1 Basics

Improve learning over time, autonomous, feeding data, optimization problem

$\mathbf{x}$: feature vec., $\mathbf{w}$: model parameters, t : (true) label, y : predictor, L : loss func., E : err. func.

Classification : discrete or categorical output

Regression : numerical or continuous output

Supervised Classification : minimize $E(\mathbf{w}) = \sum_{n=1}^{N} L(y(\mathbf{x}_n;\mathbf{w}), t_n)$

Training Testing : 1 use annotated training set to learn, 2 test set to measure performance (both must have same statistical distribution)

1D Model : -1 if observed $<$ T (threshold), 1 otherwise, 2D Model : 2 observed, 2d graph, find decision boundary line between pt.s

Overfit : good train, bad test

Underfit : bad train, bad test

1.1 Python

High-level, readable and concise code, fast prototyping, interpreted, dynamic typing

Slow run-time (dynamic typing, memory management,), modules (NumPy, SciPy,)

2 K Nearest Neighbors

2.1 Nearest Neighbors

Sensitive : pt. close to outlier misclassified

Simplest algorithm : classify new $\mathbf{x}$ according to label of nearest nehbr. in training set

2D Voronoi Diagram : given $\{\mathbf{x}_n\}_{1\le n\le N}$ training samples, Voronoi cells : $C_n = \{\mathbf{x} \in \mathbf{X} \mid \forall j \neq n, d(\mathbf{x},\mathbf{x}_n) \le d(\mathbf{x},\mathbf{x}_j)\}$, Voronoi diagram : $V = \{C_n\}_{1\le n\le N}$, decision boundary : select edges of Voronoi Diagram

2.2 K Nearest Neighbors

Classify according to majority of labels in k nearest nehbr.s

Test data not used during training, misclassifying pt.s near the decision boundary, single meta-parameter k, smaller k greater overfitting, More nehbr.s : + coverage, fewer nehbr.s : better accuracy, undefined zone (same # of neighbors, k even)

Limitations : performance (load all training data, distances to all training samples), distance metric (problematic in high dim.s and with noisy features), curse of dim.ality

Distances : euclidean : $d_2(\mathbf{x},\mathbf{x}') = \sqrt{\sum_{d=1}^{D}(x_i - x_i')^2}$, Manhattan : $d_1(\mathbf{x},\mathbf{x}') = \sum_{d=1}^{D} |x_i - x_i'|$

Cross Validation : split train. set into real training set & valid. set, choose k minimize classification error on valid. set

Run K-NN for many k, use only training and valid. set, pick best k (highest accuracy on valid. set)

Polynomial curve fitting : $y = \sum_{j=0}^{M} w_j x^j$, find $\mathbf{w} = [w_0,\dots,w_M]$ curve closest : $\sum_n (y(x_n,\mathbf{w}) - t_n)^2$

Imbalanced Training set : better represented class favored (fraud/spam detect.); sol.s : weight nehbr.s by inverse of their class size, under-sampling dominant class, augmenting other classes generating synthetic examples

Condensed Nearest nehbr.s : reduce nb. of pt.s, replace pt.s by prototypes : take pt.s that are mislabeled using pt.s already taken, faster

Gossip Based computing : highly parallel, creates a rand. graph, robust to churn partition and breakdowns, adapted to P2P networks

3 K Means

Unsupervised learning : training set not annotated, system learns classes

Clustering : identify groups without data transformation

3.1 K-Means clustering

Group samples into k clusters (no labels required), k given, works for well defined clusters (& homogenous data)

Cluster k : pt.s $\{\mathbf{x}_{i_1^k},\dots,\mathbf{x}_{i_{n_k}^k}\}$, μ_k : center of gravity, mean $\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \in \mathbb{R}^D$

Minimize : $\sum_{k=1}^{K}\sum_{j=1}^{n_k}(\mathbf{x}_{i_j^k} - \mu_k)^2$; init. (param.s) $\mu_{1\le i\le K}$ rand.ly, assign pt. $\mathbf{x}_i$ to nearest center μ_k, update center μ_k given pt.s assigned

Stop : fixed nb. iter.s, iter till convergence (guaranteed, not always to best sol.), diff. in center locations is small

Initial conditions : sensitive to initial conditions, try several diff. rand. init., keep best res.

Inhomogeneous data : euclidean dist. not always best, dim. may have diff. magnitudes, encode diff. types of info.; Sol.s : scale each dim. (subtract smallest val. and scale $[0,1]$), use diff. metric (Manhattan)

Compactness : pt.s close to center of their cluster

Connectivity : pt.s of same cluster are close to one another

Spectral clustering : graph-based connectivity, cut graph on weak connections, similarity $W_{ij} = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$ (σ hyper-param.), cost (to minimize) $\mathrm{cut}(A,B) = \sum_{i\in A, j\in B} W_{ij}$, degree of node $d_i = \sum_j W_{ij}$, volume $\mathrm{vol}(A) = \sum_{i\in A} d_i$

Normilized cut : sol. favor imbalanced partitions, $\mathrm{Ncut}(A,B) = \frac{\mathrm{cut}(A,B)}{\mathrm{vol}(A)} + \frac{\mathrm{cut}(A,B)}{\mathrm{vol}(B)}$

Relaxation : normalized cut $\approx$ eigenvalue problem $(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$, $(\mathbf{D}-\mathbf{W})$ graph Laplacian, sol. is eigenvec. with second smallest eigenvalue ($+$ val. : pt. belong to partition, $-$: doesn't belong)

K-way partition : more than 2 clusters : recursively apply 2-way partitioning (inefficient, unstable), use K eigenvec.s (pt. repr. as K-dim. vec., apply K-means to N vec.s, dim.ality reduction)

4 Linear Regression

Predict a continuous val., requires deriv. w.r.t $\mathbf{w}$

1D Linear regression : param.s w_0, w_1, N training pairs $\{(x_i,t_i)\}, y_i = w_0 + w_1 x_i$ close to t_i (true val.)

Training : squared euclidean $d^2(y_i,t_i) = (y_i - t_i)^2$, least-squares $\min_{w_0,w_1} \frac{1}{N}\sum_{i=1}^{N} d^2(y_i,t_i)$

Prediction : $y_t = w_0^* + w_1^* x_t$

HyperPlane fitting : in dim. D, $y = w_0 + w_1 x_1 + \dots + w_D x_D = \mathbf{w}^T[1\, x_1 \dots x_D]^T = \mathbf{w}^T\mathbf{x}, \mathbf{x} \in \mathbb{R}^{D+1}$; sol. gradient descent or closed-form sol.

Closed-form solution : E (error func.) $\nabla_{\mathbf{w}} E(\mathbf{w}) = \mathbf{0}, \mathbf{X} = [\mathbf{x}_1^T \dots \mathbf{x}_N^T]^T, \mathbf{t} = [t_1 \dots t_n]^T$, $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} = \mathbf{X}^\dagger\mathbf{t}, \mathbf{X}^\dagger$ Moore-Penrose pseudo-inverse

Prediction : $y_t = (\mathbf{w}^*)^T[1\, \mathbf{x}_t]$

Evaluation metrics : Mean Squared Error (MSE) $\mathrm{MSE} = \frac{1}{N_t}\sum_{i=1}^{N_t}(y_i - t_i)^2$, Mean Absolute Error (MAE) ($|y_i - t_i|$), Mean Absolute Percentage Error (MAPE) ($\frac{y_i - t_i}{t_i}$), Root Mean Squared Error (RMSE) $\sqrt{\mathrm{MSE}}$

4.1 Linear classification

Find $\tilde{\mathbf{w}}$ s.t. $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$ is $>$ 0 for $+$ samples, $<$ 0 for most $-$ samples; can't favor one sol. over infinitely others, no close and far from boundary

Line : $(u,v) \in \mathbb{R}^2, au + bv + c = 0$, normalized : $a^2 + b^2 = 1$

Normal unit vec. : $\mathbf{n} = \frac{1}{\sqrt{a^2+b^2}}[a,b]$

Signed dist. to line : $P = [u_1, v_1], O = [u_0, v_0]$ on line, $h = \mathbf{n}[u_1 - u_0, v_1 - v_0] = (au_1 + bv_1 + c)$; h : 0 (on line), $>$ 0 on one side, $<$ 0 on other side

Signed dist. N dim. : $\mathbf{x} \in \mathbb{R}^n$, $\tilde{\mathbf{x}} = [1, x_1, \dots, x_n], \tilde{\mathbf{w}} = [w_0, w_1, \dots, w_N], \sum_{i=1}^{N} w_i^2 = 1$, hyperplane : $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0, h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$

Perceptron : min. $E(\tilde{\mathbf{w}}) = -\sum_{n=1}^{N} sgn(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n)t_n$; set $\tilde{\mathbf{w}}_1 = \mathbf{0}$, pick rnd. idx. $n, \tilde{\mathbf{x}}_n$ misclassified : $\tilde{\mathbf{w}}_{t+1} = \tilde{\mathbf{w}}_t + t_n\tilde{\mathbf{x}}_n$

Test : $y(x;\tilde{\mathbf{w}}) = 1$ if $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} \ge 0$, -1 otherwise

Convergence Theorem : $\exists\gamma > 0$ (margin) & $\mathbf{w}^*$ & $\|\mathbf{w}^*\| = 1$ s.t. $\forall n, t_n(\mathbf{w}^* \cdot \mathbf{x}_n) > \gamma \implies$ Perceptron makes $\le \frac{R^2}{\gamma^2}$ errors, $R = \max_n \|\mathbf{x}_n\|$

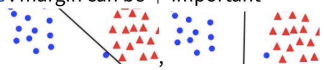
5 Logistic Regression

Logisitc Regression : $y(\mathbf{x};\tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})$, $y_n = y(\mathbf{x}_n;\tilde{\mathbf{w}})$, min. cross entropy $E(\tilde{\mathbf{w}}) = -\sum_n (t_n \ln y_n + (1-t_n)\ln(1-y_n))$ (convex), $\nabla E(\tilde{\mathbf{w}}) = \sum_n (y_n - t_n)\tilde{\mathbf{x}}_n$, find max. likelihood sol.

Sigmoid : $\sigma(a) = \frac{1}{1+\exp(-a)}, \sigma' = \sigma(1-\sigma)$, infin. differentiable, easy derivative, asymptotically 0 or 1

Interpretation : $y(\mathbf{x};\tilde{\mathbf{w}})$: prob. that $\mathbf{x}$ belongs to one class or the other (0.5 on boundary)

Outliers : margin can be $+$ important



Linear discriminant : K linear classifiers $y_k(\mathbf{x}) = \tilde{\mathbf{w}}_k \cdot \tilde{\mathbf{x}}$, boundaries : $y_k(\mathbf{x}) = y_l(\mathbf{x}) \iff (\tilde{\mathbf{w}}_k - \tilde{\mathbf{w}}_l) \cdot \tilde{\mathbf{x}} = 0$, regions are convex

Mulit-Class linear : $y^k(\mathbf{x}) = \tilde{\mathbf{w}}_k \cdot \tilde{\mathbf{x}} = \mathbf{w}_k^T\mathbf{x}$, assign $\mathbf{x}$ to class k if $y^k(\mathbf{x}) > y^l(\mathbf{x}) \forall l \neq k$, $k = \arg\max_j y^k(\mathbf{x})$, same properties as binary logistic regression

Multi-Class Cross Entropy : $t_n^k \in \{0,1\}$ is prob. $\mathbf{x}_n$ in class k, prob. $\mathbf{x}$ in class $k : y^k(x) = \frac{\exp(a^k(\mathbf{x}))}{\sum_j \exp(a^j(\mathbf{x}))}$, entropy $E(\tilde{\mathbf{w}}_1,\dots,\tilde{\mathbf{w}}_k) = -\sum_n\sum_k t_n^k \ln(y^k(\mathbf{x}_n))$, $\nabla E_{\mathbf{w}_j} = \sum_n (y^k(\mathbf{x}_n) - t_n^k)\mathbf{x}_n, a^k(\mathbf{x}) = \tilde{\mathbf{w}}_k^T\mathbf{x}$

6 Max Margin Classifiers

Larger margin, trade train. mistakes for test. acc.

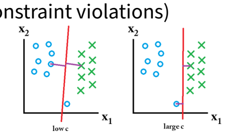
Signed distance : $\mathbf{w}, \mathbf{x} \in \mathbb{R}^N, \tilde{\mathbf{x}} = [1|\mathbf{x}], \tilde{\mathbf{w}} = [w_0|\mathbf{w}], \tilde{\mathbf{w}}' = \tilde{\mathbf{w}}/\|\mathbf{w}\|$; hyperplane : $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$, sgn. dist. : $\tilde{\mathbf{w}}' \cdot \tilde{\mathbf{x}}$, invariant to $\lambda\tilde{\mathbf{w}}$

Max. Margin Classifier : $t_n \in \{-1,1\}$, $\{(\mathbf{x}_n,t_n)_{1\le n\le N}\}, \forall n\, t_n(\tilde{\mathbf{w}}_n \cdot \tilde{\mathbf{x}}_n) \ge 0$, unsigned dist. : $d_n = t_n(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n)/\|\mathbf{w}\|$, max. : $\tilde{\mathbf{w}}^* = \arg\max_{\tilde{\mathbf{w}}} \min_n d_n$

Linear support vector machine SVM : $\min_n d_n = 1/\|\mathbf{w}\|, \mathbf{w}^* = \arg\min_{\mathbf{w}} \|w\|^2/2$ s.t. $\forall n\, t_n \cdot (\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) \ge 1$

Slack Variables : allow some training pt.s to be misclassified, ξ_n for each sample, $t_n \cdot (\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) \ge 1 - \xi_n, \xi_n \ge 0$ weakens constraint; $0 < \xi_n \le 1$ sample n inside margin but correctly classified, $\xi_n \ge 1$ sample n misclassified

Formulation Polynomial SVM : $\mathbf{w}^* = \arg\min_{(\mathbf{w},\{\xi_n\})} \|w\|^2/2 + C\sum_{n=1}^{N}\xi_n$, $\forall n\, t_n \cdot (\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) \ge 1 - \xi_n, \xi_n \ge 0$, C constant (cost of constraint violations)



7 AdaBoost

Strong classifier as weighted sum of weak ones, $Y(\mathbf{x}) = \mathrm{sign}(\sum_{t=1}^{T}\alpha_t y_t(\mathbf{x}))$

Algorithm : init. data weights $\forall n\, w_n^1 = 1/N$; for $t \in [1,\dots,T]$: find classifier y_t minimizing weighted error $\sum_{t_n \neq y_t(x_n)} w_n^t$, evaluated : $\epsilon_t = \frac{\sum_{t_n \neq y_t(x_n)} w_n^t}{\sum_{n=1}^{N} w_n^t}$ & $\alpha_t = \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$, update weights $w_n^{t+1} = w_n^t \exp(\alpha_t \mathbb{1}(t_n \neq y_t(\mathbf{x})))$

Training testing errors : $\epsilon_t < 0.5$ (better than chance) training error exponentially decrease, $1/N\sum_n \mathbb{1}[t_n \neq h(\mathbf{x}_n)] < \prod_{t=1}^{T}\sqrt{\epsilon_t(1-\epsilon_t)}$

8 Polynomial Support Vector Machines

Map data to higher dimension, use linear classifier; increases dimensionality of prob., computationally complex, too complex for large dataset, higher dimensions (irregular boundaries, noise sensitive), more accurate than KNN (param.s must well chosen)

Polynomial Approximation : $\mathbf{w} = [w_0,\dots,w_M]$, $\forall x\, f(x) \approx w_i x^i$, least squares $\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_n (t_n - \sum_{i=0}^{M} w_i x_n^i)^2$, $f_M(x) = \sum_{i=0}^{M} w_i^* x^i$

Feature expansion : $\phi(x) = [1\, x\, x^2 \dots x^M]$, $f(x) = \mathbf{w}^T\phi(x)$

least squares : $\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_n (t_n - \mathbf{w}^T\phi(x_n))^2 = \arg\min_{\mathbf{w}} \|\Phi\mathbf{w} - \mathbf{t}\|^2, \Phi = [\phi(x_1)^T, \dots, \phi(x_N)^T], \mathbf{t} = [t_0, \dots, t_N]$

Regularization : weight decay : tend weight to decay to 0, discourages quick variations, $\mathbf{w}^* = \arg\min_{\mathbf{w}} \|\Phi\mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$

Feature expansion FE : $\phi : \mathbb{R}^d \to \mathbb{R}^D$, $y(\mathbf{x}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}) + w_0)$

Polynomial FE : d-Dimensional, $\phi(\mathbf{x})$ column vec. (every possible monomials)

Polynomial SVM : $\forall n\, t_n(\tilde{\mathbf{w}} \cdot \phi(\mathbf{x}_n)) \ge 1 - \xi_n, \xi_n \ge 0$

Percentage of separab. partitions : N dimension of space, p : number of samples, $\frac{C(p,N)}{2^P}$, separable with large N when $p < 2N$

Lagrangian Formulation : constrained minimization, $L(\mathbf{w},\Lambda) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}\lambda_n(t_n\tilde{\mathbf{w}} \cdot \phi(\mathbf{x}_n) - 1), \Lambda = [\lambda_1, \dots, \lambda_n]$

Support Vectors : $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$, $\mathbf{w} = \sum_{n=1}^{N}\lambda_n t_n\phi(\mathbf{x}_n), y(\mathbf{x}) = \sum_{n=1}^{N}\lambda_n t_n k(\mathbf{x},\mathbf{x}_n) + b$; λ_n non zero only for subset of data pts., $\mathbf{x}_n$ support vectors satisfy $t_n y(\mathbf{x}_n) = 1$, only consider support vectors at test time; $\lambda_n < C$ x_n lies on margin, $\lambda_n = C$ x_n inside margin, large C minimizes number of miscalss. train. pt.s

Inference Time : $y(\mathbf{x}) = \sum_{n=1}^{N}\lambda_n t_n k(\mathbf{x},\mathbf{x}_n) + b$, $f(\mathbf{x})$ not explicit anymore, $k(\dots)$ is a similarity measure; Kernel trick : ϕ implicit (never computed), only need compute k

Kernel : polynomial kernels (small to high dim.) $1 + (\mathbf{x}^T\mathbf{x}')^d$, Gaussian kernels (small to infinite dim., still $O(N^3)$) $\exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2/\sigma^2\right)$

9 Optimization

Convex func. have a global min., find using first or second (faster) order deriv., non-convex usually yield a local min.

Partial derivative : $\frac{\partial f}{\partial x_d} = \lim_{\Delta x \to 0} \frac{f(\ldots, x_d + \Delta x, \ldots) - f(\ldots, x_d, \ldots)}{\Delta x}$

Gradient : $\nabla f = [\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_D}]$, direction of greatest increase at $\mathbf{x}$, magnitude is rate of increase, 0 at stationary pt.s (minima, maxima, saddle pt.s)

Gradient descent : init. $\mathbf{x}_0$ randomly, update $\mathbf{x}_k = \mathbf{x}_{k-1} - \eta \nabla f(x_{k-1})$, η step size (learning rate)

Conjugate gradient : faster convergence, weighted average previous search directions; start $\mathbf{x}_0, g_0 = \nabla F(\mathbf{x}_0)$; take k from 0 to $n-1$: find α_k minimizing $f(\mathbf{x}_k + \alpha_k \mathbf{g}_k)$, $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{g}_k, \beta = \frac{\|\nabla f(\mathbf{x}_{k+1})\|^2}{\|\nabla f(\mathbf{x}_k)\|^2}$, $\mathbf{g}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{g}_k$; $\mathbf{x}_0 = \mathbf{x}_n$, loop

Constrained optimization : $\min_{\mathbf{x}} f(\mathbf{x})$, subject to $f_i(\mathbf{x}) \leq 0 \ (i = 1, \ldots, M), h_i(\mathbf{x}) = 0 \ (i = 1, \ldots, P)$

Implicit func. theorem : min. $f(x, y)$ subject to $g(x, y) \leq c$, at constrained min. $\exists \lambda \in \mathbb{R}$ s.t. $\nabla f = \lambda \nabla g$, λ Lagrange multiplier

Lagrangian : $L(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^M \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^P \nu_i h_i(\mathbf{x})$, λ_i Lagrange mult. for $f_i(\mathbf{x}) \leq 0$, ν_i Lagrange mult. for $h_i(\mathbf{x}) = 0$; at min. $\nabla_{\mathbf{x}} L = \nabla_\lambda L = \nabla_\nu L = 0$

Lagrange dual func. : $g(\lambda, \nu) = \inf_{\mathbf{x}} L(\mathbf{x}, \lambda, \nu)$, concave func., single maximum, lower bound to optimal f^* : $\lambda \geq 0 \implies g(\lambda, \nu) \leq f^*$ for any (λ, ν), maxi. g, constrained to unconstrained

10 Forests

10.1 Trees

Training : compute $p_l(c)$ samples proportion in each class landing in leaf l

Testing : probability of belonging to class c $p(c|\nu) = p_l(c)$ if lands in leaf l

Weak learners : $h(\nu, \theta)$, oriented line $[\tau_1 > \phi(v) \cdot \psi > \tau_2]$, $\phi(v) = (x_1 \ x_2 \ 1)^T$

Entropy : Gini index $Q(S) = \sum_{k=1}^K p^k(1 - p^k)$, entropy $Q(S) = -\sum_{k=1}^K p^k \ln p^k$, both 0 when $\exists k$ s.t. $p^k = 1$, max. when all p^k are equal, minimizing favors leaves with most samples belong to same class

Max. info. gain : $Q(S) - \sum_{\tau \in L,R} \frac{|S^\tau|}{|S|} Q(S^\tau)$

10.2 Forests

Increase robustness, many trees, easy to interpret, behavior easy to modify, trained using moderate amount data

$p(c|v) = f(p_1(c|v), \ldots, p_T(c|v))$

Multiple trees : $S_0^T \subset S_0$ randomly sampled subsets

Fusing output : naive Bayesian $p(c|v) \propto \prod_t p_t(c|v)$, $L(c, v) = \frac{1}{T} \sum_t -\ln(p_t(c|v))$, assumes each tree independent output, training subsets disjoint, DB large enough

Randomized forests : less deep than ada boost, more balanced, good for multi-class

11 Multi-Layer Perceptrons MLP

Differentiable output, like AdaBoost but with all linear classifiers at the same time, piecewise affine result, continuous, descriptive power is larger for deep rather than shallow networks with equal nb. of param.s, perceptrons do not extrapolate well, problem of vanishing/exploding gradients, can handle huge training data, performance is hard to predict

Non-Linear Regression Problem : $(\{x_1, z_1\}, \ldots \{x_n, z_n\})$, min. $\sum_i (z_i - f(\mathbf{x}_i, \tilde{\mathbf{w}}))^2$

Generalize Log.Reg. : $y(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$

MLP : $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \mathbf{W} = [\mathbf{w}_1 \ \ldots \ \mathbf{w}_H]^T$

Binary case : $y_n = \sigma(\mathbf{w}_2(\sigma(\mathbf{W}_1 \mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2) \in [0, 1]$, min. binary cross entropy $E(\mathbf{W}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2) = \frac{1}{N} \sum_{n=1}^N E_n(\ldots)$, $E_n(\ldots) = -(t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n))$, differentiable (gradient)

Multi-Class : $\mathbf{y}_n = \sigma(\mathbf{W}_2(\sigma(\mathbf{W}_1 \mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2) \in \mathbb{R}^k$, $p_n^k = \frac{\exp(\mathbf{y}_n[k])}{\sum_j \exp(\mathbf{y}_n[j])}$, $E_n(\ldots) = -\sum t_n^k \ln(p_n^k)$

Compact : $\mathbf{w} = [\mathbf{w}_1 | \mathbf{b}_1 | \mathbf{w}_2 | \mathbf{b}_2]$, $E(\mathbf{w} = \sum_{n=1}^N E_n(\mathbf{w}))$

Stochastic descent : $\mathbf{w}^{\tau+1} = \mathbf{w}^\tau - \eta \sum_{n \in B^\tau} \nabla E_n(\mathbf{w}^\tau)$, B^τ randomly chosen set of indices (mini-batch), reduce chances falling to local min., possible to compute on GPUs with large databases, helps prevent overfitting

Sigmoid ReLu : sig. issue (value not close to zero means gradients vanish), ReLu boosts performance

ResNet : bypass (final layers only compute residuals), passing input to final layer

Forward pass : $\forall h, a_h = \sum_l w_{hl} x_l, z_h = \sigma(a_h)$; $\forall k, a_k = \sum_j w_{kj} z_j$

Backward pass : $\forall k, \delta_k = \frac{\delta E_n}{\delta a_k}$; $\forall j, \delta_j = \sigma'(a_j) \sum_k w_{kj} \delta_k$

Backprop : $\nabla E_n = [\frac{\partial E_n}{\partial w_{ji}}]$

12 Convolutional Neural Nets

Neighboring pixels are highly correlated, image filter should be translation equivariant

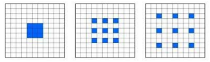
Convolution : $g * f(t) = \int_\tau g(t - \tau) f(\tau) \, d\tau$

Discrete 2D : $m * *f(x, y) = \sum_{i=0}^w \sum_{j=0}^w m(i, j) f(x - i, y - j)$, m known as kernel

2D Convolutional Layer : $a_{i,j}^1 = \sigma(b + \sum_{x=0}^{n_x} w_{x,y} a_{i+x,j+y}^0)$, same weights $w_{x,y}$ used for all activations, fewer weights than fully connected layers

Pooling Layer : reduces number of inputs, replace all activations in neighborhood by single one, max-polling is simply keeping the max. value

Stride : larger than 1 reduces and convolves at same time,



stride=1 stride=2 stride=3

Feature Maps : convolutional masks, oriented derivatives (probably like the brain)

Shriking and reexpanding : composition of convolution + transposed convolution with same param.s : signal size unchanged, create grid-structure artifacts

UNet : convolution, downsampling, upsampling (duplication, (bilinear) interpolation, transpose convo.)

Estimating Tubularity : min. $L_{BCE} = \frac{1}{N} \sum_{i=1}^N y_n \log(\hat{y}_n) + (1 - y_n) \log(\hat{y}_n)$, $\hat{y} = f_w(x)$

13 Transformers

Context matters

Self attention : I words $\mathbf{x}_i, \forall i$, sa$[\mathbf{x}_i] = \sum_{j=1}^I a[\mathbf{x}_i, \mathbf{x}_j] \mathbf{W}_v \mathbf{x}_j$

Matrix : query $\mathbf{X}_q = \mathbf{X}\mathbf{W}_q$, key $\mathbf{X}_k = \mathbf{X}\mathbf{W}_k$, value $\mathbf{X}_v = \mathbf{X}\mathbf{W}_v$

Attention Weights : $a[\mathbf{x}_i, \mathbf{x}_j] = \text{softmax}_j[(\mathbf{W}_q \mathbf{x}_i)^T \mathbf{W}_k \mathbf{x}_j]$, Sa$(\mathbf{X}) = \text{Softmax}[\mathbf{X}\mathbf{W}_q \mathbf{W}_k^T \mathbf{X}^T] \mathbf{X}\mathbf{W}_w$, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_I]$

Transformer Layer : $\mathbf{X} \to \mathbf{X} + \text{Sa}(\mathbf{X})$, $\mathbf{X} \to \text{LayerNorm}(\mathbf{X}), \mathbf{x}_i \to \mathbf{x}_i + \text{mlp}[\mathbf{x}_i] \ \forall i$, $\mathbf{X} \to \text{LayerNorm}(\mathbf{X})$

Vision Transformers : break up images into square patches, transform each patch into feature vector, feed to transformer architecture

U-Net + Transformers : CNN operates at low-resolution produces feature vector, transform on FV, upsampling like U-Net

14 Dimensionality Reduction

Discovering data manifold, finding low-dimensional representation of data, loss of information, noise reduction, unsupervised

Formalization : mapping $\mathbf{y}_i = f(\mathbf{x}_i), \mathbf{x}_i \in \mathbb{R}^D$ high-dim. data sample, $\mathbf{y}_i \in \mathbb{R}^d$ low-dim. repr.

14.1 Linear

Linear : $\mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i$

PCA : N samples $\{\mathbf{x}_i\}$, $\mathbf{y}_i = \mathbf{W}^T(\mathbf{x}_i - \bar{\mathbf{x}})$ s.t. $\mathbf{W}^T\mathbf{W} = \mathbf{I}_d, \bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$

PCA objective : keep important signal, remove noise, find directions with large variance; for j-th output dim. maxim. $\text{var}(y_i^{(j)}) = \frac{1}{N} \sum_{i=1}^N (y_i^{(j)} - \bar{y}^{(j)})^2$, $\bar{y}^{(j)}$ mean of dim. of j-th data pt. after projection

Variance maximization 1D : D-dim. vec. $\mathbf{w}_1$ s.t. $\mathbf{w}_1{}^T \mathbf{w}_1 = 1$, $\bar{y} = \mathbf{w}_1^T \bar{\mathbf{x}}$, $\text{var}(\{y_i\}) = \mathbf{w}_1^T \mathbf{C} \mathbf{w}_1$, input data covar. matrix $\mathbf{C} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}}^T)$; $\max_{\mathbf{w}_1} \mathbf{w}_1^T \mathbf{C} \mathbf{w}_1$

Solve lagrangian : $L(\mathbf{w}_1, \lambda_1) = \mathbf{w}_1^T \mathbf{C} \mathbf{w}_1 + \lambda_1(1 - \mathbf{w}_1^T \mathbf{w}_1)$, gradient to 0 $\mathbf{C}\mathbf{w}_1 = \lambda_1 \mathbf{w}_1$, $\mathbf{w}_1$ eigenvec. of $\mathbf{C}$ with the largest eigenval. λ_1

d > 1 : $\mathbf{W} = [\mathbf{w}_1 | \ldots | \mathbf{w}_d] \in \mathbb{R}^{D \times d}$, larger eigenvalues

Explained variance : $\mathbf{W}^T C \mathbf{W} = \sum_i \lambda_i$

Without dimensionality reduction : $d = D$, (3D : rotation of data), no loss of information, data projected to uncorrelated axes

Without Loss of info. : keep all eigenvectors (non-zero eigenvalues), $\{\mathbf{y}_i\}$ lower dimensional $d < D$

Retain variance : $\sum_{j=1}^d \lambda_j \geq V \sum_{k=1}^D \lambda_k$, find d, to explain V of the variance

PCA Mapping : $\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{W}\mathbf{y}$, regularization

Optimal linear mapping : some loss of information, rectangular $\mathbf{W}$ orthogonal minimizing error $e = \|\hat{\mathbf{x}} - \mathbf{x}\|^2$ where $\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{W}\mathbf{y} = \bar{\mathbf{x}} + \mathbf{W}\mathbf{W}^T(\mathbf{x} - \bar{\mathbf{x}})$

Fisher Linear discriminant analysis LDA : cluster samples form same class (C classes), minim. $E_W(\mathbf{w}_1) = \sum_{c=1}^C \sum_{i \in C} (y_i - \nu_C)^2$, ν_C mean of samples in class c after projection, y_i & ν_c depend on $\mathbf{w}_1$; $E_W(\mathbf{w}_1) = \mathbf{w}_1^T \mathbf{S}_W \mathbf{w}_1$, withing-class scatter matrix $\mathbf{S}_W = \sum_{c=1}^C \sum_{i \in c} (\mathbf{x}_i - \mu_c)(\mathbf{x}_i - \mu_c)^T$

Separating different classes : separate different clusters, push means of clusters away, maxim. $E_B(\mathbf{w}_1) = \sum_{c=1}^C N_c(\nu_c - \bar{y})^2$, $\bar{y}$ mean of all samples after projection, N_c nb. of samples in class c; $E_B(\mathbf{w}_1) = \mathbf{w}_1^T \mathbf{S}_B \mathbf{w}_1$, between-class scatter matrix $\mathbf{S}_B = \sum_{c=1}^C N_c(\mu_c - \bar{\mathbf{x}})(\mu_c - \bar{\mathbf{x}})^T$, $\bar{\mathbf{x}}$ mean of all samples, $\{\mu_c\}$ class-specific means

Fisher LDA 1D : maximize $J(\mathbf{w}_1) = \frac{E_B(\mathbf{w}_1)}{E_W(\mathbf{w}_1)}$; $\max_{\mathbf{w}_1} \mathbf{w}_1^T \mathbf{S}_B \mathbf{w}_1$ with $\mathbf{w}_1^T \mathbf{S}_W \mathbf{w}_1 = 1$; 0 gradient Lagrangian : $\mathbf{S}_B \mathbf{w}_1 = \lambda_1 \mathbf{S}_W \mathbf{w}_1$, $\mathbf{w}_1$ eigenvector with largest eigenvalue

PCA vs. LDA : max. projected var / max. btw-var min. within-var

14.2 Non-Linear

Latent Space : $\mathbf{z} = f_e(\mathbf{x}), \hat{x} = f_d(\mathbf{z}), \hat{\mathbf{x}} \approx \mathbf{x}$, removes unnecessary degrees of freedom, denoise original data

Basic autoencoder : $\mathbf{z} = \sigma_e(\mathbf{W}_e \mathbf{x} + \mathbf{b}_e)$ latent vector repr. of $\mathbf{x}$, $\hat{\mathbf{x}} = \sigma_d(\mathbf{W}_d \mathbf{z} + \mathbf{b}_d)$ reconstruction of $\mathbf{x}$, $\mathbf{W}_e$, $\mathbf{W}_d$ computed by minim. $\sum_n \|\hat{\mathbf{x}}_n - \mathbf{x}_n\|^2$ (unsupervised)

Deep autoencoder : stack layers with activ. func.

Complete : $\dim \mathbf{z} < \dim \mathbf{x}$ undercomplete (compress input, captures correlations), $\dim \mathbf{z} > \dim \mathbf{x}$ overcomplete (higher dim. can help, degenerate sol.s possible, need regularization term)

Denoising : low-dim. latent repr. encourages "intelligent" mapping, dim. expansion learn to copy input; to prevent : add noise to input and aim to reconstruct noise-free version, avoid trivial solutions using regularization term $R(\mathbf{w}) = \sum_n L(\mathbf{x}_n, \mathbf{w}) + \lambda \Omega(\mathbf{x}_n, \mathbf{w})$