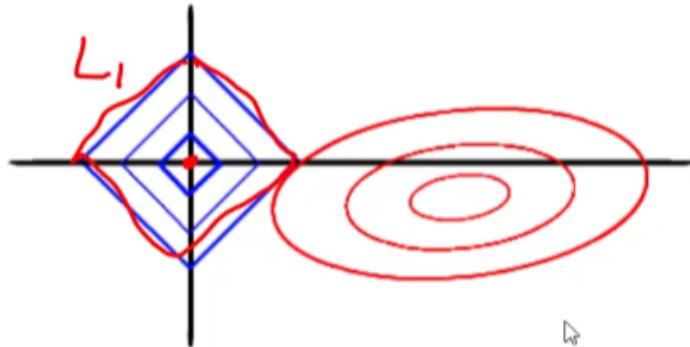
The RSS ball is the set of points in weight space that correspond to the possible values of the weight vector that minimize the RSS. The optimal weight vector is the one that lies on the boundary of the L1 ball and also minimizes the RSS, which means it must be the point at which the RSS ball and L1 ball intersect.

This is why the optimal weight vector in lasso regression is the point at which the RSS ball and L1 ball touch. It represents the trade-off between minimizing the RSS and satisfying the constraints of the L1 norm.

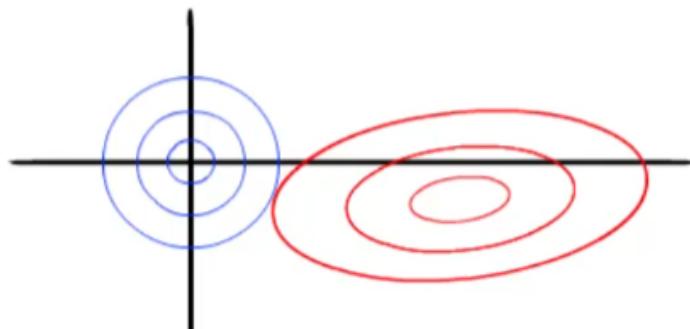
This makes sense. It is the optimal point for both functions.

The geometry of model selection (*) (1)

Lasso



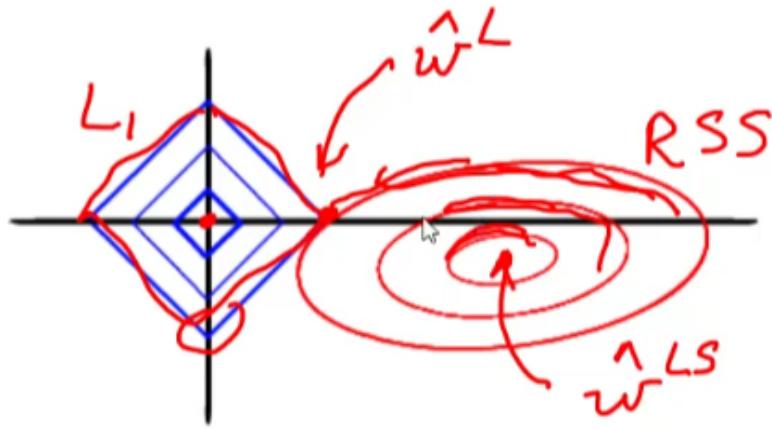
RR



Horizontal axis is $w[0]$ and vertical is $w[1]$

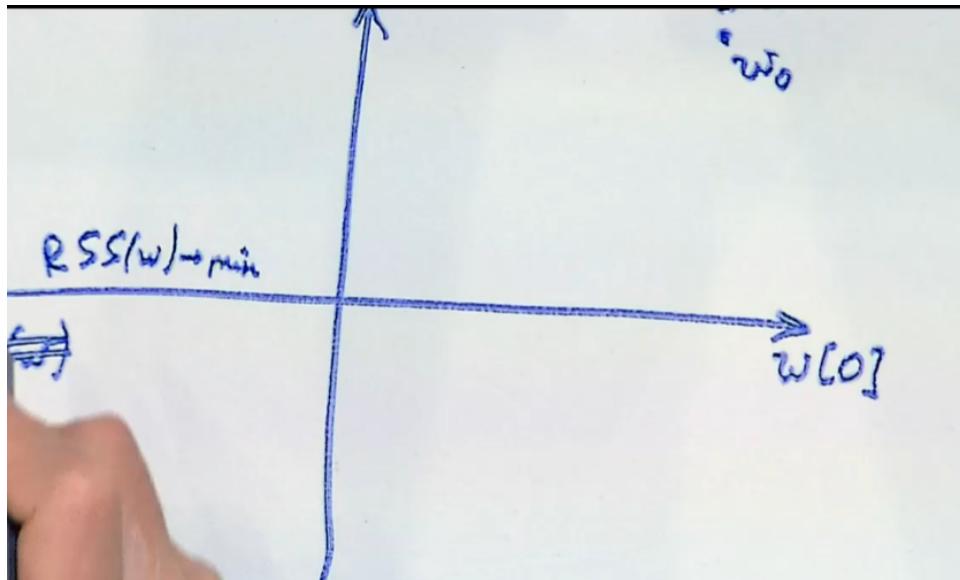
- **Why the Lasso model takes features to zero**

- the x axis will always be more likely to go to zero
- The probability of the corner hitting the ellipse is much higher than if it was an ellipse
 - It is much more prominent/poking out
- And at this point the $w[0] = 0$ where $w_{\text{hat}}(\text{lasso})$ is optimal, hence hat.

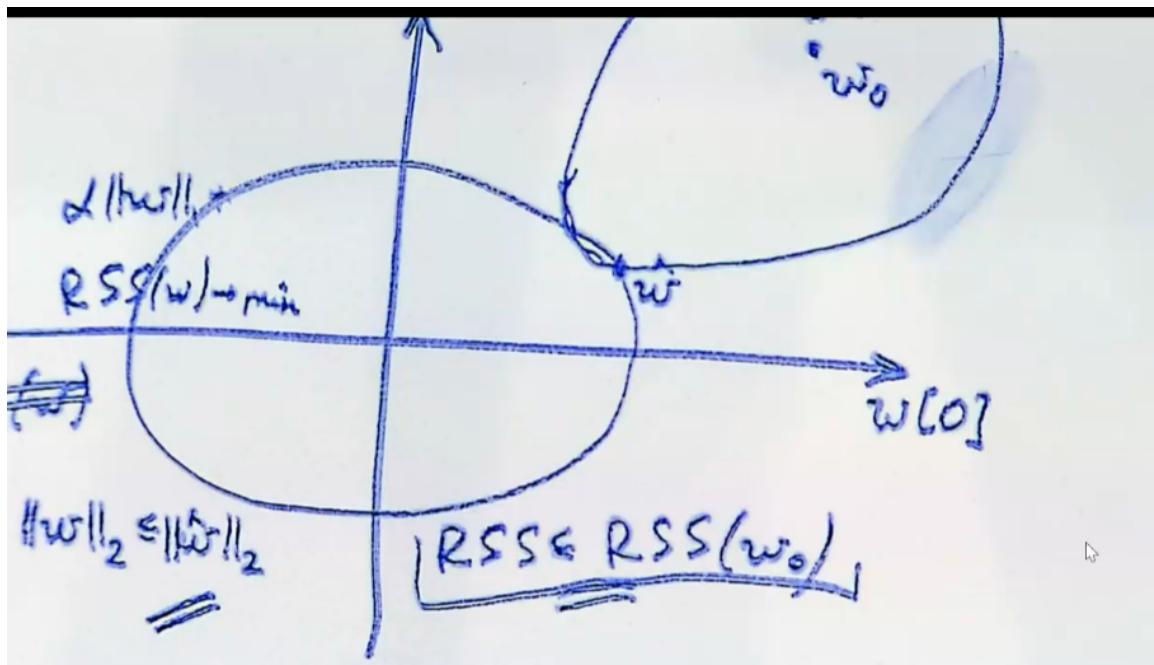
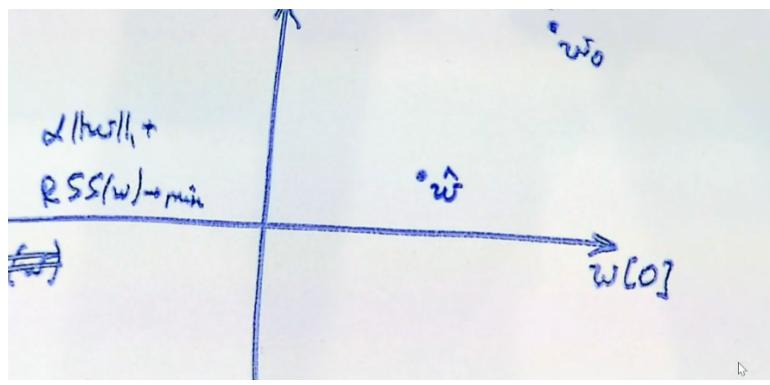


central value is optimal for minimising RSS with least squares

Live Lecture Explanation: This shows why we have different sized rings etc:
28/10/22

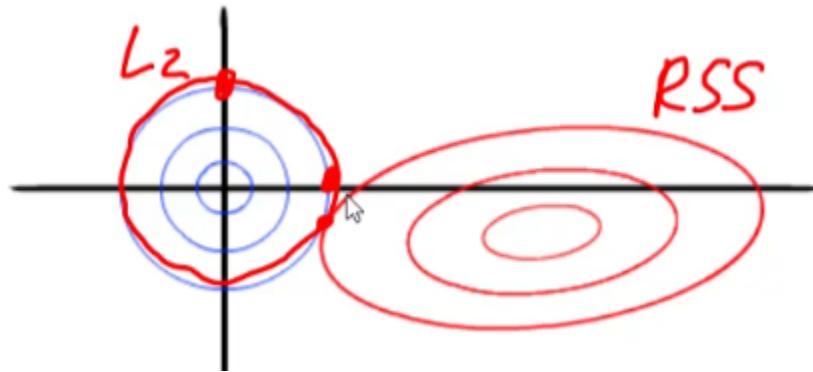


- RSS(w) minimising (as small as possible)
- Suppose we do this at point w_0 (top right)
- But now suppose we have the penalty term $RSS(w) + \alpha * \|w\|_1$
 - This minimum will be a point at another point, lets suppose here $w_{\hat{}}$:

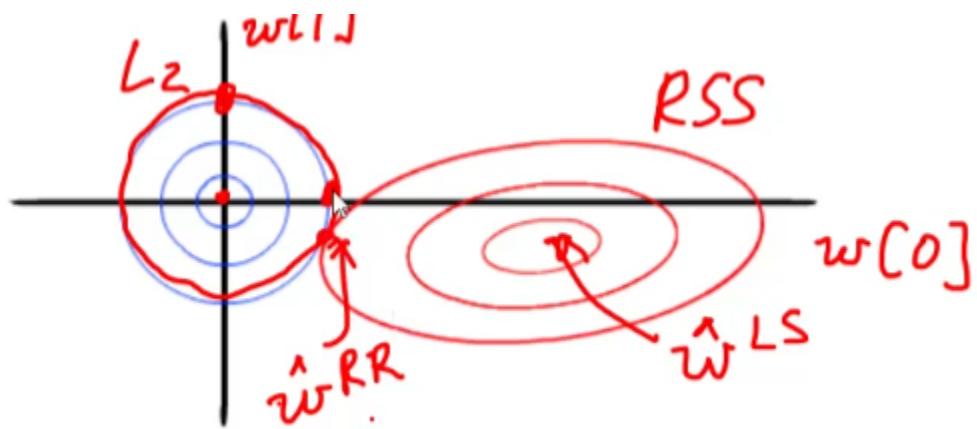


- Now:
 - Top right ellipse is RSS minimising around w_0
 - Circle around origin, with radius w_hat is $||w||_2 \leq ||w_hat||_2$
 - L2 circle with radius of w_hat (optimal)

NEXT



they usually meet at random points not on the highest x or y value



The geometry of model selection (*) (2)

The picture on the previous slide: $p = 2$.

- There is a good chance that the diamond and ellipse will touch at the diamond's corner, especially when the diamond is small (i.e., α is large). Some coefficients will then be zero.
 - In higher dimensions ($p \gg 2$), many of the coefficient estimates may equal zero simultaneously.
 - It would be a remarkable coincidence for this to happen for the circle and ellipse.
-
- if alpha is small, RSS ellipse gets larger but L1 gets smaller
 - **This is intuitive from the equation we are minimising: RSS + alpha * L1 norm \rightarrow min**
 - or L2 norm

With a smaller diamond (L1 norm) it will be more likely to hit the corner.

Elastic net

- scikit-learn also provides the `ElasticNet` class, which combines the penalties of Lasso and Ridge Regression.
- In practice, this combination works best, though at the price of having two parameters to adjust: one for the L_1 regularization, and one for the L_2 regularization.

Discussion

Parameters

- The regularization parameter of Ridge Regression and Lasso is $\alpha \geq 0$.
- Large values of α lead to simpler models.
- Least Squares does not have any parameters (it's the special case of both Ridge Regression and Lasso corresponding to $\alpha = 0$).
- Tuning α is very important.
- Usually α is searched for on a logarithmic scale (for example, we may try $\alpha = 2^k$ for $k = -5, -4, \dots, 5$).

RR smaller values of parameters and Lasso lots of paramters and coefficients will be 0 when alpha is big

Remember that small square means more pointy touches

Cross validation: measure lots of different alpha on a geometric progression, 0.01, 0.1, 1, 10

L_2 or L_1 regularization

- If you assume that only a few of your features are actually important, you should use L_1 regularization.
- Otherwise, you should default to L_2 .
- L_1 can also be useful if interpretability of the model is important: when we have only a few features, it is easier to explain which features are important to the model, and what the effects of these features are.

you dont believe features are relevant, then use L2

Strengths and weaknesses

- ++ Linear models are very fast to train, and also fast to predict.
- ++ They scale to very large datasets and work well with sparse data.
- + Linear models make it easy to understand how a prediction is made.
- It is often not entirely clear why coefficients are the way they are (especially if your dataset has highly correlated features).
- + Linear models often perform well when the number of features is large compared to the number of samples ($p \gg n$).
- However, for smaller p other models might yield better generalization performance (such as Support Vector Machines, to be discussed in Chapter 8).

For first weakness:

Here using RR or Lasso helps because both they push some coefficients to zero.
Least squares can cause volatile coefficients

Next advantage after:

RR and Lasso are perfect for this, give nice solutions, opposed to Least Squares

Using linear models for classification

- Later in the course we will also consider using linear models for classification.
- The idea for binary classification: instead of predicting using a linear function,

$$\hat{y} = w \cdot x^* + b,$$

predict $+1$ if $w \cdot x^* + b > 0$ and predict -1 if $w \cdot x^* + b < 0$ (and do what you want if $w \cdot x^* + b = 0$). The three main methods:

- Logistic regression.
- Linear discriminant analysis.
- Maximum margin classifier.
- Linear models for classification share strengths and weaknesses of linear models for regression.
- Classification is still using the same linear function
 - We still have $y_{\text{hat}} = \dots$
- But Now :
 - y_{hat} is a real number so our prediction will be different.
 - predict $+1$ if $y_{\text{hat}} > 0$ for example

Estimator classes

- Now we continue the discussion started in Lab 2.
 - All machine learning models in scikit-learn are implemented in their own classes, which are called Estimator classes.
 - The Estimator classes implementing the algorithms we have seen so far (or are about to see):
 - KNeighborsClassifier class in the neighbors module (Lab 2).
 - LinearRegression in the linear_model module. This class implements Least Squares.
 - Ridge in the linear_model module. This class implements Ridge Regression.
 - Lasso in the linear_model module. This class implements Lasso.
-

Three fundamental methods (1)

In supervised learning, each Estimator class has three fundamental methods associated with it:

- To instantiate an Estimator class into an object (and perhaps set some parameters of the model), we use the `__init__` method. Example:

```
knn = KNeighborsClassifier(n_neighbors=1)
ridge = Ridge()
```

`__init__`

Discussion Method chaining

Three fundamental methods (2)

- All estimators have a `fit` method, which is used to build the model.
 - The `fit` method always requires as its first argument the data `X`, typically represented as a NumPy array where each row represents a single sample.
 - Supervised estimators also require a `y` argument, which is a one-dimensional NumPy array containing the labels for regression or classification.
- To create a prediction in the form of a new output like `y`, you use the `predict` method.

- fit method to build the model
- first argument is the data (samples) x
 - columns are features
 - rows are samples
- y argument
 - 1D array of labels
- predict method
 - prediction on test samples y

Discussion

Method chaining

Method chaining (1)

- The `fit` method of all `scikit-learn` models returns `self`. This allows us to instantiate a model and fit it in one line:

```
linreg = LinearRegression().fit(X_train, y_train)
```

We have used the return value of `fit` (which is `self`) to assign the trained model to the variable `linreg`.

- This concatenation of method calls (here `__init__` and then `fit`) is known as **method chaining**.
- Another common application of method chaining in `scikit-learn` is to fit and predict in one line:

```
linreg = LinearRegression()
y_pred = linreg.fit(X_train, y_train).predict(X_test)
```

- fit method:

- returns: self
 - ie object its applied to
- so you can combine init and fit.

The screenshot shows a presentation slide with a dark blue header bar. On the left side of the header bar, there are three small white rectangular buttons labeled 'Linear Regression', 'Ridge Regression', and 'Lasso'. To the right of these buttons, the word 'Discussion' is written in white. On the far right of the header bar, there is a vertical column of white text under the heading 'Parameters': 'Ridge Regression or Lasso', 'Strengths and weaknesses', and 'Method chaining'. Below the header bar, the main title of the slide is 'Method chaining (2)', also in white text.

- Finally, you can even do model instantiation, fitting, and predicting in one line:

```
y-pred = LinearRegression().fit(X-train,  
                                y-train).predict(X-test)
```

- The last shortest variant is not ideal. A lot is happening in a single line, which might make the code hard to read. Additionally, the fitted linear regression model isn't stored in any variable, so we can't inspect it or use it to predict on any other data.

Chapter 6: Data preprocessing, parameter selection, and inductive conformal prediction

Vladimir Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS4920/CS5920 Machine Learning
Last edited: October 18, 2020

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Details of data normalization in scikit-learn
Normalizing the training and test sets in the same way
Further details of data normalization

Importance of data normalization

- It is sometimes essential that features be [normalized](#); roughly, they should be measured on the same scale.
- This is not essential for Least Squares.
 - Suppose the first feature, $x[0]$, is measured in meters; let $\hat{w}[0]$ be the corresponding Least Squares estimate.
 - If we decide instead to measure $x[0]$ in kilometers, all $x_i[0]$ will decrease 1000-fold.
 - If you run Least Squares on the new dataset, then $\hat{w}[0]$ will simply increase 1000-fold and the predictions will not change at all!
- This is no longer true for Ridge Regression and Lasso, because of the penalty terms.
- Features measured on the same scale

- not important for least squares
- BUT for Ridge Regression and Lasso not true because they have **Penalty Terms**
 - They use the exact number of $w_{\hat{}}$
 - if we measure $x[0]$ in km ... idk

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Details of data normalization in scikit-learn
Normalizing the training and test sets in the same way
Further details of data normalization

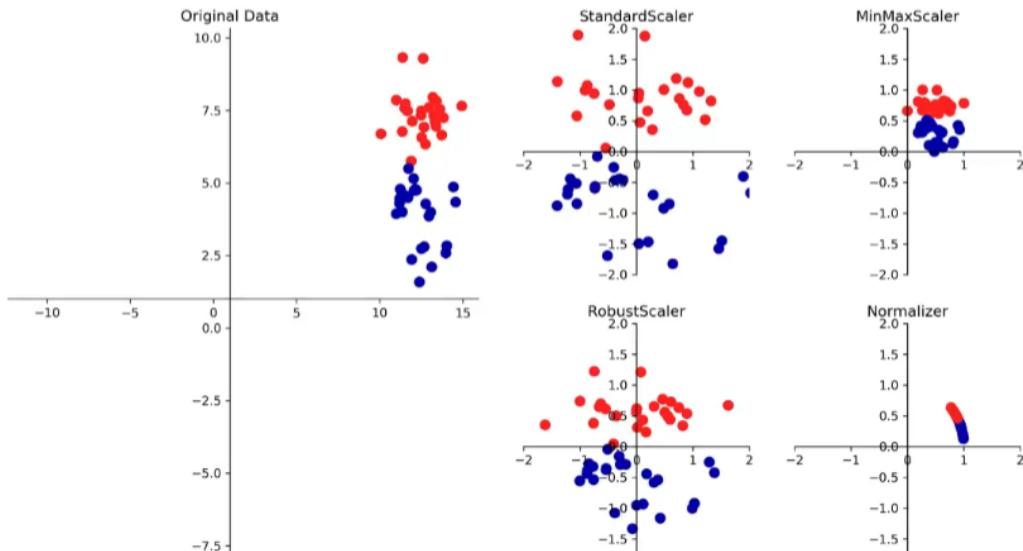
Snippet of Python code

If you look at the definition of `load_extended_boston` (used in Lab 5) in `mglearn`, you will see:

```
def load_extended_boston():
    boston = load_boston()
    X = boston.data
    X = MinMaxScaler().fit_transform(boston.data)
    X = PolynomialFeatures(degree=2,
                           include_bias=False).fit_transform(X)
    return X, boston.target
```

- The function `MinMaxScaler().fit_transform` is doing normalization.
- Let's see how; `scikit-learn` has several functions for per-feature rescaling and shift of the data.
- `MinMax Scaler` creates a normilisation.

Different scalers in scikit-learn



feature 1 is on x axis, feature 2 is on y axis

Scalers are all different but they all do the same thing, they get rid of the arbitrary position.

- **StandardScaler**

- For each Feature:
 - Mean = 0 (situates in along vertical axis)
 - Standard Deviation = 1

- **MinMaxScaler**

- Put data in a box - $\text{sqr}([0, 1])$

- **RobustScaler**

- Useful when data has **outliers**
 - Median = 0

- Inter Quartile Range = 1
- **Normalizer**
 - Normalise each **Sample**
 - as opposed to each feature and transforming the entire dataset
 - length of each sample = 1



- The first plot shows a synthetic two-class classification dataset with two features. The first feature: between 10 and 15. The second feature: between around 1 and 9.
- The following four plots show four different ways to transform the data that yield more standard ranges.
- The `StandardScaler` in `scikit-learn` ensures that for each feature, the mean is zero and the variance is one, thus bringing all features to the same scale.

Data normalization (2)

- The `RobustScaler` works similarly to the `StandardScaler` in that it ensures statistical properties for each feature that guarantee that they are on the same scale. However, the `RobustScaler` uses the median and quartiles, thus ignoring **outliers** (data points that are very different from the rest, such as measurement errors). Outliers often lead to trouble for other scaling techniques.
 - The `MinMaxScaler` shifts the data such that all features range between 0 and 1.
-
- robust, so less sensitive to outliers
 - outliers lead to trouble for other methods, but robust works well

Per-sample normalization

- Normalizing features is not always a good idea.
 - In the case of hand-written digits, such normalization is likely to lead to uniformly grey images, and labelling will become difficult or impossible.
 - In such cases, we can use per-sample normalization (standardizing the brightness, or the brightness and contrast, for each image separately).
 - The first step in this direction in scikit-learn:
`Normalizer`.
-
- Why normalize samples and not digits
 - hand written digits
 - We have pixels
 - what if we normalise pixel by pixel?
 - it means the pixel will become gray and we will lose info about the digit
 - so normalise each digit instead
 - By making its brightness standard and contrast



MinMaxScaler: exercise

- Like the two other Scaler's, MinMaxScaler applies a linear transformation to each feature.
- Suppose we have this training set:

feature 1	feature 2	label
-1	2	A
-0.5	6	B
0	10	A
1	18	C

- Normalize this dataset using MinMaxScaler (the minimum of each feature should become 0 and its maximum should become 1).

Two features not normalised. With MinMax we want to make it 0, 1

- First step:
 - make **minimal value of each feature = 0**
 - for feature 1:** $-1 + 1 = 0$. So add 1 to each feature

$$\begin{array}{c} \text{feature 1} \\ \hline 0 & -1 \\ 0.5 & -0.5 \\ 1 & 0 \\ 2 & 1 \end{array}$$

First goal achieved. Do the same for Feature 2

feature 2
0 2
4 6
8 10
16 18

-2 to each.

- Second step:
 - Maximal value isn't 1, so we need to divide everything by maximum.

feature 1	feature 2	label
0 0	0 0	A
0.25 0.5	0.25 4	B
0.5 1	0.5 8	A
1 2	1 16	C

The right way to do data normalization (1)

- You should apply the same transformation to your training and test sets. Remember that the transformation is found from the training set.
- Two common wrong ways to do data normalization:
 - Normalizing the whole dataset
 - Normalizing the training and test sets separately
- Find transformation for Training set only
 - Then apply transformation to test set.
 - It is more important to apply for test set than training
- Dont normalise whole data set or doing it separately.

The right way to do data normalization (2)

- Suppose you have the training set given on slide 9 and your test set is

	feature 1	feature 2
0		10
1		2
1		14

- You should not try to normalize it separately. (How would you do it if you have only one test sample?)
- Instead you apply the same transformation that you used when you normalized the training set (slides 9–11).
- Remember how we transformed previously:
 - Found this transformation so now apply to test set.

	feature 1	feature 2
1 0		10 8
2 1		2 0
2 1		14 12 ↴

- First steps were $f_1: +1$, $f_2: -2$
- Second steps were:

	feature 1	feature 2
0.5		0.5
1		0
1		0.75

- **BUT**, we see that our test set isn't normalised, but it doesn't need to be! not 0 → 1

Now you can work with your new training and test sets instead of the original ones.

The screenshot shows a presentation slide with a dark blue header bar containing the title 'How to do it in scikit-learn'. Above the slide, there is a navigation bar with several items:

- Data preprocessing
- Parameter selection in scikit-learn
- Inductive conformal predictors
- Details of data normalization in scikit-learn
- Normalizing the training and test sets in the same way
- Further details of data normalization

This is how this is done:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Further details: Lab 6.

- **scaler** : this does the transforming in the object
- **scaler.fit** :
 - dont need any labels, just the data. X
 - **scaler** is now transformed
- Now to apply the transformation
 - **scaler.transform**
 - transform separately using what we found in **scaler**
 - only to **X_train**

- and X_{test}

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Details of data normalization in scikit-learn
Normalizing the training and test sets in the same way
Further details of data normalization

Data snooping

- Data snooping is using the test set for developing your model.
 - There are all kinds of subtle ways in which the test set can leak into your model.
 - You saw an example in Chapter 5: test R^2 (the test set was used in constructing the no-data model).
 - Two more examples in this section: improper normalization leading to data snooping.
-
- test $\text{sqr}(R)$
 - test set was used in constructing no-data model
 - y_{bar}^* in test set

Normalizing the whole dataset (1)

- This is what the authors of M17 do for the Boston Housing dataset: see slide 4.
- It's just a pedagogical device and should not be done in practice since no use of your test data is allowed at the training stage.
- In practice, the test data is simply not available.

Normalising full training set

Normalizing the whole dataset (2)

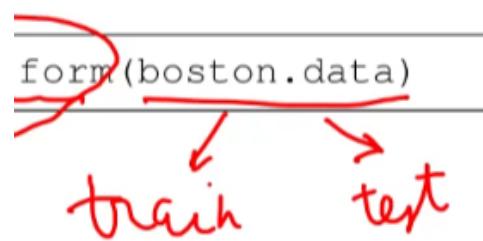
- The code

```
X = MinMaxScaler().fit_transform(boston.data)
```

is a shorthand for

```
scaler = MinMaxScaler()  
scaler.fit(boston.data)  
X = scaler.transform(boston.data)
```

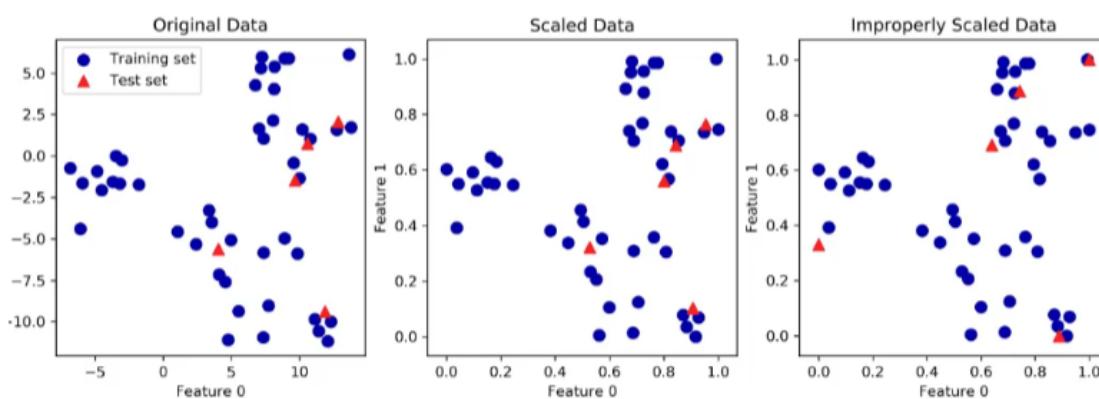
- But `scaler` isn't stored, and so we cannot use it to scale any other data.
- they normalise `boston` data by combining `fit_transform`, `fit` and then `transform`
- shorthand for the next
 - `scaler` isn't stored,



Normalizing the training and test sets separately (1)

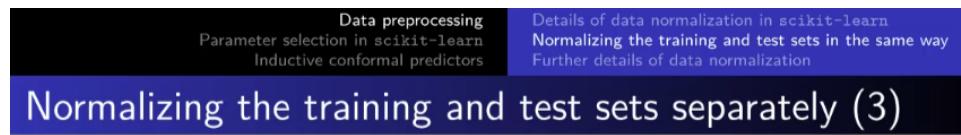
- This kind of improper normalization is even worse.
 - An example is shown on the following slide.
 - Improper scaling distorts the picture (to be discussed).
-
- normalise training and test separately like below

Normalizing the training and test sets separately (2)



- Suppose original data
- and then scaled data

- if properly done, should be based only on training
- Only feature axis are changed, all points are identical
- Improperly
 - Scale training set separately, still every feature is normalised
 - and for red points they also are normalised, from $0 \rightarrow 1$
 - But this is WRONG ! they are distorted



- The first panel is an unscaled two-dimensional dataset, with the training set shown as circles and the test set shown as triangles.
- The second panel is the same data, but scaled using the `MinMaxScaler`. (As shown on slide 15.)
- You can see that the dataset in the second panel looks identical to the first; only the ticks on the axes have changed.
- Now all the features are between 0 and 1. But the minimum and maximum feature values for the test data (the triangles) are not 0 and 1.

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Details of data normalization in scikit-learn
Normalizing the training and test sets in the same way
Further details of data normalization

Normalizing the training and test sets separately (4)

- The third panel shows what would happen if we scaled the training set and test set separately. In this case, the minimum and maximum feature values for both the training and the test set are 0 and 1.
 - But now the dataset looks different. The test points moved incongruously to the training set, as they were scaled differently.
 - As another way to think about this (already mentioned), imagine your test set is a single point.
 - There is no way to scale a single point correctly, to fulfill the minimum and maximum requirements of the MinMaxScaler.
 - But the size of your test set should not change your processing.
-
- Another way to think about separately:
 - look at special case where test set is only 1 single point
 - You can't normalise this, and this is perfectly common

All Above was for MinMaxScaler

Now for StandardScaler

StandardScaler

- You transform the training data to make the mean of each feature 0 and its standard deviation 1.
 - First step: shift each feature down by its mean.
 - Second step: divide each feature by its standard deviation.
-
- mean of features: 0
 - standard deviation: 1
 - Two Steps:
 - First: Shift each feature down by its mean
 - Second: divide each feature by its standard deviation

RobustScaler

- You transform the training data to make the median of each feature 0 and its interquartile distance 1.
- First step: shift each feature down by its median.
- Second step: divide each feature by its interquartile distance.

- A variation to StandardScaler
- Median = 0
- Inter Quartile = 1
- Steps:
 - First: shift each feature down by its median
 - Second: divide each feature by interquartile distance.

[Data preprocessing](#)
[Parameter selection in scikit-learn](#)
[Inductive conformal predictors](#)

Details of data normalization in scikit-learn
 Normalizing the training and test sets in the same way
[Further details of data normalization](#)

Definitions for RobustScaler

- Remember the definitions (for large sets or, more formally, bags):
 - The median of a set of numbers is the number x such that half of the numbers are smaller than x and half of the numbers are larger than x .
 - The lower quartile is the number x such that one-fourth of the numbers are smaller than x .
 - The upper quartile is the number x such that one-fourth of the numbers are larger than x .
 - The interquartile distance is the upper quartile minus the lower quartile.
- For small sets, the definitions are slightly arbitrary, and in this course you do not need to know the details.
- In general for data with lots of points
 - **Median:**

... - . - . - . - . - . - . - . - . - . - .
 50% / , 50%

- divide data into 50/50 and find the middle.

- **Lower Quartile**

Definitions for RobustScaler



- **Upper Quartile**



- **Interquartile distance**



- Distance here

Normalizer

- This normalizer is special in that each sample gets normalized (rather than each feature).
 - Each sample is divided by its Euclidean norm.
 - Therefore, you do not need to worry about normalizing the training and test sets in the same way; it's automatic.
-
- Normalise each sample (not each feature)
 - For each sample:
 - compute Euclidean norm
 - divide sample by Euclidean norm

1 Data preprocessing

2 Parameter selection in scikit-learn

3 Inductive conformal predictors

Problem of parameter selection

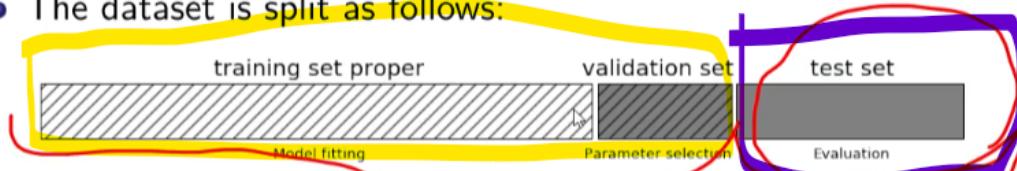
- Finding the values of the important parameters of a model (the ones that provide the best generalization performance, such as α for Lasso) is a tricky task, but necessary for almost all models and datasets.
- Because it is such a common task, there are standard methods in scikit-learn to help you with it.
- The most commonly used method is grid search, which basically means trying all possible combinations of the parameters of interest.
- You will try it in Lab 6.

The danger of overfitting the parameters and the validation set

- The naive approach is to try all combinations of parameters on the test set and choose the best combination.
- However, this would amount to data snooping: we are not allowed to use the test set in developing our model. And the best accuracy on the test set won't necessarily carry over to new data.
- One way to resolve this problem is to split the training set, so we have three sets overall: the **training set proper** to build the model, the **validation set** to select the parameters of the model, and the test set to evaluate the performance of the selected parameters.
- Naive approach
 - data snooping when using the test set (x^* , y^*) DONT USE y^*
 - Also best test set might not work on the next set
- A better approach is to split the training set
 - 2 parts
 - training set proper
 - validation set

Splitting the dataset

- The dataset is split as follows:



- The final step:

- After selecting the best parameters using the validation set, we should rebuild a model using the parameter settings we found, but now training on the full training set (the union of the training set proper and the validation set).
- This way, we can use as much data as possible to build our model.

- Yellow

- training set original
 - model fitting
 - **parameter selection**

- Blue

- test set original
 - used for evaluating final model

Grid search with cross-validation

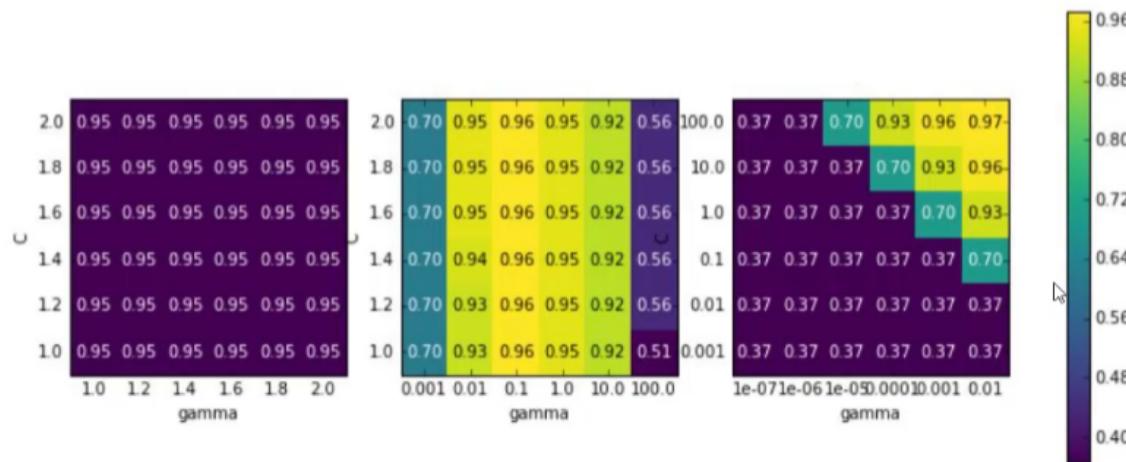
- While the method of splitting the data into a training set proper, a validation set, and a test set is workable, and relatively commonly used, it is quite sensitive to how exactly the data is split.
- For a better estimate of the generalization performance, instead of using a single split of the training set into a training set proper and a validation set, we can use cross-validation to evaluate the performance of each parameter combination.
- Details: Lab 6.
- Cross-validation
 - here we are more economical
 - Use each fold as a validation set in turn and average

Things to be aware of when using grid search with cross-validation

- Grid searches are computationally expensive.
- It is often a good idea to start with a small and coarse grid covering interesting values of parameters.
- Ideally, the values at the centre of the first grid should be significantly better than the values at the boundary. (See the next slide for examples of misspecified grids.)
- Then we can use a finer grid covering the area of good performance.
- computationally expensive
- hierarchcial approach
 - small and coarse grid
 - then consider a finer one

Examples of misspecified grids (for 2 parameters)

The values of accuracy for different pairs of parameters (called gamma and C):



- Bad choices
 - **First case**
 - not good, no dependence on either
 - accuracy, large as possible
 - we dont know best pairs, maybe improvement somewhere
 - **Second case**
 - Geometric progression in gamma
 - here dependence on gamma but not on C, not good
 - But best values are approx in the middle which is good
 - **Third case**
 - Geometric progression on both

- This isn't good either a best accuracy is in the corner, moving north east could be better

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Definition of inductive conformal predictors
Classification
Regression

Plan

- ① Data preprocessing
- ② Parameter selection in scikit-learn
- ③ Inductive conformal predictors

Desiderata for conformal prediction and related methods

- Remember from Chapter 3 that the main desiderata for conformal prediction are validity (satisfied automatically) and efficiency (might not be so easy).
- But there are (at least) two kinds of efficiency:
 - Predictive efficiency (as measured by, e.g., the average false p-value in the case of classification).
 - Computational efficiency (first of all, how long the required computations take and how much computer memory they require).
- Validity
 - automatically satisfied
- Efficiency
 - an art
 - predictive efficiency
 - average false p-value
 - computational
 - how long it takes to perform computational task with memory etc

Computational efficiency of conformal prediction

- Many machine learning algorithms (such as K Nearest Neighbours) can be “conformalized”, and the conformal predictor based on them may be computationally fairly efficient.
- But imagine data preprocessing is required. Then it needs to be redone for each test sample and for each potential label for it (in order to achieve guaranteed validity).
- The same is true for parameter selection.
- In practice, this makes conformal prediction in its pure form not applicable.
- Algos like Nearest Neighbors can be conformalised
 - But if you have to **preprocess**:
 - Has to be done for each test sample and:
 - Done for EACH potential label
 - Have to do the same for parameter selection

Modifications of conformal predictors

In this course we will discuss the following modifications:

- Inductive conformal predictors (the rest of this chapter).
 - They are computationally efficient.
 - They are automatically valid under the IID assumption.
 - They may suffer a drop in predictive efficiency (depending on the learning curve of the underlying algorithm).
- Cross-conformal predictors (part of Chapter 9).
 - They are both predictively and computationally efficient.
 - But they are not provably valid, albeit empirical validity still holds (unless you try hard to ruin it).



- Inductive conformal predictors:
 - computationally efficient
 - automatically valid under IID
 - drop in predictive efficiency
 - we are not using whole training set, only training set proper (the rest we are doing cross validation on a test set made from a part of the training set original)
- Cross-conformal predictors
 - Efficient with predictivity and computational efficiency
 - But not provably valid

Summary: conformal prediction vs inductive conformal prediction

- Both conformal predictors and inductive conformal predictors are automatically valid.
- What is at stake is their efficiency, predictive and computational.
- Inductive conformal predictors typically vastly improve computational efficiency but their predictive efficiency may suffer.
- In Carnegie Mellon speak, inductive conformal predictors are called “split conformal predictors”.
- Inductive / Full conformal prediction
 - Inductive vastly improve efficiecnay computationally but predictive can suffer

Inductive conformity measures

- An **inductive conformity measure** is a function $A : \mathbf{Z}^* \times \mathbf{Z} \rightarrow \mathbb{R}$.
 - Each finite sequence ζ of labelled samples and each labelled sample z are mapped to the **conformity score** $A(\zeta, z)$.
 - Intuitively, $A(\zeta, z)$ says how well z conforms to ζ .
 - Notice: there is no analogue of the “equivariance” requirement we had in Chapter 3.
-
- Inductive **conformity measure**:
 - Function $Z^* \times Z \rightarrow R$
 - Each finite sequence of zeta (Z^*) and each labelled sample Z are mapped to a real number. which creates the conformity score.
 - $A(\text{zeta}, z)$ tells us how well z conforms to zeta
 - how similar to zeta it looks.

Inductive conformal predictors (ICPs)

Split the training set into the **training set proper** z_1, \dots, z_{n-m} and the **calibration set** z_{n-m+1}, \dots, z_n . To find the prediction set for a test sample x^* , for each possible label y :

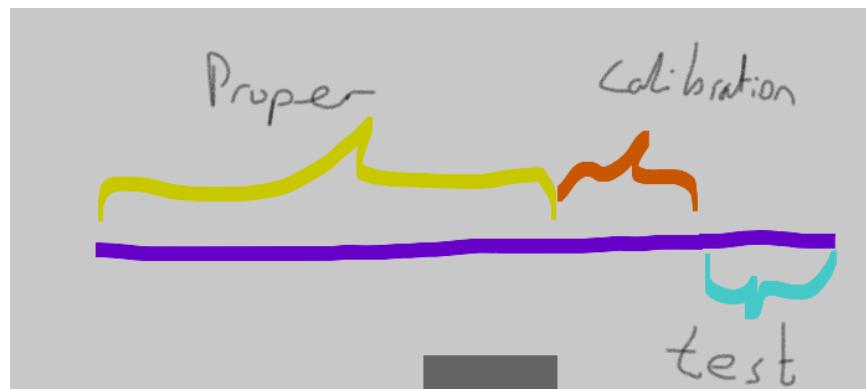
- compute the conformity scores (the **raw prediction step**)

$$\begin{aligned}\alpha_i &:= A((z_1, \dots, z_{n-m}), z_i), \quad i \in \{n-m+1, \dots, n\}, \\ \alpha^y &:= A((z_1, \dots, z_{n-m}), (x^*, y));\end{aligned}$$

- compute the p-value (the **calibration step**)

$$p(y) := \frac{\#\{i = n-m+1, \dots, n : \alpha_i \leq \alpha^y\} + 1}{m+1}.$$

Output the prediction set $\{y \mid p(y) > \epsilon\}$.



- Formal definition:
 - Training set proper: z_1, \dots, z_{n-m}
 - Calibration set : z_{n-m+1}, \dots, z_n
- **Given test sample x^***

- no label
- for each possible y : (each postulated label)

- **Two Steps:**

- **Raw Prediction step:**

- find conformity scores for our calibration set
 - $\alpha_i = A((z_1, \dots, z_{n-m}), z_i)$ where i goes from **$n-m+1, \dots, n$**
 - α_i is conformity score of calibration of labeled samples
 - find conformity scores for test sample with **postulated y**
 - $\alpha_y = A((z_1, \dots, z_{n-m}), x^*, y)$
 - α_y conformity score of test sample

- **Calibration step:**

- Find the p-value
 - compare test sample to calibration sample
 - add 1 to rank because we are interested in augmented calibration set (+ test sample)
 - m is size of augmented training set

Inductive nonconformity measures

- Formally, inductive nonconformity measures are the same thing as inductive conformity measures.
- But when A is referred to as an inductive nonconformity measure, the p-values are computed as

$$p(y) := \frac{\#\{i = n - m + 1, \dots, n : \alpha_i \geq \alpha^y\} + 1}{m + 1}$$

in the calibration step (and the α s are computed as on the previous slide).

- That is, $p(y)$ is the rank of (x^*, y) in the **extended calibration set** (the calibration set extended by adding the test sample with a postulated label) divided by the size of the extended calibration set.
 - The prediction set is still $\{y \mid p(y) > \epsilon\}$.
-
- Inductive nonconformity measure:
 - exactly the same, just changing inequality.

Numerical example: Nearest Neighbour binary classification

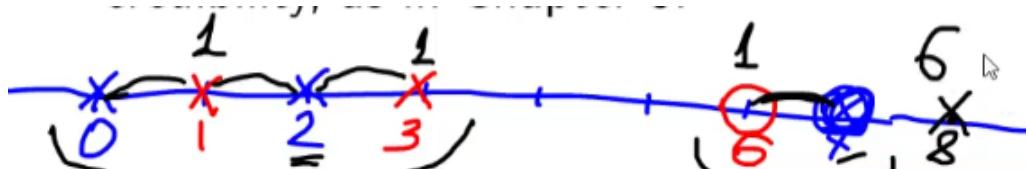
- Take the distance to the nearest neighbour (in the training set proper) of the same class as the nonconformity measure:
 $A(\zeta, (x, y))$ is the smallest distance from x to x' such that $(x', y) \in \zeta$.
 - The training set proper is: 0 and 2 are positive samples and 7 is a negative sample.
 - The calibration set is: 1 and 3 are positive samples and 6 is a negative sample.
 - Compute the two p-values of the test sample 8. Summarize them in terms of the point prediction, confidence, and credibility, as in Chapter 3.
-
- Now lets consider a simple example: **FOR CLASSIFICATION**
 - 1NearestNeighbor Binary Classification
 - Nonconformity measure:
 - We take the shortest distance to NN to sample of same class
 - from formula above: $A(\zeta, (x, y))$
 - ζ is a sequence of labelled samples
 - nonconformity score of (x, y) is shortest distance from x to x' with the same label (x', y) belongs ζ
 - Training set proper: 0, 2 are positive 7 are negative (CIRCLE)
BLUE

- Calibration set: 1, 3 are positive (CROSS) 6 are negative RED
- $x^* = 8$



Compute two p-values:

- P-cross: (postulated as cross)
 - nonconformity score, NNdistance to same class from training set proper
 - inductive non conf. pred.
 - We only compute NS for calibration but using zeta which includes ONLY training set proper so we will get the following NN scores. (Only calculating for RED from the BLUE samples. And also for the test sample)



- To rank:
 - $\alpha_i \geq \alpha_y$
 - $\alpha_i \geq 6$
 - this is only true once
 - **Rank = 1 / 4**
 - 4 because 3 training set proper + test*

Solution (1)

First assume the label of 8 is $+1$. The nonconformity score of a calibration or test sample is the distance to the nearest sample in the training set proper classified in the same way.

Calibration/test sample	Label	Nonconformity score
1	$+1$	1
3	$+1$	1
6	-1	1
8	$+1 (?)$	6

The test sample is the strangest, and the p-value is $1/4 = 0.25$.

TRY THIS FOR other postulated label BELOW

Solution (2)

Now assume that the label of 8 is -1 .

Calibration/test sample	Label	Nonconformity score
1	$+1$	1
3	$+1$	1
6	-1	1
8	$-1 (?)$	1 ↗

The test sample is one of the least strange, and the p-value is $4/4 = 1$.

The point prediction is -1 , the confidence is 0.75, and the credibility is 1.

- point prediction = label prediction = -1
- confidence = 0.75

- 2nd largest p-value
 - In Binary Case, only two are possible p-cross = 0.25, p-circle = 1
 - so confidence = 1 - (2nd larg. p-val.)
 - $1 - 0.25 = \mathbf{0.75}$
- Credibility = largest p-value = 1

This is for REGRESSION

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Definition of inductive conformal predictors
Classification
Regression

Two families of inductive nonconformity measures (as in Chapter 3)

- In the case of regression, we will use the nonconformity scores

$$\alpha = |y - \hat{y}|$$

for each calibration/test sample, where y is its label and \hat{y} is a prediction for y computed from the training set proper.

- Another way to define nonconformity scores is

$$\alpha = |y - \hat{y}| / \sigma,$$

where \hat{y} is a prediction for y and $\sigma > 0$ is an estimate of its accuracy, both computed from the training set proper.

- alpha is defined as difference between true label and predicted label
 - $y - y_{\text{hat}}$

The prediction set for regression: statement (1)

- Let us fix a significance level ϵ ; the prediction set at level ϵ can be computed explicitly.
- Sort the nonconformity scores

$$\alpha_i = \underline{|y_i - \hat{y}_i|}$$

of the calibration samples in ascending order; assume they are all different. (Here y_i is the true label and \hat{y}_i is the prediction computed from x_i as test sample and the training set proper as training set.)

- Let the sorted sequence be

$$\alpha_{(1)} \leq \dots \leq \alpha_{(m)}$$

(m is the size of the calibration set).

- Sort our NS of the calibration samples in ascending order**
- paranthesis signal alpha is sorted

The prediction set for regression: statement (2)

$$k \approx (1-\epsilon)m$$

- Set

$$k = \lceil (1 - \epsilon)(m + 1) \rceil,$$

where $\lceil t \rceil$ (the ceiling of t) is the smallest integer $\geq t$ (e.g., $\lceil 2.3 \rceil = 3$).

- Set $c = \alpha(k)$.
- The prediction set for any test sample is $[\hat{y} - c, \hat{y} + c]$, where \hat{y} is the point prediction for its label (computed from the training set proper).
- A disadvantage of this definition: all prediction sets are intervals of the same length. The second definition on slide 46 is more adaptive.

- Given significance level, **epsilon** (the target probability of error)
- and Given, **m**, size of calibration set
 - We Find the value k:
 - the ceiling of $(1-\text{epsilon}) * (m+1)$
 - Now find the kth nonconformity score:
 - $c = \alpha(k)$
 - **c** is the half width of the prediction interval
- prediction set = $[\hat{y}_\text{hat} - c, \hat{y}_\text{hat} + c]$

Now lets prove it:

First an intuitive thought I had:

- ϵ is some threshold/conf. level eg 0.05 or 5%
- $1-\epsilon = 0.95$
- $0.95 * m$ to get 95th percentile point
- and choose the alpha of that point

Data preprocessing
Parameter selection in scikit-learn
Inductive conformal predictors

Definition of inductive conformal predictors
Classification
Regression

The prediction set for regression: check (1) (*)



- Let us check the statement on the previous slide.
- Consider a test sample with postulated label y and prediction \hat{y} (computed from the training set proper).
- If $|y - \hat{y}|$ is just above $\alpha_{(k)}$, the rank of the test sample in the extended calibration set will be $m - k + 1$.
- Indeed, the nonconformity scores that are at least as large as that for the test sample are $\alpha_{(k+1)}, \dots, \alpha_{(m)}$ and the nonconformity score for the test sample itself; there are $m - k + 1$ of them.
- Considering a test sample, \mathbf{x}^*
- postulated label y
- prediction $\mathbf{y_hat}$,
 - which can be the nearest neighbor for example (and remember we do it from training set proper)
- when will the **p-value $\leq \epsilon$**

- We will get to this in next slide!
- imagine $|y - y_{\hat{}}|$ is just above $\alpha(k)$
 - proving **$m - k + 1$ is the rank**
 - because: **$m + 1$** is the size of extended calibration set
 - and its k th so $-k$
 - so from **$\alpha(k)$** we know that there are **$\alpha(k+1), \dots, \alpha(m)$** more α values ahead
 - there are **$m - (k + 1)$** of these values (which is the same as the **rank**)

- Its p-value will be $(m - k + 1)/(m + 1)$.
- The inequality

$$\frac{m - k + 1}{m + 1} \leq \epsilon$$

is equivalent to

$$k \geq (1 - \epsilon)(m + 1).$$

- so p-value = $(m - k + 1) / (m + 1)$
- when will the **p-value $\leq \epsilon$**
 - 2nd bullet point

Now for an example:

Exercise

- In the problem of regression, consider the training set proper

$$(x_1, y_1) = (0, 0), \quad (x_2, y_2) = (1, 1), \\ (x_3, y_3) = (2, 2), \quad (x_4, y_4) = (3, 3),$$

consisting of four labelled samples, and the calibration set

$$(x_5, y_5) = (0, 1), \quad (x_6, y_6) = (1, -1), \\ (x_7, y_7) = (2, 1.5), \quad (x_8, y_8) = (3, 1.5),$$

also consisting of four labelled samples.

- Use $\alpha = |y - \hat{y}|$ as inductive nonconformity measure, where \hat{y} is the Nearest Neighbour prediction computed from the training set proper.
- Find the prediction set at significance level $\epsilon = 20\%$ for the test sample $x^* = 4$.

Regression problem

- Given:
 - training set proper
 - calibration set
- $\alpha = |y - y_{\text{hat}}|$ where y_{hat} is NN from training set proper
- find prediction set at significance level $\epsilon = 20\%$ for $x^* = 4$

First Compute NonConformity Scores

Calibration observation	Nearest neighbour	Nonconformity score
(0, 1)	(0, 0)	1
(1, -1)	(1, 1)	2
(2, 1.5)	(2, 2)	0.5
(3, 1.5)	(3, 3)	1.5

- FOR (0, 1)
 - NN to (0, 1) is (0, 0)

In the problem of regression, consider the training set proper

$$\begin{array}{ll} (x_1, y_1) = (0, 0), & (x_2, y_2) = (1, 1), \\ (x_3, y_3) = (2, 2), & (x_4, y_4) = (3, 3), \end{array}$$

from training set proper.

NS for (0, 1)

$$|y - y_{\text{hat}}| = 1 - 0 = 1$$

Sorting Nonconformity scores

$$\left(\alpha_{(1)}, \alpha_{(2)}, \alpha_{(3)}, \alpha_{(4)} \right) = (0.5, 1, 1.5, 2).$$

- So now, $k = \text{ceiling} ((1 - \epsilon)(m+1))$

$$k = \lceil (1 - 0.2)(4 + 1) \rceil = 4$$

- $c = \alpha(k) = \alpha(4) = 2$
- Since, the NN of $x^*=4$, is 3
 - our prediction interval is 3 ± 2
 - **interval = [1, 5]**

What should be the size of the calibration set?

- This obviously depends on the size of the training set (you do not want to have a tiny training set proper).
- But if your training set is large enough, the rule of thumb is to include at least $5/\epsilon$ training samples in the calibration set, where ϵ is your significance level (or your smallest significance level if you have several values of ϵ in mind, which is usually a good idea).
- This rule of thumb ensures that the boundaries of the prediction set are determined by groups of samples of size at least 5 (and so are not unduly affected by randomness).
- atleast $5/\epsilon$ training samples in calibration set

Kernel Methods

Chapter 7: Kernel methods

Vladimir Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS4920/CS5920 Machine Learning

Last edited: August 31, 2020

Introduction and simple examples
General theory of kernels and a further example
Kernelization of prediction algorithms

Idea
Simple examples

Idea

- Kernel methods are another way to turn linear methods into non-linear ones.
- There are many methods in machine learning whose predictions depend only on the dot products between the samples.
- For example, the predictions computed by Nearest Neighbours only depend on (squared) distances

$$\|x - x'\|^2 = x \cdot x - 2x \cdot x' + x' \cdot x'.$$

- Now suppose that every time a dot product $x \cdot x'$ appears somewhere, we replace it with a **generalization** of the dot product of the form $K(x, x')$, where K is some function that we will refer to as a **kernel** (the precise definition is to follow).
- linear method to non linear

- before we did feature engineering
 - not efficient, feature by feature
- Expanding the expression of NN distance * NN distance, we have to do dot product

$$(x - x') \cdot (x - x')$$

Idea of Kernels

- Everytime we see $x \cdot x'$ (dot product)
 - we generalise with some function $K(x, x')$
 - Kernel

Introduction and simple examples
 General theory of kernels and a further example
 Kernelization of prediction algorithms

Idea
 Simple examples

Linear kernel

- Intuition: a kernel is a function that quantifies the similarity between two samples (details: later).
- For example, we could simply take

$$K(x, x') = \underbrace{x \cdot x'}_{= \sum_{j=0}^{p-1} x[j]x'[j]},$$

which would just give us back the original method.

- This is known as the **linear kernel**, because our method stays linear.
- The linear kernel essentially quantifies the similarity of a pair of samples using correlation (assuming the samples are standardized by dividing by their standard deviation, as explained later).

Linear Kernel

- Easiest is just the linear kernel which is dot product
- So we just take the sum of all features component wise.

Introduction and simple examples General theory of kernels and a further example Kernelization of prediction algorithms	Idea Simple examples
Polynomial kernels	

- But one could instead choose another form for the kernel, such as
$$K(x, x') = \underbrace{(1 + x \cdot x')}_{\text{where } d \text{ is a positive integer.}}^d,$$
- This is known as a **polynomial kernel of degree d** .
- Using such a kernel with $d > 1$, instead of the standard linear kernel, leads to much more flexible models.
- It amounts to fitting the original method in a higher-dimensional space involving polynomials of degree d , rather than in the original sample space.

Polynomial Kernel

- A non trivial kernel
 - raise to some power d
 - if 1, really easy
 - otherwise, above.

Radial kernel

- Another popular choice is the **radial kernel**

$$\begin{aligned} K(x, x') &= \exp(-\gamma \|x - x'\|^2) \\ &= \exp\left(-\gamma \sum_{j=0}^{p-1} (x[j] - x'[j])^2\right), \end{aligned}$$

where γ is a positive constant.

Radial Kernel

- Kernel between x and x' (typically two vectors in the euclidean space)
 - take the difference
 - measure of similarity
 - take its length (the norm)
 - square it
 - multiply by gamma
 - exponentiate it

$$K(x, x') = \exp\left(-\gamma \|x - x'\|_2^2\right) \in (0, 1]$$

$$= \exp\left(-\gamma \sum_{j=0}^{p-1} (x[j] - x'[j])^2\right),$$

- When $x - x'$ is very large, close to 0
- When $x - x'$ is very small, close to 1

Introduction and simple examples
General theory of kernels and a further example
 Kernelization of prediction algorithms

Kernel trick
 Operations on kernels
 String kernels

Plan

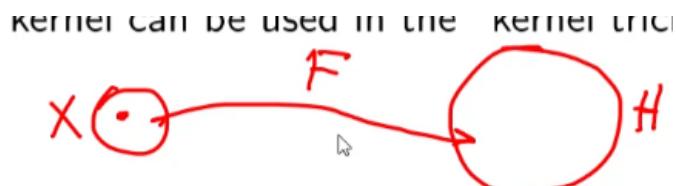
- ① Introduction and simple examples
- ② General theory of kernels and a further example
- ③ Kernelization of prediction algorithms

Formal definition of kernels

- The **definition** of a kernel: we take a **feature mapping** $F : \mathbf{X} \rightarrow H$ of the sample space \mathbf{X} into a **feature space** H equipped with dot product (formally, a “Hilbert space”). This gives us the **kernel**

$$K(x, x') = F(x) \cdot F(x').$$

- In general, a **kernel** is any function that can be obtained this way.
 - Any such kernel can be used in the “kernel trick”.
-
- $F : \mathbf{X} \rightarrow H$
 - A function F that maps \mathbf{X} to H
 - Where \mathbf{X} is the space where our unlabelled samples live
 - H is a higher dimensional space where there is also dot product



General scheme

- a feature mapping F can be combined with nearly any algorithm
- if the original algorithm contains the training and test samples only in dot products, we can write a kernel version of this algorithm
- **kernel trick:**
 - write the algorithm in such a way that all xs only appear in dot products
 - replace all dot products with kernels

An advantage of using kernels

$$K(x, x')$$

- We do not need to use F explicitly!
- For many simple-looking kernels (such as radial kernels), the corresponding feature spaces can even be infinite-dimensional.
- There is a simple criterion for K being a kernel: it should be symmetric and positive definite. (Next slide.)
- F can be extremely complicated, so we are only interested in $K(x, x')$

$$K(x, x') = f(x) \cdot f(x')$$

- There is a simple criteria for the above to hold
 - symmetric
 - positive definite

Introduction and simple examples General theory of kernels and a further example Kernelization of prediction algorithms	Kernel trick Operations on kernels String kernels
Criterion (1)	

- A function $K(x, x')$, where $x, x' \in \mathbf{X}$, is **symmetric** if it is always true that

$$K(x, x') = K(x', x).$$

- K is **positive definite** (or nonnegative definite) if, for any n , any samples x_1, \dots, x_n , and any numbers a_1, \dots, a_n ,

$$\sum_{i=1}^n \sum_{j=1}^n a_i a_j K(x_i, x_j) \geq 0.$$

- $K(x, x') = K(x', x)$
- positive definite
 - for any samples $\mathbf{x1}, \dots, \mathbf{xn}$
 - and any numbers $\mathbf{a1}, \dots, \mathbf{an}$

The combination should be non negative

$$\sum_{i=1}^n \sum_{j=1}^n a_i a_j K(x_i, x_j) \geq 0.$$

Criterion (2)

Theorem

A continuous function $K : \mathbf{X}^2 \rightarrow \mathbb{R}$ is a kernel if and only if it is symmetric and positive definite.

- If \mathbf{X} is a discrete space, then K is continuous by definition.
 - In one direction the statement is obvious (see the next slide), in the other much less so (not in this course).
 - The polynomial and radial kernels are symmetric and positive definite (although the latter is not easy to check).
 - Theorem of the method from previous slide
 - $K(x, x')$ is a kernel only if
 - If \mathbf{X} is a discrete space, then K is continuous
 - usually discrete

Criterion (3)

- Any kernel is symmetric:

$$K(x, x') = F(x) \cdot F(x') = F(x') \cdot F(x) = K(x', x).$$

↓

- Any kernel is positive definite:

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n a_i a_j K(x_i, x_j) &= \sum_{i=1}^n \sum_{j=1}^n a_i a_j F(x_i) \cdot F(x_j) \\ &= \left(\sum_{i=1}^n a_i F(x_i) \right) \cdot \left(\sum_{j=1}^n a_j F(x_j) \right) = \left\| \sum_{i=1}^n a_i F(x_i) \right\|^2 \geq 0. \end{aligned}$$

- Symmetric:
 - we know dot product is symmetric so its easy
- Positive definite:
 - We need to establish the linear combination on LHS is non negative:
 - write it out with $F()$
 - we can represent as a sum of two Sigmas
 - But they are both identical as i and j are the same
 - Simple now! the square is always non negative

Combining kernels (1)

The theorem immediately implies (assuming the kernels are continuous):

- The sum of two kernels is a kernel: if K_1 and K_2 are kernels, so is $K_1 + K_2$.
- Kernels can be scaled: if K is a kernel and $w > 0$, then wK is a kernel as well.

Exercise: prove these implications.

Not in this course: the product of two kernels is also a kernel.

Combining kernels (2)

- Suppose we have several kernels, say K_1, K_2, K_3 , and do not know which one to use.
→
- We can use their sum $K_1 + K_2 + K_3$ instead.
- More generally, we can use $w_1 K_1 + w_2 K_2 + w_3 K_3$ for any positive weights w_1, w_2, w_3 .

- If we don't know which kernel to use
 - just add all together and use the sum

$$0.8 K_1 + 0.1 K_2 + 0.1 K_3$$

Introduction and simple examples
 General theory of kernels and a further example
 Kernelization of prediction algorithms

Kernel trick
 Operations on kernels
 String kernels

Normalizing kernels (1)

- Kernels are often interpreted as measures of similarity:
 $K(x, x')$ shows how similar x and x' are.
- For the linear kernel, this interpretation makes sense only if x and x' have the same length.
- Therefore, the feature mapping F is often **normalized**, i.e., replaced by

$$\tilde{F}(x) = F(x) / \|F(x)\|.$$

- The corresponding kernel is the **normalized kernel**

$$\tilde{K}(x, x') = \tilde{F}(x) \cdot \tilde{F}(x') = \frac{F(x) \cdot F(x')}{\|F(x)\| \|F(x')\|} = \frac{K(x, x')}{\sqrt{K(x, x)} \sqrt{K(x', x')}}.$$

- Kernels are often interpreted as a measure of similarity:
- **linear kernel**

- only makes sense x and x' have the same length
- Therefore we often like to normalise feature mapping F .
- So to do this:

$$\tilde{F}(x) = F(x) / \|F(x)\|.$$

- we divide each $F(x)$ by its length
 - this ratio $\tilde{F}(x)$ always has length of 1

Normalised Kernel

$$\tilde{K}(x, x') = \frac{\tilde{F}(x) \cdot \tilde{F}(x')}{\|\tilde{F}(x)\| \|\tilde{F}(x')\|} = \frac{F(x) \cdot F(x')}{\|F(x)\| \|F(x')\|} = \frac{K(x, x')}{\sqrt{K(x, x)} \sqrt{K(x', x')}}.$$

Then using definition of a kernel we can get what is on the RHS

$$= \frac{K(x, x')}{\sqrt{K(x, x)} \sqrt{K(x', x')}}.$$

- **But be careful for division by 0**
 - There is a danger: what if $K(x, x) = 0$ or $K(x', x') = 0$? (i.e., $F(x) = 0$ or $F(x') = 0$?)
 - In this case you should set, say, $\tilde{K}(x, x') = 0$ instead of trying to apply the general formula (which would lead to a runtime error in Python).

- The numerator has $k(x, x)$ and $k(x',x')$

String Kernels

Introduction and simple examples General theory of kernels and a further example Kernelization of prediction algorithms	Kernel trick Operations on kernels String kernels
Introduction	

- This subsection: a kernel between two text documents (or “strings”).
- Idea: to compare them by means of the substrings (of a given length c) they contain.
 - the more substrings in common, the more similar they are
- It is important that such substrings do not need to be contiguous, and the degree of contiguity of one such substring in a document determines how much weight it will have in the comparison.
- In general we want to see kernel (measure of similarity) between two things
- To compare two strings
 - Count how many common substrings they contain
 - each has fixed length c
 - **more substring, more similar**
 - **substring dont need to be contiguous**
 - they can have some gaps

Decay factor λ

- For example: the substring “c-a-r” is present both in “card” and in “custard”, but with different weighting.
- For each such substring there is a dimension of the feature space, and the value of such coordinate depends on how frequently and how compactly the substring is embedded in the text.
- To deal with non-contiguous substrings (“ subsequences”), it is necessary to introduce a **decay factor** $\lambda \in (0, 1]$ that can be used to weigh the gaps.
- If we have substring “c-a-r”
 - present in “card” and “custard”
 - card is contiguous, no gaps
 - custard has gaps, less weight
- so we need some feature mapping in kernel methods
- For each substring of, length c, there will be a dimension of feature space
- to deal with noncontiguous substrings, (subsequences), we need decay factor lamda

Lets see an example:

Example (1)

- Consider the strings (mini-documents) “cat”, “car”, “bat”, and “bar”. For $c = 2$ (the length of the subsequences taken into account), we obtain an 8-dimensional feature space, where the words are mapped as follows:

	a-r	a-t	b-a	b-r	b-t	c-a	c-r	c-t
$F(\text{cat})$	0	λ^2	0	0	0	λ^2	0	λ^3
$F(\text{car})$	λ^2	0	0	0	0	λ^2	λ^3	0
$F(\text{bat})$	0	λ^2	λ^2	0	λ^3	0	0	0
$F(\text{bar})$	λ^2	0	λ^2	λ^3	0	0	0	0

- The (unnormalized) kernel between “car” and “cat” is

$$K(\text{car}, \text{cat}) = \lambda^4.$$

- Four mini documents.
 - interested in their similarities

First:

- Kernel for $K(\text{car}, \text{cat})$
- $K(\text{car}, \text{cat})$**
 - using $c = 2$
 - top row is list of all subsequences, $c = 2$, that occur in all our documents

	a-r	a-t	b-a	b-r	b-t	c-a	c-r	c-t	a-a	a-b
F(cat)	0	λ^2	0	0	0	λ^2	0	λ^3	0	0
F(car)	λ^2	0	0	0	0	λ^2	λ^3	0	0	0
F(bat)	0	λ^2	λ^2	0	λ^3	0	0	0	0	0
F(bar)	λ^2	0	λ^2	λ^3	0	0	0	0	0	0

The (unnormalized) kernel between "car" and "cat" is

- columns are subsequences
- if noncontiguous, $\lambda^{\text{length of subsequence}}$

Now to compute $K(\text{car}, \text{cat})$

- cat is mapped to first vector(first row)
- car is mapped to second vector(second row)

	a-r	a-t	b-a	b-r	b-t	c-a	c-r	c-t
F(cat)	0	λ^2	0	0	0	λ^2	0	λ^3
F(car)	λ^2	0	0	0	0	λ^2	λ^3	0
F(bat)	0	λ^2	λ^2	0	λ^3	0	0	0
F(bar)	λ^2	0	λ^2	λ^3	0	0	0	0

- we want:
 - $F(\text{cat}) \cdot F(\text{car})$
 - multiply component wise getting
 - $\lambda^2 * \lambda^2 = \lambda^4$
 - as this is the only component wise possibility

	a-r	a-t	b-a	b-r	b-t	c-a	c-r	c-t
F(cat)	0	λ^2	0	0	0	λ^2	0	λ^3
F(car)	λ^2	0	0	0	0	λ^2	λ^3	0
F(bat)	0	λ^2	λ^2	0	λ^3	0	0	0
F(bar)	λ^2	0	λ^2	λ^3	0	0	0	0

Summary of what we did:

Example (2)

- The cell in column u and row $F(s)$ is 0 if u is not a subsequence of s .
- If u is a subsequence of s , the cell is λ^k , where k is the length of the embedding of u into s (for now assume the embedding is unique).
- When computing the length of the embedding, you should also count the letters that need to be added to u to get a contiguous substring of s .

Example (3)

- The normalized kernel is:

$$\tilde{K}(\text{car}, \text{cat}) = \frac{\lambda^4}{2\lambda^4 + \lambda^6} = \frac{1}{2 + \lambda^2},$$

since

$$K(\text{car}, \text{car}) = K(\text{cat}, \text{cat}) = 2\lambda^4 + \lambda^6.$$

- In general, the document will contain more than one word, but we make it into one string by concatenating all the words and the spaces (ignoring the punctuation).
- In many cases, comparing documents of different length. we want to normalise
- Normalised $K(\text{car}, \text{cat})$

$$\tilde{K}(\text{car}, \text{cat}) = \frac{\lambda^4}{2\lambda^4 + \lambda^6} = \frac{1}{2 + \lambda^2},$$

Remember:

$$= \frac{K(x, x')}{\sqrt{K(x, x)} \sqrt{K(x', x')}}.$$

so normalised kernel is:

$$) = \frac{1}{2 + \lambda^2},$$

Example with non-unique embeddings

- Consider the string “London”. For $c = 2$, we obtain an 11-dimensional feature vector:

$$\begin{array}{cccccc}
 d-n & d-o & L-d & L-n & L-o \\
 \hline
 \lambda^3 & \lambda^2 & \lambda^4 & \lambda^3 + \lambda^6 & \lambda^2 + \lambda^5 \\
 & & & & \\
 n-d & n-n & n-o & o-d & o-n & o-o \\
 \hline
 \lambda^2 & \lambda^4 & \lambda^3 & \lambda^3 & 2\lambda^2 + \lambda^5 & \lambda^4
 \end{array}$$

- The string “London” maps to the feature vector
- $$(\lambda^3, \lambda^2, \lambda^4, \lambda^3 + \lambda^6, \lambda^2 + \lambda^5, \lambda^2, \lambda^4, \lambda^3, \lambda^3, 2\lambda^2 + \lambda^5, \lambda^4).$$
- Each cell is the sum of λ^k over all embeddings.

Non unique imbeddings

- string “London”, $c = 2$
- repeating letters
 - Just add both lamda imbeddings

Putting everything together

- Once we have created a kernel, it is natural to normalize to remove any bias introduced by document length.
- We create a new embedding $\tilde{F}(s) = F(s)/\|F(s)\|$, which gives rise to the kernel

$$\tilde{K}_c(s, t) = \frac{K_c(s, t)}{\sqrt{K_c(s, s)}\sqrt{K_c(t, t)}}$$

(remember c is the length of the subsequences taken into account).

- We can create a kernel $K(s, t)$ that combines different $\tilde{K}_c(s, t)$ giving different (positive) weights to several c (as explained on slide 15).

Efficient computation of string kernels

- String kernels can be computed efficiently.
- Not in this course: it is possible to speed up the computation of $K_c(s, t)$ to $O(c|s||t|)$ (where $|\cdot|$ is the length of a string, `len` in Python).

Plan

- ① Introduction and simple examples
- ② General theory of kernels and a further example
- ③ Kernelization of prediction algorithms

Kernelization Kernelisation of Prediction Algorithms

Kernels with Nearest Neighbours

Very easy to kernelize:

- In the Nearest Neighbours algorithm the predicted label for a test sample x^* is taken from the nearest training samples.
- Let us map all samples into a feature space; the squared distance becomes

$$\begin{aligned} [d(F(x), F(x'))]^2 &= \|F(x) - F(x')\|^2 \\ &= (F(x) - F(x')) \cdot (F(x) - F(x')) \\ &= F(x) \cdot F(x) + F(x') \cdot F(x') - 2F(x) \cdot F(x') \\ &= K(x, x) + K(x', x') - 2K(x, x'). \end{aligned}$$

- Kernelized version of Nearest Neighbours: measure the distance by

$$d_K(x, x') = \sqrt{K(x, x) + K(x', x') - 2K(x, x')}.$$

Suppose we have two samples, \mathbf{x} and \mathbf{x}' (x prime)

- We map both of them to some high dimensional feature space.

$$[d(\underline{F(x)}, \underline{F(x')})]^2$$

- And we measure this distance $d()$ and lets square it
 - Distance between $\mathbf{F(x)}$ and $\mathbf{F(x')}$ is the distance between these two vectors:

$$\|\underline{\mathbf{F}(x)} - \underline{\mathbf{F}(x')}\|^2$$

And square the norm of these two vectors

So we can write it as :

$$= (\underbrace{F(x) - F(x')}_\text{---} \cdot \underbrace{F(x) - F(x')}_\text{---})$$

And now expanding this product and putting in kernel form:

$$\begin{aligned} &= F(x) \cdot \overrightarrow{F(x) + F(x')} - 2F(x) \cdot F(x') \\ &= K(x, x) + K(x', x') - 2K(x, x'). \end{aligned}$$

So we can use this expression instead of the distance, and nearest neighbors becomes more flexible.

$$d_K(x, x') = \sqrt{K(x, x) + K(x', x') - 2K(x, x')}.$$

You can ignore sqrt root to make algo more efficient, nearest neighbors only depends on ranks of distances, not distance themselves

Exercise to do:

Exercise

- Consider the degree 1 polynomial kernel $K(x, x') = 1 + xx'$.
- Consider the regression problem with the training set
 - $x_1 = -2$ and $y_1 = 2$,
 - $x_2 = 4$ and $y_2 = 3$
- and a test sample $x^* = 1$.
- Predict its label using the Nearest Neighbour algorithm.

So we have to find: nearest neighbor to 1

- which is either -2 or 4
- and then predict using the label of nearest neighbor

Use formula from previous slide to compute distance

Another exercise

- Consider the string kernel with parameter $c = 2$ (as on slide 20). Use the decay factor $\lambda = 1$. (By default, the string kernel is unnormalized.)
- Consider the regression problem with the training set
 - $x_1 = \text{'cat'}$ and $y_1 = 0$,
 - $x_2 = \text{'car'}$ and $y_2 = 2$
- and the test sample $x^* = \text{'bat'}$.
- Predict its label using the Nearest Neighbour algorithm.

- $c = 2$, length of sub seq
- decay factor $L = 1$, no decay

String kernel is unnormalised by default

- $x_1 = \text{'cat'}$ and $y_1 = 0$
- $x_2 = \text{'car'}$ and $y_2 = 2$

and test $x^* = \text{'bat'}$

Predict using NN algorithm

our own prediction is that bat looks more similar to cat, but we will check later when revising

Kernelizing other prediction algorithms

- The prediction for Ridge Regression (and in particular, Least Squares) can be written in the form involving the samples only via their pairwise dot products (the details not in this module).
- Replacing the dot products $x \cdot x'$ by the kernels $K(x, x')$ gives **Kernel Ridge Regression**.
- Maximum margin classifiers (a linear method discussed in the next chapter) can also be written in this form.
- Therefore, they can also be kernelized (giving support vector machines).
- There has been a kernelization industry in machine learning; all kind of linear algorithms have been kernelized.

Some applications of kernels (1)

- Kernels were a big breakthrough when they appeared in the 1990s (in the form of support vector machines, the topic of the next chapter).
- Historically, the first area where kernel methods were used was handwritten digit recognition; they turned out to be competitive with tangent distance (and neural networks).

Some applications of kernels (2)

- String kernels are used in natural language processing (obviously).
- They are also widely used in bioinformatics.
- For example, string kernels can model, to some degree, the evolutionary process of insertion and deletion of DNA fragments.
- Amazingly, they make it possible to apply linear methods to discrete samples (such as texts, DNA sequences, and proteins).
- They have been used in automated phone helplines (e.g., by AT&T in the USA).

Chapter 8: Neural networks and support vector machines

Vladimir Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS4920/CS5920 Machine Learning
Last edited: September 1, 2020

Neural nets and support vector machines

- Neural nets were used from the dawn of artificial intelligence.
Motivation: this is how we think.
- SVMs were conceived by Vladimir Vapnik as competitor for neural nets (and in his book he explained that neural nets with one hidden layer can be considered to be SVMs). For a while they completely eclipsed neural nets.
- With the rise of deep learning, neural nets are in again.
- The cycle might continue. . . .

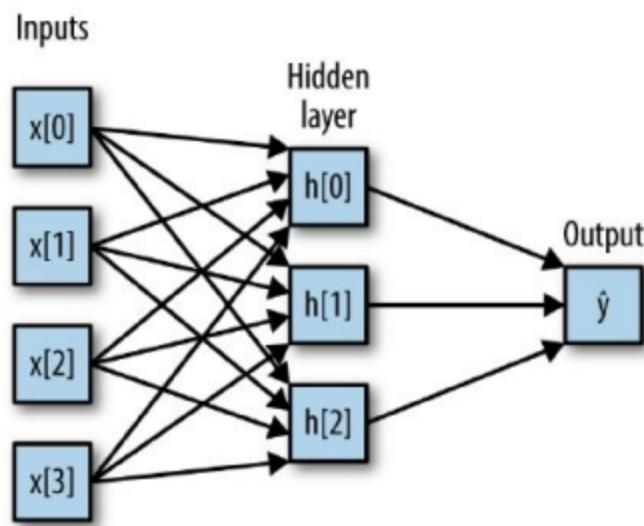
Plan

- 1 Neural networks
- 2 Maximum margin classifiers
- 3 Support vector machines
- 4 SVMs with more than two classes

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

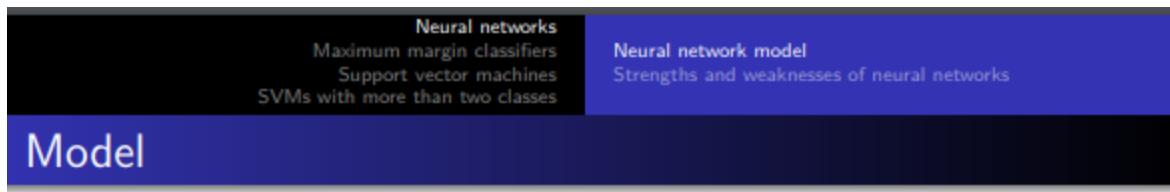
Neural network model
Strengths and weaknesses of neural networks

Neural network with 1 hidden layer



- 4 inputs, $x[0]$ to $x[3]$
- 1 hidden layer, all adjustable
- Output is $y_{\hat{}}$
- The arrows:
 - signify input

Neural nets aren't linear, they have activation functions:



- Given the input features $x[0], \dots, x[p - 1]$, the output label is computed from left to right.
- The full formula (in the case of regression) is:

$$\begin{aligned}
 h[0] &:= \tanh(w[0, 0]x[0] + w[0, 1]x[1] + w[0, 2]x[2] \\
 &\quad + w[0, 3]x[3] + b[0]) \\
 h[1] &:= \tanh(w[1, 0]x[0] + w[1, 1]x[1] + w[1, 2]x[2] \\
 &\quad + w[1, 3]x[3] + b[1]) \\
 h[2] &:= \tanh(w[2, 0]x[0] + w[2, 1]x[1] + w[2, 2]x[2] \\
 &\quad + w[2, 3]x[3] + b[2]) \\
 \hat{y} &:= v[0]h[0] + v[1]h[1] + v[2]h[2] + b.
 \end{aligned}$$

- For $h[0]$
 - we take input $x[0]$ to $x[3]$
 - with weights $w[]$

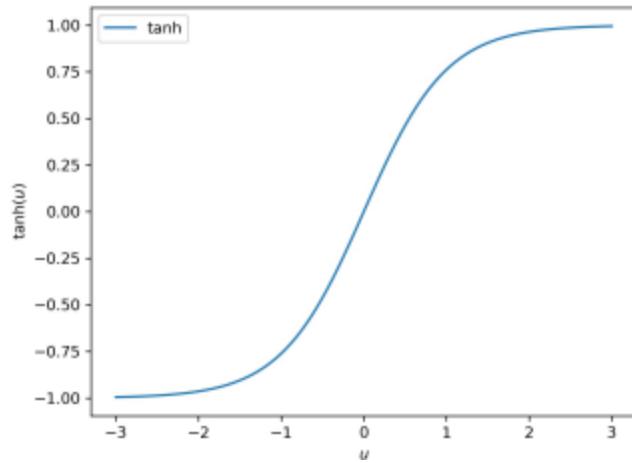
- and some intercept b
- For all these inputs, we apply the **activation function** `tanh()`
 - hyperbolic function

And THEN:

- `y_hat` is just a linear combination of $h[0]$ to $h[3]$

Activation function `np.tanh`

Here `tanh` ([tangens hyperbolicus](#), a function in NumPy) is the function nicely mapping the real line \mathbb{R} to $(-1, 1)$. We can replace it with a different **activation function**.



- Maps real line \mathbb{R} to $(-1,1)$, never achieving them

Plotting in Python (using matplotlib)

The graph on the previous slide was drawn as follows:

```
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
line = np.linspace(-3, 3, 100)
plt.plot(line, np.tanh(line), label="tanh")
plt.legend(loc="best")
plt.xlabel("u")
plt.ylabel("tanh(u)")
```

Here the “Python magic” `%matplotlib notebook` allows you to adjust your picture.

Parameters

- The parameters are w (the weights between the inputs x and the hidden layer h), v (the weights between the hidden layer h and the output \hat{y}), and the bs .
- The parameters v , w , and b are learned from data.
- An important parameter that needs to be set by the user is the number of nodes in the hidden layer.
- It is also possible to add additional hidden layers, as shown on the next slide.

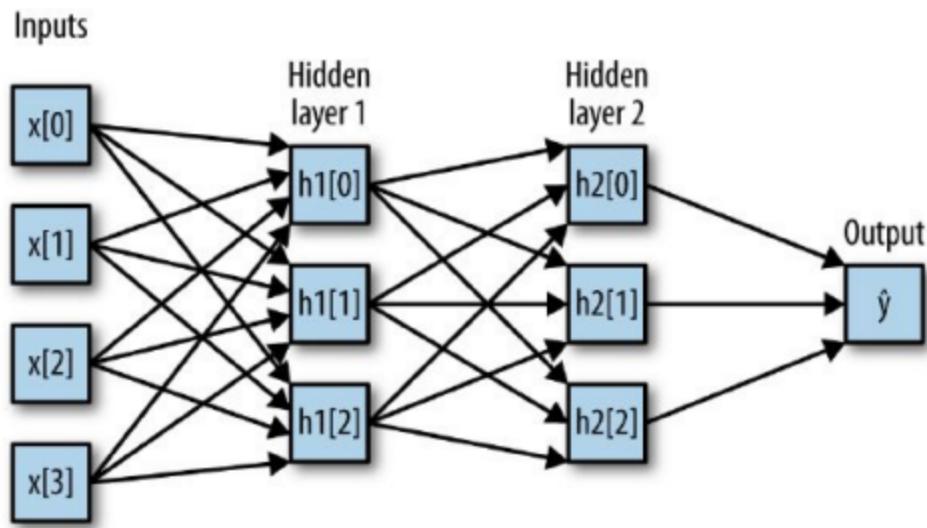
- W

- the weights between inputs x and hidden layer h
- v
 - weights between hidden layer h and output y_{hat}
- parameters are learned from the data
- parameters are just the size of hidden layer
- deep learning just means neural nets with lots of hidden layers

Neural networks
 Maximum margin classifiers
 Support vector machines
 SVMs with more than two classes

Neural network model
 Strengths and weaknesses of neural networks

Neural network with 2 hidden layers



Strengths and weaknesses (1)

- + Neural networks are able to capture information contained in large amounts of data and build incredibly complex models. Given enough computation time, data, and careful tuning of the parameters, they often beat other machine learning algorithms (for both classification and regression).
 - Neural networks—particularly the large and powerful ones—often take a long time to train.
-
- + able to capture lots of information and create complex
 - but need lots of data, computation time and Careful Tuning
 - - large ones take long time to run

cont. from next slide

- require careful preprocessing
- Works Best with “homogenous” data - data that has features with similar meanings
- it is an art to train

Strengths and weaknesses (2)

- They require careful preprocessing of the data (normalization is essential).
- They work best with “homogeneous” data, where all the features have similar meanings.
- Training neural networks is a very difficult art.

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Concept of a hyperplane
Optimal separating hyperplane
Constructing the maximum margin classifier

Plan

- 1 Neural networks
- 2 Maximum margin classifiers
- 3 Support vector machines
- 4 SVMs with more than two classes

Related to SVM Support Vector Machines:

Most simple SVM is Maximum margin classifiers.

Introduction

- One reason why neural networks are so difficult to train is that their performance (measured, e.g., by the RSS in the case of regression) is a complicated function of the parameters, with lots of local minima (and it's very easy to get stuck in one of those).
 - Vapnik considered SVM an improved version of neural networks (with one hidden layer) that does not have multiple local minima.
 - The reason it does not have multiple local minima is that the function it is trying to minimize is convex (details: later).
-
- Neural nets hard to train:
 - their accuracy is a complicated function of the parameters, not linear due to activation function.
 - Vapnik considered SVM an improved version of neural nets, that does not have multiple local minima (with one hidden layer)
 - It is trying to minimise a convex function, we will see later

What is a hyperplane? (1)

In the p -dimensional space \mathbb{R}^p , a **hyperplane** is a flat affine (i.e., not necessarily containing 0) subspace of dimension $p - 1$.

- In two dimensions: a line.
- In three dimensions: a plane.
- In $p > 3$ dimensions, it can be hard to visualize a hyperplane.

A hyperplane is like a plane in a high dimensional space typically.

- **A flat affine**
 - **affine** doesn't have to contain 0, it's not linear, when a space is linear - has to contain 0.
- **subspace of dimension p-1**
 - p features

The samples belong to the space \mathbb{R}^p

So:

- In two dimensions: a line
- In three dimensions: a plane
- in $p > 3$ dimensions: hard to visualise a hyperplane

Lets stick with two dimensions:

What is a hyperplane? (2)

- In two dimensions, a hyperplane is defined by

$$w[0]x[0] + w[1]x[1] + b = 0$$

for parameters $w[0]$, $w[1]$, and b . This means: any x for which the equation holds is a point on the hyperplane.

- This can be easily extended to the p -dimensional setting:

$$w[0]x[0] + w[1]x[1] + \cdots + w[p-1]x[p-1] + b = 0$$

defines a p -dimensional hyperplane. We will write the last equation as

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

and always assume $\mathbf{w} \neq 0$.

- In 2D hyperplane is defined:
 - linear combination of features with coefficient \mathbf{w}
 - and \mathbf{b} , responsible for origin of coordinate system.

So any x that satisfies our equation is on the hyperplane

- Easily extended to p -dimensions

We can just write as $\mathbf{w} \text{ dotproduct}(\mathbf{x}) + \mathbf{b} = 0$

- where $\mathbf{w} \neq 0$

Splitting the space

Now, suppose that x does not satisfy this equation.

- If

$$w \cdot x + b > 0,$$

then this tells us that x lies to one side of the hyperplane.

- On the other hand, if

$$w \cdot x + b < 0,$$

then x lies to the other side of the hyperplane.

We can think of the hyperplane as dividing the p -dimensional space into two halves. One can determine on which side of the hyperplane a point lies by calculating the sign of $w \cdot x + b$.

What if x doesn't satisfy equation:

- If:

$$w \cdot x + b > 0,$$

- Then x lies on one side of hyperplane

- Or If:

$$w \cdot x + b < 0,$$

- Then it lies on other side.

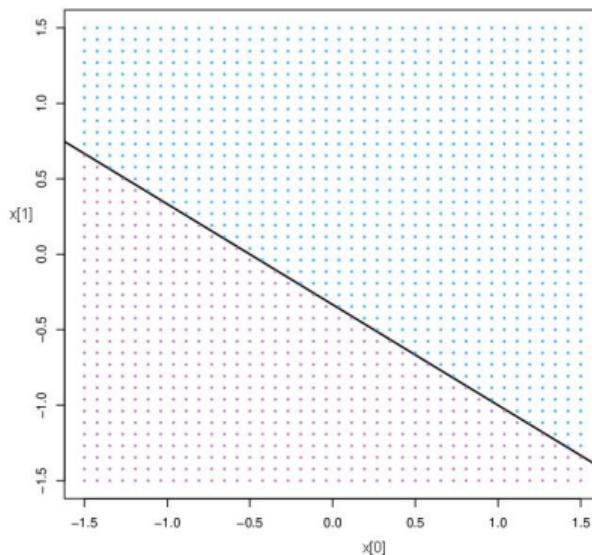
So we can think of hyperplane of splitting p -dimensional space into two halves

We just need to find the sign of \mathbf{w} dot $\mathbf{x} + b$

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Concept of a hyperplane
Optimal separating hyperplane
Constructing the maximum margin classifier

A hyperplane in two-dimensional space



Classification using a separating hyperplane (1)

- Suppose we have a training set that consists of n training samples x_1, \dots, x_n in p -dimensional space, and these samples fall into two classes:

$$y_1, \dots, y_n \in \{-1, 1\},$$

where -1 represents one class and 1 the other class.

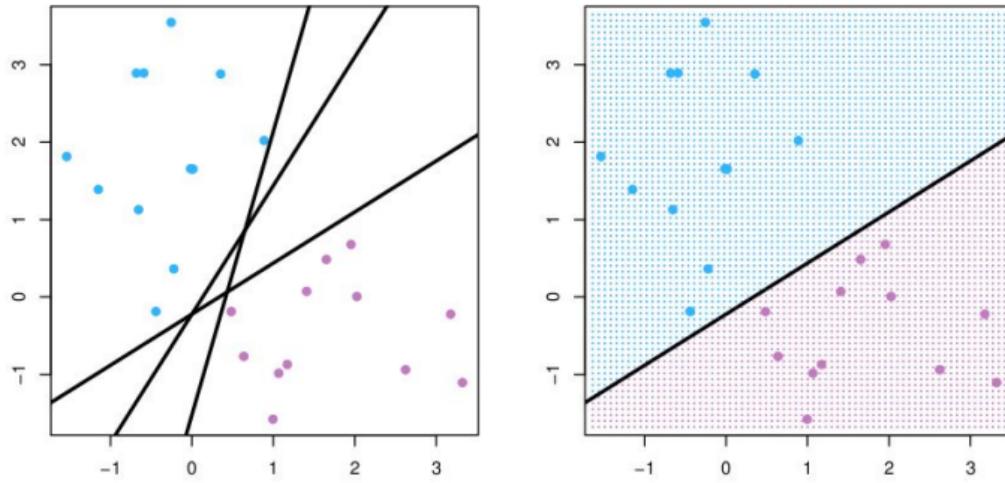
- Suppose we also have a test sample x^* .
- Our goal: develop a classifier based on the training data that will correctly classify the test sample using its features.

- Suppose we have a training set that consists of n training samples
 - x_1, \dots, x_n in p -dimensional space, and these samples fall into two class labels
 - y_1, \dots, y_n element of $\{-1, 1\}$
- Suppose we have test sample x^*
- Our goal: develop a classifier based on the training data that correctly classify test sample using its features

Classification using a separating hyperplane (2)

- Suppose it is possible to construct a hyperplane that separates the training samples perfectly according to their class labels.
 - Examples of three such separating hyperplanes are on the left of the following figure.
 - There are two classes of observations, shown in blue and in purple, each of which has two features.
-
- Suppose it is possible to construct a hyperplane that separates the training samples perfectly according to their class labels.
 - usually, if there is 1, there are infinite.
 - Examples of three such separating hyperplanes are on the left on next slide
 - There are two classes of observations, shown in blue (top left) and in purple(bottom right), each of which has two features

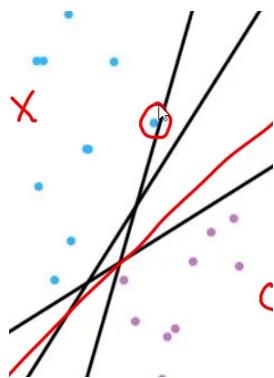
Separating hyperplanes



blue (top left) and in purple (bottom right)

We don't like when our training samples come close to the hyperplanes, like shown here:

- it becomes fragile.



Mathematics of separating by a hyperplane

Let us label the observations from the blue class as $y_i = 1$ and those from the purple class as $y_i = -1$. Then a separating hyperplane has the property that

$$\begin{cases} w \cdot x_i + b > 0 & \text{if } y_i = 1 \\ w \cdot x_i + b < 0 & \text{if } y_i = -1. \end{cases}$$

Equivalently, a separating hyperplane has the property that

$$y_i(w \cdot x_i + b) > 0$$

for all $i = 1, \dots, n$.

So now, how do we express it geometrically?

- blue class: $y_i = 1$
- purple class: $y_i = -1$

Then separating hyperplane has property:

$$\begin{cases} w \cdot x_i + b > 0 & \text{if } y_i = 1 \\ w \cdot x_i + b < 0 & \text{if } y_i = -1. \end{cases}$$

Instead we can say:

$$y_i(w \cdot x_i + b) > 0$$

for all $i = 1, \dots, n$.

- Multiply LHS by y_i (label)
 - if y_i is positive, first equation doesn't change
 - if y_i is negative, second sign becomes this.

$$\begin{cases} w \cdot x_i + b \geq 0 & \text{if } y_i = 1 \\ w \cdot x_i + b \leq 0 & \text{if } y_i = -1. \end{cases}$$

Its sign changes

Neural networks Maximum margin classifiers Support vector machines SVMs with more than two classes	Concept of a hyperplane Optimal separating hyperplane Constructing the maximum margin classifier
Classifier from a separating hyperplane	

If a separating hyperplane exists, we can use it as a classifier: a test sample is assigned a class depending on which side of the hyperplane it is located. Shown on the right of the previous figure. We classify the test sample x^* based on the sign of the **scoring function** (also called **decision function** in scikit-learn)

$$f(x^*) = w \cdot x^* + b.$$

- If $f(x^*)$ is positive, then we assign the test sample to class 1.
- If $f(x^*)$ is negative, then we assign it to class -1 .

So once we find a (good) separating hyperplane:

- Classifier for: a test sample, depending on which side it is located.
- We can classify x^* using this scoring function (decision function in scikit-learn):

$$f(x^*) = \underline{w} \cdot x^* + b.$$

- If $f(x^*)$ is positive, then assign class 1
- If $f(x^*)$ is negative, then assign class 0

Neural networks
Maximum margin classifiers
 Support vector machines
 SVMs with more than two classes

Concept of a hyperplane
 Optimal separating hyperplane
 Constructing the maximum margin classifier

Exercise

- Suppose we have a linear scoring function with parameters $b = 1$ and $w = (1, 0, -3)$ (in the notation of the previous slide).
- The test sample is $x^* = (-1, 0, 1)$.
- Calculate the predicted label for x^* .

Suppose parameters: $b = 1$, $w = (1, 0, -3)$

test sample $x^* = (-1, 0, 1)$

What is predicted label for x^*

$$w \cdot x^* + b = -1 + 0 + (-3) + 1 = -3$$

so label is -1.

The slide has a dark blue header bar with the word 'Confidence' in white. Above the header, there is a horizontal bar divided into two sections: a black section on the left containing text and a blue section on the right containing text. The black section contains the following text:
Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

The blue section contains the following text:
Concept of a hyperplane
Optimal separating hyperplane
Constructing the maximum margin classifier

Moreover, we can also make use of the magnitude of $f(x^*)$.

- If $f(x^*)$ is far from zero, then this means that x^* lies far from the hyperplane, and so we can be confident about our class assignment for x^* .
- On the other hand, if $f(x^*)$ is close to zero, then x^* is located near the hyperplane, and so we are less certain about the class assignment for x^* .

So far, we talked about the way scoring function determines the prediction, positive or negative.

But,

We can make use of the magnitude of $f(x^*)$:

- If $f(x^*)$ far from zero, then x^* lies far from hyperplane, so we can be confident
- If $f(x^*)$ close to zero, then x^* is near hyperplane, less certainty

The maximum margin classifier

- Notice: if our data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes.
- In order to construct a classifier based upon a separating hyperplane, we must have a reasonable way to decide which of the infinitely many possible separating hyperplanes to use.
- A natural choice is the **maximum margin hyperplane** (also known as the **optimal separating hyperplane**), which is the separating hyperplane that is farthest from the training samples.

The maximum margin classifier

- If our data can be perfectly separated using a hyperplane, then there will be an infinite number of such hyperplanes
- To create a classifier: we must have reasonable way to decide which of the hyperplanes to use
- A Natural Choice:
 - the **maximum margin hyperplane** (optimal separating hyperplane), which is just the plane fartherst from training samples

Now formally:

The margin (1)

- We can compute the (perpendicular) distance from each training sample to a given separating hyperplane; the smallest such distance is known as the **margin**.
- The maximum margin hyperplane is the separating hyperplane for which the margin is largest.
- We can then classify a test sample based on which side of the maximum margin hyperplane it lies.
- This is known as the **maximum margin classifier**.

The margin

- We can compute (perpendicular) distance from each training sample to a given separating hyperplane: **the smallest such distance is the margin**
 - from all distances, smallest is margin
- The maximum margin hyperplane is the separating hyperplane for which the margin is largest.
 - **Maximum margin classifier:**
 - We can then classify a test sample based on which side of the maximum margin hyperplane it lies.
 - This is known as **maximum margin classifier**

The margin (2)

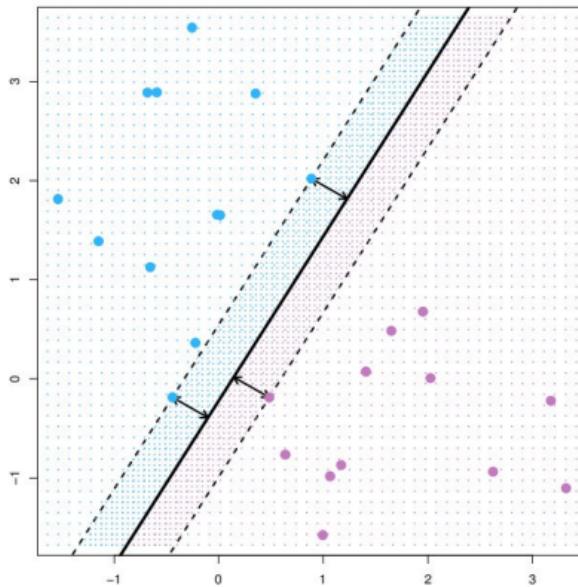
- We hope that a classifier that has a large margin on the training data will also have a large margin on the test data, and hence will classify the test samples correctly.
- Although the maximum margin classifier is often successful, it can also lead to overfitting when p is large.
- If $w = (w[0], \dots, w[p - 1])$ and b are the parameters of the maximum margin hyperplane, then the maximum margin classifier classifies the test sample x^* based on the sign of

$$f(x^*) = w \cdot x^* + b.$$

- Our hope: classifier that has a large margin on training data will also have large margin on test data, and hence will classify the test samples correctly.
- Often it is successful, but can lead to overfitting when p is large
- If $w = (w[0], \dots, w[p-1])$ and b are the parameters of maximum margin hyperplane:
 - then the maximum margin classifier classifies the test sample x^* based on sign of:

$$f(x^*) = w \cdot x^* + b.$$

The maximum margin hyperplane on the data set of the previous figure



These black arrows are the margins, they're equal

We want to separate with a thick slab, and the middle line is the separating hyperplane. So we try make slab wide as possible so all points are on the outside or border with it. Widen until it hits some vector.

These vectors are called **support vectors**, they support the slab

Geometry of the maximum margin hyperplane

- The maximum margin hyperplane represents the mid-line of the widest “slab” that we can insert between the two classes.
- In the previous figure, three training samples are equidistant from the maximum margin hyperplane and lie along the dashed lines indicating the width of the margin.
- These three samples are known as **support vectors**.

This is written by me above.

Support vectors

- They are called “support vectors” since they are vectors in the p -dimensional space \mathbb{R}^p ($p = 2$ in the figure) and they “support” the maximum margin hyperplane: if these points were moved slightly then the maximum margin hyperplane would move as well.
- The maximum margin hyperplane depends directly on the support vectors, but not on the other samples: a movement to any of the other samples would not affect the separating hyperplane, provided that the sample’s movement does not cause it to cross the boundary set by the margin.

- They are called “**support vectors**” as they are vectors in p-dimensional space \mathbb{R}^p ($p=2$ above) and they “support” the maximum margin hyperplane:
 - if these points were moved slightly then the maximum margin hyperplane would move as well.
- The maximum margin hyperplane depends directly on the support vectors, but not on the other samples:
 - A movement to any of the other samples would not affect the separating hyperplane, provided that the sample’s movement does not cause it to cross the boundary set by margin

Neural networks
Maximum margin classifiers
 Support vector machines
 SVMs with more than two classes

Concept of a hyperplane
 Optimal separating hyperplane
 Constructing the maximum margin classifier

Construction of the maximum margin classifier (1)

- How do we construct the maximum margin hyperplane based on a set of n training samples $x_1, \dots, x_n \in \mathbb{R}^p$ and associated class labels $y_1, \dots, y_n \in \{-1, 1\}$?
- The maximum margin hyperplane (determined by w and b) is the solution to the optimization problem

$$\|w\|^2 \rightarrow \min$$

subject to

$$y_i(w \cdot x_i + b) \geq 1, \quad i = 1, \dots, n.$$

How can we make it? How to express it algebraically?

- Based on set of n training samples x_1, \dots, x_n element of \mathbb{R}^p
 - and associated class labels y_1, \dots, y_n element of $\{1, -1\}$

- The maximum margin hyperplane (determined by w and b) is the solution to optimisation problem:

$$\|w\|^2 \rightarrow \min$$

minmise the distance of w

Subject to:

$$y_i (w \cdot x_i + b) \geq 1, \quad i = 1, \dots, n.$$

$|w \cdot x + b| \geq 1$

And this is equation of our slab:

$$|w \cdot x + b| \leq 1$$

slab

Construction of the maximum margin classifier (2)

- The constraint

$$y_i (w \cdot x_i + b) \geq 1$$

guarantees that each sample will be on the correct side of the hyperplane with some cushion.

- The objective

$$\|w\|^2 \rightarrow \min$$

means that we are minimizing the Euclidean length of w .

Construction of the maximum margin classifier (3)

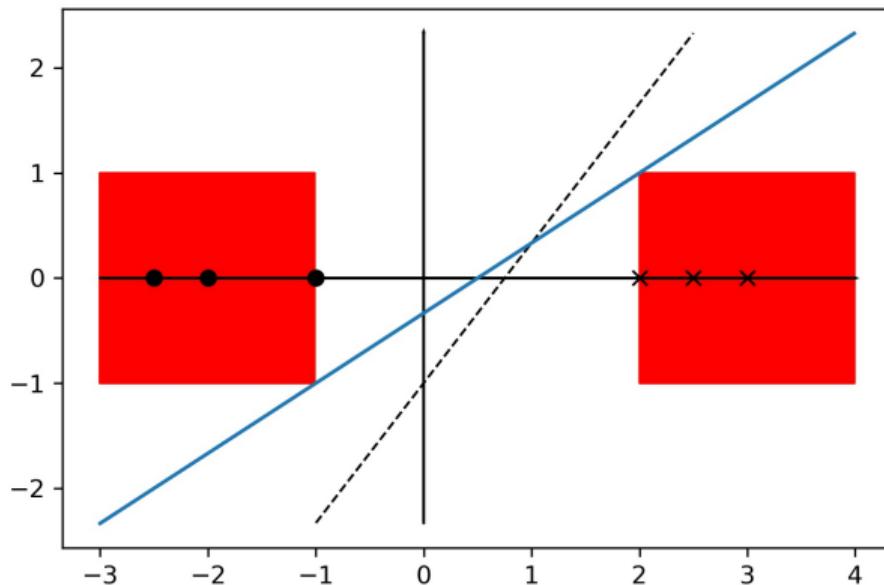
- In fact the objective $\|w\|^2 \rightarrow \min$ means that we are maximizing the margin.
- Why? Think of the 1D case, where we have only one feature, as on the next slide (where the negative and positive samples are shown as noughts and crosses).
- The optimization problem can be solved efficiently (it's nice and convex), but the details of this are outside the scope of this course.

So now lets see what it looks like geometrically.

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Concept of a hyperplane
Optimal separating hyperplane
Constructing the maximum margin classifier

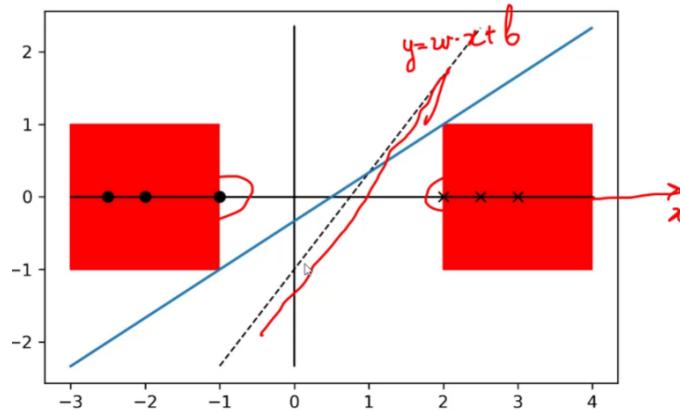
Optimal w in 1D (slope of the blue line)



Video: slides 24-34, minute - 10:46

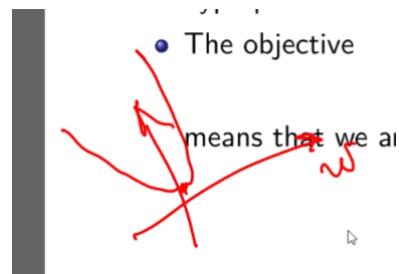
Crosses are positive samples

Noughts are negative samples



In neural nets, the functions we are minimising have lots of local minima and are thus more awkward.

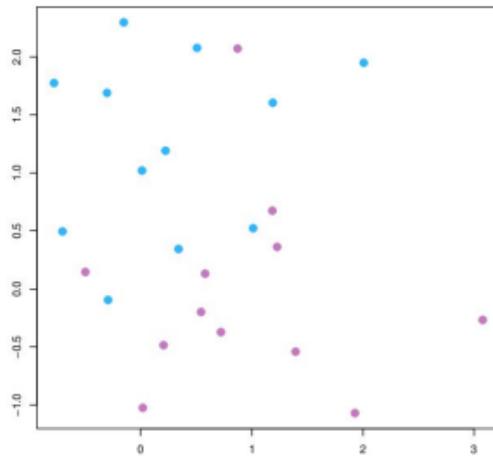
Lets visualise minimising w:



It is just a parabola, nice easy minimum

The non-separable case (1)

An example where no separating hyperplane exists (our optimization problem has no solution):



Now lets consider a non separable case:

- Here blue and purple points cant be separated.

The non-separable case (2)

- We will extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called “soft margin”.
- The extension is the “soft margin classifier”.
- But first a few exercises.

Later in chapter we will be interested in case when our hyperplane is allowed to almost separate the two classes - A soft margin, no hard separation

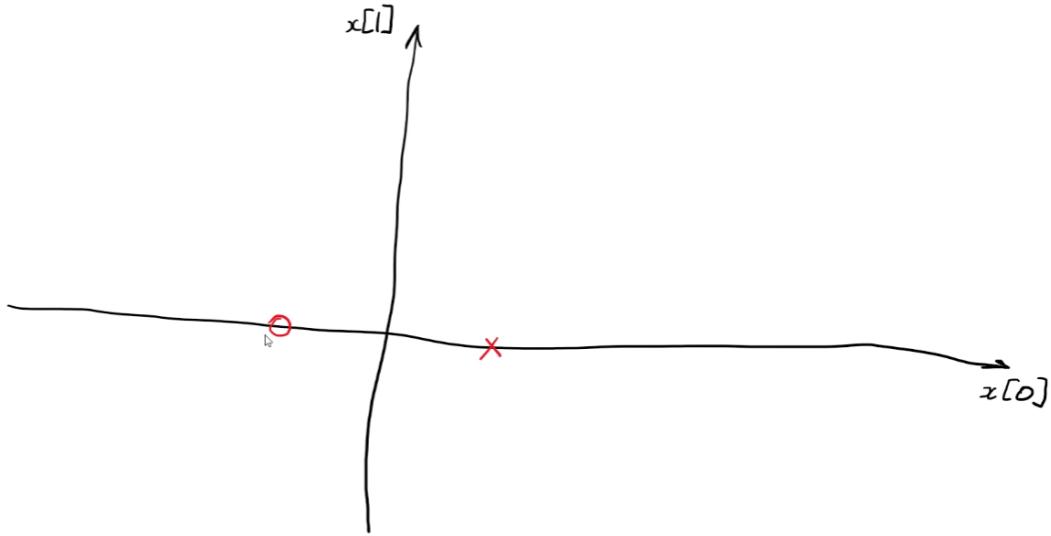
Exercises (1)

The geometry behind the maximum margin classifier is quite intuitive.

- Suppose the training set is $\{((-1, 0), -1), ((1, 0), 1)\}$ (as always, it would be better to talk about a training sequence or a training bag). What is the optimal separating hyperplane? (Draw it and write an equation for it.)
- Suppose the training samples are:
 - positive: $(0, 1)$ and $(1, -1)$
 - negative: $(1, 0)$

What is the optimal separating hyperplane?

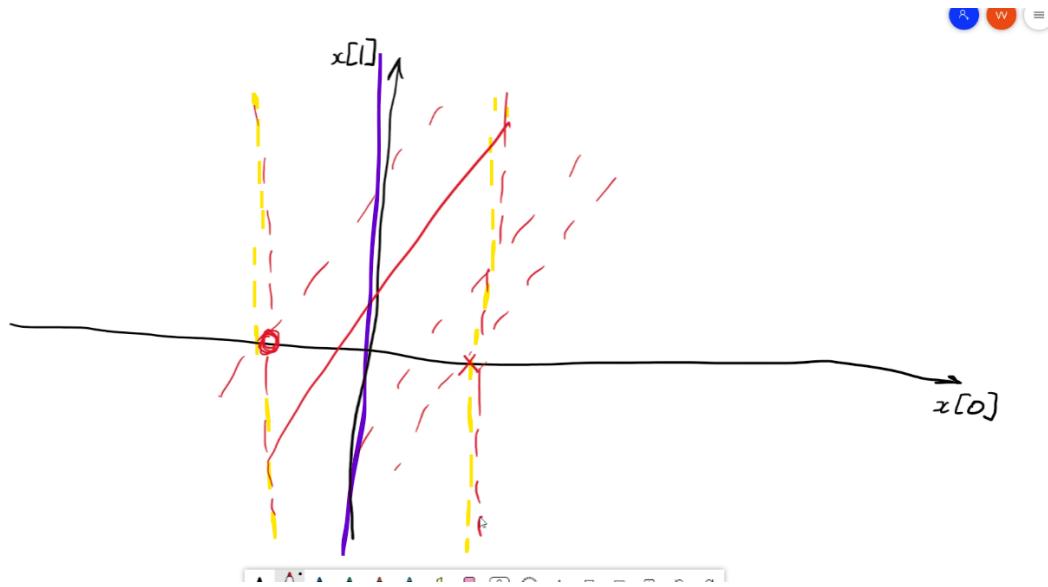
First Bullet Point



-1,0 is negative and 1,0 is positive

We want to create the widest possible slab between these points

Our final widest slab with a separating hyperplane, will just be a vertical line going down $x[1]$ (so the line has equation $x[0] = 0$) and the slabs on both points



So above, is our optimal separating hyperplane, with equation $x[0] = 0$

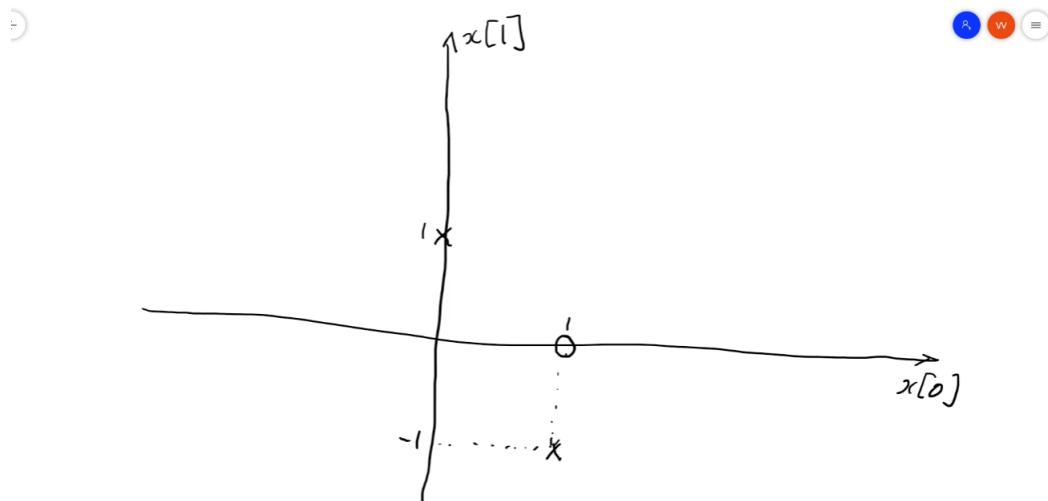
Exercises (1)

The geometry behind the maximum margin classifier is quite intuitive.

- Suppose the training set is $\{((-1, 0), -1), ((1, 0), 1)\}$ (as always, it would be better to talk about a training sequence or a training bag). What is the optimal separating hyperplane? (Draw it and write an equation for it.)
- Suppose the training samples are:
 - positive: $(0, 1)$ and $(1, -1)$
 - negative: $(1, 0)$

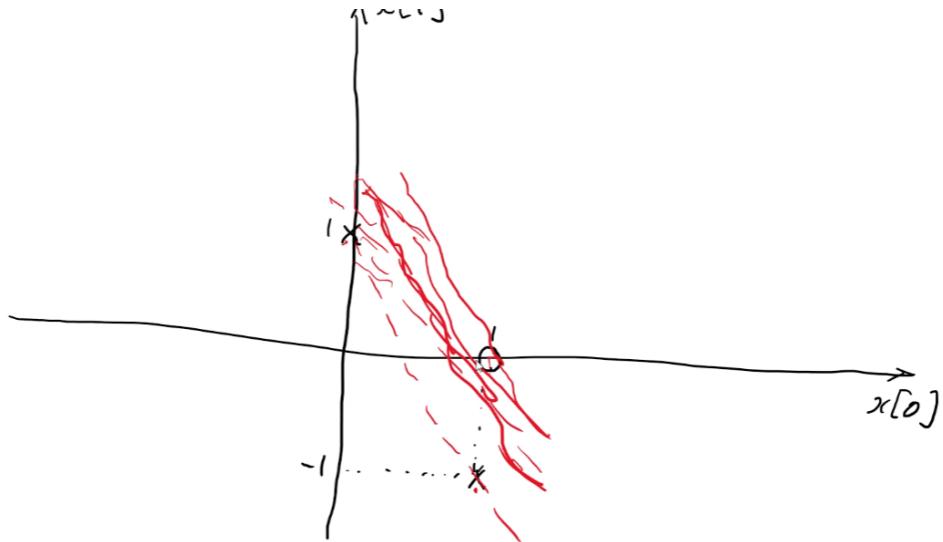
What is the optimal separating hyperplane?

Second Bullet Point



crosses positive, norts negative

7:40 in video 35-38



Our hyperplane line will have a slab line going through the two positive samples so we know that the hyperplane line will have the same gradient as it:

A gradient of -2 ($w = -2$) because goes down for 2 for every 1 in x axis.

$$x[1] = -2 [x_0] + \text{const.}$$

HE SAID HE will go through in face to face lecture SO WATCH THAT

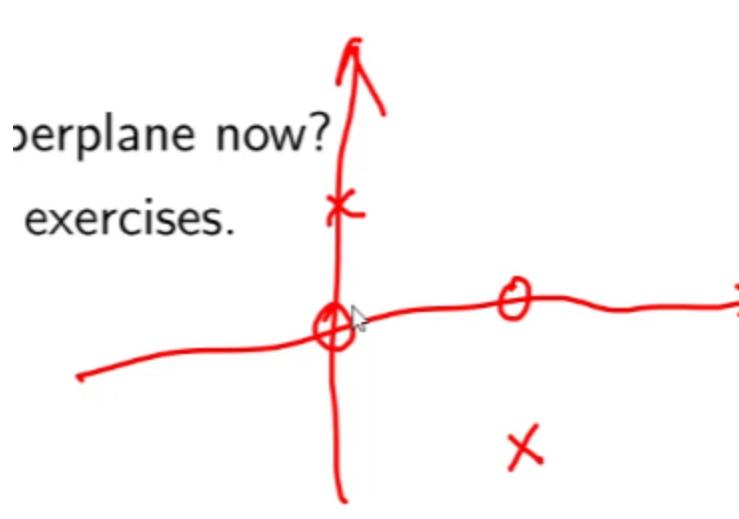
Lecture on 18/11/2022

Exercises (2)

- Add another negative observation to the training set, so that the training samples are:
 - positive: $(0, 1)$ and $(1, -1)$
 - negative: $(1, 0)$ and $(0, 0)$

What is the optimal separating hyperplane now?

Identify the support vectors in all these exercises.



It is easy to see that this is non separable

So answer, Doesn't Exist.

No support vectors

Neural networks	Soft margin classifiers
Maximum margin classifiers	Support vector machines with kernels
Support vector machines	Uses, strengths, and weaknesses
SVMs with more than two classes	Strengths and weaknesses

Plan

- 1 Neural networks
- 2 Maximum margin classifiers
- 3 Support vector machines
- 4 SVMs with more than two classes

Introduction

- Samples that belong to two classes are not necessarily separable by a hyperplane.
- And even if a separating hyperplane does exist, a classifier based on a separating hyperplane might not be desirable!
- A classifier based on a separating hyperplane will necessarily perfectly classify all of the training samples, which can lead to excessive sensitivity to individual observations.
- Example: next figure.

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Soft margin classifiers
Support vector machines with kernels
Uses, strengths, and weaknesses
Strengths and weaknesses

Plan

- 1 Neural networks
- 2 Maximum margin classifiers
- 3 **Support vector machines**
- 4 SVMs with more than two classes

Support Vector Machines are the simple development of maximum margin classifier, essentially same thing but in a feature space.

How to relax maximum margin classifiers.

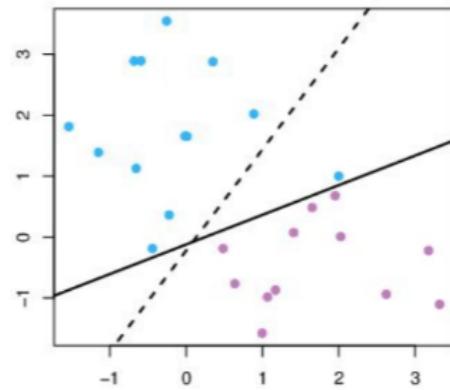
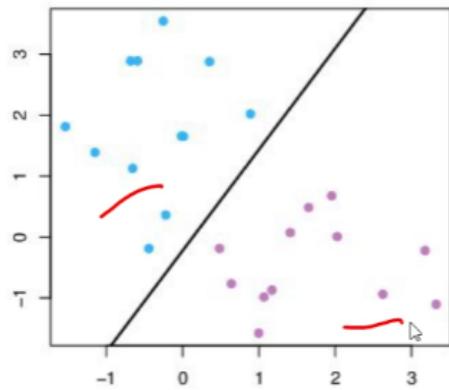
Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Soft margin classifiers
Support vector machines with kernels
Uses, strengths, and weaknesses
Strengths and weaknesses

Introduction

- Samples that belong to two classes are not necessarily separable by a hyperplane.
 - And even if a separating hyperplane does exist, a classifier based on a separating hyperplane might not be desirable!
 - A classifier based on a separating hyperplane will necessarily perfectly classify all of the training samples, which can lead to excessive sensitivity to individual observations.
 - Example: next figure.
-
- If interested in two classes, they aren't necessarily separable by hyperplane
 - Even if it exists, we might not want it
 - The idea is that we don't want excessive sensitivity to individual observations

A dramatic change in the maximal separating hyperplane



- On the left, nicely separated
 - But on the right, one single blue point changes the hyperplane completely
- So idea is to sacrifice some points.

A problem with the maximum margin classifier

- The addition of a single observation in the right-hand panel leads to a dramatic change in the maximum margin hyperplane.
- The resulting maximum margin hyperplane has only a tiny margin and so is not satisfactory.
 - As discussed earlier, the distance of a sample from the hyperplane measures our confidence that the sample was correctly classified.
 - The fact that the maximum margin hyperplane is extremely sensitive to a change in a single observation suggests overfitting the training data.

Just repeated from above.

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Soft margin classifiers
Support vector machines with kernels
Uses, strengths, and weaknesses
Strengths and weaknesses

Soft margin classifier: idea (1)

Therefore, we consider a classifier based on a hyperplane that does not perfectly separate the two classes, in the interest of:

- greater robustness to individual observations, and
- better classification of **most** of the training samples.

It could be worthwhile to misclassify a few training samples in order to do a better job in classifying the remaining samples.

Soft margin classifier: Idea

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Soft margin classifiers
Support vector machines with kernels
Uses, strengths, and weaknesses
Strengths and weaknesses

Soft margin classifier: idea (2)

- It allows some training samples to be on the wrong side of the margin, or even the wrong side of the hyperplane.
- The margin is **soft** because it can be violated by some of the training observations.
- Samples on the wrong side of the hyperplane are inevitable when there is no separating hyperplane.

Let the margin be soft, some samples can even be miss labeled.

Neural networks
Maximum margin classifiers
Support vector machines
SVMs with more than two classes

Soft margin classifiers
Support vector machines with kernels
Uses, strengths, and weaknesses
Strengths and weaknesses

Details of the soft margin classifier (1)

- The soft margin classifies a test sample depending on which side of a hyperplane it lies.
- The hyperplane is chosen to correctly separate most of the training samples into the two classes, but may misclassify a few samples.
- But we still use soft margin as a classifier like before
- We just exclude some samples

Details of the soft margin classifier (2)

It is the solution to the optimization problem

$$\|w\|^2 + C \sum_{i=1}^n \zeta_i \rightarrow \min$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, \quad i = 1, \dots, n,$$

where C is a positive parameter (and the variables are $b, w[0], \dots, w[p-1], \zeta_1, \dots, \zeta_n$).

Details of soft margin classifier

$$\|w\|^2 + C \sum_{i=1}^n \zeta_i \rightarrow \min$$

$$y_i(w \cdot x_i + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, \quad i = 1, \dots, n,$$

Before we didn't have the Zetas.

- Here we introduce so called “slack variables”
- The Idea for the slack variables:

- positive
- they relax our condition of separation.

$$y_i (w \cdot x_i + b) \geq 1 - \zeta_i,$$

$\vdash \succ \circ \quad i = 1$

- Instead of saying the LHS has to be ≥ 1
 - Which means its outside our separating slab
- We can - Zeta_i
 - in particular Zeta_i can be more than 1.
 - If more than one, we allow a wrong classification.
- If $C \rightarrow \text{Infinity}$
 - Zeta is small (no slack)
- If $C < \text{Infinity}$
 - We are using slack variables

Details of the soft margin classifier (3)

- ζ_1, \dots, ζ_n are **slack variables**; they allow individual training samples to be on the wrong side of the margin or even hyperplane.
- Once we have solved the optimization problem, we classify a test sample x^* as before, by simply determining on which side of the hyperplane it lies, i.e., based on the sign of

$$f(x^*) = w \cdot x^* + b.$$

The tuning parameter C

- We have two conflicting goals: to achieve a wide margin and to classify our training samples well.
- C reflects the importance we attach to the second goal, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate.
- If $C = \infty$ we do not tolerate any violations to the margin, and we must have $\zeta_1 = \dots = \zeta_n = 0$.
 - Our optimization problem now reduces to the old optimal separating hyperplane optimization problem.
 - An optimal separating hyperplane exists only if the two classes are separable.

Support vectors

- It turns out that only samples that either lie on the margin or violate the margin will affect the hyperplane.
- A sample that lies strictly on the correct side of the margin does not affect the soft margin classifier.
- Changing the position of that sample would not change the classifier at all, provided that it remains on the correct side of the margin.
- Samples that lie directly on the margin, or on the wrong side of the margin for their class, are known as **support vectors**.
- Support vectors do affect the soft margin classifier.