Recap

- In Bayesian estimation we treat the parameters as random variables.
- We start with a prior distribution on the parameter and use the data to transform it into a posterior distribution.

Recap

- We have

$$f(\theta \mid \mathcal{D}) = \frac{f(\mathcal{D} \mid \theta) f(\theta)}{\int f(\mathcal{D} \mid \theta) f(\theta)\, d\theta}$$

  $f(\theta)$ – prior desity
  $f(\theta \mid \mathcal{D})$ – posterior density
  $f(\mathcal{D} \mid \theta) = \prod f(x_i \mid \theta)$ – data likelihood
- Conjugate Prior ensures that prior and posterior have same parametric form

Recap

- There are different options regarding using the posterior density as an estimate
- MAP estimate: $\hat{\theta}_{\text{MAP}} = \max_\theta f(\theta \mid \mathcal{D})$
- Mean of posterior can also be used as the estimate
- Or we can get the density model as

$$f(x \mid \mathcal{D}) = \int f(x \mid \theta) f(\theta \mid \mathcal{D}) \, d\theta$$

Recap

- We have seen many examples.
- For estimating mean of a Gaussian with known variance, conjugate prior is Gaussian. For estimating variance with known mean, conjugate prior is gamma.
- For Bernoulli, conjugate prior is beta density.
- For a general discrete rv, the conjugate prior is Dirichlet.

Recap – Exponential family of densities

- A density with a (vector) parameter η

$$f(x \mid \eta) \;=\; h(x)\, g(\eta)\, \exp(\eta^T u(x))$$

  where $u(x)$ is, in general, a vector function, is in exponential family
- Many standrad densities such as Bernoulli, binomial, poisson, gamma, beta, Gaussian etc can be put in this form
- There is a uniform method for ML estimation for all members of this family. Similarly there is a general form for conjugate density
- We also 'proved' that ML estimation is consistent for all densities in this family

Recap –Sufficient Statistic

- A statistic S is said to be **sufficient** for parameter θ if $f(\mathcal{D} \mid S, \theta)$ is not a function of θ.
- **Factorization Theorem**: A statistic S is sufficient for θ if and only if the likelihood function can be factorized as

$$f(\mathcal{D} \mid \theta) = g(s, \theta)\, h(\mathcal{D}), \qquad \text{where} \quad s = S(\mathcal{D})$$

- For the exponential family, $S = \sum_i u(x_i)$ is a sufficient statistic for η.

Recap – Recursive estimates

- Both ML and Bayesian estimates can be recast in recursive or online form.
- Using the general scheme of Robbins-Munro algorithm, we can formulate a recursive version of ML estimate (in the large data limit).
- The Bayesian estimates are inherently recursive – the posterior after seeing $n-1$ data samples, becomes the 'current prior' while estimating posterior after n samples.

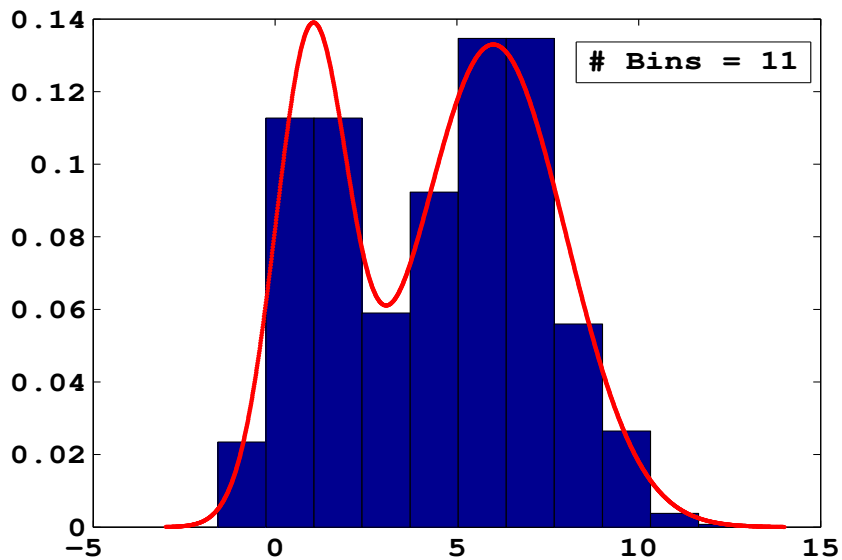
Non-parametric Estimation

- So far we have considered parametric estimation techniques.
- We assumed: $\mathcal{D} = \{x_1, \cdots, x_n\},\ x_i \sim f(x|\theta)$
- Then we can use ML or Bayesian methods for estimating θ.
- We now consider the case where we do not want to assume any parametric form for the density.
- Note that this is relevant only in case of continuous random variables.

- Consider a one dimensional case.
- We are given samples, x_i, $i = 1, \cdots, n$.
- We need to find the density function $f(x)$ and we do not know form of f.
- One simple idea is to learn a piece wise constant approximation to f.

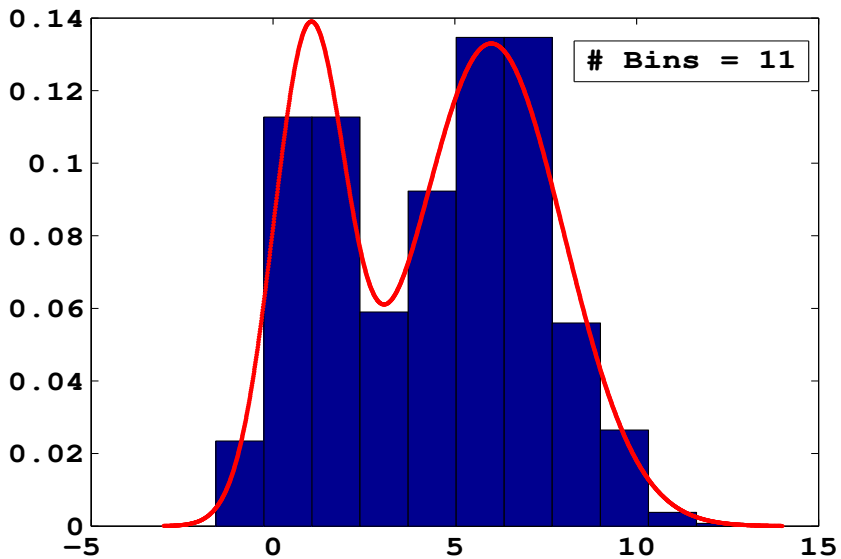
- For this, we cut the x-axis into small intervals and build a function that is constant in each of these intervals.
- If $f(x) = K$ over an interval $[a,\ b]$, then $P[a \leq X \leq b] = K(b - a)$.
- The probability above is well approximated by the fraction of data points that fall in that interval.
- Thus we can approximate f by the histogram of the data.

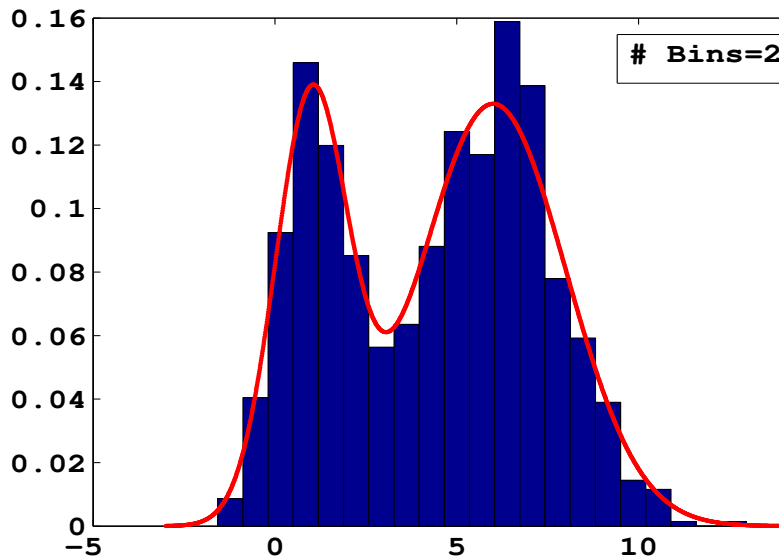
- Here is a simple example



- The quality of approximation depends on the size of intervals.

- We can get better approximation by having finer intervals





- But now, the memory needed to store $\hat{f}$ increases.

- If we extend this idea (in a simple-minded fashion) to d dimensions, the number of bins grows rapidly.
- Also, many bins may be empty.
- Curse of Dimensionality!
- However, this basic idea can be made to work.
- Essentially, we erect bins only where needed.

- Let $B(x)$ be a region (e.g., ball of some small radius) around x. Let
$$\rho = \int_{B(x)} f(x')\, dx'$$
- If f is nearly constant over $B(x)$, then $\rho \approx f(x)\, V$, where V is 'volume' of $B(x)$. Thus,
$$f(x) \approx \frac{\rho}{V}$$

- Suppose out of the n *iid* sample, k samples fall in $B(x)$.
- Then k is binomial with parameter n and ρ.
- Since, for large n, binomial distribution sharply peaks around its mean,

$$k \approx n\,\rho \qquad \text{or} \qquad \rho \approx \frac{k}{n}$$

- Combining these two, we get

$$f(x) \approx \frac{\rho}{V} \approx \frac{k}{nV}$$

- This is the basic idea of finding an approximation of f.
- At any x, we take a small volume V around x and count the number of data samples that fall in this region. This gives approximate value of $f(x)$ as above.

- Choice of V affects the quality of approximation.
- For the approximation $\rho \approx f(x)\, V$ to be good, we need V to be small.
- But if V is very small, unless n is very large, k may be zero most of the time.
- So, choice of size of V is a compromise between these two requirements.

- Let V_n denote the volume when we have n examples and let $f_n(x)$ and k_n denote the corresponding values. ($f_n(x) = \frac{k_n/n}{V_n}$)
- Then, for $f_n \to f$, as $n \to \infty$ we must have

$$V_n \to 0, \qquad k_n \to \infty, \qquad \frac{k_n}{n} \to 0$$

- We need $V_n \to 0$ to get correct estimates.
- If $f(x) \neq 0$, then we need $k_n \to \infty$.
- Finally, $\frac{k_n}{n} \to 0$ is needed to get proper estimate. (We need $nV_n \to \infty$)

- In practice we have only finite data. We choose size of V based on n.
- Actually we have a choice of two approaches.
- We can fix a V and then calculate k. Known as Parzen Window or Kernel density estimate.
- Or, we can fix k and calculate V. Known as k-nearest neightbour method.

Parzen Windows

- Define a function $\phi : \Re^d \rightarrow \Re$ by

$$\begin{aligned} \phi(u) &= 1 \quad \text{if } |u_i| \leq 0.5,\ i = 1, \cdots, d \\ &= 0 \quad \text{otherwise} \end{aligned}$$

  where $u = (u_1, \cdots, u_d)^T$.
- This defines a unit hypercube in $\Re^d$ centered at origin.
- Note that $\phi(u) = \phi(-u)$.

- $\phi(\frac{u-u_0}{h})$ would be a hypercube of side h centered at u_0.
- Let $\mathcal{D} = \{x_1, \cdots, x_n\}$ be the data samples.
- Then, for any x, $\phi(\frac{x-x_i}{h})$ would be 1 only if x_i falls in a hypercube of side h centered at x.
- Hence the number of data points falling in a hypercube of side h centered at x is

$$k = \sum_{i=1}^{n} \phi\left(\frac{x - x_i}{h}\right)$$

- Hypercube of side h in $\Re^d$ has volume h^d.
- Hence we can write our estimated density function as

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} \phi\left(\frac{x - x_i}{h}\right)$$

- Known as Parzen window estimate.
- If we store all x_i we can calculate $f(x)$ at any x.
- The value of h determines the size of volume element (and quality of estimate).

- The Parzen window density estimate is

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} \phi\left(\frac{x - x_i}{h}\right)$$

  This is a kind of generalization of the histogram idea.
- We still have artificial discontinuities like in a histogram.
- But we are essentially erecting bins where needed and counting.
- We can use other ϕ functions also in this form of estimate.

- The ϕ function should satisfy

$$\phi(u) \geq 0, \ \forall u, \qquad \text{and} \qquad \int_{\Re^d} \phi(u)\, du = 1$$

- The hypercube window function that we used satisfies these. Let

$$V = \int_{\Re^d} \phi\left(\frac{u}{h}\right)\, du = \int_{\Re^d} \phi\left(\frac{u - u_0}{h}\right)\, du$$

- For our hypercube window, $V = h^d$.
- Then the Parzen window estimate would be a density:

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V} \phi\left(\frac{x - x_i}{h}\right)$$

- We can choose many ϕ functions and with appropriate V get the density estimate
- This general method is often called Kernel density estimate.

- For example, we can have

$$\phi(u) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{1}{2}||u||^2\right]$$

- This is the d-dimensional Gaussian density
- For this ϕ also we get $V = h^d$.

- The density estimate now is

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{h\sqrt{2\pi}} \right)^{d} \exp \left[- \frac{||x - x_i||^2}{2\, h^2} \right]$$

- We are essentially taking a Gaussian centered at each data point and representing the unknown density as a mixture of these Gaussians.
- This Gaussian kernel gives a smoother density estimate.

- Kernel density estimates are essentially mixture densities.

$$\hat{f}(x) = \frac{1}{n_1} \sum_{i=1}^{n_1} \frac{1}{V} \phi\left(\frac{x - x_i}{h}\right)$$

- We store all the data samples.
- We compute the density whenever needed.

- We next look at convergence of kernel density estimates.
- Let $\hat{f}_n$ denote the density estimate with n samples and similarly let h_n and V_n denote the quantities when sample size is n.
- The density estimate is

$$\hat{f}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \phi \left(\frac{x - x_i}{h_n} \right)$$

- The question we now ask is: does $\hat{f}_n \to f$.

- Define

$$\delta_n(x) = \frac{1}{V_n}\phi\left(\frac{x}{h_n}\right)$$

- Both amplitude and width of δ_n is affected by h_n.
- We assume that as $n \to \infty$, we have $h_n \to 0$ and δ_n tends to a delta function.
- This is true for both the ϕ functions we saw.

- By the properties of ϕ, we have

$$\int \frac{1}{V_n} \phi\left(\frac{x - x_i}{h_n}\right) dx = 1$$

- Hence we have

$$\int \delta_n(x - x_i)\, dx = 1, \quad \forall i, \ \forall n.$$

- We can write $\hat{f}_n$ in terms of δ_n as

$$\begin{aligned}\hat{f}_n(x) &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\phi\left(\frac{x-x_i}{h_n}\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\delta_n(x-x_i)\end{aligned}$$

- $\hat{f}_n(x)$ is random variable because it depends in x_i, $i = 1, \cdots, n$.
- The x_i are iid with density f.

- Let $\bar{f}_n(x)$ be expectation of $\hat{f}_n(x)$. Then

$$\begin{aligned}
\bar{f}_n(x) &= E\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\,\phi\left(\frac{x-x_i}{h_n}\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} E\left[\frac{1}{V_n}\,\phi\left(\frac{x-x_i}{h_n}\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\int \frac{1}{V_n}\,\phi\left(\frac{x-z}{h_n}\right)\,f(z)\,dz \\
&= \int \frac{1}{V_n}\,\phi\left(\frac{x-z}{h_n}\right)\,f(z)\,dz \\
&= \int \delta_n(x-z)\,f(z)\,dz
\end{aligned}$$

- Thus,

$$\bar{f}_n(x) \;=\; \int \delta_n(x - z)\, f(z)\, dz$$

- We know δ_n becomes delta function as $n \to \infty$.
- Hence, as $n \to \infty$, $E[\hat{f}_n(x)] \to f(x),\ \forall x$.
- Now let us calculate variance of $\hat{f}_n(x)$.

- We have

$$\hat{f}_n(x) = \sum_{i=1}^{n} \frac{1}{n} \frac{1}{V_n} \phi\left(\frac{x - x_i}{h_n}\right)$$

- Thus it is sum of n terms each being a function of x_i.
- Since x_i are iid, variance of $\hat{f}_n(x)$ would be sum of variances of these n random variables.

- Let σ_n^2 be variance of $\hat{f}_n(x)$. Then

$$\begin{aligned}
\sigma_n^2 &= n \,\mathrm{Var}\left[\frac{1}{n}\,\frac{1}{V_n}\,\phi\left(\frac{x-x_i}{h_n}\right)\right] \\
&= n\,E\left[\frac{1}{n^2\,V_n^2}\phi^2\left(\frac{x-x_i}{h_n}\right)\right] - n\left[E\left[\frac{1}{n}\,\frac{1}{V_n}\,\phi\left(\frac{x-x_i}{h_n}\right)\right]\right]^2 \\
&= n\,E\left[\frac{1}{n^2\,V_n^2}\phi^2\left(\frac{x-x_i}{h_n}\right)\right] - n\,\frac{1}{n^2}\,\bar{f}_n^2(x) \\
&= \frac{1}{n\,V_n}\int \frac{1}{V_n}\,\phi^2\left(\frac{x-z}{h_n}\right) f(z)\,dz - \frac{1}{n}\,\bar{f}_n^2(x)
\end{aligned}$$

Thus we have

$$\begin{aligned}
\sigma_n^2 &\leq \frac{1}{n\,V_n} \int \frac{1}{V_n}\,\phi^2\left(\frac{x-z}{h_n}\right)\,f(z)\,dz \\
&\leq \frac{1}{n\,V_n}\,\sup(\phi) \int \frac{1}{V_n}\,\phi\left(\frac{x-z}{h_n}\right)\,f(z)\,dz \\
&\qquad\qquad \text{where} \quad \sup(\phi) = \max_u\ \phi(u) \\
&= \frac{\sup(\phi)\,\bar{f}_n(x)}{n\,V_n}
\end{aligned}$$

- Thus we get

$$\sigma_n^2 \leq \frac{\sup(\phi)\, \bar{f}_n(x)}{n\, V_n}$$

  where $\sup(\phi) = \max_u \phi(u)$.
- This implies $\sigma_n^2 \to 0$ as $n \to \infty$.
- This finally shows that the kernel density estimate is a consistent estimate.

- The kernel density estimators are easy to use.
- However, computationally they are expensive.
- Consider 2-class problem with $n_1 + n_2 = n$ training samples.
- If we use Gaussian window function, at any x we need to compute n Gaussians
- If we can model both class conditional densities as Gaussian, the needed computation is much less.

- Another issue is the size of the volume element.
- Choosing the value for h is difficult.
- Sometimes one may choose different h in different parts of the feature space.
- In spite of such issues, Kernel density estimates are very popular non-parametric estimates.

- A different approach to non-parametric density estimation is the k-nearest neighbour approach.
- Here we do not have to choose the size parameter, h.
- Instead we choose k and find V to enclose the k nearest neighbours of x.
- Then we take

$$\hat{f}(x) = \frac{k}{n\,V}$$

- Nearest neighbour density estimate is closely related to nearest neighbour classifier.
- Consider a 2-class problem with prior prabablities p_i and class conditional densities f_i, $i = 0, 1$.
- Let $f(x) = p_0\, f_0(x) + p_1\, f_1(x)$ be the overall density of feature vector.

- Suppose there are n data samples with n_i being from Class-i, $i = 0, 1$.
- We do k-nearest neighbour estimation of f. Suppose the needed volume is V.
- Suppose in this volume there are k_i samples of class-i, $i = 0, 1$.
- Now using the same volume element, we estimate densities f as well as f_i, $i = 0, 1$.

- Then we have

$$\hat{f}_i(x) = \frac{k_i}{n_i V}, \; i = 0, 1, \quad \text{and} \quad \hat{f}(x) = \frac{k}{n V}$$

- The estimates for priors would be $\hat{p}_i = n_i/n$.
- Using these estimates, the posterior probabilities are

$$q_j(x) = \frac{\hat{f}_j(x)\,\hat{p}_j}{\hat{f}(x)} = \frac{k_j}{n_j V}\,\frac{n_j}{n}\,\frac{n V}{k} = \frac{k_j}{k}$$

- Now if we want to implement Bayes classifier, x would be put in class-j if

$$q_j(x) \geq q_i(x), \ \forall i, \quad \text{which implies} \quad k_j \geq k_i, \ \forall i$$

- Thus the Bayes classifier with these estimated densities is the k-nearest neighbour classifier.

- Nearest neighbour methods also give good density estimates.
- Here, we need to fix k.
- As difficult as to fix a h in kernel density estimates.
- One can choose k adaptively also through some heuristics.

- The nearest neighbour methods give rise to nearest neighbour classifier.
- An interesting question: How good is a nearest neighbour classifier?
- Here is a simple analysis.

Nearest Neighbour Classifier

- The nearest neighbour rule: Given a new point, x, find the closest 'prototype', x'; and then assign y' as label for x.
- Given x, the x' is a random variable which is a function of the training set and x.
- We want to know how this classifier compares with that of Bayes.

- Let $q_m(x) = \max_i q_i(x)$. The Bayes classifier assigns c_m to x and its error is $(1 - q_m(x))$.
- If $q_m(x) \approx 1$ then all close neighbours of x are likely to be in the same class and hence nearest neighbour chooses c_m most of the time.
- If $q_m(x) \approx \frac{1}{M}$ (M is number of classes) then most probably its decision would be different from that of Bayes but both would have error prob of about $(1 - \frac{1}{M})$.
- So, can we say, on the average the NN rule would not do too much worse than Bayes?

Analysis of Nearest neighbour (NN) rule

Let

- $p_n(e|x)$ – error of NN (with n samples) on x
- $p^*(e|x)(= 1 - q_m(x))$ – error of Bayes on x
- $p_n(e) = \int p_n(e|x)f(x)dx$
- $p^* = \int p^*(e|x)f(x)dx$ – the Bayes error
- $p_{NN} = \lim_{n\to\infty} p_n(e)$ – (asymptotic) error of NN rule

Then we have

$$p^* \leq p_{NN} \leq p^* \left(2 - \frac{M}{M-1}p^*\right)$$

- Let x' denote the nearest neighbour of x.
- Let y, y' denote the labels of x, x'.
- We have

$$P[y, y'|x, x'] = P[y|x]\ P[y'|x']$$

- Recall $p_n(e|x)$ is error of NN on x. We can write

$$p_n(e|x) = \int p_n(e|x, x') \; f(x'|x) \; dx'$$

- We also have

$$\begin{aligned} p_n(e|x, x') &= 1 - \sum_{j=1}^{M} P[y = c_j, y' = c_j|x, x'] \\ &= 1 - \sum_{j=1}^{M} P[y = c_j|x] \; P[y' = c_j|x'] \end{aligned}$$

$$p_n(e|x) = \int \left(1 - \sum_{j=1}^{M} P[y = c_j|x]\ P[y' = c_j|x']\right)\ f(x'|x)\ dx'$$

- As $n \to \infty$, $f(x'|x) \to \delta(x' - x)$. Thus,

$$\begin{aligned} \lim_{n\to\infty} p_n(e|x) &= 1 - \sum_{j=1}^{M} \left(P[y = c_j|x]\right)^2 \\ &= 1 - \sum_{j=1}^{M} \left(q_j(x)\right)^2 \end{aligned}$$

- Our objective is to get an upper bound on $p_{NN} = \lim_{n\to\infty} p_n(e)$.
- We have

$$\begin{aligned} p_{NN} = \lim_{n\to\infty} p_n(e) &= \lim_{n\to\infty} \int p_n(e|x)\, f(x)\, dx \\ &= \int \left(1 - \sum_{j=1}^{M} (q_j(x))^2\right)\, f(x)\, dx \end{aligned}$$

- Hence we need to find a lower bound $\sum_{j=1}^{M} (q_j(x))^2$.

- Let $z_j = q_j(x)$. Let $\rho = p^*(e|x)$, the Bayes error on x.
- We know that $1 - z_m = 1 - q_m(x) = p^*(e|x) = \rho$.
- Hence we need to solve

$$\begin{aligned} \min \quad & \sum z_i^2 \\ \text{subject to} \quad & z_i \geq 0; \sum_{i \neq m} z_i = \rho \end{aligned}$$

- We will solve this using only the equality constraint.

- The Lagrangian is

$$L(z, \lambda) = \sum z_i^2 + \lambda \left(\sum_{i \neq m} z_i - \rho \right)$$

- Equating the partial derivatives to zero

$$\frac{\partial L}{\partial z_i} = 0 \ \Rightarrow \ 2z_i + \lambda = 0 \ \Rightarrow \ z_i = -\frac{\lambda}{2}$$

- We got $z_i = -\frac{\lambda}{2}$.
- The constraint equation gives us

$$\rho = \sum_{i \neq m} z_i = -(M-1)\frac{\lambda}{2} \;\Rightarrow\; \lambda = \frac{-2\rho}{M-1}$$

- Hence we get the final solution as $z_i = \frac{\rho}{M-1}$.
- Thus, we get a lower bound on $\sum_{j=1}^{M} \left(q_j(x)\right)^2$ by taking

$$q_j(x) = \frac{\rho}{M-1} = \frac{p^*(e|x)}{M-1},\ j \neq m$$

Thus we get

$$
\begin{aligned}
1 - \sum_{j=1}^{M} (q_j(x))^2 &= 1 - (q_m(x))^2 - \sum_{i \neq m} (q_i(x))^2 \\
&\leq 1 - (1 - p^*(e|x))^2 - \left(\frac{p^*(e|x)}{M-1} \right)^2 (M-1) \\
&= 2p^*(e|x) - (p^*(e|x))^2 \left(1 + \frac{1}{M-1} \right) \\
&= 2p^*(e|x) - \frac{M}{M-1} (p^*(e|x))^2
\end{aligned}
$$

We have

$$\begin{aligned} p_{NN} &= \int \left(1 - \sum_{j=1}^{M} (q_j(x))^2\right) \; f(x)\, dx \\ &\leq \int \left(2p^*(e|x) - \frac{M}{M-1}(p^*(e|x))^2\right) \; f(x)\, dx \end{aligned}$$

Recall that the Bayes error is

$$p^* = \int p^*(e|x)\, f(x)\, dx$$

Hence we have

$$\begin{aligned} p_{NN} &= \int \left(1 - \sum_{j=1}^{M} (q_j(x))^2\right) \; f(x)\, dx \\ &\leq \int \left(2p^*(e|x) - \frac{M}{M-1}(p^*(e|x))^2\right) \; f(x)\, dx \\ &\leq 2p^* - \int \frac{M}{M-1}(p^*(e|x))^2 \; f(x)\, dx \end{aligned}$$

To complete the proof we need a bound on this integral.

- We have

$$E\left[p^*(e|x)\right] = \int p^*(e|x)\, f(x)\, dx = p^*$$

- Also, $\text{Var}\left(p^*(e|x)\right) = E\left(p^*(e|x)\right)^2 - (p^*)^2 \geq 0.$
- Hence

$$\int \left(p^*(e|x)\right)^2 f(x)\, dx \geq (p^*)^2$$

- This finally gives us the bound on error of NN rule as

$$\begin{aligned} p_{NN} &\leq 2p^* - \int \frac{M}{M-1}(p^*(e|x))^2 \, f(x) \, dx \\ &\leq 2p^* - \frac{M}{M-1}(p^*)^2 \leq 2p^* \end{aligned}$$

- Thus, if we have large number of prototypes, the probability of error on NN rule is less than twice that of Bayes rule.

Implementing Bayes Classifier

- Bayes classifier is optimal for minimizing risk.
- To implement it, we need class conditional densities.
- Class conditional densities can be estimated, given *iid* samples from each class.
- We could estimate the densities parametrically or non-parametrically.

- Bayes classifier is optimal when we exactly know the posterior probabilities.
- When we estimate densities, there would be inaccuracies.
- This results in the non-optimality of implemented Bayes classifier.
- In general, it is not easy to relate estimation errors to classification errors.

- Bayes classifier is an example of learning Generative models
- The idea was to learn joint distribution of (X, y).
- For this we learn marginal of y (prior probabilities) and conditional of X given y (Class conditional densities).
- Thus, it amounts to learning a distribution given iid data from the distribution.
- Both maximum likelihood and Bayesian estimation that we considered are general techniques to learn a distribution model given data (learning generative models) and they are not limited to the supervised classification problem.

- We need joint distribution of the vector X.
- We need an appropriate model for this.
- Simplest choice is to assume independence – naive Bayes
- Otherwise we need to model dependencies among feature components – in general, not easy
- Bayesian networks and other graphical models are very useful for this.

- Another approach is the discriminative model approach.
- We can model the conditional distribution of y given X.
- We can essentially learn a parametric model for the classifier directly.
- We next consider such an approach in the context of linear classifiers.

Discriminative models

- Consider a 2-class classifier

$$\begin{aligned} h(X) &= 1 \text{ if } g(W, X) > 0 \\ &= 0 \text{ Otherwise} \end{aligned}$$

  where W is a parameter vector and g is called a discriminant function.
- We could take $g(W, X) = q_1(X) - q_0(X)$ and thus can think of this as directly learning the conditional distribution.
- In general, we are using a parametric form for the classifier.
- For the regression problem also we can directly use a prametric model like this.

Linear Classifier for a 2-class problem

- Let $W = (w_0,\ w_1,\ \cdots,\ w_d)^T$ be the parameter vector and let $X = (x_1,\ \cdots, x_d)^T$ be the feature vector. Then

$$\begin{aligned} h(X) &= 1 \text{ if } g(W,X) = \sum_{i=1}^{d} w_i x_i + w_0 > 0 \\ &= 0 \text{ Otherwise} \end{aligned}$$

  is called a linear classifier.
- The $g(W,X)$ is called a linear discriminant function.
- It is linear in parameters, w_i.
- It is also linear in x_i (though it is not important).

- Let $\phi_i(X)$, $i = 1, \cdots, d'$, be some fixed functions.
- Consider a classifier

$$\begin{aligned} h(X) &= 1 \text{ if } \sum_{i=1}^{d'} w_i \phi_i(X) + w_0 > 0 \\ &= 0 \text{ Otherwise} \end{aligned}$$

- This is also a 'linear' classifier (even if ϕ_i are nonlinear).
- The discriminant function is linear in w_i.

- Thus a linear discriminant function can be of the form

$$g(W, X) = \sum_{i=1}^{d'} w_i \phi_i(X) + w_0$$

- We are essentially using $z_i = \phi_i(X)$ as the features.
- As long as ϕ_i are fixed, this is a 'linear' classifier.
- We will use X as the feature vector but will remember that all the algorithms are valid if we use $\phi_i(X)$ instead of x_i.

- Define $\tilde{X} = (1, x_1, \cdots, x_d)^T$, called the **augumented feature vector**.
- Let $W = (w_0, w_1, \cdots, w_d)^T$ be the parameter vector.
- Now we have

$$g(W, X) = w_0 + \sum_{i=1}^{d} w_i x_i = W^T \tilde{X}$$

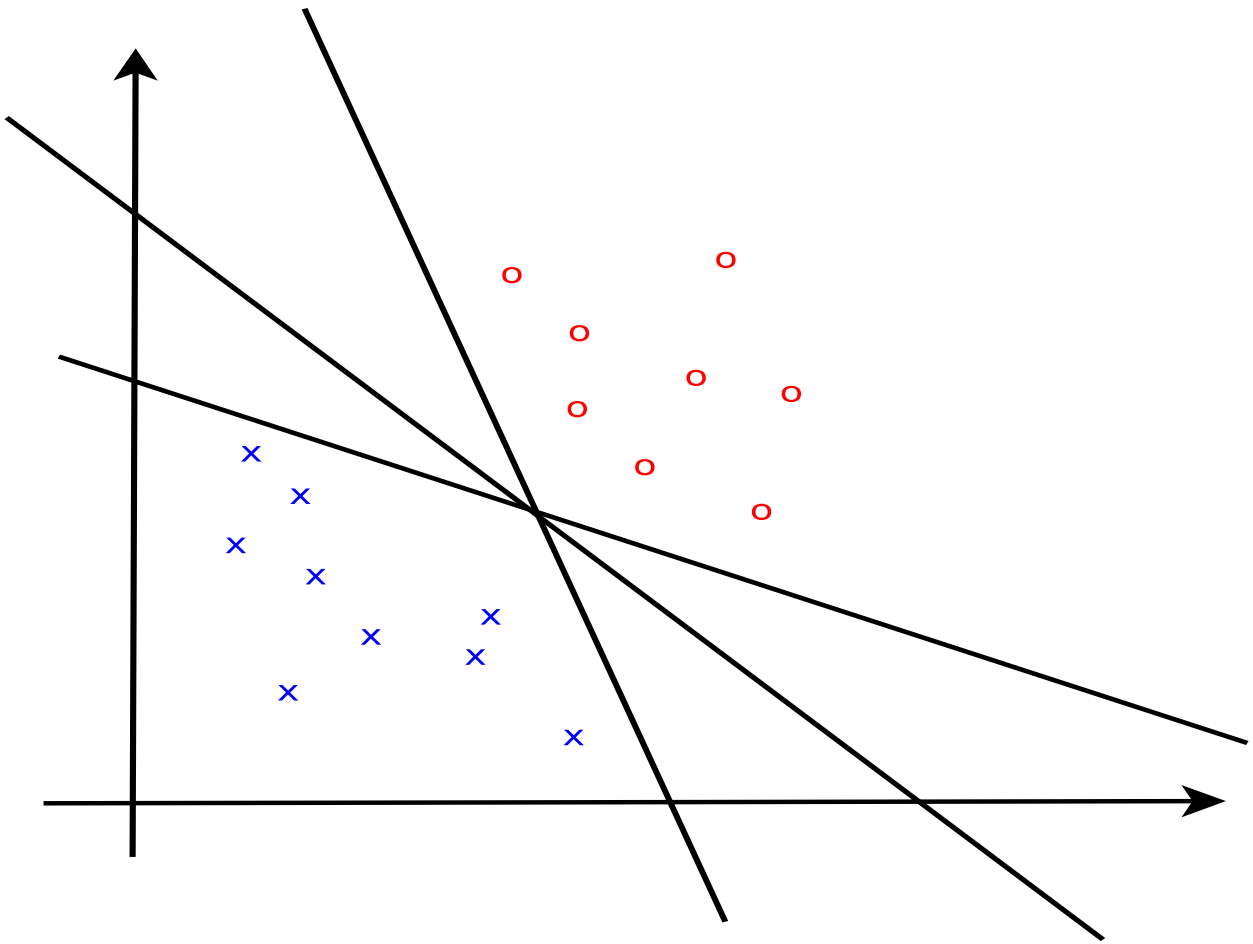
- We assume that the feature vector is augumented (whenever needed) though we write it as X.

- The training set: $\{(X_i, y_i), i = 1, \cdots, n\}$ is said to be **linearly separable** if there exists W^* such that

$$\begin{aligned} X_i^T W^* &> 0 \text{ if } y_i = 1 \\ &< 0 \text{ if } y_i = 0 \end{aligned}$$

- Any W^* that satisfies the above is called a separating hyperplane. (There exist infinitely many separating hyperplanes, if data is linearly separable)

- Example of Linearly separable data:
  (in the original, not augumented, feature space)



Learning linear classifiers

- The classifier is:

$$h(X) = \text{sgn}\left(\sum_{i=1}^{d} w_i x_i + w_0\right) = \text{sgn}\left(W^T X\right)$$

- Need to learn 'optimal' W from the training samples.
- Perceptron learning algorithm is one of the earliest algorithms.
- Finds a separating hyperplane, if it exists.
- We start with this algorithm.

Perceptron Learning Algorithm

- The algorithm is an iterative algorithm to learn W corresponding to a separting hyperplane.
- Let $W(k)$ denote the weight vector at k^{th} iteration.
- At each iteration we pick a training sample.
- Let $X(k)$ be the one picked at k and let $y(k)$ denote its class label.
- At k^{th} iteration we classify $X(k)$ using $W(k)$ and based on the correctness or otherwise of the classification, update $W(k)$ to $W(k+1)$.

- We can keep picking feature vectors one-by-one from the training data (and keep repeatedly going over the training set).
- We stop when the current weight vector correctly classifies all the training data.
- For the 'stopping criterion', we can remember when we had an incorrect classification.
- We think of this as an online or incremental algorithm

Perceptron Learning Algorithm

Let $\Delta W(k) = W(k+1) - W(k)$. Then

$$\begin{aligned}
\Delta W(k) &= 0 && \text{if} \quad W(k)^T X(k) > 0 \ \ \& \ \ y(k) = 1, \quad \text{or} \\
& && \quad\ W(k)^T X(k) < 0 \ \ \& \ \ y(k) = 0 \\
&= X(k) && \text{if} \quad W(k)^T X(k) \leq 0 \ \& \ y(k) = 1 \\
&= -X(k) && \text{if} \quad W(k)^T X(k) \geq 0 \ \& \ y(k) = 0
\end{aligned}$$

- This is a simple 'error correction' algorithm.
- Everytime the current sample is incorrectly classified, we 'locally' try to correct the error.

- Suppose $W(k)^T X(k) \leq 0$ & $y(k) = 1$. Then

$$\begin{aligned} W(k+1)^T X(k) &= (W(k) + X(k))^T X(k) \\ &= W(k)^T X(k) + X(k)^T X(k) \\ &\geq W(k)^T X(k) \end{aligned}$$

- Similarly when $W(k)^T X(k) \geq 0$ & $y(k) = 0$,

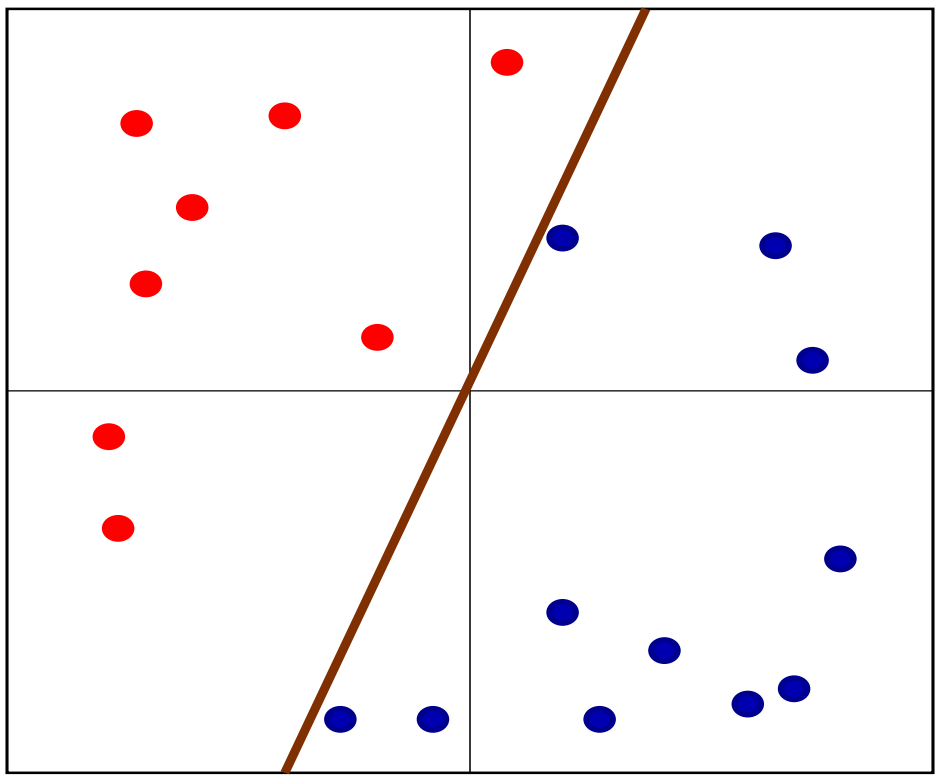
$$W(k+1)^T X(k) \leq W(k)^T X(k)$$

- Thus the corrections are intuitive.

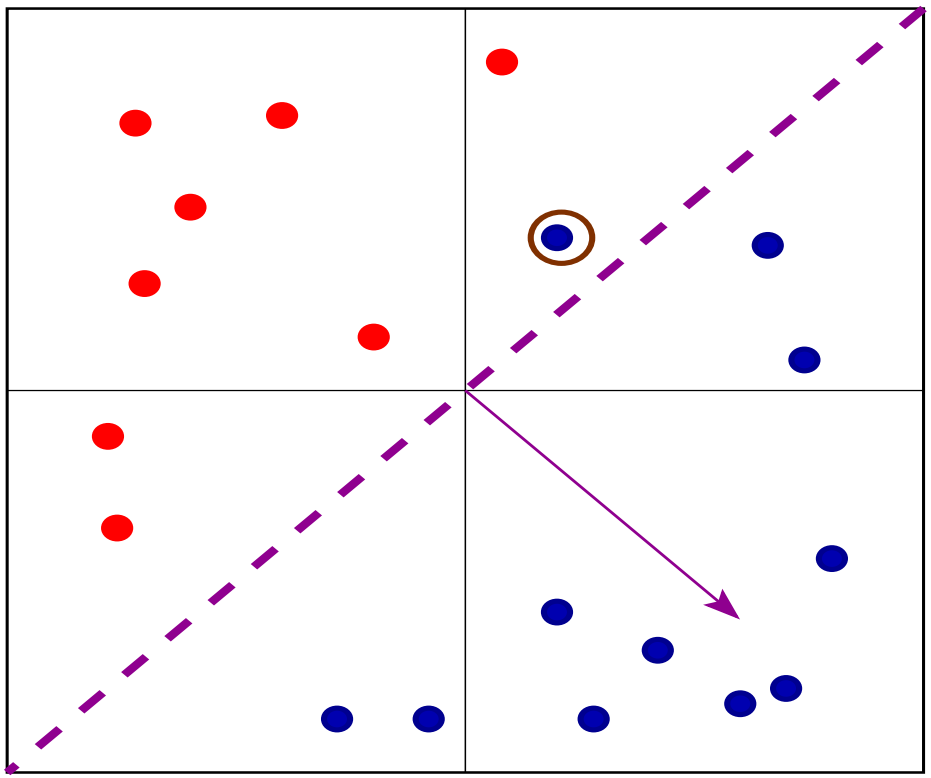
- Thus, the motivation for the algorithm is easy to see.
- However, it is not clear why the algorithm should work.
- Firstly, there is no guarentee that $W(k+1)^T X(k)$ has correct sign. (Note that the 'step size' is arbitrary).
- Secondly, when we correct $W(k)$ to take care of $X(k)$, we may now misclassify some feature vector that $W(k)$ classified correctly.
- Hence, it is remarkable that the algorithm works (as we show later).

Perceptron: Geometric view

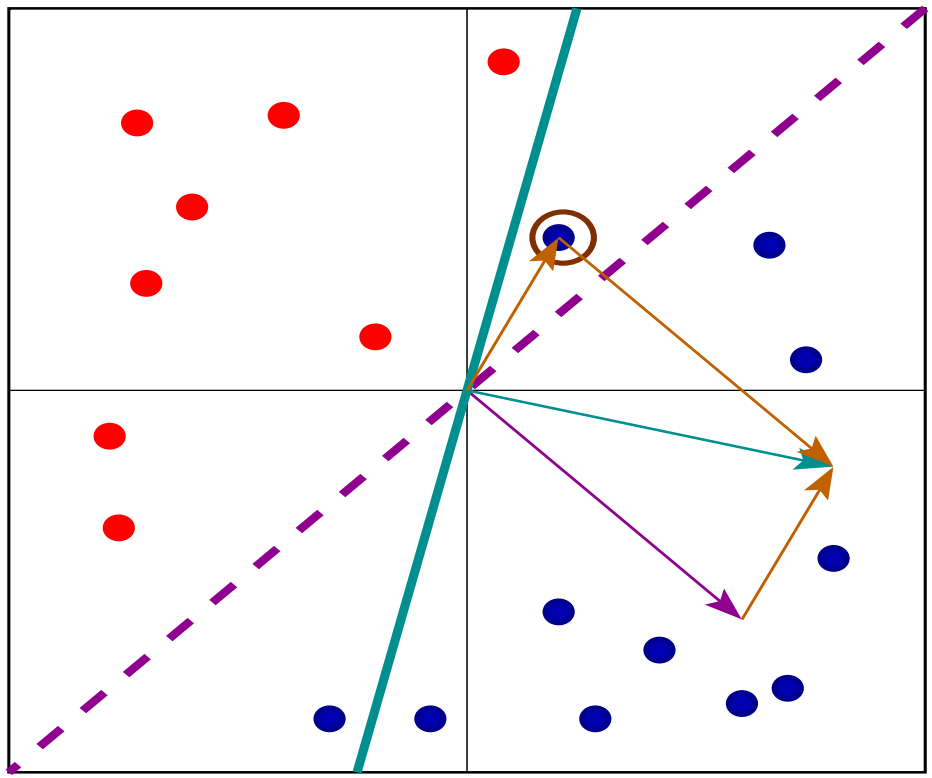
The algorithm has a simple geometric view. Consider the following data set.
(In the 2D feature space but seprable by line through origin)



- Suppose $W(k)$ misclassifies a pattern.



- Suppose $W(k)$ misclassifies a pattern.
- Now the correction made to $W(k)$ can be seen as



Convergence of Perceptron Algorithm

- We will show that the algorithm learns a separating hyperplane.
- That is, if the given data is linearly separable, then the perceptron algorithm stops after some finite number of iterations and finds a separating hyperplane.