COMP90051 Statistical Machine Learning

Lecture Notes and Final Revision

Lecturer: Associate Professor Ben Rubinstein

*** by Shijie Huang ***

October 29, 2018

Contents

1 Introduction and Probability Theory

An important hypothesis of Machine Learning

Pre-existing data repositories contain a lot of potentially valuable knowledge

Definition: What is Learning

(Semi-) automatic extraction of **valid, novel, useful and comprehensible knowledge** - in the form of rules, regularities, patterns, constraints or models - from arbitrary sets of data.
The goal is to develop efficient and useful algorithm.

Contents this subject covers:
Foundations of statistical learning, linear models, non-linear bases, kernel approaches, neural networks, Bayesian learning, probabilistic graphical models (Bayes Nets, Markov Random Fields), cluster analysis, dimensionality reduction, regularization and model selection

1.1 Machine Learning Basics

1.1.1 Terminologies

- **Instance:** measurements about individual entities/objects: e.g. a loan application (data points)
- **Attribute (Feature, explanatory variable):** component of the instances: e.g. the applicant's salary, numerics, etc. (x_i)
- **Label (Reponse, dependent variable):** an outcome that is categorical, numbers, etc. (y)
- **Examples:** instance coupled with label, i.e. $< x_i, y >$
- **Models:** discovered relationship between attributes and/or label.

1.1.2 Supervised vs Unsupervised Learning

	Data	Model used for
Supervised Learning	Labelled	Predict labels on new instances
Unsupervised Learning	Unlabelled	Cluster related instances; Project to fewer dimensions; Understand attribute relationships

Evaluations: important for supervised learning

Evaluation principle: measure quality is problem-dependent

Step 1: pick an evaluation metric Accuracy, Contingency table, Precision-Recall, F1, ROC curves

step 2: procure an independent, labelled **test sets**

Step 3: Average the evaluation metric over the test sets.

Cross Validation especially when data is poor.

1.1.3 Probability Theory

Three types of models

$\hat{y} = f(x)$: regressions, best fitting parameters given a model, we model x.
$p(y|x)$: For a given x, we model y as a distribution (likelihood of y given x).
$p(x, y)$: we model (x, y) together, the probability of having (x, y).

Definition: Random Variable

A random variable X is a **numerical function** of outcome $X(\omega) \in R$

Discrete distribution

- Govern r.v. taking discrete values
- Described by **probability mass function** $p(x)$ which is $p(X = X)$
- $P(X \leq x) = \sum_{a=-\infty}^{x} p(a)$
- **Examples:** Bernoulli, Binomial, Multinomial, Poisson

Continuous distribution

- Govern real-valued r.v.
- Described by **probability density function** $p(x)$ which is $p(X = X)$
- $P(X \leq x) = \int_{\infty}^{x} p(a)da$ (Sum from the very left)
- **Examples:** Uniform, Normal, Laplace, Gamma, Beta, Dirichlet

Definition: Bayes' Theorem

In terms of events A, B:

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \Leftrightarrow P(A|B)P(B) = P(B|A)P(A)$$

Bayesian statistical inference makes heavy use of:

- **Marginals:** probabilities of individual variables
- **Marginalisation:** summing away all but r.v.'s of interest (reduce the number of variables/distribution)

2 Statistical Schools of Thought

2.1 Frequentist statistics

Wherein unknown model parameters are treated as having fixed but unknown values.

The problem that we intend to solve

Given: $X_1, X_2, ..., X_n$ drawn i.i.d from some unknown distribution
Want to: identify unknown distribution
Approach: Parametric Approach

- **Some model:** parameterised by parameter sets θ
- **Point estimates:** point estimates $\hat{(\theta)}$ a function or statistics of data.

Definition: Bias-Variance Decomposition

- **Bias:** $B_\theta(\hat{\theta}) = E_\theta[(\hat{\theta}(X_1, ...X_n)] - \theta$
- **Variance:** $Var_\theta(\hat{\theta}) = E_\theta[(\hat{\theta} - E_\theta[\hat{\theta}])^2]$
- **Bias-variance decomposition of square loss:**

$$E_\theta[(\theta - \hat{\theta})^2] = [B(\theta)]^2 + Var_\theta(\hat{\theta})$$

What we really care is the squared loss, which contains both bias and variance. (Notice that empirically, we care about the expected loss, since we don't know the true distribution, we use the distribution of the sample to approximate).

Asymptotic properties

Consistency: $\hat{\theta}(x_1, ...X_N)$ converges to true θ as $n \to \infty$ (i.e., if we increase the sample size to infinity, does the estimated distribution converge to the true distribution?)
Efficiency: asymptotic variance is as small as possible.

The Approaches: Maximum Likelihood Estimation (MLE)

Algorithm 1 Maximum Likelihood Estimation (MLE)

```
1: We have a set of data $(X_1, ..., X_n)$
2: We propose a distribution, $p_\theta$, which is assumed to have generated the data (i.e. we assume
   that the data is generated by using this distribution function)
3: Express the likelihood of the data:
       $$\prod_{i=1}^{n} p_\theta(X_i)$$
4: Apply log trick
5: Optimise to find the best (most likely) parameters $\theta$: two ways to do that
6: (1) F.O.C: take partial derivatives of log likelihood w.r.t $\hat{\theta}$
7: (2) iterative processes, Newton method etc.
```

2.2 Bayesian Statistics

Wherein unknown model parameters have associated distributions reflecting prior belief.

Key idea: **Probabilities $\Leftrightarrow$ beliefs**

Bayesian Machine Learning:

Step 1: Start with prior $P(\theta)$ and likelihood $P(X|\theta)$

step 2: Observe data $X = x$

Step 3: Update prior to posterior $P(\theta|X = x)$

Definition: Bayes' Theorem

In terms of events A, B:

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \Leftrightarrow P(A|B)P(B) = P(B|A)P(A)$$

Bayesian statistical inference makes heavy use of:

- **Marginals:** probabilities of individual variables
- **Marginalisation:** summing away all but r.v.'s of interest (reduce the number of unwanted variables/distribution)

$$P(X = x) = \sum_t P(X = x, \theta = t)$$

Parametric	**Non-Parametric**
Determined by fixed finite number of parameters	Number of parameters grows with data potentially infinite
Limited Flexibility	More flexible
Efficient statistically and computationally	Less efficient

2.3 Parametric vs. None-parametric models

2.4 Generative vs. Discriminative models

Generative approach: model full joint $P(X, Y)$
Discriminative approach: model conditional $P(Y|X)$ only.

3 Linear Regression and Optimization

Advantage of Linear Regression

1. Simple model (convenient maths at expense of flexibility.
2. Often needs less data, "interpretable", lifts to non-linear.
3. Derivable under all Statistical Schools.

3.1 Linear Regression via Decision Theory

Idea: I've got a loss and I want to choose a model (decision rule) to minimize the loss.

Definition: Linear Regression

Assume that there is a linear relationship between response $y \in R$ and an instance with features $x_1, ...x_m \in R$. A linear regression model is:

$$\hat{y} = w_0 + \sum_{i=1}^{m} x_i w_i \Leftrightarrow \hat{y} = x^{'} w$$

3.2 Linear Regression via Frequentist Probabilistic Model

Definition: Probabilistic Linear Regression

$$Y = X^{'} w + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2)$, i.e. noise (encoded by ε) is normally distributed.
A discriminative model is therefore:

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

MLE for Parametric Probabilistic Model

We want to find parameter values that "best" explain the data, MLE: choose parameter values that maxmize the probability of observed data.
Assume independence of the data points, the probability of data is

$$p(y_1, ...y_n|x_1, ...x_n) = \prod_{i=1}^{n} p(y_i|x_i)$$

Applying the log trick (we maximize the log, which is equivalent)

$$\sum_{i=1}^{n} \log p(y_i|x_i) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i^{'} w)^2 + C$$

Note here maximizing the log likelihood as a function of w is equivalent to minimizing the sum of squared errors.

3.3 Non-Linear continuous Optimization

Typical formulation of Optimization in Machine Learning

$$\hat{\theta} \in argmin_{\theta \in \Theta} L(data, \theta)$$

θ, Θ: model parameters and model family.
L : Objective function (MLE: conditional likelihood, Decision Theory: (regularized) empirical risk)
Two ways to solve optimization problem: (1) analytical: F.O.C (2) Approximate iterative solution.

Algorithm 2 Coordinate descent (update parameters from one dimension at a time)

```
1: Choose $\theta = [\theta_1, ...\theta_k]^{'}$, k parameters/dimensions
2: choose some starting point $\theta^{(1)}$ and some T
3: **for** i from 1 to T **do**
4:     $\theta^{(i+1)} = \theta^{(i)}$
5:     **for** j from 1 to K (each dimension) **do**
6:         Fix components of $\theta^{i+1}$, except $j$-th component(dimension)
7:         Find $\hat{\theta_j^{(i+1)}}$ that minimises objective function
8:         Update $j$-th component of $\theta^{(i+1)}$
9:     **end for**
10: **end for**
```

η : step size and ∇direction of derivatives

Algorithm 3 Gradient Descent

```
1: Choose $\theta^{(1)}$ and some T
2: for i from 1 to T do
3:     $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla L(\theta^{(i)})$
4: end for
```

3.4 Affine and Convex Sets

Affine Set

An affine set is a set that contains the line through any two distinct points in it.

How to prove? Pick a linear combination of the other two points, prove that the point is on the same line

A function f is linear if $f(ax + by) = af(x) + b$ for all relevant values of a, b, x and y.

A function g is affine if $g(x) = f(x) + c$ for some linear function f and constant c. Note that we allow $c = 0$, which implies that every linear function is an affine function.

Affine transformation: a point multiply by matrix and add a vector to it.

Convex Set

A convex set contains the line segment between any two points in the set (i.e. connect two points, the line is in the set)

How to prove? Pick a line $z = \theta x_1 + (1 - \theta)x_2$, prove z is in the same convex set

Convex Combination

$$x = \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n, \quad \theta_1 + ...\theta_n = 1, \quad \theta_i \geq 0$$

Convex Hull

Set of all convex combinations of members of a set (i.e. choose all possible convex combinations, and convex hull is the smallest convex set that contains the original set) [a]

[a]Note original set is not necessarily convex, but convex hull is

Convex Cone

A convex cone is a set that contains all conic combinations of the points in the set.
Additionally, a convex cone is a **proper cone** if:
1. it is closed (Contains its boundary);
2. it is solid (has non-empty interior;
3. it is pointed (contains no lines)
e.g. Positive semi-definite cone:

$$\{X \in R^{n\times n} | X = X^T, v^T X v \geq 0. \forall v \neq 0\}$$

3.4.1 Excursion: (Semi) Definite Matrices

Definition ((Semi) Definite Matrices)

A **symmetric matrix** P [a], is positive (semi-) definite, $P \succ 0 (P \succeq 0)$ iff:

$$(1)\ \forall v \neq 0, v^T P v > 0 (v^T P v \geq 0)$$

(2) or equivalently the smallest eigen-value of matrix $\lambda_{\min}(P) > 0 (\lambda_{\min}(P) \geq 0)$

[a] $P = P^T$

Definition (Generalised Matrices Inequality)

$$X \leq Y \Leftrightarrow x_i \leq y_i, \quad i = 1, ..., n$$

3.4.2 Convex Sets (Continued)

Some convex sets and relative proofs

How to prove?
1. pick any two points that satisfy the definition
2. show their linear combination satisfy convex relationship

1. Hyperplane: $\{x | a^T x = b\}$ (affine: a system of linear equations, and all solutions of it consist of an affine set)
2. Halfspace: $\{x | a^T x \leq b\}$

$$\text{Assume } z = \theta x_1 + (1-\theta) x_2 \text{ where } a^T x_1 \leq b \text{ and } a^T x_2 \leq b$$

$$\begin{aligned} a^T z &= a^T(\theta x_1) + a^T(1-\theta) x_2 \\ &= \theta a^T x_1 + (1-\theta) a^T x_2 \end{aligned}$$

$$\leq \theta b + (1-\theta)b = b$$

3. Norm balls with centre x_c and radius r:

$$B(x_c, r) = \{x | ||x - x_c|| \leq r\}$$

Proof:

$$\text{Assume } x_1, x_2 \text{ where}$$

$$||x_1 - x_c|| \leq r$$

$$||x_2 - x_c|| \leq r$$

$$||\theta x_1 + (1-\theta)x_2 - x_c||$$

$$||\theta x_1 - \theta x_c + (1-\theta)x_2 - (1-\theta)x_c||$$

$$||\theta(x_1 - x_c) + (1-\theta)(x_2 - x_c)||$$

By Triangle inequality $||x + y|| \leq ||x|| + ||y||$

$$\leq \theta||x_1 - x_c|| + (1-\theta)||x_2 - x_c||$$

$$\leq \theta r + (1-\theta)r$$

$$\Rightarrow ||\theta x_1 + (1-\theta)x_2 - x_c|| \leq r$$

3.5 Convex Function

Definition (Convex Function)

A function $f : D \rightarrow R$ is convex if:

1. D is convex and
2. $\forall x, y \in D,\ \theta \in [0,1] : f((1-\theta)x + \theta y) \leq$ [a] $(1-\theta)f(x) + \theta f(y)$

[a] if **strictly convex (not strong!)**, the sign is $<$

Definition (Epigraph)

$$epi(f) = \{(x, s) | x \in D, s \geq f(x)\}$$

Theorem (epigraph of f)

The epigraph of a function is convex iff the function itself is convex

Definition (Strong Convexity)

A function $f : D \to R$ is strongly covex with coefficient (modulus) σ if:

$$(1)\ D \text{ is convex}$$

$$(2)\ f((1-\theta)x+\theta y)+\frac{\sigma}{2}\theta(1-\theta)||x-y||^2 \leq (1-\theta)f(x)+\theta y$$

for all $\theta \in [0,1]$ and $x, y \in D$, and some $\sigma > 0$

Note that any **strongly convex** function is **strictly convex** but **NOT** vice versa.

Properties and Results

If f is differentiable:

$$(1)\ (\nabla f(x)-\nabla f(y))^T(x-y) \geq \sigma||x-y||^2$$

$$(2)\ f(y) \geq f(x)+\nabla f(x)^T(y-x)+\frac{\sigma}{2}||x-y||^2$$

If f is twice differentiable:

$$\nabla^2 f(x) - \sigma I \geq 0\ ^{a}$$

[a]Hessian, curvature never goes to zero (bounded below by a >0 number)

3.6 Convex Optimization

Definition (Convex Optimization Problem)

The optimization problem: $\min_x f(x), \quad s.t. x \in X$
If the following holds:
(1) f is a convex function
(2) X is a convex set
Then the problem is called **convex optimization problem**

Note the problem is equivalent to: $\max -f(x), \quad s.t.\ \ x \in X$ if $-f$ is **concave function**.

Theorem (Local Implies Global Optimality for Convex Problem)

If an optimization problem is a convex optimization problem, then every **local minimum** is also a **global minimum**

Also note that if the function is strictly convex in the neighbourhood of the solution, the minimiser is unique (strict inequality).
For a convex optimization problem either there is a unique minimiser or the minimisers form a convex set ($\Rightarrow$ theorem).

Theorem (Convexity of the Set of Minimisers)

Consider the convex optimization problem described above. Then the set of its minimisers is convex.

Theorem (Convexity for Differentiable Functions)

If $f : D \to R$ is continuously differentiable, and D is a convex set, then f is convex iff the following holds:

$$\forall x, y \in D, \quad f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

Or in other words, tangent below the graph.

Theorem (Convexity for Twice Differentiable Functions)

If $f : D \to R$ is **twice** continuously differentiable, and D is a convex set, then f is convex iff the following holds:

$$\forall x \in D : \nabla^2 f(x) \geq 0$$

Theorem

The function $f : D \to R$ is convex iff:
$g : D_g \to R$ where $g(t) = f(x + tp)$ and $D_g = \{t | x + tp \in D\}$ is convex for any line restriction, i.e. for any $x \in D$ and $p \in R^n$.

3.6.1 Operations That Preserve Convexity

Operations That Preserve Convexity	
Input Transformation	
Extension	
Summation	
Point-wise Maximum	
Point-wise Supremum	
Minimisation	
Composition	
Persepective	

3.7 Standard Form of Convex Optimization Problems

An (constrained) optimization problem:

$$\min_x f(x)$$

$$s.t. \quad c_i(x) = 0, \quad i \in E, \quad c_i(x) \geq 0, \quad i \in I$$

Sufficient condition for the above to be convex problem

(1) f is convex
(2) $c_i(x)$ is affine for $i \in E$ (Or $Ax = b$)
(3) $c_i(x)$ is concave for $i \in I$

3.8 Optimality Condition for Convex Problems

Theorem (First Order Optimality (both sufficient and necessary) Condition for Convex Problems)

If $f(x)$ is continuously differentiable, and we have the following problem

$$\min_x f(x), \quad s.t. \;\; x \in X$$

A point x^* is a global minimiser iff:

$$\forall x \in X, \nabla f(x^*)^T(x - x^*) \geq 0 \quad ^a$$

[a]The value of the function will only increases in the direction of any derivatives (for all x)

Theorem (Unconstrained Convex Problems)

If $f(x)$ is continuously differentiable, and we have the following problem

$$\min_x f(x), \quad s.t. \;\; x \in X$$

A point x^* is a global minimiser iff:

$$\nabla f(x^*) = 0$$

Gradient Descent on (strictly) convex function guaranteed to find a (unique) global minimum.

3.8.1 L1 and L2 Norm

Definition: L1 and L2 Norm

L1: Manhattan distance
$$||a||_1 = |a_1| + ... + |a_n|$$

L2: Euclidean distance
$$||a||_2 = \sqrt{a_1^2 + ... + a_n^2}$$

In linear regression:
$$L = ||y - X'w||_2^2$$

Analytical Solution for Linear Regression

$$L = ||y - xw||_2^2$$

$$\frac{\partial L}{\partial w} = 2x^T(y - xw) = 0$$

$$x^T y - x^t xw = 0$$

$$x^T y = x^T xw$$

$$w = (x^T x)^{-1} x^T y$$

4 Logistic Regression and Basic Expansion

4.1 Logistic Regression: Binary Classification

Definition: Logistic Regression Model

$$P(Y=1|x) = \frac{1}{1+exp(-x^{'}w)}$$

1. Logistic regression is a linear classifier, because the decision boundary is linear.
2. Decision boundary: line(2-d) or plane(3-d) or hyperplane(higher-d) where $P(Y = 1|x) = 0.5$.

Linear Regression

- Assume **normal distribution** with a fixed variance and mean given by linear model
- $p(y|x) = Normal(x^{'}w, \sigma^2)$

Logistic Regression

- Assume a **Bernoulli distribution** with parameter given by logistic transform of linear model.
- $p(y|x) = Bernoulli(logistic(x^{'}w)$
- Bernoulli: $p(y) = \theta^y(1-\theta)^{(1-y)}$ for $y \in \{0,1\}$

$$\log(\prod p(y_i|x_i)) = \sum_{i=1}^{n} \log p(y_i|x_i)$$

$$= \sum_{i=1}^{n} \log((\theta(x_i)) + (1-y_i)\log(1-\theta(x_i)))$$

$$= \sum_{i=1}^{n} ((y_i-1)x_i^{'}w - \log(1+\exp(-x_i^{'}w)))$$

Note that this equation has no analytical solution, so we have to use iterative approach. However, the good news is that this is a strictly convex function.

4.2 Logistic Regression: Decision-Theoretical View

Definition: Cross Entropy

A measure of a divergence between reference distribution $g_{ref}(a)$ and estimated distribution $g_{est}(a)$.

$$H(g_{ref}, g_{est}) = -\sum_{a\in A} g_{ref}(a)\log g_{est}(a)$$

Logistic Regression training as a cross-entropy minimization

Reference (true) distribution:
$g_{ref}(1) = y_i$ and $g_{ref}(0) = 1 - y_i$
Estimated distribution:
$g_{est}(1) = \theta(x_i)$ and $g_{est}(0) = 1 - \theta(x_i)$
Log-likelihood for a single data-point:

$$\log p(y_i|x_i) = y_i \log(\theta(x_i)) + (1 - y_i)\log(1 - \theta(x_i))$$

Cross Entropy:

$$H(g_{ref}, g_{est}) = -\sum_a g_{ref}(a) \log g_{est}(a)$$

Hence, doing MLE to train logistic regression is equivalent to minimize cross entropy between true distribution and estimated distribution.

4.3 Basis Expansion

Idea: Extending the utility of models via data transformation

Problem and solution

Problem: data is likely to be non-linear by itself
Question: How can we apply linear method (which is convenient) on non-linear data?
Solution: Transform the data first.
Data transformation aka basis expansion: Map data onto another feature space, where the transformed data is linear in that space.

Definition: Radial Basis Function (RBFs)

$$\phi(x) = \psi(||x - z||)$$

where z is a constant

Challenges: (1). Transformation needs to be defined beforehand. If RBF, z_i needs to be defined beforehand (uniformly spaced points, or cluster training data and use cluster centroids)

5 Regularization

Idea: Process of introducing additional information in order to solve an ill-posed problem or to prevent over-fitting.

Problems addressed

(1) Avoids ill-conditioning (irrelevant features, co-linearity, lack of data)
(2) Introduce prior knowledge
(3) Constrain modelling (avoid over-fitting)

5.1 Regularization in Linear Models

5.1.1 Co-linearity

Definition: Co-linearity

For linear models, feature x_j is irrelevant if x_j is a linear combination of other columns, for some scalars α_l

$$X_j = \sum_{l \neq j} \alpha_l x_l$$

Near-irrelevance (colinearity) is also problematic.

Why ill-conditioning is a problem?

Recall normal equations,

$$\hat{w} = (X^{'}X)^{-1}X^{'}y$$

with co-linearity, inverse of $X^{'}X$ is not defined, because its eigenvalue $= 0$. Hence, solution $\hat{w}$ is not defined.

How is regularization going to help?

Original Problem

- We want to minimise

$$||y - Xw||_2^2$$

- If co-linearity among x's, solution is not defined

Regularized Problem

- We want to minimize

$$||y - Xw||_2^2 + \lambda||w||_2^2$$

- This is called **Ridge Regression**
- Solution is now

$$\hat{w} = (X^{'}X + \lambda I)^{-1}X^{'}y$$

Note: Effectively we add $\lambda > 0$ to the diagonal of the matrix to make the matrix invertible.

5.1.2 Regulariser as a prior

Recall the model:

$$Y = x^{'}w + \varepsilon$$

where Y and ε are both random variables.
Now we suppose that w is also a random variable (This is the prior knowledge that we impose on w).

$$W \sim N(0, \lambda^2)$$

Then the prior is used to compute posterior

$$p(w|X,y) = \frac{p(y|X,w)p(w)}{p(y|X)} \Leftrightarrow posterior = \frac{likelihood * prior}{marginal\ likelihood}$$

Take **maximum a posteriori (MAP)** [1],

$$\hat{\theta}_{MAP}(x) = argmax_{\theta} f(x|\theta)g(\theta)$$

and apply log trick on the posterior function:

$$\log(posterior) = \log(likelihood) + \log(prior) - \log(marginal\ likelihood)^2$$

which is equivalent to

$$||y - Xw||_2^2 + \frac{1}{\lambda}||w||_2^2 \quad ^3$$

5.2 Regularization in Non-linear Models

How to 'vary' model complexity

(1) Explicit model selection:
Split data into train and test sets, use different models and apply held out validation/cross validation on all trained models, choose the best model
(2) Regularization
Augment the problem into:

$$\hat{\theta} \in argmin_{\theta \in \Theta}(L(data, \theta) + \lambda R(\theta))$$

Note that the regularizer $R(\theta)$ does not depend on data; Use held out validation/cross validation to choose λ.

[1] or the **mode** of the posterior distribution
[2] Note that the last term is hard to get because we need to integral out the w
[3] Note that marginal likelihood does not affect optimization.

5.3 Regularization as a constraint

Idea: constrained optimization

$$\min ||y - Xw||_2^2$$

$$s.t. \ \ ||w||^2 \leq \lambda \ \ \lambda > 0$$

Algorithm	**Minimises**	**Regularizer**	**Solution**
Linear	$\|\|y - Xw\|\|_2^2$	None	$(X^{'}X)^{-1}X^{'}y$
Ridge	$\|\|y - Xw\|\|_2^2 + \lambda\|\|w\|\|_2^2$	L_2 Norm	$(X^{'}X + \lambda I)^{-1}X^{'}y$
Lasso	$\|\|y - Xw\|\|_2^2 + \lambda\|\|w\|\|_1$	L_1 Norm	No closed form solution, iterative process required, solutions are space and suitable for high-dim data

5.4 Bias-Variance Tradeoff

Idea: Explore the relationship between training error, test error and model complexity.

Consider squared loss :

$$L(Y, f(\hat{x_0})) = (Y - f(\hat{x_0}))^2$$

$$E(L) = (E[Y] - E[\hat{f}])^2 + var[\hat{f}] + var[Y]$$

$$\text{Risk/Test error} = (bias)^2 + variance + irreducible\ error$$

Note that the irreducible error $var[Y]$ is independent of X.

Trade-off

Simple Model: high bias, low variance.

Complex Model: low bias, high variance.

6 Perceptron

Idea: A building block for artificial neural networks, yet another linear classifier.

Definition: Artificial Neural Network

(1) A network of processing elements

(2) Each element converts inputs to output

(3) Output is a function(called activation function) of a weighted sum of inputs.

Perceptron

$$s = f(\sum_{i=1}^{n} x_i w_i)$$

(1) It is a **binary classifier**: decision boundary is binary

(2) It is a **linear classifier**: decision boundary is linear (2-d: line, 3-d: plane, higher-d: hyperplane)

Loss function for Perceptron

(1) If correctly classified: $L(s, y) = 0$

(2) If wrong side or misclassified: $L(s, y) = |s|$

$$L(s, y) = \max(0, -sy)^a$$

[a] assume $y \in \{-1, 1\}$

Algorithm 4 Stochastic Gradient Descent

```
1: Split all training examples in $B$ batches.
2: choose some starting point $\theta^{(1)}$ and some *epoch* T [a]
3: for i from 1 to T do
4:     for j from 1 to B do
5:         Do gradient descent update using data from batch $j$
6:     end for
7: end for
```

[a] Going through the entire dataset once is called an *epoch*

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

$$s = \sum_{i=10}^{m} x_i w_i$$

Algorithm 5 Perceptron training algorithm

```
1: randomly choose some starting point $w^{(0)}$ and some *epoch* T, K=0 [a]
2: for i from 1 to T do
3:     for j from 1 to N do
4:         consider example $\{x_j, y_j\}$
5:         update: $w^{k+1} = w^k - \eta \nabla L(w^k)$
6:     end for
7: end for
```

[a]Going through the entire dataset once is called an *epoch*

Convergence Theorem for Perceptron

If the training data is linearly separable [a], the algorithm is guaranteed to converge to a solution. That is, there exist a finite K such that $L(w^K) = 0$.

[a] This is an important and strong assumption, otherwise the algorithm will fail

7 Multilayer Perceptron and Backpropagation

Structure of Artificial Neural Networks

- Artificial Neural Network
  - Recurrent Neural Networks
  - Feed-Forward Networks
    - Convolutional Neural Networks
    - Multi-layer perceptrons
      - perceptrons

7.1 Multi-Layer Perceptron

Idea: modelling non-linearity via function composition

Definition: Function Composition

Simple example:
$f(g(h(x)))$

First hidden layer:

$$r_j = \sum_{i=10}^{m} x_i v_{ij}$$

Second (hidden) layer:

$$u_j = g(r_j)$$

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

So, how do we train the parameters?

First, we have to define loss function.

Loss Function for ANNs

$$L(\theta) = \frac{1}{2}(\hat{f}(x,\theta) - y)^2 = \frac{1}{2}(z-y)^2 \quad [a]$$

$L(\theta)$ is differentiable, however, no analytical solution. Hence, iterative approach.

[a] $\hat{f}$ is a composition function

Algorithm 6 Stochastic Gradient Descent for ANN

```
1: randomly choose some starting point $w^{(0)}$ and some *epoch* T [a]
2: for i from 1 to T do
3:     for j from 1 to N do
4:         consider example $\{x_j, y_j\}$
5:         update: $\theta^{i+1} = \theta^i - \eta\nabla L(\theta^i)$
6:     end for
7: end for
```

[a]Going through the entire dataset once is called an *epoch*

We know the loss function, but we need to calculate the partial derivative ∇L. Let $z = h(s)$

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

Recall that

$$u_j = g(r_j)$$

$$r_j = \sum_{i=10}^{m} x_i v_{ij}$$

Hence,

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

Forward propagation and Backward propagation of Errors

Forward Propagation: Use current estimates of v_{ij} and w_j to calculate r_j, u_j, s and z
Backward Propagation: calculate the errors (or loss) and use that to update backwards the parameters gradient.

Problem with ANNs

ANN is very flexible, but complicated models can lead to **over-fitting**.
Solutions:
(1) Implicit regularization: early stopping (to avoid local optima in ANNs, which is typically non-convex function).
(2) Explicit regularization (add penalties):

$$\min L + \lambda(\sum_{i=0}^{m}\sum_{j=1}^{p} v_{ij}^2 + \sum_{j=0}^{p} w_j^2)^{a}$$

This simply add $2\lambda v_{ij}$ and $2\lambda w_j$ terms to the partial derivatives.

[a]in this case, l2 norm

8 Deep Learning, Convolutional ANNs and Autoencoders

8.1 Deep Learning and Representation Learning

Idea: Hidden layers viewed as feature space transformation.

About ANNs with one hidden layer

(1) ANNs with a single hidden layer are **universal approximators.**, i.e. with lots of nodes, technically one can represent everything (Think about Fama-French three and five factor model, and how multi-factor model can effectively explain everything if we have finite/infinite factors in the true model.
(2) A more efficient way is to stack several hidden layers (although in this case you may need more data to learn).

Definition: Deep ANN (Deep Learning) as representation learning

The key idea here is that each hidden layer is learning more complex representation of the previous hidden layer. (Think Computer Vision, hidden layer 1: dots, hidden layer 2: edges, hidden layers 3: lines, circles, etc...)
An ANN can have a simple **linear** output layer, but using complex **non-linear** representations (Composition of functions).

$$z = \tanh(D^{'}(\tanh(C^{'}(\tanh(B^{'}(\tanh(A^{'}x)))))))$$

where $D^{'}, C^{'}, B^{'}$, and $A^{'}$ are learned weight parameters for each hidden layer.
Equivalently, think a hidden layer as the transformed feature space

$$u = \psi(x)$$

Model depth vs width

In theory, a single infinitely wide layer gives a universal approximatior.
Empirically, depth yields more accurate models.
Biological inspiration from eyes:
(1) first detect small edges and color patches;
(2) compose these into smaller shapes;
(3) Building to more complex detectors, of e.g., text, faces, etc.

However, depth cause a big problem: **Vanishing gradient problem**
With gradient based method, the problem occurs when the value out of the activation function is within the *flat zone*, which causes the gradient to be extremely small (likely to be 0), and as a result, the weight parameters will not get updated, causing the algorithm to stuck in the local optimum.

8.2 Convolutionary Neural Networks (CNN)

Idea: Based on repeated application of small filters to patches of a 2D image or range of a 1D input.

Components of a CNN

(1) Convolutional Layers:

1. Complex input representations based on **convolution operation**
2. Filter weights are learned from training data, each layer use the same filter weights (slide window)

(2) Downsampling, usually via **Max Pooling**:

1. Re-scales to smaller resolution, limits parameter explosion

(3) Fully connected parts and output layer:

1. Merges representations together.

8.3 Auto-encoder

Idea: An ANN training setup that can be used for unsupervised learning, initialisation, or just efficient coding

Auto-encoder

You only have x_i, no labels y_i, you set:

$$y_i = x_i$$

Then you build an ANN model, set the hidden layer u to be "thinner" than the input layer. **Input:** X, **output:** X, effectively, you want to train an ANN to predict $z(x_i) \approx x_i$. Once you train the network, you keep only lower-dimension representation u only. As a result, the data structure can be effectively described (**encoded**) by a lower dimensional representation u.

(1) Auto-encoders can be used for **compression** and **dimensionality reduction** via a non-linear transformation.

(2) If **activation function** = **linear**, and only **one hidden layer**, then the set up is almost a **Principal Component Analysis**.

9 Support Vector Machines

Assumption: data is linearly separable

9.1 Maximum-Margin Classifier: Hard Margin SVM

Perceptron

- Minimize the perceptron loss.

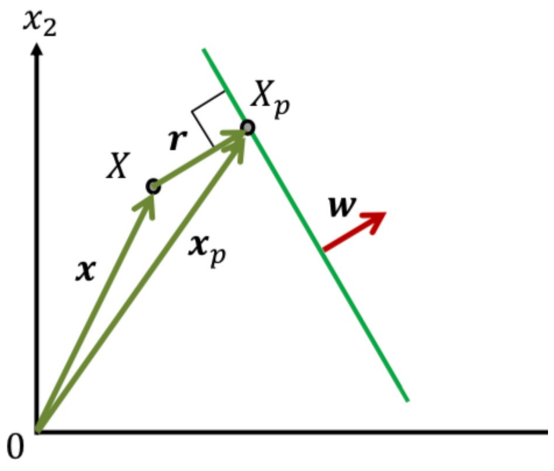
SVM

- choose parameters such that the margin (defined later) is maximize.

Definition: Margin Width

Distance between the separating boundary (hyperplane) and the nearest point(s)

What are we maxmizing then? Some maths



Assume that X is the closet point from the LHS of the hyperplane.

$$r = x_p - x$$

vector r and vector w generally have different length, however, same direction.

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

$$\Rightarrow \frac{w}{||w||}||r|| = x_p - x$$

Multiply both sides of the equation by w^T,

$$w^T \frac{w}{||w||}||r|| = w^T x_p - w^T x$$

$$||w||||r|| + w^T x = w^T x_p$$

$$\Rightarrow ||w||||r|| + w^T x + b = w^T x_p + b$$

Remember that point X_p is on the hyperplane, therefore, $w^T x_p + b = 0$.

$$\Rightarrow ||w||||r|| + w^T x + b = 0$$

$$\Rightarrow \textbf{distance}\text{: } ||r|| = -\frac{w^T x + b}{||w||}$$

If point X is on the RHS of the hyperplane, then the equation/distance becomes

$$||r|| = \frac{w^T x + b}{||w||}$$

Recall that in categorization setting, label $y_i = \{1, -1\}$, generalize the equation:

$$||r|| = \frac{y_i(w^T x + b)}{||w||}$$

Challenge: Multiple hyperplanes

There could be multiple optimal hyperplanes, we only want one. One way to resolve ambiguity: measure the distance to the closet point ($i*$), and rescale parameters such that:

$$||r|| = \frac{y_i(w^T x + b)}{||w||} = \frac{1}{||w||}$$

Then the problem becomes that we want to find the maximum of $\frac{1}{||w||}$.

Definition: Hard Margin SVM

$$argmin_w ||w||$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 \quad ^a$$

Note 1: parameter b is optimized indirectly by influencing constraints.

Note 2: All points are enforced to be on or outside the margin (**i.e. hard-margin**)

[a] We want the points to be (strictly) outside the boundary

Hard Margin SVM as Empirical Risk Minimization

Recall Ridge Regression:

$$\min \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda ||w||^2$$

Compare to Hard Margin SVM:

$$argmin_w ||w||$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1$$

The constraints can be explained as loss:

$$l_\infty \begin{cases} 0 & 1 - y_i(w^T x + b) \leq 0 \\ \infty & 1 - y_i(w^T x + b) > 0 \end{cases}$$

Think it as a penalty method, a **hard margin** means that any violation will be penalized extremely whereas if the constraint is satisfied, the penalty is 0.

10 Soft-Margin SVM, Lagrangian Duality

10.1 Soft-Margin SVMs

Idea: We allow some misplaced points, to address linear inseparability.

When data is not linearly separable

Real data is unlikely to be linearly separable, therefore, hard-margin SVM is in trouble.
In order to solve this problem, 3 approaches:
1. First transform the data, then apply hard-margin SVM (kernelization)
2. Relax the constraint (soft-margin)
3. Combo of 1 and 2.

Recall **Hard-Margin Loss**:

$$argmin_w ||w||$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1$$

The constraints can be explained as loss:

$$l_\infty \begin{cases} 0 & 1 - y_i(w^T x + b) \leq 0 \\ \infty & 1 - y_i(w^T x + b) > 0 \end{cases}$$

We relax the constraint: Soft-Margin SVM loss (hinge loss):

$$l_h \begin{cases} 0 & 1 - y_i(w^T x + b) \leq 0 \\ 1 - y_i(w^T x + b) & 1 - y_i(w^T x + b) > 0 \end{cases}$$

$$\Leftrightarrow l_h = \max(0, 1 - y_i(w^T x + b))$$

Soft-Margin SVM:

$$argmin(\sum_{i=1}^{n} l_h(x_i, y_i, w) + \lambda ||w||^2)$$

Introduce a new **slack variable** ξ_i where it is used to upper bound the loss.

$$\xi_i \geq l_h = \max(0, 1 - y_i(w^T x + b))$$

The problem is now:

$$argmin(\frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i)$$

$$s.t. \ \ \xi_i \geq 1 - y_i(w^T x_i + b) \text{ and } \xi_i \geq 0$$

Note in this case that the constraint is really "**softened**".

10.2 Lagrangian Duality for SVM

Idea: transform **primal program** to a related **dual program**, alternate to primal.

$$L(x, \lambda, v) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{j=1}^{m} v_j h_j(x)$$

(old) primal program: $\min_x \max_{\lambda \geq 0, v} L(x, \lambda, v)$
(new) dual program: $\max_{\lambda \geq 0, v} \min_x L(x, \lambda, v)$
Weak Duality: dual optimum $\leq$ primal optimum.
For convex program (including SVM!), **Strong duality**: dual optimum = primal optimal (i.e. solving dual problem is equivalent of solving the primal problem.
Back to the lagrangian equation, solve the stationary condition:

$$\nabla_w, bL(w^*, b^*, \lambda^*) = 0$$

and plug back into the Lagrangian equation:

$$argmax_\lambda \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

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

Definition: Dual for soft-margin SVM

$$argmax_\lambda \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

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

(1) complexity of solutions is $O(n^3)$ instead of $O(d^3)$

(2) Program depends on dot products of data only (Not individual x's).

Complementary Slackness and Dot products

Complementary Slackness states that:

$$\lambda^*(y_i((w^*)^T x_i + b^*) - 1) = 0$$

Therefore, either optimal $\lambda^* = 0$ or $y_i((w^*)^T x_i + b^*) - 1 = 0$ or both.
Recall that for a particular x_i, if $y_i(w^T x_i + b) - 1 > 0$, then x_i lies outside the margin.
Hence:
(1) Points outside the margin must have $\lambda_i^* = 0$
(2) Points that are on the supporting vector (i.e. on the two supporting hyperplanes) must have $\lambda_i^* \neq 0$.
Important conclusion: preditions can be made by **dot products with the supporting vectors**, hence $O(n^3)$ is not necessarily a big number.

11 Kernel Methods

11.1 Kernelising the SVM

Idea: Feature transformation by basis expansion; sped up by direct evaluation of kernels - the 'kernel' trick.

When data is not linearly separable

Real data is unlikely to be linearly separable, therefore, hard-margin SVM is in trouble. In order to solve this problem, 3 approaches:

1. First transform the data, then apply hard-margin SVM (This lecture)

2. Relax the constraint (soft-margin)

3. Combo of 1 and 2.

Feature Space Transformation (Basis Expansion)

(1) Map data into a new feature space

(2) Run hard-margin or soft-margin SVM in new space

(3) Decision boundary is non-linear in original space

e.g. data is not linearly separable in 2-D, but linearly separable in 3-D so we add one feature.

A naive workflow:

1. Choose/design a linear model

2. Choose/design a high-dimensional transformation $\phi(x)$, here we hope that adding features could make the data linearly separable.

3. For each training example, and for each new instance compute $\phi(x)$.

4. Train classifier

PROBLEM: impractical/impossible to compute $\phi(x)$ for high/infinite-dimensional $\phi(x)$

NOTE: Feature Space Transformation/basis expansion $\neq$ kernel method.
Now we derive the kernel method, recall that SVM dual:
In the training process:

$$argmax_\lambda \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j x_i^T x_j$$

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

In the prediction:

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

Notice we only need dot products

Definition: Kernel

Kernel is a function that can be expressed as a dot product in some feature space

$$K(u, v) = \phi(u)^T \phi(v)$$

Essentially $K(u, v)$ implies a dot product in some feature space.

Usually, one would:
(1) apply feature transformation on x, get $\phi(x)$
(2) Kernelization: get $\phi(x)^T \phi(x)$.
However, some kernel functions may be calculated directly, i.e. apply $K(x_i, x_j)$ directly.

Definition: Kernel hard-margin SVM

Training:

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

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

In the prediction:

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

ANNs

- Elements of $u = \phi$ are transformed input x
- This ϕ has **weights learned from data**

SVMs

- Choice of kernel K determines features ϕ
- NO LEARNING of ϕ function.
- There is no computation of ϕ in order to support high dimension.

11.2 Modular Learning

Idea: Kernelisation beyond SVMs: Separating the "learning module" from feature space transformation.

Representer Theorem

For any training set $\{x_i, y_i\}_{i=1}^n$, any empirical risk function E, monotonic increasing function g, then any solution

$$f^* \in argmin_f E(x_1, y_1, f(x_1), ...x_n, y_n, f(x_n)) + g||f||)$$

has representation for some coefficients:

$$f^*(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i)$$

(1) tells us whether we should bother doing a dual problem.

11.3 Constructing Kernels

Method 1: Let $K_1(u, v)$, $K_2(u, v)$ be kernels. $c > 0$ be a constant, and $f(x)$ be a real-valued function. Then each of the following is also a kernel:

$$K(u, v) = K_1(u, v) + K_2(u, v)$$

$$K(u, v) = cK_1(u, v)$$

$$K(u, v) = f(u)K_1(u, v)f(v)$$

Method 2: Mercer's Theorem

Mercer's Theorem

Question: given $\phi(u)$, is there a good kernel to use?
Inverse question: given some function $K(u, v)$, is this a valid kernel? In other words, is there a mapping $\phi(u)$ implied by the kernel?
Mercer's theorem:
(1) Consider a finite sequences of objects $x_1, ...x_n$
(2) Construct $n \times n$ matrix of pairwise values $K(x_i, x_j)$
(3) $K(x_i, x_j)$ is a valid kernel if this matrix is **positive semi-definite**, and this holds for all possible sequences $x_1, ..., x_n$.

Advantage of SVM

- Flexibility (non-linearity) through kernel trick
- Reduces to a convex optimization problem (Compare to ANN, which is typically non-convex)
- Robustness through regularization/-max margin

Disadvantage of SVM

- Training scales poorly with data set size (worst case cubic $O(n^3)$)
- Uncalibrated class membership probabilities
- Lack of interpretability
- Effectiveness depends on choice of kernel/parameters
- Not directly applicable to multi-class tasks.

12 Ensemble Methods

Idea: combine models together

Ensemble Method

Model combination (aka. ensemble learning) constructs a set of base models (base learner) from given training set and aggregates the outputs into a single meta-model.
(1) Classification via (weighted) majority vote
(2) Regression via (weighted) averaging
(3) More generally:

$$\text{meta-model} = \text{f(base-model)}$$

Recall bias-variance trade-off:

$$E[Loss(y, \hat{f}(x_0)] = (E[y] - E[\hat{f}])^2 + Var[\hat{f}] + Var[y]$$

Test error = (bias) square + variance + irreducible error
Averaging k **independent and identically** distributed predictions reduces variance:

$$va[\hat{f}_{avg}] = \frac{1}{k}Var[\hat{f}]$$

12.1 Bagging

Bagging (Bootstrap Aggregating)

Method: Construct multiple (near-independent) datasets via random sampling with replacement (bootstrap)
(1) Generate k datasets, each size$n$ sampled from n training data with replacement - bootstrap samples
(2) Build base classifier on each constructed dataset.
(3) Aggregate predictions via voting/averaging.

Algorithm 7 Bagging Example: Random Forest

```
1: Number of trees: $k$, number of features (depth of each tree), $l \leq m$
2: Initialize the forest as empty
3: for c from 1 to k do
4:     Create new boostrap sample of training data
5:     Select random subset of $l$ of the $m$ features
6:     Train decision tree on boostrap sample using the $l$ features
7:     Add tree to forest
8: end for
9: Make predictions via majority vote (use out-of-sample)
```

Probability of not being selected:
Each round, each training sample has a probability of $(1 - \frac{1}{n})$ **NOT** being selected. Therefore, the probability of being left-out is:

$$(1 - \frac{1}{n})^n$$

$$\lim_{n \to \infty} (1 - \frac{1}{n})^n = e^{-1} = 36.8\%$$

On average only 63.2% of the data included per bootstrap sample.
(1) Random-forest has cross-validation for free, but on overlapping sub-samples.
(2) Evaluate each base classifier on its out-of-sample 36.8%.
(3) Average these evaluations: Evaluation of ensemble.

Bagging: reflections

(1) Simple method based on sampling and voting
(2) Highly effective over noisy datasets
(3) Performance is often significantly better than base classifier, never substantially worse
(4) Improves unstable classifier by reducing variance.

12.2 Boosting

Boosting

Difference with bagging: don't have equal weights when sampling, but still with replacements, re-sample with weights (Focus on the **hard / previously wrongly classified** instances.
Intuition: focus attention of base classifiers on examples that "hard to classify."
Method: iteratively change the **distribution** on examples to reflect performance of the classifier on the previous iteration.
Steps:
(1) Start with each training instance having a 1/n probability of being included in the sample.
(2) Over k iterations, train a classifier and update the weight of each instance according to classifier's ability to classify it.
(3) Combine the base classifiers via weighted voting.

Boosting Reflections

(1) Method based on iterative sampling and weighted voting
(2) Computationally expensive than bagging
(3) The method has guaranteed performance in the form of error bounds over the training data.
(4) Boosting can overfit.

Algorithm 8 Boosting Example: AdaBoost

```
1: Number of trees: $k$
2: Initialize example distribution $P_1(i) = \frac{1}{n}$
3: for c from 1 to k do
4:     Train base classifier $A_c$ on sample with replacement from $P_c$
5:     Set confidence $\alpha_c = \frac{1}{2} ln \frac{1-\varepsilon_c}{\varepsilon_c}$ for classifier's error rate $\varepsilon_c$.
6:     update example distribution (sample distribution) to be normalised of:
```

$$P_{c+1}(i) \propto P_c(i)x \begin{cases} \exp(-\alpha_c) & if \quad A_c(i) = y_i \\ \exp((\alpha_c) & if \quad A_c(i) \neq y_i \end{cases}$$

$$\begin{cases} \text{i.e. downweight the sampling probability if it is correctly classified} \\ \text{i.e. upweight the sampling probability if it is NOT correctly classified} \end{cases}$$

```
7: end for
8: Classify as majority vote weighted by confidences
```

$$argmax_y \sum_{c=1}^{k} \alpha_t \delta(A_c(x) = y)$$

where $\delta(A_c(x) = y)$ is an indicator function $= 1$ or -1, i.e. correctly identified or incorrectly identified.

Bagging

- Parallel sampling
- Minimise variance
- Simple voting
- Classification or regression
- Not prone to overfitting

Boosting

- Iterative sampling
- Target "hard" instances
- Weighted voting
- Classification or regression
- Prone to overfitting (unless learns are simple)

12.3 Stacking

Stacking

Intuition: "Smooth" errors over a range of algorithms with different biases

Method: train a meta-model over the outputs of the base learners

(1) Train base- and meta-learners using cross-validation

(2) Simple meta-classifier: logistic regression

Steps:

1. Split the training set into two disjoint sets.
2. Train several base learners on the first part.
3. Test the base learners on the second part.
4. Using the predictions from 3) as the inputs, and the correct responses as the outputs, train a higher level learner.

In stacking, the combining mechanism is that the output of the classifiers (Level 0 classifiers) will be used as training data for another classifier (Level 1 classifier) to approximate the same target function. Basically, you let the Level 1 classifier to figure out the combining mechanism.

It is a generalization of bagging and boosting.

Stacking: reflections

(1) Compare this to ANNs and basis expansion

(2) Mathematically simple but computationally expensive method

(3) Able to combine heterogeneous classifiers with varying performance

(4) With care, stacking results in as good or better results than the best of the base classifier.

13 Multi-Armed Bandits (Assignment 2)

Similar to RL, but MAB target a simpler set of problems

13.1 Stochastic MAB Setting

We have some actions $A = \{1, ..., k\}$, called **arms**
In round $t = 1, ..., T$:
1. Play action $i_t \in A$, could be random
2. Receive reward $X_{it}(t) \sim$ some reward distribution.

Goal is to minimize cumulative regret

$$\mu^* T - \sum_{t=1}^{T} E[x_{it}(t)]$$

$\Leftrightarrow$ Best expected cumulative reward with hindsight - Expected cumulative reward of bandit

where $\mu^* = \max_i \mu_i$

13.2 e-Greedy

greedy: the idea is to **estimate value** of each arm i as average reward observed. That is to say, you use average reward observed to estimate the action value Q, and choose the action with the max action value Q in a particular round.
ε: exploitation: choose action randomly, irrespective of the action value Q.

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} X_i(s) 1[i_s=1]}{\sum_{s=1}^{t-1} 1[i_s=i]}, & if \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, & otherwise \end{cases}$$

$$i_t \sim \begin{cases} argmax_i Q_{t-1}(i), & prob = 1 - \varepsilon \\ uniformly\ random\ action\ i, & prob = \varepsilon \end{cases}$$

13.2.1 Pure Greedy vs. e-greedy

Pure greedy only visits each action once, the performance of pure greedy increases fast.
ε-greedy does better long-term by exploring (because we force it to visit all actions enough times, and hence have a better expectation on action values).

13.2.2 Q initial values: Optimistic vs. Pessimism

Pessimism: init Q value below observable rewards $\rightarrow$ Only try one arm
Optimism: init Q value above observable rewards $\rightarrow$ Explore arms once
Middle-ground: init Q explore arms **at most** once.

13.3 Limitation of e-greedy

1. Sure we can improve on greedy algorithm with optimistic initialization and decreasing ε... However,
(1) They are too **distinct:** when we explore, we completely random, when we exploit, we blindly believe high Q value = better action.
(2) Exploitation is blind to **confidence** of estimates.

13.4 Upper Confidence Bound (UCB) Algorithm

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} X_i(s) 1[i_s=1]}{\sum_{s=1}^{t-1} 1[i_s=i]} + \sqrt{\frac{\rho \log(t)}{\sum_{s=1}^{t-1} 1[i_s=i]}}, & if \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, \quad otherwise \end{cases}$$

$$i_t \sim \begin{cases} argmax_i Q_{t-1}(i), \quad prob = 1 - \varepsilon \\ uniformly\ random\ action\ i, \quad prob = \varepsilon \end{cases}$$

where $\rho > 0$ is the **exploration hyper-parameter**
Note Q is calculated in the form of: expectation (average) + upper confidence bound. Here we become **optimism in the face of uncertainty**.

13.4.1 UCB vs. e-greedy

(1) UCB quickly overtakes the ε-greedy approaches
(2) Continues to outpace on per round rewards for some time
(3) More striking when viewed as mean cumulative rewards.
(4) UCB can "pause" in a bad arm for a while, but eventually find the best.]
(5) Theoretical **regret bounds**, optimal to multiplicative constant. Grows like $O(\log t)$ i.e. averaged regret goes to zero!

14 Gaussian Mixture Model, Expectation Maximization

14.1 Unsupervised Learning

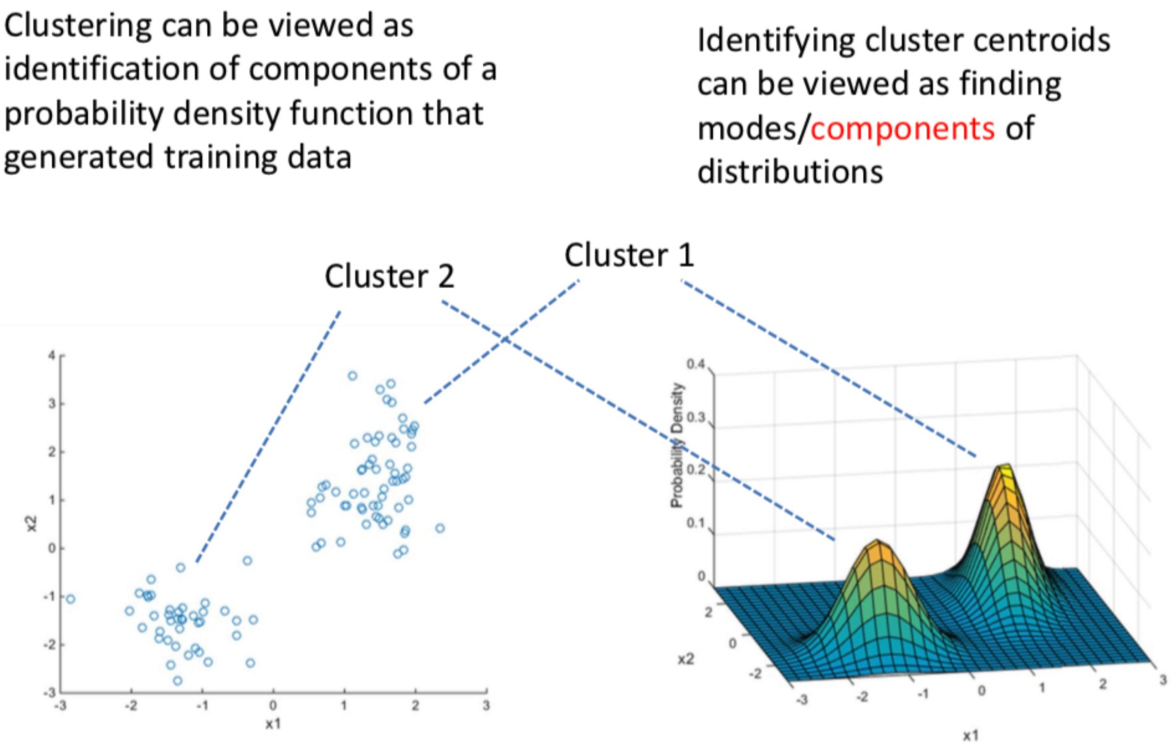
Idea: A large branch of ML that concerns with learning the structure of the data in the absence of labels. i.e. only x no y

Unsupervised Learning

Aim: is to explore the structure (patterns, regularities) of the data.
Example tasks:
(1) Clustering
(2) Dimensionality reduction
(3) Learning parameters of probabilistic models

14.2 Gaussian Mixture Model

IMPORTANT! GMM is a model while EM is an algorithm.
Just like k-means clustering, however, we model each cluster as a distribution.
(1) Choose number of clusters in advance (k)
(2) The probabilistic model gives us power to express **uncertainty about the origin of each point**. i.e., the model states that each point originates from cluster c with probability w_c, $c = 1, ...k$.
(3) That is, each point still originates from one particular cluster (aka component), but we are not sure from which one.



Clustering can be viewed as identification of components of a probability density function that generated training data

Identifying cluster centroids can be viewed as finding modes/components of distributions

Normal Distribution

1D Gaussian:

$$N(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

m-dimension Gaussian:

$$N(x|\mu,\Sigma) \equiv (2\pi)^{\frac{m}{2}}(\det\Sigma)^{-\frac{1}{2}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

where Σ is the $m\times m$ variance-covariance matrix, assumed to be positive definite.

Gaussian Mixture Density

Question to be answer: Given a cluster, how likely is the data (in this cluster)?

Definition: Gaussian Mixture Density

$$p(x) \equiv \sum_{c=1}^{k} w_c N(x|\mu_c,\Sigma_c)$$

Note:
(1) w_c is the component probabilities, indicating the probability of each data point x_i belongs to cluster c. $w_c \geq 0$ and $\sum_{c=1}^{k} w_c = 1$
(2) Components can be rare (small prob.) and common (high prob.)
(3) μ_c, Σ_c: each cluster has its own mean and variance-covariance matrix, each cluster centroid is at μ_c.
(4) $p(x)$ is a joint density of points, which indicates the probability of observing a data point.
(5) Here, "clustering" now amounts to finding parameters of the GMM that "best explain" the observed data.
(6) "MLE" optimize $p(x_1,...,x_n)$ with three parameters w_c,μ_c,Σ_c

Therefore, assuming that data points are **independent**, the optimization problem:

$$\max_{w_c,\mu_c,\Sigma}\prod_{i=1}^{n}\sum_{c=1}^{k} w_c N(x_i|\mu_c,\Sigma_c)$$

Analytical solution do not exist, log trick and derivative method (MLE) does not work, therefore, we need to use Expectation Maximization algorithm.

14.3 Expectation Maximization Algorithm

Idea: EM is a generic algorithm for finding MLE of parameters of a probabilistic model.

MLE is a frequentist principle!

EM is an algorithm, it is one way to find MLE

just like other methods, e.g. gradient descent

14.3.1 Motivation of EM

Let's say we have a parametric probabilistic model $p(X|\theta)$, where X denotes data and θ denotes a vector of parameters. We want:

$$\max p(X|\theta) \Leftrightarrow \max \log p(X|\theta)$$

Problems:
(1) Sometimes we don't observe some of the variables needed to compute the log likelihood. (E.g. GMM, we don't observe number of clusters c in advance)
(2) Sometimes log-likelihood is simply not convenient to work with.

14.3.2 Resolve the MLE Puzzle: Introduce latent variable

Denote latent variable Z as all unknown variables [4]

$$\begin{aligned}
&\log p(X|\theta) && \text{Maximize log likelihood} \\
&= \log \sum_Z p(X,Z|\theta) && \text{Marginalization, iterates over all possible values of Z} \\
&= \log \sum_z (p(Z)\frac{p(X,Z|\theta)}{p(Z)}) && \text{need Z to have non-zero marginal} \\
&= \log E_Z[\frac{p(X,Z|\theta)}{p(Z)}) && (1) \\
&\geq E_Z[\log \frac{p(X,Z|\theta)}{p(Z)}) && \text{Jensen's inequality} \\
&= E_Z[\log p(X,Z|\theta)] - E_Z[\log p(Z)] && (2)
\end{aligned}$$

Effectively we lower bounded the original log likelihood. For this lower bound, we want to push it as close to the original log likelihood as possible (i.e. intend to get the equality instead of the inequality).
The question has now become:

$$\max_{\theta, p(Z)} E_Z[\log p(X,Z|\theta)] - E_Z[\log p(Z)]$$

[4]Here we use discrete for demonstration purpose, if continuous, $\sum$ is replaced with $\int$

Idea of EM: effectively coordinate ascent, (1) fix θ and optimise the lower bound for $p(Z)$. (2) Fix $p(Z)$ and optimize for θ.

Algorithm 9 Expectation Maximization

```
1: Initialization: choose randomly initial values of $\theta^{(1)}$.
2: while true do
3:     E-step: compute
           $Q(\theta, \theta^{(t)}) \equiv E_{Z|x,\theta^{(t)}}[\log p(X, Z|\theta)]$
4:     M-step:
           $\theta^{(t+1)} = argmax_{\theta} Q(\theta, \theta^{(t)})$
5:     Termination: if either likelihood or expectation is stable (however, possibly local convergence)
6: end while
```

Definition: Tight Lower Bound

For any θ^*, setting $p(Z) = p(Z|X, \theta^*)$ maximizes the lower bound on $\log p(X|\theta^*)$ and makes it tight.

14.4 EM for GMM

Some terminologies: let $z_1, ..., z_n$ denote **true origins** of the corresponding points $x_1, ..., x_n$. Each z_i takes discrete variable that takes values in $1, ..k$, where k is a number of clusters. E.g., if x_5 belongs to cluster 3, then $z_5 = 3$. The original log likelihood:

$$\log p(x_1, ..., x_n) = \sum_{i=1}^{n} \log(\sum_{i=1}^{k} w_c N(x_i|\mu_c, \Sigma_c))$$

If we know z,

$$\log p(x_1, ..., x_n, z) = \sum_{i=1}^{n} \log(w_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i}))$$

However, we don't know z, recall from the previous section that EM can handle the unknown z by replacing $\log p(X, z|\theta)$ with expectation $E_{z|X,\theta^{(t)}}[\log p(X, z|\theta)]$ (including the tight lower bound).

Notice that in turn the expectation requires $p(z|X, \theta^{(t)})$ given the current parameter estimates.
So how do we calculate $p(z_i = c|x_i, \theta^{(t)})$? that is, how should we calculate the probability that a data point x_i originated from cluster c?

Definition: Cluster Responsibility

$$p(z_i = c|x_i, \theta^{(t)}) = \frac{w_c N(x_i|\mu_c, \Sigma_c)}{\sum_{l=1}^{k} w_l N(x_i|\mu_l, \Sigma_l)}$$

Bayes rule

Often denote $r_{ic} \equiv p(z_i = c|x_i, \theta^{(t)})$.

14.4.1 E-Step for GMM

$$\begin{aligned} Q(\theta, \theta^{(t)}) &\equiv E_{z|X,\theta^{(t)}}[\log p(X, z|\theta)] \\ &= \sum_{Z} p(Z|X, \theta^{(t)}) \log p(X, Z|\theta) \end{aligned}$$

Recall EM solution to unknown z

$$= \sum_{Z} p(Z|X, \theta^{(t)}) \sum_{i=1}^{n} \log w_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i})$$

Treat the first sum as a whole,

$$\sum_{i=1}^{n} \sum_{Z} p(Z|X, \theta^{(t)}) \log w_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i})$$

Recall Cluster responsibility,

$$\sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i})$$

Using log property,

$$\sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i} + \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i})$$

14.4.2 M-Step for GMM

Take partial derivatives of $Q(\theta, \theta^{(t)})$ w.r.t each of the parameters and set the derivatives to zero to obtain new parameter estimates:

$$w_c^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} r_{ic}$$

$$\mu_c^{(t+1)} = \frac{\sum_{i+1}^{n} r_{ic} x_i}{r_c}$$

where $r_c = \sum_{i=1}^{n} r_{ic}$

$$\Sigma_c^{(t+1)} = \frac{\sum_{i=1}^{n} r_{ik} x_i x_i^T}{r_k} - \mu_c^{(t)} (\mu_c^{(t)})^T$$

14.4.3 K-means as a EM model for restricted GMM model

In K-means,

$$w_c = \frac{1}{k}$$

$$\Sigma_c = \sigma^2 I$$

In this case, only centroids μ_c needs to estimated.
Besides, $r_{ic} = 1$ if centroid $\mu_c^{(t)}$ is closest to point x_i, otherwise $r_{ic} = 0$

EM for GMM considers the probabilities of a point belonging to different cluster by optimizing μ_i, Σ_i, w_c.
While K-means assign every point to one cluster only.

14.4.4 How to choose k

Question: why don't we optimize k? Because increase k will make the model more complicated, leading to overfitting.
Approaches:
(1) k is known from context
(2) Cross-validation, AIC/BIC, Kink method, Non-parametric models.

15 Dimensionality Reduction: PCA

Definition: Dimensionality Reduction

Refers to representing the data using a smaller number of variables (dimensions) while preserving the "important" structure of data.
Another interpretation: capturing the direction of variance in the data.
Why?
(1) Visualization
(2) Computational efficiency in a pipeline
(3) Data compression or statistical efficiency in a pipeline
However
Loss of information

15.1 Principle Component Analysis (PCA)

Idea: finding a rotation of data that minimises covariance between (new) variables

Definition: PCA

Given a dataset $x_1, ..., x_n \in R^m$, PCA aims to find a new **coordinate system** such that most of the variance is concentrated along the first coordinate, then most of the remaining variance along the second coordinate, etc.
Dimensionality reduction: discarding coordinates except first $l < m$[a]
The core of PCA is finding the new coordinate system, such that most of the variation is captured by "earlier" axes.

[a] m is the total dimension, which means that we choose only the first l important features.

So, we want to find a coordinate system where the projection of a data point onto the new coordinate system has the maximum variance (1st, 2nd, etc).

Introducing an arbitrarily new coordinate system $p_1, p_2, ..., p_m$ where each $||p_i|| = 1$ (unit vector, indicting the coordinate direction), therefore, p_is are direction of the new coordinates. Then the projection of all original data x_i onto each new axis p_i is:

$$X^T p_i$$

The variance along the first principal axis p_1 is:

$$\frac{1}{n-1}(X^T p_1)^T (x^T p_1)$$

$$= \frac{1}{n-1} p_1^T X X^T p_1 = p_1^T \Sigma_X p_1$$

Definition: PCA Optimization Problem

The question has now become an optimization problem:

$$\max_{p_1} \quad p_1^T \Sigma_X p_1$$

$$s.t. \quad ||p_1|| = 1$$

where Σ_X is the covariance matrix of the original data.

Using Lagrangian method:

$$L = p_1^T \Sigma_X p_1 - \lambda_1 (p_1^T p_1 - 1)$$

$$\frac{\partial L}{\partial p_1} = 2\Sigma_X p_1 - 2\lambda_1 p_1 = 0$$

$$\Rightarrow \Sigma_X p_1 = \lambda_1 p_1$$

p_1 is the eigenvector (First principle component) and λ_1 is the corresponding eigenvalue.

Definition: PCA Optimization Problem

So in summary, we choose p_1 as the eigenvector with largest eigenvalue, of centered covariance matrix Σ_X.
Lemma 1: a real $m \times m$ matrix has m real eigenvalues and corresponding eigenvectors are orthogonal.
Lemma 2: a positive semi-definite matrix further has non-negative eigenvalues.

PCA Steps

(1) Assume data points are arranged in columns of X. That means that the variables are in rows.
(2) Ensure that the data is centered: **subtract the mean row** (the data centroid) from each row.
(3) Find eigenvalues of centerd data covariance matrix $\Sigma_X = \frac{1}{n-1} X X^T$ i.e. spectrum [a]
(4) Sort the eigenvalues from the largest to smallest: each eigenvalue equals to variance of data along corresponding principle component (PC).
(5) Set $p_1, ..., p_m$ as the corresponding eigenvectors [b].
(6) Project data X onto these new axes to get coordinates of the transformed data.
(7) Keep only the first s coordinates to reduce dimensionality.

[a] A spectrum is a set of eigenvalues of a system
[b] $p_1, ..., p_m$ is an orthonormal basis

Additional effect of PCA: Original data is correlated. PCA should end up finding anew axes (transformation) such that **transformed data is uncorrelated.** (Note: Orthonormal axes)

15.2 Non-linear data and Kernel PCA

Idea: low-dimensional approximation need not be linear.

Definition: Kernel PCA

The idea is first mapping the data to feature space, then run PCA.

Note the solution strategy differs from regular PCA.

16 Bayesian Regression

Point estimates (frequentists) do not capture **uncertainty**. Bayesian approach preserves **uncertainty.**

Frequentists

- Only one optimal solution (point estimates)

Bayesian

- we have some expectation on the parameters (prior) and then update (posterior belief)

16.1 Problem with Frequentists: Uncertainty

Idea: If we have only a small training sets, we rarely have complete confidence in any models learned. Can we quantify the uncertainty, and use it making predictions?

Fisher Information Matrix[5]

The point is with small number of data, we are not really confident about the point estimates and inference (prediction).

16.2 Bayesian view on Uncertainty

Idea: Retain and model all unknowns (e.g. uncertainty over parameters)

Bayesian Inference

Instead of using only ONE unique parameter (frequentists), could we reason over **all** parameters that are consistent with the data?
(1) weights [a] with a better fit to the training data should be more probable than others.
(2) make predictions with all these weights, scaled by their probability.

Bayesian View

(1) Reason under **all** possible parameter values and weighted by their **posterior probability**
(2) Results in more **robust** predictions:

- Less sensitive to overfitting, particularly with small training sets
- Can give rise to more expressive model class (Bayesian logistic regression becomes non-linear.

[a] Parameters

[5] Not examinable

Frequentists

- Learning using **point estimates**, regularizations, p-values.
- backed by complex theory relying on strong assumptions
- mostly simpler algorithms, characterises much practical machine learning research

Bayesian

- maintain **uncertainty**, marginalize (sum away) out unknowns during inference.
- nicer theory with fewer assumptions
- often more complex algorithms, but not always
- when possible, results in more elegant models

16.3 Bayesian Regression

Recall probabilistic formulation of linear regression:

$$y \sim Normal(x^T w, \sigma^2)$$

$$w \sim Normal(0, \gamma^2 I_D)$$

Bayes rule:

$$p(w|X, y) = \frac{p(y|X, w)p(w)}{p(y|X)}$$

$p(y|W)$ is **evidence**.

$$\max_w p(w|X, y) = \max_w p(y|X, w)p(w)$$

Let's consider the **full posterior**

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

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

Now the question is can we compute the denominator (marginal likelihood)? If we can, then we can use the full posterior instead of its mode.

Definition: Conjugate Prior

Conjugate prior is when the product of likelihood $\times$ prior results in the same distribution as the prior.

In this case, since we have the conjugate prior (normal $\times$ normal = normal), **evidence** can be computed easily using the normalising constant of the Normal distribution, i.e., one does not have to compute the integral to get the denominator, instead, we can approximate.

$$p(w|X, y, \sigma^2) \propto Normal(w|0, \gamma^2 I_D) Normal(y|Xw, \sigma^2 I_N)$$
$$\Rightarrow p(w|X, y, \sigma^2) \propto Normal(w|w_N, V_N)$$

where

$$w_N = \frac{1}{\sigma^2} V_N X^T y \quad [6]$$
$$V_N = \sigma^2 (X^T X + \frac{\sigma^2}{\gamma^2} I_D)^{-1}$$

Algorithm 10 Sequential Bayesian Update

```
1: start from prior $p(w)$
2: see new labelled data point
3: Compute posterior $p(w|X, y, \sigma^2)$
4: The **posterior now takes role of prior** and repeat from step 2
```

State of training

1. Decide on model formulation and prior
2. Compute posterior over parameters, $p(w|x, y)$

MAP

3. Find *mode* for **w**
4. Use to make prediction on test

approx. Bayes

3. Sample many **w**
4. Use to make *ensemble* average prediction on test

exact Bayes

3. Use **all w** to make prediction on test.

16.4 Bayesian Prediction

Definition: Monte Carlo Integration

(1) Sample s (weight) parameters, $w^{(s)}$
(2) Use those parameters to predict $y^{(s)}$
(3) Compute mean and variance over these predictions

[6] Mean (and mode) are the MAP solution from before

For Bayesian and Gaussian Distribution, there is a simpler way:

$$p(\hat{y_*}|X, y, x_*, \sigma^2) = \int p(w|X, y, \sigma^2)p(y_*|x_*, w, \sigma^2)dw$$

$$p(y_*|x_*, X, y, \sigma^2) = \int p(w|X, y, \sigma^2)p(y_8|x_*, w, \sigma^2)dw$$

$$= \int Normal(w|w_N, V_N)Normal(y_*|x_*w, \sigma^2)dw$$

$$= Normal(y_*|x_*^T w_N, \sigma_N^2(x_*))$$

$$\sigma_N^2(x_*) = \sigma^2 + x_*^T V_N x_*$$

Note that for MLE/MAP estimate, variance is a fixed constant.

17 Bayesian Classification

$* * *$So, Discrete Data$* **$

17.1 Beta-Binomial Conjugacy

Setting: n toss, k heads, p is the probability of heads
Likelihood:

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

Prior:

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

Bayesian Posterior:

$$p(q|k,n) \propto p(k|n,q)p(q)$$
$$\propto Beta(q;k_\alpha, n-k+\beta)$$

Conjugate Prior

d_1, d_2 are different distributions. Suppose prior $\sim d_1$ and likelihood $\sim d_2$, if bayesian posterior $\sim d_1$, then we call d_1 is a conjugate prior of d_2.
Why?
(1) Note that this is an unrealistic assumption (in most cases)
(2) Bayesian update becomes much easier since we can use **analytical solution**: easy formula (no sample necessary, no integral necessary).
(3) Generate posterior exactly.

	Likelihood	**Conjugate Prior**
Regression	Normal	Normal (for mean)
	Normal	Inverse Gamma (for variance) or Inverse Wishart (covariance)
Classfication	Binomial	Beta
	Multinomial	Dirichlet
Counts	Possion	Gamma

17.2 Bayesian Logistic Regression

The model is ***discriminative***, with parameters defined using logistic sigmoid (or softmax for multi-class).

$$p(y|q,x) = q^y(1-q)^{1-y}$$
$$q = \sigma(x^T w)$$

Different from the previous section, here we need prior for w, not q. We need choose a prior of w so that we get a nice posterior over w in the same form.
However, no known conjugate prior exist. Thus we need a Gaussian prior.

18 Probabilistic Graphical Models (PGM)

Idea: Marriage of graph theory and probability theory. Tool of choice for Bayesian statistical learning.

Bayesian Statistical Learning

- Model joint distribution of X, Y and random variables
- Priors: marginals on parameters
- Training: update prior to posterior using observed data
- Prediction: output posterior, or some function of it (MAP)

PGM aka Bayes Nets

- Efficient joint representation
- Independence made explicit
- Trade-off between expressiveness and need for data, easy to make
- Easy for practitioners to model
- Algorithms to fit parameters, compute marginals, posteriors

18.1 Discrete Joint Distribution Tables

Number of rows grows exponentially with number of random variables. M boolean r.v.'s require $2^M - 1$ rows.

Good thing for a joint distribution table is:

1. **Probabilistic inference:** we can compute any other distribution involving our r.v.'s
2. **Patterns:** we want a distribution, we have a joint, we will use:

Bayes rule + marginalization

Bad thing: Computational complexity (size of the table grows exponentially)
Ugly thing: Model complexity (the model is too flexible and contains way too many parameters to fit, lots of data required and easy over-fitting).

Solution? We assume independence

Independence: reduce the number of parameters to fit, and more importantly, allow us to **factor the joint distributions**.

Different types of independence

Unconditional independence:

$$Pr(x_1|x_2) = Pr(x_1)$$

Conditional independence:

$$Pr(x_1|x_2, x_3) = Pr(x_1|x_2)$$

Why? We can **speed up inference** and **avoid over-fitting**

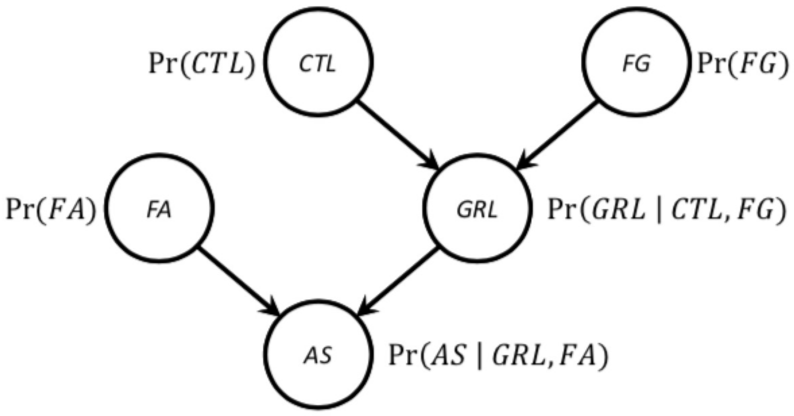
18.2 Directed PGMs

Nodes $\Leftrightarrow$ Random Variables

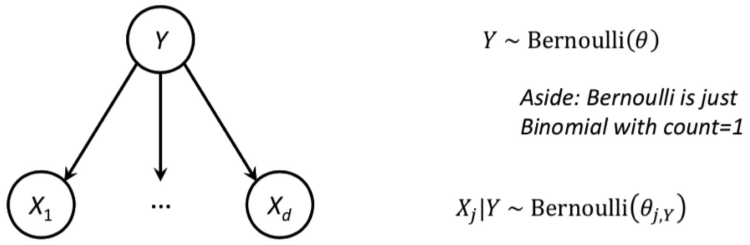
Edges (acyclic) $\Leftrightarrow$ conditional dependence $Pr(children|parents)$ Joint Factorization:

$$Pr(x_1, x_2, ...x_k) = \prod_{i=1}^{k} Pr(x_i|x_j \in parents(x_i))$$

18.2.1 Example



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



$$Pr(y, x_1, ...x_d)$$

$$= Pr(x_1|y)Pr(x_2|x_1, Y)...Pr(x_d|x_1, ..., x_d, y)Pr(y)$$

Assuming conditional independence,

$$= Pr(x_1|y)Pr(x_2|y)...Pr(x_d|y)Pr(y)$$

Prediction: predict label maximising $Pr(y|x_1, ..., x_d)$

18.2.2 PGM Bayesian or Frequentist?

If nodes have priors, then **Bayesian**, if nodes do not have priors, then **Frequentist**

18.3 Undirected PGMs: Markov Random Field

Idea: Undirected variant of PGM, parameterised by arbitrary positive valued functions of the variables, and global normalization.

Undirected PGM

- Graph: edges are undirected
- Probability:
  Each node is a r.v.
  Each clique C has "factor" $\psi_c(X_j : j \in C) \geq 0$
  Joint $\propto$ product of all factors.

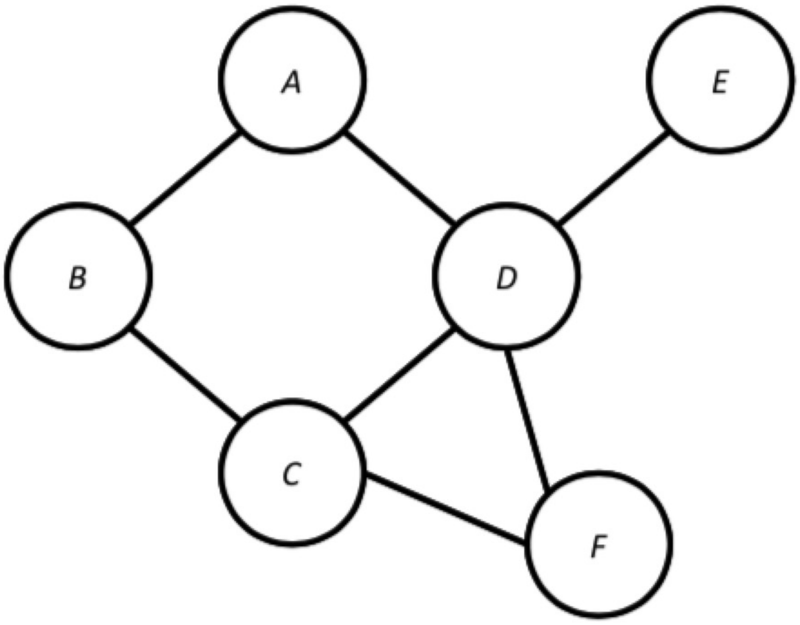
Directed PGM

- Graph: edges are directed
- Probability:
  Each node is a r.v.
  Each node has conditional probability $p(x_i|x_i \in parents(x_i))$
  Joint $=$ product of conditional probability

Notice that both cases are all about factoring the joint distribution
However, the key difference is **normalization**

Definition: Clique

It is a set of fully connected nodes.
Maximal Clique : the largest clique in the graph.



Definition: Joint probability in undirected PGM

$$P(a,,b,c,d,e,f) = \frac{1}{Z}\psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(d,c,f)\psi_5(d,e)$$

where ψ is a positive function and Z is the normalizing 'partition' function.

$$Z = \sum_{a,b,c,d,e,f} \psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(d,c,f)\psi_5(d,e)$$

In general, ψ is an exponential function.

$* * * *$ Why Undirected PGM? $* * * *$

Pro

- Generalization of Directed-PGM
- Simpler means of modelling without the need for per-factor normalization
- General inference algorithms use U-PGM representation.

Cons

- Slightly weaker independence
- Calculating global normalization term (Z) intractable in general.

18.4 Example PGMs

18.4.1 Hidden Markov Model (HMM) AND Kalman Filter

Fundamental HMM Tasks:

HMM Task	**PGM Task**
Evaluation: Given an HMM μ and observation sequence O, determine likelihood $Pr(O\|\mu)$	Probabilistic Inference
Decoding: Given an HMM μ and observation sequence O, determine The most probable hidden state sequence O	MAP Point Estimate
Learning: Given an observation sequence O and set of states, learn parameters	Statistical Inference

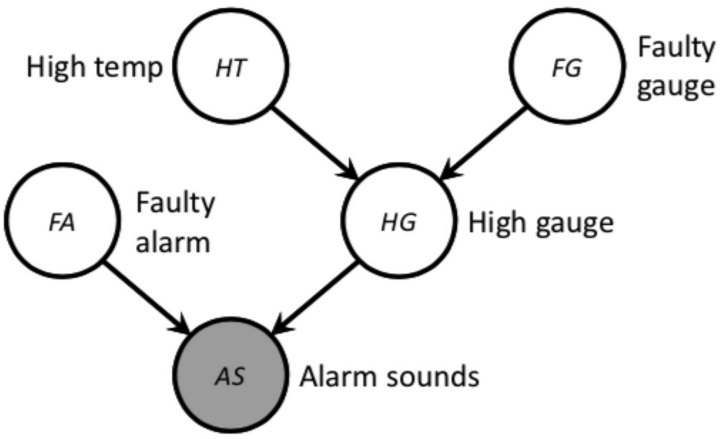
18.4.2 Conditional Random Fields (CRF)

(1) Same model applied to sequences.

(2) CRFs are discriminative, model $p(Q|O)$, vs. HMM is generative, $p(Q,O)$.

19 PGM Inference

Idea: we want to efficiently compute marginal and conditional distributions from the joint of a PGM using Bayes rule and marginalization.



Note that AS is the node that we observe. The aim is to find:

$$p(HT|AS=t)$$

Bayes rule:

$$= \frac{p(HT, AS=t)}{p(AS=t)}$$

$$= \frac{\sum_{FG,HG,FA} p(AS=t, FA, HG, FG, HT)}{\sum_{FG,HG,FA,HT'} p(AS=t, FA, HR, FG, HT')}$$

For numerator (denominator similar), (1) expanding the sums and (2) factor the joint probabilities.

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

distributing the sums as far as possible.

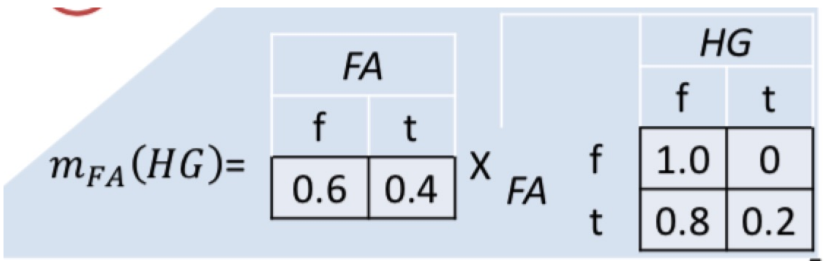
$$= p(HT)\sum_{FG} p(FG)\sum_{HG} p(HG|HT,FG)\sum_{FA} p(FA)p(AS=t|FA,HG)$$

We use **elimination algorithm** to solve this, since $p(AS=t|FA,HG)$ is observed

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

Note that the following operation is actually a matrix multiplication.

$$\sum_{FA} p(FA)m_{AS}(FA,HG)$$



$$m_{FA}(HG)= \begin{array}{|c|c|} \multicolumn{2}{c}{FA} \\ f & t \\ \hline 0.6 & 0.4 \\ \hline \end{array} \times \begin{array}{cc|c|c|} & & \multicolumn{2}{c}{HG} \\ & & f & t \\ \hline FA & f & 1.0 & 0 \\ \hline & t & 0.8 & 0.2 \\ \hline \end{array}$$

Then continue:

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

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

$$= p(HT) m_{FG}(HT)$$

20 MLE, MAP, Bayesian Regression and Bayesian Inference

20.1 Bayes Rule

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

20.2 Maximum Likelihood Estimates

$$\max_w p(y|w,x,\sigma^2) \Leftrightarrow \max_w \prod p(y_i|x_i) \Leftrightarrow \max \sum \log p(y_1|x_i)$$

Effectively once we propose a model, we want to find the parameters that best explain the data.

20.3 Maximum a Posterior

$$\max p(w|y,x,\sigma^2) \Leftrightarrow \max p(y|w,x,\sigma^2)p(w)$$

Here note that the denominator of Bayes equation $p(y|x,\sigma^2)$ does not affect the optimization equation, so we get rid of the constant, MAP effectively finds the the mode of the distribution (where the top point of the posterior distribution locates).
Both MLE and MAP are point estimates (only one optimal parameter vector). Variance of the predicted $\hat{y}$ is fixed.

Note that MAP is technically non-Bayesian Regression method.
MAP is regularized MLE (because we add prior information)
MLE is MAP with uniform prior $p(w) = 1$.

20.4 Bayesian Regression

Consider the full posterior, not only the mode of the posterior distribution.

$$p(w|y,x,\sigma^2) = \frac{p(y|w,x,\sigma^2)p(w)}{\int p(y|w,x,\sigma^2)p(w)dw}$$

Now how to calculate the denominator?
1. Conjugate prior, so we dont have to calculate the integral.
2. EM method to approximate

Sequential Bayesian Update

1. Start from the prior $p(\theta)$
2. Observe new labelled data.
3. Update full posterior.
4. Posterior becomes prior, and then iterate over.

20.5 Bayesian Inference (Prediction)

(1) Sample s parameters.

(2) Use those parameters to predict $\hat{y}$.

(3) Compute mean and variance of those predictions