# Statistical Pattern Recognition Notes

Daniel Nichols

## Contents

## 1 Overview Definitions

### 1.1 Types of Learning

**Definition 1 (Supervised Learning)** *Statistical learning where **all samples** in the dataset have a known label.*

**Definition 2 (Unsupervised Learning)** *Statistical learning where **no samples** in the dataset have a known label.*

**Definition 3 (Semi-Supervised Learning)** *Statistical learning where* ***some samples*** *in the dataset have a known label.*

## 1.2 Types of Models

**Definition 4 (Parametric Models)** *Model is determined by a set of parameters with size* ***independent*** *of the size of the dataset.*

**Definition 5 (Non-Parametric Models)** *Model is determined by a set of parameters with size* ***dependent*** *of the size of the dataset.*

## 1.3 Learning Philosophy

There are always 2 big arguments in the philosophy of statistical learning and how we should go about modeling prediction problems. These arguments are Discriminative vs Generative and Frequentist vs Bayesian. To explain, consider an output $\boldsymbol{y}$, dataset of observations $\mathcal{X}$, and parameters $\boldsymbol{\theta}$.

Table 1: Approaches to Statistical Learning

| | **Frequentist** | **Bayesian** |
|---|---|---|
| **Discriminative** | $p(\boldsymbol{y}; \mathcal{X}, \boldsymbol{\theta})$ | $p(\boldsymbol{y}, \boldsymbol{\theta}; \mathcal{X}) = p(\boldsymbol{y}\vert\boldsymbol{\theta}; \mathcal{X})p(\boldsymbol{\theta})$ |
| **Generative** | $p(\boldsymbol{y}, \mathcal{X}; \boldsymbol{\theta})$ | $p(\boldsymbol{y}, \mathcal{X}, \boldsymbol{\theta}) = p(\boldsymbol{y}, \mathcal{X}\vert\boldsymbol{\theta})p(\boldsymbol{\theta})$ |

**Definition 6 (Discriminative Modeling)** *Assumes the final distribution of the output class is parameterized by a fixed and deterministic set of parameters* $\boldsymbol{\theta}$*.*

**Definition 7 (Generative Modeling)** *Assumes the output class's final distribution is parameterized by a random variable* $\boldsymbol{\theta}$ *for which we must also model a distribution.*

**Definition 8 (Frequentist Approach)** *Only attempt to model the probability of observations given the parameters.*

**Definition 9 (Bayesian Approach)** *Model the joint probability of both the data and parameters.*

# 2 Model Performance And Metrics

Typically when using a machine learning model or parameter estimates it is beneficial to study how good those estimates perform. We can do this practically (i.e. training accuracy), but we can also study the theoretical performance of models.

**Definition 10 (Mean Squared Error)** *Let* $\hat{\theta}$ *give a parameter estimate of the true parameter* $\theta_0$*. Then the mean squared error of our parameter estimate is given by*

$$\begin{aligned} MSE(\hat{\theta}) =& \mathbb{E}\left[(\hat{\theta} - \theta_0)^2\right] \\ =& \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right] + \left(\mathbb{E}[\hat{\theta}] - \theta_0\right)^2 \\ =& Var(\hat{\theta}) + Bias^2(\hat{\theta}) \end{aligned}$$

We know that being unbiased does not necessarily minimize the MSE of an estimator because

$$\min_{\hat{\theta} \in \mathbb{R}^l} \text{MSE}(\hat{\theta}) \leq \min_{\hat{\theta}: \mathbb{E}[\hat{\theta}] = \theta_0} \text{MSE}(\hat{\theta})$$

However, if we have an unbiased estimator then we do have a methodology for iteratively reducing its MSE. To do this we need to first find a sufficient statistic for the parameter we are estimating.

**Definition 11 (Sufficient Statistic)** *$T(\mathcal{X})$ is a sufficient statistic of $\boldsymbol{\theta}$ if $P(\mathcal{X}|T(\mathcal{X});\boldsymbol{\theta})$ is independent of $\boldsymbol{\theta}$. We can prove a statistic $T(\mathcal{X})$ is sufficient by showing that $P(\mathcal{X};\boldsymbol{\theta})$ can be written in the form*

$$P(\mathcal{X};\boldsymbol{\theta}) = h(\mathcal{X})g(T(\mathcal{X});\boldsymbol{\theta})$$

*such that $h$ is independent of $\boldsymbol{\theta}$ and $g$ is only dependent on $\boldsymbol{\theta}$ through $T(\mathcal{X})$.*

Now given a statistic for a parameter $\boldsymbol{\theta}$ we can continually improve the MSE of an unbiased estimator $\hat{\boldsymbol{\theta}}$ using the Rao-Blackwell Theorem.

**Theorem 1 (Roa-Blackwell)** *If $\hat{\boldsymbol{\theta}}$ is unbiased and $T(\mathcal{X})$ is a sufficient statistic for $\boldsymbol{\theta}$, then*

$$MSE(\mathbb{E}[\hat{\boldsymbol{\theta}}|T(\mathcal{X})]) \leq MSE(\hat{\boldsymbol{\theta}})$$

## 3 Reproducing Kernel Hilbert Spaces

### 3.1 Definitions

**Definition 12 (Hilbert Space)** *An inner-product space $\mathcal{H}$ is a Hilbert space if it is also a complete metric space over the metric $\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x}\rangle}$. An inner product space is a vector space for which an inner-product is defined.*

**Definition 13 (Complete Metric Space)** *A space $V$ for which every Cauchy sequence in $V$ has a limit in $V$. A sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ is Cauchy for some metric $d$ if for all $r > 0$ there exists a positive integer $N$ such that for all $m, n > N$ then $d(\boldsymbol{x}_m, \boldsymbol{x}_n) < r$. Thus, if a space $V$ over some metric $d$ is complete, then all Cauchy sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ in $V$ must have $M = \lim_{n\to\infty} \boldsymbol{x}_n$ such that $M \in V$.*

**Definition 14 (Reproducing Kernel Hilbert Space)** *Let $\mathcal{H}$ be a Hilbert space over real-valued functions defined on some arbitrary set $\mathcal{X}$. The set of functions on $\mathcal{X}$ can be considered an inner-product space, since we can define the inner-product of functions $f(\cdot)$ and $g(\cdot)$ as*

$$\langle f, g\rangle = \sqrt{\int_{\mathcal{X}} f(\boldsymbol{x})g(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}.$$

*$\mathcal{H}$ is a Reproducing Kernel Hilbert Space if there exists a function $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ with the following two properties:*

1. *$\kappa(\cdot, \boldsymbol{x}) \in \mathcal{H}$ for all $\boldsymbol{x} \in \mathcal{X}$*
2. *$f(\boldsymbol{x}) = \langle f, \kappa(\cdot, \boldsymbol{x})\rangle_{\mathcal{H}} \quad \forall f \in \mathcal{H}\, \forall \boldsymbol{x} \in \mathcal{X}$ (a.k.a. the reproducing property)*

*The second property and the nomenclature that $\kappa$ is called a kernel gives this special space its name.*

**Definition 15 (Feature Map)** *Let $\mathcal{H}$ be a Reproducing Kernel Hilbert space (RKHS) associated with a kernel $\kappa(\cdot,\cdot)$ and some arbitrary set $\mathcal{X}$. Then the following $\phi(\boldsymbol{x})$ is a feature map to the feature space $\mathcal{H}$*

$$\mathcal{X} \ni \boldsymbol{x} \mapsto \phi(\boldsymbol{x}) \triangleq \kappa(\cdot, \boldsymbol{x}) \in \mathcal{H}$$

*Notice that $\phi$ maps vectors to functions. In practice $\mathcal{H}$ has infinite dimension, so we are mapping points in $\mathcal{X}$ from finite dimensions to the "much higher" number of infinite dimensions.*

**Definition 16 (Kernel Trick)** *A popular trick in machine learning is to use the inner product of $\phi$'s for feature mapping. $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$ are mapped to infinite dimensions by $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{y})$, but their inner-product*

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y})\rangle_{\mathcal{H}} = \langle \kappa(\cdot, \boldsymbol{x}), \kappa(\cdot, \boldsymbol{y})\rangle_{\mathcal{H}} = \kappa(\boldsymbol{x}, \boldsymbol{y})$$

*by the reproducing property. Thus, the infinite dimension inner-product of the mappings can be represented by the scalar $\kappa(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}$.*

**Definition 17 (Positive Definite Kernel)** *A kernel $\kappa$ is positive definite if*

$$\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \geq 0 \quad \forall a_n, a_m \in \mathbb{R}$$

*This can also be written as $\boldsymbol{a}^\intercal \mathcal{K} \boldsymbol{a} \geq 0$ where $\mathcal{K}$ is a matrix such that $\mathcal{K}_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Thus, $\mathcal{K}$ must be positive semi-definite.*

Some common examples of positive definite kernels:

1. Radial Basis Function (or Gaussian): $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}{2\sigma^2}\right)$
2. Homogeneous Polynomial: $\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\intercal \boldsymbol{y})^r$
3. Inhomogeneous Polynomial: $\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^\intercal \boldsymbol{y} + c)^r$
4. Laplacian: $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-t\|\boldsymbol{x}-\boldsymbol{y}\|\right)$
5. Spline: $\kappa(\boldsymbol{x}, \boldsymbol{y}) = B_{2p+1}\left(\|\boldsymbol{x}-\boldsymbol{y}\|_2^2\right)$ where $B_n(\cdot) \triangleq \bigotimes_{i=1}^{n+1} \chi_{[-\frac{1}{2},\frac{1}{2}]}(\cdot)$.
6. Sinc: $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \operatorname{sinc}(\boldsymbol{x}-\boldsymbol{y})$

**Definition 18 (Positive Definite Kernel Functions and RKHS Relationship)** *Every positive definite kernel $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ defines a unique RKHS such that $\kappa(\cdot,\cdot)$ is a reproducing kernel. This goes both ways as well. An RKHS is defined by a unique positive definite kernel.*

**Definition 19 (Representer Theorem)** *The main reason all this is important. Let $J : \mathcal{H} \mapsto \mathbb{R}$ be a function defined by*

$$J(f) = \sum_{n=1}^{N} \mathcal{L}(y_n, f(\boldsymbol{x}_n)) + \lambda\Omega(\|f\|)$$

*where $f \in \mathcal{H}$, $\Omega : [0, +\infty] \mapsto \mathbb{R}$ is strictly monotonic increasing, and $\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$. Then the minimization problem*

$$f^\star = \operatorname*{argmin}_{f \in \mathcal{H}} J(f)$$

*for some $\theta_n \in \mathbb{R}$ can be represented by the linear combination*

$$f^\star = \sum_{n=1}^{N} \theta_n \kappa(\cdot, \boldsymbol{x}_n)$$

This is important because $J$ can model the empirical loss (or risk) of a statistical prediction model. If $f(\boldsymbol{x}_n)$ makes some prediction for input $\boldsymbol{x}_n$ and $y_n$ is the ground truth, then $\mathcal{L}(y_n, f(\boldsymbol{x}_n))$ can model the loss of that classification. $\Omega(\|f\|)$ exists as a regularization term to keep the complexity of $f^\star$ lower (i.e. the VC dimension), but we can set $\lambda = 0$ to avoid this if necessary.

This is tremendously useful for machine learning. Particularly because the minimization problem $\min_f J(f)$ is very difficult. Rather than finding an optimal set of parameters as in traditional statistical models, we are finding an optimal *function*, which is a difficult problem. This is made even more difficult by the fact that $\mathcal{H}$ is often an infinite space. Since $f^\star$ is a linear combination of kernel functions, finding the optimal $f$ is much easier.

## 3.2 Kernel SVM

Section 4 has a full commentary on SVMs. Consider the quadratic program from the SVM optimization problem.

$$\begin{aligned} \text{maximize} \quad & \sum_{n=1}^{N} \lambda_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \lambda_n\lambda_m y_n y_m \boldsymbol{x}_n^\intercal \boldsymbol{x}_m \\ \text{subject to} \quad & 0 \le \lambda_n \le C \\ & \sum_{n=1}^{N} \lambda_n y_n = 0 \end{aligned}$$

We can use the RKHS feature map $\boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ and replace the inner produce $\boldsymbol{x}^\intercal \boldsymbol{x}$ with

$$\begin{aligned} \boldsymbol{x}_n^\intercal \boldsymbol{x}_m \mapsto & \phi(\boldsymbol{x}_n)^\intercal \phi(\boldsymbol{x}_m) \\ & \kappa(\cdot, \boldsymbol{x}_n)^\intercal \kappa(\cdot, \boldsymbol{x}_m) \\ & \langle \kappa(\cdot, \boldsymbol{x}_n), \kappa(\cdot, \boldsymbol{x}_m) \rangle_{\mathcal{H}} \\ & \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \end{aligned}$$

Thus the objective becomes

$$\begin{aligned} & \sum_{n=1}^{N} \lambda_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \lambda_n\lambda_m y_n y_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \\ & = \mathbb{1}^\intercal \boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}^\intercal (\boldsymbol{y}\boldsymbol{y}^\intercal) \mathcal{K} \boldsymbol{\lambda} \end{aligned}$$

## 3.3 Kernel Ridge Regression

Take the standard regression task

$$y_n = g(\boldsymbol{x}_n) + \eta_n$$

where $(y_n, \boldsymbol{x}_n) \in \mathbb{R} \times \mathbb{R}^l$ and $\eta_n \sim \mathcal{N}(0, \sigma_{\eta_n})$ is centered noise. We want to find an optimal approximation of $g$ given values of $\boldsymbol{x}_n$ and $y_n$. Let $f$ be our guess and we will assume $f$ is in an RKHS. From the representer theorem it follows that $f$ can be expressed as

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \theta_n \kappa(\boldsymbol{x}, \boldsymbol{x}_n)$$

If we take the error as $J(\boldsymbol{\theta})$ with regularization parameter $C$ such that

$$\begin{aligned} J(\boldsymbol{\theta}) \triangleq & \sum_{n=1}^{N} \left( y_n - \sum_{m=1}^{N} \theta_m \kappa(\boldsymbol{x}_n, \boldsymbol{x}_m) \right)^2 + C\langle f, f \rangle_{\mathcal{H}} \\ = & (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta})^\intercal (\boldsymbol{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^\intercal \mathcal{K}^\intercal \boldsymbol{\theta} \end{aligned}$$

then the optimal $\hat{\boldsymbol{\theta}}$ is given by

$$(\mathcal{K} + CI)\hat{\boldsymbol{\theta}} = \boldsymbol{y}$$

which in turn, if $\kappa(\boldsymbol{x}) = [\kappa(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \kappa(\boldsymbol{x}, \boldsymbol{x}_N)]^\intercal$, gives the following approximation of $\boldsymbol{y}$

$$\hat{\boldsymbol{y}} = \boldsymbol{y}^\intercal (\mathcal{K} + CI)^{-1} \kappa(\boldsymbol{x})$$

### 3.4 Kernel PCA

Principal Component Analysis (PCA) is another good application of kernels. PCA is used as a change of basis for some data $X \in \mathbb{R}^{n\times m}$ where we have $n$ data samples each with $m$ features. We can change the bases of $X$ in a way such that the covariances between features is maximized. This gives an ideal representation of the data. PCA is typically done by computing the *principal component* of each feature and projecting each data point using these components. PCA can also be used for dimensionality reduction by only projecting data points by the principal components corresponding to the $k$ largest eigenvalues. Thus the data would be reduced from $n \times m$ to $n \times k$.

Formally, PCA finds a matrix $W$ that transforms $X$ with $X_{\text{new}} = XW$. $w_{(k)}$, the k-th row of $W$, is calculated as

$$\boldsymbol{w}_{(k)} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{\boldsymbol{w}^\intercal \hat{\Sigma}_k \boldsymbol{w}}{\boldsymbol{w}^\intercal \boldsymbol{w}}$$

where $\hat{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\intercal$ is the sample covariance (assuming $X$ is centered). It can be shown that $X^\intercal X \propto \hat{\Sigma}$, so we often used $X^\intercal X$ in the numerator instead. Thus, if we want to reduce the data to $k$ dimensions, then $W$ will be the eigenvectors corresponding to the $k$ largest eigenvalues of $X^\intercal X$.

This, however, is a linear mapping from $X$ to $X_{\text{new}}$. What if such a linear data mapping is not ideal? This seems like another great place to utilize RKHS's and the kernel trick. First define a new sample covariance as

$$\hat{\Sigma}_k = \frac{1}{N}\sum_{n=1}^{N} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^\intercal$$

Now it can be shown that $\hat{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}$ has eigenvectors $\boldsymbol{u}$ in the form of $\sum_{n=1}^{N} a_n \kappa(\cdot, \boldsymbol{x}_n)$ and thus

$$\mathcal{K}\boldsymbol{a} = N\lambda \boldsymbol{a}$$

where $\boldsymbol{a} = [a_1, \ldots, a_n]^\intercal$ and $\mathcal{K}_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Finally, we can project values of $X$ using the eigenvectors corresponding to the $k$ largest eigenvalues of $\mathcal{K}$. In practice we also typically normalize $\boldsymbol{a}$ such that $\langle \boldsymbol{u}, \boldsymbol{u}\rangle_{\mathcal{H}} = 1$.

This is a seemingly odd result. We have found better dimensionality reductions by first mapping the data to a higher dimension and then reducing it. However, Kernel PCA outperforms traditional PCA when a linear mapping is not possible.

## 4 SVM

Support vector machines try to find the optimal margin which divides linearly separable data. We can give a loss function in terms of the hinge loss $\mathcal{L}_\rho = \max\{0,\ \rho - yf(\boldsymbol{x})\}$ where $\rho$ is a threshold and hyper-parameter. If we express $f$ as some linear loss function $f = \langle \boldsymbol{\theta}, \boldsymbol{x}\rangle + \theta_0$ and notice that $\|\boldsymbol{\theta}\|$ gives the margin size, then the total loss becomes

$$J(\boldsymbol{\theta}, \theta_0) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{n=1}^{N} \mathcal{L}_\rho(y_n, \langle \boldsymbol{\theta}, \boldsymbol{x}\rangle + \theta_0)$$

which in turn gives us the following optimization problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & y_n(\langle \boldsymbol{\theta}, \boldsymbol{x}\rangle + \theta_0) \geq \rho - \xi_n \\ & \xi_n \geq 0 \end{aligned}$$

If the data is truly linearly separable, then there will always be a subset of points such that $y_n(\langle \boldsymbol{\theta}, \boldsymbol{x}\rangle + \theta_0) \geq 1$ guaranteeing that $\xi_n = 0$.

To select a subset of points we can optimize using the langrangian and ignore any points where the corresponding lagrange multiplier is zero. That is $\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N_s} \lambda_n y_n \boldsymbol{x}_n$ where $\lambda_n$ are the lagrange multipliers and $N_s$ is the number of non-zero multipliers. These multipliers are known as *support vectors* and are only non-zero when $y_n(\langle \boldsymbol{\theta}, \boldsymbol{x} \rangle + \theta_0) = 1$.

We can get the bias $\theta_0$ by first computing $\hat{\boldsymbol{\theta}}$ and then taking the average of the solutions to $y_n(\langle \hat{\boldsymbol{\theta}}, \boldsymbol{x} \rangle + \theta_0) - 1 = 0$ for all $\boldsymbol{x}_n$, $y_n$ corresponding to support vectors.

Applying the langrangian and taking its dual representation gives the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \langle \boldsymbol{x}, \boldsymbol{x} \rangle \\ \text{subject to} \quad & 0 \leq \lambda_n \leq C \\ & \sum_{n=1}^{N} \lambda_n y_n = 0 \end{aligned}$$

This is a standard quadratic program and can be solved for optimal $\lambda_n$ giving the support vectors of the data. Section 3.2 gives an explanation on how to use kernels in a RKHS rather than the linear $f$. This is the best way to separate two classes. For $M$ classes we can construct all $\binom{M}{2}$ combinations of binary svm classifiers and choose whichever class gets picked the most.

## 4.1 KKT Conditions

Given an optimization problem of the form:

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leq 0 \quad \forall i = 1, \ldots, m \\ & h_j(\boldsymbol{x}) = 0 \quad \forall j = 1, \ldots, \ell \end{aligned}$$

where $\boldsymbol{x} \in \mathcal{X}$ and $\mathcal{X}$ is a convex subset of $\mathbb{R}^n$. Let $f$, $g$, and $h$ be continuously differentiable at $\boldsymbol{x}^\star \in \mathcal{X}$ and let $\boldsymbol{x}^\star$ be a local optimum. KKT states that given such a $\boldsymbol{x}^\star$, then there exists $\mu_i$ $(j = 1, \ldots, m)$ and $\lambda_j$ $(j = 1, \ldots, \ell)$ such that the following 4 conditions hold:

**Stationarity**

$$\nabla f(\boldsymbol{x}^\star) + \sum_{i=1}^{m} \mu_i \nabla g_i(\boldsymbol{x}^\star) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(\boldsymbol{x}^\star) = \mathbf{0}$$

**Primal Feasibility**

$$\begin{aligned} g_i(\boldsymbol{x}^\star) \leq 0 \quad \forall i = 1, \ldots, m \\ h_j(\boldsymbol{x}^\star) = 0 \quad \forall j = 1, \ldots, \ell \end{aligned}$$

**Dual Feasibility**

$$\mu_i \geq 0 \quad \forall i = 1, \ldots, m$$

**Complementary Slackness**

$$\mu_i g_i(\boldsymbol{x}^\star) = 0 \quad \forall i = 1, \ldots, m$$

In practice this is useful because we can solve the top-most optimization problem by setting up a series of equations and inequalities from each condition and solving for optimal $\mu_i$'s and $\lambda_i$'s. Notice in the case with no inequality constraints the KKT conditions give Lagrange multipliers.

## 5 K-means

Let $S = \{S_1, \ldots, S_k\}$ be $k$ partitions of the dataset $\boldsymbol{x} \in \mathcal{X}$. We can call each $S_i$ a cluster of the data. We want to find a partitioning, $S$, such that the between-cluster variance is maximal. Letting $\boldsymbol{\mu}_i$ be the mean of cluster $i$ this objective can be expressed as

$$\begin{aligned} S &= \underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \\ &= \underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} |S_i| \operatorname{Var}[S_i] \end{aligned}$$

**Algorithm 1:** $k$-Means Clustering

```
input: Dataset $\mathcal{X}$ with size $N$
       $k$ - the number of clusters
initialize $\boldsymbol{\mu}_0, \ldots, \boldsymbol{\mu}_k$ to random values;
repeat
    for $i \leftarrow$ to $N$ do
        assign $\boldsymbol{x}_i$ the cluster $S_j$ with nearest mean $\boldsymbol{\mu}_j$;
    end
    for $i \leftarrow$ to $k$ do
        $\boldsymbol{\mu}_i \leftarrow \frac{1}{|S_i|} \sum_{\boldsymbol{x} \in S_i} \boldsymbol{x}$;
    end
until no $\boldsymbol{\mu}_i$ change;
Result: Clusters $S_1, \ldots, S_k$
```

## 6 Spectral Clustering

Spectral clustering is another way to look at clustering algorithms that does not necessarily use norm-based metrics. Instead spectral clustering can use similarity measures to cluster, which lets us cluster non-euclidean data without assigning some ill-fit norm to the data.

Let $\mathcal{G}$ be a graph with weighted edges. We want to find a partitioning $A_0, A_1, \ldots, A_k$ of $V$ which minimizes the weights between A's. This can be expressed cleanly as an objective function. Let $w_{ij}$ be the weight between two vertices and $W(A, B) = \sum_{i \in A j \in B} w_{ij}$ the sum of weights between two sets of vertices. Then our object function could be

$$f(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A_i})$$

where $\overline{A}$ is the graph complement. Minimizing $f$ would give the desired outcome, but we have an issue with our objective. The current function will produce lots of single vertex sets. To fix this we can "penalize" the objective function with the size of $A_i$. Instead of the size we are going to use the volume of $A_i$ which takes the weights into account. Let $d_i = \sum_{j=1}^{n} w_{ij}$ be the degree of vertex $i$ and $vol(A) = \sum_{i \in A} d_i$. Now our objective is

$$f(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A_i})}{vol(A_i)}$$

So now we need to find $A_1, \ldots A_k$ such that $f$ is minimized. First let use define a matrix $H \in \mathbb{R}^{n \times k}$ such that

$$H_{ij} = \begin{cases} \frac{1}{\sqrt{vol(A_j)}}, & \text{if } v_i \in A_j \\ 0, & \text{otherwise} \end{cases}$$

Also define the diagonal matrix $D \in \mathbb{R}^{n\times n}$ where $D = \text{diag}(d_1, \dots, d_n)$, $W \in \mathbb{R}^{n\times n}$ where $W_{ij} = w_{ij}$, and the graph laplacian $L = D - W$.

Now we can express our minimization function as

$$\begin{aligned} \min_{A_1,\dots,A_k} f(A_1,\dots,A_k) &= \min_{A_1,\dots,A_k} \text{Tr}\left(H^T L H\right) && \text{subject to } H^T D H = I \\ &= \min_T \text{Tr}\left(T^T D^{-1/2} L D^{-1/2} T\right) && \text{subject to } T^T T = I \end{aligned}$$

where we substitute $T = D^{1/2}H$ in the last step. This is a standard trace minimization problem over a quadratic term with an identity constraint. The solution is $T$ such that $T$ contains the $k$ eigenvectors of $D^{-1/2}LD^{-1/2}$ corresponding to the $k$ largest eigenvalues.

We can now assign vertices to clusters by using the $k$-means clustering algorithm on the rows of $T$. Now vertex $i$ is in a cluster with vertex $j$ if rows $i$ and $j$ of $T$ are assigned to the same cluster.

This algorithm works for clustering on more than graphs. For instance, any coordinate data can by represented by a fully connected graph with the edge between each point as some scaled distance function. An example is the radial basis function,

$$RBF(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_{\boldsymbol{j}}\|_2^2}{2\sigma^2}\right)$$

You can also use any other similarity metric, such as Hamming distance for strings.

## 7 Neural Networks

A neural network is a series of layers with multiple neurons. Each neuron in a layer is connected to every neuron in the next layer (excluding the last one). Each connection has a weight associated with it which produces the output of the proceeding neuron.

For example let $y_i^{(r)}$ give the output of neuron $i$ in the $r$-th layer. We want to calculate the value of neuron $i$ in layer $r+1$. This is the sum of all the weights along each edge multiplied by their corresponding neuron. If $\theta_{ji}^{(r)}$ gives the weight from neuron $j$ in layer $r$ to neuron $i$ in layer $r+1$, then the value for $y_i^{(r+1)}$ can be calculated as

$$\begin{aligned} z_i^{(r+1)} =& \theta_{n0} + \sum_{n=1}^{\ell} \theta_{ni}^{(r)} y_n^{(r)} \\ \implies y_i^{(r+1)} =& f(z_i) \end{aligned}$$

This can be vectorized as $z_i^{(r+1)} = \boldsymbol{\theta}_i^{(r)\intercal} \boldsymbol{y}^{(r)}$ and $\boldsymbol{z}^{(r)} = [z_1^{(r)}, \dots, z_\ell^{(r)}]^\intercal$. Likewise if we put all the $\boldsymbol{\theta}$ vectors into the matrix $\Theta^{(r)} = [\boldsymbol{\theta}_1^{(r)}, \dots, \boldsymbol{\theta}_\ell^{(r)}]^\intercal$, then the update becomes

$$\begin{aligned} \boldsymbol{z}^{(r+1)} =& \Theta^{(r)} \boldsymbol{y}^{(r)} \\ \implies \boldsymbol{y}^{(r+1)} =& f(\boldsymbol{z}^{(r+1)}) \end{aligned}$$

Here $f$ is some activation function, which often differs between layers. Common examples are tanh, ReLU, sigmoid, and softmax.

**Definition 20 (Universal Approximation Theorem)** *Let $g : S \mapsto \mathbb{R}$ be a continuous function with $S \subset \mathbb{R}^l$ being compact. Let $\hat{g}$ be the output of a two layer neural network with a sigmoidal activation function $f$. Then for any $\varepsilon > 0$ there exists a $\hat{g}$ with $K_\varepsilon$ hidden nodes such that*

$$|g(\boldsymbol{x}) - \hat{g}(\boldsymbol{x})| < \varepsilon, \quad \forall \boldsymbol{x} \in S$$

Thus a two layer fully connected network can hypothetically approximate any real valued function for some number of hidden units dependent on the error magnitude.

## 7.1 Back-Propagation Algorithm

Now the network can make predictions, but it needs to train to find the optimal $\Theta^{(r)}$ for each layer. To train we first need to define an objective function to optimize. Let $f(\boldsymbol{x}; \boldsymbol{\theta})$ be the output of the neural network parameterized by $\boldsymbol{\theta}$.

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\boldsymbol{y}_n, f(\boldsymbol{x}; \boldsymbol{\theta}))$$

Here $\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R}$ is the loss of the network. Some example loss functions include the squared error ($\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = |\boldsymbol{y} - \hat{\boldsymbol{y}}|^2$) and categorical crossentropy ($\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \boldsymbol{y} \log \hat{\boldsymbol{y}} + (1 - \boldsymbol{y}) \log(1 - \hat{\boldsymbol{y}})$). Since $J(\boldsymbol{\theta})$ gives the average loss for $N$ samples and parameters $\boldsymbol{\theta}$, then we can find optimal parameters by minimizing $J$ with respect to $\boldsymbol{\theta}$. This can be done with a simple gradient descent based schema.

$$\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$$

Computing $\nabla_{\boldsymbol{\theta}} J$ is non-trivial since there are separate $\boldsymbol{\theta}$ in each layer of the network, but can be done with the back-propagation algorithm making use of the chain rule.

The computation of the output layer is straightforward. Let $\delta_{nj}^{(r)}$ denote the gradient for input sample $n$ and neuron $j$ in layer $r$. Then the output layer (layer $L$) is given by

$$\delta_{nj}^{(L)} = \frac{\partial J_n}{\partial z_{nj}^{(L)}}$$

Computing the gradients before the output layer requires the use of the chain rule. Consider the general form

$$\delta_{nj}^{(r-1)} = \frac{\partial J_n}{\partial z_{nj}^{(r-1)}} = \sum_{k=1}^{\ell} \frac{\partial J_n}{\partial z_{nk}^{(r)}} \frac{\partial z_{nk}^{(r)}}{\partial z_{nj}^{(r-1)}}$$

And we know that $z_{nk}^{(r)}$ is given by

$$\sum_{m=1}^{\ell-1} \theta_{km}^{(r)} f(z_{nm}^{(r-1)})$$

which means our gradient computation for each neuron is

$$\delta_{nj}^{(r-1)} = \left( \sum_{k=1}^{\ell} \delta_{nk}^{(r)} \theta_{kj}^{(r)} \right) f'(z_{nj}^{(r-1)})$$

Thus, we can compute the gradients for neurons in layer $r$ using only the values of its weights, the derivative of its output, and the gradients from layer $r + 1$. Since the last layers has easy to compute gradients, then we can start at the back and compute gradients from the last layer to the first. This is where the name back-propagation comes from.

## 7.2 Auto-Encoders

## 7.3 Convolutional Layers

For learning data with lots of features it is often desirable to reduce the number of features by finding and using only the "meaningful" ones. This is often done before the learning using something like PCA or LDA on the data. However, CNNs attempt to integrate the feature selection with the learning algorithm.

To accomplish this we can replace the inner-product in fully connected layers with a convolution operation. Likewise we emply *weight sharing.* In a FC network each neuron has a vector of parameters associated with it. In a convolution schema we limit each neuron to a single parameter, which is in turn calculated using a convolution over the layer's input space.

In the context of image learning these weights are typically arranged in a filter (or set of filters), which are applied to the image using discrete convolutions. When the filter is applied to an image the output matrix is a *feature map*. During training we seek to find the optimal set of filter parameters.

## 8 Sparsity Aware Learning

Sometimes we have data with lots of features, but only a small subset of those features matter for prediction and/or classification. One way to approach this problem is to use data reduction techniques such as PCA or LDA. Another way is to use *sparsity-aware* models, which learn sparse solutions in the feature space.

**Definition 21 ($\ell_p$ Norm)** *Let the $\ell_p$ norm of a vector $\boldsymbol{x} \in \mathbb{R}^d$ for $p > 1$ be given by*

$$\|\boldsymbol{x}\|_p \triangleq \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p}$$

Here $p = 2$ is the standard euclidean distance norm. We can also define some *special* cases for $p = 0$, $p = 1$, and as $p \to \infty$.

$$\begin{aligned}
\|\boldsymbol{x}\|_0 &= \lim_{p\to 0} \sum_{i=1}^{d} |x_i|^p = \sum_{i=1}^{d} \chi_{(0,\infty)}(|x_i|) \\
\|\boldsymbol{x}\|_1 &= \sum_{i=1}^{d} |x_i| \\
\|\boldsymbol{x}\|_\infty &= \lim_{p\to\infty} \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p} = \max_i\{|x_i|\}
\end{aligned}$$

Note here that the $\ell_0$ norm is the number of non-zero elements in the vector, the $\ell_1$ norm is the sum of the absolute value of the elements, and the $\ell_\infty$ norm is the maximum of the absolute value of the vector's elements. While, in a theoretical sense, none of these are norms they are still very useful.

### 8.1 Regression and Regularization

In normal regularized regression we try to find

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_n - \boldsymbol{x}_n^\intercal \boldsymbol{\theta})^2 + \lambda \|\boldsymbol{\theta}\|_2^2$$

Here the $\lambda\|\boldsymbol{\theta}\|_2^2$ regularizes the solution by penalizing large $\boldsymbol{\theta}$ norms. We can try replacing this with $\|\boldsymbol{\theta}\|_0$ or $\|\boldsymbol{\theta}\|_1$ to penalize the number of parameters in $\boldsymbol{\theta}$ at all. This seems nice, but a problem arises in that $\|\boldsymbol{\theta}\|_0$ is not differentiable. Similarly $\|\boldsymbol{\theta}\|_1$ is not differentiable where $\theta_i = 0$, since it is the absolute value. However, we can remedy this, since it is only non-differentiable at a finite number of points. We can use a sub-differential form of the KKT conditions from Section 4.1. So now we want to find

$$\begin{aligned}
\hat{\boldsymbol{\theta}} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{N} (y_n - \boldsymbol{x}_n^\intercal \boldsymbol{\theta})^2 + \lambda \|\boldsymbol{\theta}\|_1 \\
&= \operatorname*{argmin}_{\boldsymbol{\theta}} (\boldsymbol{y} - X\boldsymbol{\theta})^\intercal (\boldsymbol{y} - X\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1
\end{aligned}$$

which we can solve use gradient based methods

$$-2X^\intercal \boldsymbol{y} + 2X^\intercal X \hat{\boldsymbol{\theta}} + \lambda \nabla_{\boldsymbol{\theta}} \|\boldsymbol{\theta}\|_1 = \boldsymbol{0}$$

where

$$\partial|\theta| = \begin{cases} 1, & \text{if } \theta > 0 \\ -1, & \text{if } \theta < 0 \\ [-1,1], & \text{if } \theta = 0 \end{cases}$$

Consider the case where the regressors are mutually orthogonal and of unit norm. Then $X^\intercal X = I$ and the solution to standard unregularized least squares is $\hat{\boldsymbol{\theta}}_{\text{LS}} = (X^\intercal X)^{-1} X^\intercal \boldsymbol{y} = X^\intercal \boldsymbol{y}$. Additionally, the regularized least squares solution with the $\ell_2$ norm becomes $\hat{\boldsymbol{\theta}}_{\text{R}} = \frac{1}{1+\lambda}\hat{\boldsymbol{\theta}}_{\text{LS}}$. Using these values we find that the $\ell_1$ norm regularization has optimal solutions

$$\hat{\theta}_i = \text{sgn}(\hat{\theta}_{\text{LS},i}) \left( |\hat{\theta}_{\text{LS},i}| - \frac{\lambda}{2} \right)_+$$

where $(x)_+ \triangleq \max(x, 0)$. This solution is ideal, but the assumption that $X^\intercal X = I$ is very ambitious. Let $X \in \mathbb{R}^{N\times\ell}$. If $N > \ell$ and $X$ is full rank, then we can achieve $X^\intercal X = I$ with a finite sequence of linear transformations on $X$. However, if $N < \ell$ (we have more features than observations), then the problem is underdetermined as $\text{rank}(X) \le N$, while $\text{rank}(I_\ell) = \ell$.

## 8.2 Characterizing Sparse Solutions

Finding the sparsest solution to a linear problem can be framed as

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \|\boldsymbol{\theta}\|_0 \\ \text{subject to} \quad & X\boldsymbol{\theta} = \boldsymbol{y} \end{aligned}$$

We want to determine if there is a unique solution and, if so, how to find it.

**Definition 22 (Matrix Spark)** *The spark of a matrix $X \in \mathbb{R}^{N\times\ell}$ where $\ell \ge N$ is the minimum number of its linearly dependent columns.*

Based on this definition if we pick any $m$ columns of $X$ such that $m < \text{spark}(X)$, then these columns are linearly independent. Also given this notion of spark we can prove two very important lemmas.

**Lemma 1** *If $\boldsymbol{\theta} \in \text{null}(X)$ and $\boldsymbol{\theta} \neq \boldsymbol{0}$ then*

$$\|\boldsymbol{\theta}\|_0 \ge \text{spark}(X)$$

**Lemma 2** *If the linear system $X\boldsymbol{\theta} = \boldsymbol{y}$ has a solution $\boldsymbol{\theta}^\star$ such that*

$$\|\boldsymbol{\theta}^\star\|_0 < \frac{1}{2}\text{spark}(X)$$

*then $\boldsymbol{\theta}^\star$ is the sparsest solution and is unique.*

These bounds are very helpful, but computing $\text{spark}(X)$ is very difficult as it is combinatorial in the number of columns of $X$. So to help we define a new value which can bound the spark and is a lot easier to compute.

**Definition 23 (Mutual Coherence)** *Let $X \in \mathbb{R}^{N\times\ell}$. Then the mutual coherence of $X$ is defined as*

$$\mu(X) \triangleq \max_{1\le i<j\le\ell} \frac{|\boldsymbol{x}_i^{c\intercal}\boldsymbol{x}_j^c|}{\|\boldsymbol{x}_i^c\|\|\boldsymbol{x}_j^c\|}$$

Note that $\mu(X)$ gives the maximum absolute value of the cross-correlations between columns of $X$. This value becomes very useful due to the following lemma

**Lemma 3** *For any $X \in \mathbb{R}^{N\times\ell}$,*

$$\text{spark}(X) \ge 1 + \frac{1}{\mu(X)}$$

From this we can show that

$$\|\boldsymbol{\theta}\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(X)}\right)$$

Now we have a bound on the $\ell_0$ norm which does not involve computing the spark. In fact this fact leads to an even more interesting result involving the $\ell_1$ norm.

**Lemma 4** *Let $\boldsymbol{y} = X\boldsymbol{\theta}$ be an underdetermined system. That is $X \in \mathbb{R}^{N\times\ell}$, $N < \ell$, and $X$ is full row rank. If a solution $\boldsymbol{\theta}^\star$ exists and*

$$\|\boldsymbol{\theta}^\star\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(X)}\right)$$

*then $\boldsymbol{\theta}^\star$ minimizes the $\ell_0$ and $\ell_1$ norms.*

This is an immense result as minimizing the $\ell_0$ norm is very difficult, however, minimizing the $\ell_1$ norm can be done using linear programs. Linear programming is well studied and lots of optimal solutions are known. Since the solutions are the same, then we can just solve the $\ell_1$ norm mimimizer.

## 8.3 Orthogonal Matching Pursuit Algorithm

While linear programming is very helpful in finding the $\ell_1$ solution we can in fact do it faster using the greedy Orthogonal Matching Pursuit algorithm.

**Algorithm 2:** Orthogonal Matching Pursuit

```
input: Dataset $X \in \mathbb{R}^{N\times\ell}$ with size $\boldsymbol{x}_j$ as the $j$-th column
initialize $\boldsymbol{\theta}^{(0)} = \boldsymbol{0}$, $\boldsymbol{r}^{(0)} = \boldsymbol{y}$, $S^{(0)} = \varnothing$, and $m = 1$
repeat
    $m \leftarrow m + 1$ ;
    $j_m \leftarrow \text{argmax}_j |X_j^\intercal \boldsymbol{r}^{(m-1)}|$ ;
    $S^{(m)} \leftarrow S^{(m-1)} \cup \{j_m\}$;
    $\boldsymbol{\theta}^{(m)} \leftarrow \text{argmin}_{\boldsymbol{\theta}} \|\boldsymbol{y} - \sum_{i\in S^{(m)}} X_i\theta_i\|_2^2$;
    $\boldsymbol{r}^{(m)} \leftarrow \boldsymbol{y} - X\boldsymbol{\theta}^{(m)}$;
until $\boldsymbol{r}^{(m)} = \boldsymbol{0}$;
Result: Sparse $\boldsymbol{\theta}^{(m)}$
```

This algorithm recursively selects columns of $X$ which are are *supports*. At any iteration $S^{(m)}$ holds the indices of the support columns of $X$. After finding support columns the algorithm solves the least squares problem with just the support columns of $X$ to get the optimal value of $\boldsymbol{\theta}$ corresponding to that column. In the ideal case $\boldsymbol{y}$ is always orthogonal to the columns of $X$ not in $S$. This only occurs when $\|\boldsymbol{\theta}\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(X)}\right)$. More generally $\boldsymbol{y}$ always forms sufficiently small angles with the non-support columns. In this case the algorithm terminates when $\|\boldsymbol{r}^{(m)}\|_2 < \varepsilon$ for some small $\varepsilon > 0$.

# 9 Expectation Maximization

If we have $p(\boldsymbol{y}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$ from a regression task and we want to find a distribution for $p(\boldsymbol{y})$ parameterized by non-random $\boldsymbol{\xi}$. This is the maximum likelihood problem of maximizing $p(\boldsymbol{y};\boldsymbol{\xi})$. However, a closed form for this is usually impossible. This is due to the fact that we have two random variables: $\boldsymbol{\theta}$ and $\boldsymbol{y}$. If $\theta$ were known, then we could maximize $p(\boldsymbol{y},\boldsymbol{\theta};\boldsymbol{\xi})$, but $\boldsymbol{\theta}$ is often unknown. This is called a *hidden variable.* We can use the expectation maximizing algorithm to solve these problems with hidden variables.

Let $\boldsymbol{x}$ be a random vector and $\mathcal{X}$ the corresponding set of observations. Now let $\mathcal{X}^l = \{\boldsymbol{x}_1^l, \ldots, \boldsymbol{x}_N^l\}$ be the corresponding set of latent variables. Even thought $\mathcal{X}^l$ cannot be observed assume that we know $p(\mathcal{X}^l|\mathcal{X};\boldsymbol{\xi})$. We want to find $p(\mathcal{X},\mathcal{X}^l;\boldsymbol{\xi})$, however, we cannot since we have no observations for $\mathcal{X}^l$. Because maximizing the expectation of the log-likelihood (in this case) is equivalent to maximizing the evidence function $p(\mathcal{X};\boldsymbol{\xi})$ the EM algorithm maximizes this expectation using an iterative procedure.

**Algorithm 3:** Expectation Maximization

```
input: Some $\varepsilon > 0$
initialize $\boldsymbol{\xi}^{(0)}$ to some arbitrary value and $j = 0$
repeat
    Compute $p(\mathcal{X}^l|\mathcal{X};\boldsymbol{\xi}^{(j)})$;
    Compute $\mathcal{Q}(\boldsymbol{\xi}|\boldsymbol{\xi}^{(j)}) = \mathbb{E}_{p(\mathcal{X}^l|\mathcal{X};\boldsymbol{\xi}^{(j)})}\left[\ln p(\mathcal{X},\mathcal{X}^l;\boldsymbol{\xi})\right]$;
    $\boldsymbol{\xi}^{(j+1)} \leftarrow \text{argmax}_{\boldsymbol{\xi}}\, \mathcal{Q}(\boldsymbol{\xi},\boldsymbol{\xi}^{(j)})$
    $j \leftarrow j+1$;
until $\|\boldsymbol{\xi}^{(j+1)} - \boldsymbol{\xi}^{(j)}\| < \varepsilon$;
Result: $\boldsymbol{\xi}$
```

## 10 Hidden Markov Models

Hidden Markov Models are used on data where observations are not independent. Typically this is represented by a set of states and observations at the corresponding states with the probability of transition between different states.

Let $a_{ij}$ be the probability to transition from state $i$ to state $j$. Let $A$ be a matrix such that $A_{ij} = a_{ij}$. Here we make a homogeneity assumption such that $a_{ij} \geq 0$ and $\sum_{i=1}^{N} a_{ij} = 1$. Let $\{v_1, \ldots, v_M\}$ be a set of observations such that $b_j(k)$ is the probability of observing $v_k$ on state $j$. Additionally, let $B$ be a matrix such that $B_{ij} = b_j(j)$. Finally let the initially probability of state $i$ be given by $\pi_i$.

Given parameters $\lambda = (A, B, \boldsymbol{\pi})$ and a sequence of observations $O = \{O_1, O_2, \ldots, O_T\}$ how can we compute the likelihood $P(O|\lambda)$? Additionally, what is the optimal sequence of states $q_1, q_2, \ldots, q_T$? Finally, if we only have $O$, then how can we estimate $\lambda$? We can write an expression for the first question as

$$P(O|\lambda) = \sum_{\text{all } Q} P(O|Q,\lambda)P(Q,\lambda)$$

where "all $Q$" denotes all possible sequences of transitions $Q$. We can show that

$$\begin{aligned} P(O|Q,\lambda) =& \prod_{t=1}^{T} P(O_t|q_t,\lambda) \\ =& b_1(O_1) \cdot b_2(O_2) \cdots b_T(O_T) \end{aligned}$$

and

$$P(Q|\lambda) = \pi_1 a_{12} a_{23} \cdots a_{T-1,T}$$

Thus computing $P(O|\lambda)$ will take $\approx 2T \cdot N^T$ calculations for $N$ states and $T$ observations. One can quickly see that for even small problems such as $N = 5$ and $T = 100$ we need to do almost $10^{72}$ calculations, which is not feasible. Luckily there is a smarter way to approach the problem by factoring out values from this ginormous sum.

We want to define a sequence $\alpha_t(i) = P(O_1 O_2 \cdots O_t, i|\lambda)$ and compute these partial probabilities using *forward substitution.*

Initially, let

$$\alpha_1(i) = \pi_i b_i(O_1)$$

Next compute each $\alpha$ inductively as

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1})$$

for all $1 \leq t \leq T-1$. Finally, we can compute the total probability as

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

We can also consider a backwards pass over the data defining $\beta_t(i) = P(O_{t+1}O_{t+2}\cdots O_T|i,\lambda)$, which gives the probability of the partial observations from $t$ to the end. This can be solved similarly to $\alpha$ by first assigning

$$\beta_T(i) = 1$$

and then repeatedly computing $\beta_t$ as

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

## 10.1 Viterbi Algorithm

Now we want to address the second problem. Given some sequence of observations and parameters how do we determine what the "optimal" set of sequences is? First let $\gamma_t(i)$ be the probability of being in state $i$ at time $t$ given $O$ and $\lambda$. This can be simply expressed as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

Now we want to find the most likely state at time $t$, which is given by $\operatorname{argmax}_i \gamma_t(i)$.

This approach is near optimal, but has some problems. It only considers the most likely state at a single time point, but does not consider *sequences* of states. To solve this problem let us introduce the *Viterbi Algorithm.*

First let

$$\begin{aligned} \delta_1(i) =& \pi_i b_i(O_1) \\ \psi_1(i) =& 0 \end{aligned}$$

Then define an iterative step

$$\begin{aligned} \delta_t(i) =& \max_i \delta_{t-1}(i) a_{ij} b_j(O_t) \\ \psi_t(i) =& \operatorname*{argmax}_i \delta_{t-1}(i) a_{ij} \end{aligned}$$

The algorithm terminates with

$$\begin{aligned} P^\star =& \max_i \delta_T(i) \\ q_T^\star =& \operatorname*{argmax}_i \delta_T(i) \end{aligned}$$

Finally we choose the optimal sequence as

$$q_t^\star = \psi_{t+1}(q_{t+1}^\star)$$