Advanced Machine Learning

Over-Fitting

Overfitting, regularisation, feature selection

Outline

1. **Over-fitting?**
2. Controlling Complexity
3. Hidden structure
4. Regularisation

Over-fitting

- Complex machine can **over-fit**

  over-fitting*: fitting the training data well at the cost of getting poorer generalisation performance*

- Three red cars. . .

- If we used an infinitely flexible machine we can fit our training data perfectly, but would have no generalisation ability

Binary Classification Task for You

Class 1

Class 2

Which Category?

- Which category does the following image belong to?

Spurious Rules

- You ask a learning machine to solve a task based on data
- It will find a rule that does this, but not necessary the rule you had in mind—machine learning isn't magic, it can't read your mind
- Infinitely flexible machines have an infinity of spurious rules they can learn—they are useless
- What should we do?

All Binary Functions

$$\boldsymbol{x}_0 = 0\,0\,0 \quad y_0 = \begin{cases} 0 \\ \cancel{1} \end{cases}$$

$$\boxed{\boldsymbol{x}_1 = 1\,0\,0 \quad y_1 = \begin{cases} 0 \\ 1 \end{cases}} \; \textcolor{red}{\text{unseen}}$$

$$\boldsymbol{x}_2 = 0\,1\,0 \quad y_2 = \begin{cases} \cancel{0} \\ 1 \end{cases}$$

$$\boldsymbol{x}_3 = 1\,1\,0 \quad y_3 = \begin{cases} \cancel{0} \\ 1 \end{cases}$$

$$\boxed{\boldsymbol{x}_4 = 0\,0\,1 \quad y_4 = \begin{cases} 0 \\ \cancel{1} \end{cases}} \; \textcolor{red}{\text{seen}}$$

$$\boldsymbol{x}_5 = 1\,0\,1 \quad y_5 = \begin{cases} 0 \\ \cancel{1} \end{cases}$$

$$\boldsymbol{x}_6 = 0\,1\,1 \quad y_6 = \begin{cases} 0 \\ 1 \end{cases}$$

$$\boxed{\boldsymbol{x}_7 = 1\,1\,1 \quad y_7 = \begin{cases} 0 \\ 1 \end{cases}} \; \textcolor{red}{\text{unseen}}$$

$$\mathcal{D} = \big\{(0\,0\,0\,,\,0),\,(0\,1\,0\,,\,1),\,(1\,1\,0\,,\,1),\,(0\,0\,1\,,\,0),\,(1\,0\,1\,,\,0)\big\}$$

Are MLPs Universal Approximators?

- Yes and No
- Yes: If you give me any function, I can find an MLP that approximates that function to any desired accuracy
- No: If you give me an MLP, I can find a function with an arbitrary high approximation error
- Would an MLP that could approximate any function be useful?
- **Absolutely not!**

Outline

1. Over-fitting?
2. **Controlling Complexity**
3. Hidden structure
4. Regularisation

Controlling Complexity

- Infinitely flexible machine don't generalise (because any unseen data could have any value)
- **Machine learning only works because there is some structure in the data**
- A successful machine should capture this structure
- Even deep learning machines with millions of parameters only work because they successfully capture the structure of images or text
- Different learning machines have different performance on different problems because the data has different structure

Training Examples

- As we increase the number of training examples, we make it hard to find a spurious rule
- Bigger data sets allow us to use more complicated machines
- Part of the success of deep learning is because they use huge training sets—but this is only a part of their success
- (Labelled) data is often expensive to collect so we sometimes have no choice but to use a small training set
- One of the limitations of using deep learning comes because we often have limited data

Identifying Structure

- In some cases we know *a priori* some of the structure in the data
- In images we believe the identity of an object is invariant to translation and scaling
- The success of *convolutional neural networks* (CNNs) in deep learning is in large part because the convolutions respect translational invariance

Preprocessing

- Structure might often be obscure to the learning machine
- If we are trying to predict the spread of disease then a list of place names might be a lot less useful than their coordinates
- Imposing an ordering on an unordered set might **not** be useful

$$\Big\{\text{“blue”} : 0,\ \text{“brown”} : 1,\ \text{“green”} : 2,\ \text{“black”} : 3\Big\}$$

- Choosing an encoding that reflect meaningful structure is essential to successful machine learning

Automatic Preprocessing

- One view of deep learning is that each layer (particularly in CNNs) acts as a preprocessor
- That is, it finds filters that captures features salient to the problem being tackled
- For both images and texts we expect salient features to be spatially localised (CNN finds localised filter)
- The deep structure allows ever more complicated features to be captured—that is, we can find spatially localised features on different scales
- Having very large datasets and simple filters (the number of weights in the CNN layers tends to be small) stops overfitting

Outline

1. Over-fitting?
2. Controlling Complexity
3. **Hidden structure**
4. Regularisation

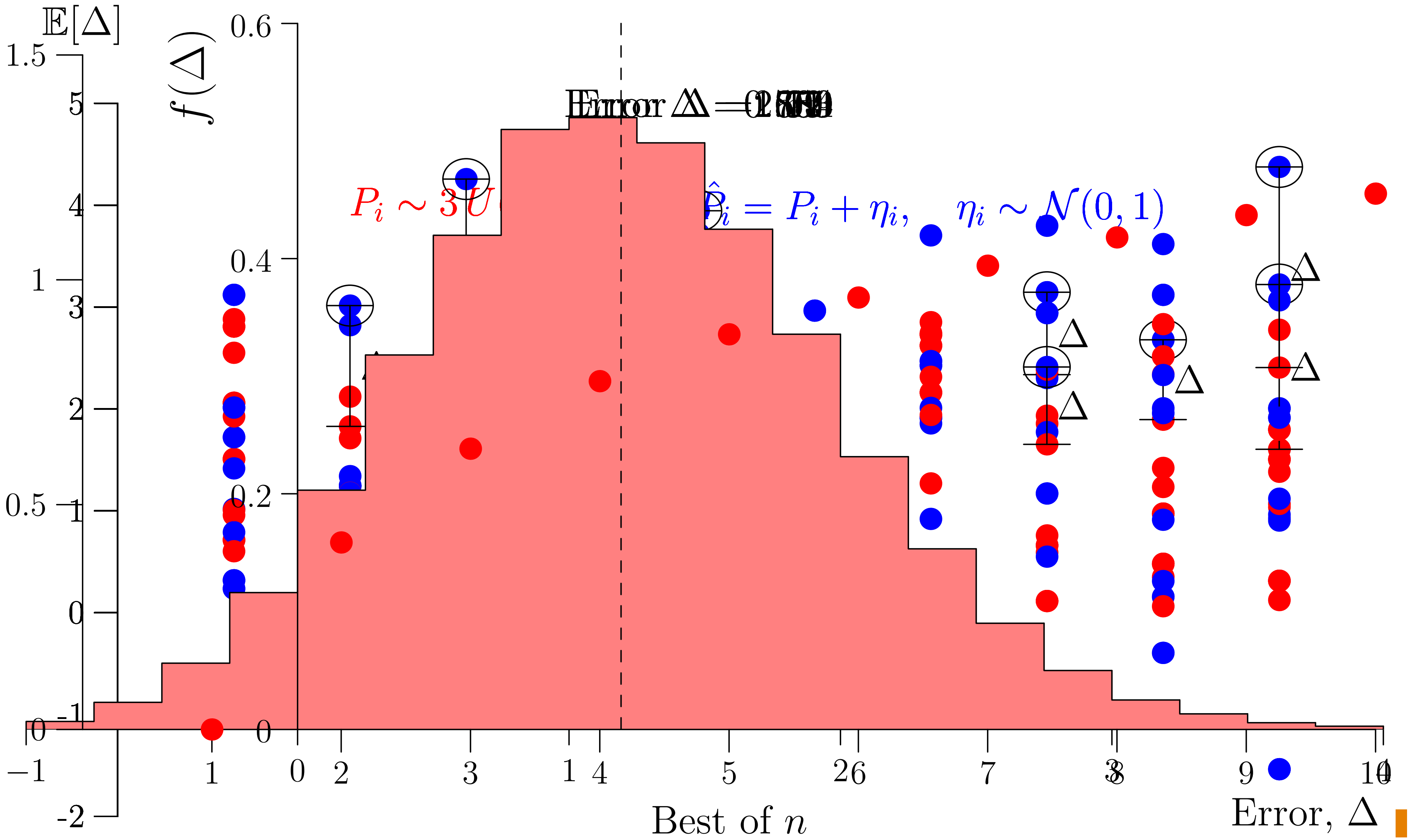
Hidden Structure

- Often the structure of data is invisible to us
- A very successful strategy is to try many different machine learning techniques and choose the best (stupid but effective)
- Often learning machines have adjustable parameters (hyper-parameters) that we have to set (they are the same for all input data, but change with the problem)
- We need to choose the hyper-parameters to fit the data in our problem
- Fine tuning hyper-parameter is important and almost always required (true in SVMs, MLP, deep learning)

Measuring Generalisation Performance

- Recall, we want to predict **unseen** data
- **You cannot use data that you have trained on!**—you will overfit
- Need to split your data up into training and validation set
- Use the validation set to choose the hyper-parameters
- You need a separate testing set if you want to measure your generalisation performance

The Overfitting Game



Cross Validation

- If you want to use more data for training then you can use cross validation
- K-fold cross validation splits the data into K groups

$$\mathcal{D} = \{D_i\}_{i=1}^{P} \quad D_i = (\mathbf{x}_i, y_i)$$

D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}	D_{18}	D_{19}	D_{20}

Test Set Training Set

Leave-one-out cross validation / 10-fold cross validation

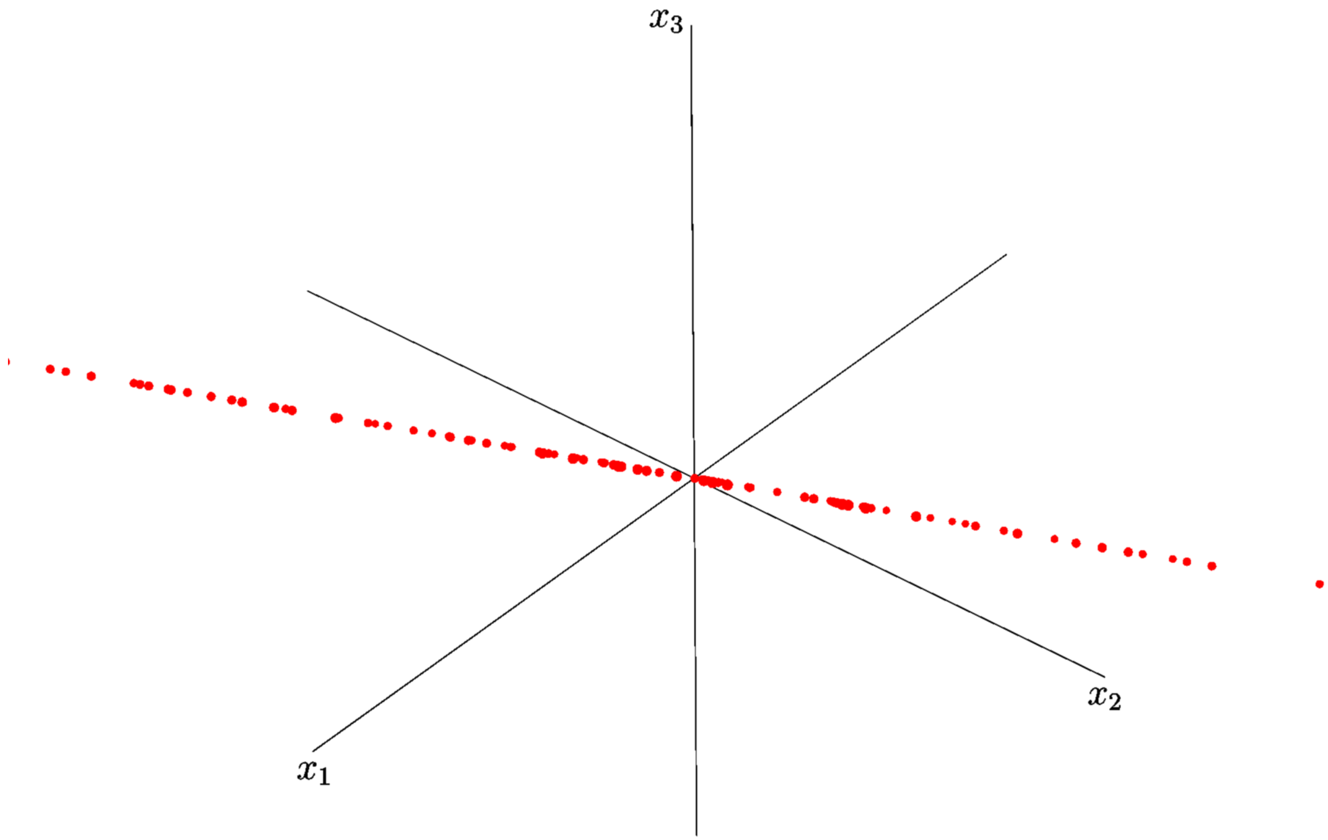
$E_g = 6.5$

$$\langle E_g \rangle = \frac{5 \; 5.8 \; 7.5 + 1.8 \; 2.2 + 3.4 \; 4.8 \; 3.7 \; 3.6 + 7.4 \; 4.6 \; 0.99 + 4.5 \; 4.6 \; 5.4 + 6.2 \; 3.3 \; 2.7}{15 \;/\; 10} = 4.3$$

- Leave-one-out cross-validation is extreme case

Hidden Structure

Can't spot low dimensional data by looking at numbers



Dimensionality Reduction

- We can sometimes simplify our machines by using less features
- We can project our data onto a lower dimensional sub-space (e.g. one with the maximum variation in the data: PCA)
- We can use clustering to find exemplars and recode our data in terms of distances from the exemplars (radial basis functions)
- Whether this helps depends on whether the information we discard is pertinent to the task we are trying to perform

Feature Selection

- Spurious features will allow us to find spurious rules (**over-fitting**)
- We can try different combinations of features to find the best set, although it rapidly becomes intractable to do this in all ways
- We can use various heuristics to decide which features to keep, but no heuristic is fail-safe method to find the best set of features
- Feature selection however can be powerful, often we can get very good results by keeping only a few variables
- As well as possibly improving generalisation we also get a more **interpretable** rule

Normalising Features

- Measuring a feature in millimeters or kilometers is going to make a lot of difference to the size of that feature
- Many learning algorithms are sensitive to the size of a feature (larger features are more important)
- If we don't know how important different features are then it makes sense to normalise our features, E.g.

$$x_i^\alpha \leftarrow \frac{x_i^\alpha - \hat{\mu}_i}{\hat{\sigma}_i}, \quad \hat{\mu} = \frac{1}{m}\sum_{\beta=1}^{m} x_i^\beta, \quad \hat{\sigma}_i^2 = \frac{1}{m-1}\sum_{\beta=1}^{m}(x_i^\beta - \hat{\mu}_i)^2$$

Outline

1. Over-fitting?
2. Controlling Complexity
3. Hidden structure
4. **Regularisation**

Explicit Regularisation

- As you've seen in the foundations of ML course, we can modify our error function to choose smoother functions

$$L = \sum_{k=1}^{m} \left(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}_k - y_k \right)^2 + \nu \|\boldsymbol{w}\|^2$$

(Good to normalise features)

- Second term is minimised when $w_i = 0$

- If w_i is large then

$$f(\boldsymbol{x}|\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x} = \sum_{i=1}^{p} w_i x_i$$

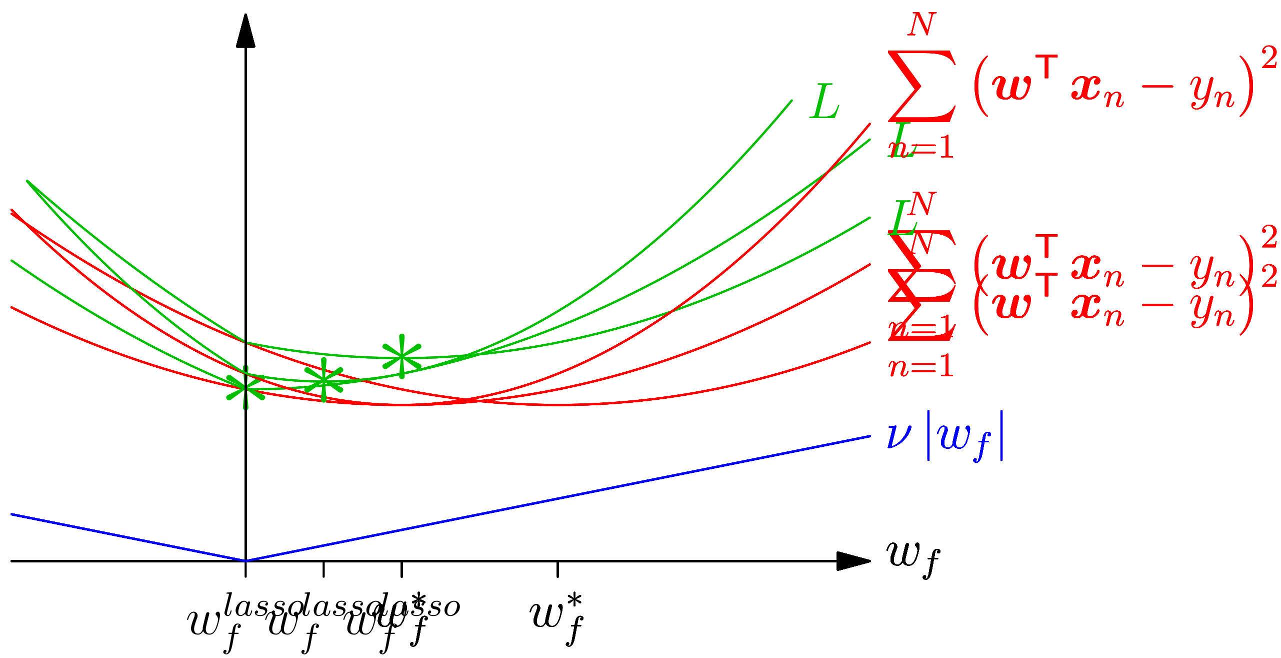
varies rapidly as we change x_i

Lasso

- We can use other regularisers

$$L = \sum_{k=1}^{m} \left(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}_k - y_k\right)^2 + \nu \sum_{i=1}^{p} |w_i|$$

- Spurious features (e.g. colour of flag on energy consumption) will give us a small improvement in training error

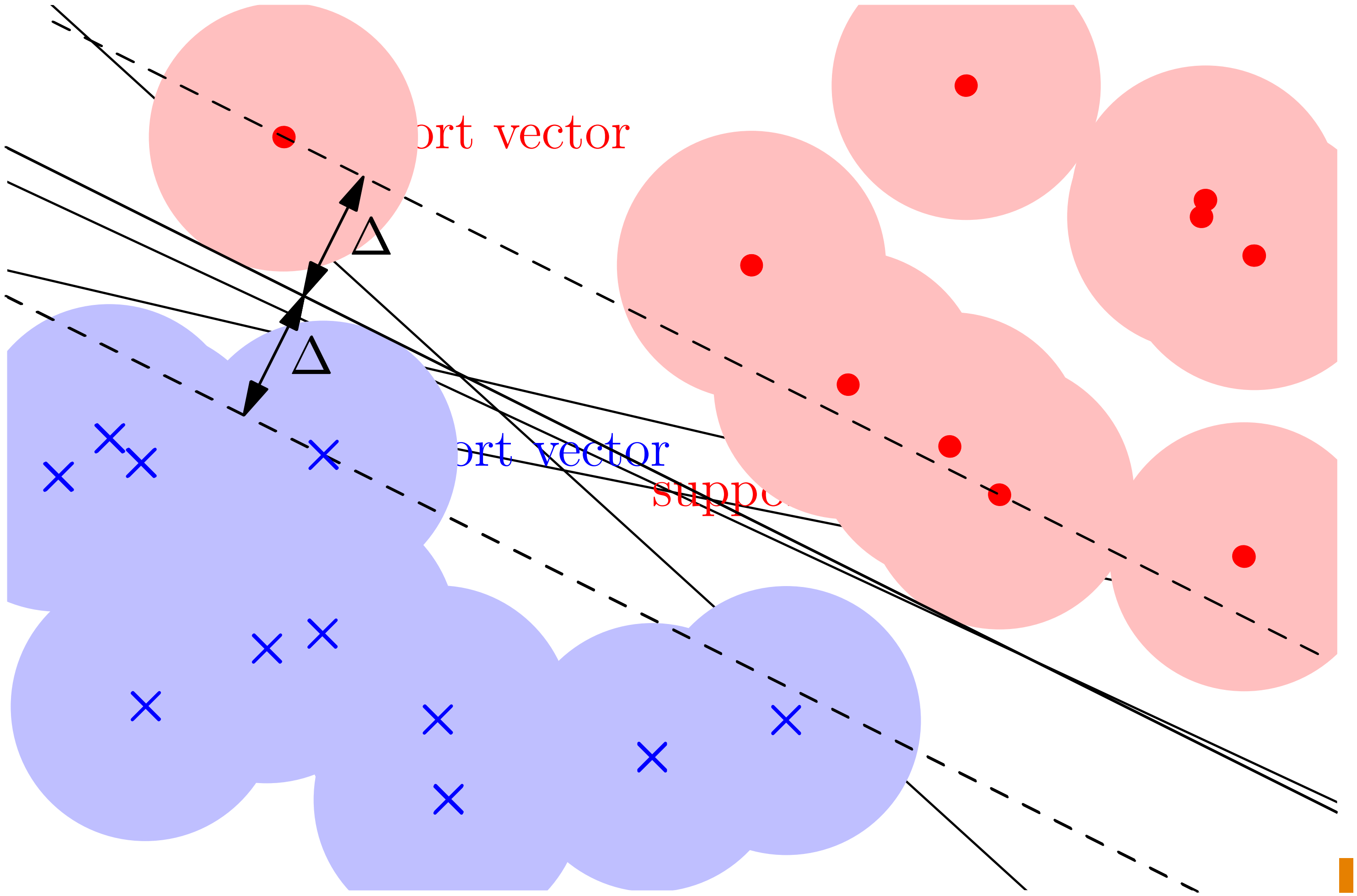


Implicit Regularisation

- In the last two examples we added an explicit regularisation term that made the function we learnt simpler
- Some learning machines do this less explicitly
- Some deep learning architectures do subtle averaging
- Sometimes the architecture biases the machine to find a simple solution

Maximum Margin Machines

- Perceptrons have many options to separate data



- SVMs choose the machine with the biggest margins

Success of SVMs

- SVMs regularise themselves by choosing the machine with the largest margin
- This ensures maximum stability to noise on the data
- It leads to very good generalisation on small datasets—usually beats everything else
- But you still need to normalise the features
- You also need to tune its hyper-parameters (C and sometimes γ)

Lessons

- Machine learning isn't magic
- It works when the learning machine is well attuned to the problem
- Sometimes you can help by preprocessing your data
- Sometimes there is a regularisation term that helps select a simpler machine
- Most machines have hyper-parameter that you tune to match the machine to the data
- Really clever machines try to do this matching automatically