ML1 – notes – Ferdinand Krammer

L1 – Decision stump

- ML (computer learns form examples (data) and generalizes to all inputs): subdiscipline of both programing (computer is an idiot and dose exactly what it is told to do) and Artificial intelligence (computer uses optimization to find the solution to a well defined problem)
- The process:
  - Choose the problem
  - Obtain the required data
  - Choose or design a model
  - Fit model to data using optimization
  - Measure performance
- Decision stumps are 1 rule algorithms used for classification problems (only works on easy problems)

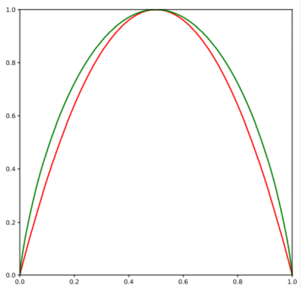
L2 – glossary of terms

- Supervised learning
  - Classification
    - Learn a function $y = f(\vec{x})$ where y is discrete
    - input: image → output: animal, input: demographic → output: voting preference
  - Regression:
    - Learning function $y = f(\vec{x})$ where y is continuous
    - Input: material properties → output: temp, input: detector energy → output: particle paths
  - Multi-label classification:
    - y is a set
    - Identifying objects in an image, text summarisation
  - Structure prediction: y is any thing else
    - Sentence tagging: y is a sequence
    - Automated design: y is a CAD model
- Unsupervised learning: no y, finds pattens in data
  - Clustering: groups similar data points with arbitrary similarity definitions
    - Input: many expression level measurements, output: groups of genes that tend to express at same time
  - Density estimation: learns distribution of data $x_i \sim P$
    - Input: locations that explain detection → output: search order for optical follow up
    - Input: normal outputs → output: possible failures
  - Dimensionality reduction/manifold learning:
    - Reduce dimensions while preserving information, also used for visualization
    - Input: word vectors → output position in layout
- Semi-supervised: as the collecting of data is cheap but its labelling is expensive
  - This uses some labelled data but most unlabelled
- Weakly-supervised: learns form "weak" labels outputting "strong" labels
- Graphical models: Represents structure by drawing relationships (voice recognition, recommender systems)
- Reinforcement learning: actions in an environment, delayed reward
- ML models can also be classified by there answer quality: Point estimate, Probabilistic, Bayesian
- ML can also be classified by there workflow: Batch learning, Incremental learning, Active learning
- Area: traditional, computer vision, NLP, Interactive

Decision tree: good as ignore useless data, can be interpretable, overfits most of the time (very basic)

- The goal: learn $y = f(x)$ form data where $x \in \mathcal{R}^n$ is a n dimensional vector and $y \in N$ is the output class
- Every possible function can't be considered as there are infinitely many so instead the function $f_\theta(x)$ is used which belongs to a space of functions parametrized by θ
- As such the goal becomes learning the parameter $y = f_\theta(x)$ from the data this is done by the incorporation of a loss function so as to find the best θ, $L(y_i, f_\theta(x_i))$ with the total loss being given by the summation $\Sigma_{i=1}^{N} L(y_i, f_\theta(x_i))$, resulting in the models being created to find θ to minimize this
- For a decision stump this allows us to split it in to multiple parts (normally being used for classification)
  - Parameters: $\theta = \{feature, match\}$
  - Function: $f_\theta(x) = \delta(x_{feature}, match)$
  - Loss function $L(y_i, f_\theta(x_i)) = \begin{cases} 0 \; if \; y_i = f_\theta(x_i) \\ 1 \; otherwise \end{cases}$ known as a 0-1 loss
  - It learns the best input feature which is a match to the output feature (so only learns it its answer is in in its input)
- A decision stump where data is continuous, a split parameter is used $f_\theta(x) = \begin{cases} 0 & x_{feature} < split \\ 1 & x_{feature} \geq split \end{cases}, \; where \; \theta = \{feature, split\}$, the best parameters are found through the use of brute force

- For each dimension every split is considered with the loss function for every split evaluate and the best one chosen
- decision tree: applies the decision stump recursively
  - as data can't normally be split by in on location perfectly a decision tree can be used as this is made up of recursive splitting
  - parameter θ is binary tree is a binary tree (dose not have to be a fixed size)
    - the feature at each node, the value on which to do the split (internal nodes contain the split)
    - for leaf nodes the class to output (answer)
  - to evaluate the function $f_\theta(x)$:
    - start at the top move to the first split go to the next and repeat (recursion) until you get to a leaf node and then out put the results
  - tree construction:
    - training data contains only one class → generate leaf node
    - otherwise:
      - try all features/splits selecting the best (lowest total loss)
      - repeat (reclusively) for each split with the data that reaches it
    - this requires a loss function $L\left(y_i, f_\theta(x_i)\right)$, 0-1 loss works for a decision stump but not for a decision tree as it has no sense of partial success (what has been done is good but not solved the entire problem
      - we can use instead Gini impurity or information gain
- Loss functions:
  - Gini impurity: probability that if you select two items form a data set at random (with replacement they will have a different class (only works for discrete target variables)
    - When there is lowest (0) is when there is only one class, this is highest (<1) when every class is different
    - $p_i = P(selecting\ class\ i\ form\ data\ set), G(p) = \Sigma_i p_i(1-p_i) = 1 - \Sigma_i p_i^2$
    - Can handle partial success
    - Weighting the split is done by calculating the $G\left(p^{left}\right)$ and $G\left(p^{right}\right)$ then combining to thet the loss of the split $L(split) = \frac{n_L}{n} G\left(p^{left}\right) + \frac{n_R}{n} G\left(p^{right}\right)$, where n_L, n_R are the number of examples following the left and right branches respectively and n is the total number of exemplar count
    - The weighting means that more data following a branch the more important it is (results in it being less likely to over fit)
  - Information gain: how much is learnt from traversing a split
    - Uses Entropy (therefor always positive)
    - $H(p) = -\Sigma_i p_i \log(p_i)$, log can be base 2 (bits) or base e (nats)
    - Information gain is the number of bits/nats obtained form traversing the split
    - $I(split) = H(p^{parent}) - \frac{n_L}{n} H\left(p^{left}\right) - \frac{n_r}{n} H(p^{right})$
  - Both loss functions are almost identical as seen in fig (green Gini impurity, red information gain), Gini is normally used as faster to compute, however information gain is better working when gini cant (eg regression)



- Overfitting: when the noise is modelled not the signal (solved by comparing a test set)
  - This can be solved by stopping early limiting the tree depth or the minimum leaf node size
  - These are known a hypermeters
  - Can be optimised often done by hand
- L3 – Inputs
  - Real input: loss function can be either Gini impurity or information gain for all axis
  - Quantised real input: data can be split in to bins (information gain shown as spikes)
  - Categorical input: splits no longer make sense as unordered (this or that)
    - The split is the same for 2 or more classes
    - One class by convention goes left with the rest going right this means that the combinations to test grows linearly as $\mathcal{O}(n)$ not as $\mathcal{O}(2^{n-1})$ where n is the number of classes
  - Directional input: real numbers that wrap around (such as days of the week, angles…)
    - Single split makes no sense so instead it is split in half
    - Simplified by using unit length vector $[\cos(\theta), \sin(\theta)]^T$ instead of angle
    - Can give quantised results split using bin centres
- Outputs:
  - Classification:
    - Split to minimise Gini impurity or maximise information gain
    - Leaf gives answer as most common class to reach it
  - Regression:
    - Split to minimise variance of maximise information gain
    - Leaf gives answer as mean value to reach it
  - variance reduction: measures how consistent the output of a node is (so minimised)

        - variance of left node: $\sigma_l^2 = \mathbb{E}[(X_l - \mathbb{E}[X_l])^2]$ where X_l is the data that goes left, similar for the right node
        - minimise weighted combination: $L(split) = \frac{n_l}{n}\sigma_l^2 + \frac{n_r}{n}\sigma_r^2$ where n is the total number and n_r/n_l is the number which goes right/left
    - information gain: much slower than variance reduction
        - fit Gaussian to output variable, entropy is $\frac{1}{2}\log(2\pi e\sigma^2)$
        - information gain is: $I(split) = \frac{1}{2}\log(2\pi e\sigma_p^2) - \frac{n_l}{2n}\log(2\pi e\sigma_l^2) - \frac{n_r}{2n}\log(2\pi e\sigma_r^2)$ where σ_p^2 is for the parent (input)
    - directional output: regression with different rules
        - represent angles with unit length vectors, $\hat{x}_i = [cos\theta_i, sin\theta_i]^T$
        - mean angle $= \frac{\Sigma\hat{x}}{|\Sigma\hat{x}|}$ (used to determine leaf node)
        - variance$= 1 - \frac{|\Sigma\hat{x}|}{n}$ (used to select splits with variance reduction)
    - multiple outputs: predicting multiple outputs from a single tree
        - can be done by the use of information gain plus independence
        - 3D positions: use entropy of multivariate Gaussian
        - Otherwise aome cikd of hack is needed
        - Could train m models one per outputs (works if outputs are uncorrelated)
        - If outputs are correlated then only one model can be used
- Missing data
    - Usually fill in (with mean, mode or another ML model)
    - If missing data is rare then go down both branches
    - If missing data is common only go down one branch

Random Forest: combines decision trees with ensemble learning

- Ensemble learning = combining multiple estimators
    - Combining different models
    - Same model: randomised training so each is different
- Random forests: combine may decision trees
    - Training randomised using bagging (a specific ensemble learning technique)
    - Needed due the fact that models can be fitted in many different ways due to: insufficient data, noisy data, model not complex enough
    - Ensemble captures the inherent ambiguity present
- Ensemble learning
    - Needs diversity in in ensembles
    - Estimators must make different mistakes
    - By increasing the estimator diversity at expense of individual performance $\rightarrow$ better ensemble (to a limit)
- Accuracy:
    - Cant ask everyone so ask sets of people by considering the output of multiple sets the accuracy of the estimate is increased
- Bootstrapping
    - Instead of collecting more data we pretend that we have
    - Done by using a dataset of size n, to create a new data set by drawing with replacement n times
    - Statistics can be calculated on each new statistic
- Bagging: bootstrap aggregating
    - Bootstrapping applied to estimator output
    - Algorithm
        - Select S, size of ensemble
        - Create S bootstrap draws of original data set
        - Train estimator on each
        - Combine outputs of all estimators for each query
- Random subspace method
    - Bootstrap applied to features
    - Remove duplicates
    - Selects around 63% of features, instead select $\left[\sqrt{f}\right]$ where f is number of features
- Random forest = decision trees + bagging + random subspace method
    - Algorithm
        - Select S number of trees
        - Create S bootstrap draws of original data set
        - Train decision tree on each with random subspace method
        - Combine outputs of all trees for each query
- Combing outputs

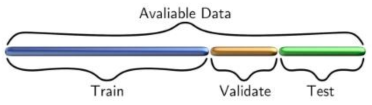
- Classification(mode), Regression (mean/median)
- Can generate probabilities: classification (crates a histogram), regression (gaussian distribution)

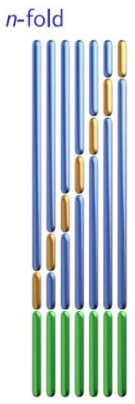
Is it working – L4

- The ML algorithm fails if it
  - Underfits
    - Caused by
      - Model not sufficient for the problem at hand
      - Bad fitting
      - Insufficient data
  - Overfits
    - Learns the noise of the data (to powerful for problem at hand)
    - Insufficient regularisation (smoothing out the noise)
    - Incorrect hyper-parameter tuning (Hyper-parameters: parameters that affect algorithm behaviour including regularisation)
  - Misinterprets the data
  - Uses bad data

Random Forest	Train	Test
Underfitting	79.2%	79.2%
Balanced	97.6%	95.0%
Overfitting	99.6%	94.7%

- To see if a model overfits/underfits the data contains a training/test split
  - Train set → used to train the model
  - Test set → used to verify the performance
  - If there is a large gap between the train/test accuracy then it is usually over fitted
- Parameters: fit to training data (what the data learns)
- Hyperparameters: parameters that cannot be fitted to training data (set before the model is turn)
  - Used to avoid overfitting (decision tree depth)
  - Used to reduce Heavy computation (ensemble size)
  - Hard to optimise
  - Bayesian priors
  - test set is not used to tune the hyperparameters as otherwise it leaks in to the model instead another split is introduced



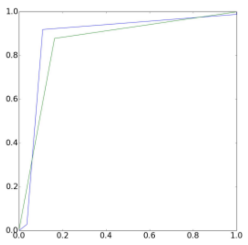


- test set is split in to train (used to train model) and validation (used to optimise hyperparameters) sets
- Measuring performance:
  - Split sides are decided based on
    - Train large → Algorithm performs well
    - Validation large → Hyperparameter optimisation performs well, to a limit
    - Test large → Accuracy performance estimate, to a limit
    - Normally good to have as small as possible test and validation splits however is computational costs are large shrink train set
  - N-fold (rare)
    - Validation and tests used to measurements
    - Can average measurements as long as independent
    - Train/validation set split into n different folds witch are all trained and average performance on test reported
    - n-fold= $n \times slower$ typically $4 \leq n \leq 20$
    - general case: all combinations of train/validation/test
    - jack-knife resampling is the extreme example where test/validation size is 1 so extremely slow
- out of bag error (not on just done with random forest)
  - each exemplar gets estimate for m all trees that didn't train on it
  - tree predictions merged for each exemplar and get an accuracy measured
  - grate for tuning hyperparameters, not for train and test

		Actual	
		False	True
Predicted	False	True Negative (TN)	False Negative (FN)
	True	False Positive (FP)	True Positive (TP)

$\frac{TP}{TP+FN}$	sensitivity, **recall**, hit rate, **true positive rate**
$\frac{TN}{TN+FP}$	specificity, **true negative rate**
$\frac{TP}{TP+FP}$	**precision**, positive predictive value
$\frac{TP+TN}{TP+TN+FP+FN}$	**accuracy**
$\frac{2\times TP}{2\times TP+FP+FN}$	**F1 score**

- final model
  - may train algorithms thousands of times
  - n-fold is a trade of between accuracy and time (grater accuracy longer run time)
  - run on clusters
  - final model trained on entire data set
- performance (what we actually measure)
  - confusion matrices (used to give the classification)
    - can show which classes are confused as tells us if the actual is the same as predicted
  - Imbalanced data: imbalanced training set makes training difficult



            - If instance of something occurring is rare then a high accuracy and be achieved by just returning one result however this is meaningless
            - To prevent the training sent needs to be adjusted (e.g. oversampling)
            - F1 score is a better measure
            - Balanced accuracy(considers accuracy of each class of output): $\frac{1}{|C|}\sum_{c\epsilon C}\frac{|\{y_i=c\wedge f_\theta(x_i)=c\}|}{|\{y_i=c\}|}$ where C is the set of classes of size|C|, (x_iy_i) are the data points, $f_\theta(\cdot)$ is the ML model
        - Receiver operating characteristic (ROC):
            - Previous assume mistakes are equally bad this dose not by setting threshold
            - Threshold sweep lets us see trade-off between true positive rate (y-axies) and false positive rate (x-axis) as such want to be top left corner
        - Regression
            - Consider root bean squared error (RMSE)
            - Mean absolute error (MAE)
            - Histogram, confidence intervals have there uses to show where the error originates form as it could be systematic or bi modal
        - All outlined are intermediates but need to consider the problem which is being considered as need to optimise to the real world as other factors are part of the problem
            - Cost/loss, Gain, Error, Risk
        - Overall test complete system form beginning to end to see where the actual errors are
- Measurement: if it cant be measured it you can't apply ML to it
    - Measurement risk → probability of an incorrect decision due to a measurement
        - Measurement is wrong due to false negatives and false positives
        - Measurement is right but the wrong thing is measured or itis results are incorrectly interpreted (takes correlation to mean causation)
- Bad data:
    - Insufficient: algorithm is unable to fit correctly dur to lack of data
    - Spurious correlation: learns a correlation which is not related to what is wanted (e.g. a different object present)
    - No correlation: input has no relationship to what the output is supposed to be (algorithm should learn nothing)
    - Unbalanced data: classifier not doing anything as almost all data is one case so high accuracy given if everything is assigned that
    - Selection bias: reacting to the wrong thing (remember the wrong thing)
    - Runtime mismatch: learn something which dose not always hold true in all places (in regression this is known as extrapolation(outside of data which has already been seen) vs interpolation(with in space of data))
    - Missing context: dose not understand the reason why something like survival rate in ICU is greater
    - Biased data: learns issues present in society (sentencing/predictive policing)/ leans what it has previously seen
    - Detecting bad data:
        - Visualise it (bad visualisation can be a bad thing)
        - Use multiple performance metrics
        - Test for failure scenarios (in initial testing put a human in the loop to validate it)
    - Failure is an option: all systems fail
        - Structured noise: make the same mistakes each time (not necessarily visible in the data need a feed back loop form deployment to development)

Optimisation Basics 1 – L5

- Finding the best search 2 types (convex easy to find solution as one minima, however in the real world non-convex cases are more prevalent with local minima and maxima)
    - Function is convex if for any pare of pointsx_1, x_2 $f\big((1-\beta)x_1+\beta x_2\big)\leq(1-\beta)f(x_1)+\beta f(x_2), where\ \beta\epsilon[0,1]$
    - No elegant approach to deal with non-convex cases especially as maths is better established to solve convex cases (non-convex cases are often solved by performing convex cases with in them)
    - Convex case can be solved by closed formed solutions to do this we
- Optimisation methods
    - Naïve exhaustive search(brute force)
        - Creates and evaluates every possible solution (always)
        - Computationally intensive
    - Random search
        - Grid layout(uses a regular grid to find the optimal solution) (generally less efficient than random search due to fixed interval between points)
        - Random layout(uses a randomised spread of parameters for search) (not guaranteed to do better)
        - Neither can guarantee that they will find the optimal solution as could find only local minima, however saves time as we don't explore all possible solutions to the problem

        - Latin hypercube sampling (LHC) is used for higher dimensional space (when a search is carried out in a column then that column/row is blocked reducing the search space)
    - Numerical approximations/meta-hypercritic methods (used especially when there are no analytical solutions available)
        - Simulated annealing: starts with large search step (higher temp) over time the search step reduces (temp decrease)
        - Partial swarm algorithm: lots of initial search points and if on finds something then the density of search points in that aria increase
        - Genetic algorithm: in each new generations the children undergo a small change form the parents which might lead to a worse result at which point they are discarded but it might also find a more optimal solution
        - Monti-Carlo method: places points randomly and based on the distribution of the points it comes up with some results (area can be determined by the number of points inside the object/total inside the square as the area of the square is known)
        - Examples of this are Markov chain monte Carlo and nested sample
    - Gradient based method (main search method)
        - Considers the first order derivatives (gradient descent)
        - Considers the second order derivatives (Newton's method) (use the Hessian matrix)

```
1 $t = 0$; Make an initial guess $\mathbf{x}_t$;
2 Iterate until the termination condition is met.
    1 Find a direction $\mathbf{p}_t$ to move;
    2 Decide how much ($\alpha_t$) to move along $\mathbf{p}_t$ direction;
    3 $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t$;
    4 $t = t + 1$;
```

- Gradient descent: 1,2D
    - Start of with an initial guess which can then be optimised iteratively by considering its local labours and if they are lower take it as the new solution are repeat by exploring to find a value of $f(x)$ which is lower until a minimum is reached at which point it has converged
    - Similar process is done for 2D whereby an initial point is selected and then by considering the solution to the function $f(x)$ for local neighbours, if it is less than the starting point then this point is selected and the process is repeated until it converges to the minimum
    - Converged when the differences in the function are below a threshold value
    - How to decide(there are a lot of techniques to do this)
        - Direction to move $\boldsymbol{p}_t$
        - Step size α_t
        - The termination condition

$\mathbf{x} = [x_1, x_2, \cdots, x_n]$.

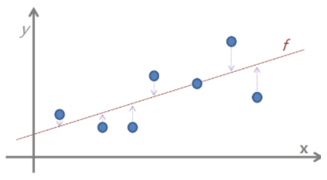
$$\mathbf{H}(f(\mathbf{x})) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Maths

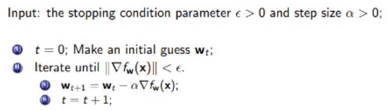
- Derivatives $f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$ used when two parameters are given
- Derivative of logistic regression function: $f(x) = \frac{1}{1+e^{-x}}, f'(x) = e^{-x}(1+e^{-x})^{-2} = (e^{-x} + 1 - 1)(1+e^{-x})^{-2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = f(x)(1-f(x))$
- Partial derivatives: $f(x) = f(x_1, x_2) = x_1^2 + x_2^2$ therefore partial derivatives of x_1 and x_2 are simply
    - $\frac{\partial f(x_1,x_2)}{\partial x_1} = \lim_{\Delta x_1 \to 0} \frac{f(x_1+\Delta x_1, x_2) - f(x_1,x_2)}{\Delta x_1} = 2x_1$
- Gradient: $\nabla f(x) = [2x_1, 2x_2]^T$ where ∇ is the vector differential operator
- Hessian matrix: square matrix of second-order partial derivatives of the function
    - $\boldsymbol{H}(f(\cdot))$ is symmetric if the function is twice-continuously differentiable
    - $\boldsymbol{H}(f(\cdot)) = \boldsymbol{J}(\nabla f(\cdot))$
- Gradient tells us how steep a line is, hessian matrix tells us the nature of the gradient at a point, Jacobean matrix is the transpose of the gradient matrix

Optimisation Basics 2 – L6

- Convex optimisation
    - Function is convex if C is, $\beta \epsilon [0,1]$
        - Conves if $f((1-\beta)\boldsymbol{x} + \beta\boldsymbol{y}) \leq (1-\beta)f(\boldsymbol{x}) + \beta f(\boldsymbol{x})$
        - Strictly convex if $f((1-\beta)\boldsymbol{x} + \beta\boldsymbol{y}) < (1-\beta)f(\boldsymbol{x}) + \beta f(\boldsymbol{x})$
    - optimisation formulation:
        - for a given function it is optimised by minimisation by finding $f(x_*) \leq f(x)$ (this can become maximisation by multiplying by -1)
        - the optimisation can be constrained through the use of constraint functions
            - $x_* = \arg\min_{x \epsilon \mathcal{X}} f(x)$ such that $g_i(x) \leq 0$(inequality), $h_j(x) = 0$(equality), $i, j = \{1, \ldots n\}$
        - There are many types of classification problems
            - Constrained vs unconstrained
            - Discrete vs continuous
            - Deterministic(exact result(analytic) vs stochastic(deterministic (eg monti-carlo method)
- Linear least-square regression



  - For a given data set(pairs of inputs and outputs) $D = \{(x_1, y_1) \dots (x_N, y_N)\} \epsilon\ x, y$
  - Goal is find best fit that minimises the sum of square errors (SSE): $SSE = \Sigma_i (prediction - output)^2$ equivalent to $\arg\min_{f(\cdot)} \Sigma_{i=1}^N (f(x_i) - y_i)^2$ (minimising the overall distance of the outputs to their predictions)
  - Linear function is defined as $f_W(x) = w_0 x_0 + \dots + w_M x_M = \boldsymbol{w}^T \boldsymbol{x}$ where M is the number of input dimensions, thus the optimal solution is written as $\boldsymbol{w}_* = \arg\min \Sigma_{i=1}^N (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2$ it the data matrix is written as $\boldsymbol{X} = (x_1, \dots, x_N)$ and label vector as $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{pmatrix}$ this can be re written as $\boldsymbol{w}_* = \arg\min \left|\left|\boldsymbol{X}^T \boldsymbol{w} - \boldsymbol{y}\right|\right|^2$



```
Input: the stopping condition parameter $\epsilon > 0$ and step size $\alpha > 0$;
  $t = 0$; Make an initial guess $\mathbf{w}_t$;
  Iterate until $\|\nabla f_{\mathbf{w}}(\mathbf{x})\| < \epsilon$.
    $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f_{\mathbf{w}}(\mathbf{x})$;
    $t = t + 1$;
```

    - Closed form solution generated through normal equations is $\boldsymbol{X}\boldsymbol{X}^T \boldsymbol{w}_* = \boldsymbol{X}\boldsymbol{y}$
      - This is the deterministic solution and not guaranteed to be the best
  - Steepest descent solution: we use $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \boldsymbol{p}_t$
    - Still need to determine step size and direction in which to travel
    - Finding the optimal direction of travel is done by finding the decent direction as $\boldsymbol{p}_t = -\nabla f_{\boldsymbol{w}}(\boldsymbol{x})$ leading to $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha_t \nabla f_{\boldsymbol{w}}(\boldsymbol{x})$
    - The step size can be decided by a number of methods
      - Simple solution is fixing it to a constant but this can result in it being unable to converge
      - Complicated process it the use of an adaptive step length through a process such as simulated annealing (as it gets closer to the minimum it reduces the step size)
    - When to terminate: the simple solution is to terminate below a certain threshold $\left|\left|\nabla f_w(x)\right|\right| < \varepsilon$, where $\varepsilon > 0$ is a small prescribed criteria (gradient below a threshold)
    - Good for isotropic functions not so good for anisotropic functions (struggling even more as if becomes extreme but will still converge)
- Gradient decent variants
  - Batch gradient decent: process groups of data together
    - $w_{t+1} = w_t - \alpha_t \Sigma_{i=1}^N \nabla f_w(x_i, y_i)$
    - Faster but can be computationally slow if size of data points in each batch is large
    - Outliers can have an effect on the direction however it is stabilised as it is an average of a range
  - Stochastic gradient descent: randomly select a sample (can be equivalent to vanilla decent)
    - $w_{t+1} = w_t - \alpha_t \nabla f_w(x_{i_s}, y_{i_s}), where\ i_s$ is the randomly selected sample
    - Speed similar to that of the vanilla version
    - Unstable direction (similar to vanilla version)
    - Can easily get trapped in local optimum but might do a better job than the vanilla version depending on data distribution as it might jump to a better point
  - Mini-batch gradient descent: combis both batch and stochastic methods by using mini-batches of B randomly selected data points where B is smaller than the total number of points N
    - $w_{t+1} = w_t - \alpha_t \Sigma_{i=1}^B \nabla f_w(x_i, y_i)$
    - Works as data is correlated with the gradient of the mini-batch being a good approximation of the gradient of the full batch
    - Often now referred to as SDG
    - Size often set as 32, 128 256 … as 2 to the power of something is bets as operations are carried out bit wise

Logistic regression – L7

- Binomial distribution $f(k, n, p)$ discrete probability distribution that describes the probability of k successes out of sequence of n independent experiments with success probability p
  - $f(k, n, p) = \binom{n}{k} p^k (1-p)^{N-k}, \quad where\ \binom{n}{k} = \frac{n!}{k!(n-k)!}$
- Bernoulli distribution $f(k, p)$: the special case of the binomial distribution where n=1
  - $f(k, p) = p^k (1-p)^{1-k} = \begin{cases} p, & if\ k = 1 \\ 1-p, & if\ k = 0 \end{cases} for\ k\epsilon\{0,1\}$
- Binary classification problem:
  - given set of data points $D = \{(x_1, y_1), \dots, (x_N, y_N)\} \subset X \times Y \subset \mathbb{R}^n \times \{0,1\}$, $\boldsymbol{x}_i$ is a vector of observations where as y_i is the corresponding label
  - the goal is to train a model with N samples to predict the Boolean-valued outcome given a new feature vector
  - this can be done through the use of logistic regression (used in a wide range or applications such as recommender systems
- logistic regression:
  - logistic function (sigmoid function):
    - $p = \sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$, which maps $x\epsilon[-\infty, \infty]$ to $p\epsilon[0,1]$ with p different form binary value y
    - The invers is known as the logit function $logit(p) = \sigma^{-1}(x) = \log\left(\frac{p}{1-p}\right)$ mapping $p \in [0,1]$ to $x \in [-\infty, \infty]$
    - $\sigma'(x) = \frac{d}{dx}\frac{1}{1+e^{-x}} = \sigma(x)\big(1 - \sigma(x)\big)$
    - We modify this by the inclusion of $f_w(x) = \boldsymbol{w}^T \boldsymbol{x}$ so that

ML1 – notes – Ferdinand Krammer

Input: step size parameter $\alpha_t > 0$;

1. $t = 0$; make an initial guess (e.g., $w_{0,0} = w_{0,1} = 0$);
2. Iterate i, the samples, until termination conditions are met (e.g., $\log L(\mathbf{w})$ stops increasing).
   1. $w_{t+1,j} = w_{t,j} + \alpha_t \frac{\partial \log L(\mathbf{w})}{\partial w_j}$ for $j = 0, 1$;
   2. $t = t + 1$;

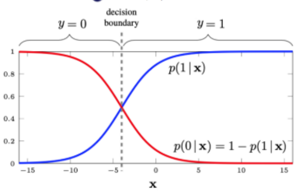
- For multivariate case, we have:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x},$$

where

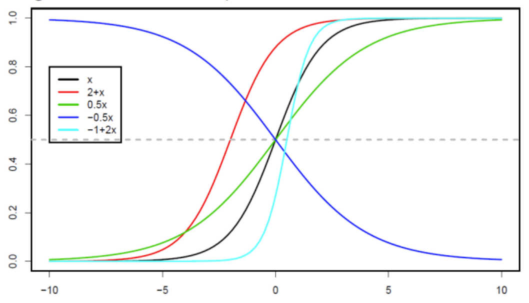
$$\mathbf{w} = (w_0, w_1, w_2, \ldots)^\top, \qquad \mathbf{x} = (1, x_1, x_2, \ldots)^\top.$$

- Input: step size parameter $\alpha_t > 0$;
  1. $t = 0$; Make an initial guess (e.g., $\mathbf{w}_0 = 0$);
  2. Iterate until termination conditions are met (e.g., $\log L(\mathbf{w})$ stops increasing).
     1. $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \nabla_{\mathbf{w}} \log L(\mathbf{w})$;
     2. $t = t + 1$;



- $f_w(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1-e^{-\boldsymbol{w}^T x}}$
  - Connects to the generalised linear model
  - This can be used for binary classification problems where output is 0 or 1
  - Assume there is a single input $\boldsymbol{z} = \boldsymbol{w}^T\boldsymbol{x} = w_0 + w_1 x$
  - The probably of predicting one can thus be written as $P(y = 1|x, \boldsymbol{w}) = f_w(x) = \frac{1}{1+e^{-w^T x}}$ where w_0 *and* w_1 are unknown parameters which we are trying to estimate so as to make predictions
  - Maximum likelihood estimation
    - The probability can be expressed in a similar form to the Bernoulli distribution $p(y|x; \boldsymbol{w}) = [f_w(x)]^y[1 - f_w(x)]^{1-y}$
    - The likelihood with respect to w is $L(w) = \Pi_{i=1}^N p(y_i|x_i, w_i)$
    - Therefore log-likelihood → $\log L(\boldsymbol{w}) = \Sigma_{i=1}^N [y_i \log(f_w(x_i)) + (1 - y_i)\log(1 - f_w(x_i))]$

Logistic curves in the simple univariate case with different $\mathbf{w}$.

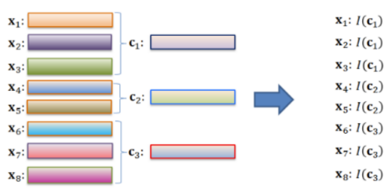


- Optimal solution as $\boldsymbol{w}_* = \arg\max_{\boldsymbol{w}} \log L(w)$ (gradient decent: done by differentiating and setting = 0)
- Optimising using gradient decent:
  - As $w_{t+1} = w_t + \alpha_t \boldsymbol{p}_t$ so the direction is obtained via $\boldsymbol{p}_t = \nabla \log(L(w))$
  - $\boldsymbol{p}_{t,j} = (y - f_w(x))x_j$
- this allows a decision boundary's which are not linear to be modelled unlike linear regression, decision boundary' is still linear as it is based on $w_0 + w_1 x = 0$

- Summary
  - Has simpler calculations compared to other ML algorithms
  - Results in probabilistic outputs for categorical classification problems (binary by setting threshold)
  - Easily regularised to prevent overfitting
  - Can accommodate different forms of the sigmodal function
  - Can be insufficient for complicated problems, such as presence of multiple classes mixing with one another

Unsupervised learning – L8

- Supervised learning: learns from labelled data (inputs with the desired outputs
- Unsupervised learning: learns form unlabelled data $D = \{x_1, \ldots, x_N\}$
  - Only receives the input data with no corresponding output
  - Attempts to detect the underlying structure of data
  - E.g. clustering, dimensionality reduction, auto-encoders and self-organising map
- Clustering: a method to organise unlabelled data into similarity groups
  - A cluster is a collection of similar data samples
  - Many different algorithms exist:
    - Centroid-based partitioning methods: K-mean
    - Hierarchical clustering methods: top-down, bottom-up
    - Density-based methods: density-based spatial clustering of applications with noise (DBSCAN)
    - Distribution-based methods: Gaussian mixture models (GMM) using expectation-maximisation (EM) algorithm



$\mathbf{x}_1$:
$\mathbf{x}_2$:
$\mathbf{x}_3$:
$\mathbf{x}_4$:
$\mathbf{x}_5$:
$\mathbf{x}_6$:
$\mathbf{x}_7$:
$\mathbf{x}_8$:
$\mathbf{c}_1$:
$\mathbf{c}_2$:
$\mathbf{c}_3$:
$\mathbf{x}_1$: $I(\mathbf{c}_1)$
$\mathbf{x}_2$: $I(\mathbf{c}_1)$
$\mathbf{x}_3$: $I(\mathbf{c}_1)$
$\mathbf{x}_4$: $I(\mathbf{c}_2)$
$\mathbf{x}_5$: $I(\mathbf{c}_2)$
$\mathbf{x}_6$: $I(\mathbf{c}_3)$
$\mathbf{x}_7$: $I(\mathbf{c}_3)$
$\mathbf{x}_8$: $I(\mathbf{c}_3)$

  - Applications

1. Initialisation: pick K random data points from data set D as the initial cluster centroids $\{c_1, \cdots, c_k, \cdots, c_K\}$
2. Also initialise K corresponding clusters $\{C_1, \cdots, C_k, \cdots, C_K\}$ (now each only contain one sample!)
3. Assign all the rest data points according to:

$$ind^i = \arg\min_k \|\mathbf{x}_i - c_k\|^2, k = 1, \cdots, K$$

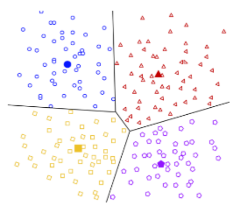
4. Assign $\mathbf{x}_i$ to the corresponding ind^ith cluster C, note the value of ind^i ranges between $1, \cdots, K$. For example, if $ind^i = k$, then assign $\mathbf{x}_i$ to C_k
5. Update the centroids of cluster C_k:

$$c_k = \frac{1}{n_k}\sum_{\mathbf{x}\in C_k} \mathbf{x},$$

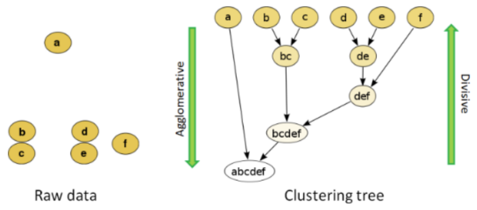
where n_k is the number of elements in C_k

6. Repeat steps $3 - 5$ until converge
7. Outputs are $\{c_1, \cdots, c_K\}$ and $\{C_1, \cdots, C_K\}$

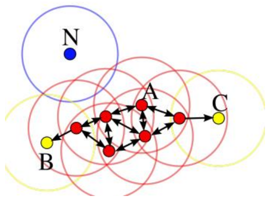
- Social network analysis: takes features and determines underlying pattens (can normalise features before manipulation)
- Anomaly detection: detect differences (changes in data) looks for indicates which are likely to correlate to the animally
- Market segmentation: takes freaturs and splits in into classification and clustering
- Samanic segmentation task for image processing: used for classification
  - Eg N data points: number of pixels $D = \{x_1, \ldots, x_n\}$
  - Feature in vector x_i colour values + location
  - Used to discover some underlying structures



- The idea: calculate the mean vector for each cluster and replace all elements in a cluster by their respective centre, this allows it to also be used for
  - Data generalisation: to represent missing data
  - Privacy protection: may not have features masked by values which can be used for clustering preventing the need for the original data



        - Data compression: used to represent a cluster of data (reduced the number data points which need to be remembered as instead of n just 1 needs to be
    - Centroid-based partitioning methods:
        - K-means clustering:
            - Pick two random points
            - Assign data points to the closest cluster centre
            - Average data points and update the cluster centre to the average of the data points in the cluster
            - Repeat by reassigning data points to the closest cluster centre and continue repeating until there is no further change as converged
        - K-mean clustering advantages:
            - Guaranteed to converge within a finite number of iterations
            - Simple to implement
            - Easy to scale up for large data sets
            - Adaptive to new data points
        - Disadvantages:
            - Need to specify K (the number of clusters) before the run
            - Non-deterministic: as depends on centre initialisation
            - Cant handle outliers well (outliers can become a separate cluster even though the clusters are supposed to be else where)
            - Dose not like data with varying density or size
            - Does not scale up well with regards to the number of features (high dimensional data)
        - Evaluated through both error 1 (L_1) norm and error 2 (L_2) norm
            - L1-norm $S = \Sigma|y_i - f(x_i)|$: robust, unstable, multiple potential solutions
            - L2-norm $S = \Sigma|y_i - f(x_i)|^2$: not very robust, stable solution, always one solution
        - K-mean complexity: for K clusters, with N data points and M features the data points are assigned to the closest cluster O(KNM), change the cluster centre to the average of its assigned points O(NM)

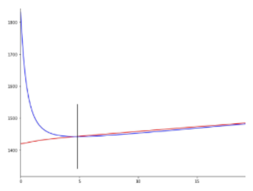


    - Hierarchical clustering:
        - Agglomerative clustering (Bottom up): treat every sample as a cluster, clusters merge with every iteration
            - Single-linkage clustering(also known as nearest neighbour clustering) each point is treated as an element to begin with and each steps combines 2 clusters with the smallest distance
            - Normally faster for computation
        - Divisive clustering (Top down): treat all samples as a single cluster, cluster splits with every iteration
            - Divisive clustering: start with on cluster then split them according to average dissimilarity (also distance based measurements)
            - Has a better view of global structures of data set
        - A Sensitivity parameter has to be included which controls the iteration step or the degree of clustering at which termination occurs as otherwise either all clusters will merge or all data points will split.
        - Clustering best if there are low intra-cluster distances and high inter-cluster distances
    - Density-based methods:
        - Used as centroid-based methods favour spherical clusters and hierarchical clustering can be unstable in different runs
        - Density based methods aim to follow the shape of dense neighbourhoods of the data points
        - Pt is the minimum number of neighbours a point has to have in a sphere of radius ϵ around it
        - Core point, Any point x_i in the data set with a neighbour count grater than or equal to Pt is marked as a core point and part of the same cluster
        - Boarder point, Any point x_i which has a number of neighbours which is less than Pt but with in the sphere of radius ϵ of a core point can be grouped as part of the same cluster
        - Noise point, If a point x_i is outside the radius of both a core and boarder point then it is a noise point
        - Dose not require the number of clusters to be known, can detect outliers
        - Handles arbitrary data better, fast neighbour search becomes harder in higher dimensions
        - Complexity $O(n \log n)$
- Semi-supervised learning
    - Uses small amounts of labelled data with large amounts of unlabelled data which can produce considerable improvement in accuracy as data labelling can be expensive and time consuming
        - Low-density separation, self training/pseudo labelling: uses the labelled data to train a model to predict the labels of the unlabelled data, these are combined to retrain the model and repeated iteratively
        - Generative models:
        - Graphical-based methods
        - Co-training and Multiview algorithms

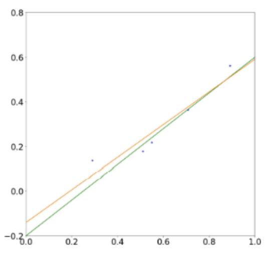
ML1 – notes – Ferdinand Krammer

Regularisation – L9

- Regularisation – try's to prevent overfitting through the use of hyperparameters (with additional information)
  - Needed as the obvious solution might not always be the one which the computer generates (linear solution can have many potential other solutions)
  - Emphasises the simpler models (common sense for models) as overfitting is generally due to unjustifiably complex explanations of the data (Occam's razor: the simplest explanation is usually the correct one)
- Why regularise
  - Reduces over fitting
  - Ill posed problem: multiple equally good solutions exist regularisation forces a selection of one even if arbitrary preventing the optimisation from drifting (never converges, all solutions (often containing parts of multiple) are bad) between solutions
  - Auxiliary data: may reflect extra information, e.g. noise form a camera sensor reeds regularisation applied so as to process the information
  - Human understanding: makes an algorithm go via something which we understand so if the goal is to learn $y = f_\theta(x)$ we could learn instead $y = f_\theta(z)$ and $z = f_\eta(x)$ where z is generally human interpretable
    - Done in attribute learning, allowing you to go form human understanding which can be advantageous
    - Allows for explanation of why an algorithm did what it did
    - Sharing statistical strengths – by using something to help learn something else
    - Zero shot learning – recognise something which has never been seen before from a description
  - Easier optimisation: takes a function which is hard to optimise to something which is easier to optimise
    - Removes local minima increasing the speed of optimisation

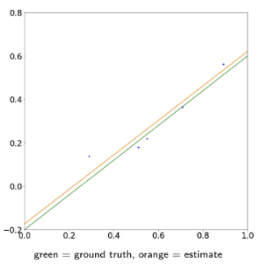


- Aside
  - Model limits: sometimes model used is also regularisation (such as logistic regression which prevents you form turning if you want to get away form a straight line), this is however good if there are invariants/equivariants in the data
  - Early stopping: performance increaseis overtime which can result in over fitting (if you have to stop early the regularisation is to weak
  - Quantity: amount of regularisation needed is depended on data quantity (the more data that is had the less regularisation is needed (infinite data needs no regularisation)
    - Hyperparameters control the strength of regularisation (if the amount of data used changes different hyperparameters are needed so by scaling up the data set they can only be used as a start but need to be re optimised)
    - Models have different sweet spots: not enough data they fail to train, to much and they stop improving (under fitting, or plateauing effect)
- Model types
  - Non-probabilistic

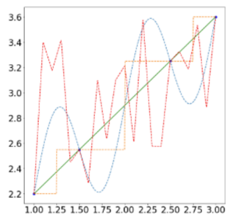


green = ground truth, orange = estimate

    - Arbitrary loss function: minimises loss function $L(\theta)$
      - E.g. L2 $L(\theta) = \Sigma(\mathrm{y_i} - \mathrm{f_\theta} x_i)^2$
    - Regularise it by including as second term to the loss function
      - $L(\theta) = \frac{1}{n}\Sigma_{\mathrm{i=1}}^{\mathrm{n}}\left(y_i - f_\theta(x_i)\right)^2 + \lambda\Sigma_{\mathrm{j=1}}^{\mathrm{k}}\theta_k^2$, where λ is the regularisation strength
    - Ridge regression uses $f_\theta(x_i)$ as $y_i = ax_i + b_i$, $\quad \theta = [a, b]$, uses L2 norm
      - With a loss function $L(\theta) = \Sigma_{i=1}^{n}\left(y_i - (ax_i + b)\right)^2 + \lambda(a^2 + b^2)$
      - Hyper parameter set when loss for validation set is minimised
    - Lasso regression uses L1 norm
      - $L(\theta) = \Sigma_{i=1}^{n}(y_i - (ax_i + b)^2) + \lambda(|a| + |b|)$
    - Elastic net regression: uses both L1 and L2 norms
      - $L(\theta) = \Sigma_{i=1}^{n}(y_i - (ax_i + b)^2) + \lambda\left(\gamma(|a| + |b|)\right) + (1 - \gamma)(a^2 + b^2)$
    - Lasso is often betters for larger data sets/more complex problems as it is better at ignoring irrelevant features
    - Elastic-net lets hyper parameters decide the best optimisation
    - There are other methods



green = ground truth, orange = estimate



  - Probabilistic
    - Maximum likelihood (ML): no regularisation
      - Find the model parameters which maximise data probability $\arg\max_\theta P(data|\theta)$
      - Linear regression: $y_i = ax_i + b + \epsilon_i, \quad \epsilon_i = N(0, \sigma^2)$
        - $P(y_i|x_i a, b, \sigma) \propto \frac{1}{\sigma}\exp\left(-\frac{(ax_i+b-y_i)^2}{2\sigma^2}\right)$
        - With the maximum likelihood solution given a $[a, b]^T = (X^T X)^{-1} X^T y$ where $X = \left[[x_1, 1], [x_2, 1] \dots [x_n, 1]\right]$ $and\ y = [y_1, \dots y_n]^T$
    - Maximum a posteriori (MAP)

- Introduces a prior for the model which is a probability distribution, complete model can generate prediction with out data
- Let $y_i = ax_i + b + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$, adding the priors $a, b \sim N(\mu_0, \Sigma_0)$, $\sigma^2 \sim Inv-Gamma(\alpha_0, \beta_0)$, with $\mu_0 \Sigma_0, \beta_0, \alpha_0$ being hyperparameters
- We gen $[a,b]^T = (X^T X + \Sigma_0^{-1})^{-1}(\Sigma_0^{-1}\mu + X^T y)$

▪ Bayesian, same model as MAP

- Instead of ML solution of finding the most probable solution we use posterior distributions form bays rule
  - $P(model\ param|data) = \frac{P(data|model\ param)P(model\ param)}{P(data)}$
- $[a,b]^T \sim N(\mu_n, \Sigma_n)$, $\mu_n = (X^T X + \Sigma_0^{-1})^{-1}(\Sigma_0^{-1}\mu_0 + X^T y)$, $\Sigma_n = \sigma^2(X^T X + \Sigma_0^{-1})^{-1}$
- Gets a probability distributions of lines out

▪ With infinite data they would be identical, with not enough data ML fails MAP gives a solution and Bayesian gives a solution telling you how confident it is

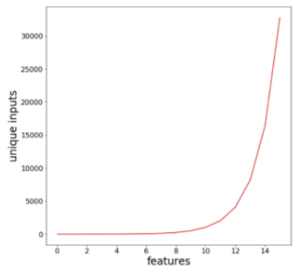
▪ Ideal solution is Bayesian however this is slower and harder to code and optimise, and have a good prior problem

○ Priors: Regularisation is bias towards a simple solution (judges which model is best (assumes parameters in model are sensible)

▪ Types: uninformative (says nothing can result in ML solution), improper (probably distribution dose not give any use full information), minimum description length (don't just care about the model cost but the care about the model and the data)

- Can be extra knowledge, data drive, or human belief

▪ Conjugate prior: gives an analytical solution, problem is that they are simple and so can be bad matches for the data meaning that Bayesian methods often under perform



- Model kinds
  - ○ Discriminative: (ML, MAP)
    - ▪ Learns $P(y|x)$
    - ▪ Used directly
    - ▪ Learns boundary's between data
    - ▪ Only discriminates between classes
  - ○ Generative: (Bayes)
    - ▪ Learns $P(y,x)$
    - ▪ Apply bases rule
    - ▪ Learns distribution of data (must be probabilistic)
    - ▪ Can generate data, handle missing data, less likely to overfit, know when unreliable

1. Start with empty set
2. Try n modifications: Adding (not in set) or removing (in set) each variable
3. Select best (usually with regularisation term on number of features used)
4. If best changed return to step 2, otherwise exit

○ Generative is ideal however harder to code and optimise, slower and a discriminative approach often is wins (it keeps jumping between generative and discriminative being better)

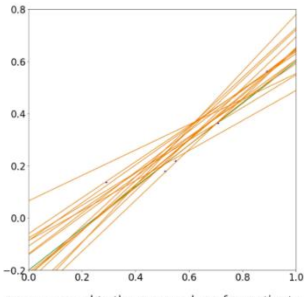
Curse of Dimensionality – L10

- Curse of dimensionality – in short is the more dimensions that the data contains the more data is required
- Combinatorics: If there an n binary features then there are 2^n possible inputs means that as n increases the number of unique exemplars decreases with if fitting an exponential curve

$$\left|\frac{\mathbb{E}[(x_i-\mu_{x_i})(y-\mu_y)]}{\sqrt{\mathbb{E}[(x_i-\mu_{x_i})^2]\,\mathbb{E}[(y-\mu_y)^2]}}\right| > t$$

(often adjust threshold to achieve target feature count)

- Coverage: this is important as ML makes the assumption that if two features are similar then they will generate similar answers, however if there is low coverage if there is a large number of dimensions then nothing is similar however we can still consider most similar and least similar
- For 1000 points in nD space in a [0,1] for n dimensions the differences between the max, mean and min distances are the same but they increase and the ratios decrease meaning the they all start to look the same as the dimension increase
- At higher dimension you get poor coverage thus similarity no longer works



green = ground truth, orange = draws from estimate

- Affects probabilistic distributions: as dimensions increase the data forms a hypersphere around the mean (thin shell avoiding the mean)
- Avoiding the curse: Affect all models (related to regularisation (sometimes a potential solution)

○ Feature selection: based on selecting a smaller number of features and ignoring the rest (not a reliable method)

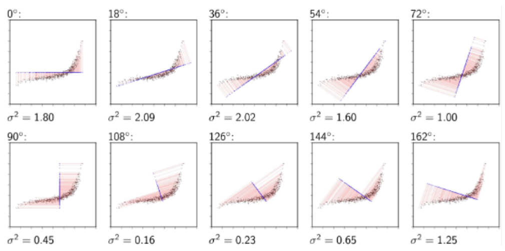
▪ Model selection: train the model with as subset of features and select the best

- Model trained: on subset of features (ideally all) this works but impractical typical proses is to use an optimisation algorithm (steepest assent hill climbing),
- Very slow
- Same as stepwise regression if using a regression model, often gets stuck at a local minima

▪ Filtering: use an estimate of feature usefulness and filter accordingly

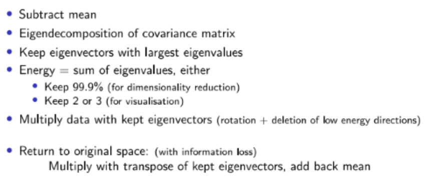
- Questionable but fast

- Calculates how useful each feature is and select a set of features based on a threshold, then train a model based on this (only done once)
- Threshold absolute correlation is generally used (calculates the correlation between the input and output features
- Dose not work as usefulness depends on other variables
  - Two variables might contain the same information (including both is pointless)
  - A variable on its own might be useless but a combination is useful



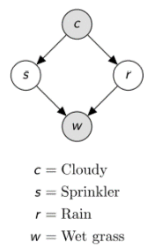
- Embedded: an algorithm with feature selection built in
  - Where this is part of the learning process (as done with random forest) results in a good trade of between speed and performance
- Feature engineering
  - Design features which fit the problem better (requires some domain knowledge) often about invariance
  - Example would be to include an additional feature which would allow logistic regression to predict a circle
  - Can improve the performance of a model however requires domain knowledge
- Observation the problem doesn't exist for real data (problem still has to be solved)

- Real data: Computers can see many more dimensions than there are in the real data (image has 1024x1024x3, where as if the object only has 6 (3 for position and 3 for orientation)), with real data being mostly low dimensional
- Manifolds: low dimensional surface embedded in higher dimensional space
  - Real data lies generally on a manifold
  - The manifold is assigned a coordinate system so that dimensionality reduction can be performed



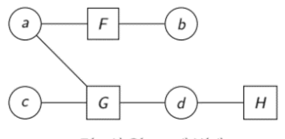
- Subtract mean
- Eigendecomposition of covariance matrix
- Keep eigenvectors with largest eigenvalues
- Energy = sum of eigenvalues, either
  - Keep 99.9% (for dimensionality reduction)
  - Keep 2 or 3 (for visualisation)
- Multiply data with kept eigenvectors (rotation + deletion of low energy directions)
- Return to original space: (with information loss)
  Multiply with transpose of kept eigenvectors, add back mean

  - This dose not work the there are loops
  - Dimensionality reduction is typically done using Principle component analysis (PCA)
    - Simple and fast
    - Grater for visualisation and dimensionality reduction
    - Unsupervised
    - Only linear
    - Still valuable for non-linear manifolds for removing the linear component
- PCA
  - One feature: represent nD data set using only one feature
    - Don't specify specific features but instead a linear combination of features
    - Scale doesn't matter therefor the feature we keep $= \hat{d} \cdot \boldsymbol{x},\ |\hat{d}| = 1$
    - For visualisation we get a linear combination (project onto a line) by trying to minimise the projection distances (information loss(squared)) with the line going through the mean
    - All that matters is the orientation distance
  - More principle components
    - First principle component (the first component we want to keep) is the direction with the maximum variance
    - The second principle component: select another direction, maximising the variance, this needs to be orthogonal to the principle component so as to minimise shared information
    - Repeat orthogonality for all components each time try's to ensure that the earlier dimensions have as much of the information as possible
    - Energy $= \Sigma_{i=1}^{n} \sigma_i^2$ what we are trying to maximised
      - Invariant to rotation, PCA is a rotation followed by removal of low variance dimensions
      - Maximising the variance for all principle components minimises later components
      - PCA minimises lost E (typically keeping 99.9% E) selects principle components until a threshold is passed
    - PCA algorithm: is an analytic solution
      - Principle components are the eigenvectors of covariance matrix, variance along eigenvector is the eigenvalue
      - Therefore PCA uses eigen decomposition
  - Further manifold algorithms
    - All non-linear
      - T-SNE:
      - Isomap – probably most common for visualisation
      - Autoencoder – probably most used for dimensionality reduction
      - Probabilistic PCA
      - Gaussian process latent variable model – Probably most theoretically sweet
- Other approaches to avoiding curse of dimensionality include: using convolutional NN, and using structure
- Classic processing: uses whitening which starts by subtracting mean and the dividing the standard deviation before running PCA

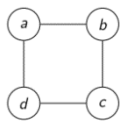


- Improves most algorithms except random forest

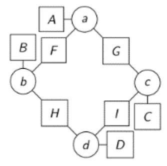
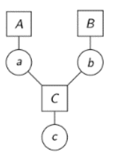
Graphical Models – L11


$= F(a, b)G(a, c, d)H(d)$

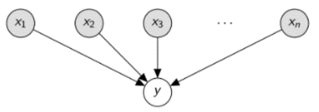
- Traditional method takes and applies classification
- Instead of learning one model (P(crashed|6 features)) we learn instead two models (P(crashed|3 features, speed)P(speed|3 feature))
- Dose not have to be probabilistic (NN)
- Why:
  - Less dimensions reduces curse of dimensionality as they build an explicit manifold
  - More data as part can have a large amount of data and another can have a small amount of data
  - Support missing data:
  - Performance accuracy and speed (removes possibilities when they can't be true) this is in principle


$= F(a, b)G(b, c)H(c, d)I(d, a)\ A(a)B(b)C(c)D(d)$

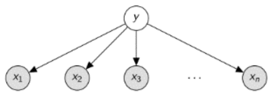
- Structure is conditional independence (things that happen before it are no longer relevant as they provide no additional information at hand, what makes these useful are the edges which are omitted and as such no longer useful

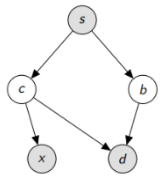
- Notation
  - Things in circles are random variables (variable)(b,c), things not in circles are (hyper-)parameters (fixed)(a)
  - Shaded means observed (a) and unshaded means unobserved (b)
  - Modes are seen in different states
    - For training they can all be seen, for testing/runtime then somethings will be seen, for missing information even less is seen. In each case they can still be used to make predictions (normally only the training model is reported in lit)



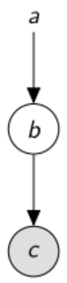
  - Latent variables: always unobserved
    - Hidden – something we know exists but cant be measured
    - Hypothetical – something which might not actually exist



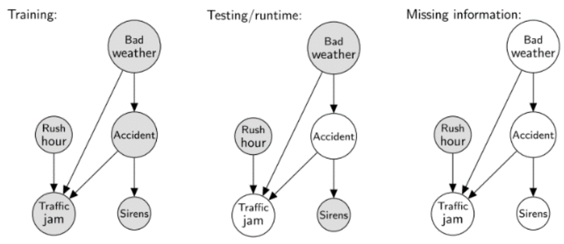
- Three representations
  - Bayesian network (Bays net/network, Belief net/network)
    - Intuitively $x \to y$ means x causes y
    - Represents the equations for the product of terms one per RV



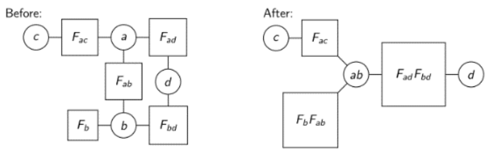
  - Factor graphs
    - Products of factors
    - Circles are RVs, squares are factors, edges are dependency
    - Often used for unnormalized probability distributions
  - Markov random fields (Markov network)
    - Factor graphs without boxes as implicit function, only functions of 2 RV
    - Functions
      - F,G,H,I pairwise terms of two variables
      - A,B,C,D: unary terms (function of one variable)
- The representation is piked based on what is most convenient sometimes multiple are used
- Conversion



  - Factor graphs are used to code it as they reduce complexity in the code (always possible to convert to not always form factor graphs)
  - For Markov network to factor graph we add factors in to each of the edges then add factors for the implicit unary terms
  - For Bayesian network to factor graph we add factors for each RV then link corresponding RV then tails stay on RV and heads move to factor
- Examples
  - Classification & regression:
    - Represent $P(y|x_1, x_2, x_3 \dots x_n)$
    - Everything connected so no conditional independence thus graphical model uninteresting
  - Naïve Bayes
    - Opposite to classification/regression with everything independent $P(x_1|y)P(x_2|y) \dots P(x_n|y)$
    - While makes sense as everything is independent this is a strong assumption and rarely true so best avoided
  - Expert systems:
    - Encode knowledge as Baysian network
    - Nice idea but generally can't assigned probabilities to things, cause effect not always clear, complexity
    - Represents conditional probability distribution for random variables



ML1 – notes – Ferdinand Krammer



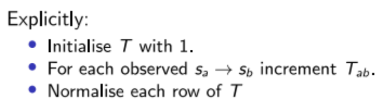
- Can't have loops in the arrows as that would be a clause violation
- Probabilities can back propagate

- Markov chain
  - Next state depends only on previous state (state has no memory of the past
  - Commonly applied to time but also to space
  - Everywhere (simulation, stock market, speech recognition, data transfer)
- Markov grid
  - RVs only depend on neighbours, can be 2D, 3D grid
  - Uses in Ising model, image labelling, image segmentation…

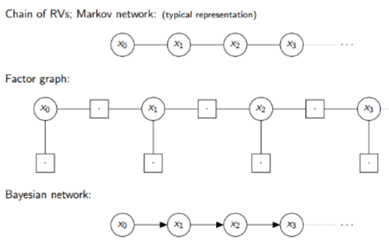
- Further concepts
  - Conditional random field
    - Markov random field: distribution learned from data always the same (generative (can generate data))
    - Conditional random field: distribution learning form data distribution depends on data (look at data and then updates distributions)
  - Explaining away
    - As probability of one thing goes up the probability of something else decreases
    - When observing a parent dose not make the children conditionally independent
  - d-separation (directional separation)
    - RVs a and b are d-separated if independent given observed RVs z (as flow of information is blocked)(d-connected (dependent) if intermediates are unseen)
    - Consider all paths ignoring directions
    - dependent if when we learn something about a our under standing of b is updated and visaversa
    - Information can bounce of other observed RVs

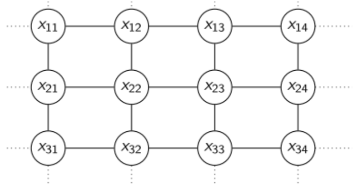
d-connected (dependent)

  - Distribution choices
    - RVs can represent anything
    - Functions can be anything
    - Graphical modes need these to be specified, writing down the model dose not mean it can be solved
  - Merging RV
    - By merging factors it can make the faster and more convenient but not always possible, can be used to remove loops creating trees

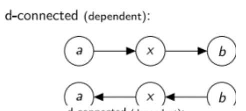

Explicitly:
- Initialise T with 1.
- For each observed $s_a \to s_b$ increment T_{ab}.
- Normalise each row of T

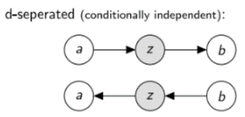
- Learning factors
- Just learning each probability independently (e.g. Naïve bayes)
- Binary naïve Bayes: every probably is Bernoulli $P(x_i|y) = Bernoulli\ (p_i)$
- Maximum likelihood: $p_{iy} = \frac{C_i(1,y)}{C_i(0,y)+C_i(1,y)}$
- Maximum a posteriori (MAP) with prior $p_{iy} \sim Beta(1,1)$: $p_{iy} = \frac{1+C_i(1,y)}{2+C_i(0,y)+C_i(1,y)}$
- Bayesian: $p_{iy} \sim Beta\big(1 + C_i(1,y), 1 + C_i(0,y)\big)$
- Assignment probabilities
- Probability of RV assignment: just evaluate all factors and multiply together
- For naïve Bayse with binary RV:
  - $P(y|x) \propto \Pi_{i=1}^{n} P(x_i|y)P(y)$
  - $P(y|x) \propto \Pi_{i=1}^{n} \left(p_{iy}\right)^{x_i} \left(1 - p_{iy}\right)^{1-x_i} P(y)$
  - To prevent numerical underflow this is logged
    - $\log\big(P(y|x)\big) = \Sigma x_i \log\left(p_{iy}\right) + (1 - x_i) \log\left(1 - p_{iy}\right) + \log\big(P(y)\big) + C$
  - +C is due to proportionality from applying Bayes Rule without knowing P(X)

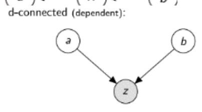


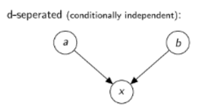


  - Causality
    - Graphical models are often causal
    - Learn causality → a research problem
    - Often ambiguous as data cannot resolve, unobserved RVs could destroy all conclusions



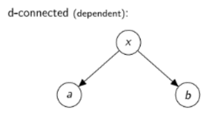


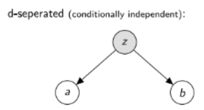


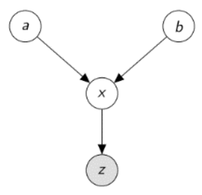


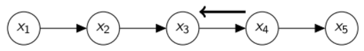
Belief Propagation – L12

- Markov chain
  - Finite state machine is an example of a Markov chain (MC are more generalist and probabilistic, where as FSM are usually not, MC can have infinite states)
  - Transition matrix is a matrix $T \in \mathbb{R}^{n \times n}$, such that $P(s_a - s_b) = T_{ab}$ for a set of n states $\{s_0, s_1, \dots s_{n-1}\}$ (the probabilities of transition)
    - Each row is a categorical distribution (has to sum to 1)





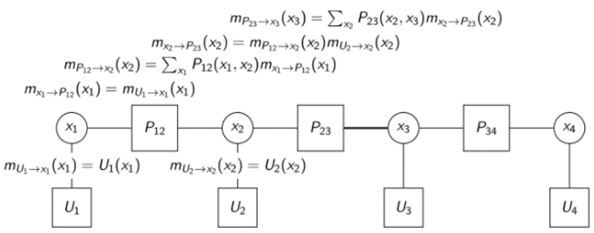




- $m_{1\to 2}(x_2) = \sum_{x_1} P(x_2|x_1)P(x_1)$
  (message from node 1 to 2)
- $m_{2\to 3}(x_3) = \sum_{x_2} P(x_3|x_2)m_{1\to 2}(x_2)$
- $m_{5\to 4}(x_4) = \sum_{x_5} P(x_5|x_4)$
- $m_{4\to 3}(x_3) = \sum_{x_4} P(x_4|x_3)m_{5\to 4}(x_4)$

  - Choose a prior (uniform distribution), update with data, set T to maximum posterior
- Simulation
  - Chose a start state and select length
  - Better to introduce both a start and stop state

- Dynamic programming
  - We know something in a sequence about the sequence (location of a letter / the probability of a letter being in a specific location) and uses this with any other constraints to fill in the rest
  - Dynamic programming uses
    - Maximum likelihood (ML) – most probable state sequence
    - Maximum a posteriori (MAP) – most probable sequence with prior


$m_{P_{23}\to x_3}(x_3) = \sum_{x_2} P_{23}(x_2,x_3)m_{x_2\to P_{23}}(x_2)$
$m_{x_2\to P_{23}}(x_2) = m_{P_{12}\to x_2}(x_2)m_{U_2\to x_2}(x_2)$
$m_{P_{12}\to x_2}(x_2) = \sum_{x_1} P_{12}(x_1,x_2)m_{x_1\to P_{12}}(x_1)$
$m_{x_1\to P_{12}}(x_1) = m_{U_1\to x_1}(x_1)$
x_1
P_{12}
x_2
P_{23}
x_3
P_{34}
x_4
$m_{U_1\to x_1}(x_1) = U_1(x_1)$
$m_{U_2\to x_2}(x_2) = U_2(x_2)$
U_1
U_2
U_3
U_4

▪ Marginals – distribution over state of each node
○ Marginals ("belief"): posterior of single RV given evidence
▪ $b_i(x_i) = \Sigma_{./x_i} P(x_1, \dots x_n)$ where $x_i, i \in [1,..N]$ are RV, $b_i(x_i)$ is the belief of RV i, and $P(x_1, \dots x_n)$ is the joint distribution updated with what is known (./x is all except x)
▪ $P(x_1, \dots x_n)$ is impractical as the numbers of values scales quickly ($states^n$)
○ As can be represented as a graphical model (Markov chain)
▪ If factorises as $P(x_1, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_2)\dots$

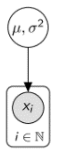
    - The discrete distributions are obtained form the transition matrix
  - As it factorises it allows us to manipulate it: let there be 5 variables then
    - $b_3(x_3) = \Sigma_{x_1,x_2,x_4,x_5}P(x_1,x_2,x_2,x_4,x_5)$ goes to
    - $b_3(x_3) = \Sigma_{x_1}\Sigma_{x_2}\Sigma_{x_4}\Sigma_{x_5}P(x_5|x_4)P(x_4|x_3)P(x_3|x_2)P(x_2|x_1)P(x_1)$

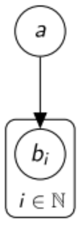
$$b_3(x_3) = \Sigma_{x_4}\{[\Sigma_{x_5}P(x_5|x_4)]P(x_4|x_3)\}\Sigma_{x_2}P(x_3|x_2)\Sigma_{x_1}P(x_2|x_1)P(x_1)$$

    - This is the forwards-backwards algorithm
    - The rearranged sums and products sums minimises the computation/storage (reduces form n^5 for some terms to nothing grater than n^2)
    - Rearranging equations is messy

Sum–product:
- RV to factor:
$$m_{v\to F}(x_v) = \prod_{G\in N_v/F} m_{G\to v}(x_v)$$
- Factor to RV:
$$m_{F\to v}(x_v) = \sum_{x\notin v}\left[F(\cdot)\prod_{u\in N_F/v} m_{u\to F}(x_u)\right]$$
- Belief:
$$b(v) = \prod_{G\in N_v} m_{G\to v}(x_v)$$

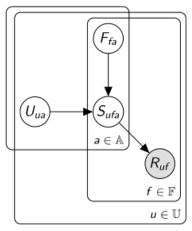
  - An equivalent method is message passing along the edge of the graph (2 messages per edge one in each direction)
    - Messages are functions, discrete functions are arrays of probabilities
    - $b_3(x_3) = m_{4\to 3}(x_3)m_{2\to 3}(x_3)$
  - Message passing:
    - In general $m_{i\to j}(x_j) = \Sigma_{./x_j}U(x_i)F(x_i,x_j)m_{k\to i}(x_i)$, where $F(x_i,x_j)$ could be $P(x_i|x_j)$, $P(x_j|x_i)$ or something non-probabilistic and $U(x_i)$ is any unary term
    - Belief is $b_j(x_j) = U(x_j)\Pi_{i\in N_j}m_{i\to j}(x_j)$
    - Message is a partial joint distributions (summary of terms behind the message, RVs marginalised out irrelivet to destination, generalise to a continuous distribution
  - Usually want marginals at every RV so all of the for wards and backwards messages are sent with the believe being the product of the incoming messages at a single node and a unary term (allows for messages to be reused) (have to be sent in order but can be sent in both directions)
- Can be done with factor graph instead as this makes the whole thins simpler
  - x_i – Random variable, U_i – unary a factor on one RV, P_i – pairwise a factor on two RVs
  - RV to factors: $m_{v\to F}(x_v) = \Pi_{G\in N_v/F}m_{G\to v}(x_v)$ where $G \in N_v/F$ is all neighbours except message destination
  - Factors to RV: $m_{F\to v}(x_v) = \Sigma_{x\notin v}[F(\cdot)\Pi_{u\in N_F/v}m_{u\to F}(x_u)]$
  - Once all messages are sent then $B_v(x_v) = \Pi m_{G-v}(x_v)$
  - Is we already know something about the $B_v(x_v)$ then $m_{v\to F}(x_v) = B_v(x_v)$ (allows for the fixing of a message)
- Viterbi: used if we want to do maximum likelihood or maximum a posteriori
  - Forward-backwards used sum and product
  - Viterbi changes sum to max and so uses max and product
    - $m_{v\to F}(x_v) = \Pi_{G\in N_v/F}m_{G\to v}(x_v)$
    - $m_{F\to v}(x_v) = \max_{x\notin v}[F(\cdot)\Pi_{u\in N_F/v}m_{u\to F}(x_u)]$
  - Record which RV won each max (as draws happen and you can look it the up in reverse and so can be implemented recursively)
- Dynamic programming: is any algorithms where the solution can be found recursively by solving slightly smaller problems first
- Alignment: sometimes the label order is know but not its position (such as voice data)

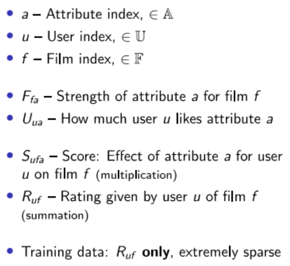




- Viterbi algorithm can be used to force an alignment
- Transition matrix tell to stay in the same state or move to the next

- believe propagation: generalisation of forward-backward/Viterbi to a tree
  - sum-product: generalisation of forward-backward
  - max-product: generalisation of Viterbi
  - min-sum: max-product in negative log space
  - messages are identical but forward/backward direction not defined (thus a node is chosen to be a root node (messages towards this are forward and going away is backward)
  - messages are iteratively sent (a potential method is to use a data structure called a heap)
  - Belief propagation does not work for loops as it creates a circular dependency if loop can be collapsed then it could work
  - Loopy BP: ignores dependency and sends messages anyway and keeps sending until messages converge
    - May converge may not



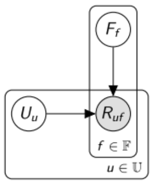


- a – Attribute index, $\in \mathbb{A}$
- u – User index, $\in \mathbb{U}$
- f – Film index, $\in \mathbb{F}$

- F_{fa} – Strength of attribute a for film f
- U_{ua} – How much user u likes attribute a

- S_{ufa} – Score: Effect of attribute a for user u on film f (multiplication)
- R_{uf} – Rating given by user u of film f (summation)

- Training data: R_{uf} **only**, extremely sparse



- Answer is wrong as double counts however often good enough
- Messages are ordered to maximise information spread
- As double counting you need to keep renormalising
- A momentum term can be used to improve convergence by averaging across runs
- Better algorithms exist (graph cuts, tree reweighted message passing)
- Few loops
- Observed node has no dependencies (breaks a loop)
  - Break all loops and run BP on combinations taking weighed average or higher probability
  - Expensive but exact

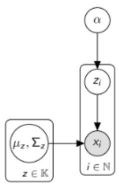
  - Limitations and extensions
    - Have to pass probability distributions between nodes (normally done by discrete distributions)
    - Analytic solutions are rare
    - MCMC and variational more flexible
    - Clique size: number of connections as this gets large efficiency decreases rapidly
    - Gaussian belief propagation: Kalman smoothing generalised to graph
    - Continuous distributions hard in general

Latent variables – L13

- Plate notation: represent a tree through the use of a plate avoids having to explicitly write ever node of a tree
  - Allows parameter sharing (useful for complex models)
- Latent variables: RVs are never seen they are hypothetical
- Recommender system: estimate what you like/dislike, there are 2 approaches:
  - Content-based filtering: matches user to products with features similar to previously liked (needs less data)
  - Collaborative filtering: identifies users with similar likes and assumes they have similar opinions (needs large amounts of data)
  - Generally they are combined however collaborative filtering is nearly always best
- Latent semantic analysis (latent semantic indexing):
  - Similar to: principle component analysis, factor analysis, linear discriminate analysis
  - Have many graphical models which are equivalent
  - Not probabilistic
  - Unknow attributes
    - Attributes inferred form data, there is probably no human interpretation
  - Graphical model
    - Normally simplified by hiding attributes
    - This is a PCA graphical model
  - Fitting to data: U contains user attribute weights, V film attribute weights

1. Calculate A where $A_{uf} = R_{uf}$ when known otherwise 0
2. Perform SVD(matrix decomposition method), obtain $A = U\Sigma V^T$, assumes Σ (singular values) sorted high to low
3. Keep top N columns of U and V where N is the number of attributes (hyper parameter)
4. Scale column n by $\sqrt{\Sigma_n}$ (U and V) (don't want diagonal matrix there so take its square root and pull the dimensions out of both sides)
   - In real life there is to much data so we use gradient descent instead
   - Identical to PCA except
     - Not a covariance matrix

ML1 – notes – Ferdinand Krammer



- Model RVs: α, μ, Σ
- Missing RVs: z
- Observed RVs: x

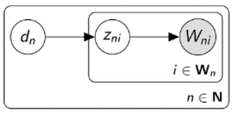
- Likelihood function of z_i:
  Categorical distribution, $z_i \sim \mathtt{Cat}(w_i)$
  $\sum_{z=1}^{K} w_{iz} = 1$

- ▪ Subtracting means optional (recommended for numerical stability)
- ▪ Outputs divided by standard deviation (doesn't preserve Euclidean distance)
- ▪ Trained with sparce data
- ○ Needs lots of data (dose badly for users/films with few ratings)
- ○ Traditional graphical model LSA for document model presented linearly, (user→document, attribute → topic, film → word)

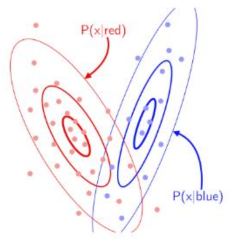
- Density estimation (gaussian mixture model): the inverse process of any generative system
  - ○ Imput: samples, x, drawn for unknown probability distribution D
  - ○ Output: estimate of D
  - ○ Examples: histogram, fitting a distribution, Gaussian mixture model
    - ○ Used to visualisation (create heat map, clustering (probabilistic k-means), abnormality detection)
    - ○ Used for classification:
    - ▪ Do a density estimate for each class ($P(x|C)$)
      - ▪ Choose a prior ($P(C) = p$)
      - ▪ Apply a bays rule $P(C|x) = P(x|C)P(C),\ P(x)$ is ignored and the results are normalised as must probabilities must sum to 1
      - ▪ If gaussian this is gaussian discriminate analysis

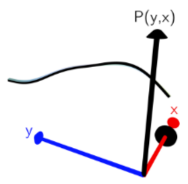
1. Run k-means with K centres
2. Set $\alpha \propto$ number assigned to each centre
3. Fit μ and Σ to points assigned to each centre

  - ○ Used for regression:
    - ▪ $P(y,x)$ form density estimate
    - ▪ Given $\hat{x}$ evaluate, $P(y|x=\hat{x}) \propto P(y, x=\hat{x})$
    - ▪ Equivalent to taking a slice through a surface
  - ○ These are proper generative models but are often beaten by discriminative but are advantageous still if you need the probabilities in other places when used as part of a larger system
  - ○ GMM: sum of gaussian distributions
    - ▪ $P(x|\alpha,\mu,\Sigma) = \Sigma_{z=1}^{K} \alpha_z N(x|\mu_z, \Sigma_z)$, where α is the probability of each mixture component
    - ▪ Fitting the data: there is no analytical solution, cliques to large for belief propagation so an new algorithm is used Expectation maximisation (EM)

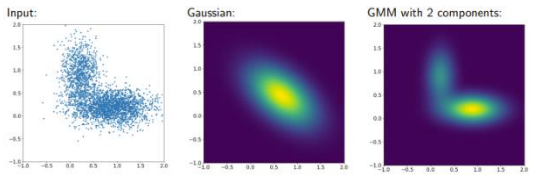


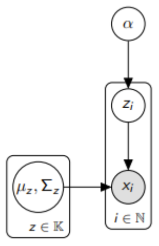
  - ○ Expectation maximisation: can calculate maximum likelihood and maximum a posteriori only
    - ▪ Similar to gradient descent
    - ▪ Initialise, series of steps to improve model fit till stuck in a local minima (if implemented well you will get the same answer each time)
    - ▪ RVs split in to 3 groups
      - Model RVs: latent variables shared between all data (for the entire model)
      - Missing RVs: latent variables associated with data
      - Observed RVs: the Data
    - ▪ Two steps: alternate between them until it converges
      - E-step: calculate likelihood function of missing values with model parameters fixed
      - M-step: maximise model parameters given expectation of missing values





    - ▪ Initialisation:
      - Is done well then easier faster to optimises so optimise using the k-means so as to initialise as close to the optimised results as possible
      - Closer to the goal the faster to converge
    - ▪ E-step: fitting a probability distribution over what the membership at each of the mixture components is where w_i is the weighting that a data point belongs to a mixture component
      - $w_{iz} \propto \alpha_z N(x_i|\mu_z, \Sigma_z)$, (weight proportional to alpha for the mixture component multiplied by the probability that the point comes form the gaussian distribution) and then renormalise so that $\Sigma_{z=1}^{K} w_{iz} = 1$
      - Soft assignment rather than the hard assignment of k-means as probabilistic





- i – Exemplar index, $\in \mathbb{N}$
- z – Component index, $\in \mathbb{K}$
- μ_z, Σ_z – Mean and covariance for component z
- α – Categorical distribution over component membership
- z_i – Which component feature vector i belongs to
- x_i – Feature vector for exemplar i

▪ M-step: weighted update of the gaussian distribution fitted to it

- $\alpha_z \propto T_z, T_z = \Sigma_{i=1}^{N} w_{iz}$,

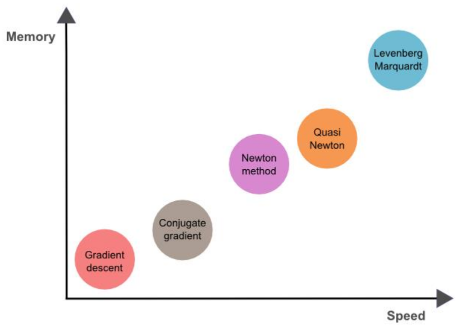
where T the sum of all the weight for the mixture component, normalised so $\Sigma_{z=1}^{K} \alpha_z = 1$

- $\mu_z = \frac{1}{T_z} \Sigma_{(i=1)}^{N} w_{iz} x_i$, a weighted mean
- $\Sigma_z = \frac{1}{T_z} \Sigma_{i=1}^{N} w_{iz} (x_i - \mu_z)(x_i - \mu_z)^T$

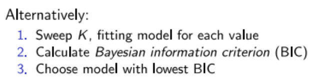
            - Weighted mean
        - Soft update less local minima
        - Variational inference EM without M
        - If K is to large we can overfit
            - To prevent over fitting K is treated as a hyperparameter and a hyperparameter optimisations carries out
            - Can alternatively use Bayesian information criterion
                - Idea penalise the model with more parameters
                - Try's to minimise $\log(N)\,\mathrm{len}(\theta) - 2\log\big(P(x|\theta)\big)$, where N is the number of data points, $len(\theta)$ is the model parameter count, $P(x|\theta)$ is the probability of data given optimal model parameters
                - Crude fails easily, may others, non-parametric techniques better

- K-means
    - Assign points to nearest cluster – E-step without probabilities
    - Update cluster centres to mean of assigned points – M-step
    - EM assigns points to multiple cluster centres
    - EM is generalisation of k-means optimisation
    - EM is probabilistic k-means

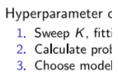
Optimisation basics 3 – L14



- For optimisation of gradient descent solution $w_{t+1} = w_t - \alpha_t \nabla f_w(x)$, Let $g(w) = \nabla f_w(x)$
    - We iteratively update w_t to that it takes the steepest decent
    - Works well for isotropic functions but less well for anisotropic

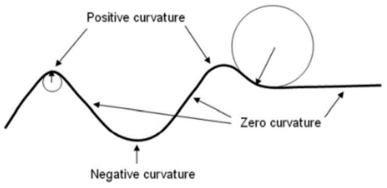


Alternatively:
1. Sweep K, fitting model for each value
2. Calculate *Bayesian information criterion* (BIC)
3. Choose model with lowest BIC



Hyperparameter c
1. Sweep K, fitti
2. Calculate prol
3. Choose model

- Better solution for anisotropic case is Newton's method – use curvature information (second order derivative) to take a more direct route
    - Curvature is the amount by which a curve deviates form a straight line (described by a hessian matrix in certain conditions)
    - Hessian matrix is a square matrix of second order partial derivatives of $f_w(\cdot)$
        - $H(w) = \nabla^2 f_w(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial w_1^2} & \cdots & \frac{\partial^2 f}{\partial w_1 w_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_1 w_M} & \cdots & \frac{\partial^2 f}{\partial w_M^2} \end{pmatrix}$
    - This means that the updating gradient decent formula where step size $\alpha_t = 1$ is $w_{t+1} = w_t - g(w)$
    - For newtons method this becomes
        - $w_{t+1} = w_t - [H(w)]^{-1} g(w)$
        - To minimise w for a random function $f_w(\cdot)$ which is unknown we assume that $f_w(\cdot)$ is much simpler than it really is (this is done using a Taylor approximation)
    - Taylor series: representation of a fnction as an infinite sum of terms calculated from values of the functions derivatives as a singe point
        - $f(x)|_{x_0} = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$
    - Function $f(w)$ approximated at random point w_t at a step t using second order taylor series
        - $f(w) = f(w_t) + \nabla f(w_t)(w - w_t) + \frac{1}{2}(w - w_t)^T \nabla^2 f(w_t)(w - w_t)$ where $\nabla^2 f(w_t) = H(w_t)$
        - $f'(w) = 0 + \nabla f(w_t) + \nabla^2 f(w_t)(w - w_t)$ and set = 0
        - $\therefore -\frac{\nabla f(w_t)}{\nabla^2 f(w_t)} = w - w_t \rightarrow w = w_t - [H(w_t)]^{-1} g(w_t)$

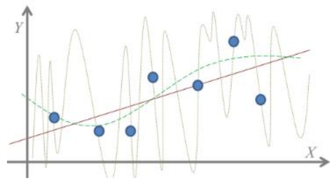


    - The previous gradient decent was $w_{t+1} = w_t - \alpha[2X(X^T w_t - y)]$
        - As $g(w_t) = 2XX^T w_t - 2Xy$ when we then obtain the second derivative with respect to w we get $H(w_t) = 2XX^T$
        - So the gradient decent equation is updated to be $w_* = w_t - (2XX^T)^{-1}[-2Xy + XX^T w_t] = (XX^T)^{-1} Xy$
        - So resulting equation is independent of w_t and thus only requires one step to converge
    - However step size is generally kept as a variable thus
        - $w_{t+1} = w_t - \alpha_t [H(w)]^{-1} g(w)$
        - So no longer a single step however still a good method
    - Gradient decent is preferred over newtons method generally as Hessian is computationally expensive
    - Newtons method:
        - Has quick convergence
        - High complexity(as invers of hessian needs to calculated

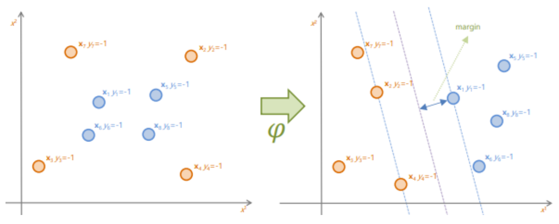
- Local convergence: as based on Taylor
- Requires function to be second-order differentiable and invers exists
- Other variants exists (so as to attempt to avoid the inverse Hessian matrix

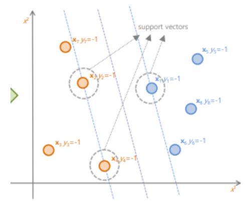
Regularisation 2, kernel methods and SVM:

- Regularisation for linear regression
  - Many ways to fit a function to data this can lead to overfitting (as model can be learnt which fits exactly the training data but dose not generalise to test data as learns the noise as well)
  - Regularisation: modifications are made to the ML algorithms witch is intended to reduce their test error
  - For a data set $D = \{(x_1, y_1), \dots (x_N, y_N)\} \subset X \times Y \subset \mathbb{R}^n \times \mathbb{R}$ the linear function is $f_w(x) = w^T x$ where N is the number of samples
    - Therefore $w_* = \arg\min_{\boldsymbol{w}\in\mathbb{R}^n} \frac{1}{N}\Sigma_{i=1}^N (f_w(x_i) - y_i)^2 = \arg\min_{\boldsymbol{w}\in\mathbb{R}^n} \frac{1}{N}\Sigma_{i=1}^N (w^T x_i - y_i)^2 = \arg\min_{\boldsymbol{w}\in\mathbb{R}^n} ||X^T w - y||^2$
    - so let the objective function be $J(w; X, y) = \left||X^T w - y|\right|^2$
  - the regularised objective function becomes
    - $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \lambda J_{Pen}(\boldsymbol{w})$
    - $J_{Pen}(w)$ is the penalty term aimed at penalising the parameters of interests w to mitigate overfitting (a function with respect to w)
    - $\lambda \in [0, \infty)$ is the regularisation coefficient, controls the relative importance between the data-dependent error ($J(w; X, y)$) and the penalty term
  - Ridge regression (L_2 regularisation)
    - Simplest form of $J_{pen}(w)$ is given by the sum or the squares of $\boldsymbol{w}$ as $J_{Pen}(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{w}$ this is know as L_2 regularisation (also known as weight decay)
    - Therefor complete form is
      - $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \left||\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}|\right|^2 + \lambda\boldsymbol{w}^T\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}) + \lambda\boldsymbol{w}^T\boldsymbol{w}$
    - Regularised solution obtained by setting $\nabla_w \tilde{J}(w; X, y)$ to 0
    - Thus as $w_* = \arg\min_{w\in\mathbb{R}^n}\left(\left||\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}|\right|^2 + \lambda\boldsymbol{w}^T\boldsymbol{w}\right)$ then $w_* = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$ where $\boldsymbol{I}$ is the identity matrix
  - Tikhonov regularisation: Ridge regression is a special case
    - Generalised Tikhonov regularisation minimises the objective function as
      - $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \left||\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}|\right|^2 + \left||\Gamma\mathbf{w}|\right|^2,$ where Γ is the Tikhonov matrix
      - So $w_* = (\boldsymbol{X}\boldsymbol{X}^T + \Gamma^T\Gamma)^{-1}\boldsymbol{X}\boldsymbol{y},$
      - If $\Gamma = 0$ then returns unregularized solution
      - If $\Gamma = \alpha\boldsymbol{I}$ then it gives Ridge regression where $\alpha^2 = \lambda$

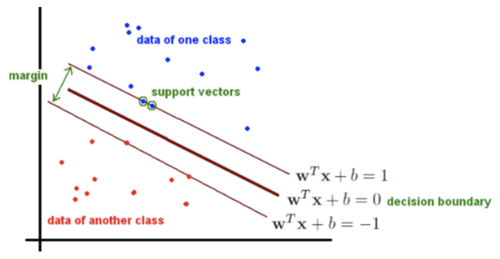


  - for the unregularized solution for high-dimensional problems there are infinitely many solutions
  - for regularised solution $(XX^T + \lambda I)w = Xy$, for $\lambda > 0, XX^T + \lambda I$ has s unique solution
  - other penalty terms exist
    - Generalised form is $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \left||\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}|\right|^2 + \lambda\left||\mathbf{w}|\right|^{\mathbf{q}}$, where q is the norm coefficient
    - $\left||x|\right|_q = (\Sigma_{i=1}^n x_i^q)^{\frac{1}{q}}$
    - q can have any value but we normally only consider 2 which are $q = 2$ leading to L_2 regularisation (ridge regression) and $q = 1$ leading to L_1 regularisation (lasso regression)
  - L_1, L_2 pros and cons
    - Choosing λ is difficult (cross validation often needed)
    - L_1 is biased toward providing a sparse solution
    - L_1 minimises the absolute error while L_2 minimises sum of squared absolute error
    - L_2 is more sensitive to outliers (as squared), and has a unique solution
- Regularisation for linear classification
  - $D = \{(x_1, y_1), \dots (x_N, y_N)\} \subset X \times Y \subset \mathbb{R}^n \times \{0,1\}$, (now y only takes binary values)
  - We are trying to find a function such that
    - $f: \mathbb{R}^n \to \{0,1\}$ which minimises the empirical classification error $J_f = \Sigma_{i=1}^N \mathbf{1}[f(x_i) \neq y_i]$, where $\mathbf{1}[A] = \begin{cases} 1, & if\ A\ true \\ 0, & otherwise \end{cases}$
  - Instead of minimising J_f directly (as this is a complex discrete optimisation problem) we minimise a continuous approximation of $J_f = \Sigma_{i=1}^N I(f(x_i), y_i) = \Sigma_{i=1}^N \max(0, 1 - f(x_i)y_i)$, where $I(a, b) = \max(0, 1 - ab)$ is the hinge loss (plain linear classifier)
    - Interpretation: we want to $f(x_i)$ to have the same signe as label y_i as this makes max be 0, so we want a high confidence ($|f(x_i)y_i| > 1$)
  - Linear support vector machine (SVM): regularised linear classifier



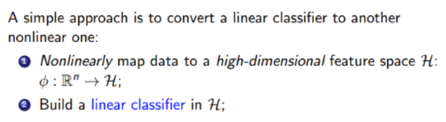


- $\tilde{J}_f = \Sigma_{i=1}^{N} \max(0,1 - f(x_i)y_i) + \lambda||w||^2,\ where\ f(x) = w^T x$
- Representer theorem: the minimise function f can be represented as a final linear combination of kernel products evaluated on training samples
    - From generalised form $w_* = \Sigma_{i=1}^{N}\alpha_i x_i$, with $\{\alpha_i\}_{i=1}^{N}$
        - $f(x) = (w^*)^T x = \Sigma_{i=1}^{N}\alpha_i (x_i)^T \boldsymbol{x} = \Sigma\alpha\langle x_i, \boldsymbol{x}\rangle$ where $\langle a, b\rangle$ is the inner product of a and b


data of one class
margin
support vectors
$\mathbf{w}^T\mathbf{x}+b=1$
$\mathbf{w}^T\mathbf{x}+b=0$ decision boundary
$\mathbf{w}^T\mathbf{x}+b=-1$
data of another class

- Interpretation of the unregularized solution
    - $w_* = (XX^T)^{-1}Xy$, to find the optimal w_* for function $f_w(x) = w^T x$ then for a new point x_{new} we have the new prediction $\hat{f}_{w_*}(x_{new}) = x_{new}^T(XX^T)^{-1}Xy$
    - As the data matrix $X = (x_1, \ldots x_n)$ the new prediction can be rewritten as $\hat{f}_{w_*}(x_{new}) = x_{new}^T(XX^T)^{-1}(x_1, \ldots x_n)y = \Sigma_{i=1}^{N}[x_{new}^T(XX^T)^{-1}x_i]y_i$
    - Then $[x_{new}^T(XX^T)^{-1}x_i]$ as some form of weight (corresponds to all training point and the new point
- The term XX^T performs a standardisation to the data points
- Are computing the inner product of x_{new} and x_i, the larger the inner product value is the smaller the distance in parameter space describing the similarity between them

- Nonlinear regression: kernel methods
    - Sometimes you need to convert form a linear space to a non linear space
    - Done by mapping the linear space to another space (H space) allows for the fitting of a linear boundary this can be done by kernel function
        - higher dimensional space is needed as in low dimensional space things might be mixed in higher dimensions they may separate (increasing the dimensionality gives other potential methods to interpret the angle)
        - How grate the dimensional of H is unknown however we try to minimise this as much as possible as otherwise the curse of dimensionality becomes a problem
    - Nonlinear classifiers minimise $\tilde{J}_f = \Sigma_{i=1}^{N} \max\big(1,1 - f(\phi(x_i))y_i\big) + \lambda||w||^2, f(x) = w^T\phi(x),\ w \in \mathcal{H}$ and $\phi(x)$ is the kernel function
    - Using representer theorem
        - $f^*(x) = (w^*)^T\phi(x) = \Sigma_{j=1}^{N}\alpha\langle\phi(x_j), \phi(x)\rangle, where\ w^* = \Sigma_{j=1}^{N}\alpha_j\phi(x_j)$
    - Therefor the compete nonlinear classifier is given as
        - $\tilde{J}_f = \Sigma_{i=1}^{N} \max\Big(1,1 - \big(\Sigma_{j=1}^{N}\alpha\langle\phi(x_j), \phi(x)\rangle\big)y_i\Big) + \lambda\Sigma_{i,j=1}^{N}\alpha_i\alpha_j\langle\phi(x_i), \phi(x_j)\rangle$
    - Positive definite kernels
        - A symmetric function $k: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is called positive definite if for any N the matrix K formed by evaluating k on any N data points $\{x_1, \ldots x_n\} \subset \mathbb{R}^n$ is positive definite:
            - $[K]_{ij} = k(x_i, x_j)$
    - Kernel methods
        - For a positive definite kernel function there is a non-linear map that maps x to elements of the space of function $\mathcal{H}_k$ such that $\phi(x) = k(x,\cdot) \to \langle\phi(x), \phi(x')\rangle = \langle k(x,\cdot), k(x',\cdot)\rangle = k(x, x')$
        - $\mathcal{H}_k$ is the reproducing kernel Hilbert space corresponding to kernel k


A simple approach is to convert a linear classifier to another nonlinear one:

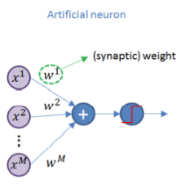
1. *Nonlinearly* map data to a *high-dimensional* feature space $\mathcal{H}$: $\phi : \mathbb{R}^n \to \mathcal{H}$;
2. Build a linear classifier in $\mathcal{H}$;

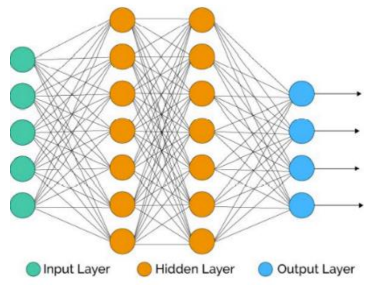
- Gaussian kernel: $k(x, x') = \exp\left(-\frac{||x-x'||^2}{\sigma_k^2}\right), where\ \sigma_k^2$ is a hyperparameter
- Polynomial kernel $k(x, x') = (x^T x' + c)^d$
- Support vector machine: maximises the margin between the decision boundary and the support vector, (determines the optimum decision boundary)
    - $f(x) = \Sigma_{j=1}^{N}\alpha_j\langle\phi(x_j), \phi(x)\rangle$, if x_j is not a support vector then $\alpha_j = 0$

- The decision boundary in its generalised form is a hyperplane defined as $\boldsymbol{w}^T\boldsymbol{x} + b = 0$
    - if $w^T x + b \geq 1$, y=1 then (positive classification results)
    - if $w^T x + b \leq -1$, y=0 then (negative classification results)
    - the distance between a support vector to the hyperplane is $d = \frac{|w^T x + b|}{||w||}$
    - the goal is to minimise $\frac{1}{2}||w||^2$

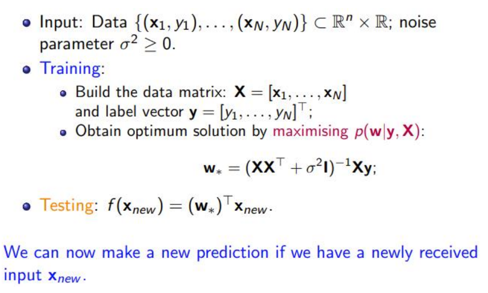
Bayesian Linear Regression and gaussian process – L16

- probabilistic modelling:
    - starts with a simple model with the inclusion of a noise variable
        - makes the assumption that there exists an unknown ground-truth function $f^*(x_i) = y_i$, however measured data/observations are noisy (this is described by ϵ). It is assumed that noise is iid meaning that observation are measured independently and follow the same distributions





            - ϵ is modelled by gaussian distributions $\mathcal{N}(\mu,\sigma^2)$
    - Noise variable ϵ_i represents the deviation between the noisy observation y_i and the model prediction $f(x_i)$ (ie the error)
    - Model with zero noise
        - $y_i = w^T x_i + \epsilon_i, \quad \epsilon_i \sim N(0,\sigma^2)$
        - $\therefore y_i - w^T x_i \sim N(0,\sigma^2)$ giving the likelihood expression as $p(y_i|x_i,w) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$
        - Therefore maximum likelihood estimator becomes $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w}) = \Pi_{i=1}^N p(y_i|x_i,w) = \frac{1}{\sqrt{(2\pi\sigma^2)^N}}\exp\left(-\frac{\left|\left|\boldsymbol{X^T w}-\boldsymbol{y}\right|\right|^2}{2\sigma^2}\right)$
        - ML estimation maximises $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})$ $w_* = \arg\min_{w\in\mathbb{R}^n}\left|\left|X^T w - y\right|\right|^2$ -> $\boldsymbol{XX^T w_*} = \boldsymbol{Xy}$ ML solution equivalent to least-square solution as under iid
        - MAP:
            - Maximise posterior $\boldsymbol{w}_* = \arg\max_w p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y})$, where $p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})$
            - Assume $p(\boldsymbol{w})$ is a gaussian prior: $p(\boldsymbol{w}) = N(0,1) = \frac{1}{\sqrt{(2\pi)^n}}\exp\left(-\frac{\left|\left|\boldsymbol{w}\right|\right|^2}{2}\right)$
            - $p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) = \frac{1}{\sqrt{(2\pi\sigma^2)^N}}\exp\left(-\frac{\left|\left|\boldsymbol{X^T w}-\boldsymbol{y}\right|\right|^2}{2\sigma^2}\right)\frac{1}{\sqrt{(2\pi)^n}}\exp\left(-\frac{\left|\left|\boldsymbol{w}\right|\right|^2}{2}\right) \to \arg\max p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) = \arg\min\left|\left|X^T w - y\right|\right|^2 + \sigma^2\left|\left|w\right|\right|^2$
            - $w_* = \arg\min\left|\left|X^T w - y\right|\right|^2 + \sigma^2\left|\left|w\right|\right|^2 \to (XX^T + \sigma^2 I)w_* = Xy$
            - This returns the results for regularised least-square solution with σ^2 as the regularisation coefficient
- Bayesian linear regression: the predictive distribution is given as $p(y_{new}|x_{new},y,X)$ (describes the distribution of new predictions given new data points)
    - Marginalisation: for a given joint distribution $p(a,b)$ and its marginal distribution $p(a)$ can be obtained by integrating b
        - $p(a) = \int p(a,b)db,\ p(b) = \int p(a,b)\,da$
        - Similarly for joint distribution $p(a,b,c)$: $p(a,c) = \int p(a,b,c)\,db = \int p(a|b,c)p(b|c)\,db$
    - Predictive distribution can be obtained by marginalising w in the product likelihood and posterior



- Input: Data $\{(\mathbf{x}_1,y_1),\dots,(\mathbf{x}_N,y_N)\} \subset \mathbb{R}^n \times \mathbb{R}$; noise parameter $\sigma^2 \geq 0$.
- Training:
    - Build the data matrix: $\mathbf{X} = [\mathbf{x}_1,\dots,\mathbf{x}_N]$ and label vector $\mathbf{y} = [y_1,\dots,y_N]^\top$;
    - Obtain optimum solution by maximising $p(\mathbf{w}|\mathbf{y},\mathbf{X})$:

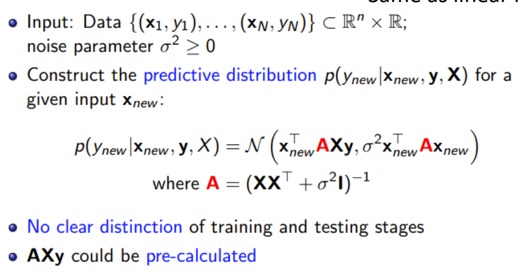
$$\mathbf{w}_* = (\mathbf{XX}^\top + \sigma^2\mathbf{I})^{-1}\mathbf{Xy};$$

- Testing: $f(\mathbf{x}_{new}) = (\mathbf{w}_*)^\top\mathbf{x}_{new}$.

We can now make a new prediction if we have a newly received input $\mathbf{x}_{new}$.

- $P(y_{new}|x_{new},y,X) = \int p(y_{new}|x_{new},w)p(w|y,X)dw$
- $p(y_{new}|x_{new},w) = N(w^T x_{new},\sigma^2)$
- $p(w|y,X) = N((XX^T + \sigma^2 I)^{-1}Xy, \sigma^2(XX^T + \sigma^2 I)^{-1})$
- As these are all in gaussian form $P(y_{new}|x_{new},y,X)$ can be expressed in gaussian form as $N(x_{new}^T(XX^T + \sigma^2 I)^{-1}Xy, x_{new}^T\sigma^2(XX^T + \sigma^2 I)^{-1}x_{new})$
- The mean of this is equivalent to the MAP solution
- Predictive mean: $x_{new}^T(XX^T + \sigma^2 I)^{-1}Xy$
- Predictive variance: $x_{new}^T\sigma^2(XX^T + \sigma I)^{-1}x_{new}$ (under iid gaussian noise the variance is independent of training labels)

- Bayesian non-linear regression: map RV x to a new feature space $\mathcal{F}$ using the nonlinear mapping function $\phi(\cdot)$
    - $f(x) = w^T\phi(x) \to y = f(x) + \epsilon = w^T\phi(x) + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$
    - Using the general gaussian prior $w \sim N(0,\Sigma_w)$
    - The predictive distribution is $p(y_{new}|x_{new},y,\Phi)$ where $\Phi = [\phi(x_1),\dots\phi(x_M)]$
        - $p(y_{new}|x_{new},y,\Phi) = N\left(\frac{1}{\sigma^2}\phi(x_{new})^T A^{-1}\Phi y, \phi(x_{new})^T A^{-1}\phi(x_{new})\right)$, *where* $A = \sigma^{-2}\Phi\Phi^T + \Sigma_w^{-1}$
        - Same as linear however computation needs to invert matrix A, with the complexity corresponding to the number of dimensions M



- Input: Data $\{(\mathbf{x}_1,y_1),\dots,(\mathbf{x}_N,y_N)\} \subset \mathbb{R}^n \times \mathbb{R}$; noise parameter $\sigma^2 \geq 0$
- Construct the predictive distribution $p(y_{new}|\mathbf{x}_{new},\mathbf{y},\mathbf{X})$ for a given input $\mathbf{x}_{new}$:

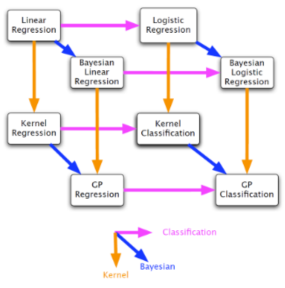
$$p(y_{new}|\mathbf{x}_{new},\mathbf{y},X) = \mathcal{N}\left(\mathbf{x}_{new}^\top\mathbf{AXy}, \sigma^2\mathbf{x}_{new}^\top\mathbf{A}\mathbf{x}_{new}\right)$$

where $\mathbf{A} = (\mathbf{XX}^\top + \sigma^2\mathbf{I})^{-1}$

- No clear distinction of training and testing stages
- $\mathbf{AXy}$ could be pre-calculated

Neural Networks – L17

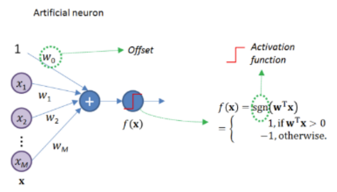
- Neural networks (NN) (Artificial neural networks (ANN)): are computing systems that are inspired by but not identical to biological neural networks
- In the NN structure every node connect with all of the nodes in the previous layer and every node in the next layer
- Based on a collection of units or nodes which loosely model the neurons in a biological brain, each connection (represented by the weights) can transmit a signal to other artificial neurons



- Example:
    - How is a decision made (all of the different factors) (assume all binary)
    - A basic decision rule would be to create a function with weighting which if above a threshold then it outputs true
        - $w_1x_1 + w_2x_2 + w_3x_3 > threshold$ (this requires some prior knowledge)
    - To make a reasonable decision the weights are normalised $(w_1 + w_2 + w_3 = 1)$
    - Weights are decided according to the preferences (grater weight means that it is more important for the decision)

  - Therefore the statement can become useful with the lower the threshold the more likely the true outcome will be (as less conditions need to be true)
  - $y = \begin{cases} 1 & w_1x_1 + w_2x_2 + w_3x_3 > threshold \\ 0 & w_1x_1 + w_2x_2 + w_3x_3 \leq threshold \end{cases}$
    - this can be rewritten as $y = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ 0 & \boldsymbol{w}^T\boldsymbol{x} + b \leq 0 \end{cases}$, where b is a constant offset (negative threshold term) and $\boldsymbol{w} = (w_1, w_2, w_3)^T$ and $\boldsymbol{x} = (x_1, x_2, x_3)^T$ (this is simply separating the hyper-plane as has been done previously)
    - made simpler by letting $b = w_0$ and $x_0 = 1$ then $y = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} + w_0 > 0 \\ 0 & \boldsymbol{w}^T\boldsymbol{x} + w_0 \leq 0 \end{cases} = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} > 0 \\ 0 & \boldsymbol{w}^T\boldsymbol{x} \leq 0 \end{cases} = sgn(\boldsymbol{w}, \boldsymbol{x})$, where $\boldsymbol{w} = (w_0, w_1, w_2, w_3)^T$ and $\boldsymbol{x} = (x_0 x_1, x_2, x_3)^T$ and sgn() is the sign function

- Perceptron
  - Output y=1 is equivalent to sending out an electrical pulse otherwise it is not sent
  - The offset w_0 simply describe the difficulty level for a neuron to send a pulse
  - Learning steps:
    - Given data $D = \{(x_1, y_1), \dots, (x_N, y_N)\} \subset \mathcal{X} \times Y \subset \mathbb{R}^n \times \{0,1\}$
    - The learning rule of a perceptron is:
      - $w_{t+1} = w_t + \alpha(y_n - sgn(w_t, x_n))x_n,\ where\ \alpha$ is the learning rate
      - If output is correct $y_n - sgn(w_t, x_n) = 0$, doesn't change anything
      - If output is low $y_n = 1,\ sgn(w_t, x_n) = 0$, increments weight
      - If output is high $y_n = 0,\ sgn(w_t, x_n) = 1$, decrements weight

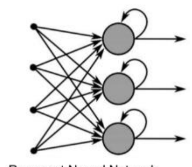

Artificial neuron
1
w_0
Offset
x_1
x_2
x_M
w_1
w_2
w_M
$\mathbf{x}$
$f(\mathbf{x})$
Activation function
$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1, \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1, \text{otherwise.} \end{cases}$

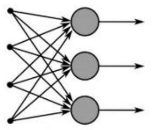
▪Convergence: if data is linearly separable perceptron algorithms will find a linear classifier that classifies all data correctly in finite iterations

○ If a perceptron only sends signals in one direction it is a feed-forward neural network

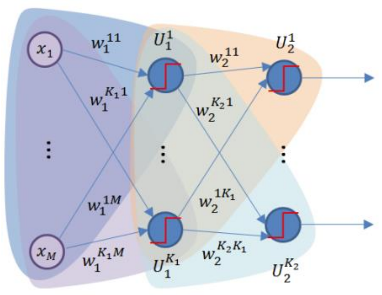
○ If a perceptron can use their internal state to process sequences of inputs (feedback is possible) then it is a Recurrent neural network (passes the signal to its self so has memory of previous state)

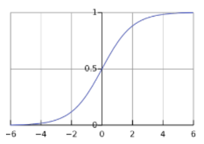
  - Issues of perceptron
    - As it is a linear classifier a perceptron can't solve linearly non-separable problems to solve this MLP can be used
    - The sign function can cause changes of outputs when given small changes of each input as each neuron only has two states (other functions such as sigmoid function can be used instead)

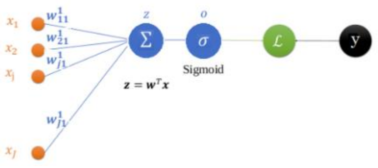

Recurrent Neural Network

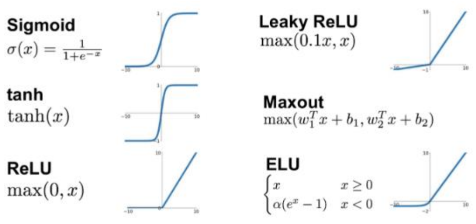

Feed-Forward Neural Network

- Multi-layer perceptron (MLP):
  - Stack multiple perceptron's in to layers to generate non-linear decision boundaries
  - U denotes the layers, K is the index of the units in each layer
- Sigmoid function:
  - $\sigma(w, x) = \frac{1}{1+e^{-w^Tx}}$
  - For values closer to the boundary of the separating hyperplane the output will be closer to 0.5, for other positive or negative values the output will be closer to 0 or 1
  - Any activation function can be used in NN however they need to be differentiable so as to allow back propagation
  - The inclusion of a coefficient α so that $\sigma(w, x) = \frac{1}{1+e^{-\alpha w^Tx}}$ then as α increases the jump becomes steeper
- Back-propagation(BP
  - Allows for updating weight of a neural net
  - Z is the summation, o is the output of the activation function and $\mathcal{L}$ is the loss function
  - $\frac{d\sigma(x)}{dx} = \sigma(1-\sigma)$, the loss function (e.g. MSE) $\mathcal{L} = \frac{1}{2}\Sigma^N(y_i - o_i)^2$ where $o_i = \sigma(z) = \sigma(w^Tx)$
  - We have $\frac{\partial\mathcal{L}}{\partial o_i} = o_i - y_i$
  - BP is to compute the gradient with respect to the weight w using the chain rule for each of the joining parts therefore $\frac{d\mathcal{L}}{dw_{j1}^1} = \frac{d\mathcal{L}}{do} \cdot \frac{do}{dw_{j1}} = (o_i - y_i)\frac{d\sigma(z)}{dw_{j1}} = (o_i - y_i)\sigma(z)(1-\sigma(z))\frac{dz}{dw_{j1}}$
  - As $\frac{dz}{dw_{j1}} = \frac{dw^Tx}{dw_{j1}} = x_j$ then $\frac{d\mathcal{L}}{dw_{j1}^1} = (o_i - y_i)\sigma(z)(1-\sigma(z))x_j$ and as $o = \sigma(z)$
  - $\frac{d\mathcal{L}}{dw_{j1}^1} = (o_i - y_i)o_i(1-o_i)x_j = \delta_1 x_j$


x_1
x_M
w_1^{11}
$w_1^{K_1 1}$
w_1^{1M}
$w_1^{K_1 M}$
U_1^1
$U_1^{K_1}$
w_2^{11}
$w_2^{K_2 1}$
$w_2^{1K_1}$
$w_2^{K_2 K_1}$
U_2^1
$U_2^{K_2}$


1
0.5
-6
-4
-2
0
2
4
6


x_1
x_2
x_j
x_J
w_{11}^1
w_{21}^1
w_{j1}^1
w_{J1}^1
z
Σ
$z = \boldsymbol{w}^T\boldsymbol{x}$
o
σ
Sigmoid
$\mathcal{L}$
y


Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh
$\tanh(x)$
ReLU
$\max(0, x)$
Leaky ReLU
$\max(0.1x, x)$
Maxout
$\max(w_1^Tx + b_1, w_2^Tx + b_2)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$