Boosting/Ensemble Methods

- Generally boosting to reduce variance
- Improve error rate of a **weak base learning algorithm**
- Final classifier combines all the weak classifiers to create a single **strong classifier w** small test error and train error
- Update is done **sequentially**

AdaBoost algorithm:

For t=1,2,…,T
Construct reweighting D_t for the data
Run A(D_t) to get a (weak) classifier h_t
Calculate a weight $\alpha_t \in \mathbb{R}$
Output: $h(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

- How to set α_t:

Define $\varepsilon_t = \text{error}_{D_t}(h_t) = \sum_{i=1}^{n} D_t(i)\mathbb{I}[h_t(x_i) \neq y_i]$

Define $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ — Higher weight for h_t with lower error

- How to construct D_t:

$D_1(i) = \frac{1}{n}$, for every i — Start with uniform D

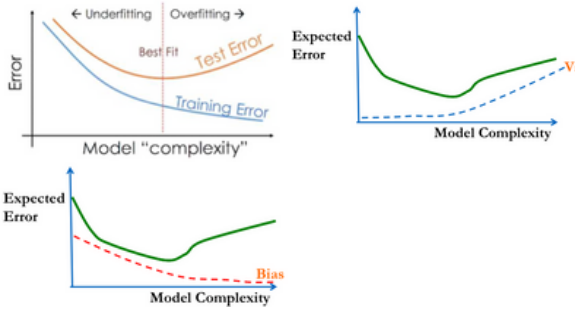
for $t \geq 1$, $D'_{t+1}(i) = D_t(i)e^{-\alpha_t}$ if $h_t(x_i) = y_i$ — Decrease D(i) if h_t correct on (x_i, y_i)
$D'_{t+1}(i) = D_t(i)e^{\alpha_t}$ if $h_t(x_i) \neq y_i$ — Increase D(i) if h_t wrong on (x_i, y_i)

$D_{t+1}(i) = \frac{D'_{t+1}(i)}{Z_{t+1}}$, where $Z_{t+1} = \sum_{i=1}^{n} D'_{t+1}(i)$ — Renormalize so it sums to 1

- Training error converges to zero exponentially quickly (AdaBoost guarantee)
- AdaBoost maximizes margins of the predictions, even after training error is 0, thus it avoids overfitting (Increasing margin of voted predictions on data, even after reaching 0 training error)

Bias/Variance Trade-off

- **Overfitting**: training error low, test error high
  - Model too complex
  - Not enough training data
- **Underfitting**: training error and test error high
  - Model too simple



Model simple -> High bias, low variance
Model too complex -> High variance, low bias
Expected error = Bias + Variance

Model Selection

- **Cross-Validation**
  - Split data into train, test, validation
    - Tune hyperparams of model only on validation set
    - Pick model with lowest validation error and use it to eval test error on test set
- **K-Fold Cross Validation**
  - Train on data S - S_i (Each of size K)
  - Test on S_i
- **Cross-Validation error:**

$$\frac{1}{K}\sum_{i=1}^{K} \text{error}_{S_i}(\hat{h}_i)$$

Theoretically, if k = N -> Leave one out CV

Neural Networks/Backpropagation

Perceptron: $z = h\left(\sum_{i=1}^{D} w_i x_i + w_0\right) = h(w^T x)$

Common Activation Functions + Derivative:

ReLU $h(x) = \max(0, x)$

Sigmoid $h(x) = \frac{1}{1+e^{-x}}$ deriv: $\sigma(x)(1-\sigma(x))$

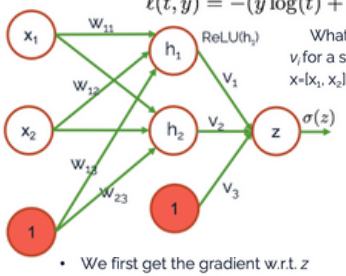
Tanh $h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ deriv: $1 - \tanh^2(x)$

Sigmoid range: [0,1]
Tanh range: [-1, 1]
Limitation of Perceptron: Cannot handle non linearly separable data e.g. XOR problem
Use Gradient Descent to minimize loss function since it is differentiable

Backpropagation:

- Apply chain rule, start from output and work backwards
- Example:

$$\ell(t, y) = -(y\log(t) + (1-y)\log(1-t))$$



What is the gradient of w_{ij} and v_i for a single training example $x=[x_1, x_2]$?

- We first get the gradient w.r.t. z

$$\frac{\partial \ell}{\partial z} = \frac{\partial}{\partial z}(-y\log(t) - (1-y)\log(1-t)) = \frac{-y}{t}\frac{\partial t}{\partial z} - \frac{y-1}{1-t}\frac{\partial t}{\partial z}$$

where $\frac{\partial t}{\partial z}$ is the derivative of the sigmoid. We know $\frac{\partial t}{\partial z} = t(1-t)$.

Plug it in, we have

$$\frac{\partial \ell}{\partial z} = (t)(1-t)\left(\frac{-y}{t} - \frac{y-1}{1-t}\right) = (t)\left(\frac{y(t-1)}{t} - (y-1)\right) = y(t-1) - t(y-1) = yt - y - yt + t = t - y$$

- Then we get the gradient w.r.t. v_i

$$\frac{\partial z}{\partial v_1} = \frac{\partial}{\partial v_1}(v_1\text{ReLU}(h_1) + v_2\text{ReLU}(h_2) + v_3) = \text{ReLU}(h_1)$$

By the chain rule, we have

$$\frac{\partial \ell}{\partial v_1} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial v_1} = (t-y)\text{ReLU}(h_1)$$

Similarly, we can get the gradient w.r.t v_2 and v_3

Finally we get the gradient w.r.t. w_{11}

$$\frac{\partial z}{\partial h_1} = \frac{\partial}{\partial h_1}(v_1\text{ReLU}(h_1) + v_2\text{ReLU}(h_2) + v_3) = v_1\text{ReLU}'(h_1)$$

$$\frac{\partial h_1}{\partial w_{11}} = \frac{\partial}{\partial w_{11}}(w_{11}x_1 + w_{12}x_2 + w_{13})$$

-> By the chain rule

$$\frac{\partial \ell}{\partial w_{11}} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial h_1}\frac{\partial h_1}{\partial w_{11}} = (t-y)v_1x_1\text{ReLU}'(h_1)$$

Convolutional Neural Networks(CNN)

- **Hyperparameters:**
  - Stride size **S**
  - Padding size **P**
  - Filter size **F**
  - Number of filters **K**
  - Size of layer(# of units)
- **Formula to calculate output size after filter**

Output: $W_2 \times H_2 \times K$

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- **Filters**
  - Preserves the spatial structure of data
- **Pooling**
  - Reduce dimensionality of input

Pattern Mining/Association rules/Freq items

- **Support** (Fraction of data that satisfy rule)
  - **Monotonic** (Subset of a frequent itemset must also be frequent)
- **Confidence** (Accuracy)

Clustering, K-Means

- **Partition-Based Clustering**
  - Partition data into k groups where all examples in a group are close
  - **K-Means**
    - Initialise with k randomly chosen centroids
    - Assign each sample to closest centroid
    - Recompute cluster centroids
    - Repeat until no changes in assignments

Gaussian Mixture Models/EM algorithm

Gaussian Mixture Models (Probabilistic modelling approach)

- Assume that the data are generated from a mixture of k multi-dimensional Gaussians, where each component is has parameters: $\mathcal{N}(\mu_k, \Sigma_k)$ — covariance matrix (next slide); average (vector)
- For each data point:
  - Pick component Gaussian randomly with probability p(k)
  - Draw point from that Gaussian randomly by sampling from: $\mathcal{N}(\mu_k, \Sigma_k)$

$$p(x) = \sum_{k=1}^{K} p(k)p(x|k) \quad \text{(probability)}$$
$$= \sum_{k=1}^{K} p(k)p\left(x|x \sim N(\mu_k, \Sigma_k)\right)$$

EM for GMM

- Suppose we have a current estimate of all parameter values w_k, μ_k, Σ_k
- Use these to estimate probabilities of cluster memberships $\Gamma(x_i) = [\gamma_1(x_i), \ldots, \gamma_K(x_i)]$ — E-Step
- Now compute the **expected** log-likelihood using estimated probabilities of cluster memberships

$$\mathbb{E}_{\mathbf{z}} \log p(D, \mathbf{z}|w, \mu, \Sigma) = \sum_{i=1}^{n}\sum_{k=1}^{K} \gamma_k(x_i)\left[\log(w_k) + \log(\mathcal{N}(x_i|\mu_k, \Sigma_k))\right]$$

- Maximize the expected log-likelihood over all w_k, μ_k, Σ_k to update parameters — M-Step
- Repeat

E Step Details

- Suppose we have a current estimate of all parameter values w_k, μ_k, Σ_k
- Use these to estimate probabilities of cluster memberships $\Gamma(x_i) = [\gamma_1(x_i), \ldots, \gamma_K(x_i)]$

$$\gamma_k(x_i) \leftarrow \frac{w_k N(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} w_j N(x_i|\mu_j, \Sigma_j)}$$

Hierarchical Clustering

- **Distance Measures** between clusters
  - Single-linkage/nearest neighbor:
    - $D(C_i, C_j)$ = **minimum** $\|\underline{x} - \underline{x}'\|$, such that $\underline{x} \in C_i$, $\underline{x}' \in C_j$ ⇒ can produce long thin clusters
  - Complete-linkage/furthest neighbor:
    - $D(C_i, C_j)$ = **maximum** $\|\underline{x} - \underline{x}'\|$, such that $\underline{x} \in C_i$, $\underline{x}' \in C_j$ ⇒ is sensitive to outliers
  - Average linkage:
    - $D(C_i, C_j)$ = **average** $\|\underline{x} - \underline{x}'\|$, such that $\underline{x} \in C_i$, $\underline{x}' \in C_j$ ⇒ compromise between the two

**Note: Always taking the minimum of distance measures

Principal Component Analysis(PCA)

- In higher-dimension:
  - Given n samples: $\underline{x}_1, \ldots, \underline{x}_n \in \mathbb{R}^d$

$$\text{Mean}(x_{,j}) = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

$$\text{Cov}(x_{,j}, x_{,j'}) = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \text{Mean}(x_{,j}))(x_{ij'} - \text{Mean}(x_{,j'}))$$

- Simplifying: Let X be the **centered** data matrix:

$$X_{ij} = x_{ij} - \text{Mean}(x_{,j})$$

- Then the **sample covariance matrix** is

$$\Sigma = \frac{1}{n}X^\top X$$

Linear algebra: Eigenvalues, Eigenvectors

- Square matrix A (k × k)

$$Av = \lambda v$$

Square matrix — Eigenvector (||v||=1) — Eigenvalue

- At most k eigenvalues
- If A is also **symmetric**, then all distinct eigenvalues have eigenvectors that are **orthogonal**: $v_i^\top v_j = 0$
- If U is the matrix with columns equal these vectors
- And Λ is a diagonal matrix with all the eigenvalues on diagonal

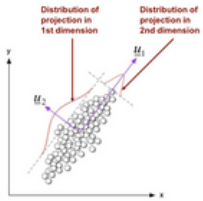
$$A = U\Lambda U^\top$$

PCA Interpretation

- Let $\underline{u}_1, \underline{u}_2, \ldots$ be orthonormal eigenvectors of Σ
- With corresponding eigenvalues $\lambda_1, \lambda_2, \ldots$
- And order them so that $\lambda_1 \geq \lambda_2 \geq \ldots$
- $\underline{u}_1$ called the **1st principal component**
- $\underline{u}_2$ called the **2nd principal component**
- etc.
- **Another interpretation**:
  - 1st basis vector ($\underline{u}_1$) ***minimizes*** mean-squared reconstruction error

$$\text{minimize} \quad \frac{1}{n}\sum_{i=1}^{n}\|\underline{x}_i - (v^\top \underline{x}_i)v\|^2$$

  - 2nd basis ($\underline{u}_2$) again ***minimizes*** reconstruction error, subject to the constraint that it be orthogonal to previous bases, etc.



M Step Details

- Now compute the **expected** log-likelihood using estimated probabilities of cluster memberships

$$\mathbb{E}_{\mathbf{z}} \log p(D, \mathbf{z}|w, \mu, \Sigma) = \sum_{i=1}^{n}\sum_{k=1}^{K} \gamma_k(x_i)\left[\log(w_k) + \log(\mathcal{N}(x_i|\mu_k, \Sigma_k))\right]$$

- Maximize the expected log-likelihood over all w_k, μ_k, Σ_k to update parameters

$$w_k \leftarrow \frac{1}{n}\sum_{i=1}^{n}\gamma_k(x_i)$$

$$\mu_k \leftarrow \frac{\sum_{i=1}^{n}\gamma_k(x_i)x_i}{\sum_{i=1}^{n}\gamma_k(x_i)}$$

$$\Sigma_k \leftarrow \frac{\sum_{i=1}^{n}\gamma_k(x_i)(x_i - \mu_k)(x_i - \mu_k)^\top}{\sum_{i=1}^{n}\gamma_k(x_i)}$$

Updates are exactly what you'd guess they should be

Principal Component Analysis(PCA) cont

Applying PCA

- New data vectors are formed by projecting the data onto the first few principal components (**m eigenvectors with highest eigenvalues**)
- Original instance: Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \square \geq \lambda_d$ Eigenvectors:

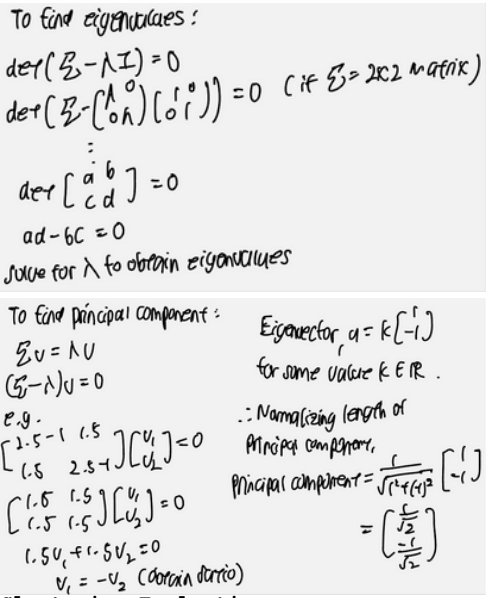
$$\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad U = [\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_d], \quad \underline{u}_j = \begin{bmatrix} u_{1,j} \\ \vdots \\ u_{d,j} \end{bmatrix}$$

- Transformed instance:

$$\underline{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_m \end{bmatrix} \qquad p_1 = \underline{u}_1 \bullet \underline{x} \quad \ldots \quad p_m = \underline{u}_m \bullet \underline{x}$$

- If **m=d** then data is transformed, if **m<d** then transformation is lossy, and dimensionality is reduced
- Can then run a learning algorithm on the projected data
- For a test instance $\underline{x}$, first projected (the same way) then apply the classifier

PCA Question - Answers:



Clustering Evaluation

Purity:

The purity of each cluster *i* in G is determined by the number of examples of class *j* (N_j) inside each cluster:

$$purity(C, G) = \frac{1}{N} \sum_{i=1}^{K} \max_j N_j \in G_i$$

Normalized Mutual Info Gain:

- **Normalized mutual information gain:**
  - Measures the amount of information by which our knowledge about the classes (C) increases when we are told what the clusters (G) are

$$NMI(C,G) = \frac{I(C;G)}{[H(C)+H(G)]/2} = \frac{\sum_c \sum_g p(c,g) \log \frac{p(c,g)}{p(c)p(g)}}{[-\sum_c p(c)\log p(c) - \sum_c p(g)\log p(g)]/2}$$

$$I(C;G) = H(C,G) - H(C) - H(G) = H(C) - H(C|G) = H(G) - H(G|C)$$

Class label entropy: measure how labeled data are distributed in different classes

$$H(C) = -\sum_{i=1}^{C} p(c_i)\log p(c_i) = -\sum_{i=1}^{C} \frac{C_i}{N} \log \frac{C_i}{N}$$

Cluster label entropy: measure how labeled data are distributed in different clusters

$$H(G) = -\sum_{i=1}^{G} p(g_i)\log p(g_i) = -\sum_{i=1}^{G} \frac{G_i}{N} \log \frac{G_i}{N}$$

Knowing the cluster membership

Conditional entropy in one cluster

$$H(C|G_i) = -\sum_{j=1}^{C} p(c_j|g_i)\log p(c_j|g_i) = -\sum_{j=1}^{C} \frac{N_{ij}}{N_i} \log \frac{N_{ij}}{N_i}$$

The number of data labeled with c_j in cluster g_i

The number of data in cluster g_i

Conditional entropy across all clusters

$$H(C|G) = -\sum_{j=1}^{C} p(g_i)H(C|G_i) = -\sum_{i=1}^{G} \frac{N_i}{N} \sum_{j=1}^{C} \frac{N_{ij}}{N_i} \log \frac{N_{ij}}{N_i} = -\sum_{i=1}^{G} \sum_{j=1}^{C} \frac{N_{ij}}{N} \log \frac{N_{ij}}{N_i}$$

Extra:

Number of parameters in CNN layer:
(Filter height x Filter width + 1) x Number of filters -> (+1) to account for bias term for each filter.

Probability Basics and Linear Algebra

- Marginal probability $P(A) = \sum_b P(A, B = b)$
- Conditional probability $P(A \mid B) = \frac{P(A,B)}{P(B)}$

$$P(A,B) = P(A \mid B)P(B)$$

- Independence: P(A, B) = P(A)P(B)

Equivalently: P(A | B) = P(A) or P(B | A) = P(B)

- Mutual independence:
- Every $I \subset \{1,2,\ldots,n\}$ and $J \subset \{1,2,\ldots,n\}$ have

$$P\left(\bigcap_{i\in I} A_i \cap \bigcap_{j\in J} A_j\right) = P\left(\bigcap_{i\in I} A_i\right) P\left(\bigcap_{j\in J} A_j\right)$$

- Conditional independence:
- P(A ∧ B | C) = P(A | C) P(B | C)
- Equivalently: P(A | B ∧ C) = P(A | C) or P(B | A ∧ C) = P(B | C)
  - Independence does not imply conditional independence, or vice versa
- Expectation
  - Discrete: $E[X] = \sum_x x \ \ p(x)$
  - Continuous: $E[X] = \int_x x \ \ p(x)dx$
  - Axioms:
    - E[aX+b] = aE[X] + b
    - E[h(X)] = $\sum_x h(x) \ \ p(x)$
    - $E[h(X)] = \int_x h(x)p(x)dx$
- Variance
  - Var(X) = $E[X^2] - (E[X])^2$
  - Standard deviation, $\sigma = \sqrt{Var(X)}$
  - Axioms:
    - Var(aX+b) = $a^2 \ \ Var(X)$

Perceptron stuff

Theorem: Perceptron makes at most $\frac{r^2+1}{\gamma^2}$ updates and the final (w_m, b_m) is correct on the data set

- r = max value of x in x-axis (furthest x-value data point)

Large Language Models(LLMs)

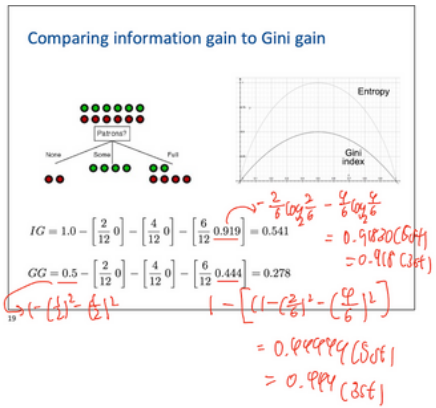
- Sequence-to-Sequence architecture
- Autoregressive(One by one input)
- Attention mechanism (Attention is a weighted sum)
  - No learnable parameters
  - 3 learnable matrices (Query Q, Key K, Value V)
  - **Self-attention: Q = K = V**

K-Nearest Neighbours

- Assumption: Close inputs have similar labels
- Use distance metric and take majority vote idea for classification, take average of k labels for regression
- Have to calculate distance between current and all other data in training set
- KNN is a non-parametric algorithm
- **Problem:**
  - kNN training is fast, but prediction is very slow due to computation of distance metric(esp for data with high dimensionality)
- **Higher values of k:** smoother decision boundary(less complex functions)
- **Lower values of k:** More expressive and complex functions
- Finding right balance: **Find k** that minimizes the **validation error**(set aside some of the data to test on after training)
- **Distance metrics:**
  - Manhattan distance (L1-norm)
  - Euclidean distance (L2-norm)
  - Minkowski distance (L-p norm)

Decision Trees

- **Entropy:** $H(X) = -\sum p(x)\log_2 p(x)$
- $Gain(S,A) = Entropy(S) - \sum_{a \in values(A)} \frac{|S_{A=a}|}{|S|} Entropy(S_{A=a})$
- **Note:** Calculate new entropy after splitting, then use this new entropy to calculate info gain / gini
- $Gain(S,A) = \overbrace{Gini(S)}^{original} - \sum_{a \in values(A)} \frac{|S_{A=a}|}{|S|} Gini(S_{A=a})$
- Example:



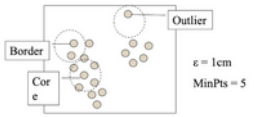
PURDUE UNIVERSITY — DBSCAN: The Algorithm

- Randomly select a point **p**
- Retrieve all points directly density-reachable from **p** w.r.t. **ε** and **MinPts**
- Put **p** together with these in a cluster (merge clusters if they're already in a cluster)
- Continue the process until all of the points have been processed.

The clusters this finds satisfy:
- Core points will be uniquely clustered into density-connected components
- Border points will be assigned to cluster containing a core point they are close to (if it's close to multiple clusters, it's assigned to the earliest formed cluster it's close to)
- Noise/outlier points aren't assigned to any cluster

PURDUE UNIVERSITY — DBSCAN: Density-based Spatial Clustering of Applications with Noise

- Relies on a *density-based* notion of cluster: A *cluster* is defined as a maximal set of **density-connected points**
- Discovers clusters of arbitrary shape in spatial databases with noise



Core Point: A point is a core point if it has more than MinPts points within ε.

Border Point: A point which has fewer than MinPts within ε but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core point or border point.

Spectral Clustering
- **Pro:** Can find non-spherical clusters
- **Weakness:** O(n^3) to find eigenvalues

Density-based Clustering
- Can handle noise and outliers

- Two parameters:
  - **ε**: Maximum radius of the neighborhood
  - ***MinPts***: Minimum number of points in an ε-neighborhood of that point
- $N_\varepsilon(p)$: ***{q belongs to D | dist(p,q) <= ε}***
- Directly density-reachable: A point ***p*** is directly density-reachable from a point ***q*** w.r.t. ***ε***, ***MinPts*** if
  - 1) ***p*** belongs to $N_\varepsilon(q)$
  - 2) core point condition:
    $|N_\varepsilon(q)|$ >= ***MinPts***