Introduction to Machine Learning Summary

Supervised Learning

Regression

Dataset: $D=\{(\vec{x}_1,y_1)...(\vec{x}_n,y_n)\} \rightarrow X=[\vec{x}_1^T...\vec{x}_n^T]^T \in \mathbb{R}^{n\times d}$, $\vec{y}=[y_1...y_n]^T \in \mathbb{R}^n$

Empirical Risk: $\hat{R}_D(f)=\frac{1}{n}\sum_{i=1}^n L(y_i,f(\vec{x}_i))$ → $\hat{f}(\vec{x})=\text{argmin}_f \hat{R}_D(f)$

Population Risk: $R(f)=\mathbb{E}_{(\vec{x},y)\sim P(\vec{x},y)}[L(y,f(\vec{x}))]$ → f^* minimizer of $R(f)$ (Generalization Error/True Risk)

Least Squares Regression:

Objective: $\hat{w}=\text{argmin}_{\vec{w}} \sum_{i=1}^n (y_i-\vec{w}^T\vec{x}_i)^2 = \text{argmin}_{\vec{w}} \underbrace{\|\vec{y}-X\vec{w}\|_2^2}_{\hat{R}(\vec{w})/L(\vec{w})}$

CF: $n>d$ → one best solution → $\hat{\vec{w}}=(X^TX)^{-1}X^T\vec{y}$

Samples → $n<d$ → many exact solutions → $\hat{\vec{w}}=X^T(XX^T)^{-1}\vec{y}$ (Features)

Ridge Regression: Uses regularization to encourage small weights

Objective: $\hat{w}=\text{argmin}_{\vec{w}} \|\vec{y}-X\vec{w}\|_2^2+\lambda\|\vec{w}\|_2^2$ ← L_2 Regularization

Closed form solution: $\hat{\vec{w}}=(X^TX+\lambda I)^{-1}X^T\vec{y}$

Lasso Regression: Uses regularization to encourage some $w_i=0$

Objective: $\hat{w}=\text{argmin}_{\vec{w}} \|\vec{y}-X\vec{w}\|_2^2+\lambda\|\vec{w}\|_1$ ← L_1 regularization

No CF in general, intrinsic/embedded method for feature selection

Bias-Variance Tradeoff

$\lambda \rightarrow 0$: Variance ↑, Bias ↓

$\lambda \rightarrow \infty$: Variance ↓, Bias ↑

Model Selection & Validation

True Risk $R(\vec{w})$ would lead to ideal weights $\vec{w}^*=\text{argmin}_{\vec{w}} R(\vec{w})$, but is unknown → estimation over empirical Risk $\hat{R}(\vec{w})$ based on D

Empirical Risk Minimization:

$\hat{f}(x)=\text{argmin}_f \hat{R}_D(f)$

At $\hat{f}$ the empirical Risk $\hat{R}(\hat{f})$ systematically underestimates the population Risk: $\mathbb{E}[\hat{R}_D(\hat{f})] \leq \mathbb{E}[R(\hat{f})]=R(\hat{f})$

For $n\rightarrow\infty$: $\hat{R}_D(f) \rightarrow R(f)$ not

→ on validation set: $\mathbb{E}_D[\hat{R}_D(\hat{f})]=R_D(f)$

Cross Validation: (avoids Bias with independent validation sets)

K-Fold CV: D | D_1 | D_2 | ... | ... | ... | ... | D_k |

1. Training set is k-1 Folds, Validation Set 1 Fold
2. Validation set gets varied and then the prediction error is the average of validation errors

⇒ smaller bias $R(\hat{f}_{D\setminus i})-\hat{R}(\hat{f}_{D\setminus i})$

bigger variance in $|\hat{R}_{D\setminus i}(\hat{f}_{D\setminus i})-R(\hat{f}_{D\setminus i})|$

smaller variance in estimating avg. population Risk $\mathbb{E}_D[R(\hat{f}_D)]$

Optimization

Gradient Descent: (minimizes objective function $\hat{R}(\vec{w})$)

1. Initialization of random $\vec{w}_0 \in \mathbb{R}$
2. Update $\vec{w}^{(t+1)}=\vec{w}^{(t)}-\eta_t\cdot\nabla_w L(\vec{w}^{(t)})$

- Convergence: $\|\vec{w}_t-\hat{\vec{w}}\|_2 \leq \underbrace{\|I-\eta X^TX\|_{op}^t}_{<1 \text{ for stability}} \cdot \|\vec{w}_0-\hat{\vec{w}}\|$ (p: $\|\cdot\|_{op}$ = max Eigenvalue/Singular)

→ Exponential/Linear Convergence if $p<1$, $p=\frac{\kappa-1}{\kappa+1}$

- Iterations for ϵ-close to $\hat{w}$: $T \geq \frac{\log(c/\epsilon)}{\log(1/p)}$
- Optimal Stepsize $\eta_{opt}=\frac{2}{\lambda_{min}(X^TX)+\lambda_{max}(X^TX)}$, $\kappa=\frac{\lambda_{max}(X^TX)}{\lambda_{min}(X^TX)}$

↳ $T \geq O(\kappa\log(\frac{1}{\epsilon}))$ → Large κ → low convergence

GD is less computationally complex than CF for big problems

Minibatch GD: (GD with subset of samplex S)

1. Initialization of random $\vec{w}_0 \in \mathbb{R}^d$
2. Update $w^{(t+1)}=w^{(t)}-\eta_t\frac{1}{|S|}\sum_{i\in S}\nabla_w L(y_i,w^Tx_i)$ (Batchsize)

Stochastic Gradient Descent: for $|S|=1$

→ only one point $(\vec{x},y)\in D$ is chosen randomly

→ converges if $\sum \eta_t=\infty$ & $\sum \eta_t^2<\infty$ (eg: $\eta_t=\frac{1}{t}$)

→ Batchsize ↓ → Variance ↑, memory ↓, p(escaping saddlepoint) ↑

SGD with momentum: avoids being stuck in local optimum & oscillations

→ $v^{(t)}=m\cdot v^{(t-1)}+\eta_t\nabla L(w^{(t)})$, $w^{(t+1)}=w^{(t)}-v^{(t)}$

Classification

Binary Classification: (minimizing number of mistakes)

Labelling: $Y=\{+1,-1\}/\{+,-\}$ with $\hat{y}=\text{sign}(\hat{f}(x))$

Objective: $\hat{w}=\text{argmin}_w \frac{1}{n}\sum_{i=1}^n L_x(y_i,\vec{x}_i,\vec{w})$ $R(f)=P[y\neq\text{sign}(\hat{f}(x))]$

Loss Functions:

- 0-1: $L_{01}=\mathbb{1}[y_i\neq\text{sign}(\vec{w}^T\vec{x}_i)]$ (no optimization via subgradient)
- Perceptron: $L_P=\max\{0,-y_i\vec{w}^T\vec{x}_i\}$
- Hinge: $L_H=\max\{0,1-y_i\vec{w}^T\vec{x}_i\}$
- Logistic: $L_{Log}=\log(1+\exp(-y_i\vec{w}^T\vec{x}_i))$ (Close to Less points)
- Exponential: $L_{exp}=\exp(-y_i\vec{w}^T\vec{x}_i)$

Logistic Regression: (uses logistic loss)

Objective: $\hat{\vec{w}}=\text{argmin}_{\vec{w}} \frac{1}{n}\sum_{i=1}^n \log(1+\exp(-y_i\vec{w}^T\vec{x}_i))$

GD: $w_{t+1}=w_t+\eta_t\frac{1}{n}\sum_{i=1}^n \frac{y_i\vec{x}_i}{1+\exp(y_i\vec{w}^T\vec{x}_i)}$

Support Vector Machine: (uses hinge loss to encourage margin)

Objective: $\hat{\vec{w}}=\text{argmin}_{\vec{w}} \frac{1}{n}\sum_{i=1}^n \max\{0,1-y_i\vec{w}^T\vec{x}_i\}+\lambda\|\vec{w}\|_2^2$

Margin(w) $=\min_i y_i\vec{w}^T\vec{x}_i$ → min distance $=\frac{\text{margin}(w)}{\|\vec{w}\|_2}$

- Hard Margin SVM: $\min_{\vec{w}} \|\vec{w}\|_2$ s.t. $y_i\vec{w}^T\vec{x}_i\geq 1$ $\forall i\in\{1,2,...,n\}$

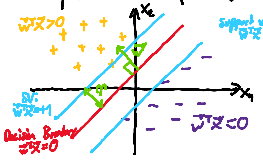
→ for linear seperable Data

- Soft Margin SVM: $\min_{\vec{w}} \frac{1}{2}\|\vec{w}\|_2^2+C\cdot\sum_{i=1}^n \max\{0,1-y_i\vec{w}^T\vec{x}_i\}$ (ξ_i)

→ for not linear seperable Data

→ $\xi=0$ were correctly, ξ = positiv were false side margin

max margin $\gamma=\frac{1}{\|\vec{w}\|}$



Multiclass Classification:

Labels: $Y=\{1,...,K\}$

One & All: 1. Train K binary classifiers seperately → Current Labels → +1, Rest → -1

2. Apply softmax $\Rightarrow \sigma(f(x))_k=\frac{e^{\vec{w}_k^T\vec{x}}}{\sum_{i=1}^n e^{\vec{w}_i^T\vec{x}}}$
3. For test point: $\hat{y}=\text{argmax}_k \sigma(f(x))_k=\text{argmax}_k \hat{f}_k(x)$

Decision boundary for f_i & f_j is $f_i=f_j$ linear slope

Hypothesis Testing / Other Metrics:

Pick class as Hypothesis ($y=+$) if critical to predict correctly when true

- reject $H_0(x) \Leftrightarrow \hat{y}=-1$ if $\hat{f}(x)>J$
- accept $H_0(x) \Leftrightarrow \hat{y}=+$ if $\hat{f}(x)<J$ (used to control significance Level, FPR)

Accuracy $=\frac{TP+TN}{n}=P(\hat{y}=y)$

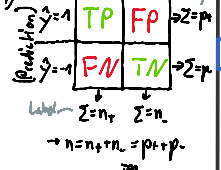
Recall/TPR $=\frac{TP}{TP+FN}=\frac{TP}{n^+}=P(\hat{y}=1|y=1)$

FDR $=1-\text{Precision}=\frac{FP}{TP+FP}=P(y=-1|\hat{y}=1)$

Precision: $\frac{TP}{TP+FP}=P(y=1|\hat{y}=1)$

FPR $=\frac{FP}{TN+FP}=\frac{FP}{n^-}=P(\hat{y}=1|y=-1)$

F1 score $=\frac{2}{1/\text{Recall}+1/\text{Precision}}=\frac{2\,TP}{2\,TP+FP+FN}$

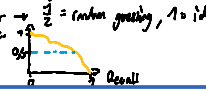


ROC curve: varying J for fixed $\hat{f}$ → plot TPR vs FPR

→ J small → TPR↑, FPR↑ vs J large → TPR↓, FPR↓

→ Area under curve: indicator → $\frac{1}{2}$ = random guessing, 1 = ideal

Precision vs. Recall curve:



Non Parametric Methods

k-Nearest Neighbors: (no training, inference during test time → $O(nd)$)

1. Given dataset D, pick k and distance metric d
2. For a given $\vec{x}$, find k closest $\vec{x}_{i_1},...,\vec{x}_{i_k}$ in D
3. Output majority vote/average of $y_{i_1},...,y_{i_k}$

→ sensitive to k, pick k using CV, erratic in high dim., bagging needed

Decision Tree:

1. Training Set D → each node v contains $D(v)\subseteq D$
2. Present node split $D(v)$ with chosen rule $(>,<)$
3. repeat until impurity $I(S)$ is low

Kernels

Feature expansion needs to be avoided → optimal hyperplane lies in span of data → $\vec{w}$ has fixed structure

Feature expansion: $\vec{x}\in\mathbb{R}^d$, $\phi(\vec{x})\in\mathbb{R}^p \rightarrow p=\binom{d+m}{m}\approx O(d^m)$ (d=variables, m=maximal degree)

d=2, m=3 → $(1,x_1,x_2,x_1x_2)$

Kernel Trick:

$\hat{\vec{w}}=\sum_{i=1}^n y_i\alpha_i\vec{x}_i \overset{\text{Kernel Trick}}{=} \sum_{i=1}^n \alpha_i\phi(\vec{x}_i)=\Phi^T\vec{\alpha}$ with $\Phi=\begin{bmatrix}-\phi(x_1)^T-\\ \vdots\\ -\phi(x_n)^T-\end{bmatrix}$

Kernelization: $X\rightarrow\Phi$, $\vec{x}_i\rightarrow\vec{\phi}(\vec{x}_i)$

Kernel Function: $k(\vec{x},\vec{x}')=\vec{\phi}(\vec{x})^T\vec{\phi}(\vec{x}')=\langle\vec{\phi}(\vec{x}),\vec{\phi}(\vec{x}')\rangle$ (non-linear feat. transform)

Kernel/Gram Matrix: $K=\Phi\Phi^T=\begin{bmatrix}k(x_1,x_1)&\cdots&k(x_1,x_n)\\ \vdots&&\vdots\\ k(x_n,x_1)&\cdots&k(x_n,x_n)\end{bmatrix}$

Properties:

- Symmetry: $k(\vec{x},\vec{x}')\overset{!}{=}k(\vec{x}',\vec{x})$
- K is positive semidefinit → Eigenvalues ≥ 0

Composition of Kernel Functions:

- $k(\vec{x},\vec{x}')=k_1(\vec{x},\vec{x}')+k_2(\vec{x},\vec{x}')$
- $k(\vec{x},\vec{x}')=k_1(\vec{x},\vec{x}')\cdot k_2(\vec{x},\vec{x}')$
- $k(\vec{x},\vec{x}')=c\cdot k_1(\vec{x},\vec{x}')$ for $c>0$
- $k\left(\binom{x}{y},\binom{x'}{y'}\right)=k_1(x,x')k_2(y,y')$
- $k(\vec{x},\vec{x}')=f(k_1(\vec{x},\vec{x}'))$ f polynomial with pos. coeff. or exp
- $k(\vec{z},\vec{z}')=k_3(v(z),v(z'))$ $v:Z\rightarrow X$ $k\left(\binom{x}{y},\binom{x'}{y'}\right)=k_1(x,x')+k_2(y,y')$

Important Kernel Functions:

- Constant: $k(\vec{x},\vec{x}')=c$ $c\geq 0$
- Laplacian: $k(\vec{x},\vec{x})=\exp(-\frac{\|\vec{x}-\vec{x}'\|_1}{h})$
- Linear: $k(\vec{x},\vec{x}')=\vec{x}^T\vec{x}'$
- Gaussian/RBF: $k(\vec{x},\vec{x}')=\exp(\frac{\|\vec{x}-\vec{x}'\|_2^2}{2h^2})$ ← infinite dimensional
- Polynomial (Deg. m): $k(\vec{x},\vec{x}')=(\vec{x}^T\vec{x}')^m$ ← $\binom{d+m}{m}=O(d^m)$ for Deg. m monomials
- Polynomial (Up to Deg. m): $k(\vec{x},\vec{x}')=(1+\vec{x}^T\vec{x}')^m$

Neural Networks

Parametrize feature maps ϕ & optimize over parameters

→ results in a non-convex function for multilayer networks

Activation Functions: (→ map the Data (non-linearly))

- Linear: $\varphi(z)=z$, $\varphi(z)'=1$ (Identity)
- Sigmoid: $\varphi(z)=\frac{1}{1+\exp(-z)}$, $\varphi(z)'=\varphi(z)\cdot(1-\varphi(z))$
- Tanh: $\varphi(z)=\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$, $\varphi(z)'=1-\varphi(z)^2$ (soft edges)
- ReLU: $\varphi(z)=\max\{0,z\}$, $\varphi(z)'=\begin{cases}1 & z\geq 0\\ 0 & z\leq 0\end{cases}$ (more edges)

Forward Propagation: (predict)

- Input Layer 0: $\vec{v}^{(0)}=[\vec{x};1]$ initialize with x values or 1
- Hidden Layer l: $\vec{z}^{(l)}=W^{(l)}\vec{v}^{(l-1)}$ sum weighted outputs of l-1

$\vec{v}^{(l)}=[\varphi(\vec{z}^{(l)});1]$ output sum with act. function

- Output Layer L: $\vec{f}=W^{(L)}\vec{v}^{(L-1)}$ sum weighted outputs of L-1

→ Predict $y_i=f_i$ for regression & $y=\text{argmax}_i f_i$ for classification

Backpropagation: (adjust weights)

- Output Layer: Error: $\delta^{(L)}=\nabla_f l(f,y)=[l'(f_1,y_1),...,l'(f_p,y_p)]$

Gradient: $\nabla_W L(y,x;W)=\delta^{(L)}v^{(l-1)T}$

- Hidden Layer: Error: $\delta^{(l)}=(\frac{dl}{d\vec{z}^{(l)}})^T=\varphi'(\vec{z}^{(l)})\odot(W^{(l+1)T}\delta^{(l+1)})$ (pointwise multiplication)

Gradient: $\nabla_W^{(l)}L(y,x;W)=\delta^{(l)}v^{(l-1)T}$

→ Take derivative of loss function up to weight $\frac{df}{dx}g(f(x))=\frac{dg}{df}\cdot\frac{df}{dx}$

Exploding/Disappearing Gradient, Weight Initialization:

Exploding Gradient: $\|\nabla_W^{(l)}L\|\rightarrow\infty$, Vanishing Gradient: $\|\nabla_W^{(l)}L\|\rightarrow 0$

By using ReLU and keeping the variance of the weights a constant across layers, vanishing/exploding gradients can be avoided

For a good weight initialization we need to set $\sigma^2=\frac{2}{d}$

→ Glorot (tanh) → $w_{ij}\sim N(0,\frac{1}{n_{in}})/N(0,\frac{2}{n_{in}+n_{out}})$, He (ReLU) → $w_{ij}\sim N(0,\frac{2}{n_{in}})$

n_{in} = number of units in previous layer, n_{out} = number of units in next layer

Regularization: (avoid overfitting)

- Add penalty term: + $\lambda \|w\|_2^2$
- Early Stopping: Stop training with GD before convergence
- Dropout: randomly dropout units for each iteration with prob. = p
- Batch Normalization: Normalization of inputs to $\mathcal{N}(0,1)$ with
  → $\mu_S = \frac{1}{|S|}\sum_{i\in S} x_i$ $\sigma^2 = \frac{1}{|S|}\sum_{i\in S}(x_i-\mu_S)^2$ → $\tilde{x} = \gamma \frac{x_i-\mu_S}{\sqrt{\sigma^2+\varepsilon}} + \beta$ (Shift, Scale, Normalize, Uncertain)

Convolutional Neural Networks: ($m \cdot f^2$ Parameters)

- Stride s: Number of Pixels Filter moves
- Padding p: Number of "0" Pixels added to side of image
- Filter & Kernel: Filter is a concatenation of Kernels ($dim(F) = dim(K)+1$)
- Output Dimension: Applying m different f×f filters to an n×n image results in an Output with $L = \frac{n+2p-f}{s}+1$
- Pooling: Reduces parameters by only considering average/maximum of multiple units

Unsupervised Learning

Clustering

k-Means Clustering: (find centres that minimize avg. squared distances)

Objective: $\vec{\mu} = \arg\min_{\vec{\mu}} \hat{R}(\vec{\mu}) = \arg\min_{\vec{\mu}} \sum_{i=1}^{n} \min_{j\in\{1,\dots,k\}} \|\vec{x}_i - \vec{\mu}_j\|_2^2$ (non-convex)

↳ Lloyd's Heuristic:

1. Initialize cluster centres $\vec{\mu}^{(0)} = [\vec{\mu}_1^{(0)}, \dots, \vec{\mu}_k^{(0)}]$
2. While not converged: (as more changes)
   i) Assign points to closest centre: $z_i = \arg\min_{j\in\{1,\dots,k\}} \|x - \vec{\mu}_j^{(t-1)}\|_2$
   ii) Update centres: $\vec{\mu}_j^{(t)} = \frac{1}{n_j}\sum_{i:z_i=j} x_i$ (n_j: point i to member of centre j)

→ converges to local optimum with cost per iteration = $O(nkd)$

k-Means ++: (initialization for k-Means)

1. Start with random data point as first cluster centre
2. Add 2-k centres randomly proportional to squared distance to closest centre = $\frac{d^2}{\sum_{\vec{x}} d^2}$ $d = \min_{j\in\{1,\dots,k\}} \|x_i - \vec{\mu}_j^{(t)}\|_2^2$

→ To find the optimal number of clusters find the "elbow" in $\hat{R}$ vs. k plot or add regularizer ($+\lambda k$) (optimal Solution)

Expected Loss of k-Means ++: $O(\log(k)) \cdot \min_{\vec{\mu}} \hat{R}(\vec{\mu})$

Dimension Reduction - Principal Component Analysis (PCA)

PCA: (lower dimensional representation in direction with highest variance)

Given: • Centered Data with $\mu_X = \sum_{i=1}^n x_i = 0$
• Empirical Covariance Matrix $\Sigma = \frac{1}{n}\sum_{i=1}^n \vec{x}_i\vec{x}_i^T = \frac{1}{n}XX^T = \frac{1}{n}\begin{bmatrix}\vec{x}_1 & \dots & \vec{x}_n\end{bmatrix}\begin{bmatrix}\vec{x}_1^T \\ \vdots \\ \vec{x}_n^T\end{bmatrix}$

Objective: $\hat{W} = \arg\max_{W^TW=I} \sum_{i=1}^n \|Wx_i - x_i\|_2^2$

Solution: k principle eigenvectors of Σ → $\hat{W} = [\vec{v}_1 | \dots | \vec{v}_k] \in \mathbb{R}^{d\times k}$ with $z_i = W^Tx_i$, $\Sigma = \sum_{i=k+1}^d \lambda_i \vec{v}_i\vec{v}_i^T$

→ minimizes Euclidean rec. error, perfect reconstruction for k>n & k=d

Kernel PCA: (can be used to discover non-linear feature maps k(x))

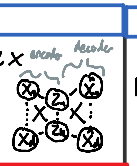
Objective: $\vec{\alpha} = \arg\max_{\alpha^TK\alpha=I} \alpha^TK^TK\alpha$

Solution: k principle eigenvectors of K → $\vec{\alpha} = [\vec{\alpha}^{(1)}|\dots|\vec{\alpha}^{(k)}]$ with $\vec{\alpha}^{(i)} = \frac{\vec{v}_i}{\sqrt{\lambda_i}}$ and $K = \sum_{i=1}^n \lambda_i \vec{v}_i\vec{v}_i^T = \Phi\Phi^T$

Projection: $z_i = W^{(k)T}x = \sum_{j=1}^n \alpha_j^{(i)} k(x_j, x)$

Autoencoder

Objective: Learn $f(x;W) = f_{dec}(f_{enc}(x;W_{enc});W_{dec}) \approx x$ (encoder, decoder)
with $\hat{W} = \arg\min_W \sum_{i=1}^n \|x_i - f(x_i;W)\|_2^2$



→ Neural Network with k<d hidden layer units
→ equivalent to PCA if activation function is identity

Probabilistic Modelling

Probability Formulas

Exp. Value: $\mathbb{E}[X] = \sum_x x \cdot P(x)$, Variance: $Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Bayes Rule: $P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)} = \frac{P(B|A)\cdot P(A)}{\sum_A P(B|A)\cdot P(A)} = \frac{P(A,B)}{P(B)}$

MLE: $\hat{\theta}_{MLE} = \arg\max_\theta P(D;\theta) = \arg\max_\theta \prod_{i=1}^n P(x_i;\theta) = \arg\min_\theta -\sum_{i=1}^n \log(P(x_i;\theta))$

MAP: $\hat{\theta}_{MAP} = \arg\max_\theta P(\theta|D) = \arg\max_\theta P(D|\theta)\cdot P(\theta) = \arg\min_\theta -\sum_{i=1}^n \log(P(x_i;\theta)) + \log(P(\theta))$ (i.i.d.)

Normal Distribution $\mathcal{N}(x;\mu,\Sigma=\sigma^2) \to P(X=x_i) = \frac{1}{\sqrt{2\pi|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$

Bernoulli Distribution $Be(p) \to P(X=x_i) = p^{x_i}\cdot(1-p)^{1-x_i}$, $x_i \in \{0,1\}$

Binomial Distribution $Bin(n,p) \to P(X=x_i) = \binom{n}{x_i} p^{x_i}(1-p)^{n-x_i}$, $x_i \in \{0,1,\dots,n\}$

Poisson Distribution $Po(\lambda) \to P(X=x_i) = \lambda^{x_i}/x_i! \cdot e^{-\lambda}$, $x_i \in \{0,1,\dots,\infty\}$

Decision Theory

Decision Rule:
A decision rule $f: X \to A$ takes an observable input $x \in X$ and outputs an action $f(x)$ in some actionspace A which causes cost/loss $L(f(x),y)$
→ optimal decision rule f^* minimizes $\mathbb{E}[L(f(x),y)|x]$ and hence $R(f)$
↳ optimal action: $\alpha^*(x) = \arg\min_{\alpha\in A} \mathbb{E}_{y|x}[L(\alpha,y)|x]$ ← need $P(y|x)$

Asymmetric Losses: (underestimation, overestimation)
Regression: $L(a,y) = J\cdot\max(y-a,0) + (1-J)\max(a-y,0) = (y-a)(J - \mathbb{1}_{\{y-a<0\}})$
0-1 Loss: $L(a,y) = C_{FP}\mathbb{1}_{\{a=1,y=0\}} + C_{FN}\mathbb{1}_{\{a=0,y=1\}} + C\,\mathbb{1}_{\{a=y\}}$ (abstaining, from post.)

Discriminative & Generative Models

Discriminative Models: Y is an uncertain consequence of X → $P(y|\vec{x})$
- Food ingredients (X) → Healthy? (Y)
- Model: $P(x,y;\theta) = P(y|x;\theta)\cdot P(x)$
- Regression or Classification

Generative Models: X is an uncertain consequence of Y → $P(\vec{x},y)$
- Disease (Y) → Symptoms (X)
- Model: $P(x,y;\Theta) = P(x|y;\eta)\cdot P(y;\vec{\pi})$ $\Theta = (\eta,\vec{\pi})$
- More common for classification (harder problem)

Gaussian Bayes Classifier (GBC)

Class Prior: $Y \sim Cat(\vec{\pi})$, $P(Y=y;\vec{\pi}) = p_y = \pi_y = \frac{Count(Y=y)}{Count(Y=\cdot)}$
Feature Distribution: $X|Y \sim \mathcal{N}(x;\mu_y,\Sigma_y) = \frac{Count(X=x,Y=y)}{Count(Y=y)}$
MLE's: • $\hat{\pi}_y = \hat{p}_y = \frac{Count(Y=y)}{n}$ $\hat{\Sigma}_{i:Y=y} = \frac{\Sigma_{y=diag(\sigma_y^2)}}{\text{Gaussian Naive Bayes}}$
• $\hat{\mu}_y = \frac{1}{Count(Y=y)}\hat{\Sigma}_{i:Y=y} x_i$ $\hat{\mu}_{y,k} = \frac{1}{Count(Y=y)}\hat{\Sigma}_{i:Y=y} x_{i,k}$
• $\hat{\Sigma}_y = \frac{1}{Count(Y=y)}\hat{\Sigma}_{i:Y=y}(x_i-\hat{\mu}_y)(x_i-\hat{\mu}_y)^T$ $\hat{\sigma}^2_{y,k} = \frac{1}{Count(Y=y)}\hat{\Sigma}_{i:Y=y}(x_{i,k}-\hat{\mu}_{y,k})^2$ (Variance)

Gaussian Naive Bayes: features independent → $\Sigma_y = diag(\sigma_{1,y}^2,\dots)$
Quadratic Discriminant Analysis: π_y is uniform
→ Linear Discriminant Analysis: $\Sigma_y = \Sigma$ for all y
→ Fisher LDA: $c=2$, $p_1 = p_2 = 1/2$, $\Sigma_1 = \Sigma_2$

Frequentists Inference

Bootstrap: (No assumptions on model needed → estimate dist. via sample)
Estimate $R(f_\theta) \approx \mathbb{E}_{\hat{P}}[R(f_\theta)]$ with $\hat{P}_n(x,y) = \frac{1}{n}\sum_{i=1}^n \mathbb{I}_{(x,y)=(x_i,y_i)}$
Principle: 1. Sample $D = \{x_i\}_{i=1}^n$ i.i.d. from $P(x;\theta)$ uniformly
2. Bootstrap: For b = 1,...,B D_b consists of uniform samples x_i^b with replacement from D
3. Compute $\hat{\theta}_b = \frac{1}{n}\sum_{i=1}^n x_i^b$ for every D_b → Distribution for $\hat{\theta}$

Jackknife: Instead of sampling D_b with replacement, use K-fold CV

Gaussian Mixture Models (GMM)

Combination of Gaussians: $P(\vec{x}|\theta) = \sum_{i=1}^k w_i \mathcal{N}(\vec{x};\vec{\mu}_i,\Sigma_i)$ with $w_i \geq 0$, $\sum w_i = 1$
↳ $(\mu^*,\Sigma^*,w^*) = \arg\min -\log(P(x|\theta)) = \arg\min -\sum_{i=1}^n \log(\sum_{j=1}^k w_j \mathcal{N}(x_i|\mu_j,\Sigma_j))$ (non-convex)

$P(j|x) \to$ GBC classifier

Hard EM: assign fixed labels

E-Step: Predict most likely class for each data point (Class Assignment)
→ $z_i^{(t)} = \arg\max_z P(z|\vec{x}_i;\theta^{(t-1)}) = \arg\max_z P(z|\theta^{(t-1)})P(\vec{x}_i|z,\theta^{(t-1)})$ (w_z, $\mathcal{N}(\vec{x}_i|\mu_z^{(t-1)},\Sigma_z^{(t-1)})$)

M-Step: MLE for now complete Data $D^{(t)} = \{(\vec{x}_1,z_1^{(t)}),\dots,(\vec{x}_n,z_n^{(t)})\}$

→ $\Theta^{(t)} = \arg\max_\theta P(D^{(t)}|\theta)$ $\hat{\mu}_j$ & $\hat{\Sigma}_j$ as with GBC

→ problematic if 2 clusters of same class overlapping

Soft EM: assign probabilities

E-Step: Calculate Responsibilities (Datapoints with only Labels)
$Q(\theta,\theta^{(t-1)}) = \mathbb{E}_{z_i\sim n}[\log(P(x_{i=n},z_{i=n}|\theta))|x_{i=n},\theta^{(t-1)}]$

→ $\gamma_j^{(t)}(\vec{x}_i) = P(z_i=j|\vec{x}_i,\theta^{(t-1)}) = \frac{w_j P(\vec{x}_i|\theta_j^{(t)})}{\sum_{q=1}^k w_q P(\vec{x}_i|\theta_q)} = \frac{w_j\mathcal{N}(\vec{x}_i|\mu_j,\Sigma_j)}{\sum_{q=1}^k w_q\mathcal{N}(\vec{x}_i|\mu_q,\Sigma_q)}$ ($P(z_i=j) = w_j$)

M-Step: Fit clusters to weighted data point by maximizing log-likelihood

$\Theta^{(t)} = \arg\max_\theta Q(\theta,\theta^{(t-1)}) = \arg\max_\theta \sum_{i=1}^n \sum_{j=1}^k \gamma_j^{(t)}(\vec{x}_i)\cdot\log(w_j P(\vec{x}_i|z=j,\theta))$

→ $w_j^{(t)} = \frac{1}{n}\sum_{i=1}^n \gamma_j^{(t)}(\vec{x}_i)$, $\mu_j^{(t)} = \frac{\sum_{i=1}^n \gamma_j^{(t)}(\vec{x}_i)\vec{x}_i}{\sum_{i=1}^n \gamma_j^{(t)}(\vec{x}_i)}$

$\Sigma_j^{(t)} = \frac{\sum_{i=1}^n \gamma^{(t)}(\vec{x}_i)(\vec{x}-\vec{\mu}_j^{(t)})(\vec{x}-\vec{\mu}_j^{(t)})^T}{\sum_{i=1}^n \gamma_j^{(t)}(\vec{x}_i)} + \gamma^2 I$ (Variance, Without prior)

Initialization:

Weights	Means	Variances
uniform distribution	randomly/kmeans++	spherical with empirical variance

→ either k or γ^2 is selected with CV maximize log-likelihood on testset

Constrained GMM's:

	Spherical	Diagonal	Tied	Full
	$\Sigma_j = \sigma_j^2 I$	$\Sigma_j = diag(\sigma_{j1}^2,\dots,\sigma_{jd}^2)$	$\Sigma_1 = \dots = \Sigma_k$	Σ_j arbitrary
Parameters	k	kd	$\frac{d(d-1)}{2}$	$k\frac{d(d+1)}{2}$

Degeneracy: GMM chooses k=n (1 Gaussian around each Datapoint), thus $\hat{\Sigma}=0$ which leads to overfitting. Can be avoided by Wishart prior ($\gamma^2 I$)

Anomaly Detection: $P(\vec{x}_i|\theta) < J$ → detect x_i as anomaly (control FPR via J)

Generative Adversarial Networks (GAN)

2 Neural networks are trained. The generator tries to produce realistic data out of noise and the discriminator tries to spot fake data. Training leads to a saddle point between both.

Objective: $\min_{\vec{w}_G}\max_{\vec{w}_D} \mathbb{E}_{x\sim p_{data}}[\log(D(\vec{x};\vec{w}_D))] + \mathbb{E}_{\vec{z}\sim p}[\log(1-D(G(\vec{z};\vec{w}_G);\vec{w}_D))]$

Performance Metric: Duality Gap → $DG(\vec{w}_G,\vec{w}_D) = \max_{\vec{w}_D} M(\vec{w}_G,\vec{w}_D) - \min_{\vec{w}_G} M(\vec{w}_G,\vec{w}_D)$

Optimal Discriminator: $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$ → Splits Data with $p = \frac{p_{data}}{p_{data}+p_{model}}$

Optimal Generator: $p_{data}(x) = p_{model}(x)$ → $D_G^*(x) = 1/2$

Other Formulas

Linear Algebra

Matrix composition: $XX^T = [\vec{x}_1,\vec{x}_2,\vec{x}_3]\begin{bmatrix}\vec{x}_1^T\\ \vec{x}_2^T\\ \vec{x}_3^T\end{bmatrix} = \sum_{i=1}^3 \vec{x}_i\vec{x}_i^T$

Transpose: $S = A^Tf \to S^T = f^TA$

Scalar Product: $\langle\vec{u},\vec{v}\rangle = \vec{u}^T\vec{v} = \sum_i u_iv_i$, $\langle\vec{v},\vec{v}\rangle = \|\vec{v}\|_2^2 = \vec{v}^T\vec{v}$

Trace: $tr(A) = \sum_i a_{ii}$, $Tr(\vec{v}\vec{v}^T) = Tr(\vec{v}^T\vec{v})$

Orthogonal Matrices: $A^T = A^{-1} \to A^TA = I$ (columnvectors $\|v_i\|=1$ & orthogonal → $\vec{v}_i^T\vec{v}_j = 0$)

Inverse: $A^{-1} = \begin{bmatrix}a & b\\ c & d\end{bmatrix}^{-1} = \frac{1}{det(A)}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}$ only works squared and rank(A) = full

Rank Nullity Theorem: $rank(A) = rank(A^T) \to dim(ker(A)) + rank(A) = n$

Eigenvalues and vectors: $A\vec{v}_i = \lambda_i\vec{v}_i$ $v_i \in EV$ $\lambda_i \in EW$

Using Differential Calculus: • $\frac{d\,A\vec{x}}{d\vec{x}} = A$, $\frac{d\,\vec{x}^TA}{d\vec{x}} = A$
• $\frac{d\,\vec{x}^TA\vec{x}}{d\vec{x}} = (A+A^T)\vec{x}$

SVD: $A = U\Sigma V^T$ for $m \geq n$ (else $A^T = US^TV^T$)
→ $V = [\vec{v}_1,\dots,\vec{v}_r]$ $\vec{v}_i$ is EV of A^TA
→ $S = diag(s_1,\dots,s_r)+0$ $s_i = \sqrt{\lambda_i}$ λ_i EW of A^TA
→ $U = [\vec{u}_1,\dots,\vec{u}_r]$ $u_i = 1/\lambda_i A\vec{v}_i$ (orthogonal)

LP-Norm: $\|x\|_p = \sqrt[p]{|x_1|^p+\dots+|x_n|^p}$

Complexity: $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$, $C \in \mathbb{R}^{n\times n} \to AB = O(nmp)$ $C^{-1} = O(n^3)$

Analysis

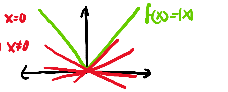
Differentiation Rules: • $f(x)g(x) = f'(x)g(x) + f(x)g'(x)$
• $h(g(x)) = h'(g(x))\,g'(x)$ ← right side important!

Convexity: $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$, $0 \leq \theta \leq 1$ (< for str. convex)
→ f convex if $f''(x) \geq 0$ $\forall x$
→ f strictly convex if $f''(x) > 0$ $\forall x$

Local minima of convex & strictly convex functions are global minima

Gradient: $\nabla(f)(x_0) = \begin{bmatrix}\frac{d}{dx_1}f\\ \vdots\\ \frac{d}{dx_n}f\end{bmatrix}\Big|_{x=x_0}$ Minima if $\nabla f = 0$ & H(f) psd/pd
Maxima if $\nabla f = 0$ & H(f) nsd/pd

Subgradient: Gradient at non-differentiable point of function



Hessematrix: $H = \nabla^2 = \begin{bmatrix}\frac{d^2f}{dx_1^2} & \dots & \frac{d^2f}{dx_1dx_n}\\ \vdots & & \vdots\\ \frac{d^2f}{dx_ndx_1} & \dots & \frac{d^2f}{dx_n^2}\end{bmatrix}\Big|_{x_n=x_0}$ (partial Differentials, Point of Observation)

→ H positive semi definit ($\lambda_i \geq 0$) ⇒ f convex ($\vec{v}^TH(f)\vec{v} \geq 0$) $\forall\vec{v}$ (EV)
→ H positive definit ($\lambda_i > 0$) ⇒ f strictly convex ($\vec{v}^TH(f)\vec{v} > 0$) $\forall\vec{v}$