# CS 480 Cheat Sheet

## Perceptron

We assume $(x_1, y_1), ..., (x_n, y_n)$ belongs to some distribution.
Choose predictive function $h_n$ such that $\max Pr(h(x_i) = y_i)$
**Dot Product:** $\langle w, x\rangle = \sum w_i x_i$
**Padding:** $\langle w, x\rangle + b = \langle (w, b), (x, 1)\rangle$
Note: $z = (w, b)$, $a_i = y_i(x_i, 1)$
**Linear Seperable:** if $s > 0$ and $Az > s1$
If data is not linearly seperable perceptron stalls.
Margin is determined by closest point to the hyperplane.
Perceptron finds a solution, may not be best solution.
**l2 Norm:** $||x||_2 = \sqrt{\sum_i x_i^2}$
**Error Bound** $\leq \frac{R^2||z||_2^2}{s^2}, R = \max ||a_i||_2$
**Margin:** $\gamma = \max_{||z||_2=1} \min_i \langle a_i, z\rangle$
**One-versus-all:** $\hat{y} = \text{argmax}_k \ w_k^T x + b_k$
**One-versus-one:** $\#\{x^T w_{k,k'} + b_{k,k'} > 0, x^T w_{k',k} + b_{k',k} < 0\}$

## Linear Regression

**Gradient:** if $f(x)\mathbb{R}^d \to \mathbb{R}$, $\Delta f(v) = \left(\frac{\delta f}{\delta v_1}, ..., \frac{\delta f}{\delta v_d}\right) \mathbb{R}^d \to \mathbb{R}^d$

**Hessian:** $\Delta^2 f(v) = \begin{bmatrix} \frac{\delta^2 f}{\delta^2 v_1^2} & ... & \delta v_d^2 \delta v_1^2 \\ \vdots & & \vdots \\ \frac{\delta^2 f}{\delta v_1^2 \delta v_d^2} & ... & \delta^2 v_d^2 \end{bmatrix} : \mathbb{R}^d \to \mathbb{R}^{d \times d}$

**Emprical Risk Minimization:** $\text{argmin}_w \frac{1}{n} \sum_d l_w(x, y)$
**Convexity #1:** $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$
**Convexity #2:** if $\Delta^2 f(x)$ is positive semi definite.
**Positive Semidefinite:** if $M \in \mathbb{R}^{d \times d}$, PSD iff $v^T M v \geq 0$
Loss function needs to be convex to optimizes.
Setting loss function to 0 optimizes our solution.
MLE principle: pick paramaters that maximize likelihood.
**Ridge Regularization:** $\arg\min_w ||Aw - z||_2^2 + \lambda ||w||_2^2$
**Lasso Regularization:** $\arg\min_w ||Aw - z||_2^2 + \lambda ||w||_2$

## k-Nearest Neighbour Classification

**Bays Optimal Classifier:** $f^*(x) = \arg\max_c \Pr(y = c|x)$
**NN Assumption:** $Pr[y = c|x] \approx Pr[y' = c|x']$, $x \approx x'$
No classifier can do as good as bayes.
Cant be descriped as a parameter vector.
Can express non linear relationships.
Takes 0 training time, and $O(nd)$ to $O(d \log n)$ testing time.
Small values of k lead to overfitting.
Large values of k lead to high error.
**1NN Limit:** as $n \to \infty$ then $L_{1NN} \leq 2L_{Bayes}(1 - L_{Bayes})$

## Logistic Regression

Classifications are take into account confidence: $\hat{y} \in (-1, 1)$
**Bernoulli Model:** $Pr[y = 1|x, w] = p(x, w) \in (-1, 1)$
**Logit Transform:** $\log\left(\frac{p(x,w)}{1-p(x,w)}\right) = \langle x, w\rangle = \frac{1}{1+exp(-\langle x,w\rangle)}$
**Optimizing Loss:** $\Delta_w l_w(x_i, y_i) = (p_i(x_i, w) - y_i)x_i$
**Iterative Update:** $w_t = w_{t-1} - \eta d_i$
**Gradient Descent:** $d_t = \frac{1}{n}\sum_i^n \Delta_w l_{wt-1}(x_i, y_i)$
**Stochastic GD:** $B \in [n], d - t = \frac{1}{|B|}\sum_{i \in B} \Delta_w l_{wt-1}(x_i, y_i)$
**Newton's Method** $d_t$ is given by the equation below:

$$d_t = (\tfrac{1}{n}\textstyle\sum_i^n \Delta_w^2 l_{wt-1}(x_i, y_i))^{-1}(\tfrac{1}{n}\sum_i^n \Delta_w l_{wt-1}(x_i, y_i))$$

**Multiclass Logisitc Regression** where k = class: $Pr[y = k|x, w] = \frac{exp(\langle w_k, x\rangle)}{\sum_l exp(\langle w_l, x\rangle)}$

## Hard-Margin SVM

Assume that dataset is linearlly seperable. Hard Margin SVM's will try to find the "best" solution. The best solution is the one that maximizes margin.
**Optimize:** $min_{w,b} \frac{1}{2}||w||_2^2 \ s.t \ y\hat{y} \geq 1$
**Primal:** $min_{w,b} \frac{1}{2}||w||_2^2 \ s.t \ y_i(\langle w, x_i\rangle + b) \geq 1$
**Duel:** $min_a \frac{1}{2}\sum_i \sum_j a_i a_j y_i y_j \langle x_i, x_j\rangle \ s.t \ \sum a_i y_i = 0$
**Complimentary Slackness:** $a_i(y_i(\langle w, x_i\rangle + b) - 1) = 0, \forall i$
**Support Vector:** if $a_i > 0$ then w=$\sum a_i y_i x_i$

## Soft-Margin SVM

Data does not need to be linearlly seperable.
**Soft-Margin:** $\min_{w,b} \frac{1}{2}||w||_2^2 + C\sum_i \max(0, 1 - y_i\hat{y}_i)$
if $1 - y_i\hat{y}_i \leq 0 \implies$ Correct side of margin.
if $0 < 1 - y_i\hat{y}_i \leq 1 \implies$ Correctly classified, inside of margin.
if $y_i\hat{y}_i \leq 0 \implies$ incorrectly classified.
If C=0 ignore data, if C=$\infty$, hard-margin.
**Slack Variable:** define $\gamma_i$ such that $max(0, 1 - y_i\hat{y}_i) \leq \gamma_i$
**Split in Two:** $0 \leq \gamma_i$ and $1 - y_i\hat{y}_i \leq \gamma_i$
**Duel Solution:** Note $0 \leq \gamma_i$ and $1 - y_i\hat{y}_i \leq \gamma_i$ implies:
$=\max_{\alpha,\beta} min_{w,b,\gamma} \frac{1}{2}||w||_2^2 + \sum(C\gamma_i + \alpha(11 - y_i\hat{y}_i - \gamma_i) - \beta_i\gamma_i)$
$=\min_\alpha \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - \sum a_i$ s.t $\sum a_i y_i = 0$
if $a_i = 0$ then $y_i = 0$, point is classified correctly.
if $a_i > 0$ and $y_i = 0$, point is on margin.
if $a_i > 0$ and $y_i > 0$, point is on within margin.
**Loss Function:** L $= \frac{C}{n}\sum_i l_{w,b}(x_i, y_i) + \frac{1}{2}||w||_2^2$
**Gradient Descent**: $\frac{\delta L}{\delta w} = w + C/N \sum \ \delta_i$
if $1 - y_i\hat{y}_i \geq 0$, then $\delta = -y_i x_i$ else $\delta = 0$

## Kernels

Map data to new space where it is linearlly seperable.
**Padding Trick:** $\o(x) = [w, 1]$ and $w = \langle x, p\rangle$
**New Classifier:** $\langle \o(x), w\rangle = \langle x, p\rangle + b > 0$
**Quadratic Feature:** $x^t Q x + \sqrt{2}x^T p + b$, wich gives us:
$\o(x) = [xx^t, \sqrt{2}x, 1]$ and $w = [Q, p, b]$
With feature map $\o : \mathbb{R}^d \to \mathbb{R}^{d \times d + d + 1}$, time O(d) to O($d^2$)
This can take infinite time in high dimensions. For the duel we only need to calculate dot product.
**Kernal:** k:$\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ if k(x,x') $= l\angle\o(x), \o(x')\rangle$
A kernel if valid if its symmetric and positive semi-definite.
**New Kernel:** $K_{ij} = \langle \o(x^i), \o(x^j)\rangle = k(x^i, x^j)$
**Classifiy New:** sign($\sum a^i y^i k(x^i, x)$)
**Polynomial Kernel t:** $k(\langle x, x'\rangle + 1)^t$
**Gaussian Basis:** $exp(||x - x'||_2^2)$
SVM (Linear Kenrel): O(nd) train time, O(d) test time.
General Kernel: $O(n^2 d)$ train time O(nd) test time.

## Decision Trees

Can do classification or regression, handle non-linear functions.
May fail on linear functions.
Start at one node, and split each. Select the pure node.
**Loss:**t*=$\text{argmin}_t\, l(\{(x_i, y_i) : x_i \leq t\}) + l(\{(x_i, y_i) : x_i > t\})$
Let $p_c$ = frac of S with label c. $\hat{y} = arg\max_c p_c$
**Misclassification Loss:** l(s) $= 1 - p_y$
**Entropy Loss:** l(s) $= -\sum_{classesc} p_c \log p_c$
**Gini Index Loss:** l(s) $= \sum_{classesc} p_c(1 - p_c)$
**Regression:** l(s) $= min_p \sum_{i \in S}(y_i - p)^2 = \sum_{i \in S}(y_i - y_s)^2$
We can stop based on run time, depth or splits.
Once a tree is fully grown we can prune it.

## Bagging

Training on empirical mean gives a variance of: $E[\hat{\mu}] = \mu$

$Var[\hat{\mu}] = Var[\frac{1}{n}\sum X_i] = \frac{1}{n^2}Var[\sum X_i] = \sigma^2/n$

We can reduce variance by taking a sample of B points:

$Var[\hat{\mu}] = Var[\frac{1}{B}\sum X_i] = \frac{1}{B^2}Var[\sum X_i] = \sigma^2/Bn$

We can sample these points with replacement, and in practice this will work.

We aggregate by doing regression $f(x) = \frac{1}{B}\sum f^j(x)$.

Classification done by majority vote.

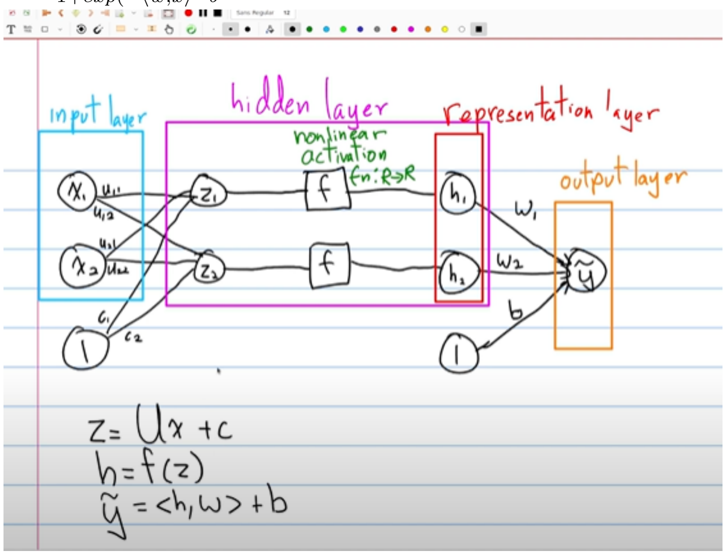
**Random Forests** Bootstrap but select $\sqrt{d}$ features.

## Multilayer Perceptron

Neural Networks learn mapping from the data.

**Sigmoid:** $\sigma(t) = \frac{1}{1+e^{-t}}$

$\hat{y} = \frac{1}{1+exp(-\langle w,x\rangle - b})$




**ReLU(t)** $= \max(0, t)$

**Loss** $l_\theta(x, y) = -\sum^m y_i log y_i$

**Tanh(t)**$= \frac{e^t - e^{-1}}{e^t + e^{-t}}$

**Gradient Descent** $\theta^t = \theta^{t-1} - \eta\Delta L_{\theta t-1}$

**Chain rule:** $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

Any contionus function can be approximated well by a 2 layer nn.

## Multilayer Perceptron

Most neural networks have many paramaters. We can minimize overfiting with:

Regularazation loss, gradient descent, and equivalently:

$\theta_t \leftarrow (1-\eta\lambda)\theta_{t-1} - \eta\Delta L\theta t - 1)(x, y)$

We can drop nodes and use normlizaiton of features:

mean$= \frac{1}{n}\sum X_i$, $X_i \leftarrow X_i - \mu$, $\sigma_j^2 = \frac{1}{n}\sum X_{i,j}$, $X_{i,j} = X_{i,j}/\sigma_j$

We do normilizaiton on each batch, such that:

$z^i = W^i h^{i-1} + b^i$ or $h^i = f(z^i)$

We can do normlization on each neuron (batchnorm) or each layer.

**Batch GD:** $\theta \leftarrow \theta - \eta * \frac{1}{n}\sum \delta l_\theta(x_i, j_i)$, optomize gradient.

**Momentum:** $v_t = \gamma v_{t-1} + (1-\gamma)\mu, \theta_t \leftarrow \theta_{t-1} - v_t$ **RMSProp** let $g \in R^p$, $G_{t,i} = \sum^t g_{j,i}^2$and $\theta_t \leftarrow \theta_{t-1} - \frac{\mu}{\sqrt{G_{t,i}+\epsilon}} g_{t,i}$

- $\beta_1, \beta_2, \varepsilon$ hyperparameters
  - $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$
- $m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1)g_{t,i}$ (momentum)
- $v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2)g_{t,i}^2$ (RMSProp)
- $\widehat{m}_{t,i} = \frac{m_{t,i}}{1-\beta_1^t}, \hat{v}_{t,i} = \frac{v_{t,i}}{1-\beta_2^t}$
- $\theta_{t,i} \leftarrow \theta_{t-1,i} - \frac{\eta}{\sqrt{\hat{v}_{t,i}+\varepsilon}}\widehat{m}_{t,i}$