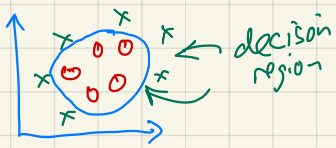
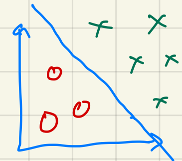
UNIT 5 | CLASSIFICATION

- predict a <u>categorical</u> label y from $\underline{x}$ given training data $\{(\underline{x}_i, y_i)\}_{i=1}^{n}$

- <u>binary</u> y:
  $y=1$ or $y \neq 1$

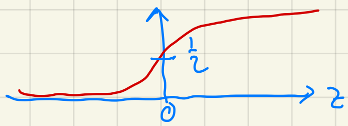


linear $\longrightarrow$



- binary <u>linear</u> classification: 1) compute a "score" $z = b + \underline{w}^T \underline{x} \in \mathbb{R}$
  2) threshold: $\begin{cases} z \geq t & \Rightarrow \hat{y} = 1 \\ z < t & \Rightarrow \hat{y} \neq 1 \end{cases}$
  choose $t=0$ to maximize classification accuracy

- training via logistic model
  - model: $\Pr\{y=1 \mid z\} = \dfrac{1}{1+e^{-z}}$



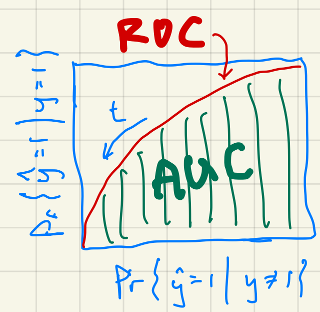
"sigmoid"

  - ML estimation: $(b,\underline{w})_{ML} = \arg\max_{(b,\underline{w})} \prod_{i=1}^{n} p(y_i \mid z_i)$ for $z_i = b + \underline{w}^T \underline{x}_i$
    $= \arg\min_{(b,\underline{w})} \sum_{i=1}^{n} \left[\ln(1+e^{z_i}) - y_i z_i \right]$ for $y_i \in \{0,1\}$
    binary cross entropy (BCE) loss

- $K>2$ classes
  - linear classifier: 1) $z_k = b_k + \underline{w}_k^T \underline{x} \in \mathbb{R}$ for $k=1..K$
    2) $\hat{y} = \arg\max_k z_k$
  - training: A) one-vs-rest (OVR): for each class $k=1..K$, train $(b_k, \underline{w}_k)$ using binary logistic regression
    B) multinomial logistic regression (MLR):
      - model: $\Pr\{y=k \mid \underline{z}\} = \dfrac{e^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$ where $z_k = b_k + \underline{w}_k^T \underline{x}$
      - ML estimation: $\arg\max_{(\underline{b},\underline{W})} \prod_{i=1}^{n} \Pr\{y_i = k \mid \underline{z}_i\}$ where $\underline{z}_i = \underline{b} + \underline{W}\underline{x}_i$

- Performance metrics (binary)
  - accuracy: $\Pr\{\hat{y} = y\}$
  - precision: $\Pr\{y=1 \mid \hat{y}=1\}$
  - recall: $\Pr\{\hat{y}=1 \mid y=1\}$
  - others...

by choosing threshold "t" we can trade off between various metrics, like precision vs recall



AUC is a threshold-independent metric

UNIT 6 OPTIMIZATION

- gradient: say $\underline{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$ and $J(\underline{w}) \in \mathbb{R}$

  then $\nabla J(\underline{w}) = \begin{bmatrix} \frac{\partial J(\underline{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_d}(\underline{w}) \end{bmatrix}$ same shape as $\underline{w}$

  also can do for matrices, tensors, scalars

  tells us the slope and direction of maximum increase of J at $\underline{w}$

- gradient descent (GD) optimization of a smooth cost $J(\cdot)$:

  Initialize $\underline{w}$ and $\underline{w}^0$ and iterate for $t = 1, 2, 3, \ldots$

  $$\underline{w}^{t+1} = \underline{w}^t - \alpha_t \nabla J(\underline{w}^t)$$

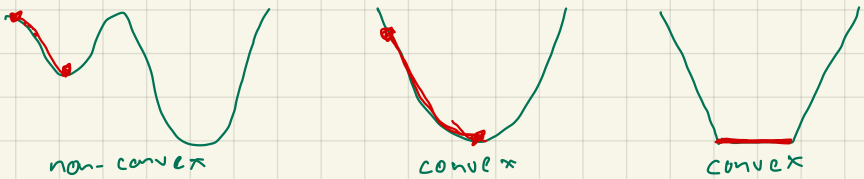
  α_t ← small positive stepsize

- analysis: for sufficiently small α_t, we proved $J(\underline{w}^{t+1}) \leq J(\underline{w}^t)$ for all t

  via $$J(\underline{w}^{t+1}) = J(\underline{w}^t) \underbrace{- \alpha_t \|\nabla J(\underline{w}^t)\|^2 + \alpha_t^2\, O(\|\nabla J(\underline{w}^t)\|^2)}_{\leq 0 \text{ if } \alpha_t \text{ is small enough}}$$

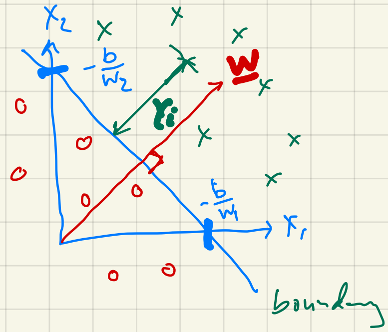
- Armijo stepsize adaptation: At each iteration t, perform GD update, then
  - check if $J(\underline{w}^{t+1}) \leq J(\underline{w}^t) - \frac{1}{2}\alpha_t \|\nabla J(\underline{w}^t)\|^2$ adjustable
  - if not, set $\alpha_t \leftarrow \frac{1}{2}\alpha_t$ and try t'th iteration again
  - if yes, set $\alpha_t \leftarrow 2\alpha_t$ and accept the iteration and increment t

- If $J(\cdot)$ is convex, then all local minimizers are global minimizers



examples: RSS, logistic regression (and L2 & L1 regularized versions)
hinge loss (for SVC)

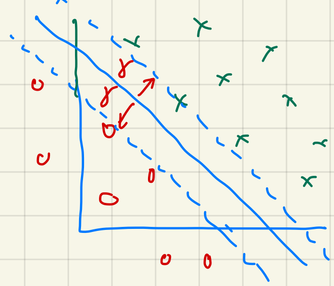
UNIT 7 SVC & SVM



- binary linear classification

$$\gamma_i = \frac{b + \underline{w}^T x_i}{\|\underline{w}\|}$$ signed distance from x_i to boundary

- "linearly separable" data means that there exists some $(b, \underline{w})$ such that



$$y_i(b + \underline{w}^T \underline{x}_i) \geq 1 \quad \forall i$$

$$\iff y_i \gamma_i \geq \frac{1}{\|\underline{w}\|} \triangleq \gamma \quad \forall i$$ "margin"

- hard margin classifier

$$\arg\min_{b, \underline{w}} \frac{1}{2}\|\underline{w}\|^2 \quad \text{s.t. } y_i(b + \underline{w}^T x_i) \geq 1 \ \forall i$$

only feasible when data is linearly seperable

- soft margin or support vector classifier (SVC)

$$\arg\min_{b, \underline{w}} \left\{ C \sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i z_i\}}_{\text{hinge loss}} + \frac{1}{2}\|\underline{w}\|^2 \right\} \quad z_i = b + \underline{w}^T x_i$$

as C increases, the margin γ decreases (bias-variance tradeoff)

the solution $(b_*, \underline{w}_*)$ depends only on the support vectors

$$\{ \underline{x}_i : y_i(b_* + \underline{w}_*^T x_i) \leq 1 \}$$

- the SVM is SVC with a (usually nonlinear) transformation $\underline{x} \to \underline{\phi}(\underline{x})$

But the SVM avoids directly computing (or even defining) $\underline{\phi}(\cdot)$

Instead it computes the kernel $K(\underline{x}_i, \underline{x}_j) \triangleq \underline{\phi}^T(\underline{x}_i)\, \underline{\phi}(\underline{x}_j)$

using the "dual" form of the SVC optimization problem

The most popular kernel is RBF:

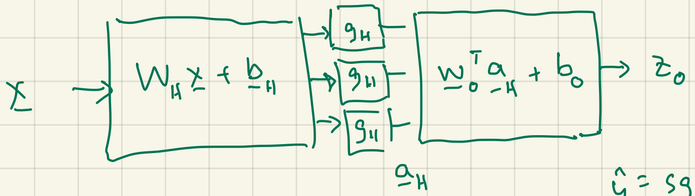
$$K(\underline{x}_i, \underline{x}_j) = \exp\left(-\alpha \|\underline{x}_i - \underline{x}_j\|^2\right)$$

α tunable > 0

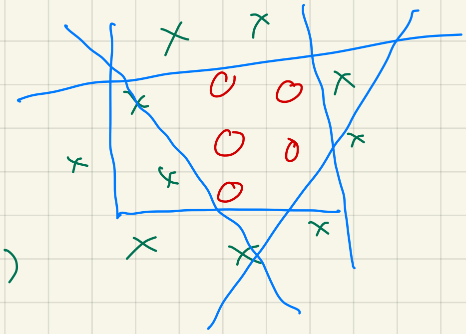
UNIT 8 NEURAL NETS

- Can we learn a feature transformation $\phi(\cdot)$ from the training data?
- Yes, we will use the form $[\underline{\phi}(\underline{x})]_\ell = g_H(b_{H,\ell} + \underline{w}_{H,\ell}^T \underline{x})$ for $\ell = 1 \dots d_H$

  and then do linear classification / regression



$\hat{y} = \mathrm{sgn}(z_o)$



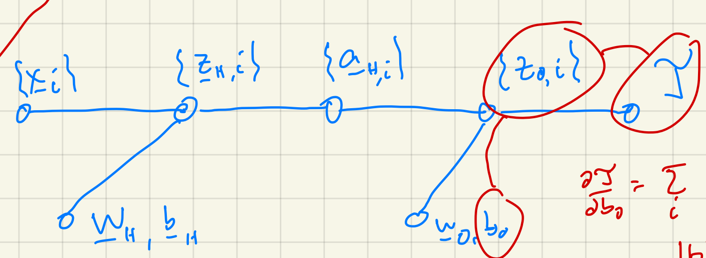
- Train $\underline{\theta} = \{ \underline{W}_H, \underline{b}_H, \underline{w}_o, b_o \}$

  to minimize $\mathcal{J}(\underline{\theta}) \triangleq \sum_{i=1}^{n} J(\underline{\theta}, \underline{x}_i, y_i)$

  via GD: $\underline{\theta}^{t+1} = \underline{\theta}^t - \alpha_t \nabla \mathcal{J}(\underline{\theta}^t)$ or split into mini-batches

- To compute gradients

  Computation graph



$\frac{\partial \mathcal{J}}{\partial b_o} = \sum_i \frac{\partial \mathcal{J}}{\partial z_{o,i}} \frac{\partial z_{o,i}}{\partial b_o}$

multivariable chain rule

Backpropagation: 1) forward pass:

compute $\{\underline{z}_{H,i}\}, \{\underline{a}_{H,i}\}, \{z_{o,i}\}, \mathcal{J}$

2) backward pass

compute: $\left\{\frac{\partial \mathcal{J}}{\partial z_{o,i}}\right\}, \left\{\frac{\partial \mathcal{J}}{\partial \underline{a}_{H,i}}\right\}, \left\{\frac{\partial \mathcal{J}}{\partial \underline{z}_{H,i}}\right\}$

$\frac{\partial \mathcal{J}}{\partial b_o}, \frac{\partial \mathcal{J}}{\partial \underline{w}_o}, \frac{\partial \mathcal{J}}{\partial \underline{b}_H}, \frac{\partial \mathcal{J}}{\partial \underline{W}_H}$