Machine Learning and Bayesian Inference

Dr Sean Holden

Computer Laboratory, Room FC06

Telephone extension 63725

Email: sbh11@cl.cam.ac.uk

www.cl.cam.ac.uk/~sbh11/

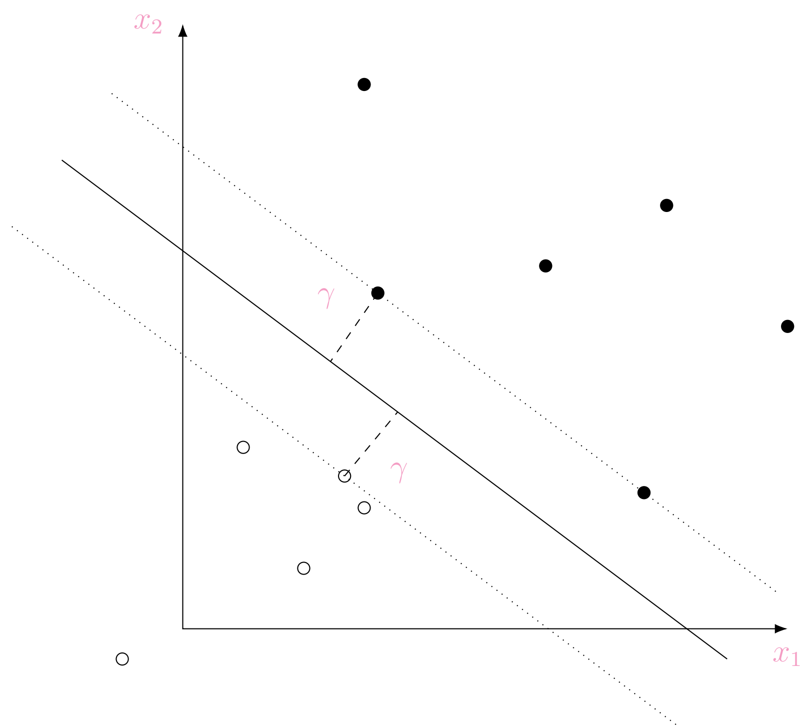
Part II

Support vector machines

General methodology

The maximum margin classifier

Suggestion: why not drop all this probability nonsense and just do this:



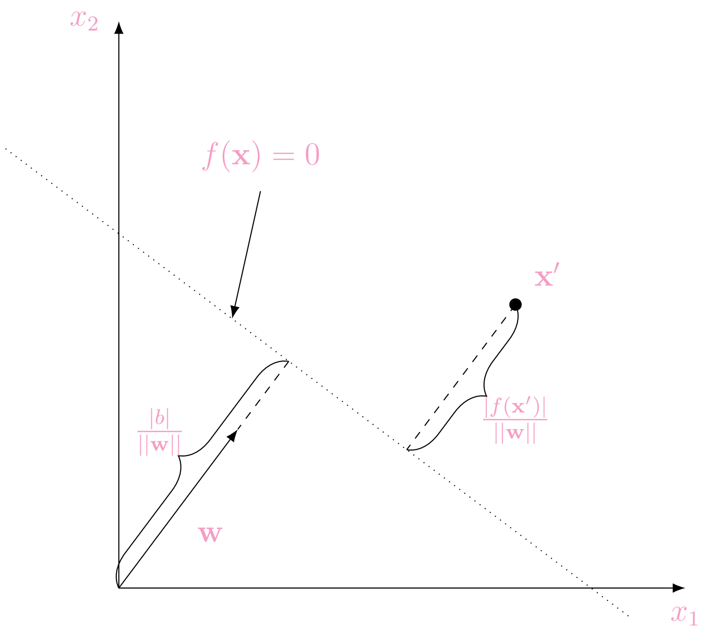
Draw the boundary *as far away from the examples as possible*.

The distance γ is the *margin*, and this is the *maximum margin classifier*.

The maximum margin classifier

If you completed the *exercises for AI I* then you'll know that linear classifiers have a very simple geometry. For

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



For $\mathbf{x}'$ on one side of the line $f(\mathbf{x}) = 0$ we have $f(\mathbf{x}') > 0$ and on the other side $f(\mathbf{x}') < 0$.

Problems:

- Given the usual training data $\mathbf{s}$, can we now find a *training algorithm* for obtaining the weights?
- What happens when the data is not *linearly separable*?

To derive the necessary training algorithm we need to know something about *constrained optimization*.

We can address the second issue with a simple modification. This leads to the *Support Vector Machine (SVM)*.

Despite being decidedly *"non-Bayesian"* the SVM is currently a *gold-standard*:

Do we need hundreds of classifiers to solve real world classification problems, Fernández-Delgardo at al., Journal of Machine Learning Research 2014.

Constrained optimization

You are familiar with *maximizing* and *minimizing* a function $f(\mathbf{x})$. This is *unconstrained optimization*.

We want to extend this:

1. Minimize a function $f(\mathbf{x})$ with the constraint that $g(\mathbf{x}) = 0$.
2. Minimize a function $f(\mathbf{x})$ with the constraints that $g(\mathbf{x}) = 0$ and $h(\mathbf{x}) \geq 0$.

Ultimately we will need to be able to solve problems of the form: find $\mathbf{x}_{\text{opt}}$ such that

$$\mathbf{x}_{\text{opt}} = \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x})$$

under the constraints

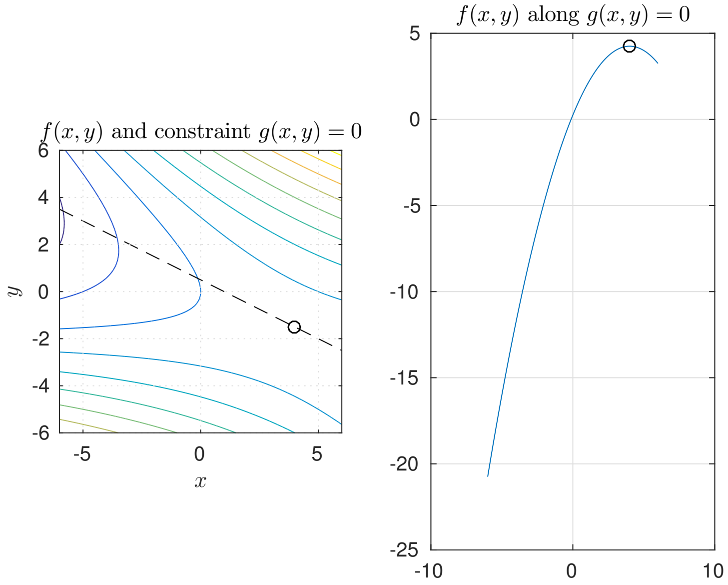
$$g_i(\mathbf{x}) = 0 \text{ for } i = 1, 2, \ldots, n$$

and

$$h_j(\mathbf{x}) \geq 0 \text{ for } j = 1, 2, \ldots, m.$$

Constrained optimization

For example:



Minimize the function

$$f(x,y) = -\left(2x + y^2 + xy\right)$$

subject to the constraint

$$g(x,y) = x + 2y - 1 = 0.$$

Step 1: introduce the *Lagrange multiplier* λ and form the *Langrangian*

$$L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

Necessary condition: it can be shown that if (x', y') is a solution then $\exists \lambda'$ such that

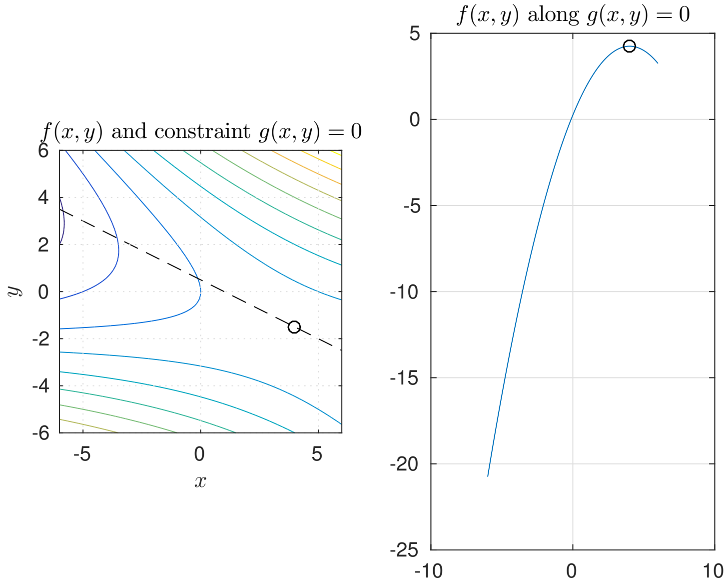
$$\frac{\partial L(x', y', \lambda')}{\partial x} = 0 \qquad \frac{\partial L(x', y', \lambda')}{\partial y} = 0$$

So for our example we need

$$\begin{aligned} 2 + y + \lambda &= 0 \\ 2y + x + 2\lambda &= 0 \\ x + 2y - 1 &= 0 \end{aligned}$$

where the last is just the constraint.

Step 2: solving these equations tells us that the solution is at:



$$(x, y) = (4, -\frac{3}{2})$$

With multiple constraints we follow the same approach, with a *Lagrange multiplier for each constraint*.

How about the full problem? Find

$$\mathbf{x}_{\text{opt}} = \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x}) \text{ such that } g_i(\mathbf{x}) = 0 \text{ for } i = 1, 2, \ldots, n$$
$$h_j(\mathbf{x}) \geq 0 \text{ for } j = 1, 2, \ldots, m$$

The Lagrangian is now

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\mathbf{x}) - \sum_{i=1}^{n} \lambda_i g_i(\mathbf{x}) - \sum_{j=1}^{m} \alpha_j h_j(\mathbf{x})$$

and the relevant necessary conditions are more numerous.

The necessary conditions now require that when $\mathbf{x}'$ is a solution $\exists \boldsymbol{\lambda}', \boldsymbol{\alpha}'$ such that

1. $$\frac{\partial L(\mathbf{x}', \boldsymbol{\lambda}', \boldsymbol{\alpha}')}{\partial \mathbf{x}} = 0.$$
2. The equality and inequality constraints are satisfied at $\mathbf{x}'$.
3. $\boldsymbol{\alpha}' \geq \mathbf{0}$.
4. $\alpha'_j h_j(\mathbf{x}') = 0$ for $j = 1, \ldots, m$.

These are called the *Karush-Kuhn-Tucker (KKT) conditions*.

The *KKT conditions* tell us some important things about the solution.

We will only need to address this problem when the constraints are *all inequalities*.

What we've seem so far is called the *primal problem*.

There is also a *dual* version of the problem. Simplifying a little by dropping the equality constraints.

1. The *dual objective function* is

$$\tilde{L}(\boldsymbol{\alpha}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}).$$

2. The *dual optimization problem* is

$$\max_{\boldsymbol{\alpha}} \tilde{L}(\alpha) \text{ such that } \boldsymbol{\alpha} \geq \mathbf{0}.$$

Sometimes it is *easier to work by solving the dual problem* and this allows us to obtain actual learning algorithms.

We won't be looking in detail at methods for solving such problems, only the *minimum needed to see how SVMs work*.

For the full story see *Numerical Optimization*, Jorge Nocedal and Stephen J. Wright, Second Edition, Springer 2006.

The maximum margin classifier

It turns out that with SVMs we get particular benefits when using the *kernel trick*.

So we work, as before, in the *extended space*, but now with:

$$f_{\mathbf{w},w_0}(\mathbf{x}) = w_0 + \mathbf{w}^T \mathbf{\Phi}(\mathbf{x})$$
$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(f_{\mathbf{w},w_0}(\mathbf{x})\right)$$

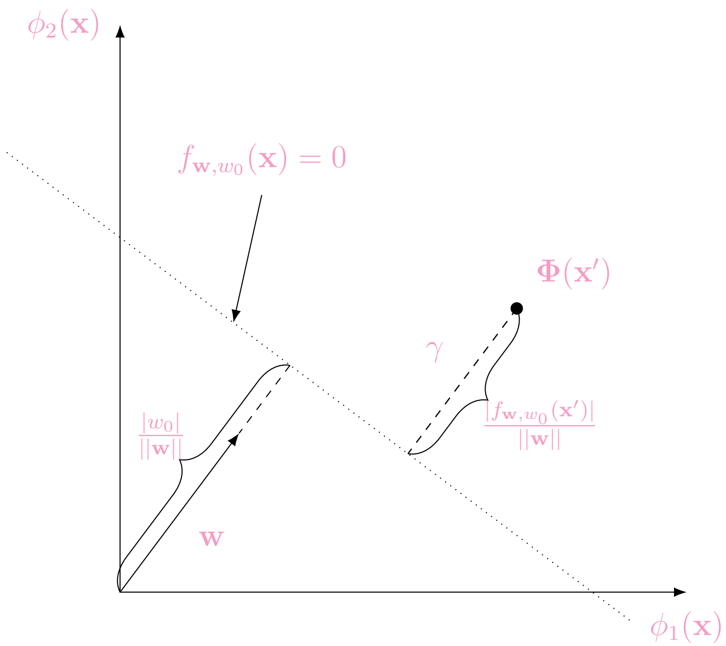
where

$$\text{sgn}(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Note the following:

1. Things are easier for SVMs if we use labels $\{+1, -1\}$ for the two classes. (Previously we used $\{0, 1\}$.)
2. It also turns out to be easier if we keep w_0 separate rather than rolling it into $\mathbf{w}$.
3. We now classify using a "hard" threshold sgn, rather than the "soft" threshold σ.

Consider the geometry again. *Step 1:*



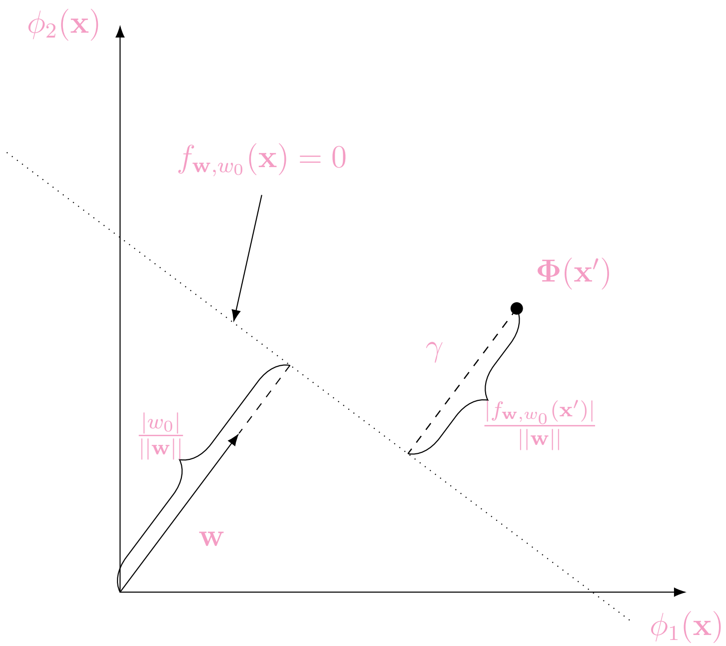
1. We're classifying using the sign of the function

$$f_{\mathbf{w},w_0}(\mathbf{x}) = w_0 + \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}).$$

2. The distance from any point $\mathbf{\Phi}(\mathbf{x}')$ in the extended space to the line is

$$\frac{|f_{\mathbf{w},w_0}(\mathbf{x}')|}{||\mathbf{w}||}.$$

Step 2:



- But we also want the examples to fall on the correct *side* of the line according to their *label*.
- *Noting that for any labelled example* $(\mathbf{x}_i, y_i)$ *the quantity* $y_i f_{\mathbf{w},w_0}(\mathbf{x}_i)$ *will be positive* if the resulting classification is correct…
- …the aim is to solve:

$$(\mathbf{w}, w_o) = \underset{\mathbf{w},w_0}{\text{argmax}} \left[\min_i \frac{y_i f_{\mathbf{w},w_0}(\mathbf{x}_i)}{||\mathbf{w}||} \right].$$

The maximum margin classifier

YUK!!!

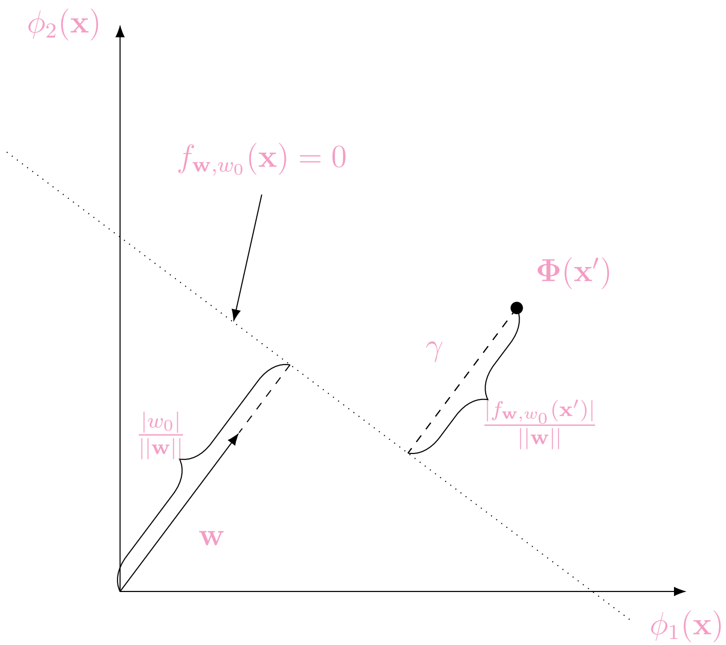
(With bells on. . .)

The maximum margin classifier

Solution, version 1: convert to a *constrained optimization*. For *any* $c \in \mathbb{R}$

$$
\begin{aligned}
f_{\mathbf{w},w_0}(\mathbf{x}) = 0 &\iff w_0 + \mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x}) = 0 \\
&\iff cw_0 + c\mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x}) = 0.
\end{aligned}
$$

That means you can fix $||\mathbf{w}||$ to be *anything you like*! (Actually, fix $||\mathbf{w}||^2$ to avoid a square root.)



Version 1:

$$
(\mathbf{w}, w_o, \gamma) = \underset{\mathbf{w},w_0,\gamma}{\operatorname{argmax}}\, \gamma
$$

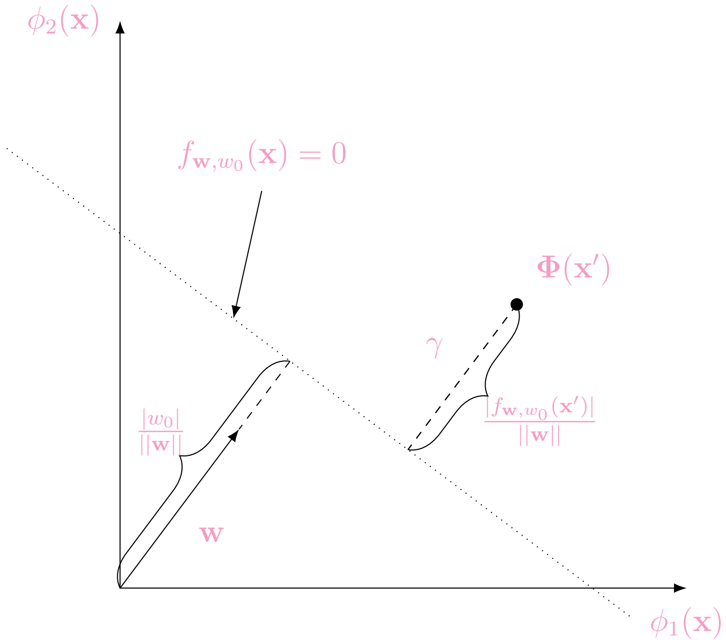
subject to the constraints

$$
y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq \gamma, i = 1, 2, \ldots, m
$$

$$
||\mathbf{w}||^2 = 1.
$$

Solution, version 2: still, convert to a *constrained optimization*, but instead of fixing $||\mathbf{w}||$:

Fix $\min\{y_i f_{\mathbf{w},w_0}(\mathbf{x}_i)\}$ to be *anything you like*!



Version 2:

$$(\mathbf{w}, w_o) = \underset{\mathbf{w},w_0}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2$$

subject to the constraints

$$y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq 1, i = 1, 2, \ldots, m.$$

(This works because maximizing γ now corresponds to *minimizing* $||\mathbf{w}||$.)

The maximum margin classifier

We'll use the second formulation. (You can work through the first as an *exercise*.)

The *constrained optimization problem* is:

Minimize $\frac{1}{2}||\mathbf{w}||^2$

such that

$y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq 1$ for $i = 1, \ldots, m$.

Referring back, this means the *Lagrangian* is

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i \left(y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) - 1 \right)$$

and a *necessary condition* for a solution is that

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \qquad \frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial w_0} = 0.$$

Working these out is easy:

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} ||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i \left(y_i f_{\mathbf{w}, w_0}(\mathbf{x}_i) - 1 \right) \right)$$
$$= \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \frac{\partial}{\partial \mathbf{w}} \left(\mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x}_i) + w_0 \right)$$
$$= \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\Phi}(\mathbf{x}_i)$$

and

$$\frac{\partial L(\mathbf{w}, w_0, \boldsymbol{\alpha})}{\partial w_0} = -\frac{\partial}{\partial w_0} \left(\sum_{i=1}^{m} \alpha_i y_i f_{\mathbf{w}, w_0}(\mathbf{x}_i) \right)$$
$$= -\frac{\partial}{\partial w_0} \left(\sum_{i=1}^{m} \alpha_i y_i \left(\mathbf{w}^T \boldsymbol{\Phi}(\mathbf{x}_i) + w_0 \right) \right)$$
$$= -\sum_{i=1}^{m} \alpha_i y_i.$$

Equating those to 0 and adding the *KKT conditions* tells us several things:

1. The weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}(\mathbf{x}_i)$$

   with $\boldsymbol{\alpha} \geq \mathbf{0}$. This is important: we'll return to it in a moment.

2. There is a constraint that

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

   This will be needed for working out the *dual Lagrangian*.

3. For each example

$$\alpha_i [y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) - 1] = 0.$$

The maximum margin classifier

The fact that for each example

$$\alpha_i[y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) - 1] = 0$$

means that:

Either $y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) = 1$ *or* $\alpha_i = 0$.

This means that examples fall into two groups.

1. Those for which $y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) = 1$.

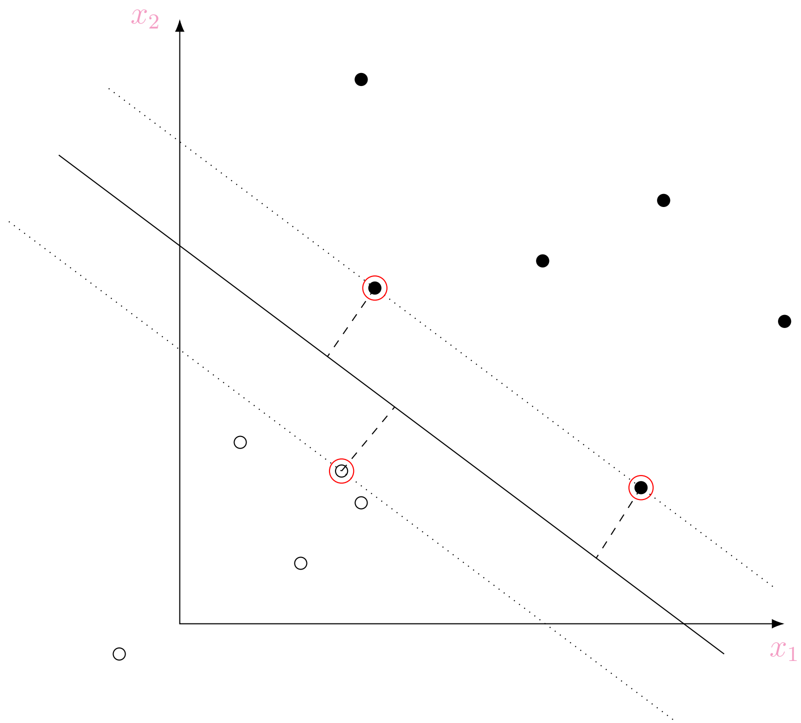
   As the contraint used to maxmize the margin was $y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq 1$ these are *the examples that are closest to the boundary.*

   They are called *support vectors* and they can have *non-zero weights*.

2. Those for which $y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \neq 1$.

   These are non-support vectors *and in this case it must be that* $\alpha_i = 0$.

Support vectors:



1. *Circled examples:* support vectors with $\alpha_i > 0$.
2. *Other examples:* have $\alpha_i = 0$.

Remember that

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}(\mathbf{x}_i).$$

so *the weight vector* $\mathbf{w}$ *only depends on the support vectors*.

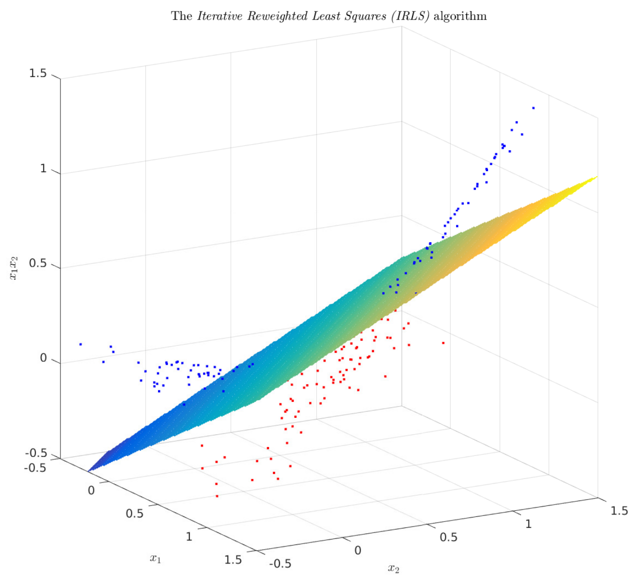
ALSO: the dual parameters $\boldsymbol{\alpha}$ can be used as an *alternative* set of weights. The overall classifier is

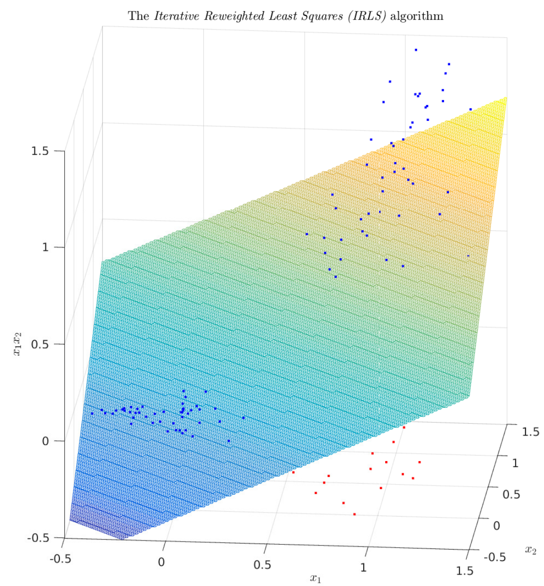
$$\begin{aligned} h_{\mathbf{w},w_0}(\mathbf{x}) &= \operatorname{sgn}\left(w_0 + \mathbf{w}^T \mathbf{\Phi}(\mathbf{x})\right) \\ &= \operatorname{sgn}\left(w_0 + \sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x})\right) \\ &= \operatorname{sgn}\left(w_0 + \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right) \end{aligned}$$

where $K(\mathbf{x}_i, \mathbf{x}) = \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x})$ is called the *kernel*.

The maximum margin classifier

Remember where this process started:





The kernel is computing

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{\Phi}^T(\mathbf{x})\mathbf{\Phi}(\mathbf{x}')$$
$$= \sum_{i=1}^{k} \phi_i(\mathbf{x})\phi_i(\mathbf{x}')$$

This is generally called an *inner product*.

The maximum margin classifier

If it's a *hard problem* then you'll probably want *lots of basis functions* so k *is BIG:*

$$\begin{aligned}
h_{\mathbf{w},w_0}(\mathbf{x}) &= \operatorname{sgn}\left(w_0 + \mathbf{w}^T\mathbf{\Phi}(\mathbf{x})\right) \\
&= \operatorname{sgn}\left(w_0 + \sum_{i=1}^{k} w_i\phi_i(\mathbf{x})\right) \\
&= \operatorname{sgn}\left(w_0 + \sum_{i=1}^{m} \alpha_i y_i \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x})\right) \\
&= \operatorname{sgn}\left(w_0 + \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)
\end{aligned}$$

What if $K(\mathbf{x}, \mathbf{x}')$ is easy to compute even if k is *HUGE*? (In particular $k >> m$.)

1. We get a definite computational advantage by using the dual version with weights $\boldsymbol{\alpha}$.
2. *Mercer's theorem* tells us exactly when a function K has a corresponding set of *basis functions* $\{\phi_i\}$.

Designing good kernels K is a subject in itself.

Luckily *for the majority of the time* you will tend to see one of the following:

1. *Polynomial:*
$$K_{c,d}(\mathbf{x}, \mathbf{x}') = (c + \mathbf{x}^T \mathbf{x}')^d$$
where c and d are parameters.

2. *Radial basis function (RBF):*
$$K_{\sigma^2}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{x}'||^2\right)$$
where σ^2 is a parameter.

The last is particularly prominent. Interestingly, the corresponding set of basis functions is *infinite*. (So we get an improvement in computational complexity from infinite to *linear in the number of examples*!)

Maximum margin classifier: the dual version

Collecting together some of the results up to now:

1. The Lagrangian is
$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_i \alpha_i (y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) - 1).$$

2. The weight vector is
$$\mathbf{w} = \sum_i \alpha_i y_i \boldsymbol{\Phi}(\mathbf{x}_i).$$

3. The KKT conditions require
$$\sum_i \alpha_i y_i = 0.$$

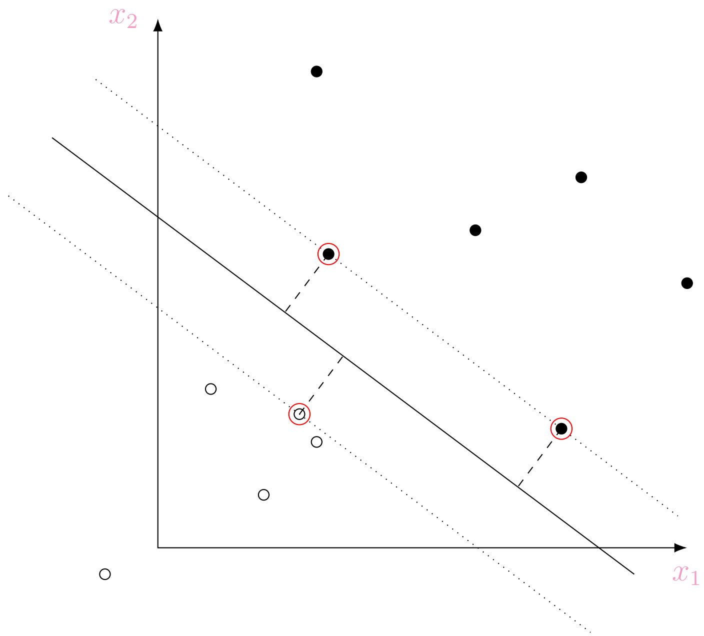
It's easy to show (this is an *exercise*) that the *dual optimization problem* is to maximize
$$\tilde{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$
such that $\boldsymbol{\alpha} \geq \mathbf{0}$.

There is one thing still missing:

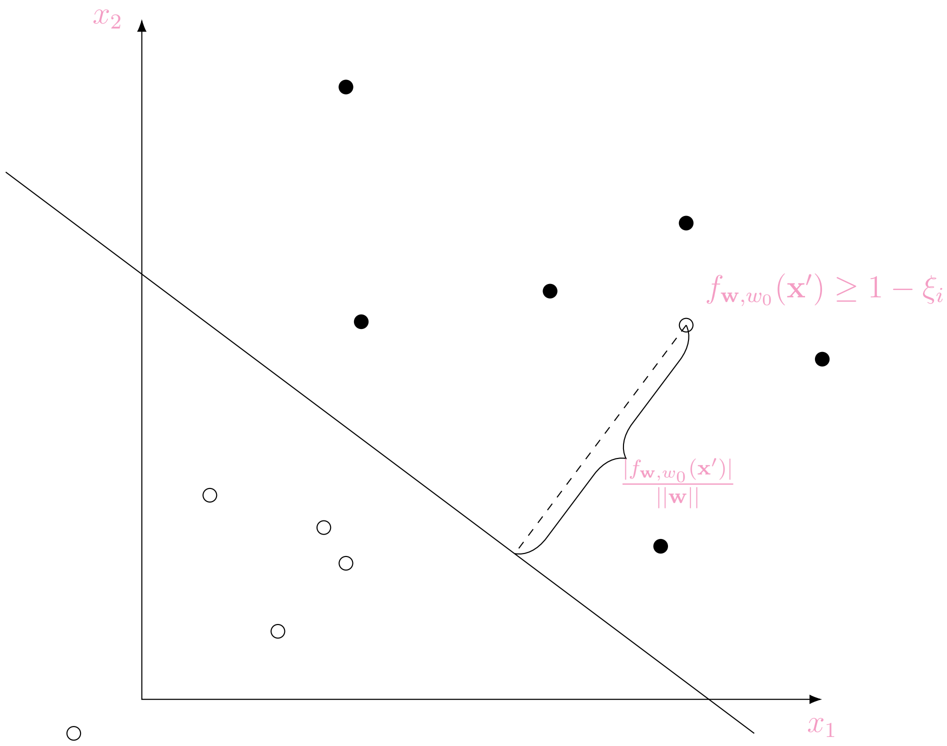
Problem: so far we've only covered the *linearly separable* case.

Even though that means linearly separable *in the extended space* it's still not enough.

By dealing with this we get the *Support Vector Machine (SVM)*.



Fortunately a small modification allows us to let *some* examples be misclassified.



We introduce the *slack variables* ξ_i, one for *each example*.

Although $f_{\mathbf{w},w_0}(\mathbf{x}') < 0$ we have $f_{\mathbf{w},w_0}(\mathbf{x}') \geq 1 - \xi_i$ and we try to force ξ_i to be small.

The *constrained optimization problem* was:

$$\text{argmin}_{\mathbf{w},w_0} \tfrac{1}{2}||\mathbf{w}||^2 \text{ such that } y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq 1 \text{ for } i = 1, \ldots, m.$$

The *constrained optimization problem* is now modified to:

$$\underset{\mathbf{w},w_0,\boldsymbol{\xi}}{\text{argmin}} \quad \underbrace{\frac{1}{2}||\mathbf{w}||^2}_{\text{Maximize the margin}} \quad + \quad \underbrace{C\sum_{i=1}^{m} \xi_i}_{\text{Control misclassification}}$$

such that

$$y_i f_{\mathbf{w},w_0}(\mathbf{x}_i) \geq 1 - \xi_i \text{ and } \xi_i > 0 \text{ for } i = 1, \ldots, m.$$

There is a *further new parameter* C that controls the trade-off between *maximizing the margin* and *controlling misclassification*.

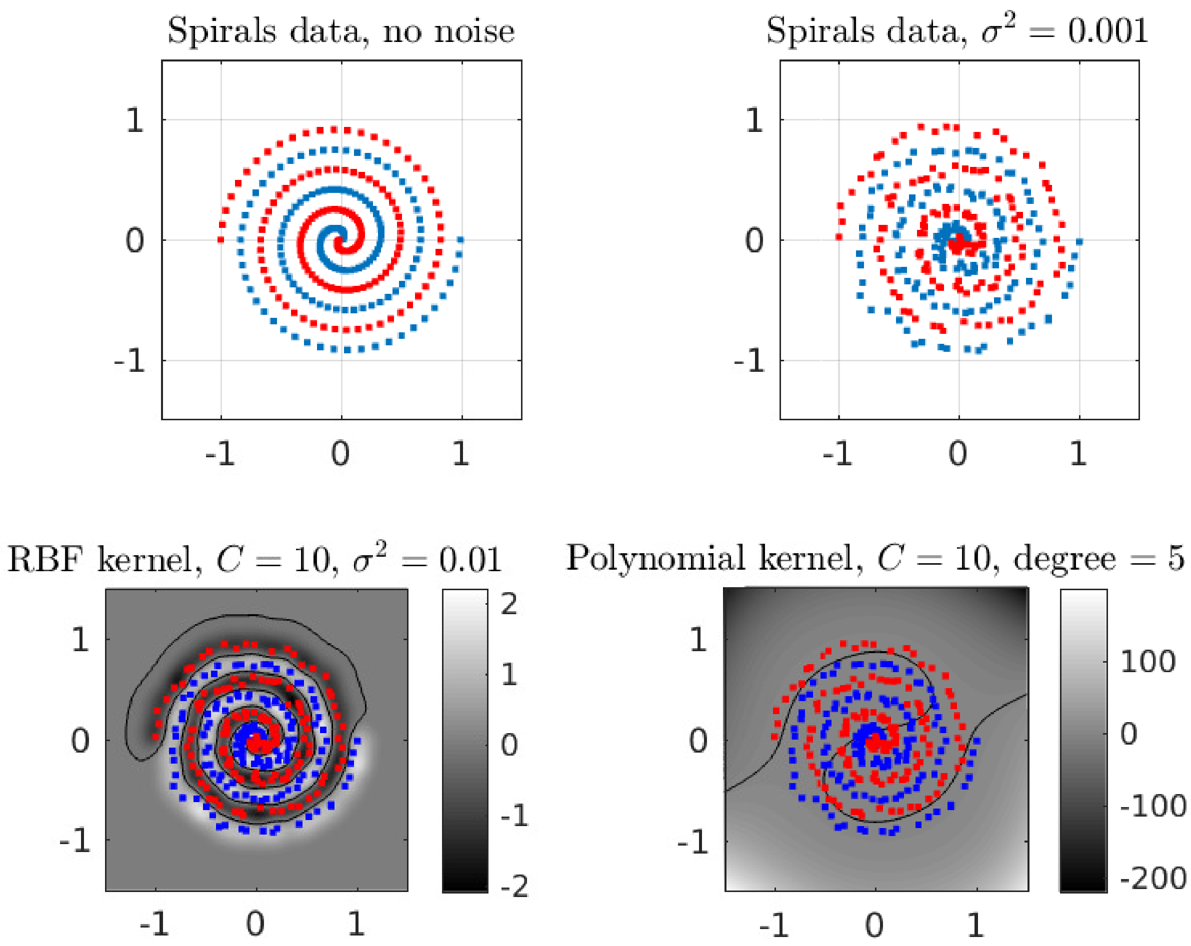
Once again, the theory of *constrained optimization* can be employed:

1. We get the *same insights* into the solution of the problem, and the *same conclusions*.
2. The development is exactly analogous to what we've just seen.

However as is often the case it is not straightforward to move all the way to having a functioning training algorithm.

For this some attention to good *numerical computing* is required. See:

Fast training of support vector machine using sequential minimal optimization, J. C. Platt, *Advances in Kernel Methods*, MIT Press 1999.



Spirals data, no noise
Spirals data, $\sigma^2 = 0.001$
RBF kernel, $C = 10$, $\sigma^2 = 0.01$
Polynomial kernel, $C = 10$, degree $= 5$
1
0
-1
2
-2
100
-100
-200

Supervised learning in practice

We now look at several issues that need to be considered when *applying machine learning algorithms in practice*:

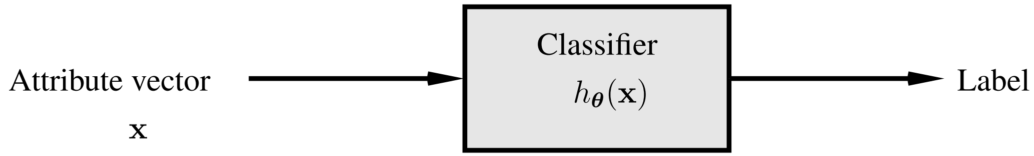
- We often have more examples from some classes than from others.
- The *obvious* measure of performance is not always the *best*.
- Much as we'd love to have an optimal method for *finding hyperparameters*, we don't have one, and it's *unlikely that we ever will*.
- We need to exercise care if we want to claim that one approach is superior to another.

This part of the course has an *unusually large number of Commandments*.

That's because *so many people get so much of it wrong!*.

Supervised learning

As usual, we want to design a *classifier*.



It should take an attribute vector

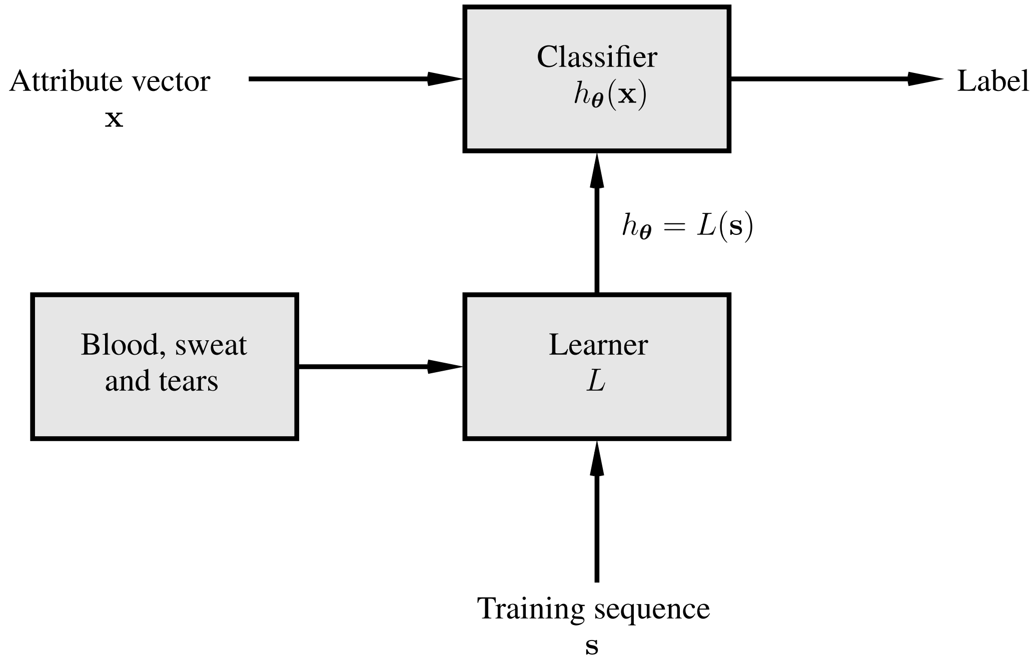
$$\mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

and label it.

We now denote a classifier by $h_{\boldsymbol{\theta}}(\mathbf{x})$ where $\boldsymbol{\theta}^T = \begin{pmatrix} \mathbf{w} & \mathbf{p} \end{pmatrix}$ denotes any weights $\mathbf{w}$ and (hyper)parameters $\mathbf{p}$.

To keep the discussion and notation simple we assume a *classification problem* with *two classes* labelled $+1$ *(positive examples)* and -1 *(negative examples)*.

Previously, the learning algorithm was a box labelled L.



Unfortunately that turns out not to be enough, so *a new box has been added*.

We've already come across the Commandment:

Thou shalt *try a simple method*. Preferably *many* simple methods.

Now we will add:

Thou shalt use an *appropriate measure of performance*.

Measuring performance

How do you assess the performance of your classifier?

1. That is, *after training*, how do you know how well you've done?
2. In general, the only way to do this is to divide your examples into a smaller *training set* $\mathbf{s}$ of m examples and a *test set* $\mathbf{s}'$ of m' examples.

The *GOLDEN RULE*: *data used to assess performance must NEVER have been seen during training*.

This might seem obvious, but it was a major flaw in a lot of early work.

Measuring performance

How do we choose m and m'? Trial and error!

Assume the training is complete, and we have a classifier $h_{\boldsymbol{\theta}}$ obtained using only $\mathbf{s}$. How do we use $\mathbf{s}'$ to assess our method's performance?

The obvious way is to see how many examples in $\mathbf{s}'$ the classifier classifies correctly:

$$\hat{\text{er}}_{\mathbf{s}'}(h_{\boldsymbol{\theta}}) = \frac{1}{m'} \sum_{i=1}^{m'} \mathbb{I}\left[h_{\boldsymbol{\theta}}(\mathbf{x}'_i) \neq y'_i\right]$$

where

$$\mathbf{s}' = \left[(\mathbf{x}'_1, y'_1) \;\; (\mathbf{x}'_2, y'_2) \;\; \cdots \;\; (\mathbf{x}'_{m'}, y'_{m'})\right]^T$$

and

$$\mathbb{I}\left[z\right] = \begin{cases} 1 & \text{if } z = \text{true} \\ 0 & \text{if } z = \text{false} \end{cases}.$$

This is just an estimate of the *probability of error* and is often called the *accuracy*.

Unbalanced data

Unfortunately it is often the case that we have *unbalanced data* and this can make such a measure misleading. For example:

If the data is naturally such that *almost all examples are negative* (medical diagnosis for instance) then simply *classifying everything as negative* gives a high performance using this measure.

We need more subtle measures.

For a classifier h and any set $\mathbf{s}$ of size m containing m^+ positive examples and m^- negative examples...

Define

1. The *true positives*

$$P^+ = \{(\mathbf{x}, +1) \in \mathbf{s} | h(\mathbf{x}) = +1\}, \text{ and } p^+ = |P^+|$$

2. The *false positives*

$$P^- = \{(\mathbf{x}, -1) \in \mathbf{s} | h(\mathbf{x}) = +1\}, \text{ and } p^- = |P^-|$$

3. The *true negatives*

$$N^+ = \{(\mathbf{x}, -1) \in \mathbf{s} | h(\mathbf{x}) = -1\}, \text{ and } n^+ = |N^+|$$

4. The *false negatives*

$$N^- = \{(\mathbf{x}, +1) \in \mathbf{s} | h(\mathbf{x}) = -1\}, \text{ and } n^- = |N^-|$$

Thus $\hat{\text{er}}_{\mathbf{s}}(h) = (p^+ + n^+)/m$.

This allows us to define more discriminating measures of performance.

Performance measures

Some standard performance measures:

1. Precision $\frac{p^+}{p^+ + p^-}$.
2. Recall $\frac{p^+}{p^+ + n^-}$.
3. Sensitivity $\frac{p^+}{p^+ + n^-}$.
4. Specificity $\frac{n^+}{n^+ + p^-}$.
5. False positive rate $\frac{p^-}{p^- + n^+}$.
6. Positive predictive value $\frac{p^+}{p^+ + p^-}$.
7. Negative predictive value $\frac{n^+}{n^+ + n^-}$.
8. False discovery rate $\frac{p^-}{p^- + p^+}$.

In addition, plotting sensitivity (true positive rate) against the false positive rate while a parameter is varied gives the *receiver operating characteristic (ROC)* curve.

Performance measures

The following specifically take account of unbalanced data:

1. Matthews Correlation Coefficient (MCC)

$$\text{MCC} = \frac{p^+n^+ - p^-n^-}{\sqrt{(p^+ + p^-)(n^+ + n^-)(p^+ + n^-)(n^+ + p^-)}}$$

2. F1 score

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

When data is unbalanced these are preferred over the accuracy.

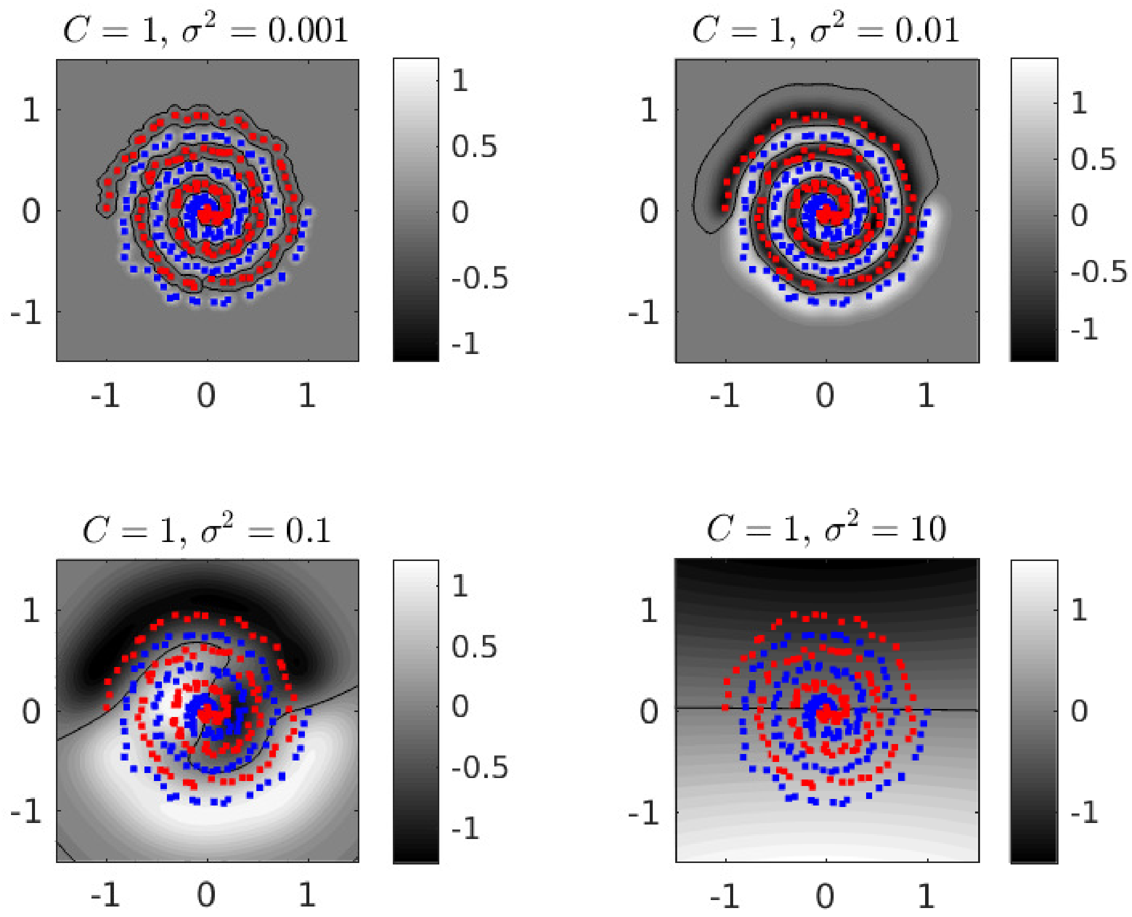
Thou shalt not use *default parameters*.

Thou shalt not use parameters chosen by an *unprincipled formula*.

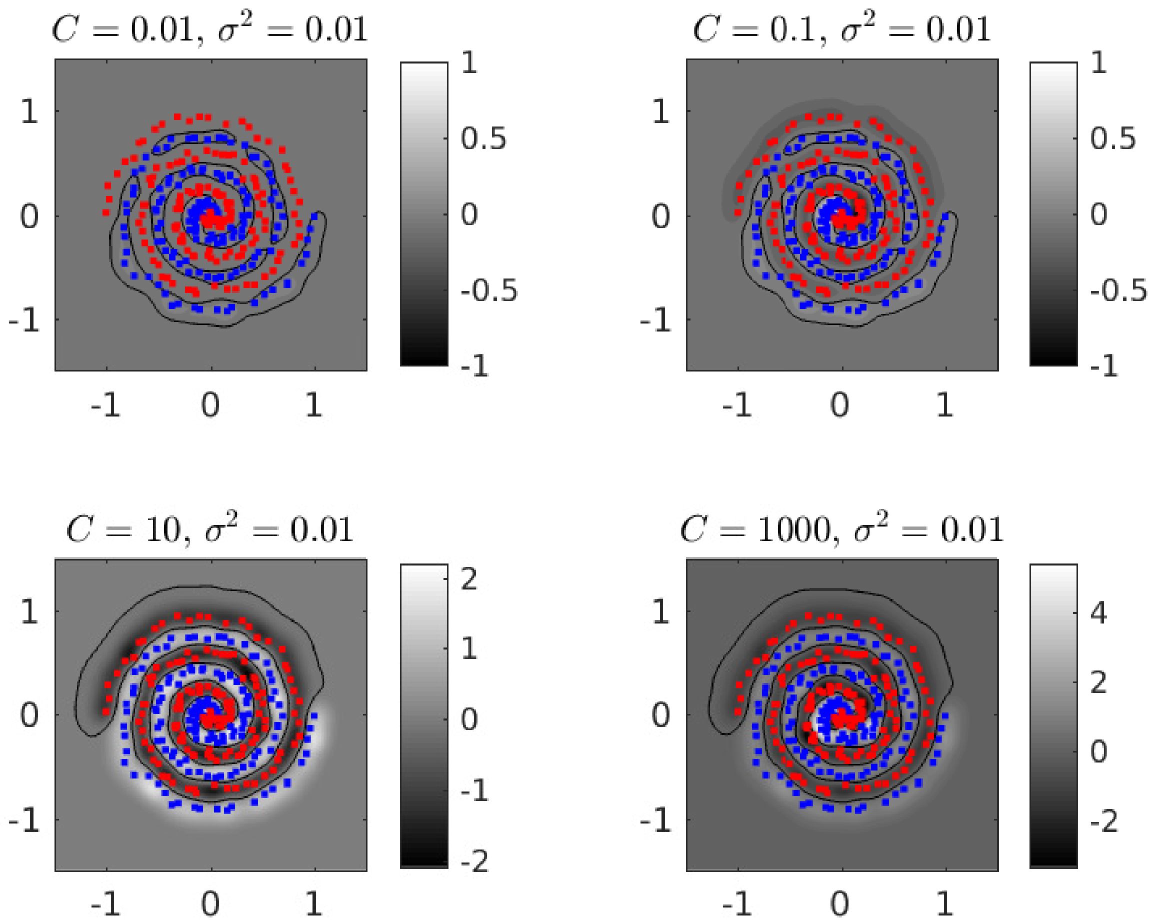
Thou shalt not avoid this issue by clicking on ‘Learn’ and *hoping it works*.

Thou shalt either *choose them carefully* or *integrate them out*.

Bad hyperparameters give bad performance



Bad hyperparameters give bad performance



Validation and crossvalidation

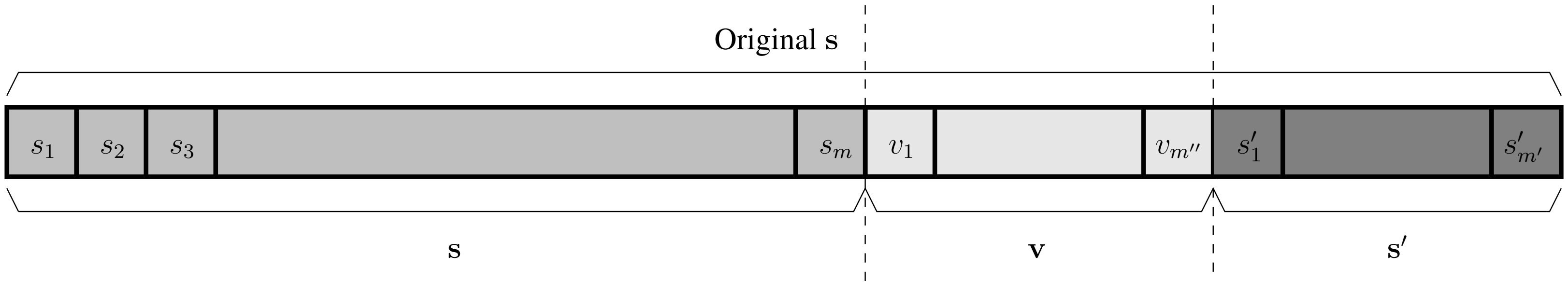
The next question: how do we choose hyperparameters?

Answer: *try different values and see which values give the best (estimated) performance*.

There is however a problem:

If I use my test set $\mathbf{s}'$ to find good hyperparameters, *then I can't use it to get a final measure of performance*. (See the Golden Rule above.)

Solution 1: make a further division of the complete set of examples to obtain a third, *validation* set:



Validation and crossvalidation

Now, to choose the value of a hyperparameter p:

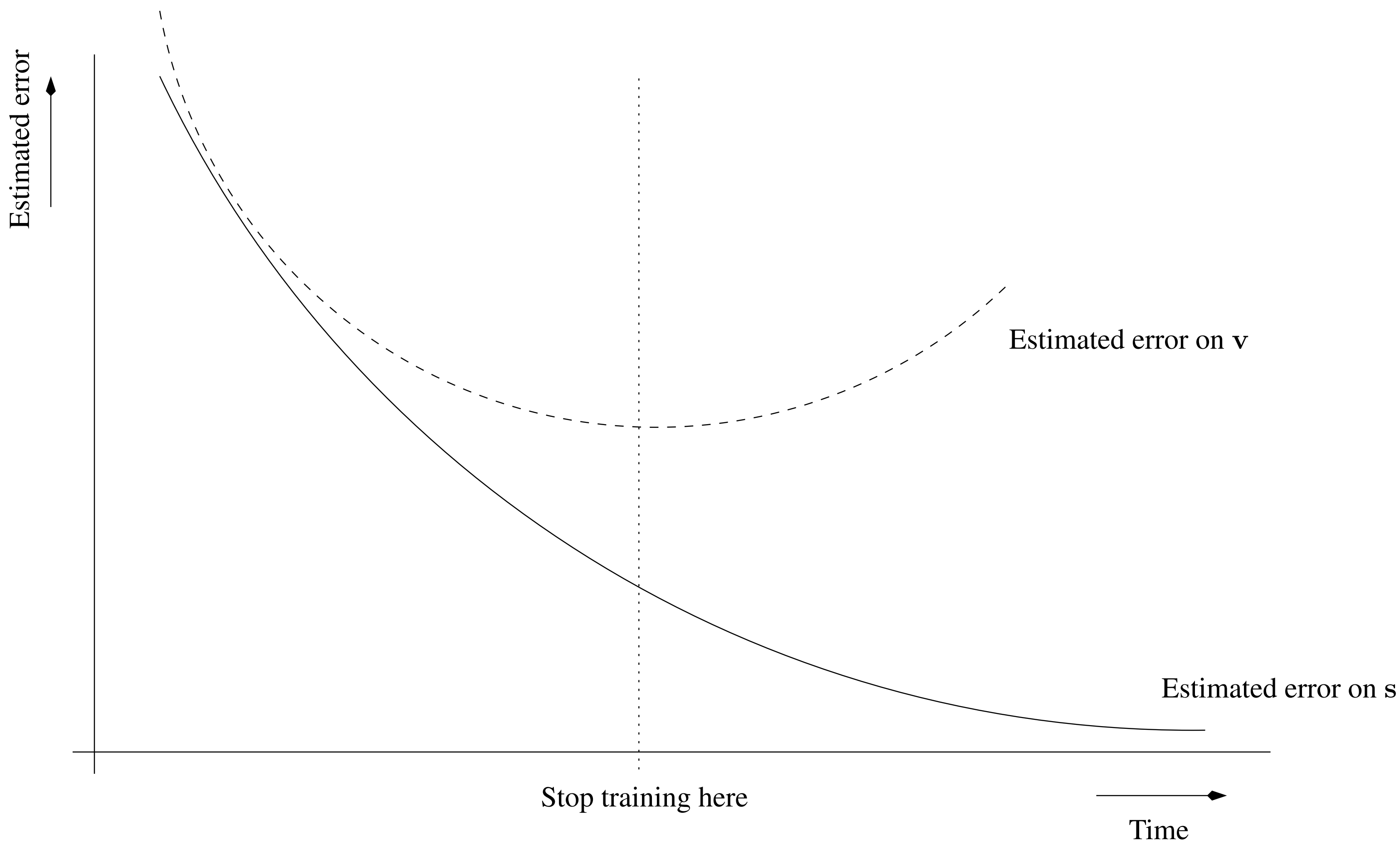
For some range of values $p_1, p_2, \ldots, p_n$

1. Run the training algorithm using training data $\mathbf{s}$ and with the hyperparameter set to p_i.
2. Assess the resulting $h_{\boldsymbol{\theta}}$ by computing a suitable measure (for example accuracy, MCC or F1) using $\mathbf{v}$.

Finally, select the $h_{\boldsymbol{\theta}}$ with maximum estimated performance and assess its *actual* performance using $\mathbf{s}'$.

Validation and crossvalidation

This was originally used in a similar way when deciding the best point at which to *stop training* a neural network.



The figure shows the typical scenario.

Crossvalidation

The method of *crossvalidation* takes this a step further.

We our complete set into training set $\mathbf{s}$ and testing set $\mathbf{s}'$ as before.

But now instead of further subdividing $\mathbf{s}$ just once we divide it into n *folds* $\mathbf{s}^{(i)}$ each having m/n examples.

Typically $n = 10$ although other values are also used, for example if $n = m$ we have *leave-one-out* cross-validation.

Crossvalidation

Let $\mathbf{s}_{-i}$ denote the set obtained from $\mathbf{s}$ by *removing* $\mathbf{s}^{(i)}$.

Let $\hat{\text{er}}_{\mathbf{s}^{(i)}}(h)$ denote any suitable error measure, such as accuracy, MCC or F1, computed for h using fold i.

Let $L_{\mathbf{s}_{-i},\mathbf{p}}$ be the classifier obtained by running learning algorithm L on examples $\mathbf{s}_{-i}$ using hyperparameters p.

Then,

$$\frac{1}{n}\sum_{i=1}^{n} \hat{\text{er}}_{\mathbf{s}^{(i)}}(L_{\mathbf{s}_{-i},\mathbf{p}})$$

is the *n-fold crossvalidation error estimate*.

So for example, let $\mathbf{s}_j^{(i)}$ denote the jth example in the ith fold. Then using accuracy as the error estimate we have

$$\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m/n} \mathbb{I}\left[L_{\mathbf{s}_{-i},\mathbf{p}}(\mathbf{x}_j^{(i)}) \neq y_j^{(i)}\right]$$

Crossvalidation

Two further points:

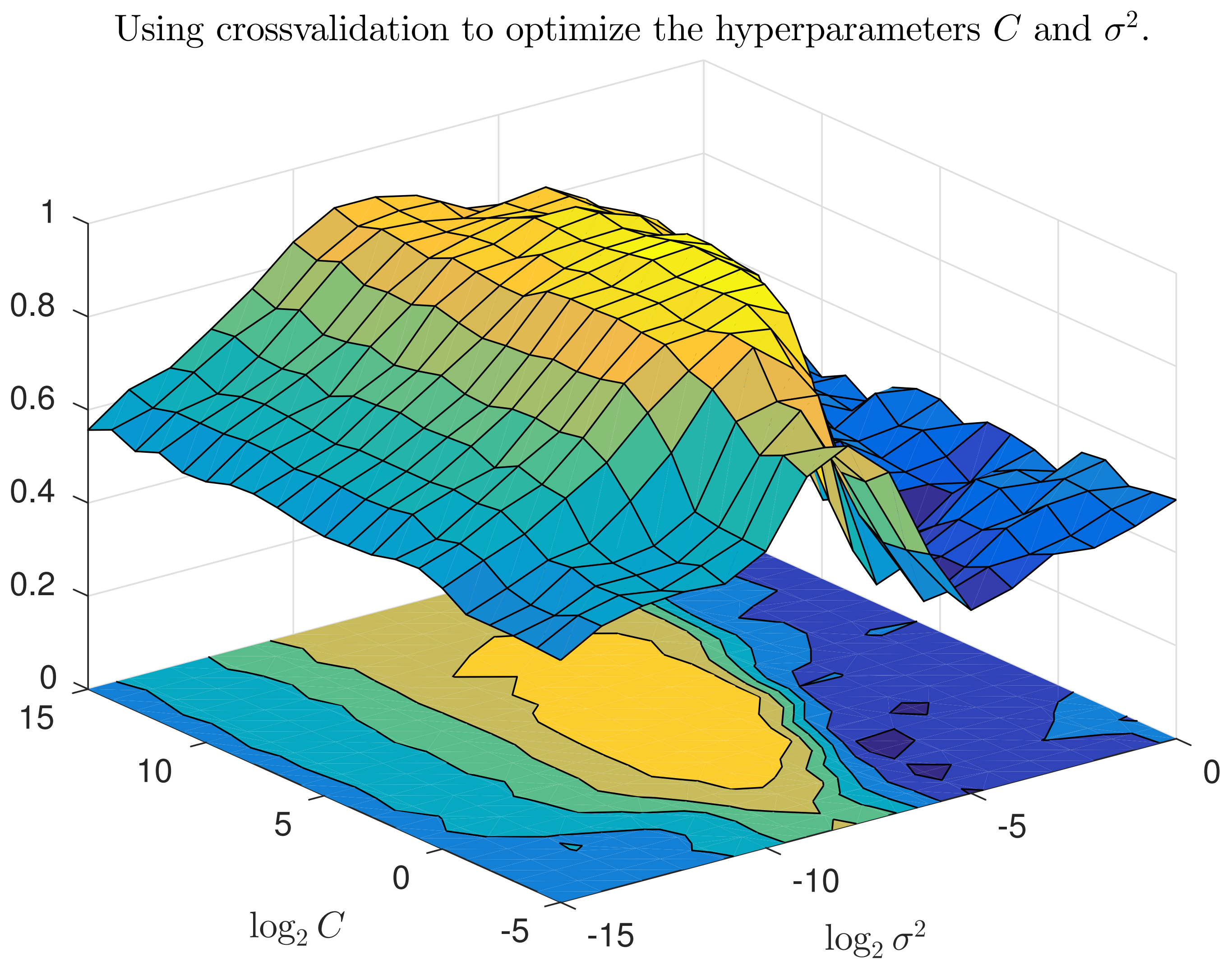
1. What if the data are unbalanced? *Stratified crossvalidation* chooses folds such that the proportion of positive examples in each fold matches that in $\mathbf{s}$.
2. Hyperparameter choice can be done just as above, using a basic search.

What happens however if we have multiple hyperparameters?

1. We can search over all combinations of values for specified ranges of each parameter.
2. This is the *standard method in choosing parameters for support vector machines (SVMs)*.
3. With SVMs it is generally limited to the case of only two hyperparameters.
4. Larger numbers quickly become infeasible.

Crossvalidation

This is what we get for an *SVM* applied to the *two spirals*:



Thou shalt *provide evidence* before claiming that *thy method is the best*.

The shalt take extra notice of this Commandment if *thou considers thyself a True And Pure Bayesian*.

Comparing classifiers

Imagine I have compared the *Bloggs Classificator 2000* and the *CleverCorp Discriminotron* and found that:

1. Bloggs Classificator 2000 has estimated accuracy 0.981 on the test set.
2. CleverCorp Discriminotron has estimated accuracy 0.982 on the test set.

Can I claim that the CleverCorp Discriminotron is the better classifier?

Answer:

NO! NO! NO! NO! NO! NO! NO! NO! NO!!!!!!!!!!!!!!

Comparing classifiers

NO!!!!!!!

Note for next year: include photo of grumpy-looking cat.

From *Mathematical Methods for Computer Science*:

The *Central Limit Theorem*: If we have independent identically distributed (iid) random variables $X_1, X_2, \ldots, X_n$ with mean

$$\mathbb{E}[X] = \mu$$

and standard deviation

$$\mathbb{E}\left[(X-\mu)^2\right] = \sigma^2$$

then as $n \to \infty$

$$\frac{\hat{X}_n - \mu}{\sigma/\sqrt{n}} \to N(0,1)$$

where

$$\hat{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

Assessing a single classifier

We have tables of values z_p such that if $x \sim N(0,1)$ then

$$\Pr\left(-z_p \leq x \leq z_p\right) > p.$$

Rearranging this using the equation from the previous slide we have that with probability p

$$\mu \in \left[\hat{X}_n \pm z_p\sqrt{\frac{\sigma^2}{n}}\right].$$

We don't know σ^2 but it can be estimated using

$$\sigma^2 \simeq \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \hat{X}_n\right)^2.$$

Alternatively, when X takes only values 0 or 1

$$\sigma^2 = \mathbb{E}\left[(X-\mu)^2\right] = \mathbb{E}\left[X^2\right] - \mu^2 = \mu(1-\mu) \simeq \hat{X}_n(1-\hat{X}_n).$$

Assessing a single classifier

The *actual probability of error* for a classifier h is

$$\text{er}(h) = \mathbb{E}\left[\mathbb{I}\left[h(\mathbf{x}) \neq y\right]\right]$$

and we are *estimating* $\text{er}(h)$ using the *accuracy*

$$\hat{\text{er}}_{\mathbf{s}}(h) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left[h(\mathbf{x}_i) \neq y_i\right]$$

for a test set $\mathbf{s}$.

We can find a confidence interval for this estimate using precisely the derivation above, simply by noting that the X_i are the random variables

$$X_i = \mathbb{I}\left[h(\mathbf{x}_i) \neq y_i\right].$$

Assessing a single classifier

Typically we are interested in a 95% confidence interval, for which $z_p = 1.96$.

Thus, when $m > 30$ (so that the central limit theorem applies) we know that, with probability 0.95

$$\boxed{\mathrm{er}(h) = \hat{\mathrm{er}}_{\mathrm{s}}(h) \pm 1.96\sqrt{\frac{\hat{\mathrm{er}}_{\mathrm{s}}(h)(1-\hat{\mathrm{er}}_{\mathrm{s}}(h))}{m}}}.$$

Example: I have 100 test examples and my classifier makes 18 errors. With probability 0.95 I know that

$$\begin{aligned}\mathrm{er}(h) &= 0.18 \pm 1.96\sqrt{\frac{0.18(1-0.18)}{100}} \\ &= 0.18 \pm 0.075.\end{aligned}$$

This should perhaps *raise an alarm* regarding our suggested comparison of classifiers above.

Assessing a single classifier

There is an important distinction to be made here:

1. The *mean of* X is μ and the *variance of* X is σ^2.
2. We can also ask about the mean and variance of $\hat{X}_n$.
3. The *mean of* $\hat{X}_n$ is

$$\begin{aligned}\mathbb{E}\left[\hat{X}_n\right] &= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[X_i\right] \\ &= \mu.\end{aligned}$$

4. It is left as an *exercise* to show that the *variance of* $\hat{X}_n$ is

$$\sigma^2_{\hat{X}_n} = \frac{\sigma^2}{n}.$$

Comparing classifiers

We are using the values z_p such that if $x \sim N(0,1)$ then

$$\Pr(-z_p \leq x \leq z_p) > p.$$

There is an *alternative* way to think about this.

1. Say we have a random variable Y with variance σ_Y^2 and mean μ_Y.
2. The random variable $Y - \mu_Y$ has variance σ_Y^2 and mean 0.
3. It is a straightforward exercise to show that dividing a random variable having variance σ^2 by σ gives us a new random variable with variance 1.
4. Thus the random variable $\frac{Y-\mu_Y}{\sigma_Y}$ has mean 0 and variance 1.

So: with probability p

$$Y = \mu_Y \pm z_p \sigma_Y$$
$$\mu_Y = Y \pm z_p \sigma_Y.$$

Compare this with what we saw earlier. *You need to be careful to keep track of whether you are considering the mean and variance of a single RV or a sum of RVs.*

Comparing classifiers

Now say I have classifiers h_1 *(Bloggs Classificator 2000)* and h_2 *(CleverCorp Discriminotron)* and I want to know something about the quantity

$$d = \text{er}(h_1) - \text{er}(h_2).$$

I estimate d using

$$\hat{d} = \hat{\text{er}}_{\mathbf{s}_1}(h_1) - \hat{\text{er}}_{\mathbf{s}_2}(h_2)$$

where $\mathbf{s}_1$ and $\mathbf{s}_2$ are *two* independent test sets.

Notice:

1. The estimate of d is a sum of random variables, and *we can apply the central limit theorem*.
2. The estimate is *unbiased*

$$\mathbb{E}\left[\hat{\text{er}}_{\mathbf{s}_1}(h_1) - \hat{\text{er}}_{\mathbf{s}_2}(h_2)\right] = d.$$

Also notice:

1. The two parts of the estimate $\hat{\text{er}}_{s_1}(h_1)$ and $\hat{\text{er}}_{s_2}(h_2)$ are each sums of random variables and *we can apply the central limit theorem to each*.
2. The variance of the estimate is the sum of the variances of $\hat{\text{er}}_{s_1}(h_1)$ and $\hat{\text{er}}_{s_2}(h_2)$.
3. Adding Gaussians gives another Gaussian.
4. *We can calculate a confidence interval for our estimate*.

With probability 0.95

$$\boxed{d = \hat{d} \pm 1.96\sqrt{\frac{\hat{\text{er}}_{s_1}(h_1)(1 - \hat{\text{er}}_{s_1}(h_1))}{m_1} + \frac{\hat{\text{er}}_{s_2}(h_2)(1 - \hat{\text{er}}_{s_2}(h_2))}{m_2}}}.$$

In fact, if we are using a split into training set s and test set s' we can generally obtain h_1 and h_2 using s and use the estimate

$$\hat{d} = \hat{\text{er}}_{s'}(h_1) - \hat{\text{er}}_{s'}(h_2).$$

Comparing classifiers—hypothesis testing

This still doesn't tell us directly about *whether one classifier is better than another*—whether h_1 is better than h_2.

What we actually want to know is whether

$$d = \text{er}(h_1) - \text{er}(h_2) > 0.$$

Say we've measured $\hat{D} = \hat{d}$. Then:

- Imagine the *actual value* of d is 0.
- Recall that the *mean* of $\hat{D}$ is d.
- So *larger* measured values $\hat{d}$ are *less likely*, even though some random variation is inevitable.
- If it is highly *unlikely* that when $d = 0$ a measured value of $\hat{d}$ would be observed, then we can be confident that $d > 0$.
- Thus we are interested in
$$\Pr(\hat{D} > d + \hat{d}).$$

This is known as a *one-sided bound*.

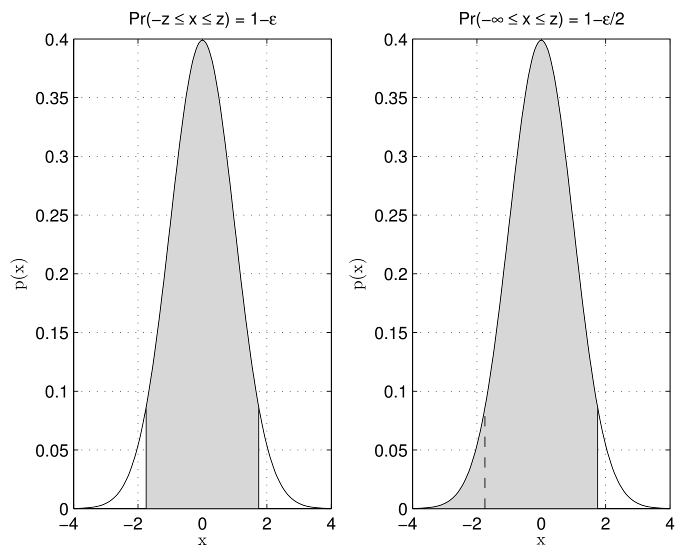
One-sided bounds

Given the *two-sided bound*

$$\Pr(-z_\epsilon \le x \le z_\epsilon) = 1 - \epsilon$$

we actually need to know the *one-sided bound*

$$\Pr(x \le z_\epsilon).$$



Clearly, if our random variable is *Gaussian* then $\Pr(x \le z_\epsilon) = 1 - \epsilon/2$.

Comparing algorithms: paired t-tests

We now know how to compare *hypotheses* h_1 and h_2.

But we still haven't properly addressed the comparison of *algorithms*.

- Remember, a learning algorithm L maps training data $\mathbf{s}$ to hypothesis h.
- So we *really* want to know about the quantity

$$d = \mathbb{E}_{\mathbf{s} \in S^m} \left[\text{er}(L_1(\mathbf{s})) - \text{er}(L_2(\mathbf{s})) \right].$$

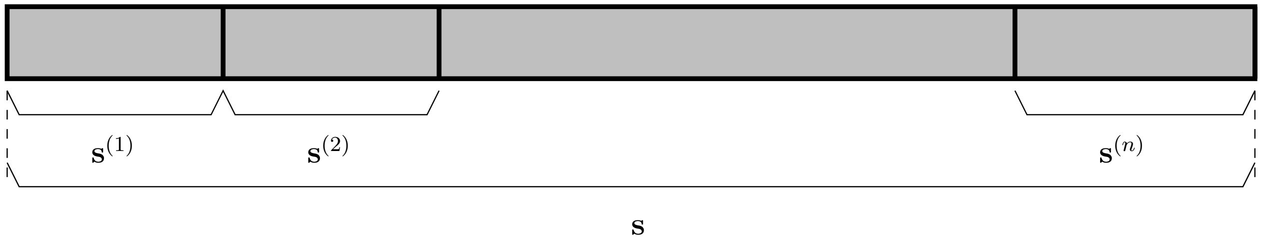
- This is the *expected difference* between the *actual errors* of the *two different* algorithms L_1 and L_2 .

Unfortunately, we have *only one set of data* $\mathbf{s}$ available and we *can only estimate* errors $\text{er}(h)$—we don't have access to the *actual quantities*.

We can however use the idea of *crossvalidation*.

Comparing algorithms: paired t-tests

Recall, we subdivide $\mathbf{s}$ into n *folds* $\mathbf{s}^{(i)}$ each having m/n examples



and denote by $\mathbf{s}_{-i}$ the set obtained from $\mathbf{s}$ by *removing* $\mathbf{s}^{(i)}$. Then

$$\frac{1}{n}\sum_{i=1}^{n} \hat{\text{er}}_{\mathbf{s}^{(i)}}(L(\mathbf{s}_{-i}))$$

is the n*-fold crossvalidation error estimate*. Now we estimate d using

$$\hat{d} = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{\text{er}}_{\mathbf{s}^{(i)}}(L_1(\mathbf{s}_{-i})) - \hat{\text{er}}_{\mathbf{s}^{(i)}}(L_2(\mathbf{s}_{-i}))\right].$$

Comparing algorithms: paired t-tests

As usual, there is a *statistical test* allowing us to assess *how likely this estimate is to mislead us*.

We will not consider the derivation in detail. With probability p

$$d \in \left[\hat{d} \pm t_{p,n-1}\sigma_{\hat{d}}\right].$$

This is analogous to the equations seen above, however:

- The parameter $t_{p,n-1}$ is analogous to z_p.
- The parameter $t_{p,n-1}$ is related to the area under the *Student's t-distribution* whereas z_p is related to the area under the normal distribution.
- The relevant estimate of *standard deviation* is

$$\sigma_{\hat{d}} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} \left(d_i - \hat{d}\right)^2}$$

where

$$d_i = \hat{\text{er}}_{\mathbf{s}(i)}(L_1(\mathbf{s}_{-i})) - \hat{\text{er}}_{\mathbf{s}(i)}(L_2(\mathbf{s}_{-i})).$$