Matrix differentiation rules

$$f(x) = a^T x$$
$$\frac{\partial f}{\partial x} = a$$
$$f(x) = x^T A x$$
$$\frac{\partial f}{\partial x} = (A + A^T)x$$

Hessian and proving convexity

The Hessian for a given loss function $L(w)$ is given by

$$H_{kl} = \frac{\partial L(w)}{\partial w_k \, \partial w_l}$$

We show that the loss function is **convex** if we show that the Hessian is positive semi-definite, that is

$$u^T H u \geq 0$$

Univariate and multivariate normal distribution density

For $X \sim N(\mu, \sigma^2)$, we have

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

For $X \sim N(\mu, \Sigma)$, we have

$$f_X(x) = \frac{1}{|\Sigma|^{1/2}(2\pi)^{n/2}} exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Kernel functions and the kernel trick

Kernel functions are functions which for some higher dimensional mapping $v: R^n \to R^{n+k}$ where $n, k > 0$, which provide the dot product in R^{n+k} with purely the elements of R^n.

Higher dimensions mean higher computational complexity – and thus kernel functions are useful to reduce this computational complexity and still find linear separability.

Perceptrons in primal and dual form

Perceptrons are linear classifiers. In the primal form, we train a weight vector w such that

$$\hat{y}(x) = \text{sign}(w^T x)$$

We update the weight vector w on *misclassified* instances, such that

$$w \leftarrow w + \alpha y_i x_i$$

For learning rate α. In the **dual** form, each instance has a value α_i that counts the total misclassifications of the instance. Then

$$w = \sum_{i=1}^{n} \alpha_i \, y_i x_i$$

Dual form has two primary advantages

1. Kernel functions are easy to use
2. No weight vector is explicitly needed

Local regression

Local regression uses the groups formed by k-nearest neighbours to form linear function in each group.

Entropy and information gain

The entropy for a set S of k classes, each with sample probability $p_i : i \in \{1, \ldots, k\}$, is given by

$$H(S) = \sum_{i=1}^{k} -\mathrm{p_i} \log_2 \mathrm{p_i}$$

Information gain, from partitioning a set S with attribute A, is given by

$$Gain(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Where S_v is the set of values in S with the attribute $v \in A$. *We estimate class probabilities as the sample probability.*

Pruning

(Deep) trees are prone to overfitting, and so we can post-prune. Consider f, the empirical error, e, the pessimistic error and c, the confidence. Then our pessimistic error is given by

$$e = f + c\sqrt{\frac{f(1-f)}{N}}$$
$$= Empirical\, error \; + \; std.of\; error$$

If $e_{child} \geq e_{parent}$, then the child node is adding more error and thus should be pruned.

Support vector machines

Support vector machines are linear classifiers which maximise the *margin* between classes. The margin is given by the smallest distance between the two classes; found by *support vectors.*

The angle between the linear classifier and some support vector is given by

$$w_i \cdot x_i = ||w|| \cdot ||x|| \cos(\theta)$$

The margin is formed by a subset of points. The maximal margin classifier maximises the margin.

$$w \cdot x - t = 0$$
$$w^*, t^* = argmin_{w,t} \frac{1}{2} ||w||^2$$

Subject to $y_i(w \cdot x_i - t) \geq 1$.

To allow misclassifications, *slack variables* can be introduced for the **soft margin classifier**.

$$w \cdot x_i - t \geq 1 - E_i$$
$$w^*, t^*, E_i^* = \text{argmin}_{w,t,E_i} \frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} E_i$$

SVMs are good for problems where the number of features is much larger than the number of samples.

Bias-variance decomposition

It is true that for a given estimator θ, that

$$MSE(\theta) = Bias^2(\theta) + Var(\theta)$$

To be *high variance* is to have high variability in the hypotheses that a model produces, and vice versa.

To be *high bias* is to have large deviation from the true value in the hypotheses that a model produces, and vice versa.

Performance metrics/evaluation measures

P(pred | pos)	=	$\frac{TP}{TP+FN}$	(Sensitivity, Recall)
P(pred | neg)	=	$\frac{FP}{FP+TN}$	(FP rate)
P(not_pred | pos)	=	$\frac{FN}{TP+FN}$	(FN rate)
P(not_pred | neg)	=	$\frac{TN}{FP+TN}$	(Specificity)
P(pos | pred)	=	$\frac{TP}{TP+FP}$	(Pos. Pred. Value, Precision)
P(neg | pred)	=	$\frac{FP}{TP+FP}$	
P(pos | not_pred)	=	$\frac{FN}{FN+TN}$	(FN rate)
P(neg | not_pred)	=	$\frac{TN}{FN+TN}$	(Neg. Pred. Value)

Bagging trees

Bagging trees train trees for bootstrapped training sets and then creates hypotheses using voting/averaging.

Bootstrapping is simply sampling with replacement. Bootstrapping reduces variance proportional to the correlation between the models.

Random forests

Random forests choose a random set of $p < n$ parameters and then combine the trained models. The motivation is that due to the **greedy** nature of tree training, that moderately good predictors are overshadowed by strong predictors.

Randomly choose p predictors allow these moderately good predictors to have more influence on the final hypothesis.

Boosting

Boosting uses voting/averaging, but models are *weighted according to their performance.* Boosting generally uses shallow 'base learners', which train on the **errors/residuals** of the previous model.

Neural networks

Neural networks are defined by layers of non-linear 'activation functions' which take input from the previous layer. Neural networks are strong for:

- Highly dimensional input data
- Noisy data

ReLu versus Sigmoid

Sigmoid functions can be used to ensure that values from layers are between 0 and 1 – but they are often saturated (either become very close to 0 *or* 1).
ReLu is defined by

$$\text{ReLu}: f(x) = max(0, x)$$

Universal approximator theorem and how many layers
One hidden layer is usually enough to *represent* an approximation of **any function**. As more layers are added however, test accuracy generally increases (but so does computational time).
Hierarchical clustering
Bottom up: at each step, join the two closest clusters starting from single-instance clusters
Top-down: Find two clusters and then proceed recursively for the two subsets
Dendogram
Dendograms are charts that are used to represent hierarchical clustering. Average-linkage distance computation is used to represent how similar clusters are from eachother.

The distance between a a new cluster w formed by combining two clusters u and v and all other clusters x is given by

$$D_{w,x} = \frac{m_u D_{u,x} + m_v D_{v,x}}{m_u + m_v}$$

Where m_u is the number of objects in cluster u, and $D_{w,x}$ is the *weighted mean* of the distance between the cluster w and other clusters x.

The distance between the new edge (v, w) and (u, w) is

$$L_{v,w} = \frac{1}{2} D_{u,v} - L_{v,y_v}$$

Where y_v is a leaf of the sub-tree with root v.
DBSCAN and the need for n clusters
Unlike k-means, DBSCAN does not require the number of clusters, but rather ϵ, the minimum distance between two *neighbour* points, and $MinPts$, the number of points required to form a 'dense' region.

It then creates three types of points:

1. A core point has more than MinPts of neighbours within ϵ
2. A border point has fewer neighbours within MinPts within ϵ but is within ϵ of a **core point.**
3. Everything else is a noise point

To create the clusters

1. Make a cluster for each core point by connecting them if they are within ϵ
2. Assign each border point to the cluster of their corresponding core point

How many clusters?
For clustering methods that require the number of clusters n as an input; how do we decide the optimal number of clusters?

Dispersion is a metric which considers the distance between points within a cluster (usually squared distances to centroids).

1. Elbow method
    a. Compute dispersion for growing n
    b. Find the 'elbow' – where the rate of decrease for dispersion slows down
2. Gap statistic
    a. Compare dispersion for clustering of n clusters between the training data and some *random reference dataset*
    b. The "gap" is given be

$$Gap(k) = E\left[log\left(W_k^{ref}\right)\right] - log(W_k^{data})$$

We wish to choose the smallest k such that $Gap(k) \geq Gap(k+1) - s_{k+1}$ where s_{k+1} is the standard deviation term for stability.
Principal components analysis
Given a standard feature matrix X and it's covariance matrix $\Sigma = \frac{1}{n} X^T X$, we find the eigenvectors v_i such that

$$\Sigma v_i = \lambda_i v_i$$

Where λ_i is the amount of variance captured in that direction.
Concept functions and target function
A concept function is a function that labels data as **positive** or **negative**.

A target function is the specific concept you're trying to learn from the hypothesis space H.
Hypothesis space
A hypothesis h is a specific function that maps inputs to outputs – we generally refer to predictors and trained models/learners as hypotheses.

A hypothesis space H is the set of all possible hypotheses that a learning algorithm can consider.
Empirical risk minimisation (ERM)
Learner L should output a hypothesis $h \in H$ as an estimate of a target concept c such that

$$h = \text{argmin}_{h \in H} \text{error}_{train}(h)$$

No free lunch theorem
Under the assumption that the training set D can be learned correctly by all algorithms; *averaged over all target functions*, no learning algorithms given an off-training set error superior to any other.

$$\sum_F [E_1(E|F, D) - E_2(E|F, D)] = 0$$

Where F is the set of all target functions. This implies:

1. No universal best algorithm
2. Empirical validation is required
3. Algorithms must be customised and tuned

Conservation theorem of generalisation performance
If on some problems, a learner L performs well, it must mean that on other problems, it performs poorly with the exact same magnitude **according to** the no free lunch theorem.
Version space
A hypothesis h is consistent with a set of training examples D of some target concept c if all $h(x) = c(x)$.

The version space $VS_{H,D}$ w.r.t the hypothesis space H and training examples D is the subset of hypotheses from H **consistent with all training examples.**

The sample complexity is given by

$$m \geq \frac{1}{\epsilon}(ln|H| + ln(1/\delta))$$

For accepted error rate ϵ, hypothesis space H and passing probability $1 - \delta$.
PAC (Probably Approximately Correct)
A class C of possible target concepts, defined over set X of length n, with learner L and hypothesis space H is PAC-learnable if for all $c \in C$, distributions D over X we have $0 < \epsilon < ½$ and $0 < \delta < ½$.
Vapnik-Chervonenkis (VC) Dimension
A dichotomy of a set S is a partition of S into two disjoit subsets.
A set of instances S is **shattered** by a hypothesis space H if and only if for every dichotomy of S there exists a hypothesis H consistent with the dichotomy.

1. Understand the hypothesis space H
2. Try to find a set it can shatter, and incrementally build the amount of points
3. Try to find a set of n points it can **not** shatter; the VC dimension is then $n - 1$.