A summary of machine learning algorithms [1]

	K-nearest neighbour	**Naive Bayes**	**Logistic regression**
Description	The label of a new point $\hat{x}$ is classified with the most frequent label $\hat{t}$ of the k nearest training instances.	Learn $p(\mathcal{C}_k\|x)$ by modelling $p(x\|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$, using Bayes' rule to infer the class conditional probability. Assumes each feature $\boldsymbol{x}$ independent of all others, thus 'Naive.' Suitable for data of high dimension.	Model the posterior probabilities of K classes via linear functions in x, while ensuring they fall within $[0, 1]$.
Model	$$\hat{t} = \arg\max_{\mathcal{C}} \sum_{i:x_i \in N_k(\boldsymbol{x},\hat{x})} \delta(t_i, \mathcal{C})$$ • $N_k(\boldsymbol{x}, \hat{x}) \leftarrow k$ points in $\boldsymbol{x}$ closest to $\hat{x}$ • Euclidean distance formula: $\sqrt{\sum_{i=1}^{D}(x_i - \hat{x}_i)^2}$ • $\delta(a, b) \leftarrow 1$ if $a = b$; 0 o/w	$$\begin{aligned} y(\boldsymbol{x}) &= \arg\max_k \ p(\mathcal{C}_k\|x) \\ &= \arg\max_k \ p(x\|\mathcal{C}_k) \times p(\mathcal{C}_k) \\ &= \arg\max_k \ \prod_{i=1}^{D} p(x_i\|\mathcal{C}_k) \times p(\mathcal{C}_k) \\ &= \arg\max_k \ \sum_{i=1}^{D} \log p(x_i\|\mathcal{C}_k) + \log p(\mathcal{C}_k) \end{aligned}$$	$$\log \frac{P(G=1\|X=x)}{P(G=K\|X=x)} = \beta_{10} + \beta_1^T x$$ $$\log \frac{P(G=2\|X=x)}{P(G=K\|X=x)} = \beta_{20} + \beta_2^T x$$ $$\vdots$$ $$\log \frac{P(G=K-1\|X=x)}{P(G=K\|X=x)} = \beta_{(K-1)0} + \beta_2^T x$$
Objective	No optimization needed.	No optimisation needed.	$$\updownarrow(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta),$$ where $$p_k(x_i; \theta) = P(G = k\|X = x_i; \theta).$$ In two-class cases, the log-likelihood can be written as $$\begin{aligned} l(\beta) &= \sum_{i=1}^{N} \{y_i \log p(x_i; \beta) + (1 - y_i) \log(1 - p(x_i; \beta))\} \\ &= \sum_{i=1}^{N} \{y_i \beta^T x_i - \log(1 + e^{\beta^T x_i})\}. \end{aligned}$$
Training	Use cross-validation to learn the appropriate k; otherwise no training, classification based on existing points.	**Multivariate likelihood** $p(x\|\mathcal{C}_k) = \sum_{i=1}^{D} \log p(x_i\|\mathcal{C}_k)$ $$p_{\text{MLE}}(x_i = v\|\mathcal{C}_k) = \frac{\sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k \wedge x_{ji} = v)}{\sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k)}$$ **Multinomial likelihood** $p(x\|\mathcal{C}_k) = \prod_{i=1}^{D} p(\text{word}_i\|\mathcal{C}_k)^{x_i}$ $$p_{\text{MLE}}(\text{word}_i = v\|\mathcal{C}_k) = \frac{\sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k) \times x_{ji}}{\sum_{j=1}^{N} \sum_{d=1}^{D} \delta(t_j = \mathcal{C}_k) \times x_{di}},$$ where • x_{ji} is the count of word i in test example j; • x_{di} is the count of feature d in test example j. **Gaussian likelihood** $p(x\|\mathcal{C}_k) = \prod_{i=1}^{D} \mathcal{N}(v; \mu_{ik}, \sigma_{ik})$	**MLE**: To maximize the log-likelihood, we set its derivatives to zero. $$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i; \beta)) = 0.$$ **Gradient descent** is written as: $$\theta_{k+1} = \theta_k - \eta_k g_k,$$ where η is the learning rate. The selection of η and other training methods including Newton's method should refer to [**?**].

[1] Initiated by Emanuel Ferm, and enriched by Guanqun Cao (guanqun.cao@tut.fi) on July 23, 2013.

	K-nearest neighbour	Naive Bayes	Logistic regression
Regularization	k acts as to regularise the classifier: as $k \to N$ the boundary becomes smoother.	Use a Dirichlet prior on the parameters to obtain a MAP estimate. **Multivariate likelihood** $p_{\text{MAP}}(x_i = v\|\mathcal{C}_k) = \frac{(\beta_i - 1) + \sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k \wedge x_{ji} = v)}{\|x_i\|(\beta_i - 1) + \sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k)}$ **Multinomial likelihood** $p_{\text{MAP}}(\text{word}_i = v\|\mathcal{C}_k) = \frac{(\alpha_i - 1) + \sum_{j=1}^{N} \delta(t_j = \mathcal{C}_k) \times x_{ji}}{\sum_{j=1}^{N}\sum_{d=1}^{D}(\delta(t_j = \mathcal{C}_k) \times x_{di}) - D + \sum_{d=1}^{D} \alpha_d}$	The L_1 penalty used in the lasso for variable selection and shrinkage applies for any linear regression model. Here we have $\arg\max \{\sum_{i=1}^{N}[y_i(\beta_0+\beta^T x_i) - \log(1+e^{\beta_0+\beta^T x_i} - \lambda \sum_{j=1}^{p} \|\beta_j\|)]\}.$
Complexity	$\mathcal{O}(NM)$ space complexity, since all training instances and all their features need to be kept in memory.	$\mathcal{O}(NM)$, each training instance must be visited and each of its features counted.	$\mathcal{O}(INMK)$, since each training instance must be visited and each combination of class and features must be calculated for the appropriate feature mapping.
Non-Linear	Natively finds non-linear boundaries.	Can only learn linear boundaries for multivariate/multinomial attributes. With Gaussian attributes, quadratic boundaries can be learned with uni-modal distributions.	**Kernel smoothing methods** can estimate the regression function $f(x)$ over the domain R^p by fitting a different but simple model separately at each query point x_0, (not to be confused with kernel methods). This localization is achieved via a weighting function or kernel $K_\lambda(x_0, x_i)$, which assigns a weight to x_i based on its distance from x_0. The kernels K_λ are typically indexed by a parameter λ that dictates the width of the neighborhood. Locally weighted regression solves a separate weighted least squares problem at each target point x_0: $\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \beta(x_0)x_i]^2.$
Online learning	To be added.	To be added.	The objective in online learning is the *regret*, which is the averaged loss incurred relative to the best using a single fixed parameter value: $regret_k = \frac{1}{k}\sum_{t=1}^{k} f(\theta_t, z_t) - \min_{\theta^* \in \Theta} \frac{1}{k}\sum_{t=1}^{k} f(\theta_*, z_t),$ where $f(\theta, z_t)$ is the loss function of certain kind. *Online gradient descent* describes that at each step k, update the parameters using $\theta_{k+1} = proj_\Theta(\theta_k - \eta_k g_k),$ where $proj_V(v) = \arg\min_{w \in V} \|w - v\|_2$ is the projection of vector v onto space $\mathcal{V}$, $g_k = \bigtriangledown f(\theta_k, z_k)$ is the gradient, and η_k is the step size.

Perceptron

Description Directly estimate the linear function $y(x)$ by iteratively updating the weight vector when incorrectly classifying a training instance.

Model Binary, linear classifier:

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

where:

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Multiclass perceptron:

$$y(x) = \arg\max_{\mathcal{C}_k} \ \boldsymbol{w}^T \phi(x, \mathcal{C}_k)$$

Objective Tries to minimise the Error function; the number of incorrectly classified input vectors:

$$\arg\min_{\boldsymbol{w}} \ E_P(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \ -\sum_{n \in \mathcal{M}} \boldsymbol{w}^T x_n t_n,$$

where $\mathcal{M}$ is the set of misclassified training vectors.

Training Iterate over each training example x_n, and update the weight vector if misclassification:

$$\begin{aligned} \boldsymbol{w}^{i+1} &= \boldsymbol{w}^i + \eta \Delta E_P(\boldsymbol{w}) \\ &= \boldsymbol{w}^i + \eta x_n t_n, \end{aligned}$$

where typically $\eta = 1$.

For the multiclass perceptron:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i + \phi(x, t) - \phi(x, y(x))$$

Support vector machines

A maximum margin classifier: finds the separating hyperplane with the maximum margin to its closest data points.

$$y(x) = \sum_{n=1}^{N} \lambda_n t_n x^T x_n + w_0$$

Primal

$$\arg\min_{\boldsymbol{w}, w_0} \ \frac{1}{2} ||\boldsymbol{w}||^2$$

$$\text{s.t.} \quad t_n(\boldsymbol{w}^T x_n + w_0) \geq 1 \quad \forall n$$

Dual

$$\tilde{\mathcal{L}}(\wedge) = \sum_{n=1}^{N} \lambda_n - \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m t_n t_m x_n^T x_m$$

$$\text{s.t.} \quad \lambda_n \geq 0, \quad \sum_{n=1}^{N} \lambda_n t_n = 0, \quad \forall n$$

- Quadratic Programming (QP)
- SMO, Sequential Minimal Optimisation (chunking).

Linear discriminant analysis

	Perceptron	Support vector machines	Linear discriminant analysis
Regularization	The Voted Perceptron: run the perceptron i times and store each iteration's weight vector. Then: $$y(x) = \text{sign}\left(\sum_i c_i \times \text{sign}(\boldsymbol{w}_i^T x)\right),$$ where c_i is the number of correctly classified training instances for $\boldsymbol{w}_i$.	The soft margin SVM: penalise a hyperplane by the number and distance of misclassified points. **Primal** $$\arg\min_{\boldsymbol{w}, w_0} \frac{1}{2}\|\|\boldsymbol{w}\|\|^2 + C\sum_{n=1}^{N}\xi_n$$ $$\text{s.t.} \quad t_n(\boldsymbol{w}^T x_n + w_0) \geq 1 - \xi_n, \quad \xi_n > 0 \quad \forall n$$ **Dual** $$\tilde{\mathcal{L}}(\wedge) = \sum_{n=1}^{N}\lambda_n - \sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m t_n t_m x_n^T x_m$$ $$\text{s.t.} \quad 0 \leq \lambda_n \leq C, \quad \sum_{n=1}^{N}\lambda_n t_n = 0, \quad \forall n$$	
Complexity	$\mathcal{O}(INML)$, since each combination of instance, class and features must be calculated.	• QP: $\mathcal{O}(n^3)$; • SMO: much more efficient than QP, since computation based only on support vectors.	
Non-Linear	Use a kernel $K(x, x')$, and 1 weight per training instance: $$y(x) = \text{sign}\left(\sum_{n=1}^{N} w_n t_n K(x, x_n)\right)$$ . . . and the update: $$w_n^{i+1} = w_n^i + 1$$	Use a non-linear kernel $K(x, x')$: $$\begin{aligned} y(x) &= \sum_{n=1}^{N}\lambda_n t_n x^T x_n + w_0 \\ &= \sum_{n=1}^{N}\lambda_n t_n K(x, x_n) + w_0 \end{aligned}$$ $$\begin{aligned} \tilde{\mathcal{L}}(\wedge) &= \sum_{n=1}^{N}\lambda_n - \sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m t_n t_m x_n^T x_m \\ &= \sum_{n=1}^{N}\lambda_n - \sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m t_n t_m K(x_n, x_m) \end{aligned}$$	
Online learning	The perceptron is an online algorithm per default.	Online SVM. See, for example: • *The Huller: A Simple and Efficient Online SVM*, Bordes & Bottou (2005) • *Pegasos: Primal Estimated sub-Gradient Solver for SVM*, Shalev-Shwartz et al. (2007)	

Mixture Models

DESCRIPTION A probabilistic clustering algorithm, where clusters are modelled as latent Guassians and each data point is assigned the probability of being drawn from a particular Gaussian.

MODEL Assignments to clusters by specifying probabilities

$$p(x^{(i)}, z^{(i)}) = p(x^{(i)}|z^{(i)})p(z^{(i)})$$

... with $z^{(i)} \sim$ Multinomial(γ), and $\gamma_{nk} \equiv p(k|x_n)$ s.t. $\sum_{j=1}^{k} \gamma_{nj} = 1$. I.e. want to maximise the probability of the observed data $\boldsymbol{x}$.

OBJECTIVE

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \pi, \mu, \Sigma) &= \log p(\boldsymbol{x}|\pi, \mu, \Sigma) \\ &= \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}_k(x_n|\mu_k, \Sigma_k) \right)\end{aligned}$$

TRAINING **E**xpectation: For each n, k set:

$$\begin{aligned}\gamma_{nk} &= p(z^{(i)} = k|x^{(i)}; \gamma, \mu, \Sigma) \quad (= p(k|x_n)) \\ &= \frac{p(x^{(i)}|z^{(i)} = k; \mu, \Sigma)p(z^{(i)} = k; \pi)}{\sum_{j=1}^{K} p(x^{(i)}|z^{(i)} = l; \mu, \Sigma)p(z^{(i)} = l; \pi)} \\ &= \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}\end{aligned}$$

Maximisation:

$$\begin{aligned}\pi_k &= \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} \\ \Sigma_k &= \frac{\sum_{n=1}^{N} \gamma_{nk}(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma_{nk}} \\ \mu_k &= \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}\end{aligned}$$

k-means

A hard-margin, geometric clustering algorithm, where each data point is assigned to its closest centroid.

Hard assignments $r_{nk} \in \{0, 1\}$ s.t. $\forall n \sum_k r_{nk} = 1$, i.e. each data point is assigned to one cluster k.

Geometric distance: The Euclidean distance, l^2 norm:

$$||x_n - \mu_k||_2 = \sqrt{\sum_{i=1}^{D} (x_{ni} - \mu_{ki})^2}$$

$$\arg\min_{\boldsymbol{r}, \mu} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||_2^2$$

... i.e. minimise the distance from each cluster centre to each of its points.

Expectation:

$$r_{nk} = \begin{cases} 1 & \text{if } ||x_n - \mu_k||^2 \text{ minimal for } k \\ 0 & \text{o/w} \end{cases}$$

Maximisation:

$$\mu_{\text{MLE}}^{(k)} = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}},$$

where $\mu^{(k)}$ is the centroid of cluster k.

	Mixture Models	***k*-means**
Regularization	The mixture of Gaussians assigns probabilities for each cluster to each data point, and as such is capable of capturing ambiguities in the data set.	Only hard-margin assignment to clusters.
Complexity	To be added.	The heuristic solution under the assumption of each point perturbed by a normal distribution with mean 0 and variance σ^2 incurs a time complexity $O(n^{34}k^{34}d^8log^4(n)/sigma^6)$.
Non-Linear	Not applicable.	For non-linearly separable data, use kernel k-means as suggested in: *Kernel k-means, Spectral Clustering and Normalized Cuts*, Dhillon et al. (2004).
Online learning	Online Gaussian Mixture Models. A good start is: *A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants*, Neal & Hinton (1998).	Sequential k-means: update the centroids after processing one point at a time.

References

[1] Kevin P Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[2] Trevor. Hastie, Robert. Tibshirani, and J Jerome H Friedman. *The elements of statistical learning*, volume 1. Springer New York, (Online version), 2013.

[3] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.