Math

Definition (Positive Semidefinite, PSD):

M is PSD if $\langle v, Mv\rangle = v^\top Mv \geq 0, \forall v \in \mathbb{R}^d, M \in \mathbb{R}^{d\times d}$

Proposition (Convex Functions): A twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if the Hessian $\nabla^2 f(x)$ is positive semi-definite for all $x \in \mathbb{R}^n$

Proposition (Fermat's necessary condition): If $\boldsymbol{w}$ is a local extreme of f, then $\nabla f(\boldsymbol{w}) = 0$. If f is convex the condition is $\iff$.

Halfspaces & The Perceptron Algorithm

Definition (Binary classification): Given n known pairs $\{(\boldsymbol{x_i}, y_i)\}$ where $x_i \in \mathbb{R}^d, y_i \in \{\pm 1\}$ we want to learn a classification rule

$h(\boldsymbol{x}) = y$ Such that,

$\mathbb{P}_{(\boldsymbol{x},y)\sim P}[h(\boldsymbol{x}) = y]$ is Large

Definition (Affine Set/Hyper Planes): The signed distance of a vector $\boldsymbol{x}$ is given by

$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$

$\boldsymbol{w}$: Feature Weights , b : Bias

Algorithm (Perceptron):

```
def perceptron(D, d, delta, k):
  w = np.zeros(d)  # Initialize w
  b = 0            # Initialize b
  for t in range(k):
    # Receive training example (xt, yt)
    xt, yt = D[t]
    # Compute prediction ŷ
    y_hat = np.dot(w, xt) + b
    # Check the update condition
    if yt * y_hat <= delta:
      # Update w and b
      w = w + yt * xt
      b = b + yt
    return w, b
```

Note (Padding): $\vec{x_i}' = (x_1, x_2, ..., x_d, 1)^\top$,

$\boldsymbol{w} = (w_1, w_1, ..., w_d, b)$. We can write $\langle \boldsymbol{w}, \vec{x}'\rangle$

Definition (Linear separability): D is linearly separable if $\exists \vec{u}, \|\vec{u}\| = 1$, and a margin $\gamma > 0$ such that

$\forall (\boldsymbol{x}, y) \in D, y(\vec{u}^\top \boldsymbol{x}) > \boldsymbol{\gamma}$

Theorem (Perceptron Convergence theorem): If D is linearly separable with margin $\boldsymbol{\gamma} > 0$ and $\forall (\vec{x}, y) \in D, \|\vec{x}\| \leq \boldsymbol{R}$ the Perceptron algorithm will converge by update

$$k^2\boldsymbol{\gamma}^2 \leq \|w_k\|^2 \leq kR^2,$$

$$k \leq \frac{\boldsymbol{R^2}}{\boldsymbol{\gamma^2}}$$

Note (Multiclass):

- one-vs-all: Train one for each class output $\arg\max_{k=1,..C}(\boldsymbol{w_k}^\top \boldsymbol{x} + b_k)$
- one-vs-one: For each pair (k, k') train $\boldsymbol{w}_{k,k'}$ total of $\binom{c}{2}$ perceptrons, output majority vote.

Example (Loss Functions):

$\max(0, 1 - yf(x)) = (1 - yf(x))^+$ (Hinge Loss)

$\log(1 + e^{-yf(x)})$ (Cross Entropy Loss)

$(y - f(x))^2$ (Squared Error)

Support Vector Machines

Definition (Maximum Margin Classifier): For dataset $\mathcal{D} = \{(\vec{x}_i, y_i)\}, y_i \in \{\pm 1\}$ linearly separated by a hyperplane $w^\top x = -b$ with margin γ

$$\gamma = \underbrace{\min_i \frac{1}{\|w\|} y_i (w^\top x_i + b)}_{\text{Min Distance from the hyperplane}} = \frac{1}{\|w\|} \min_i y_i (w^\top x_i + b)$$

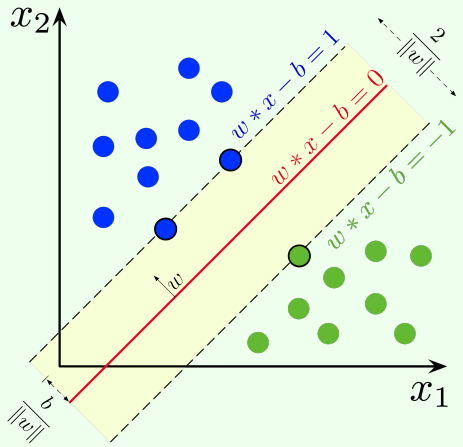
The maximum margin classifier problem is defined as

$$\hat{w}, \hat{b} = \arg\max_{w,b} \frac{1}{\|w\|} \min_i y_i (w^\top x_i + b)$$

Since scaling w, b does not affect the distance we have:

$$\hat{w}, \hat{b} = \arg\max_{w,b} \frac{1}{\|w\|} \quad \text{subject to: } y_i (w^\top x_i + b) \geq 1 \forall i$$

$$\hat{w}, \hat{b} = \arg\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to: } y_i (w^\top x_i + b) \geq 1 \forall i$$



Note (Lagrangian dual of maximum margin):

$$\min_{w,b} \max_{\lambda \geq 0} \mathcal{L}(w, b, \lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i \hat{y}_i - 1)$$

Let $\hat{y}_i = (w^\top x_i + b)$

KKT Conditions:

- Primal feasibility: $y_i \hat{y}_i - 1 \geq 0$ (Each datapoint is correctly classified)
- Dual feasibility: $\lambda_i \geq 0$
- Complementary slackness: $\lambda_i (y_i \hat{y}_i - 1) \geq 0$, either $\lambda_i = 0$ (inactive), when $\lambda_i > 0$ then its directly influencing the decision boundary.

Note (Hard vs Soft Margin SVM):

Hard	**Soft**
$\arg\min_{w,b} \frac{1}{2}\|w\|^2$ Subject to $y_i \hat{y}_i \geq 1$	$\arg\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_i \underbrace{(y_i \hat{y}_i - 1)^+}_{\text{Hinge Loss}}$
$\min_{\lambda_i \geq 0} -\sum_i \lambda_i + \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$	$\min_{0 \leq \lambda \leq C} -\sum_i \lambda_i + \frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle$
$w^\star = \sum_{i=1}^{n} \lambda_i y_i x_i \ , \ \lambda_i \geq 0$	$w^\star = \sum_{i=1}^{n} \lambda_i y_i x_i \ , \ 0 \leq \lambda_i \leq C$
$b^\star = \frac{1}{N_{SV}} \sum_{i \in SV} (y_i - w^\top x)$	$b^\star = \frac{1}{N_{SV}} \sum_{i \in SV} (y_i - w^\top x)$
High Variance	$y_i \hat{y}_i < 0$: Incorrectly classified, $0 \leq y_i \hat{y}_i < 1$: Weakly correct, $y_i \hat{y}_i > 1$: Strong correct,

Kernel Methods

Theorem (Mercer kernels):

(1) Any symmetric function $k : \mathbb{R}^d \times \mathbb{R}^r \to \mathbb{R}$ is a kernel if and only if there exists some $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$

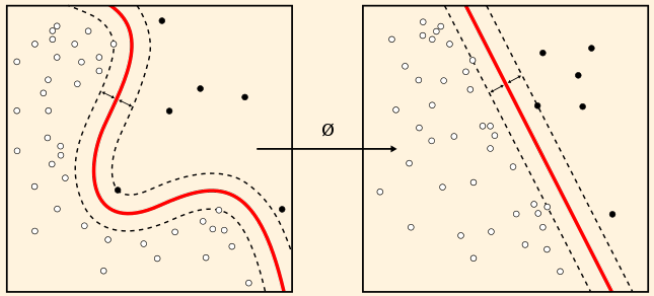
(2) A function $k : \mathbb{R}^d \times \mathbb{R}^r \to \mathbb{R}$ is a Mercer kernel $\iff$ for any $n \in \mathbb{N}, x_i \in \mathcal{X}$ the kernel matrix $\boldsymbol{K}$ with $K_{ij} = k(x_i, x_j)$ is Symmetric and *PSD*

Example (Kernels):

$$\exp\left(-\gamma \|x - x'\|_2^2\right) \quad \text{(Gaussian)}$$

$$\exp(-\gamma \|x - x'\|) \quad \text{(Laplace)}$$

&



Note (Kernel Trick SVM):

$$b^\star = \frac{1}{N_{SV}} \sum_{x_i \in SV} \left(y_i - \sum_{x_j \in SV} \lambda_i y_i K_{ij} \right) \ , \ K_{ij} = k(x_i, x_j)$$

$$\hat{y} = \sum_{x_i \in SV} \lambda_i y_i k(x, x_i) + b^\star$$

Linear Regression

Definition (Regression): Given n known pairs $\{(\boldsymbol{x_i}, y_i)\}$ where $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ and $(\vec{x}_i, y_i) \underset{\text{iid}}{\sim} P$, we want to learn $h : \mathbb{R}^r \to \mathbb{R}$

$$\mathbb{E}_{(\vec{x},y)\sim P}[l_w(\vec{x}, y)] \text{ is small}$$

$$\Rightarrow \arg\min_w \frac{1}{n} \sum l_w(\vec{x}_i, y)$$

Where $\ell_w(\cdot, \cdot)$ is the loss function

Note (Calculus): Let $f : \mathbb{R}^d \to \mathbb{R}$

$$\nabla f(\vec{x}) : \mathbb{R}^d = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_d} \right) \text{ (Gradient)}$$

$$\nabla^2 f(\vec{x}) : \mathbb{R}^{d\times d} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix}$$

Note (Equivalent notation of loss): Let $A \in \mathbb{R}^{n\times(d+1)}$ be a matrix of padded feature vectors. Let $z \in \mathbb{R}^{n\times 1}$ be the vector of outputs in training set. Then we can write total loss as

$$L = \sum_{i=1}^{n} (\langle \boldsymbol{w}, \vec{x}' \rangle - y_i)^2$$

$$L = \|A\boldsymbol{w} - z\|_2^2$$

Note (least-squares loss):

$$L = \|X\boldsymbol{w} - \vec{y}\|_2^2$$

$$\nabla_w L = 2X^\top X\boldsymbol{w} - 2X^\top \vec{y}$$

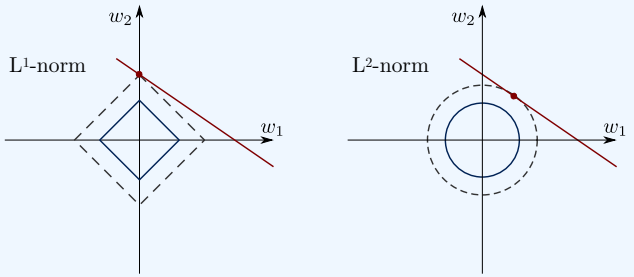
$$\nabla_w^2 L = 2X^\top X \Rightarrow v^\top 2X^\top Xv = 2\|Xv\|_2^2 \geq 0$$

least-squares loss convex. Solving for $\nabla_w L = 0$

$$2X^\top X\boldsymbol{w} - 2X^\top \vec{y} = \vec{0} \to \hat{\boldsymbol{w}} = (X^\top X)^{-1} X^T \vec{y}$$

Note (Regularization):

L_1 Lasso	L_2 Ridge/Tikhonov
$\min_{\boldsymbol{w}} \|A\boldsymbol{w} - \vec{y}\|_2^2 + \boldsymbol{\lambda}\|\boldsymbol{w}\|_1$	$\min_{\boldsymbol{w}} \|A\boldsymbol{w} - \vec{y}\|_2^2 + \boldsymbol{\lambda}\ \|\boldsymbol{w}\|_2^2$
▸ Penalizes non-zero weights. Many w_i will be 0. Weights $\boldsymbol{w}$ will be sparse.	▸ Penalizes large weights, stabilizes the weights.
no closed-form	$\hat{\boldsymbol{w}}(XX^\top + \boldsymbol{\lambda} I) = X^T Y$



Algorithm (Cross-validation):

```
def cross_validation(dataset, k):
  Perf = []
  # create k folds
  folds = folds(dataset, k)
  for λ in [λ1,   λ2, ...]:
    perf_λ = []
    for i in range(k):
      # train on everything except holdout
      w_λ = train(folds[0:i] + folds[i+1:])
      # perf on holdout fold
      perf_λ += [loss(w_λ, folds[i])]
    Perf[λ] = np.mean(perf_λ)
  # λ with the best average performance
  return argmax(Perf, λ)
```

Definition (Entropy):

$$h(X) = \mathbb{E}(-\log p(X))$$

$$= \sum_{i=1}^{n} \Pr[x_i] \log \Pr[x_i] = \int_{\mathcal{X}} p(x) \log p(x) dx$$

Note (Kullback-Leibler divergence):

$$D_{KL}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}$$

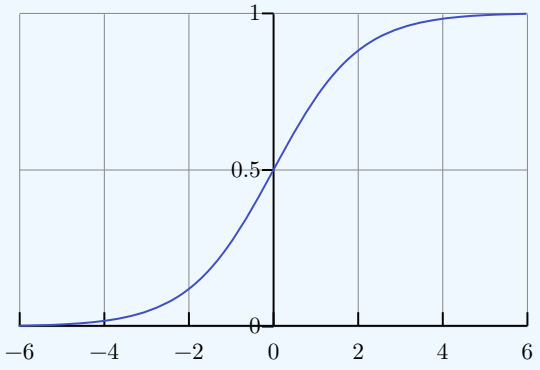
$$D_{KL}(P \parallel Q) = \int p(x) \log \frac{p(x)}{q(x)} dx$$

measures dissimilarity between a reference and model distribution (relative entropy)

Logistic regression

Note (Sigmoid Function):

$$h(x) = \frac{1}{1+\exp(-\langle \boldsymbol{w}, \boldsymbol{x}\rangle)} = \text{sigmoid}(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)$$



Note (Logistic Regression): log-likelihood function for logistic regression is:

$$\log \mathcal{L}(w \mid \boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} -\log\left(1 + e^{-\tilde{y}_i \langle w, x_i\rangle}\right)$$

$$\tilde{y}_i = \begin{cases} -1 \text{ if } y_i = 0 \\ +1 \text{ if } y_i = 1 \end{cases}$$

Cross-entropy loss $l_w(x, \tilde{y}) = \log\left(1 + e^{-\tilde{y}\langle w, x\rangle}\right)$

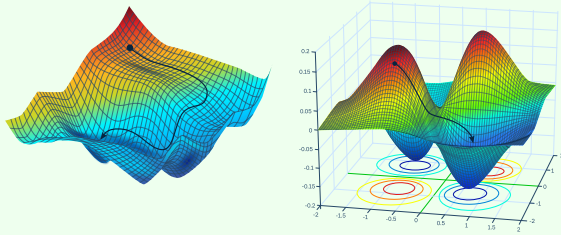
Definition (Gradient descent): In Gradient descent the parameter vector w is updated in the direction opposite to ∇f with step size η (learning rate)

$$w_t = w_{t-1} - \eta \nabla f(w_{t-1})$$

For a loss function this is computed over a batch of n

$$\nabla l_w(x, y) = \frac{1}{n} \sum_{i-1}^{n} \nabla l_{w_{t-1}}(x_i, y_i)$$

In Stochastic gradient descent the gradient is estimated using a random sample (minibatch) of size m



,

$$w_t = w_{t-1} - \underbrace{\left(\boldsymbol{\nabla}^2 l_{w_{t-1}}\right)^{-1}}_{\text{Inverse Hessian}} \boldsymbol{\nabla} l_{w_{t-1}} \quad \text{(Newtons method)}$$

Note (Linear vs Logistic):

	Linear	Logistic
▸ Loss	$\sum_{i=1}^{n} \underbrace{(y_i - \hat{y}_i)^2}_{\text{Square Error}}$	$\sum_{i=1}^{n} \underbrace{\log\left(1 + e^{-\tilde{y}_i \langle w, x\rangle}\right)}_{\text{Cross Entropy}}$
▸ Predict	$\hat{y}_i = \langle \boldsymbol{w}, \boldsymbol{x_i}\rangle$	$\hat{p}_i = \text{sigmoid}(\langle \boldsymbol{w}, \boldsymbol{x_i}\rangle)$
▸ Objective	$\lVert \boldsymbol{y} - \hat{\boldsymbol{y}} \rVert_2^2$	$\text{KL}\left(\frac{1+\tilde{\boldsymbol{y}}}{2} \mid\mid \hat{\boldsymbol{p}}\right)$
▸ Gradient	$\vec{w} - \boldsymbol{\eta} X(\hat{\boldsymbol{y}} - \boldsymbol{y})$	$\vec{w} - \boldsymbol{\eta} X\left(\hat{\boldsymbol{p}} - \frac{1+\tilde{\boldsymbol{y}}}{2}\right)$

Note (Multiclass Logistic Regression):

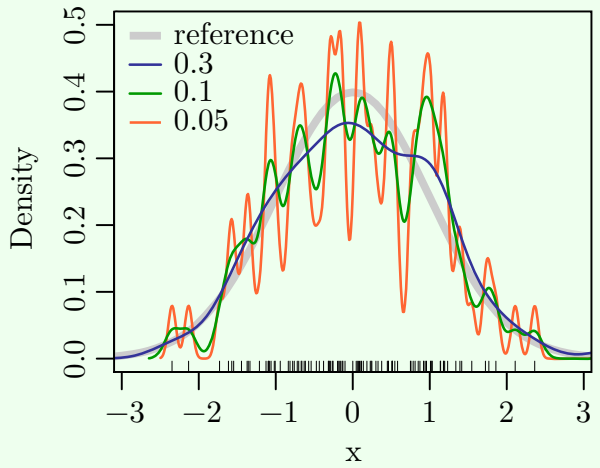
$$\Pr[y = k \mid \boldsymbol{x}, \boldsymbol{w}] = \frac{e^{\langle w_k, \boldsymbol{x}\rangle}}{\sum_{i=1}^{c} e^{\langle w_i, \boldsymbol{x}\rangle}} = \text{softmax}$$

Non-Parametric Methods

Definition (Kerne Density Estimation): Given $\{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^d$ the Parzen estimate for probability at point x_o is

$$\hat{p}(\boldsymbol{x_o}) = \frac{1}{n\lambda} \sum_{i=1}^{n} K_\lambda(\boldsymbol{x_o}, x_i)$$

Where $K_\lambda : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ is the kernel function. λ is a hyperparameter called length scale (or bandwidth). K_λ should be symmetric and shift invariant. $K_{\lambda(x_o, x_i)} = K\left(\frac{x_o}{\lambda} - \frac{x_i}{\lambda}\right)$



KDE of 100 normally distributed numbers.

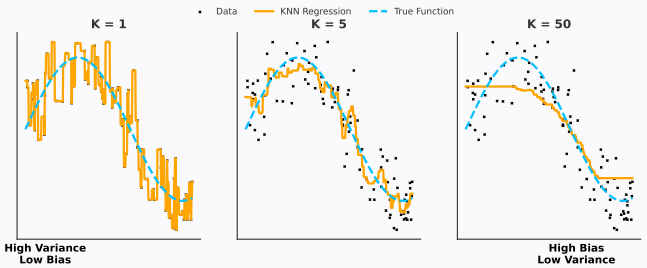
k-Nearest Neighbors

Algorithm (KNN):

```
import numpy as np
def k_nearest_neighbors(data, k, x):
  distances = []
  for xi, yi in data:
    # Euclidean norm
    distance = np.linalg.norm(x - xi)
    distances += [(distance, yi)]
    distances.sort(key=lambda x: x[0])
    k_nearest_labels = distances[:k]
  majority_label = count(k_nearest_labels)
  return majority_label
```

Training: time: $O(1)$, space: $O(nd)$

Testing: time $O(nd + n \log k)$, space: $O(nd)$



KNN with $k = 1, 5, 50$

Note (An upper bound on KNN error rate): Cover-Hart Theorem, for classifier with $k = 1$

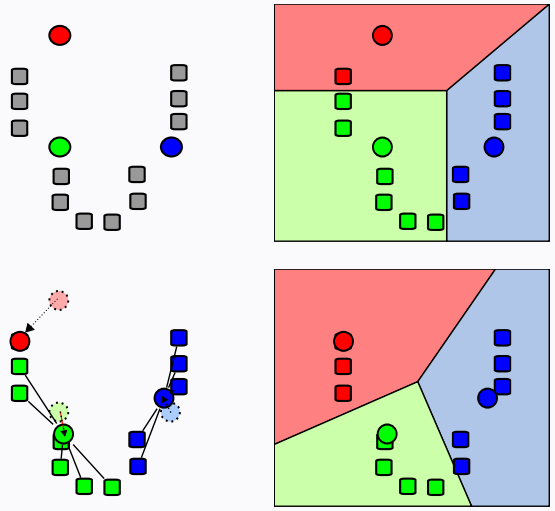
$$\lim_{n\to\infty} \epsilon_{\text{1-NN}} < 2\epsilon_{\text{Bayes}}(1 - \epsilon_{\text{Bayes}})$$

- Number of samples needed to train KNN to reach lower bound on error grows exponentially. Euclidean distance is unhelpful in high dimensions all vectors are almost equidistant

K-Means Clustering

Algorithm (K-Means):

```
def kmeans(X, k, max_iter=300):
  N = len(X)
  C0 = randint(0, high=k, size=N)
  C1 = zeros(N)
  iter = 0
  while iter < max_iter:
    # Centroids of each cluster
    μ = [mean(X[C0==i]) for i in 0..k]
    for i in range(N):
      # Best centrioid for x_i
      best = argmin([norm(X[i] - μ[j])**2
                     for j in 0..k])
      C1[i] = best
      if array_equal(C0, C1):
        # No updates detected
        break
      C0 = C1
      iter += 1
  return C0
```



,

Bayesian Learning

Definition (Prior and Posterior): In a Bayesian approach assume the parameter θ follows a prior pdf $\Pr(\theta)$.

Given a prior pdf $\Pr(\theta)$, after observing some data $\mathcal{D}$ belief on the probable values of θ will have changed. We get the posterior:

$$\Pr[\boldsymbol{\theta}|\mathcal{D}] = \frac{\Pr[\mathcal{D}|\boldsymbol{\theta}]\Pr[\boldsymbol{\theta}]}{\Pr[\mathcal{D}]} = \frac{\Pr[\mathcal{D}|\boldsymbol{\theta}]\Pr[\boldsymbol{\theta}]}{\int \Pr[\mathcal{D}|\boldsymbol{\theta}]\Pr[\boldsymbol{\theta}]d\boldsymbol{\theta}}$$

Note: computing the integral may not be tractable.

Theorem (Bayes classifier): A classification problem with $\vec{x} \in \mathbb{R}^d$ and $Y \in [c] := \{1...c\}$ The optimal (Bayes) classifier is

$$\operatorname*{argmin}_{h:\mathbb{R}^d\to[c]} \Pr(Y \neq h(\vec{x}))$$

$$h^\star(\vec{x}) = \operatorname*{argmax}_{k\in[c]} \Pr(Y = k \mid \vec{x})$$

$$\operatorname*{argmax}_{k\in[c]} \underbrace{\Pr(X = \vec{x} \mid Y = k)}_{\text{likelihood}} \underbrace{\Pr(Y = k)}_{\text{prior}}$$

Definition (Maximum likelihood estimate (MLE)): MLE estimate maximizes the likelihood of observing the data $\mathcal{D}$. This ignores any prior on $\boldsymbol{\theta}$.

$$L(\boldsymbol{\theta}; \mathcal{D}) := \Pr(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{n} \Pr(\boldsymbol{x}_i|\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{\text{MLE}} = \operatorname*{argmax}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} L(\boldsymbol{\theta}; \mathcal{D})$$

$$\boldsymbol{\theta}_{\text{MLE}} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \sum_{i=1}^{n} -\log \Pr(\boldsymbol{x}_i|\boldsymbol{\theta})$$

Definition (Maximum a posteriori (MAP)):

$$\theta_{\text{MAP}} = \underset{\theta\in\Theta}{\text{argmax}}\, \Pr(\theta \mid \mathcal{D})$$

$$\rightarrow \underset{\theta\in\Theta}{\text{argmax}}\ \underbrace{-\log \Pr(\mathcal{D}|\theta)}_{\text{negative log-likelihood}} - \underbrace{\log \Pr(\theta)}_{\text{prior as regularization}}$$

Example (Ridge Regression and MAP): Linear regression with λ is equivalent to MAP estimate of $\boldsymbol{w}$ using a prior on $\boldsymbol{w} \sim \mathcal{N}_d(\mathbf{0}, \Sigma)$ where $\boldsymbol{\Sigma}^{-1} = \lambda I$

Example (Bayesian linear regression):

$\Pr(\boldsymbol{w}) = \mathcal{N}\left(\vec{0}, \boldsymbol{\Sigma}\right)$ (Gaussian prior on the weights)

$\Pr(\epsilon) = \mathcal{N}(0, \sigma^2)$

$\Pr(\boldsymbol{w}|\mathcal{D}) \propto \Pr(\boldsymbol{w}) \Pr(\mathcal{D}|\boldsymbol{w}) \propto \mathcal{N}(\overline{\boldsymbol{w}}, A^{-1})$ (Posterior)

$$A = \frac{1}{\boldsymbol{\sigma}^2} X^\top X + \boldsymbol{\Sigma}^{-1}\ ,\ \ \overline{\boldsymbol{w}} = A^{-1}\frac{1}{\boldsymbol{\sigma}^2} X^T \boldsymbol{y}$$

$$\Pr(y^\star|\ \vec{x}^\star, \mathcal{D}) = \int_{\boldsymbol{w}} \Pr(y^\star|\vec{x}^\star, \boldsymbol{w}) \Pr(w|\mathcal{D}) \quad \text{(Prediction)}$$

$$\Pr(y^\star|\ \vec{x}^\star, \mathcal{D}) = \mathcal{N}\left(\langle \vec{x}^\star, \overline{\boldsymbol{w}}\rangle, \boldsymbol{\sigma}^2 + \langle \vec{x}^\star, A^{-1}\vec{x}^\star\rangle\right)$$

Gaussian Processes

Definition (Gaussian Processes): Function space view $\Pr(f|\mathcal{D})$ posterior over the function f (Infinite dimensional Gaussian Distribution).

$$f(\boldsymbol{x}) \sim \text{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))\forall \boldsymbol{x}', \boldsymbol{x}$$

Gaussian Processes are specified completely by μ, Σ

$$m(x) = \mathbb{E}[f(x)]\ ,$$
$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$$

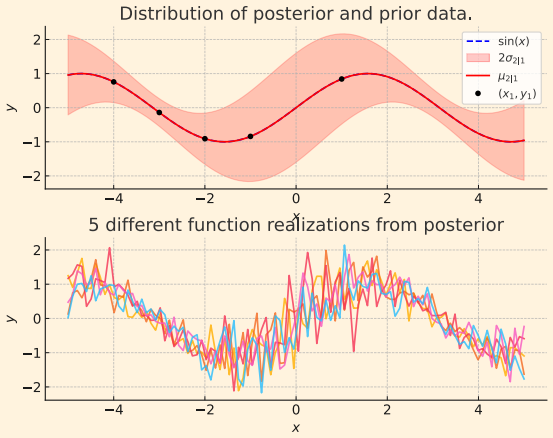
k is the kernel covariance function.

Example (Gaussian Processes): Let $y(\boldsymbol{x}) = \boldsymbol{w}^\top \phi(x)$, consider a prior on $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \alpha^{-1}\boldsymbol{I})$

$$\boldsymbol{y} = \Phi \boldsymbol{w}$$
$$\mathbb{E}[\boldsymbol{y}] = \Phi E[\boldsymbol{w}] = \mathbf{0}$$
$$\text{Cov}[\boldsymbol{y}] = \mathbb{E}[\boldsymbol{y}^\top \boldsymbol{y}] = \frac{1}{\alpha}\Phi\Phi^\top = \boldsymbol{K}$$

Where $\boldsymbol{K}$ is the Gram matrix with

$$\boldsymbol{K}_{ij} = k(x_i, x_j) = \tfrac{1}{\alpha}\phi(x_i)^\top \phi(x_j)$$



Note (Gaussian processes for regression):

$$\underbrace{t_n}_{\text{True Value}} = \underbrace{y_n}_{\boldsymbol{y}(\vec{x}_n)} + \underbrace{\epsilon_n}_{\text{Noise}}$$

Assuming the noise process is Gaussian

$$\Pr[\boldsymbol{t}|\boldsymbol{y}] = \mathcal{N}(\boldsymbol{t}|\boldsymbol{y}, \beta^{-1}\boldsymbol{I}_N)$$

By definition of the gaussian process marginal pdf of $\boldsymbol{y}$ is defined by a Gram matrix $\boldsymbol{K}$

$$\Pr[\boldsymbol{y}] = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \alpha^{-1}\boldsymbol{K})$$

$$\Pr[\boldsymbol{t}] = \int \Pr(\boldsymbol{t}|\boldsymbol{y}) \Pr[\boldsymbol{y}] \text{dy} = N(\boldsymbol{t}|\mathbf{0}, \boldsymbol{C})$$

$$\boldsymbol{C}(\boldsymbol{x}_n, \boldsymbol{x}_m) = \alpha^{-1}k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1} \times \mathbb{1}[n = m]$$

For prediction we augment the covariance matrix

$$\Pr[\boldsymbol{t}_{N+1}] = \mathcal{N}(\boldsymbol{t}_{N+1} \mid \mathbf{0}, \boldsymbol{C}_{N+1})$$

$$\boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_n & \boldsymbol{k} \\ \boldsymbol{k}^\top & c \end{pmatrix}\ ,\ \ \boldsymbol{k} \in \mathbb{R}^N\ ,\ \ \boldsymbol{k}_i = \alpha^{-1}k(x_{N+1}, x_i)$$

$$c = \alpha^{-1}k(x_{N+1}, x_{N+1}) + \beta^{-1}$$

Then the predictive distribution is

$$\Pr[t_{N+1}|\boldsymbol{t}] = \mathcal{N}(\mu(\boldsymbol{x}_{N+1}), \sigma^2(x_{N+1}))$$
$$\mu(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^\top \underbrace{\boldsymbol{C}_N^{-1}\boldsymbol{t}}_{\vec{\boldsymbol{a}}}\ ,\ \ \sigma^2(\boldsymbol{x}_{\boldsymbol{N+1}}) = c - \boldsymbol{k}^\top \boldsymbol{C}_N^{-1}\boldsymbol{k}$$

$$\mu(\boldsymbol{x}_{N+1}) = \langle \boldsymbol{k}, \vec{\boldsymbol{a}}\rangle = \sum_{i=1}^{N} k(\boldsymbol{x}_i, \boldsymbol{x}_{N+1})a_i$$

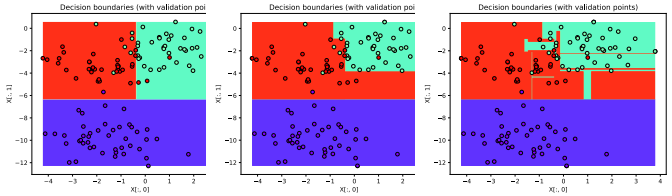
Note (Gaussian Process Time Complexity):

Type	Fixed Basis Functions m Features	Gaussian Process N Data Points
Train	$O(M^3)$	$O(N^3)$
Test	$O(M^2)$	$O(N^2)$

Decision Trees

Definition (Decision Trees Growing): Growing a tree means we choose a dimension or feature $j^\star$ and a threshold $t^\star$ to split that minimized the loss ℓ.

$$L_{j,t} = \{(\boldsymbol{x}_i, y_i) \in S : \boldsymbol{x}_{ij} \leq t\}$$
$$R_{j,t} = \{(\boldsymbol{x}_i, y_i) \in S : \boldsymbol{x}_{ij} > t\},$$
$$(j^\star, t^\star) = \underset{j=1..d}{\operatorname{argmin}} \min_{t \in T_j} |L_{j,t}| \ell(L_{j,t}) + |R_{j,t}| \ell(R_{j,t})$$



Decision with increasing depths, $k = 2, 4, 6$

Note (Decision Trees Classification Cost): For a region $\mathcal{D}$ we define

$$\hat{p}_k = \frac{1}{|\mathcal{D}|} \sum_{y_i \in \mathcal{D}} \mathbb{1}(y_i \in k) \text{ (Empirical fraction of class } k)$$

$$\hat{y}_i = \underset{k=1..c}{\operatorname{argmax}} \hat{p}_k \quad \text{(Majority class in } \mathcal{D})$$

$$\text{Loss}(\mathcal{D}) = 1 - \max_k \hat{p}_k = 1 - \hat{p}_{\hat{y}} \text{ (Misclassification error)}$$

$$\text{Loss}(\mathcal{D}) = \sum_{k=1}^{c} \hat{p}_k (1 - \hat{p}_k) \quad \text{(Gini index)}$$

$$\text{Loss}(\mathcal{D}) = -\sum_{k=1}^{c} \hat{p}_k \log(\hat{p}_k) \quad \text{(Entropy)}$$

Note (Minimal Cost-Complexity Pruning): Grow a tree fully first then regularize using a hyperparameter α.

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}|$$

Where $|\tilde{T}|$ is the number of terminal nodes in T and $R(T)$ is the total loss for the leaves. Penalizes deep trees/more leaves.

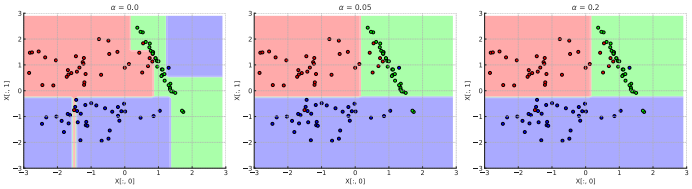


Figure 1: Decision Tree Pruning