Math Review

$\mathbf{X}$ and $\mathbf{Y}$ are **independent** iff $\mathbb{P}(\mathbf{X},\mathbf{Y}) = \mathbb{P}(\mathbf{X})\,\mathbb{P}(\mathbf{Y})$
$\mathbf{X}$ and $\mathbf{Y}$ are **uncorrelated** iff $\mathbb{E}(\mathbf{X},\mathbf{Y}) = \mathbb{E}(\mathbf{X})\,\mathbb{E}(\mathbf{Y})$
Expected value of $g(X)$: $E[g(X)] = \int_{-\infty}^{\infty} g(x) f(x)\,dx$
Variance $\sigma^2 = E[(X-\mu)^2] = E[X^2] - \mu^2$
Determinant of matrix is product of its eigenvalues.

$$f(\mathbf{w}) = \mathbf{A}\vec{\boldsymbol{x}} + \vec{\boldsymbol{x}}^\top \mathbf{A} + \vec{\boldsymbol{x}}^\top \vec{\boldsymbol{x}} + \vec{\boldsymbol{x}}^\top \mathbf{A}\vec{\boldsymbol{x}} \Rightarrow \frac{df(\vec{\boldsymbol{x}})}{d\vec{\boldsymbol{x}}} = \mathbf{A}^\top + \mathbf{A} + 2\vec{\boldsymbol{x}} + \mathbf{A}\vec{\boldsymbol{x}} + \mathbf{A}^\top \vec{\boldsymbol{x}}$$

$$\nabla_{\vec{\boldsymbol{x}}}(\vec{\boldsymbol{y}} \cdot \vec{\boldsymbol{z}}) = (\nabla_{\vec{\boldsymbol{x}}} \vec{\boldsymbol{y}})\vec{\boldsymbol{z}} + (\nabla_{\vec{\boldsymbol{x}}} \vec{\boldsymbol{z}})\vec{\boldsymbol{y}} \qquad \nabla_{\vec{\boldsymbol{x}}} f(\vec{\boldsymbol{y}}) = (\nabla_{\vec{\boldsymbol{x}}} \vec{\boldsymbol{y}})(\nabla_{\vec{\boldsymbol{y}}} f(\vec{\boldsymbol{y}}))$$

$$\nabla_w w^T A w = (A + A^T) w \qquad \mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Perceptron (Lecture 2)

$f(\vec{\boldsymbol{x}}) = \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} + \alpha = \sum_{i=1}^{d} w_i x_i + \alpha$,
Goal: find w s.t all constraints $y_i X_i \cdot w \geq 0$. Define a risk function and optimize it, where the loss is defined as $L(z, y_i) = -y_i z$ if $y_i z < 0$, else 0.
Therefore risk $R(w) = \sum_{i \in V} -y_i X_i \cdot w$

Decision boundary, a **hyperplane** in $\mathbb{R}^d$:
$H = \{\vec{\boldsymbol{x}} \in \mathbb{R}^d : f(\vec{\boldsymbol{x}}) = 0\} = \{\vec{\boldsymbol{x}} \in \mathbb{R}^d : \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} + \alpha = 0\}$

$\vec{\boldsymbol{w}}$ is the **normal** of the hyperplane,
α is the **offset** of the hyperplane from origin,
$\frac{f(\vec{\boldsymbol{x}})}{\|\vec{\boldsymbol{w}}\|}$ is the **signed distance** from the $\vec{\boldsymbol{x}}$ to hyperplane $\mathcal{H}$.

Perceptron algorithm,
Input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \{\pm 1\}$
while some $y_i \neq \text{sign}(\vec{\boldsymbol{w}} \cdot \mathbf{x}_i)$
  pick some misclassified $(\mathbf{x}_i, y_i)$
  $\vec{\boldsymbol{w}} \leftarrow \vec{\boldsymbol{w}} + y_i \mathbf{x}_i$

Given a **linearly separable data**, perceptron algorithm will take no more than $\frac{R^2}{\gamma^2}$ updates to **converge**, where $R = \max_i \|\mathbf{x}_i\|$ is the radius of the data,
$\gamma = \min_i \frac{y_i(\vec{\boldsymbol{w}} \cdot \mathbf{x}_i)}{\|\vec{\boldsymbol{w}}\|}$ is the margin.
Also, $\frac{\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}}}{\|\vec{\boldsymbol{w}}\|}$ is the signed distance from H to $\vec{\boldsymbol{x}}$ in the direction $\vec{\boldsymbol{w}}$.

Gradient descent view of perceptron, minimize margin cost function
$J(\vec{\boldsymbol{w}}) = \sum_i (-y_i(\vec{\boldsymbol{w}} \cdot \mathbf{x}_i))_+$ with $\vec{\boldsymbol{w}} \leftarrow \vec{\boldsymbol{w}} - \eta \nabla J(\vec{\boldsymbol{w}})$

Support Vector Machine (Lecture 3, 4)

Hard margin SVM,
This method makes the margin as wide as possible. The signed distance from the hyperplane to X_i is $\frac{f(\mathbf{x}_i)}{\|w\|}$ Hence the margin is
$\min_i \frac{1}{\|w\|} |w \cdot X_i + \alpha| \geq \frac{1}{\|w\|} \implies \min_{\vec{\boldsymbol{w}}} \|\vec{\boldsymbol{w}}\|^2$, such that
$y_i \vec{\boldsymbol{w}} \cdot \mathbf{x}_i \geq 1 (i = 1, \ldots, n)$
Soft margin SVM,
$\min_{\vec{\boldsymbol{w}}} \|\vec{\boldsymbol{w}}\|^2 + C \sum_{i=1}^{n} \xi_i$

Regularization and SVMs: Simulated data with many features $\phi(\vec{\boldsymbol{x}})$; C controls trade-off between margin $1/\|\vec{\boldsymbol{w}}\|$ and fit to data; Large C: focus on fit to data (small margin is ok). More overfitting. Small C: focus on large margin, less tendency to overfit. Overfitting increases with: less data, more features.

Decision Theory (Lecture 6)

Bayes Theorem: $\underbrace{\mathbb{P}(Y = C|X)}_{\text{Poster. Prob}} = \frac{\mathbb{P}(X|Y=C)\overbrace{\mathbb{P}(Y=C)}^{\text{Prior Prob.}}}{\mathbb{P}(X)}$
Assume $(\mathbf{X}, \mathbf{Y})$ are chosen i.i.d according to some probability distribution on $\mathcal{X} \times \mathcal{Y}$. **Risk** is misclassification probability:

$$\begin{aligned} R(r) &= \mathbb{E}[L(r(\mathbf{X}), \mathbf{Y})] = \mathbb{P}(r(\mathbf{X}) \neq \mathbf{Y}) \\ &= \sum_{\vec{\boldsymbol{x}}} \left[L(r(\vec{\boldsymbol{x}}), 1)\,\mathbb{P}(Y = 1|x) + L(r(x), -1)\,\mathbb{P}(Y = -1|X = \vec{\boldsymbol{x}}) \right] \times \mathbb{P}(\vec{\boldsymbol{x}}) \\ &= \mathbb{P}(Y = 1) \sum_{x} L(r(\vec{\boldsymbol{x}}), 1)\,\mathbb{P}(\vec{\boldsymbol{x}}|Y = 1) + \mathbb{P}(Y = -1) \sum_{x} L(r(\vec{\boldsymbol{x}}), -1)\,\mathbb{P}(\vec{\boldsymbol{x}}|Y = -1) \end{aligned}$$

Bayes Decision Rule is

$$r^*(x) = \begin{cases} 1, & \text{if } L(-1,1)\,\mathbb{P}(\mathbf{Y} = 1|x) > L(1,-1)\,\mathbb{P}(\mathbf{Y} = -1|x) \\ -1, & \text{otherwise.} \end{cases},$$

and the optimal risk (Bayes risk) $R^* = \inf_r R(r) = R(r^*)$

Generative and Discriminative Models (Lecture 6)

Discriminative models: $\mathbb{P}(\mathbf{X}, \mathbf{Y}) = \mathbb{P}(\mathbf{X})\,\mathbb{P}(\mathbf{Y}|\mathbf{X})$.
Estimate $\mathbb{P}(\mathbf{Y}|\mathbf{X})$, then pretend our estimate $\hat{\mathbb{P}}(\mathbf{Y}|\mathbf{X})$ is the actual $\mathbb{P}(\mathbf{Y}|\mathbf{X})$ and plug in bayes rule expression.

Generative model: $\mathbb{P}(\mathbf{X}, \mathbf{Y}) = \mathbb{P}(\mathbf{Y})\,\mathbb{P}(\mathbf{X}|\mathbf{Y})$.
Estimate $\mathbb{P}(\mathbf{Y})$ and $\mathbb{P}(\mathbf{X}|\mathbf{Y})$, then use bayes theorem to calculate $\mathbb{P}(\mathbf{Y}|\mathbf{X})$ and use discriminative model.

Gaussian class conditional densities $\mathbb{P}(\mathbf{X}|Y = +1)$,$\mathbb{P}(\mathbf{X}|Y = -1)$ (with the same variance), the posterior probability is **logistic**:
$\mathbb{P}(Y = +1|\vec{\boldsymbol{x}}) = \frac{1}{1+\exp(-\vec{\boldsymbol{x}} \cdot \vec{\boldsymbol{w}} - \beta_0)}$,
$\vec{\boldsymbol{w}} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$, $\beta_0 = \frac{\boldsymbol{\mu}_0' \Sigma^{-1} \boldsymbol{\mu} 0 - \boldsymbol{\mu}_1 \Sigma^{-1} \boldsymbol{\mu} 1}{2} + \log \frac{\mathbb{P}(Y=1)}{\mathbb{P}(Y=0)}$

Multivariate Normal Distribution (Lecture 7)

$\vec{\boldsymbol{x}} \in \mathbb{R}^d : p(x) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} e^{(-\frac{1}{2}(\vec{\boldsymbol{x}} - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\vec{\boldsymbol{x}} - \boldsymbol{\mu}))}$

QDA: Class-conditional densities $X_C \sim \mathcal{N}(\vec{\boldsymbol{\mu}}_C, \Sigma_C)$. Optimal decision rule $r^*(x)$ for 0–1 loss: Choose class $\mathbf{C}$ that maxes $\mathbb{P}(Y = C|X) \propto f_C(x)\pi_C$.
Parameters estimated via MLE:

LDA: Assumes equal covariance matrices across classes ($\Sigma_C = \Sigma$), simplifying to linear decision surfaces.

$\boldsymbol{\Sigma} = \mathbb{E}(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})'$
Symmetric: $\boldsymbol{\Sigma}_{i,j} = \boldsymbol{\Sigma}_{j_i}$
Non-negative diagonal entries: $\boldsymbol{\Sigma} i, i \geq 0$
Positive semidefinite: $\forall \mathbf{v} \in \mathbb{R}^d, \mathbf{v}' \boldsymbol{\Sigma} \mathbf{v} \geq 0$

Given a d-dimensaional Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,
matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$ and vector $\mathbf{b} \in \mathbb{R}^m$, define $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}$.
Then $\mathbf{Y} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top)$

Given a d-dimensaional Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,
with $\boldsymbol{\Sigma}$ positive definite,
$\mathbf{Y} = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\mathbf{X} - \boldsymbol{\mu}) \sim \mathcal{N}(\vec{\mathbf{0}}, \mathbf{I})$

MLE's

Maximum a posterior probability: the mode of the posterior. If uniform prior, MAP is MLE; if not uniform prior, MAP is Penalized MLE.

Prior: $\hat{\pi}_C = \mathbb{P}(Y = C) = \frac{N_C}{n}$
Mean: $\hat{\mu}_C = \mathbb{E}[\mathbf{X}|Y = C] = \frac{1}{N_C} \sum_{i:Y_i = C} X_i$
Covariance: $\hat{\Sigma}_C = \frac{1}{N_C} \sum_{i:Y_i = C} (X_i - \hat{\mu}_C)(X_i - \hat{\mu}_C)^\top$
Pooled Cov: $\hat{\Sigma} = \frac{1}{n} \sum_{C_k} \sum_{i:Y_i = C_k} (X_i - \hat{\mu}_{C_k})(X_i - \hat{\mu}_{C_k})^\top$

Discriminant Analysis (Lecture 7)

Discriminant Functions for LDA and QDA for class C, denoted $Q_C(\vec{\boldsymbol{x}})$ is:

$$Q_C(\vec{\boldsymbol{x}}) = \ln\left(\frac{f_{\vec{\boldsymbol{X}}|Y=C}(\vec{\boldsymbol{x}})\pi_C}{(2\pi)^{\frac{d}{2}}}\right) = -\frac{1}{2}(\vec{\boldsymbol{x}} - \boldsymbol{\mu}_C)^T \boldsymbol{\Sigma}_C^{-1}(\vec{\boldsymbol{x}} - \boldsymbol{\mu}_C) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_C| + \ln\frac{\pi_C}{(2\pi)^{\frac{d}{2}}}$$

Here we have, the class-conditional density $f_{\vec{\boldsymbol{X}}|Y=C}(\vec{\boldsymbol{x}})$, prior probability π_C, the mean vector of class C, and covariance matrix $\boldsymbol{\Sigma}_C$ of class C.

Linear Decision Function between Classes C and D: Compares classes by:

$$Q_C(\vec{\boldsymbol{x}}) - Q_D(\vec{\boldsymbol{x}}) = (\boldsymbol{\mu}_C - \boldsymbol{\mu}_D)^T \boldsymbol{\Sigma}^{-1} \vec{\boldsymbol{x}} - \frac{1}{2}(\boldsymbol{\mu}_C^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_C - \boldsymbol{\mu}_D^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_D) + \ln\frac{\pi_C}{\pi_D}.$$

This expression includes a linear term $(\boldsymbol{\mu}_C - \boldsymbol{\mu}_D)^T \boldsymbol{\Sigma}^{-1} \vec{\boldsymbol{x}}$, which describes how the decision boundary is formed linearly in the feature space, and a constant term that adjusts the boundary based on the means of the classes and their prior probabilities.

Multi-class LDA: The optimal class for a given feature vector $\vec{\boldsymbol{x}}$ is chosen by

Choose class C such that $Q_C(\vec{\boldsymbol{x}})$ is maximized.

Effectively determining the most likely class based on the feature measurements and statistical properties of each class.

Linear Regression (Lecture 10)

Objective Function: The goal in linear regression is to minimize the sum of squared residuals (RSS), expressed as:

$$\text{RSS}(\vec{\boldsymbol{w}}) = \sum_{i=1}^{n} (\vec{\boldsymbol{x}}_i^\top \vec{\boldsymbol{w}} + \alpha - y_i)^2 = \|\mathbf{X}\vec{\boldsymbol{w}} - \vec{\boldsymbol{y}}\|^2.$$

Where, $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$ is the **design matrix**, and $\vec{\boldsymbol{y}}$ is the **response vector**.
Minimize via Calculus: $\nabla \text{RSS}(\vec{\boldsymbol{w}}) = 2\mathbf{X}^\top \mathbf{X}\vec{\boldsymbol{w}} - 2\mathbf{X}^\top \vec{\boldsymbol{y}}$.
Setting this gradient to zero leads to the **normal equations**:

$$\mathbf{X}^\top \mathbf{X}\vec{\boldsymbol{w}} = \mathbf{X}^\top \vec{\boldsymbol{y}} \implies \vec{\boldsymbol{w}}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{\boldsymbol{y}},$$

$(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is known as the *Moore-Penrose pseudo-inverse* $X^\dagger$ of $\mathbf{X}$.
Projection Theorem also leads to the normal equations, by asserting that the solution $\vec{\boldsymbol{w}}^*$ projects the residuals orthogonally onto the column space of $\mathbf{X}$:

$$\mathbf{X}^\top (\vec{\boldsymbol{y}} - \mathbf{X}\vec{\boldsymbol{w}}) = 0 \implies \mathbf{X}^\top \mathbf{X}\vec{\boldsymbol{w}} = \mathbf{X}^\top \vec{\boldsymbol{y}},$$

Logistic Regression (Lecture 10, 11)

Fits "probabilities" to data and usually used for classification.
$\mathbb{P}(Y = 1|\vec{\boldsymbol{x}}_i) = \frac{1}{1+\exp(-\vec{\boldsymbol{w}}^\top \vec{\boldsymbol{x}}_i)} = \sigma(\vec{\boldsymbol{w}}^\top \vec{\boldsymbol{x}}_i)$
Logistic Loss: $L(z, y) = -y \ln z - (1 - y)\ln(1 - z)$

Cost Function: $J(\vec{w}) = -\sum_{i=1}^{n} L\left(\sigma(\vec{w}^\top \vec{x}_i), y_i\right)$
Find $\vec{w}^* = \arg\min J(\vec{w})$, Cost function is convex, and solved by gradient descent. Converges to max margin classifier.

Let $\sigma_i = \sigma(\vec{w}^\top \vec{x}_i)$ and $\vec{\sigma} = \begin{bmatrix}\sigma_1 & \cdots & \sigma_n\end{bmatrix}^\top$

$\nabla_{\vec{w}} \sigma_i = \sigma_i(1-\sigma_i)\vec{x}_i$
$\nabla_{\vec{w}} J(\vec{w}) = \mathbf{X}^\top(\vec{\sigma} - \vec{y})$
$\nabla^2_{\vec{w}} J(\vec{w}) = \mathbf{X}^\top \operatorname{diag}(\vec{\sigma}(1-\vec{\sigma}))\mathbf{X}$

Gradient descent:
$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \nabla_{\vec{w}} J(\vec{w}^{(t)})$: $O(nd)$ per step
Stochastic gradient descent:
$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \nabla L_i(\vec{w}^{(t)})$: $O(d)$ per step
Newton's method:
$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \left(\nabla^2 J(\vec{w}^{(t)})\right)^{-1} \nabla J(\vec{w}^{(t)})$

LDA vs. Logistic Regression

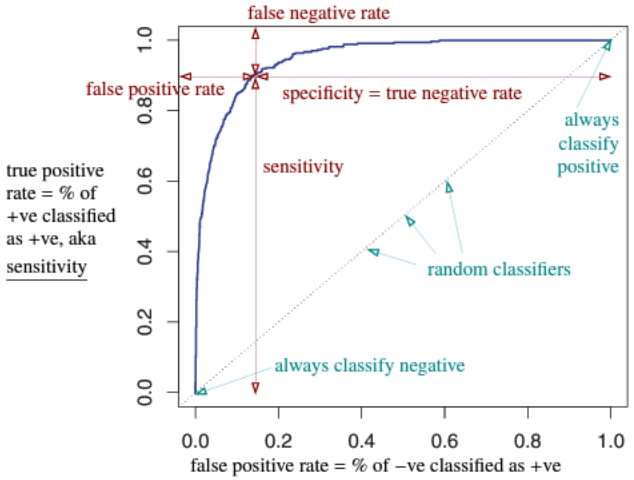
Advantages of LDA:

- For well-separated classes, LDA is stable; log. reg. is suprisingly unstable.
- > 2 classes? Easy & elegant; log. reg. needs modifying. (softmax regr.)
- slightly more accurate when classes nearly normal, especially if n is small.

Advantages of Log. Regression:

More emphasis on desc. boundary; always separates linearly separable pts.
More robust on some non-Gaussian distributions (e.g., dists w/ large skews.)
Naturally fits labels between 0 and 1.

ROC CURVES



Polynomial Regression (Lecture 11)

Replace each x_i with feature vector $\Phi(x_i)$ with all terms of degree p polynomial. e.g.,$\Phi(x_i) = \begin{bmatrix} x_{i1}^2 & x_{i1}x_{i2} & x_{i2}^2 & x_{i1} & x_{i2} & 1\end{bmatrix}^\top$

Log. Reg. + quadratic features = same form of posteriors as QDA.

Statistical Justification For Regression (Lecture 12)

Typical model of reality: $\forall X_i,\ y_i = g(X_i) + \epsilon_i : \epsilon_i \sim D'$ where D' has mean 0.

Ideal approach: choose $h(X_i) = \mathbb{E}_Y[Y|X = X_i] = g(X_i) + \mathbb{E}[\epsilon_i] = g(X_i)$

Least Squares Cost Function From MLE

Suppose $\epsilon_i \sim \mathcal{N}(0, \sigma^2) \implies y_i \sim \mathcal{N}(g(\vec{x}_i), \sigma^2)$. We're going to try to estimate $g(\vec{x})$, which is defined by some weights $\vec{w}$. The probability of y_i, given it's parameters for its distribution, is a pdf $f(y_i)$ & the log likelihood is:

$$\ell(g; \mathbf{X}, \vec{y}) = \ln\left(\prod_{i=1}^{n} f(y_i)\right) = \sum_{i=1}^{n} \ln\left(f(y_i)\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - g(\vec{x}_i)\right)^2 - C$$

Maximumizing this is equivalent to minimizing $\sum_{i=1}^{n}\left(y_i - g(\vec{x}_i)\right)^2 = \text{RSS}$

Logistic Loss from MLE

Consider $P[\vec{x}_i \in \text{class C}] = y_i$ as the actual probability, with $h(\vec{x}_i)$ as the predicted probability. For β hypothetical repetitions, where $y_i\beta$ samples belong to class C and $(1-y_i)\beta$ do not. The likelihood of h given the data is:

$$\mathscr{L}(h; \mathbf{X}, \vec{y}) = \prod_{i=1}^{n} h(\vec{x}_i)^{y_i\beta}(1 - h(\vec{x}_i))^{(1-y_i)\beta}$$

$$\ell(h; \mathbf{X}, \vec{y}) = \beta \sum_{i=1}^{n}\left[y_i \ln\left(h(\vec{x}_i)\right) + (1-y_i)\ln\left(1 - h(\vec{x}_i)\right)\right]$$

Maximizing the log likelihood $\ell(h)$ is equivalent to minimizing logistic losses.

The Bias-Variance Decomposition

There are 2 sources of error in a hypothesis h:

bias: error due to inability of hypothesis h to fit g perfectly.
variance: error due to fitting random noise in data, e.g., we fit linear g with a linear h, yet $h \neq g$.

Model: $X_i \sim D, \epsilon_i \sim D'$ and $y_i = g(X_i) + \epsilon_i$. Fit hypothesis h to $\mathbf{X}, \vec{y}$. Now h is a r.v., because of the random noise ϵ. Let z be an arbitrary pt and $\gamma = g(z) + \epsilon$
Now the risk fn when loss is the squared errror is:
$R(h) = \mathbb{E}[L(h(z), \gamma)] \leftarrow$ exp over possible training set X, y and values of γ
$R(h) = \mathbb{E}[(h(z) - \gamma)^2]$
$R(h) = \mathbb{E}[h(z)^2] + \mathbb{E}[\gamma^2] - 2\mathbb{E}[\gamma h(z)]$
$R(h) = \text{Var}(h(z)) + \mathbb{E}[h(z)]^2 + \text{Var}(\gamma) + \mathbb{E}[\gamma]^2 - 2\mathbb{E}[\gamma]\,\mathbb{E}[h(z)]$
$R(h) = \left(\mathbb{E}[h(z)] - \mathbb{E}[\gamma]\right)^2 + \text{Var}(h(z)) + \text{Var}(\gamma)$
$R(h) = \underbrace{\left(\mathbb{E}[h(z)] - \mathbb{E}[\gamma]\right)^2}_{\text{bias}^2 \text{ of method}} + \underbrace{\text{Var}(h(z))}_{\text{variance of method}} + \underbrace{\text{Var}(\epsilon)}_{\text{irreducible error}}$

- **Underfitting**: Too much bias
- **Overfitting**: Too much variance
- Training Error reflects bias but not variance
- Test error reflects both.

Linear Regression Regularization (Lecture 13)

Trading off bias/variance: some increase in bias can give big decrease in var.

Ridge regression is like $L2$ regularization:
$\hat{\vec{w}} = \arg\min_{\vec{w}} \|\vec{y} - \mathbf{X}\vec{w}\|^2 + \lambda\|\vec{w}\|^2$, for some cost function J on classifier h
Minimized w/ Calculus
$\hat{\vec{w}}^{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\vec{y}$

Lasso regression is like $L1$ regularization:
Minimized w/ Calculus (it's also a quadratic program)
While ridge regression leads to reduced, but rare non-zero values of the weights, Lasso regression forces some weights to be zero.

Bayesian analysis: Ridge regression is equivalent to a MAP estimate with a gaussian prior. Lasso regression is equivalent to a MAP estimate with a Laplace prior.

Misc

Centering X: This involves subtracting $\boldsymbol{\mu}^T$ from each row of $\mathbf{X}$. Symbolically, $\mathbf{X}$ transforms into $\bar{\mathbf{X}}$.
Decorrelating X: This process applies a rotation $\mathbf{Z} = \bar{\mathbf{X}}\mathbf{V}$, where $\text{Var}(\mathbf{R}) = \mathbf{V}\Lambda\mathbf{V}^T$. This step rotates the sample points to the eigenvector coordinate system.
Sphering: $\tilde{\mathbf{X}}$: applying transform $\mathbf{W} = \bar{\mathbf{X}}\text{Var}(\mathbf{R})^{-\frac{1}{2}}$
whitening X: centering + sphering, $\mathbf{X} \to \mathbf{W}$

Decision Trees (Lecture 14, 15)

Nonlinear method for classification or regression. Cuts x-space into rectangular cells. Works well with both categorial and quantiative features. They're fast, interpretable, robost to irrelevant features, but not best at prediction → high variance often mitigated by ensemble learning.

For classification the learning algorithm is a greedy, top-down learning heuristic. Let S be subset of sample point indices:

function GROWTREE(S)
  if all $y_i = C$ for all $i \in S$ and some class C **then**
    return *new* leaf(C)
  else
    Choose best splitting feature k and splitting value β
    $S_L = \{i \in S : \mathbf{X}_{ik} < \beta\}$
    $S_R = \{i \in S : \mathbf{X}_{ik} \geq \beta\}$
    return *new* node$\left(k, \beta, \text{GROWTREE}(S_L), \text{GROWTREE}(S_R)\right)$

Optimal Split Selection: Evaluate all possible (k, β) splits and choose the one that minimizes the weighted impurity cost:

$$J(S, k, \beta) = \frac{|S_L|J(S_L) + |S_R|J(S_R)}{|S_L| + |S_R|}$$

where entropy $J(S) = -\sum_c p_c \log_2(p_c)$, with $p_c = \frac{|\{i \in S : y_i = c\}|}{|S|}$
Information Gain, defined as $IG(S, k, \beta) = J(S) - J(S, k, \beta)$
IG maximization is equivalent to entropy minimization post-split.
Pruning: Grow tree too large; greedily remove each split whose removal improves validation performance on test data that lies on these splits.
For regression: Leafs stores mean response for training pts in that region. Different cost function: $J(S) = RSS = \frac{1}{|S|}\sum_{i \in S}(y_i - \mu_s)^2 = \text{Var}\left(\{y_i : i \in S\}\right)$
Ensemble Methods Decision trees often suffer from high variance, which can lead to overfitting. Ensemble methods, by combining multiple trees, aim to reduce this variance and improve predictive performance.

Bagging: Given n-point training samples, generate random subsamples of size n' by sampling *with replacement*. If $n' = n \to \sim 63.2\%$ are chosen. Idea: Points chosen j times have greater weight in cost when misclassified.
Random Forest: Enhance Bagging by reducing correlation between different trees with feature randomness at each split. This process ensures that the ensemble does not rely too heavily on any single feature and makes trees within the forest less identical.

Kernels (Lecture 16)

In many learning algorithms, the weights can be written as a linear combo of training points, that is, $\vec{w} = \mathbf{X}^\top \vec{\alpha} = \sum_{i=1}^n \alpha_i \vec{x_i}$, for some $\vec{\alpha} \in \mathbb{R}^n$.

We can subsitute this identity into alg, and optimize n dual weights α instead of the $\vec{w}$ of the primal problem. **Kernel Trick:** Enables learning algorithms to operate in high-dimensional space without explicitly computing the transformations.

Kernel Ride Regression

Transform the training set X using a kernel function $k(x,z)$, creating a kernel matrix K.

Solve $(K+\lambda I)a = y$ for the dual coefficients a, where λ is the regularization parameter.

Predictions for a new point z are made by $h(z) = \sum_{i=1}^n a_i k(x_i, z)$.

—

Center $\mathbf{X}$: $\vec{x_i} \leftarrow \vec{x}_i - \mu_x$, except ficticious dimension, $\vec{x}_{i,d+1} = 1$

Center $\vec{y}$: $\vec{y} \leftarrow \vec{y} - \mu_y$.

In ridge regression, recall the normal equations:

$$\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}'\right) \vec{w} = \mathbf{X}^\top \vec{y}$$

Where I' is the identity matrix with 0 at the position of the ficticious dimension so that we don't penalize the bias. In order for the condition that the weights be a linear combination of training points to be satisfied, we need to penalize the bias.

Now the objective function is: (subsiting $\vec{w} = \mathbf{X}^\top \vec{\alpha}$)

$$\left\| \mathbf{X}^\top (\mathbf{X}\vec{\alpha}) - \vec{y} \right\|^2 + \lambda \left\| \mathbf{X}^\top \vec{\alpha} \right\|^2$$

Training: Solve $\left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)\vec{\alpha} = \vec{y}$, for $\vec{\alpha}$.

Testing: Regression fn is $h(\vec{z}) = \vec{w}^\top \vec{z} = \vec{\alpha}^\top \mathbf{X}\vec{z} = \sum_{i=1}^n \alpha_i (\vec{x}_i^\top \vec{z})$

Let $k(\vec{x}, \vec{z}) = \vec{x}^\top \vec{z}$ be the kernel function.

Let $\mathbf{K} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{n \times n}$ be the kernel matrix with $\mathbf{K}_{ij} = k(\vec{x}_i, \vec{x}_j)$.

Kernel Ridge Regr. Alg:

Initialize the kernel matrix $\mathbf{K}$ ▷ $\mathcal{O}(n^2 d)$ time

Solve $(\mathbf{K} + \lambda \mathbf{I})\vec{\alpha} = \vec{y}$ ▷ $\mathcal{O}(n^3)$ time

for each test point z, compute **do**

$h(z) = \sum_{i=1}^n \alpha_i k(\vec{x}_i, z)$ ▷ $\mathcal{O}(nd)$ time per test pt

The Kernel Trick (aka Kernelization)

Polynomial kernel of degree p is $k(\vec{x}, \vec{z}) = (\vec{x}^\top \vec{z} + 1)^p$

Theorm: $(\vec{x}^\top \vec{z} + 1)^p = \Phi^T(\vec{x})\Phi(\vec{z})$ for some feature map Φ

Note now we can compute $\Phi^T(\vec{x})\Phi(\vec{z})$ in $\mathcal{O}(d)$ time instead of $\mathcal{O}(d^p)$

Gaussian (RBF) Kernel: $k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$, measures the similarity, or closeness, of x and z with a parameter σ controlling the width of the Gaussian.

Choosing σ in Gaussian Kernels:

Trade-off between bias and variance: larger σ leads to smoother models with higher bias and lower variance.

Selected based on validation performance, optimizing for generalization.

Neural Networks (Lecture 17)

Input Layer: $x_1, \ldots, x_d : x_{d+1} = 1$

Hidden Units: $h_1, \ldots, h_m : h_{m+1} = 1$

Output Layer: $z_1, \ldots, z_k$

Layer 1 weights $V \in \mathbb{R}^{m \times (d+1)}$ (row i, $V_i^\top$ contains all the weights from previous layer, coming into hidden unit i)

Layer 2 weights $W \in \mathbb{R}^{k \times (m+1)}$

$\vec{z} = \vec{s}(\mathbf{W}\vec{h}) = \vec{s}(\mathbf{W}\vec{s}_1(\mathbf{V}\vec{x}))$ where $\vec{s}_1$ is the activation function applied element wise to vector, with last component set to 1.

Acitvation Functions:

$$s(\gamma) = \sigma(\gamma) = \frac{1}{1+e^{-\gamma}} \implies \sigma'(\gamma) = \sigma(\gamma)(1-\sigma(\gamma))$$

$$s(\gamma) = \sigma_{\text{ReLU}}(\gamma) = \max(0,\gamma) \implies \sigma'_{\text{ReLU}}(\gamma) = \begin{cases} 1 & \gamma > 0 \\ 0 & \gamma \le 0 \end{cases}$$

1. Derive the gradients of the loss $L \in \mathbb{R}$ with respect to weight matrix $W \in \mathbb{R}^{n^{[l]} \times n^{[l+1]}}$ and bias *row* vector $b \in \mathbb{R}^{1 \times n^{[l+1]}}$ in the fully-connected layer, $\partial L/\partial W$ and $\partial L/\partial b$. You will also need to take the gradient of the loss with respect to the input of the layer $\partial L/\partial X$, which will be passed to lower layers, where $X \in \mathbb{R}^{m \times n^{[l]}}$ is the batched input. **Again, you must arrive at a solution that uses batched X and Z. Please express your solution in terms of $\partial L/\partial Z$, which you have already obtained in 4.1, where $Z = XW + 1b$ and $1 \in \mathbb{R}^{m \times 1}$ is a column of ones.** Note that $1b$ is a matrix (it's an outer product) whose each row is the row vector b so we are adding the same bias vector to each sample during the forward pass: this is the mathematical equivalent of numpy broadcasting.

Solution: Let L be a function operating on Z, where $Z = XW + 1b$. Then:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z}\frac{\partial Z}{\partial W} = X^T \frac{\partial L}{\partial Z} \quad (1)$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial Z}\frac{\partial Z}{\partial b} = 1^T \frac{\partial L}{\partial Z} \quad (2)$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z}\frac{\partial Z}{\partial X} = \frac{\partial L}{\partial Z} W^T \quad (3)$$

1. First, derive the gradient of the downstream loss with respect to the input of the ReLU activation function, Z. **You must arrive at a solution in terms of $\partial L/\partial Y$, the gradient of the loss w.r.t. the output of ReLU $Y = \sigma_{\text{ReLU}}(Z)$, and the batched input Z, i.e., where $Z \in \mathbb{R}^{m \times n}$.**

Hint: you are allowed to use operations like elementwise multiplication and/or division!

Solution: We know that $\sigma'_{\text{ReLU}}(\gamma) = \begin{cases} 0 & \gamma < 0, \\ 1 & \text{otherwise.} \end{cases}$

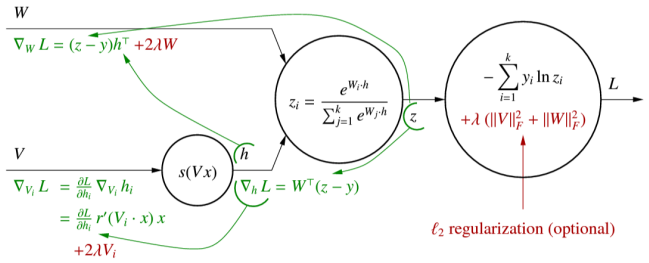
L is some function operating on Y, the output of ReLU, that is:

$L = f(Y) = f(\sigma_{\text{ReLU}}(Z))$.

Thus, by chain rule, the gradient of L with respect to Z is.

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial Z} = \frac{\partial L}{\partial Y} \circ \sigma'_{\text{ReLU}}(Z)$$

Where $\circ$ is the element-wise multiplication.



PCA: Principal Component Analysis (Lecture 20)

Goal: Identify k principal directions to maximize variance capture after projecting data onto these components, enhancing computational efficiency and reducing overfitting by eliminating irrelevant features.

Assumptions: Assume $\mathbf{X}$ is centered, i.e., $\sum_{i=1}^n \mathbf{x}_i = 0$.

Orthogonal Projection of a point $\mathbf{x}$ onto a vector $\mathbf{w}$ is given by:

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} \cdot \mathbf{w}}{\|\mathbf{w}\|^2} \mathbf{w}$$

Aim: Find the best $\mathbf{w}$ that captures the maximum variance after projecting.

Principal Coordinates and Components: Given orthonormal directions $\mathbf{v}_1, \ldots, \mathbf{v}_k$, the *projection* of $\mathbf{x}$ onto these vectors is:

$$\tilde{\mathbf{x}} = \sum_{i=1}^k (\mathbf{x} \cdot \mathbf{v}_i)\mathbf{v}_i$$

where $\mathbf{x} \cdot \mathbf{v}_i$ are the *principal coordinates*. The *principal components* are unit eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_d$ of the covariance matrix $\hat{\Sigma} = \frac{1}{n}\mathbf{X}^\top \mathbf{X}$, associated with its eigenvalues $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_d$.

function PCA ALGORITHM($\mathbf{X}$, $\mathbf{X}_{\text{test}}$)

Center $\mathbf{X}$

Compute unit eigenvectors and eigenvalues of $\mathbf{X}^\top \mathbf{X}$

Choose k (optionally based on the size of eigenvalues)

Select the top k eigenvectors $\mathbf{v}_{d-k+1}, \ldots, \mathbf{v}_d$ for the best k-dimensional subspace

Compute the k principal coordinates $\mathbf{x} \cdot \mathbf{v}_i$ for each training/test point

Derivation 2: Variance Maximization: Find $\mathbf{w}$ that maximizes

$$\text{Var}(\{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_n\}) = \frac{1}{n}\frac{\|\mathbf{X}\mathbf{w}\|^2}{\|\mathbf{w}\|^2} = \frac{1}{n}\frac{\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}}{\mathbf{w}^\top \mathbf{w}} = \frac{1}{n}\text{Raylegh Quotient}(\mathbf{X}^\top \mathbf{X})$$

Theorm: $\vec{v}_d$ maximizes this variances.

Derivation 3: Projection Distance Minimization:

Find direction $\mathbf{w}$ that minimizes the mean of squared projection distance.

$$\sum_{i=1}^n \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2 = \sum_{i=1}^n \left\| \mathbf{x}_i - \frac{\mathbf{x}_i \cdot \mathbf{w}}{\|\mathbf{w}\|^2}\mathbf{w} \right\|^2$$

$$= \sum_{i=1}^n \left(\|\mathbf{x}_i\|^2 - \left(\mathbf{x}_i \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|}\right)^2 \right) = C - n(\text{variation from derivation 2})$$

Minimizing mean squared project distance ≡ Maximizing variance.

SVD, Clustering (Lecture 21)

SVD: Problem with PCA is that computing $X^\top X$ is expensive, taking $\mathcal{O}(nd^2)$.

$$[\mathbf{X}]_{n \times d} = [\mathbf{U}]_{n \times d}\,[\mathbf{D}]_{d \times d}\,[\mathbf{V}]_{d \times d}^T = \sum_{i=1}^d \delta_i \vec{u}_i \vec{v}_i^T \quad \text{when } n \ge d$$

- The number of non-negative singular values corresponds to the rank of $\mathbf{X}$.
- $\vec{v}_i$ is an eigenvector of $\mathbf{X}^T X$ with eigenvalue δ_i^2.
- We can find the k greatest singular values & corresp vectors in $\mathcal{O}(ndk)$
- Row i of **UD** gives the principle coordinates of sample $\mathbf{X}_i$

Clustering: Partition data into K clusters so pts in a cluster are more similar than across clusters.

K-Means Clustering goal is to parition pts into k disjoint clusters to minimize the sum of squared distances from each point to its cluster center.

$$\boxed{\text{Find } \vec{y} \text{ that minimizes } \sum_{i=1}^k \sum_{y_j = i} \left\| \mathbf{X}_j - \vec{\mu}_i \right\|^2}$$

Where, $\vec{\mu}_i = \frac{1}{n_i}\sum_{y_j = i} \mathbf{X}_j$ is the mean of cluster i, given n_i pts in cluster i.

Problem is NP-hard, $\mathcal{O}(nk^n)$. Use heuristic, alternating between 2 steps.

1. Fixing labels y_j and updating cluster means μ_i to min the obj. fn.
2. Fixing $\vec{\mu}_i$ and updating labels y_j to minimize the objective function.

Repeats until no assignments change. Both steps decrease obj. fn, *unless* they change nothing. Hence alg. must terminate. We can show through calculus:

1. The optimal $\vec{\mu}_i$ is the mean of its cluster

2. The optimal $\vec{y}$ assigns each pt $\mathbf{X}_j$ to nearest center $\vec{\mu}_i$

Initalization Stratigies

Forgy Method: Choose k random sample pts to be inital $\vec{\mu}_i$'s; go to step 2.
Random Parition: randmly assign each sample pt to a cluster; go to step 1.
K-Means++: Like Forgy, but biased distribution, choosing centers far apart.

Equivalent Object Function: called *The Within-Cluster Variation*: (no $\vec{\mu}_i$)
$\min_{\vec{y}} \sum_{i=1}^{k} \frac{1}{n_i} \sum_{y_j=i} \sum_{y_m=i} \left\| \mathbf{X}_j - \mathbf{X}_m \right\|^2$. Minimum is twice the prev. obj. fn.

Hierarchical Clust: Builds tree of clusters; every subtree is cluster. Built using

- **Agglomerative (Bottom-up)**: Start with each point as a cluster, then merge the closest pairs until one cluster remains.
- **Divisive (Top-down)**: Start with all points in one cluster, then iteratively split the most diverse cluster.

We need a distance fn for clusters A, B. Here are 4 common ones :

Complete Linkage: Max distance between any points in two clusters.
Single Linkage: Min distance between any points in two clusters.
Average: Avg dist between all points in clusters.
$d(A,B) = \frac{1}{|A||B|} \sum_{w \in A} \sum_{x \in B} d(w,x)$
Centroid Linkage: dist between centroids of clusters. $d(A,B) = d(\mu_A, \mu_B)$

Greedy Agglomerative Alg: repeatedly *fuse* the 2 clusters that minimize $d(A,B)$. Naive takes $\mathcal{O}(n^3)$, done in $\mathcal{O}(n^2)$ for complete and single linkage
Dendrogram: Illustration of the cluster hierarchy. Vertical axis encodes all the linkage distances. Horizontal axis has no meaning. The complete linkage tends to give balanced trees.

Learning Theory (Lecture 23)

Learning theory explores how machine learning algs generalize from training data to unseen data. It involves understanding the constraints needed for a learner to make predictions on new data effectively.

Range Space: Defined by (P, H) where $\boldsymbol{P}$ is the set of all train/test points and $\boldsymbol{H}$ is the hypothesis class or a set of classifiers. Each hypothesis $h \in H$ identifies which points are predicted to be in a particular class.

Risk and Empirical Risk: *Risk (Generalization Error)* $R(h)$: The prob. that a hypothesis h misclassifies a randomly drawn point from distribution $\mathcal{D}$.
Empirical Risk $\hat{R}(h)$: The proportion of training data misclassified by h, which better approximates $R(h)$ as the size of the training set increases.

Hoeffding's Inequality: Tells us probability of bad Risk estimate.

$$\mathbb{P}\left(|\hat{R}(h) - R(h)| > \epsilon\right) \leq 2e^{-2\epsilon^2 n},$$

where ϵ is the error margin and n is # of samples. This shows that larger n expoenntially decays the chance of deviation between $\hat{R}(h)$ and $R(h)$.

Empirical Risk Minimization (ERM): Selects the hypothesis $\hat{h}$ that minimizes $\hat{R}(\hat{h})$ among H. ERM is infeasible to pick optimal hypothesis.

Dichotomies and Sample Complexity: A dichotomy created by a hypothesis h over set X categorizes points into class C or its complement. The number of possible dichotomies, Π, influences the likelihood of misleadingly low empirical risks. If H induces Π dichotomies, then:

$$\mathbb{P}\left(\text{at least one dichotomoy has } |\hat{R} - R| > \epsilon\right) \leq \delta = 2\Pi e^{-2\epsilon^2 n}$$

with confidence $1-\delta$, $\forall h \in H$, $|\hat{R}(h) - R(h)| \leq \epsilon = \sqrt{\frac{1}{2n} \ln \frac{2\Pi}{\delta}}$. This formulation reveals that increasing n and reducing Π enhances risk estimates' reliability.

Optimal Risk Approximation: If $h^* \in H$ minimizes $R(h^*)$, then with high probability $1-\delta$, the selected hypothesis $\hat{h}$ that minimizes $\hat{R}(\hat{h})$ fulfills:

$$R(\hat{h}) \leq \hat{R}(\hat{h}) + \epsilon \leq \hat{R}(h^*) \leq R(h^*) + 2\epsilon$$

Sample Complexity: The minimum # of samples required to ensure that ϵ approximation accuracy with confidence $1-\delta$ is:

$$n \geq \frac{1}{2\epsilon^2} \ln \frac{2\Pi}{\delta}$$

AdaBoost: Adaptive Boosting (Lecture 24)

An ensemble method for classification (or regression) that trains T learners on weighted samples points, gives bigger votes to more accurate learners, and increases weights of misclassified points (pay more attention to this point), with the goal of *reducing bias*. If every learner has err < 0.5, $M(\vec{x})$ training err $\to 0$ as $T \to \infty$.

Input: $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\vec{y} \in \mathbb{R}^n$ with $y_i = \pm 1$ (not natural fit if labels arent bin).
Idea:

Training Classifiers: Sequentially train T classifiers $G_1, \ldots, G_T$
Weight Adjustment: Weight for training pt $\vec{x}_i$ in iteration t for G_t grows accoring to how many of $G_1, \ldots, G_{t-1}$ misclassified it.
Meta Learner: is a linear combobination of all T classifiers. Output is not binary; apply the sign function for binary classification.

Risk Minimization: Risk of the meta learner is average exponential loss $L(\rho, \ell) = e^{-\rho \ell}$, and minimize it to adjust β_T and select the optimal classifier G_T for each round:

$$\begin{aligned} n * \text{Risk} &= \sum_{i=1}^{n} L(M(\vec{x}_i), y_i) = \sum_{i=1}^{n} \prod_{t=1}^{T} \exp\left(-y_i \beta_t G_t(\vec{x}_i)\right) \\ &= \sum_{i=1}^{n} w_i^{(T)} \exp\left(-y_i \beta_T G_T(\vec{x}_i)\right) \\ &= e^{-\beta_T} \sum_{y_i = G_T(\vec{x}_i)} w_i^{(T)} + e^{\beta_T} \sum_{y_i \neq G_T(\vec{x}_i)} w_i^{(T)} \\ &= e^{-\beta_T} \sum_{i=1}^{n} w_i^{(T)} + (e^{\beta_T} - e^{-\beta_T}) \sum_{y_i \neq G_T(\vec{x}_i)} w_i^{(T)} \end{aligned}$$

where $w_i^{(T)} = \prod_{t=1}^{T-1} \exp\left(-y_i \beta_t G_t(\vec{x}_i)\right)$ is the cumulative weight adjusted for each sample up to classifier $T-1$. Therefore, the optimal classifier G_T minimizes the sum of weights of misclassified points.

Recursive Weight Update: To focus training on misclassified points:

$$w_i^{(T+1)} = w_i^{(T)} \exp\left(-y_i \beta_T G_T(\vec{x}_i)\right) = \begin{cases} w_i^{(T)} \exp\left(-\beta_T\right) & \text{if } y_i = G_T(\vec{x}_i) \\ w_i^{(T)} \exp\left(\beta_T\right) & \text{if } y_i \neq G_T(\vec{x}_i) \end{cases}$$

Optimization for β_T: Solve for β_T by setting the derivative of the risk with respect to β_T equal to zero, deriving that:

$$\beta_T = \frac{1}{2} \ln\left(\frac{1 - \text{err}_T}{\text{err}_T}\right),$$

where err_T is the weighted proportion of misclassified instances by G_T. Therefore a perfect learner ($\text{err}_T = 0$) has $\beta_T = \infty$

---

function AdaBoost($\mathbf{X}, \vec{y}$)
  Initialize weights $w_i = \frac{1}{n}$ for all i
  for $t \leftarrow 1$ to T **do**
    Train classifier G_t on $(\mathbf{X}, \vec{y})$, using weights $\vec{w}$
    Calculate error err_t
    Calculate β_t using err_t
    Update weights: $w_i \leftarrow w_i \exp\left(-y_i \beta_t G_t(x_i)\right)$
  Return metalearner $h(\vec{z}) = \text{sign}(\sum_{t=1}^{T} \beta_t)$

K-Nearest Neighbor (Lecture 25)

Idea: Given query point $\vec{q}$, find the k training pts nearest to $\vec{q}$.
Theorem: as $n \to \infty$, the 1-NN error rate is $< 2B - B^2$ where B = bayes risk.
Theorem: As $n \to \infty$, $k \to \infty$, and $\frac{k}{n} \to 0$ (n grows faster than k), k-NN error rate converges to B

Exhaustive k-NN: Algorithm:
Given query point q:

- Scan through all n training pts, computing (squared) distances to q.
- Maintain a max-heap with the k closest training points seen so far.
- Time to train classifier: $\mathcal{O}(0)$ (no time)
- Query Time: $\mathcal{O}(nd + n \log k)$, expected $\mathcal{O}(nd + k \log(n) \log(k))$ if random point order.

Voronai Diagrams:
Let X be a point set. the Voronoi cell of $w \in X$ is
Vor $w = \{p \in \mathbb{R}^d : \|p - w\| \leq \|p - v\| \ \forall v \in X\}$
The *Voronoi diagram* of X is the set of X's Voronoi cells.

K-Nearest Neighbor (Lecture 25)

Principle: For a query point $\vec{q}$, classify based on the majority label among its k nearest training points.

Theorem: Under certain conditions:

- For 1-NN, as $n \to \infty$, the error rate is less than twice the Bayes risk B.
- For k-NN, as $n \to \infty$, $k \to \infty$, and $\frac{k}{n} \to 0$, the error rate converges to B.

Exhaustive k-NN Algorithm:

- Compute squared distances from $\vec{q}$ to all n training points.
- Use a max-heap to maintain the k closest points.
- *Complexity*: Training: $\mathcal{O}(0)$, Query: $\mathcal{O}(nd + n \log k)$.

Voronoi Diagrams:

- The Voronoi cell for a point $w \in X$ is defined as: $\text{Vor}(w) = \{p \in \mathbb{R}^d : \|p - w\| \leq \|p - v\|, \forall v \in X\}$.
- Used for 1-NN to locate the nearest training point to a query point.
- *Complexity*: For low dimensions $(2-5)$, efficient; for higher, consider k-d trees.

k-d Trees:

- Space-partitioning data structure to organize points in k-dimensional space.
- Splits along axis-aligned hyperplanes for efficient nearest neighbor search.
- *Building*: $\mathcal{O}(nd \log n)$, *Query*: sublinear, depends on dimensionality.

Key Advantages:

- Non-parametric and simple.
- Effective for small datasets and low dimensionality.

Challenges:

- High query time in large datasets.
- Curse of dimensionality: performance degrades as dimensionality increases.