Statistical Machine Learning Notes

Linear Regression

Simple linear regression

Assume a model of the form:

$$y = \tilde{x}'\tilde{w} + \epsilon$$

Assume the error follows a normal distribution:

$$\epsilon \sim N(0, \sigma^2)$$

Hence we have:

$$p(y|\tilde{x}, \tilde{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \tilde{x}'\tilde{w})^2}{2\sigma^2}\right)$$

Logistic regression

Logistic regression is a form of linear regression for predicting categorical variables. The model takes the form:

$$y = \begin{cases} 1, & f(\tilde{x}'\tilde{w}) + \epsilon > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

Where the logistic 'squashing function' is:

$$f(\tilde{x}'\tilde{w}) = \frac{1}{1 + \exp(-\tilde{x}'\tilde{w})}$$

Assume the error follows a Bernoulli distribution:

$$\epsilon \sim \text{Be}(\theta)$$

Hence we have:

$$p(y|\tilde{x}, \tilde{w}) = \left(\frac{1}{1 + \exp(-\tilde{x}'\tilde{w})}\right)^{y} \left(\frac{\exp(-\tilde{x}'\tilde{w})}{1 + \exp(-\tilde{x}'\tilde{w})}\right)^{1-y}$$

Basis expansion

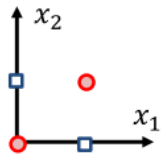
A form of data transformation used when we expect non-linear relationships in our data.

$$y = \tilde{x}'\tilde{w} \rightarrow y = \psi(\tilde{x})\tilde{w}$$

- Define transformation as

$$\varphi_i(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{z}_i\|, \text{ where } \boldsymbol{z}_i \text{ some pre-defined constants}$$

- Choose $\boldsymbol{z}_1 = [0,0]', \boldsymbol{z}_2 = [0,1]', \boldsymbol{z}_3 = [1,0]', \boldsymbol{z}_4 = [1,1]'$



there exist weights that make data separable, e.g.:

w_1	w_2	w_3	w_4
1	0	0	1

The transformed data is linearly separable!

x_1	x_2	y
0	0	Class A
0	1	Class B
1	0	Class B
1	1	Class A

φ_1	φ_2	φ_3	φ_4
0	1	1	$\sqrt{2}$
1	0	$\sqrt{2}$	1
1	$\sqrt{2}$	0	1
$\sqrt{2}$	1	1	0

$\boldsymbol{\varphi}'\boldsymbol{w}$	y
$\sqrt{2}$	Class A
2	Class B
2	Class B
$\sqrt{2}$	Class A

Polynomial basis: $\phi_1(\tilde{x}) = \tilde{x}, \phi_2(x) = \tilde{x}^2$

Radial basis function: $\phi_1(\tilde{x}) = |\tilde{x} - \tilde{z}|, \phi_2(\tilde{x}) = \exp\left(-\frac{1}{\sigma}|\tilde{x} - \tilde{z}|^2\right)$

The type and number of basis transformation functions need to be chosen beforehand, which places a limit on the flexibility of this approach. Neural networks learn the transformation automatically, and so are more flexible.

Parameter selection

There are a number of choices as to how to select the parameters $\tilde{w}$. Typically we attempt to minimise a loss function, the choice of which depends on the model.

Simple linear regression

$$\hat{w} = \text{argmin}_\theta\left(\left|\tilde{y} - \tilde{X}\tilde{w}\right|_2\right)$$
$$\hat{w} = (X'X)^{-1}X'y$$

Ridge regression

$$\hat{w} = \text{argmin}_\theta\left(\left|\tilde{y} - \tilde{X}\tilde{w}\right|_2 + \lambda|\tilde{w}|_2\right)$$
$$\hat{w} = (X'X + \lambda I)^{-1}X'y$$

Lasso regression

$$\hat{w} = \text{argmin}_\theta\left(\left|\tilde{y} - \tilde{X}\tilde{w}\right|_1 + \lambda|\tilde{w}|_1\right)$$

In lasso regression and for other more complicated models, there is no closed form solution for the optimal model parameters. Instead we have to solve numerically using gradient descent algorithms. The basic procedure here is to start with some guess for $\tilde{w}$ and then iteratively update until convergence is reached. The update rules vary from one algorithm to another.

Coordinate descent: consider one weight at a time, find component w_j which minimises $L(\theta_j)$ holding all other components fixed.

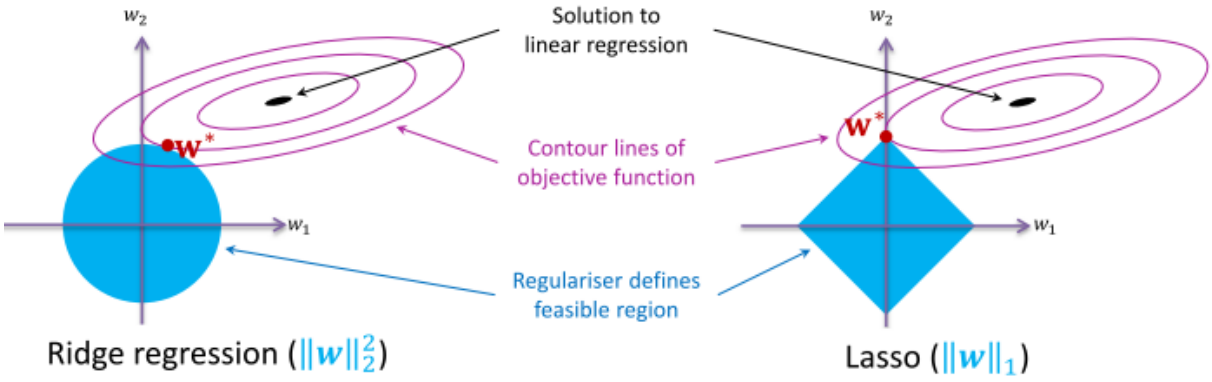
Gradient descent: consider all weights at once, update according to $\tilde{\theta}^{i+1} = \tilde{\theta}^{i} - \eta \nabla L(\tilde{\theta}^{i})$, where the parameter η is dynamically updated at each step. This assumes the function L is differentiable.

Regularisation

Without regularisation model parameters are found based entirely on the information contained in the training set X. Regularisation essentially means introducing additional information, so is equivalent to incorporating a prior over our weights.

Regularisation also can help to avoid problems intrinsic when there are irrelevant features (one feature is a linear combination of the others), or when there are more parameters than observations. In both cases, optimisation of a regression model becomes an ill-posed problem, and generally cannot be solved. The use of regularisation parameters is one way around these problems.

- For illustrative purposes, consider a *modified problem*:
  minimise $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$ *subject to* $\|\boldsymbol{w}\|_2^2 \leq \lambda$ for $\lambda > 0$



- Lasso (L1 regularisation) encourages solutions to sit on the axes

Bias variance trade-off

In supervised learning the model is trained on the training data by minimising the training error, and the generalisation capacity is then judged by computing the test error on a new dataset. Under these conditions, the expected test error decomposes into three components:

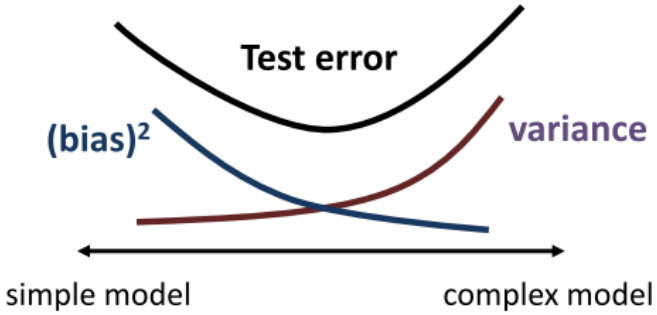
$$\mathbb{E}\left[l\left(y, \hat{f}(\boldsymbol{x}_0)\right)\right] = \left(\mathbb{E}[y] - \mathbb{E}[\hat{f}]\right)^2 + Var[\hat{f}] + Var[y]$$

test error for $\boldsymbol{x}_0$	$(\text{bias})^2$	variance	irreducible error

Note that here we treat $\hat{f}(x_0)$ as a random variable that depends on the particular sample D that we have drawn, thus variance is computed across samples. The irreducible error is due to the noise in the model from ϵ. The bias term expectations are taken with respect to ϵ and D.

In general, simpler models are more likely to incompletely capture all regularities in the data, and thus more likely to exhibit bias. More complex models are likely to overfit the data in the training set, and

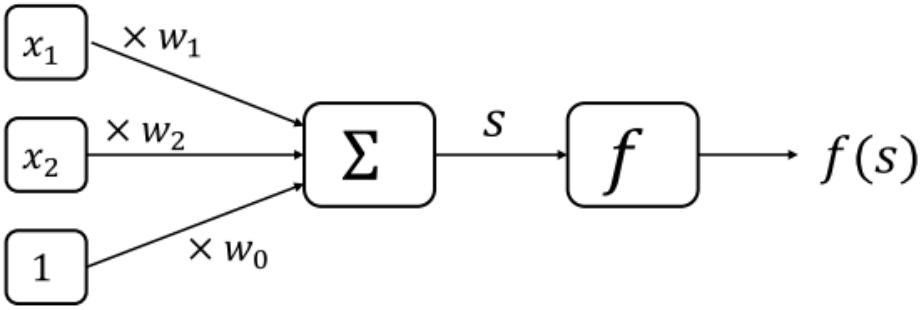
hence often exhibit higher variance between samples. There is thus often a trade-off between bias and variance that should be optimised by validation against test samples.



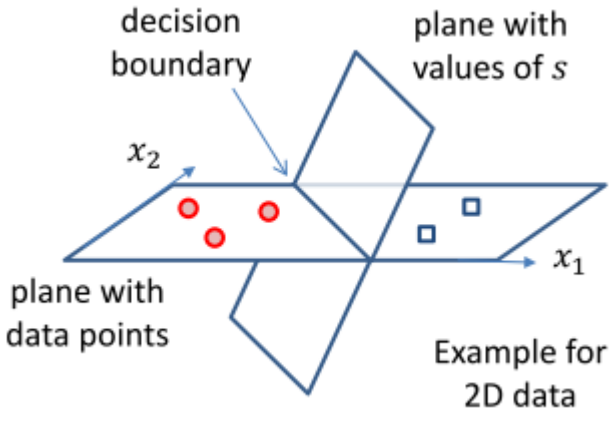
Neural Networks

The perceptron

The perceptron is a linear binary classifier, similar in operation to logistic regression. It consists of a single layer feedforward neural network.



Although the activation function in nonlinear in its argument $s = \tilde{x}'\tilde{w}$, the perceptron is a linear classifier because its decision boundary represents a hyperplane in the space of datapoints.



Decision rule for sign function $f(s)$: predict class A if $s \geq 0$, predict class B if $s < 0$.

If the data is linearly separable, the perceptron training algorithm will always converge to one of infinitely many possible solutions (separating boundaries). However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution.

A simple loss function for training the perceptron gives no penalty for correctly classified examples, and a loss equal to $|s|$ for each misclassified example.

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update*: $\boldsymbol{w}^{(k++)} = \boldsymbol{w}^{(k)} - \eta \boldsymbol{\nabla} L(\boldsymbol{w}^{(k)})$

$L(\boldsymbol{w}) = \max(0, -sy)$

$s = \sum_{i=0}^{m} x_i w_i$

η is learning rate

*There is no derivative when $s = 0$, but this case is handled explicitly in the algorithm, see next slides

If $y = 1$, but $s < 0$

$w_i \leftarrow w_i + \eta x_i$

$w_0 \leftarrow w_0 + \eta$

If $y = -1$, but $s \geq 0$

$w_i \leftarrow w_i - \eta x_i$

$w_0 \leftarrow w_0 - \eta$

Nodes in ANN can have various activation functions

Step function $f(s) = \begin{cases} 1, & if\ s \geq 0 \\ 0, & if\ s < 0 \end{cases}$

Sign function $f(s) = \begin{cases} 1, & if\ s \geq 0 \\ -1, & if\ s < 0 \end{cases}$

Logistic function $f(s) = \frac{1}{1 + e^{-s}}$

Many others: $tanh$, rectifier, etc.

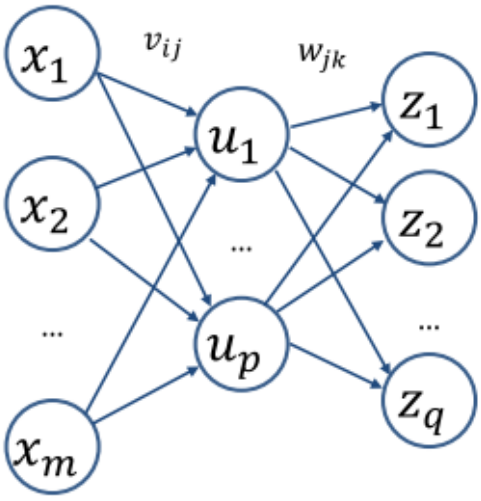
The perceptron is similar to logistic regression, in that both use the same likelihood and are usually evaluated using gradient descent. However, the gradient is taken from different functions. For a single training example, logistic regression aims to minimise negative log-likelihood, while perceptron aims to minimise a special quantity called perceptron loss. Also, logistic regression is not *necessarily* trained using gradient descent, but can be trained using algorithms that use second derivatives. In contrast, the Perceptron training algorithm is specifically stochastic gradient descent.

Multilayer perceptron

Multilayer perceptrons have one or more hidden layers in between the input and output layers. Nodes in each layer have their own activation functions and unique set of weights connecting them to each adjacent layer.

$$u_j = g(r_j)$$

$$r_j = v_{0j} + \sum_{i=1}^{m} x_i v_{ij}$$



$$z_k = h(s_k)$$

$$s_k = w_{0k} + \sum_{j=1}^{p} u_j w_{jk}$$

note that z_k is a function composition (a function applied to the result of another function, etc.)

here g, h are activation functions. These can be either same (e.g., both sigmoid) or different

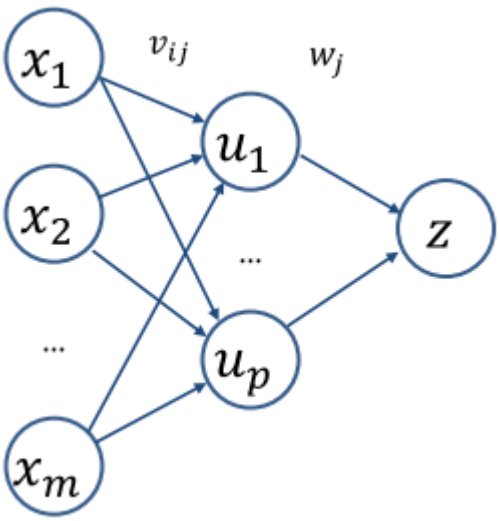
you can add bias node $x_0 = 1$ to simplify equations: $r_j = \sum_{i=0}^{m} x_i v_{ij}$

similarly you can add bias node $u_0 = 1$ to simplify equations: $s_k = \sum_{j=0}^{p} u_j w_{jk}$

Universal approximation theorem: An ANN with a hidden layer with a finite number of units, and mild assumptions on the activation function, can approximate continuous functions on compact subsets of $\mathbb{R}^n$ arbitrarily well.

The following neural network has bias nodes x_0 and u_0 that are not shown. Recalling that bias nodes do not take any input, the total number of parameters (weights) for this network is:

$$(m+1)p + (p+1) = mp + p + p + 1 = (m+2)p + 1$$



Backpropagation

A common loss function for neural networks is to use squared error between the output y and the network prediction $z = \hat{f}(\tilde{x}, \tilde{\theta})$, where $\tilde{\theta} = (\tilde{v}, \tilde{w})$:

$$L = \frac{1}{2}\left(\hat{f}(\tilde{x}, \tilde{\theta}) - y\right)^2$$

There is no general analytic solution to this, so we use a gradient descent method.

Choose initial guess $\boldsymbol{\theta}^{(0)}$, $k = 0$

Here $\boldsymbol{\theta}$ is a set of all weights form all layers

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update: $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \nabla L(\boldsymbol{\theta}^{(i)})$

Note the sign: if the partial derivative $\frac{\partial L}{\partial \theta_j} < 0$, then the loss shrinks as θ_j increases, so we should increase that parameter, as in fact occurs because of the double minus signs. If $\frac{\partial L}{\partial \theta_j} > 0$ then the loss increases as θ_j increases, so we should decrease that parameter, as also occurs thanks to the single minus sign.

In order to use this training method it is necessary to calculate the partial derivative of the loss function with respect to each model parameter. This is complicated in multilayer networks by the fact that the loss function depends only indirectly on the weights at earlier stages in the network – this is called the credit assignment problem. The solution to this problem is to use backpropagation, which is essentially an application of the chain rule.

$$\begin{aligned}
L &= \frac{1}{2}(z - y)^2 \\
&= \frac{1}{2}(h(s) - y)^2 \\
&= \frac{1}{2}\left(h\left(\sum u_j w_j\right) - y\right)^2 \\
&= \frac{1}{2}\left(h\left(\sum g(r_j) w_j\right) - y\right)^2 \\
L &= \frac{1}{2}\left(h\left(\sum g\left(\sum x_i v_{ij}\right) w_j\right) - y\right)^2
\end{aligned}$$

Thus we have the partial derivatives:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$$

$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$$

Hence:

$$\frac{\partial L}{\partial w_j} = (z-y)g'(s)\frac{\partial s}{\partial w_j}$$
$$\frac{\partial L}{\partial v_{ij}} = (z-y)g'(s)\frac{\partial s}{\partial u_j}h'(r_j)\frac{\partial r_j}{\partial v_{ij}}$$

We also compute:

$$\frac{\partial s}{\partial w_j} = \frac{\partial}{\partial w_j}u_j w_j = u_j$$
$$\frac{\partial s}{\partial u_j} = \frac{\partial}{\partial u_j}u_j w_j = \frac{\partial}{\partial u_j}u_j w_j = w_j$$
$$\frac{\partial r_j}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}}x_i v_{ij} = x_i$$

Yielding the backpropagation equations in the identity activation function case:

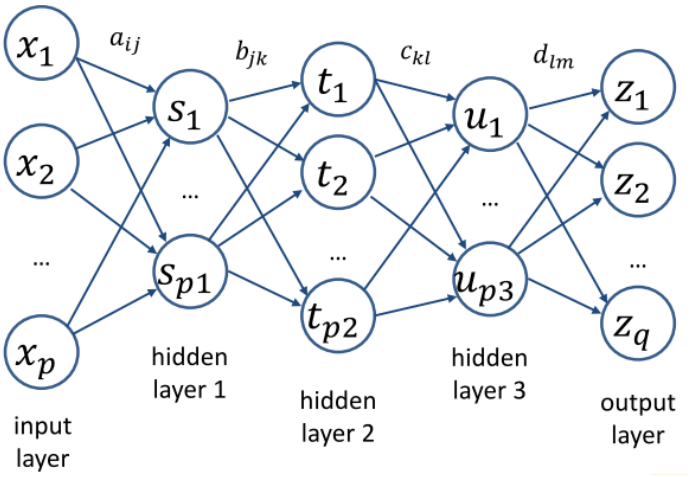
$$\frac{\partial L}{\partial w_j} = (z-y)g'(s)u_j$$
$$\frac{\partial L}{\partial v_{ij}} = (z-y)g'(s)w_j h'(r_j)x_i$$

The idea then is that errors are propagated backwards through the network, and are then used to compute the partial derivatives, which in turn are used to update the weights.

The flexibility of neural network models means they are liable to overfit. This tendency can be avoided by implicit regularisation in the form of 'early stopping' of training, or through explicitly including a regularisation term in the loss function to drive the weights to zero.

Deep learning

While any Boolean function over m variables can be implemented using a single hidden layer with up to 2^m elements, it is often more efficient to stack several hidden layers to form a deep network.

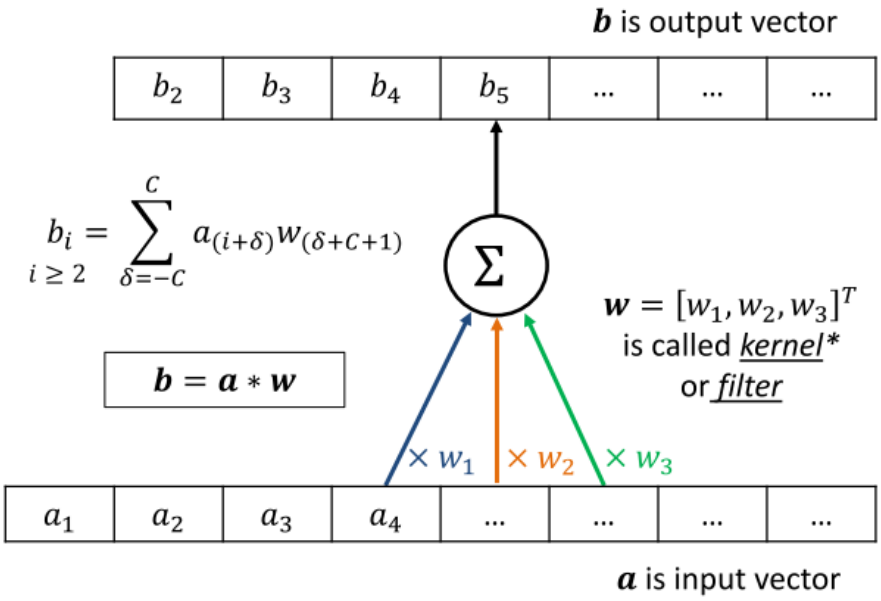


Each hidden layer can be thought of as a transformation of the underlying feature space.

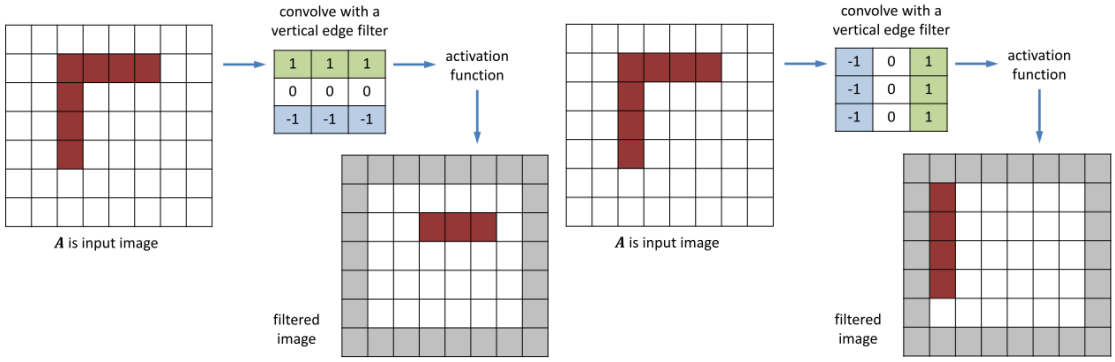
One problem with deep networks is the vanishing gradient problem: partial derivatives get very small moving through many layers.

Convolutional neural networks

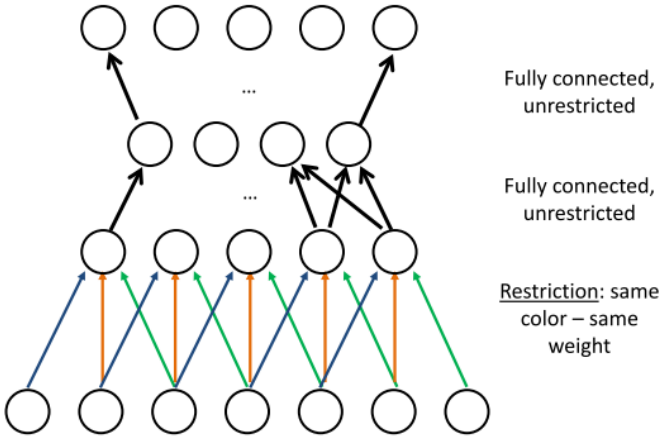
Convolutional neural networks are hierarchically structured such that each output unit in one layer only has a limited input layer (called the receptive field) from the previous layer, which is passed through a kernel or filter function.



Careful selection of filer functions allows one to detect the presence of particular features, such as edges in images.

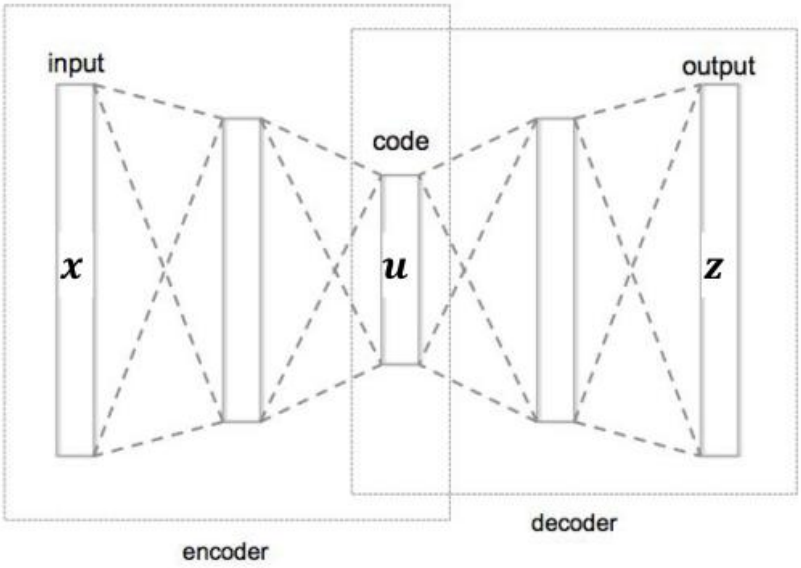


Usually the representations are then passed through further, fully-connected hidden layers so as to merge representations together before passing them through the output.



Autoencoders

An autoencoder is a special type of network that is designed to predict its own input as output. This would be pointless if it weren't for the process of passing the network through a 'bottleneck' which aims to ensure the compression of the input with minimal information loss.



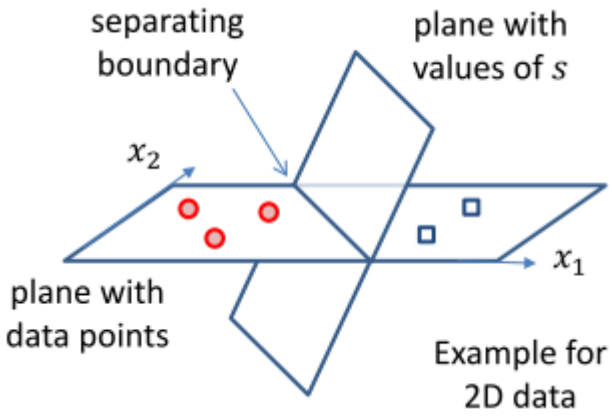
Autoencoders can be used for compression and dimensionality reduction via a non-linear transformation.

Support Vector Machines

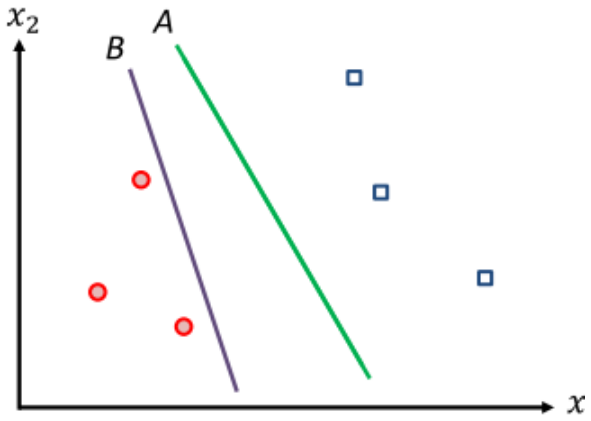
Maximum margin classifiers

A support vector machine is a linear binary classifier that works by finding a hyperplane to separate two classes in a dataset.

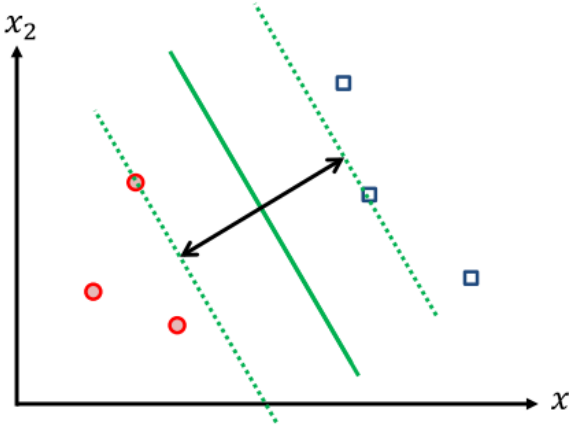
Predict class A if $s \geq 0$
Predict class B if $s < 0$
where $s = b + \sum_{i=1}^{m} x_i w_i$



In fact hard margin support vector machines work exactly like a perceptron, except that they optimise a different objective function in choosing the parameters. For the perceptron, all linear boundaries that separate the two classes are equally good, because the perceptron loss is zero for each of them. In the example below, however, it seems clear that line A is a better choice than line B.



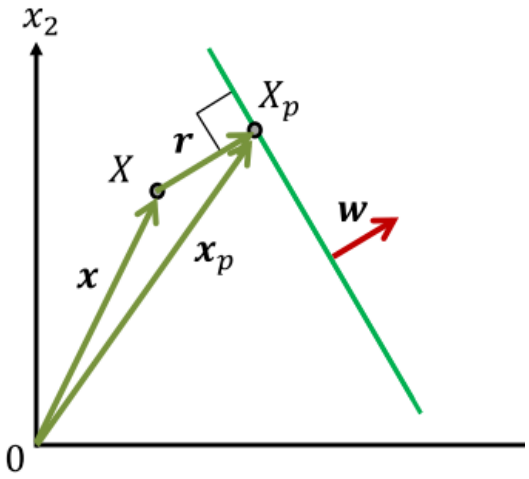
The support vector machine formalises this notion by finding the separating boundary that maximises the margin between classes.



The points on the margin boundaries (the dotted lines) in the figure above are called the support vectors (they are a vector of observation variables). They play an important role in defining the margin width.

Deriving the objective

The key goal of hard margin SVMs is to find the hyperplane with the maximum distance to the support vectors. To define the needed concepts, consider the figure below. Let x be an arbitrary vector of data (in either class, not necessary a support vector), and let x_p be the projection of this vector onto the separating boundary. Let r be the vector $x_p - x$, whose length $|r|$ is the margin width we are looking to calculate. Finally the weights in the classification expression $s = \sum x_i w_i$ themselves represent a vector which always is perpendicular to the separating boundary, and hence is parallel to r. To see this, note that the decision boundary is the x such that $x \cdot w = 0$, meaning the two are perpendicular.



We thus have the relation between the parallel or anti-parallel vectors:

$$r = w\frac{|r|}{|w|}$$

Substituting in the value of r:

$$x_p - x = w\frac{|r|}{|w|}$$
$$x_p = w\frac{|r|}{|w|} + x$$
$$w'x_p = w'w\frac{|r|}{|w|} + w'x$$
$$w'x_p + b = w'w\frac{|r|}{|w|} + w'x + b$$

Since x_p lies on the boundary obviously $w'x_p + b = s = 0$, and so:

$$w'w\frac{|r|}{|w|} + w'x + b = 0$$
$$|w|^2\frac{|r|}{|w|} = -(w'x + b)$$
$$|r| = -\frac{w'x + b}{|w|}$$

Turns out we have to add a sign ambiguity since r and w could be anti-parallel if x was on the other side of the decision boundary:

$$|r| = \pm\frac{w'x + b}{|w|}$$

Alternatively we can use the class labels $y_i = \pm 1$ to do this:

$$|r_i| = \frac{y_i(w'x_i + b)}{|w|}$$

So effectively hard-margin SVMs have the following objective:

$$\max_{w}\left[\min_{i=1,..,n}\left[\frac{y_i(w'x_i+b)}{|w|}\right]\right]$$

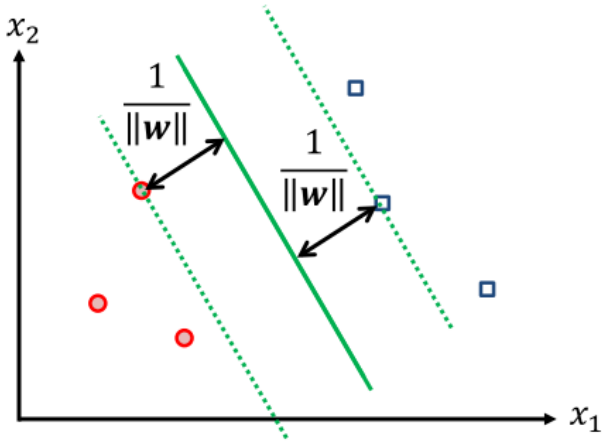
Unfortunately, however, there is an ambiguity to this algorithm; there are infinitely many solutions because $\widetilde{w}=\alpha w$ will also be a solution for any α. To get around this we need to introduce an arbitrary scaling convention:

$$\frac{y_i(w'x_i+b)}{|w|}=\frac{1}{|w|}$$

So now our objective becomes:

$$\underset{w}{\operatorname{argmin}}|w| \text{ s.t. } y_i(w'x_i+b)\geq 1 \text{ for } i\in\{1\ldots n\}$$

This means we pick the w whose length is the smallest, subject to the condition that every point is on or outside of the margins.



These constraints can be interpreted as the following loss function:

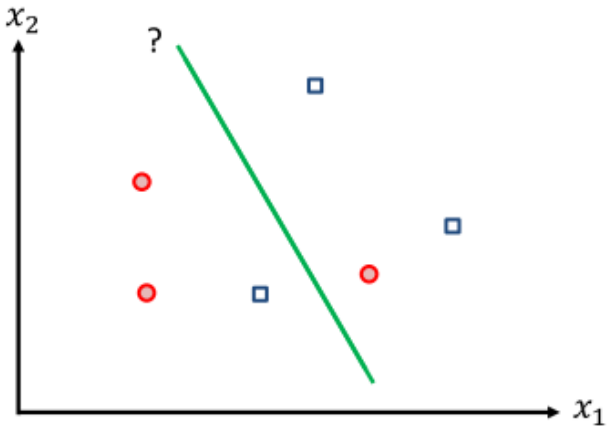
$$l_\infty=\begin{cases}0 & 1-y_i(\boldsymbol{w}'\boldsymbol{x}_i+b)\leq 0\\ \infty & 1-y_i(\boldsymbol{w}'\boldsymbol{x}_i+b)>0\end{cases}$$

In other words, for each point:

* If it's on the right side of the boundary and at least $\frac{1}{\|w\|}$ units away from the boundary, we're OK, the loss is 0
* If the point is on the wrong side, or too close to the boundary, we immediately give infinite loss thus prohibiting such a solution altogether

Soft margin classifiers

When the data is not linearly separable, the hard margin SVM approach will not work.



The two main methods of dealing with this problem are to use soft margin SVMs, or to transform the data. Here we consider the first approach.

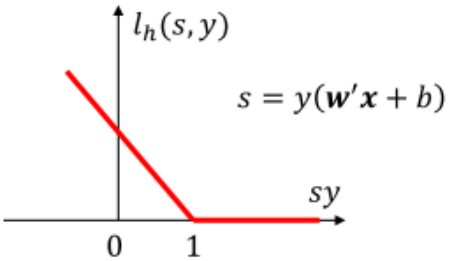
In the soft margin SVM formulation we relax the constraints to allow points to be inside the margin or even on the wrong side of the boundary. However, we penalise boundaries by the amount that reflects the extent of "violation".

- Hard margin SVM loss

$$l_\infty = \begin{cases} 0 & 1 - y(\boldsymbol{w}'\boldsymbol{x} + b) \leq 0 \\ \infty & otherwise \end{cases}$$

- Soft margin SVM loss (hinge loss)

$$l_h = \begin{cases} 0 & 1 - y(\boldsymbol{w}'\boldsymbol{x} + b) \leq 0 \\ 1 - y(\boldsymbol{w}'\boldsymbol{x} + b) & otherwise \end{cases}$$



compare this with perceptron loss

We can rewrite the soft margin objective in terms of slack variables ξ_i which represent a point being on the wrong side of the margins:

$$\underset{w,\xi}{\operatorname{argmin}} \left(\frac{1}{2}|w|^2 + C \sum_{i=1}^{n} \xi_i \right) \text{ s.t. } \xi_i \geq 1 - y_i(w'x_i + b) \text{ and } \xi_i \geq 0 \text{ for } i \in \{1 \dots n\}$$

Solving the optimisation

Training a SVM simply means solving the corresponding optimisation problem to find w. Here we focus on training a hard margin SVM. Since gradient descent methods cannot be used to solve a constrained optimisation problem, we instead use the method of Lagrangian multipliers, also called KKT method.

To this end, we first define the Lagrangian/KKT objective

$$L_{KKT}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i \left(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1\right)$$

primal objective — constraints

It turns out that if w^* and b^* is a solution of the primal hard margin SVM problem, then there exists λ^* such that all three together satisfy the KKT conditions.

The corresponding KKT conditions are:

$$y_i\left((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*\right) - 1 \geq 0 \text{ for } i = 1, \dots, n$$

this just repeats constraints, trivial

$$\lambda_i^* \geq 0 \text{ for } i = 1, \dots, n$$

require non-negative multipliers in order for the theorem to work

$$\lambda_i^*\left(y_i\left((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*\right) - 1\right) = 0$$

"complementary slackness", we'll come back to that

$$\nabla_{\boldsymbol{w},b} L_{KKT}(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}^*) = 0$$

zero gradient, somewhat similar to unconstrained optimisation

We thus have the Lagrangian:

$$L(w, b, \lambda) = \frac{1}{2}|w|^2 - \sum_{i=1}^{n} \lambda_i (y_i(w'x_i + b) - 1)$$

Taking partial derivatives:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0$$

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^{n} \lambda_i (y_i x_{ij}) = 0$$

Substituting these conditions into the Lagrangian we obtain:

$$L(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i' x_j$$

Choosing the positive set of λ_i which maximises this Lagrangian dual problem will therefore give us the solution to the original problem. In summary therefore we have found:

Training: finding $\boldsymbol{\lambda}$ that solve

$$\underset{\boldsymbol{\lambda}}{\text{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

$$\text{s.t. } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i y_i = 0$$

Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

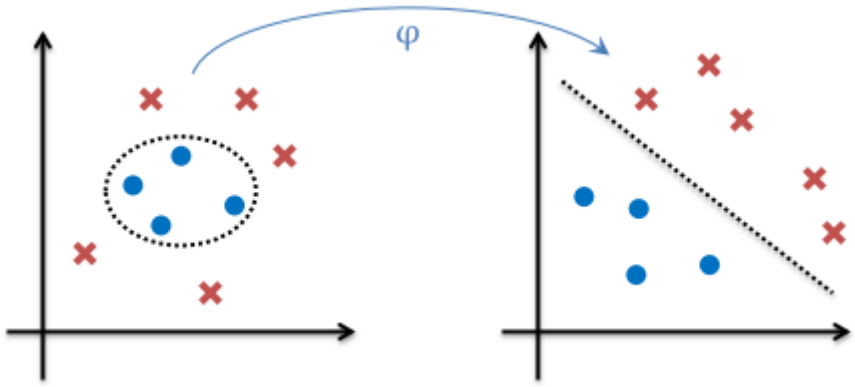
$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}$$

Note that the terms in the above sum are non-zero only for support vectors. Thus the prediction is made essentially by taking the dot-product of each new point with the set of support vectors.

The SVM Lagrangian dual problem is a quadratic optimisation problem. Using standard algorithms this problem can be solved in $O(n^3)$. One algorithm called chunking exploits the fact that many of the λs will be zero since this is true for all points outside the margins.

Kernel methods

Kernel methods are a way of performing a feature space transformation on the original data. If we apply a SVM to the transformed data, our model is still linear in the transformation space, but it will be non-linear in the original space, allowing us to capture more complex relations.



One problem with simply applying a transformation to each data point is that it is impractical to compute $\psi(x)$ for each point with very high dimensional data. We can get around this, however, by noting (see slide above) that the data x only appears in both training and prediction equations in the form of the dot product $x'x$. This means that we never actually need to compute $\psi(x)$, but only $\psi(x)'\psi(x)$. This naturally gives rise to the definition of a kernel:

- Kernel is a function that can be expressed as a dot product in some feature space $K(\boldsymbol{u}, \boldsymbol{v}) = \varphi(\boldsymbol{u})'\varphi(\boldsymbol{v})$

Training: finding $\boldsymbol{\lambda}$ that solve

feature mapping is implied by kernel

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$\text{s.t. } \lambda_i \geq 0 \text{ and } \textstyle\sum_{i=1}^{n} \lambda_i y_i = 0$$

Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

feature mapping is implied by kernel

Advantages of using kernels include:

- Saved computational time in not having to compute $\psi(x)$ for all observations
- Ability to extend method to infinite-dimensional data
- Kernels can be applied to objects that are not vectors, such as graphs, sequences, even movies; it serves as a general similarity measure

If K is a kernel then the following are also valid kernels:

* $K(\boldsymbol{u}, \boldsymbol{v}) = K_1(\boldsymbol{u}, \boldsymbol{v}) + K_2(\boldsymbol{u}, \boldsymbol{v})$
* $K(\boldsymbol{u}, \boldsymbol{v}) = cK_1(\boldsymbol{u}, \boldsymbol{v})$
* $K(\boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{u})K_1(\boldsymbol{u}, \boldsymbol{v})f(\boldsymbol{v})$

The Representer Theorem states that a large class of linear methods can be formulated (represented) such that both training and making predictions require data only in a form of a dot product. This includes:

- Hard margin support vector machine
- Ridge regression
- Logistic regression
- Perceptron
- Principal components analysis

One of the advantages of the representer theorem is that is highlights the fact that all information about feature mapping is contained within the kernel. The algorithm this decouples into choose a learning method (e.g. SVM vs logistic regression) and then choosing a feature space mapping (i.e. a kernel).

Examples of kernels

- Polynomial kernel: $K(u,v) = (u'v + c)^d$, where d is the integer order of the polynomial
- Radial basis function kernel: $K(u,v) = \exp(-\gamma|u-v|^2)$, where γ is a spread parameter

Mercer's theorem states that any finite sequence of vectors arranged in an $n \times n$ matrix of pairwise values $K(x_i, x_j)$ will be a kernel if the matrix is positive semidefinite, and this olds for all possible sequences.

Bagging

Bagging is a method of constructing 'novel' datasets by resampling with replacement from our actual data set. The idea is to generate k new datasets each of the same size as our original set, then build a classifier on each set separately and combine predictions via voting. The purpose of this is that including more independent observations should reduce prediction variance.

Original training dataset:
$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Bootstrap samples:
$\{7, 2, 6, 7, 5, 4, 8, 8, 1\ 0\}$ – out-of-sample $3, 9$
$\{1, 3, 8, 0, 3, 5, 8, 0, 1, 9\}$ – out-of-sample $2, 4, 6, 7$
$\{2, 9, 4, 2, 7, 9, 3, 0, 1, 0\}$ – out-of-sample $3, 5, 6, 8$

At each round of selection, a particular datum has a probability of $\left(1 - \frac{1}{n}\right)$ of NOT being selected. Thus the probability of that observation being left out of the new bootstrapped sample altogether is: $\left(1 - \frac{1}{n}\right)^n$. In the limit of large n, this approaches $\frac{1}{e} = 0.368$, so over a third of our data won't be used in each training dataset we produce. We can use this excluded data for cross-validation.

Bagging is typically an effective method to reduce variance, and the performance is generally significantly better than the base classifiers but never substantially worse

Clustering and EM Algorithm

Unsupervised learning

Unsupervised learning differs from supervised learning in that there are no specified data labels that we try to predict. Instead, the goal is simply to find patterns and structure in the data.

Common applications of unsupervised learning:

- Clustering
- Dimensionality reduction
- Probabilistic graphical models
- Outlier detection

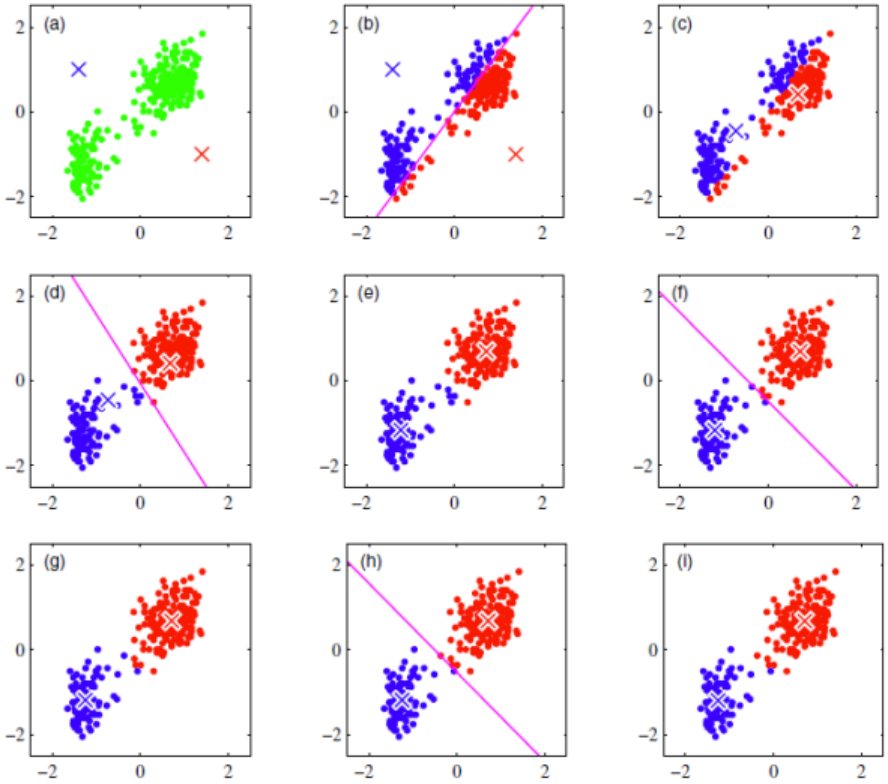
K-means clustering

Clustering is automatic grouping of objects such that the objects within each group (cluster) are more similar to each other than objects from different groups. In order to do this we need a measure of similarity, or as is often used instead a measure of dissimilarity. One common method is Euclidean distance:

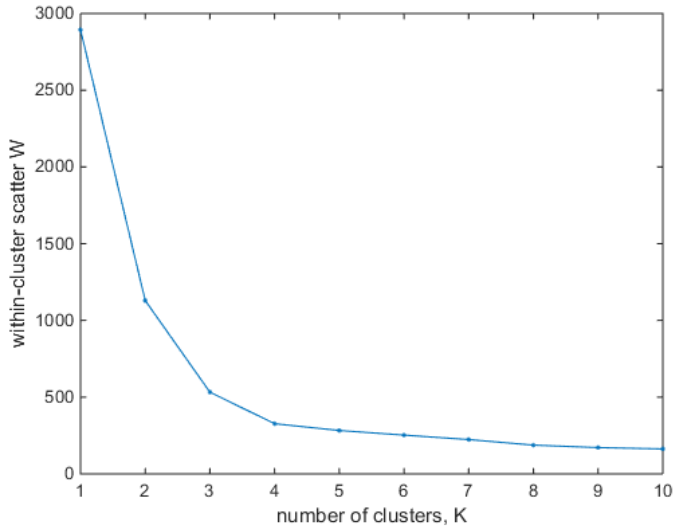
$$d_{uv} = \sqrt{\sum_{l=1} (\tilde{u}_l - \tilde{v}_l)^2}$$

K-means is a very popular iterative clustering algorithm, which requires specifying the number of clusters in advance. It proceeds as follows:

1. Initialisation: choose k cluster centroids randomly
2. Update:
   a) Assign points to the nearest centroid
   b) Compute centroids under the current assignment
3. Termination: if no change then stop
4. Go to Step 2

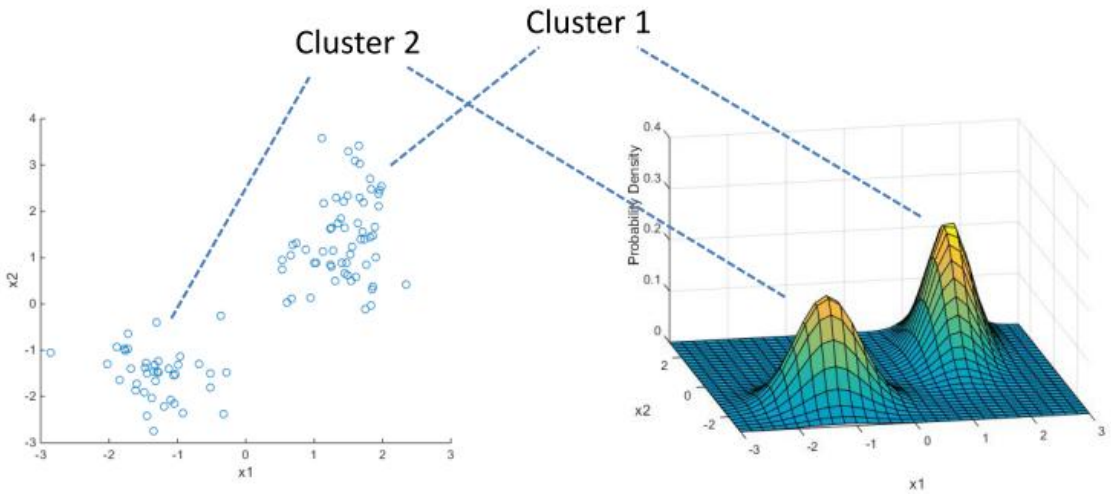


Higher values of k will always result in better fit of the data, since they permit a more flexible model. The 'kink method' is a simple way of deciding the value of k, which involves plotting the intra-cluster variation against the number of clusters, and locating the 'kink'. More abstract information-theoretic methods can also be used to determine k.



Gaussian mixture model

GMM clustering is a generalisation of k-means which applies a probabilistic approach. The key insight is to regard cluster centers as nodes of different distributions which generated each cluster.



While still requiring the number of clusters/distributions to be set in advance, GMM does not require each point be assigned to exactly one cluster. Instead, we assign a particular point to multiple clusters each with some probability.

We typically take each of the component probability distributions to be a multivariate Gaussian:

$$N(\tilde{x}|\tilde{\mu},\Sigma) = (2\pi)^{-\frac{m}{2}}(\det\Sigma)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(\tilde{x}-\tilde{\mu})'\Sigma^{-1}(\tilde{x}-\tilde{\mu})\right)$$

For n-dimensional data and k clusters we will have $n \times k$ mean parameters, $\frac{n(n+1)}{2} \times k$ covariance parameters, and $k-1$ weight parameters. Using this method, the probability that an observation occurs at point $\tilde{x}$ is equal to the weighted sum over the k Gaussians:

$$p(X=\tilde{x}) = \sum_{c=1}^{k} w_c N(\tilde{x}|\tilde{\mu}_c,\Sigma_c)$$

To solve this problem using the standard maximum likelihood approach, therefore, we aim to find the set of parameters $\tilde{w}_c, \tilde{\mu}_c, \Sigma_c$ for $c \in \{1, k\}$ that maximise the joint log likelihood:

$$p(\tilde{x}_1, \dots, \tilde{x}_n) = \sum_{i=1}^{n} \log \left(\sum_{c=1}^{k} w_c \, N(\tilde{x} | \tilde{\mu}_c, \Sigma_c) \right)$$

Notice that unlike the usual log likelihood expression, we have an inner summation term (over the multiple Gaussian centers) which we cannot take outside the logarithm. This makes finding an analytical solution impossible. Instead we usually solve a GMM using a method called the Expectation Maximisation algorithm.

Expectation Maximisation algorithm

Expectation Maximisation (EM) is a generic algorithm for optimising the parameters of the log likelihood, even when no analytic solution is available. It is often used for the Gaussian mixed model clustering method, but can be used in many other machine learning methods too. The key insight is to introduce a series of latent (unobserved) variables Z which can make calculations easier.

$$\begin{aligned}
\log p(\tilde{x}_1, \dots, \tilde{x}_n | \hat{\theta}) &= \log \sum_z p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}) \\
&= \log \sum_z p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}) \frac{p(\tilde{z})}{p(\tilde{z})} \\
&= \log \sum_z p(\tilde{z}) \frac{p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta})}{p(\tilde{z})} \\
&= \log E_z \left[\frac{p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta})}{p(\tilde{z})} \right] \\
&\geq E_z \left[\frac{p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta})}{p(\tilde{z})} \right] \\
&\geq E_z \left[\log p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}) \right] - E_z [p(\tilde{z})] \\
\log p(\tilde{x}_1, \dots, \tilde{x}_n | \hat{\theta}) &= E_z \left[\log p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}) \right] - E_z \left[\log p(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}) \right]
\end{aligned}$$

This gives rise to the following iterative optimisation scheme (to find local maximum only):

EM as iterative optimisation

1. Initialisation: choose initial values of $\boldsymbol{\theta}^{(1)}$
2. Update:
   - E-step: compute $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \equiv \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{(t)}} [\log p(\boldsymbol{X}, \boldsymbol{Z} | \boldsymbol{\theta})]$
   - M-step: $\boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$
3. Termination: if no change then stop
4. Go to Step 2

This algorithm will eventually stop (converge), but the

Note that most of the work is done in the maximisation step. The initial expectation step computes all the parameters (specifically r_{ic}) that are needed to fully evaluate the derivatives used in the M-step, but does not involve any actual maximisation itself.

Applying the EM algorithm to GMM

The latent variables in GMM are taken to be the true cluster from which each observation x_i was generated. Thus $z_i \in \{1 \dots k\}$ for k-clusters. If we knew all the zs, we would not need to sum over all possible clusters, since obviously we would already know which cluster each observation came from. Our modified likelihood would then take the form:

$$\log p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z}) = \sum_{i=1}^{n} \log\left(w_{z_i} N(\tilde{x}_i | \tilde{\mu}_{z_i}, \Sigma_{z_i}\right)$$

Unlike the original log likelihood, this expression can be maximised analytically, so long as we know the values of $\tilde{z}$. Since we don't actually know the true cluster assignments, we can instead use the expectation (where $\tilde{\theta}$ are all the GMM model parameters, and the sum is over all possible allocations of xs to clusters):

$$E_z\left[\log p\left(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}\right)\right] = \sum_{z} p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right) \log p(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z})$$

Since we need a definition of $p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)$, we use what is called the *responsibility* r_{ic} that cluster c takes for data point i:

$$r_{ic} = p\left(z_i = c | \tilde{x}_i, \tilde{\theta}\right) = \frac{w_c N(\tilde{x}_i | \tilde{\mu}_c, \Sigma_c)}{\sum_l w_l N(\tilde{x}_i | \tilde{\mu}_l, \Sigma_l)}$$

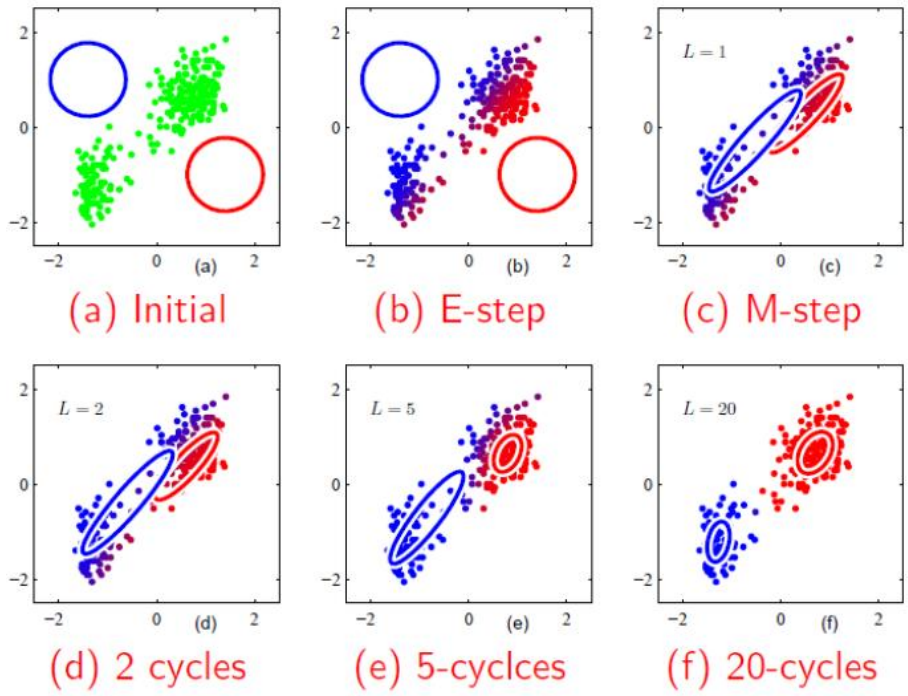
And so for the joint distribution (assuming all data points are independent):

$$\sum_{z} p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right) = \sum_{i=1}^{n} \sum_{c=1}^{k} p\left(z_i = c | \tilde{x}_i, \tilde{\theta}\right) = \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic}$$

Now we are ready to evaluate the log likelihood using the equality derived previously:

$$\begin{aligned}
\log p\left(\tilde{x}_1, \dots, \tilde{x}_n | \hat{\theta}\right) &= E_z\left[\log p\left(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}\right)\right] - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right] \\
&= \sum_{z} p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right) \log p\left(\tilde{x}_1, \dots, \tilde{x}_n, \tilde{z} | \hat{\theta}\right) - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right] \\
&= \sum_{z} p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right) \sum_{i=1}^{n} \log\left(w_{z_i} N(\tilde{x}_i | \tilde{\mu}_{z_i}, \Sigma_{z_i}\right) - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right] \\
&= \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log\left(w_{z_i} N(\tilde{x}_i | \tilde{\mu}_{z_i}, \Sigma_{z_i}\right) - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right] \\
&= \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log\left(w_{z_i} N(\tilde{x}_i | \tilde{\mu}_{z_i}, \Sigma_{z_i}\right) - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right] \\
\log p\left(\tilde{x}_1, \dots, \tilde{x}_n | \hat{\theta}\right) &= \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i} + \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic}\, N\left(\tilde{x}_i | \tilde{\mu}_{z_i}, \Sigma_{z_i}\right) - E_z\left[\log p\left(\tilde{z} | \tilde{x}_1, \dots, \tilde{x}_n, \tilde{\theta}\right)\right]
\end{aligned}$$

We then take partial derivatives to maximise this expression with respect to w, μ and Σ. The new values of the parameters are then used in the next step of the iteration to calculate new values of r_{ic}, which are then used for another maximisation step, and so on.



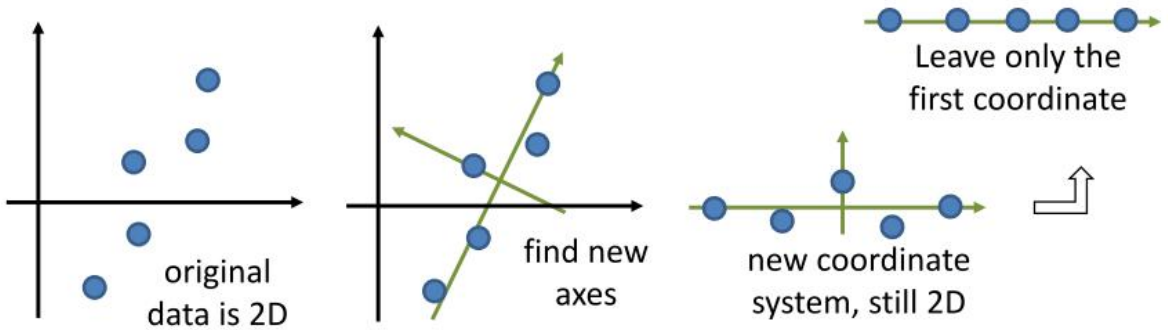
It turns out that K-means is a special case of GMM in which all components have the fixed probability $w_c = 1/k$, and in which each Gaussian has a fixed covariance matrix $\Sigma_c = \sigma^2 I$.

Dimensionality Reduction

Principle components analysis

The purpose of dimensionality reduction is to reduce the number of variables/dimensions in the data while still preserving the structure of interest. This can aid in visualisation, computational efficiency, and data storage compression.

Principal components analysis is a popular method for achieving dimensionality reduction. Given a dataset, PCA aims to find a new coordinate system such that most of the variance is concentrated in the first coordinate, and most of the remainder in the second coordinate, and so forth. Most of the dimensions are then discarded, leaving a reduced number of dimensions that still preserve most of the variance.



Consider our new coordinate system as a set of vectors $p_1, \dots, p_m$ each with unit length. To transform a single original data point into our new coordinate system, we use the transformation: $y_1^i = p_i' x_1$. If we put all the original data points into columns we have $X = (\tilde{x}_1, \dots, \tilde{x}_n)$, and then the ith coordinate of all the new variables is given by:

$$Y_i = X' p_i$$

We need to find a method for determining all the $\tilde{p}$ vectors. To do this we can use the covariance matrix of the centered data, which is simply:

$$\Sigma_X = \frac{1}{n-1} XX'$$

We want to choose p_1 such that it minimises the variance of the transformed data. This variance is:

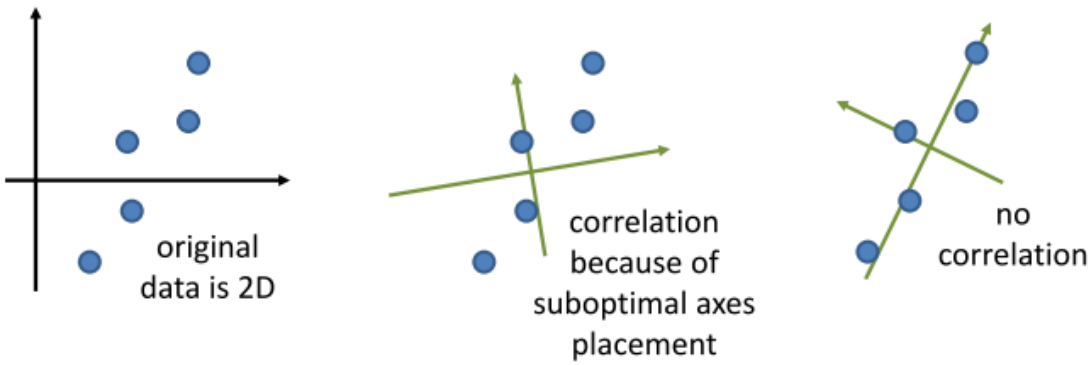
$$\begin{aligned}
\Sigma_{Y_1} &= \frac{1}{n-1} Y_1' Y_1 \\
&= \frac{1}{n-1} (X' p_1)' (X' p_1) \\
&= \frac{1}{n-1} p_1' X X' p_1 \\
\Sigma_{Y_1} &= p_1' \Sigma_X p_1
\end{aligned}$$

We want to find the p_1 that maximises this variance Σ_{Y_1} subject to p_1 being of unit length. We can solve this constrained optimisation using a Lagrangian:

$$\begin{aligned}
\mathcal{L} &= p_1' \Sigma_X p_1 - \lambda_1 (p_1' p_1 - 1) \\
\frac{\partial \mathcal{L}}{\partial p_1} &= 2\Sigma_X p_1 - 2\lambda_1 p_1 \\
\Sigma_X p_1 &= \lambda_1 p_1
\end{aligned}$$

Since $\lambda_1 = p_1' \Sigma_X p_1 = \Sigma_{Y_1}$ and we know that p_1 has the largest variance of all the coordinates (by design), it follows that p_1 is simply the eigenvector of the centered covariance matrix with the largest eigenvalue. We find all the other vectors by the same process, adding the requirement that they be orthogonal to all previous solutions. This is always possible since a symmetric $m \times m$ matrix always has m real eigenvalues.

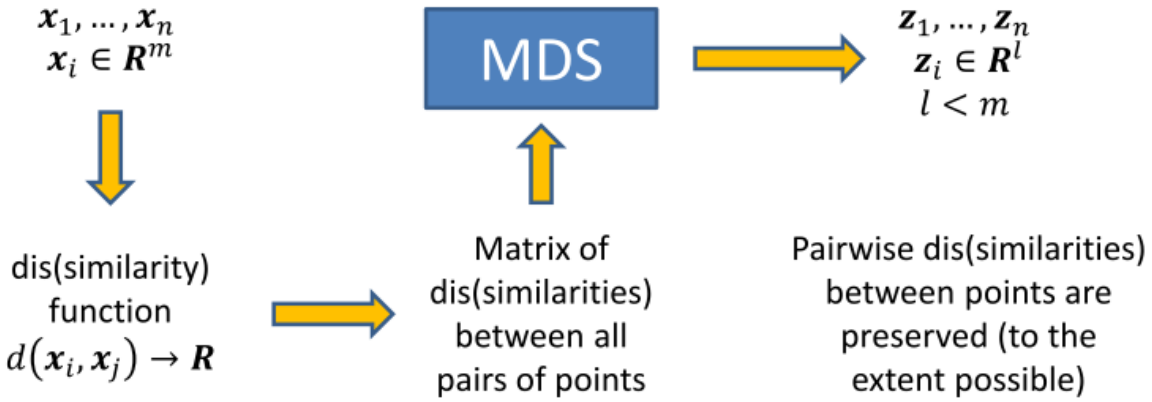
One other benefit of PCA is that it should result in coordinates which are uncorrelated with each other.



An extension of regular PCA is to use kernel PCA, in which a more complex kernel function is used in place of $X'X$ to capture nonlinear variations.

Multidimensional scaling

Multidimensional scaling is a non-linear dimensionality reduction method which seeks to map data to a lower-dimensional space preserving pairwise differences (dissimilarities) as much as possible. How exactly dissimiliarity is defined, and also how 'preservation' is defined, distinguishes distinct instantiations of MDS from each other.



The degree of preservation of dissimilarity is measured using a stress function. One possible definition in terms of differences d is given by:

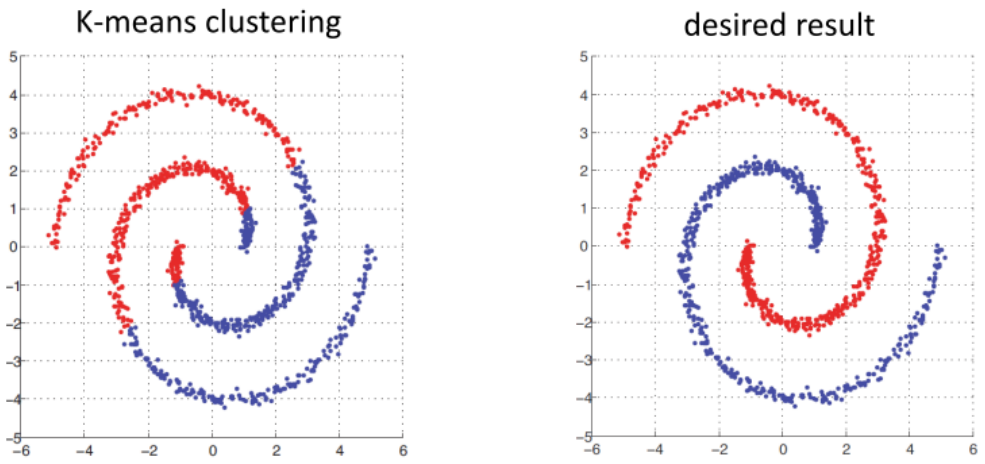
$$S(z_1, \dots, z_n) = \frac{\sum_{ij} \left(d(x_i, x_j - d(z_i, z_j)\right)^2}{\sum_{ij} d(z_i, z_j)^2}$$

The idea is to find the zs such that $S(z_1, \dots, z_n)$ is minimised. The logic behind MDS is that if there are genuine clusters in high dimensional data, then points within these clusters are close to each other, while points from different clusters are far away. MDS attempts to preserve this distance structure, so that clusters are preserved in the low dimensional map.

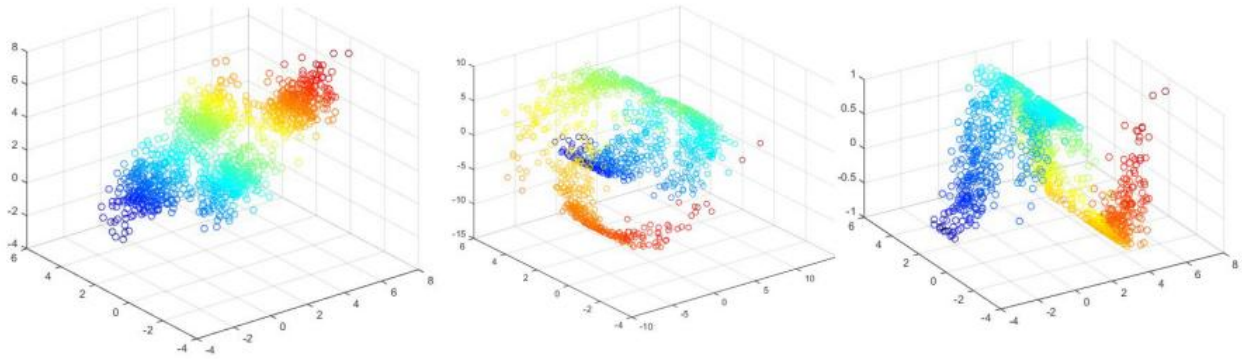
One additional application of MDS is to use it for producing a set of coordinates given just the pairwise dissimilarities between various instances. This could be used, for example, to provide a mapping of the characteristics of different movies based only on pairwise similarity ratings by viewers.

Manifold learning

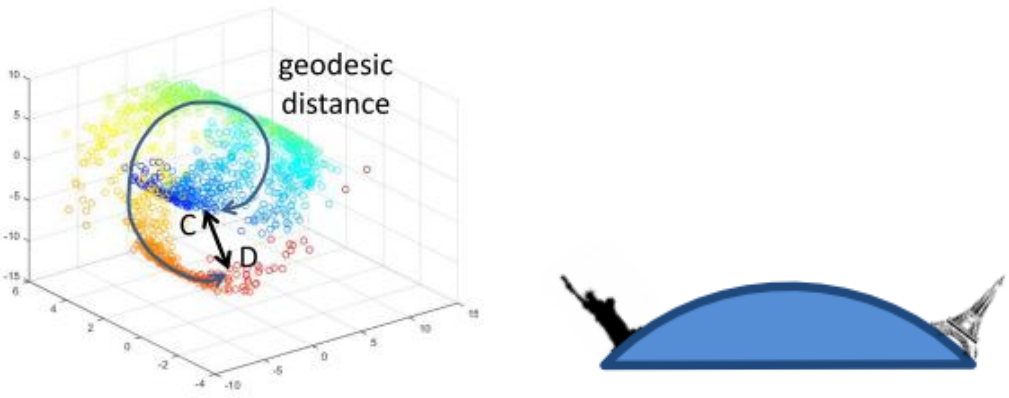
The k-means algorithm can find spherical clusters, and GMM extends this to be able to find elliptical clusters, however both algorithms will be unable to correctly classify highly irregular clusters:



If we could transform this data in some way, that may make it easier for simple clustering algorithms to find the desired results. The key assumption of manifold learning is that high dimensional data actually consists of a low-dimensional manifold that is locally Euclidean, but is 'rolled up' in a higher dimensional space. The manifold is that subset of points in the high-dimensional space that locally looks low-dimensional (e.g. treat the arc of a large circle as a line).



We need a way of 'unfolding' these manifolds, on which we can then apply regular clustering algorithms. In doing this we are interested in preserving only local or geodesic distances along the manifold, not global distances.



The unfolding itself is just the process of constructing a new similarity-preserving coordinate system. All we need to do this is define all the pairwise geodesic distances, and then simply input these into the MDS algorithm. We then need a way to determine the geodesic distances. It turns out we can do this using weighted, undirected graphs using the following steps:

1. Define some local radius ϵ, and connect all vertices (one for each observation) if $d(i.j) \leq \epsilon$ in the original space.
2. Set the weights between vertices i and j to $d(i,j)$.
3. Compute the shortest path between each pair of non-connected nodes in the graph.
4. Construct the geodesic distance matrix as the length of the shortest path between each pair of nodes in the graph.
5. Perform MDS on the resulting geodesic similarity matrix.

This method is known as the isomap algorithm.

Spectral clustering

Spectral clustering is an alternative non-linear dimensionality reduction method to the isomap algorithm. It proceeds as follows:

1. Construct a similarity graph by starting with a fully connected graph, and setting all weights equal to $w_{ij} = \exp\left(-\frac{1}{\sigma}|x_i - x_j|^2\right)$, which is called a Gaussian kernel.
2. Define the graph Laplacian matrix $L = D - W$, where D is the degree matrix with the degree of each vertex $\left(\deg(i) = \sum w_{ij}\right)$ on the diagonal.
3. Perform dimensionality reduction using Laplacian eigenmaps.

In this method the new set of coordinates produced by the Laplacian eigenmaps method is chosen such that the following objective is minimised:

$$\frac{1}{2}\sum_{ij}|Z_i - Z_j|^2 w_{ij} = f'Lf$$

This minimisation is taken subject to the restriction that $f'f = 1$, as otherwise we can always reduce the result by reducing the length of f. We thus have the problem:

$$\begin{aligned}\mathcal{L} &= f'Lf - \lambda(f'f - 1)\\ 0 &= 2f'L' - 2\lambda f'\\ Lf &= \lambda f\end{aligned}$$

Each of these f_i vectors represents the set of ith components of the original data points in the new coordinate system. As with PCA, we order the vectors by size of the eigenvalue. In this case, however, the largest eigenvalue always corresponds to the vector $\tilde{1}$ which is not helpful, so we just ignore this solution. Putting down all the f_i in columns then gives as a matrix with the mapped points as rows.

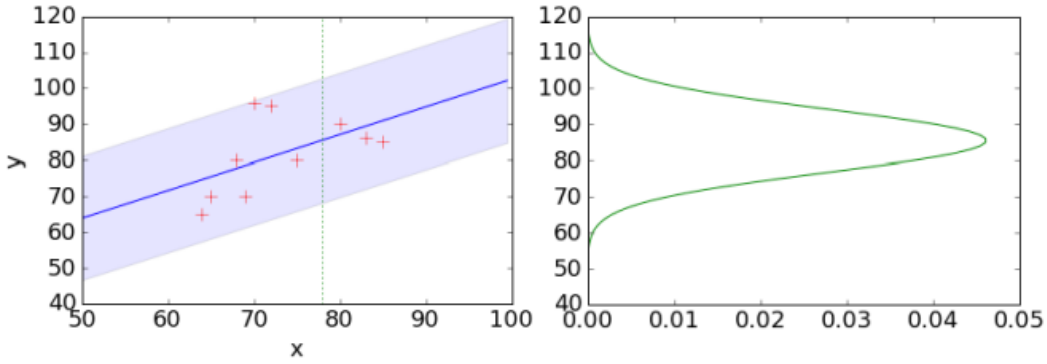
Bayesian Methods

The Bayesian approach to inference

The frequentist approach to inference is to first specify a model, then use maximum likelihood to find the optimised parameters for that model, and finally to use the resulting optimised model to make predictions. The Bayesian approach differs in that it rejects the attempt to reduce the model to a single optimal parameter, instead considering the full space of possible parameter values. Instead of making a point prediction, therefore, Bayesians compute the expected value of the posterior distribution:

$$p(y|\tilde{x}) = E_{p(w)}[p(y|\tilde{x})]$$

The key insight is that providing a single number as a prediction gives us no information about the spread or uncertainty in this value.



The Bayesian approach does not just take the single most likely value of the parameters $\tilde{w}$, but uses all possible values weighted by their probability of being consistent with the observed data. Aside from giving more information about data spread, this approach is also less sensitive to overfitting.

Frequentist: learning using *point estimates*, regularisation, p-values …

- backed by complex theory relying on strong assumptions
- mostly simpler algorithms, characterises much practical machine learning research

Bayesian: maintain *uncertainty*, marginalise (sum) out unknowns during inference

- nicer theory with fewer assumptions
- often more complex algorithms, but not always
- when possible, results in more elegant models

Bayesian regression

The Bayesian approach to linear regression does not compute the maximum likelihood estimate of $\tilde{w}$, but instead considers the full posterior distribution:

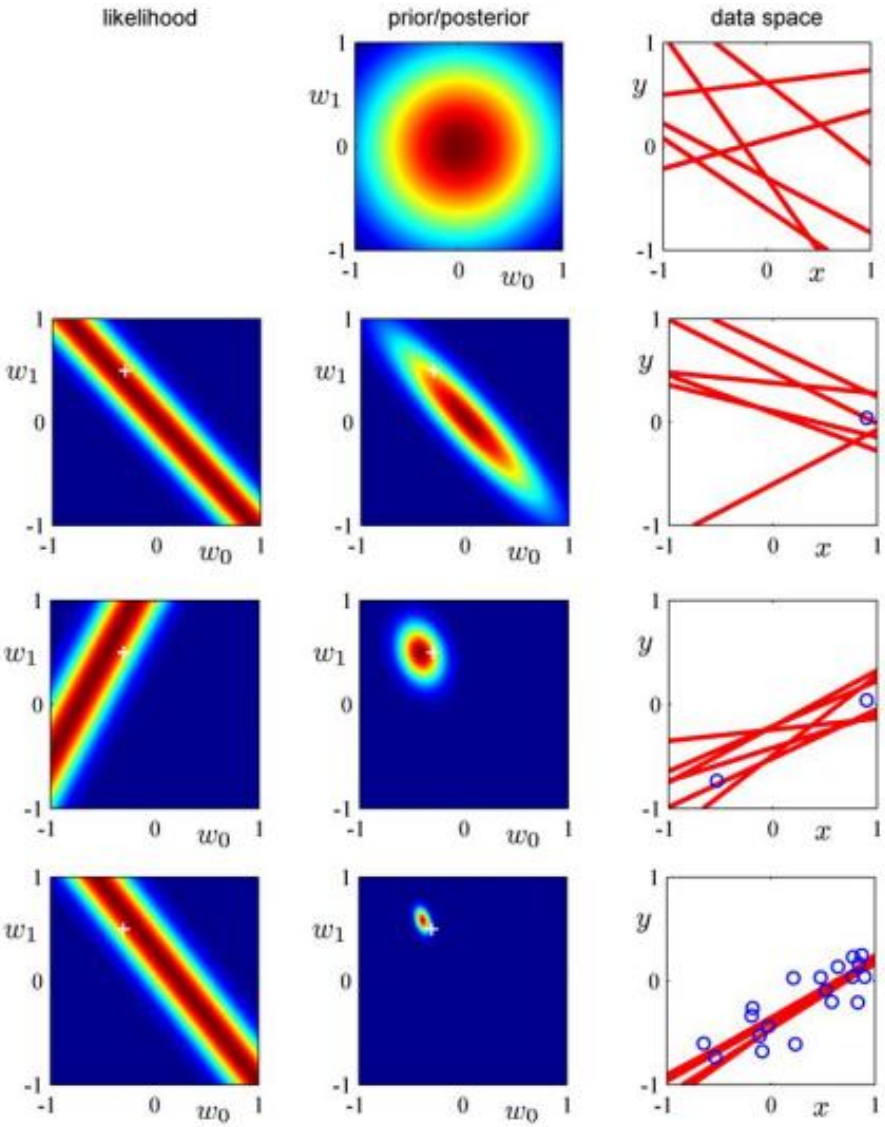
$$p(\tilde{w}|X,\tilde{y},\sigma^2) = \frac{p(\tilde{y}|X,\tilde{w},\sigma^2)p(\tilde{w})}{p(\tilde{y}|X)}$$

$$p(\tilde{w}|X,\tilde{y},\sigma^2) = \frac{p(\tilde{y}|X,\tilde{w},\sigma^2)p(\tilde{w})}{\int p(\tilde{y}|X,\tilde{w},\sigma^2)p(\tilde{w})d\tilde{w}}$$

If both our likelihood and our prior $p(w)$ are normally distributed, their product is also normally distributed. This is an example of a conjugate prior, where the posterior and prior share the same distributional form (in general they need not).

$$p(w|X,y,\sigma^2) \propto N(w|0,\gamma^2 I_D) \times N(y|Xw,\sigma I_N)$$
$$\propto N\left(w\middle|\frac{1}{\sigma^2}V_N X'y, V_N\right)$$
$$\text{where } V_N = \sigma^2\left(X'X + \frac{\sigma^2}{\gamma^2}I_D\right)^{-1}$$

Bayesian regression can be conducted in a sequential manner by updating on a single observation at a time, in each instance using the previous posterior distribution as the new prior. An example of this is given in the figure below.



Expected predictions in the Bayesian regression are given by:

$$p(y^*|x^*, X, \sigma^2) = \int p(w|X, y, \sigma^2) p(y^*|x^*, w, \sigma^2)\, dw$$

Note how this differs from the frequentist approach to prediction:

$$p(y_*|\mathbf{x}_*) = \text{sigmoid}(\mathbf{x}_*' \hat{\mathbf{w}}) \qquad E[y_*] = \mathbf{x}_*' \hat{\mathbf{w}}$$

$$p(y_*|\mathbf{x}_*) = E_{p(\mathbf{w}|\mathbf{X}_i, \mathbf{y})} \left[\text{sigmoid}(\mathbf{x}'\mathbf{w})\right]$$

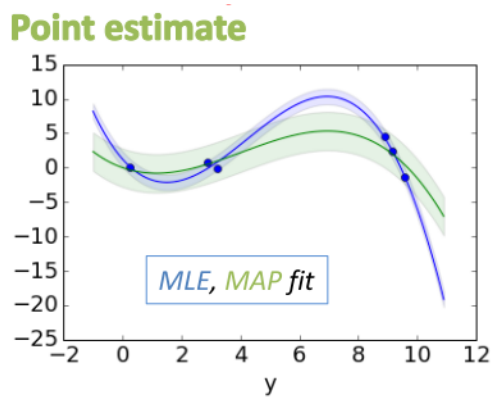
$$p(y_*|\mathbf{x}_*) = E_{p(\mathbf{w}|\mathbf{X}_i, \mathbf{y})} \left[\text{Normal}(\mathbf{x}_*'\mathbf{w}, \sigma^2)\right]$$

Also compare the workflow for maximum likelihood, approximate Bayes, and exact Bayes methods:

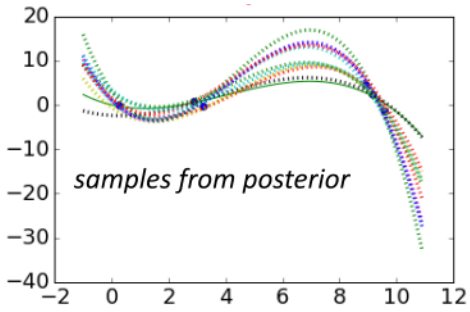
1. Decide on model formulation & prior
2. Compute *posterior* over parameters, $p(\mathbf{w}|\mathbf{X},\mathbf{y})$

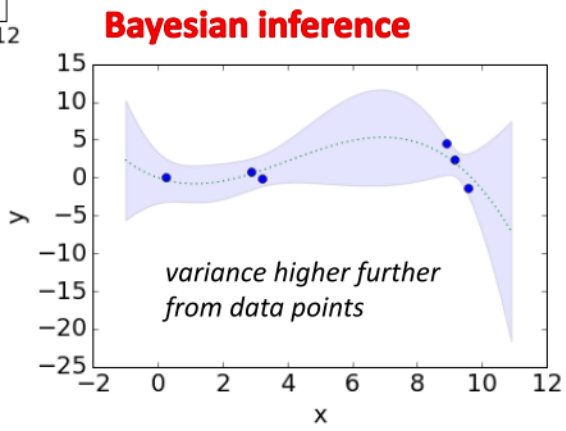
MAP	approx. Bayes	exact Bayes
3. Find *mode* for $\mathbf{w}$	3. Sample many $\mathbf{w}$	3. Use *all* $\mathbf{w}$ to make *expected* prediction on test
4. Use to make prediction on test	4. Use to make *ensemble* average prediction on test	

The Bayesian approach results in a prediction variance that depends upon the data value x, and thus has a higher variance at further distance from the data points, which makes sense.



MLE (blue) and MAP (green) point estimates, *with fixed variance*





Data: y = x sin(x); Model = cubic

Bayesian classification

Generative models produce a joint distribution $p(x, y)$ allowing us to model the input as well. Traditional discriminative models only have a conditional distribution $p(y|x)$. Bayesian methods for discrete inference in classification constitute generate models, which use conjugate priors for discrete distributions, such as the beta-binomial pair.

For the likelihood and the prior:

$$p(k|n,q) = \mathrm{Bi}(q;n,k) = \binom{n}{k} q^k (1-q)^{n-k}$$

$$p(q) = \mathrm{Beta}(q;\alpha,\beta) = \frac{\gamma(\alpha+\beta)}{\gamma(\alpha)\gamma(\beta)} q^{\alpha-1}(1-q)^{\beta-1}$$

The posterior is then:

$$\begin{aligned} p(q|k,n) &\propto p(k|n,q) \times p(q) \\ &= q^k(1-q)^{n-k} q^{\alpha-1}(1-q)^{\beta-1} \\ &= q^{k+\alpha-1}(1-q)^{n-k+\beta-1} \\ p(q|k,n) &\propto \text{Beta}(q; k+\alpha, n-k+\beta) \end{aligned}$$

Suite of useful conjugate priors

	likelihood	conjugate prior
regression	Normal	Normal (for mean)
	Normal	Inverse Gamma (for variance) or Inverse Wishart (covariance)
classification	Binomial	Beta
	Multinomial	Dirichlet
counts	Poisson	Gamma

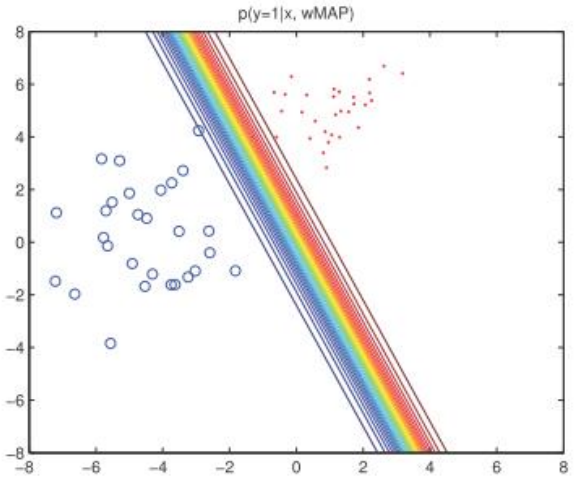
Applying these ideas now to the case of Bayesian logistic regression, we have the likelihood:

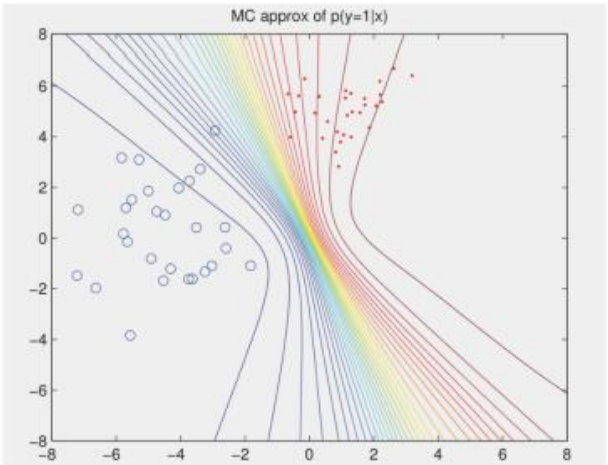
$$\begin{aligned} p(y|q,x) &= q^y(1-q)^{1-y} \\ q &= \sigma(x'w) \end{aligned}$$

We now need a prior over w, unlike q as was the case before. Unfortunately there is no known conjugate prior in this instance, so we typically just use a Gaussian prior over w.

$$\begin{aligned} p(w|X,y) &\propto p(w)p(y|X,w) \\ p(w|X,y) &= N(0,\sigma^2 I)\prod_{i=1}^{n}\sigma(x_i'w)^{y_i}\big(1-\sigma(x_i'w)\big)^{1-y_i} \end{aligned}$$

We observe once again how the variance of our prediction increases as we move away from the data in the Bayesian case (right), unlike the frequentist case (left).





Bayesian model selection

Model selection involve the selection of the likelihood function or kernel function, setting any relevant hyperparameters, and optimising algorithm settings (such as number of clusters in K-nearest neighbours). The frequentist approach to model selection involves using holdout validation, selecting the model with the lowest error on a heldout validation set not used in training.

Problems with heldout validation:

- Data inefficient in not using all available data for training
- Computationally inefficient in repeated rounds of training and evaluating
- Ineffective when optimising many parameters at once, as one is liable to overfit the heldout set

Bayesian model selection computes the Bayes factor for two model classes based on the data D:

$$\frac{p(M_1|D)}{p(M_2|D)} = \frac{\frac{p(D|M_1)p(M_1)}{p(D)}}{\frac{p(D|M_2)p(M_2)}{p(D)}}$$

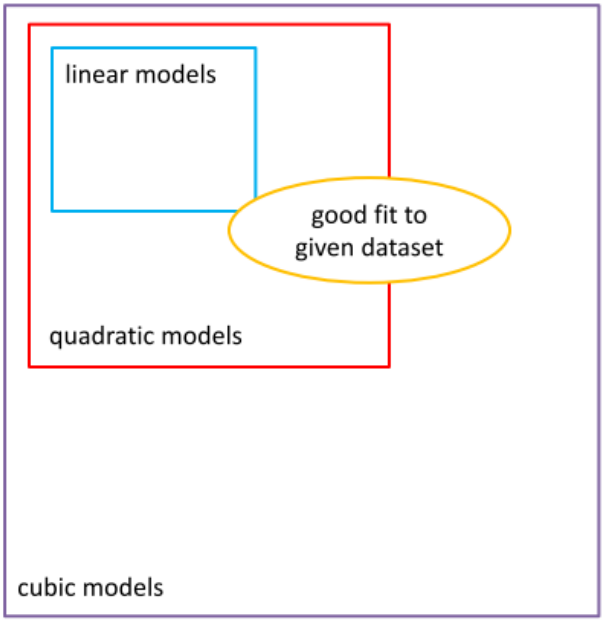
If we have a uniform prior over all models M_i then this simplifies to:

$$\frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)}{p(D|M_2)}$$

To calculate the ratio on the right hand side, we integrate over all possible parameter values for the model in question:

$$\frac{p(D|M_1)}{p(D|M_2)} = \frac{\int p(y|X, M_1, w)p(w_1)dw_1}{\int p(y|X, M_2, w)p(w_2)dw_2}$$

Thus the Bayes factor essentially tells us the relative fit of the two models to the data, averaged over all possible parameter settings. Complex models might better fit the data over some parameter choices, but they also have a much larger space of possible parameter values, and so on average may have poorer fit.



- Space of models: linear < quadratic < cubic
- *Assuming quadratic data*, this is best fit by >= quadratic model
- As complexity class grows, space of models grows too
  - fraction of params offering 'good' fit to data will shrink
- Ideally, would select quadratic model as fraction is greatest

Probabilistic Graphical Models

Introduction

Probabilistic graphical models are an efficient way of representing a joint probability distribution, which allows us to explicitly represent independence relationships between variables. Traditionally joint representations of discrete random variables can be represented by tables, however these increase exponentially in size with the number of random variables. Also a full joint distribution will generally have far too many parameters to easily estimate, and will likely result in overfitting.

Graphical models allow representation of independence relationships that greatly simplify our model. Each node with P parents will have a number of parameters equal to 2^P.The total number of parameters is equal to $2^N - 1$ for a fully connected graph with N nodes.

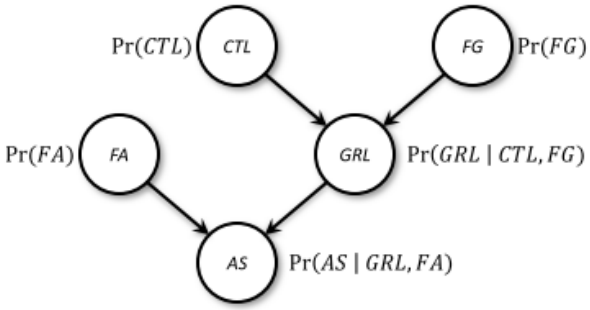
Lazy Lecturer Model	Model details	# params
Our model with S, T independence	$\Pr(S, T)$ factors to $\Pr(S)\ \Pr(T)$	2
	$\Pr(L \mid T, S)$ modelled in full	4
Assumption-free model	$\Pr(L, T, S)$ modelled in full	7

- Independence assumptions
  - Can be reasonable in light of domain expertise
  - Allow us to factor → Key to tractable models

These graphs are particularly useful in representing causal relationships between variables.

Example: Nuclear power plant

- Core temperature
  → Temperature Gauge
  → Alarm
- Model uncertainty in monitoring failure
  - GRL: gauge reads low
  - CTL: core temperature low
  - FG: faulty gauge
  - FA: faulty alarm
  - AS: alarm sounds
- PGMs to the rescue!



Joint $\Pr(CTL, FG, FA, GRL, AS)$ given by

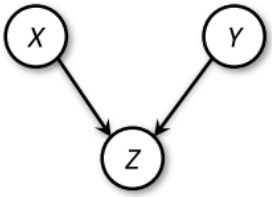
$$\Pr(AS|FA, GRL)\ \Pr(FA)\ \Pr(GRL|CTL, FG)\ \Pr(CTL)\ \Pr(FG)$$

Marginal Independence

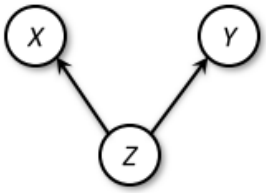
The notation $A \perp B$ means that A is marginally independent of B. If this is true then we can write:

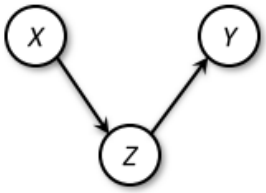
$$P(X, Y) = P(X)P(Y)$$

Two unconnected parent nodes in a directed graph are always marginally independent. However a parent in such a graph is NOT always independent of its children.



In the following two graphs, X and Y are not marginally independent of each other.

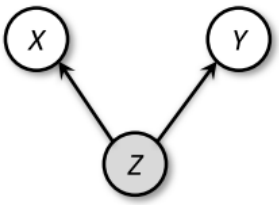


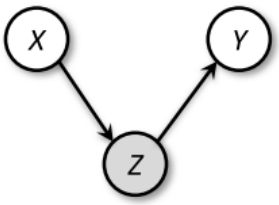


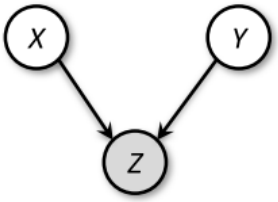
Marginal independence is thus loosely related to the concept of causality: can X cause Y, or can they both share a common cause? If potentially yes, then they are not marginally independent.

Conditional Independence

Conditional independence is different to marginal independence in that it involves setting the value of one or more variables (shown by greying out that node on the graph).

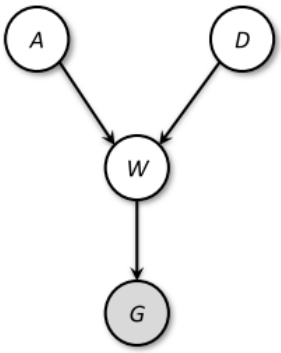






In the first two examples, X and Y are conditionally independent of one another given Z, meaning that they can only influence each other via their effects on Z. In the third example, however, X and Y are NOT conditionally independent. Interestingly, however, in the third example X and Y are NOT conditionally independent, even though they are marginally independent. This phenomenon is known as explaining away. To see how explaining away works, consider the case when X and Y are binary coin flips, and Z is whether they land the same side up. Given Z, then X and Y become completely dependent.

Explaining away also occurs for observed children of the head-head node:

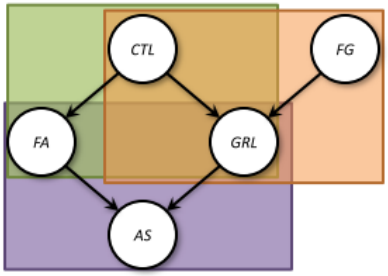


D-Separation

D-separation is a concept used in determining independence relations for larger graphs. Basically we consider all possible paths between two nodes, ignoring the direction of any arrows in the graph (directions matter for determining pairwise independence relations, just not for finding paths). For each path we then consider 'is this path blocked by an independence relation between any nodes along the path?' If all possible paths are blocked, then we say the two nodes are D-separated.

Consider pair of nodes
FA $\perp$ FG?

Paths:
FA – CTL – GRL – FG
FA – AS – GRL – FG



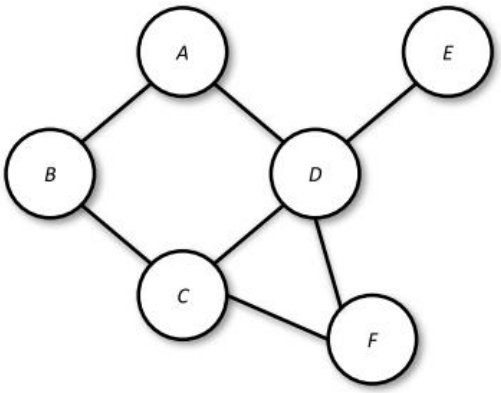
Paths can be blocked by independence

This concept is useful because we need to understand all the independence assumptions encoded in the graph, not just the obvious ones. The Markov blanket of a node is the smallest set of other nodes (variables) on which that node can be conditioned to make it independent of the rest of the graph.

Undirected Graphical Models

Undirected graphs lack directions to the edges that connect nodes together. They also lack a probabilistic interpretation, and so instead of having a conditional probability, each clique of nodes has a factor that is always weakly greater than zero. The joint distribution over a set of nodes will be proportional to the product of all maximal clique factors, but without the appropriate normalisation.

A clique is a set of fully connected nodes in a graph. A maximal clique is the largest clique in a graph.



Clique: a set of fully connected nodes (e.g., A-D, C-D, C-D-F)
Maximal clique: largest cliques in graph (not C-D, due to C-D-F)

In this example then the joint probability distribution is defined using normalisation constant Z:

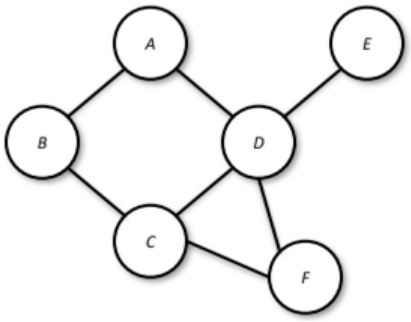
$$P(a,b,c,d,e,f) = \frac{1}{Z}\psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(c,d,f)\psi(d,e)$$

$$P(a,b,c,d,e,f) = \frac{\psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(c,d,f)\psi(d,e)}{\sum_{a,b,c,d,e,f}\psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(c,d,f)\psi(d,e)}$$

Undirected graphs actually have simpler dependence semantics than directed graphs, as now any two nodes are independent so long as we condition upon any intermediate connecting nodes:

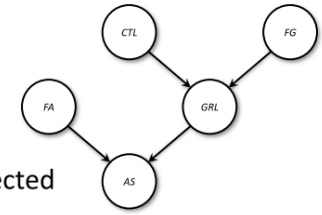
For example B $\perp$ D | {A, C}

Markov blanket of node = its immediate neighbours

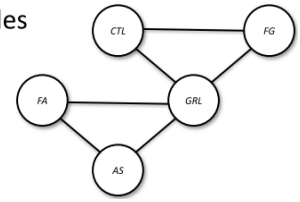


To convert a directed probabilistic graph to an undirected one, we follow the method of 'moralising' the parent nodes by connecting them together.

1. copy nodes
2. copy edges, undirected

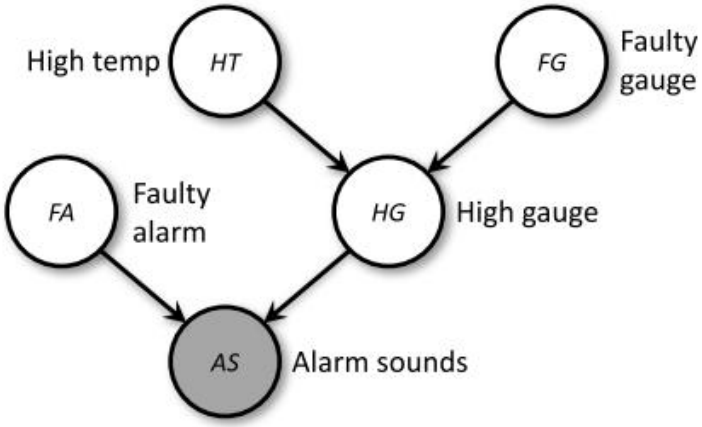


3. 'moralise' parent nodes



Inference with Graphs

Consider the following example graph:



Now imagine we want to calculate the query $\Pr(HT|AS = t)$. We do this as follows:

$$\Pr(HT|AS = t) = \frac{\Pr(HT, AS = t)}{\Pr(As = t)}$$

$$\Pr(HT|AS = t) = \frac{\sum_{FA,HG,FG} \Pr(HT, FA, HG, FG, AS = t)}{\sum_{FA,HG,FG,HT} \Pr(HT, FA, HG, FG, AS = t)}$$

Focus on the numerator:

$$\sum_{FA,HG,FG} \Pr(HT, FA, HG, FG, AS = t) = \sum_{FA} \sum_{HG} \sum_{FG} \Pr(HT, FA, HG, FG, AS = t)$$

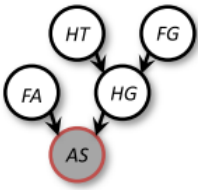
$$= \sum_{FG} \sum_{HG} \sum_{FA} \Pr(HT) \Pr(FG) \Pr(FA) \Pr(HG|HT, FG) \Pr(AS = t|FA, HG)$$

$$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT, FG) \sum_{FA} \Pr(FA) \Pr(AS = t|FA, HG)$$

Nuclear power plant (cont.)

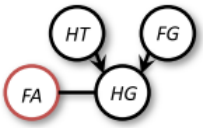
$$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG) \sum_{FA} \Pr(FA) \Pr(AS = t|FA,HG)$$

eliminate *AS*: since *AS* observed, really a no-op



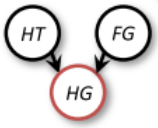
$$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG) \sum_{FA} \Pr(FA)\, m_{AS}(FA,HG)$$

eliminate *FA*: multiplying 1x2 by 2x2



$$= \Pr(HT) \sum_{FG} \Pr(FG) \sum_{HG} \Pr(HG|HT,FG)\, m_{FA}(HG)$$

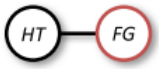
eliminate *HG*: multiplying 2x2x2 by 2x1



Multiplication of tables, followed by summing, is actually matrix multiplication

$$= \Pr(HT) \sum_{FG} \Pr(FG)\, m_{HG}(HT,FG)$$

eliminate *FG*: multiplying 1x2 by 2x2



$$= \Pr(HT)\, m_{FG}(HT)$$

$m_{FA}(HG)$=

FA	f	t
	0.6	0.4

X

FA	HG: f	HG: t
f	1.0	0
t	0.8	0.2

Each step of the algorithm removes one node from the graph and connects the nodes remaining neighbours, forming a clique. Time complexity of this algorithm is exponential in the largest clique. Because of this complexity it is often easier to make inferences numerically by sampling from the desired distribution and constructing a histogram to estimate the relevant probability distribution.

Another example:

independence - solution

P(J=T | E=T) ?? Symbols form ..

Tips:

- Marginalisation of all possible values of *A, B and M*
- Reorder the multiplications and push the summations inward

$$P(j\,|e) = \frac{P(e,j)}{P(e)} = \frac{\sum_a \sum_b \sum_m P(e,j,m,b,a)}{\sum_a \sum_b \sum_m \sum_j P(e,j,m,b,a)}$$

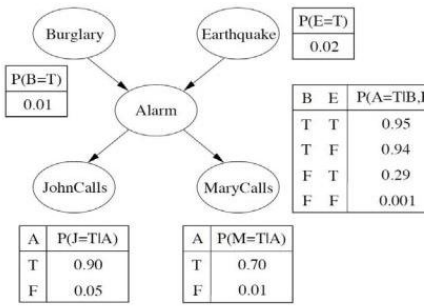
numer. $= \sum_a \sum_b \sum_m P(e,j,m,b,a) = \sum_a \sum_b \sum_m P(b)\, P(e)\, P(a|b,e)\, P(j|a)\, P(m|a)$

$= P(e) \sum_b P(b) \sum_a P(a|b,e)\, P(j|a) \sum_m P(m|a) = P(e) \sum_b P(b) \sum_a P(a|b,e)\, P(j|a)$

denom. $= \sum_a \sum_b \sum_m \sum_j P(e,j,m,b,a) = \sum_a \sum_b \sum_m \sum_j P(b)\, P(e)\, P(a|b,e)\, P(j|a)\, P(m|a)$

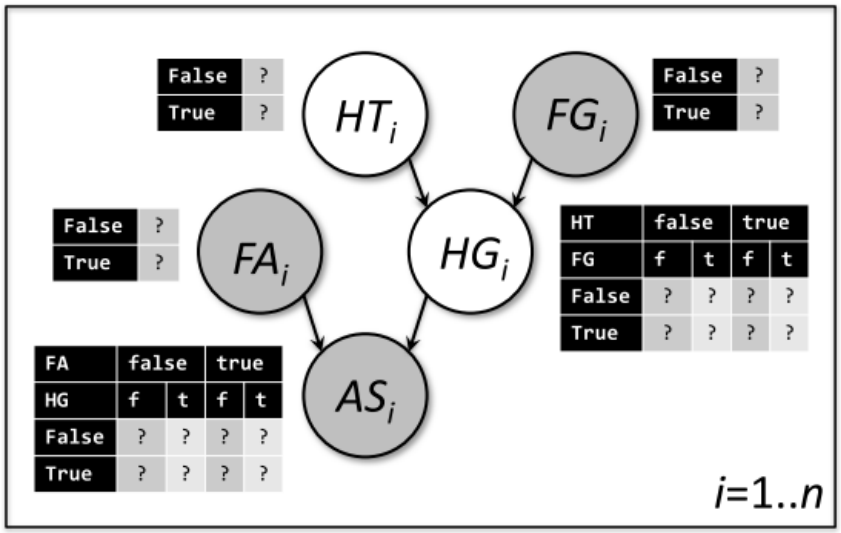
$= P(e) \sum_b P(b) \sum_a P(a|b,e) \sum_j P(j|a) \sum_m P(m|a) = P(e)$

$$P(j\,|e) = \frac{P(e) \sum_b P(b) \sum_a P(a|b,e)\, P(j|a)}{P(e)} = \sum_b P(b) \sum_a P(a|b,e)\, P(j|a)$$

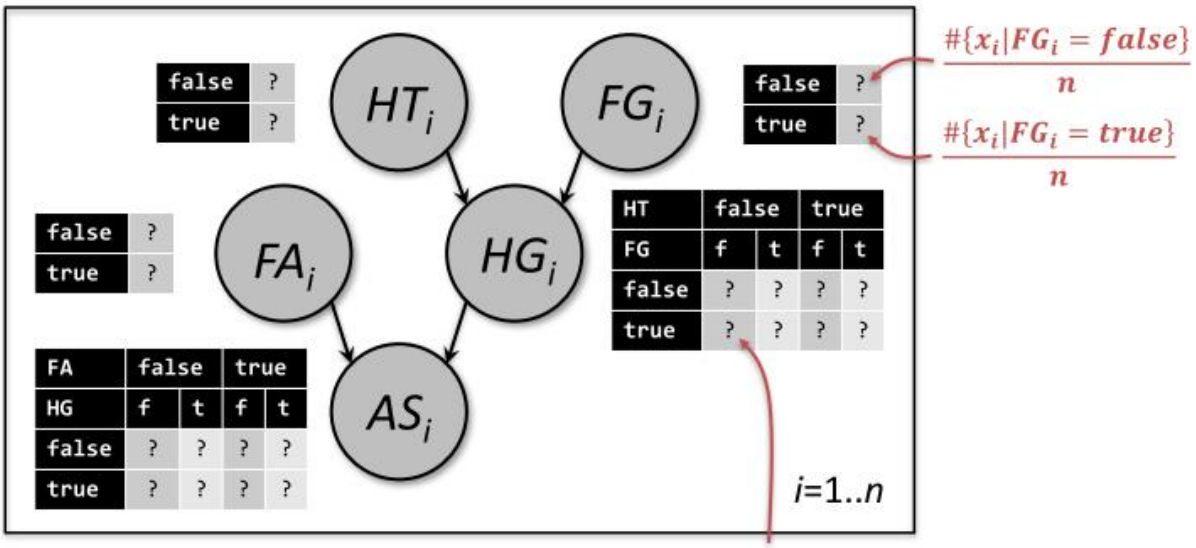


Learning Graph Parameters

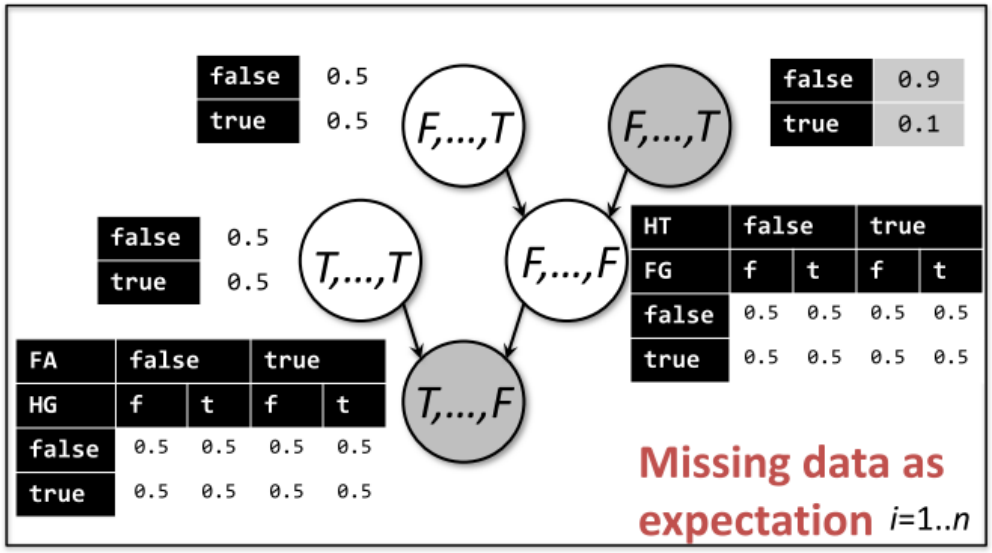
Suppose we have our probabilistic graphical model and a bunch of observations, but we as let lack any of the values for the model parameters.



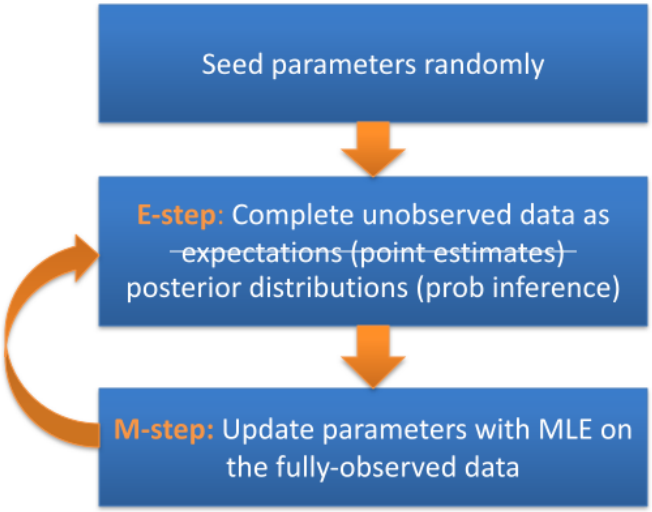
If all our variables are observed, it is easy enough to simply count all the relevant occurrences and use them to calculate the probabilities:



More common, however, is the case where we are unable to observe some variables. These are called latent variables. Basically the way to deal with this is to fill in the parameters we can observe, then begin with a series of guesses for all the rest.

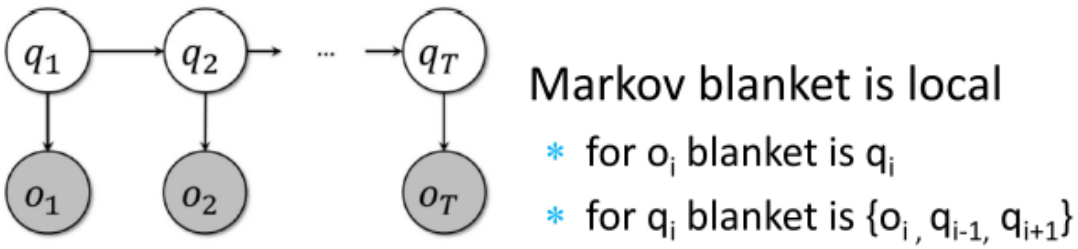


We then follow an iterative process of using our observations to update the most likely values of the unknown parameters, given our data and also the guesses for the other unknown parameters. This is effectively the expectation maximisation algorithm again.



Hidden Markov Models

A hidden markov model involves a sequence of observed discrete outputs from a hidden state:



These have many applications, for example tagging parts of speech of words in a sentence, identifying biological sequences, computer vision, etc. They are formulated as a PGM:

$$P(\tilde{o}, \tilde{q}) = P(q_1)P(o_1|q_1)\prod_{i=2}^{T} P(q_i|q_{i-1})P(o_i|q_i)$$

Key Formulae

Expected value: $E(x) = \sum_i x_i p_i$

Variance: $\text{Var}(x) = E((x-\mu)^2) = \sum_i (x_i - \mu)^2 p_i$

Covariance: $\text{Cov}(x,y) = E\left((x-\mu_x)(y-\mu_y)\right) = \frac{1}{n-1}\sum_i (x_i - \mu_x)(y_i - \mu_y)$

Simple linear regression: $\hat{w} = \text{argmin}_w \left(\left|\tilde{y} - \tilde{X}\tilde{w}\right|_2\right) = (X'X)^{-1}X'y$

Ridge regression: $\hat{w} = \text{argmin}_w \left(\left|\tilde{y} - \tilde{X}\tilde{w}\right|_2 + \lambda|\tilde{w}|_2\right) = (X'X + \lambda I)^{-1}X'y$

Gradient descent: $\tilde{\theta}^{i+1} = \tilde{\theta}^i - \eta\nabla L(\tilde{\theta}^i)$

Stochastic gradient descent: $\tilde{\theta}^{i+1} = \tilde{\theta}^i - \eta\nabla_j L(\tilde{\theta}^i)$ for each observation j at a time

Special perceptron loss: $L(\tilde{w}) = \max(0, -y\sum_{i=0}^{n} x_i w_i)$

SVM objective: $\text{argmin}_w |w|$ s.t. $y_i(w'x_i + b) \geq 1$ for $i \in \{1 \dots n\}$

Kernel definition: $K(\tilde{u}, \tilde{v}) = \psi(\tilde{u})'\psi(\tilde{v})$

Polynomial kernel: $K(u,v) = (u'v + c)^d$, where d is the integer order of the polynomial

Radial basis function kernel: $K(u,v) = \exp(-\gamma|u-v|^2)$, where γ is a spread parameter

Bayesian regression: $p(\tilde{w}|X,\tilde{y},\sigma^2) = \frac{p(\tilde{y}|X,\tilde{w},\sigma^2)p(\tilde{w})}{\int p(\tilde{y}|X,\tilde{w},\sigma^2)p(\tilde{w})d\tilde{w}}$

GMM parameters: For n-dimensional data and k clusters we will have $n \times k$ mean parameters, $\frac{n(n+1)}{2} \times k$ covariance parameters, and $k-1$ weight parameters

Bayesian model selection: $\frac{p(D|M_1)}{p(D|M_2)} = \frac{\int p(y|X,M_1,w)p(w_1)dw_1}{\int p(y|X,M_2,w)p(w_2)dw_2}$

Lagrangian method: $\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda(g(x,y) - C)$

Logistic regression: $y = \begin{cases} 1, & f(\tilde{x}'\tilde{w}) + \epsilon > 0.5 \\ 0, & \text{otherwise} \end{cases}$ with $f(\tilde{x}'\tilde{w}) = \frac{1}{1+\exp(-\tilde{x}'\tilde{w})}$

Quadratic loss function: $L(\tilde{x},\tilde{w}) = \sum_{i=0}^{N}(y_i - \tilde{w}'\tilde{x}_i)^2$