Supervised Learning: Find a mapping f from input features x to target outputs t using **labelled** training data (input-output pairs).
Unsupervised Learning: Find interesting patterns in the data, e.g., find clusters, where **no target labels are given** (only inputs are given).
Reinforcement Learning: Find a sequence of actions that produce a high reward signal in an environment.

Training Set: Used to learn the (*parameters* of) the mapping f from input features to the predicted output
Validation Set: Used to choose *hyperparameters* or model settings that are not provided by by the learning algorithm
Test Set: Used to estimate how well the model will *generalize* to new data and how well the model will perform when deployed

---

Training:
k-Nearest Neighbours: No training necessary; just store the data
Decision Trees: Greddy method: find the optimum split at each node, where "optimum" is determined by a loss, e.g., information gain
Linear Regression and Classification: Find weights w that minimize a cost function, e.g., mean square error or average cross entropy loss across the training set

---

$x_j^{(i)}$ represents the j-th feature of the i-th training example
Hypothesis (model, predictor) is a function f that maps input x to output prediction y
Voroni diagram partitions space into regions closest to each data
kNN: k too small may overfit (sensitive to noise), k too large would underfit (fail to capture regularities)
$k < \sqrt{n}$, choose k with highest validation accuracy
We use test data to estimate generalization error
60% training, 20% validation and test
We normalize each dimension to zero mean and unit variance
Curse of dimensionality: data not dense in high dimensions, and hard to calculate distances. Most points are approximately equally distanced, squared Euclidean distance becomes dominated by the average behavior, making distances converge
Overfitting: High training accuracy, poor test accuracy; model too complex, small data set, noise in data; regularization, better hyperparameters, better model
Underfitting: Low training and test accuracy; model too simple, bad hyperparameters, bad model; adjust hyperparameters, regularization, normalization

---

Decision Tree Output: Most common value for discrete classification, or mean value for regression
Use greedy heustic to construct decision tree based on (highest) information gain
More certain outcome has lower entropy, i.e. not uniformly distributed
Entropy characterizes uncertainty of random variable
Conditional Entrypy characterizres uncertainty of random variable given another random variable
Information Gain measures informativeness of variable
Process: Pick a feature, compare all possible splits, choose the one with highest IG, repeat recursively
We may minimize squared error instead: $SE = \sum_{i=1}^{N}(f(x^{(i)}) - t^{(i)})^2$

A model is a set of assumptions (restrictions) about the structure of f
The model or architecture restricts the hypothesis f space
Goal of Linear Regression: find w (weights) and b (bias) minimizing $\mathcal{E}(w,b) = \mathcal{E}(\mathbf{w})$
Let X be matrix with $N \times (D+1)$ shape with each $x^{(i)}$, then $\mathbf{y} = \mathbf{Xw}$ **Gradiant Descent** is more general than matrix inversion for finding $\mathbf{w}$
Each GD update costs $O(ND)$, $\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) = \mathbf{w} - \frac{\alpha}{N}\mathbf{X}^T(\mathbf{Xw} - \mathbf{t})$, or $w_i \leftarrow w_i - \alpha\frac{\partial \mathcal{E}}{\partial w_i}$, where initially $\alpha = 0.002$, $w_i = b = 0$
If α too small, training is slow, if α large, diverges

We may generalize linear regression by mapping input feature to another space, by feature mapping (or basis expansion): $\psi(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$, and treat $\mathbb{R}^d$ as input space, e.g. polynomial $\psi(x) = (1, x, ..., x^M)^T$, then $y = \psi(x)^T\mathbf{w}$
Adjust the degree of polynomial to avoid underfitting or overfitting
regularizer or **penalty** is function that qualifies how much we prefer one hypothesis (penalize large weights)
Both polynomial degree M and λ are hyperparameters
λ large cause underfit, λ small cause overfit
We want each feature's values to be indicators or orderable
ML Algorithm Design: 1. Choose model; 2. Define loss function; 3. Choose regularizer; 4. Fit model by minimizing regularized cost, using **optimization algorithm**.
$t = 1$ for positive example, $t = 0$ or -1 for negative example
For **Linear Binary Classification Model**, let $z = \mathbf{w}^T x$, then $y = 1$ if $z \geq 0$, else $y = 0$, decision boundary being hyperplane $z = 0$
Given small training set, go over each data point to find restrictions of w_i, if resion in weight space satisfying all constraints is non-empty, then this **feasible region** makes the problem (in)-**festible** and the training set **linearly separable**
Use square error loss function would cause correctly classified points with high loss; use indicator loss function would cause gradiant be zero e.w.; use sigmoid plus square loss won't update much for very wrongly classified example, i.e. poor gradient signal
We use sigmoid to smooth the output of each choice of $\mathbf{w}$ between 0 and 1, then use cross entropy loss function to penalize the model for being wrong
$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})\mathbf{x}^{(i)}$, same as linear regression
hypothesis space $\mathcal{H}$ set of all possible h a model can represent, i.e. all linear functions for linear regression
Members of $\mathcal{H}$ with algorithm's preference determine **inductive bias**
Algorithm is **parametric** if $\mathcal{H}$ is defined by a set of parameters (linear and logistic regression, neutral network; counter-example kNN)
Previously we used **full batch gradient descent** of $\frac{\partial \mathcal{E}}{\partial w}$, we may estimate $\frac{\partial \mathcal{E}}{\partial w}$ using subset of data if N too large, via **stochastic gradient descent**: computing average of a small number of examples called **mini-batch** with a **batch size**.
inner loop is **iteration** (each mini-batch), outer is **epoch** (one update)
num_iter = num_epoch $*$ N/batch_size

Accuracy: $\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[t^{(i)} = y^{(i)}]$
Error: $1 -$ Accuracy

Euclidean:

$$\left|\left|x^{(a)} - x^{(b)}\right|\right|_2 = \sqrt{\sum_{j=1}^{d}(x_j^{(a)} - x_j^{(b)})^2}$$

Cosine Similarity:

$$\text{cosine}(x^{(a)}, x^{(b)}) = \frac{x^{(a)} \cdot x^{(b)}}{\left|\left|x^{(a)}\right|\right|_2 \left|\left|x^{(b)}\right|\right|_2}$$

Mean: $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} = \mu$

Variance: $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \bar{x})^2$

Standard Deviation: $\sigma = \sqrt{\sigma^2}$

Normalization: $\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$

Entropy:

$$\begin{aligned} H(X) &= -\mathbb{E}_{X\sim p}[\log_2 p(X)] \\ &= -\sum_{x\in X} p(x)\log_2 p(x) \end{aligned}$$

$$H(X,Y) = -\sum_{\substack{x\in X\\ y\in Y}} p(x,y)\log_2 p(x,y)$$

$$H(Y|x) = -\sum_{y\in Y} p(y|x)\log_2 p(y|x)$$

$$\begin{aligned} H(Y|X) &= \sum_{x\in X} p(x)H(Y|x) \\ &= -\sum_{\substack{x\in X\\ y\in Y}} p(x,y)\log_2 p(y|x) \end{aligned}$$

$$H(X,Y) = H(Y,X) = H(X|Y) + H(Y)$$

$$H \geq 0, H(Y|Y) = 0, H(Y|X) \leq H(Y)$$

$$H(Y|X) = H(Y) \text{ if } X \text{ and } Y \text{ ind.}$$

Information Gain:
$IG(Y|X) = H(Y) - H(Y|X)$

Linear Model:
$y = f(x) = \sum_{j=1}^{D} w_j x_j + b = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$
where $\tilde{\mathbf{w}}_1 = b, \tilde{\mathbf{w}}_i = w_{i-1}, \tilde{\mathbf{x}}_1 = 1, \tilde{\mathbf{x}}_i = x_{i-1}$
Squared error loss function:
$\mathcal{L}(y,t) = \frac{1}{2}(y-t)^2$
$y - t$ is called residual
Cost (average loss) function:

$$\begin{aligned} \mathcal{E}(w,b) &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y^{(i)}, t^{(i)}) \\ &= \frac{1}{2N}(\mathbf{y} - \mathbf{t})^T(\mathbf{y} - \mathbf{t}) \\ &= \frac{1}{2N}||\mathbf{Xw} - \mathbf{t}||_2^2 \end{aligned}$$

$\frac{\partial \mathcal{E}}{\partial w} = \frac{1}{N}\sum_{j=1}^{N}(y^{(i)} - t^{(i)})x^{(i)}$
$\frac{\partial \mathcal{E}}{\partial b} = \frac{1}{N}\sum_{j=1}^{N}(y^{(i)} - t^{(i)})$
$\frac{\partial \mathcal{L}}{\partial w_j} = (y-t)x_j$
$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = (y-t)\mathbf{x}$

$$\begin{aligned} \nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) &= \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})\mathbf{x}^{(i)} \\ &= \frac{1}{N}\mathbf{X}^T(\mathbf{Xw} - \mathbf{t}) \end{aligned}$$

$\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) = 0$: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$

L^2 **Regularizer**:
$\mathcal{R}(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2 = \frac{1}{2}\sum_{j=1}^{D} w_j^2$
Regularized cost function:
$\mathcal{E}_{\text{reg}}(\mathbf{w}) = \mathcal{E}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w})$

Logistic / Sigmoid function:
$y = \sigma(z) = \frac{1}{1+e^{-z}}$
Both sigmoid and square loss:
$\frac{\partial \mathcal{L}}{\partial w_j} = (y-t)y(1-y)x_j$
Cross Entropy Loss:

$$\begin{aligned} &\mathcal{L}_{\text{CE}}(y,t) \\ &= \begin{cases} -\log(y) & \text{if } t = 1 \\ -\log(1-y) & \text{if } t = 0 \end{cases} \\ &= -t\log(y) - (1-t)\log(1-y) \end{aligned}$$

$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_j} = (y-t)x_j$

3rd column cont., batch size too large = expensive, too small = noisy
center and normalize to avoid ravines

Graph of kNN: Draw tangent line at each midpoint, connect regions with same class
Graph of Decision Tree: Draw axis-aligned lines to split regions
IG of Decision Tree: First calculate the entropy $H(Y)$ with probability (percentages) of each class, similarly for $H(Y|\text{right})$, $H(Y|\text{left})$, then $IG = H(Y) - p(\text{right})H(Y|\text{right}) - p(\text{left})H(Y|\text{left})$
Shape: # of rows $\times$ # of columns

Linear Regression: Square error loss function, with average loss cost function
Logistic Regression: Sigmoid function, with cross entropy loss function
Decision Tree: Greedy method, with information gain loss function
kNN: No training, with Euclidean distance for lower dimension D

Multi-class Classification
One-hot vector $t \in \mathbb{R}^K$ has 1 at the index of the correct class, 0 elsewhere, for K targets
For $y \in \mathbb{R}^K$, $\sum_{k=1}^{K} y_k = 1$
For input $x \in \mathbb{R}^D$, do linear prediction for each z_i, $z = Wx + b$, then $y_k = \text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$, their sum is 1, each $y_k \in [0,1]$

$$z = Wx + b$$

$$y = \text{softmax}(z) = \frac{e^z}{\sum_{j=1}^{K} e^{z_j}}$$

$$\mathcal{L}_{\text{CE}}(y,t) = -\sum_{k=1}^{K} t_k\log(y_k) = -t^T\log(y)$$

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_k} &= \frac{\partial \mathcal{L}_{\text{CE}}}{\partial z_k}\frac{\partial z_k}{\partial w_k} \\ &= (y_k - t_k)x \end{aligned}$$

$$\mathcal{E}(W,b) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\text{CE}}(y^{(i)}, t^{(i)})$$

$$w_k \leftarrow w_k - \frac{\alpha}{N}\sum_{i=1}^{N}(y_k^{(i)} - t_k^{(i)})x^{(i)}$$

Linear models can only separate data into 2 half-planes, does not work with XOR

Neural Network
Multi-Class: $x_1, ..., x_D$ as inputs, w_{ij} connecting x_j to y_i as connection
$b_1, ..., b_K$ as bias, f = softmax, y = softmax$(Wx + b)$ as output
Multi-Layer: Input layer, hidden layers, output layer (input layer doesn't count in n-layer n.n.)
Input size: num of features, hidden size: hyperpar., output size: num of classes
One layer:

$$\begin{aligned} h_j &= f(\textstyle\sum_{i=1}^{D} w_{j,i}^{(1)}x_i + b_j^{(1)}) \\ h &= f(W^{(1)}x + b^{(1)}) \\ z_k &= \textstyle\sum_{j=1}^{K} w_{k,j}^{(2)}h_j + b_k^{(2)} \\ z &= W^{(2)}h + b^{(2)} \\ y &= \text{softmax}(z) \end{aligned}$$

Multiple layer:

$$\begin{aligned} h^{(1)} &= f(W^{(1)}x + b^{(1)}) \\ h^{(\ell)} &= f(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}) \\ z &= W^{(L)}h^{(L-1)} + b^{(L)} \\ y &= \text{softmax}(z) \end{aligned}$$

Common f: Sigmoid, Tanh, ReLU
Sigmoid: problematic due to gradient signal at extreme inputs, pos only
Tanh: Sigmoid but centered at 0, pos and neg
ReLU: $f(x) = \max(0, x)$, often used, pos, problematic if bias too large and neg (activation always 0)
N.N.: learning features s.t. becomes lin. sep. after $L - 1$ layers; final layer as linear classifier
Expressive power: Deep linear networks with no activ. func. have same expr. power as linear regression
is universal approximator with nonlinear activ. func., e.g. single layer with 2^D hidden units
Backpropagation
For $\mathcal{L} = \frac{1}{2}(\sigma(wx+b) - t)^2$,
$\frac{\partial \mathcal{L}}{\partial w} = (\sigma(wx+b) - t)\sigma'(wx+b)x$ and
$\frac{\partial \mathcal{L}}{\partial b} = (\sigma(wx+b) - t)\sigma'(wx+b)$
So, more efficiently $\frac{\partial \mathcal{L}}{\partial y} = y - t$, $\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial y}\sigma'(z)$
$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z}x$, $\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$
Let $\bar{y}$ denote $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}$ called error signal
Computing predictions: $z = wx + b$, $y = \sigma(z)$, $\mathcal{L} = \frac{1}{2}(y-t)^2$
Computing gradients: $\bar{y} = y - t$, $\bar{z} = \bar{y}\sigma'(z)$, $\bar{w} = \bar{z}x$, $\bar{b} = \bar{z}$
Multiclass Logistic Regression with 2 features and 2 classes (computation graph):
$x_1, x_2 \to z_1, z_2 \to y_1, y_2 \to \mathcal{L}$, $w_{11}, b_1, w_{12} \to z_1$; $w_{21}, b_2, w_{22} \to z_2$; $t_1, t_2 \to \mathcal{L}$
$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$
$\bar{t} = \bar{x}\frac{\mathrm{d}x}{\mathrm{d}t} + \bar{y}\frac{\mathrm{d}y}{\mathrm{d}t}$
E.g. $\frac{\partial \mathcal{L}}{\partial b_1^{(1)}} = (\frac{\partial \mathcal{L}}{\partial y_1}\frac{\partial y_1}{\partial h_1} + \frac{\partial \mathcal{L}}{\partial y_2}\frac{\partial y_2}{\partial h_1})\frac{\mathrm{d}h_1}{\mathrm{d}z_1}\frac{\partial z_1}{\partial b_1^{(1)}}$
Forward pass: $z = W^{(1)}x + b^{(1)}$, $h = \sigma(z)$, $y = W^{(2)}h + b^{(2)}$, $\mathcal{L} = \frac{1}{2}\sum_k(y_k - t_k)^2 = \frac{1}{2}||y - t||^2$
Backward pass: $\bar{\mathcal{L}} = 1$, $\bar{y} = \bar{\mathcal{L}}(y - t)$, $\bar{W}^{(2)} = \bar{y}h^T$, $\bar{b}^{(2)} = \bar{y}$, $\bar{h} = W^{(2)T}\bar{y}$, $\bar{z} = \bar{h} \circ \sigma'(z)$
If $\bar{z_j} = \sum_k \bar{y_k}\frac{\partial y_k}{\partial z_j}$, then $\bar{z} = \frac{\partial y}{\partial z}^T\bar{y}$, with Jacobian matrix $\frac{\partial y}{\partial z}$
For softmax, $\frac{\partial y_i}{\partial z_j} = \begin{cases} y_i(1 - y_j) & \text{if } i = j \\ -y_iy_j & \text{if } i \neq j \end{cases}$

Bias Variance Decomposition
Decompose generalization error into: variance, bias, and irreducible error
Variance: Error from sensitivity to small fluctuations in the training data. Low Variance: less spread out. High Variance = overfitting
Bias: Error from poor assumptions in the model. Low Bias: more towards the center. High Bias = underfitting
Irreducible Error (Bayes Error): Error due to noise in the problem
Treat hypothesis y as a random variable.
Lemma. Best prediction is $y_* = \mathbb{E}[t|x]$.
Decompose $\mathbb{E}[(y-t)^2|x]$ ignoring $|x$:

$$\begin{aligned} \mathbb{E}[(y-t)^2] &= \mathbb{E}[(y - y_*)^2] + \text{Var}(t) \\ &= \mathbb{E}[y_* - y]^2 + \text{Var}(y - y_*) + \text{Var}(t) \end{aligned}$$

I.e., expected prediction error = bias (perform on average), variance (amount of variability in predictions), and Bayes error (irreducible error from data)

Ensembling methods: combine multiple models performs better than individual models
begging (bootstrap aggregation): train independently on different subsets of data, then average the predictions
Bayes error: unchanged; Bias: unchanged; Variance: reduced by factor $1/m$
Take $\mathcal{D}$ with n examples; Generate m new datasets, each sample n training examples from $\mathcal{D}$ with replacement; Averaging the predictions
$y_{\text{bagged}} = \mathbb{I}(\sum_{i=1}^{m}\frac{y_i}{m} > 0.5)$
Datasets not indep., not precisely $1/m$ reduc.
Random forests = bagged decision trees, but choose a random set of d input features for each node of decision tree
Decreases correlation between classifiers by adding randomness

---

Probabilistic Classifiers
For binary classification, let Y be the hypothesis, $Y \sim \text{Bernoulli}(\theta)$ for unknown $\theta \in [0,1]$
$p(y_i|\theta) = \theta^{y_i}(1-\theta)^{1-y_i}$
If y_i are i.i.d. Bernoullis, then $p(y_1, ..., y_N|\theta) = \prod_{i=1}^{N} p(y_i|\theta) = \prod_{i=1}^{N}\theta^{y_i}(1-\theta)^{1-y_i}$
We may find θ by given y_i and maximizing $p(y_1, ..., y_N|\theta)$
Likelihood function $L(\theta) = p(y_1, ..., y_N|\theta)$
maximum likelihood criterion:
pick $\hat{\theta}_{ML} = \arg\max_{\theta\in[0,1]} L(\theta)$

Work with log-likelihoods in practice: $\ell(\theta) = \log L(\theta) = \sum_{i=1}^{N} y_i \log\theta + (1-y_i)\log(1-\theta)$

Let $N_H = \sum_i y_i, N_T = N - \sum_i y_i$, then set $\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \frac{N_H}{\theta} - \frac{N_T}{1-\theta} = 0$

Obtain $\hat\theta_{ML} = \frac{N_H}{N_H+N_T} = \frac{N_H}{N}$

maximum likelihood estimation: define a model that assigns a probability to a dataset, maximize the likelihood (minimize cross-entropy)

$\log p(y_i, x_i; \theta) = (1-x_i)(y_i\log\theta_0 + (1-y_i)\log(1-\theta_0)) + x_i(y_i\log\theta_1 + (1-y_i)\log(1-\theta_1)) + (x_i\log\pi + (1-x_i)\log(1-\pi))$

$\log p(\mathcal{D};\theta) = \sum_{i=1}^{N}\log p(y_i, x_i;\theta)$

Decompose summation into 3 parts, take derivative of each part, set derivative to zero, solve for θ_0, θ_1, π

E.g.: $p(\mathcal{D}|\theta)$

$$= \prod_{i=1}^{N} p(x_1^{(i)}, x_2^{(i)}, y^{(i)};\theta)$$

$$= \prod_{i=1}^{N} p(y^{(i)}|x_1^{(i)}, x_2^{(i)})p(x_1^{(i)}, x_2^{(i)})$$

$$= \prod_{i=1}^{N}(\theta_{x_1^{(i)},x_2^{(i)}})^{y^{(i)}}(1-\theta_{x_1^{(i)},x_2^{(i)}})^{1-y^{(i)}}$$

$$\pi_{x_1^{(i)},x_2^{(i)}}$$

Then take log, decompose into 3 sums, for $\theta_{0,0}$, consider the first 2 sums only, set derivative to zero,

$$\frac{\partial\ell}{\partial\theta_{0,0}} = \frac{\partial}{\partial\theta_{0,0}}\sum$$

$$\mathbb{I}[x_1^{(i)} = 0, x_2^{(i)} = 0, y^{(i)} = 1]\log\theta_{0,0}$$

$$+\mathbb{I}[x_1^{(i)} = 0, x_2^{(i)} = 0, y^{(i)} = 0]\log(1-\theta_{0,0})$$

$$= \frac{N_{0,0;p}}{\theta_{0,0}} - \frac{N_{0,0;N}}{1-\theta_{0,0}}$$

$$\hat{\theta_{0,0}} = \frac{N_{0,0;p}}{N_{0,0}}$$

$$\ell(\pi) = N_{0,0}\log\pi_{0,0} + \ldots$$

$$+ N_{1,1}\log(1 - \ldots - \pi_{1,0}) + C_0$$

$$\frac{\partial\ell}{\partial\pi_{0,0}} = 0 = \frac{N_{0,0}}{\pi_{0,0}} - \frac{N_{1,1}}{1-\pi_{0,0}-\pi_{0,1}-\pi_{1,0}}$$

$N_{0,0}\pi_{1,1} = N_{1,1}\pi_{0,0}, N_{0,1}\pi_{1,1} = N_{1,1}\pi_{0,1}$

$N_{1,0}\pi_{1,1} = N_{1,1}\pi_{1,0}$

Discriminative classifier: learns a mapping from inputs to outputs, e.g. logistic regression, neural networks; model $p(c|x)$ directly (estimate parameters directly from labelled examples)

Generative model: Model $p(x,c), p(x|c)$, i.e. distribution of inputs characteristic of class (Bayes classifier)

Bayes rule: $p(y|x) = \frac{p(y)}{p(x)}p(x|y)$

E.g. sample c from Bernuolli $p(c)$; sample $x_1, \ldots, x_D$ from $p(x|c)$ $(c = 1, c = 0)$

Learn via MLE

Discriminative: Given x, predict $p(c|x)$

Generative: Model $p(x,c)$, given $p(c)$ can compute $p(x|c)$ Binary Bag-of-words Features: $x_i = 1$ if word i appears in the document, 0 otherwise

Inference:

$$p(c|x) = \frac{p(x,c)}{p(x)} = \frac{p(x|c)p(c)}{p(x)}$$

$$\text{Posterior} = \frac{\text{Class likelihood} \times \text{Class Prior}}{\text{Evidence}}$$

If we want to compare $p(c=0|x)$ with $p(c=1|x)$, it suffices to compare $p(x|c=0)p(c=0)$ with $p(x|c=1)p(c=1)$

Evidence:

$$p(x) = \sum_c p(x|c)p(c)$$

$$= p(x|c=1)p(c=1) + p(x|c=0)p(c=0)$$

$p(c, x_1, \ldots, x_D)$ is enough to obtain $p(c)$ and $p(x|c)$ (using $2^{D+1} - 1$ entries)

Naive Bayes classifier assumes x_i are conditionally independent given c

$p(x_1, \ldots x_D|c) = \prod_{i=1}^{D} p(x_i|c)$

$p(c, x_1, \ldots, x_D) = p(c)\prod_{i=1}^{D} p(x_i|c)$

$P(c=1) = \pi, P(x_j = 1|c = i) = \theta_{ji}$

A directed graphical model (Bayesian network): joint distr. factorizes as a product of condi. distr. for each variable given parent

$$\ell(\theta) = \sum_{i=1}^{N}\log p(c^{(i)}, x^{(i)})$$

$$= \sum_{i=1}^{N}\log(p(x^{i}|c^{(i)})p(c^{(i)}))$$

$$= \sum_{i=1}^{N}\log(p(c^{(i)}) + \prod_{j=1}^{D} p(x_j^{(i)}|c^{(i)}))$$

$$= \sum_{i=1}^{N}\log p(c^{(i)})$$

$$+ \sum_{j=1}^{D}\sum_{i=1}^{N}\log p(x_j^{(i)}|c^{(i)})$$

where the first sum is Bernoulli log-likelihood of labels ($p(c) = \pi^{c}(1-\pi)^{1-c}, \hat\pi$ = pos/tot, $p(c)$ Ber.), second inner of feature x_j (decompose to learn each θ_{jc} separately, $p(x_j|c)$ Bern., $\hat{\theta_{jc}}$ = num of word j in pos/pos num)

$$\theta_{jc} = p(x_j^{(i)} = 1|c) = \theta_{jc}^{x_j^{(i)}}(1-\theta_{jc})^{1-x_j^{(i)}}$$

$$\sum_{i=1}^{N}\log p(x_j^{(i)}|c^{(i)}) = \sum_{i=1}^{N} c^{(i)}$$

$$\left\{x_j^{(i)}\log\theta_{j1} + (1-x_j^{(i)})\log(1-\theta_{j1})\right\}$$

$$+ \sum_{i=1}^{N}(1-c^{(i)})$$

$$\left\{x_j^{(i)}\log\theta_{j0} + (1-x_j^{(i)})\log(1-\theta_{j0})\right\}$$

Inference: Compute numerator of $p(c|x) = \frac{p(c)p(x|c)}{\sum_{c'} p(c')p(x|c')}$

data sparsity: overfit when data too little

MLE: observations are R.V., parameters fixed

Bayesian: parameters as R.V., prior distr. $p(\theta)$, likelihood $p(\mathcal{D}|\theta)$

Update Posterior distr of θ: $p(\theta|\mathcal{D}) = \frac{p(\theta)p(\mathcal{D}|\theta)}{\int p(\theta')p(\mathcal{D}|\theta')\,\mathrm{d}\theta'}$

E.g. coin $L(\theta) = p(\mathcal{D}|\theta) = \theta^{N_H}(1-\theta)^{N_T}$, choose prior beta $p(\theta; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$ (ignore normalization)

$p(\theta|\mathcal{D}) \propto \left[\theta^{a-1}(1-\theta)^{b-1}\right]\left[\theta^{N_H}(1-\theta)^{N_T}\right]$

which is beata with $a' = (a+)N_H, b' = (b+)N_T$

Posterior: $\mathbb{E}[\theta|\mathcal{D}] = \frac{N_H+a}{N_H+H_T+a+b}$

Maximum A-Posteriori Estimation: find most likely para. setting under posterior; converts par esti. to maxi. prob.

$\hat\theta = \arg\max_\theta p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\theta) + \log p(\mathcal{D}|\theta) = \frac{N_H+a-1}{N_H+N_T+a+b-2}$

Gaussian Discriminate Analysis

Discriminative classifiers: Model $p(c|x)$ directly

Generative classifiers: Model $p(x|c)$ then Bayes $p(c|x)$; flexiable - easy to change classes; handle missing data naturally; more natural

Naive Bayes works for binary features, with Bernoulli distribution for each $p(x_j|c)$; for continuous features we use Gaussian distribution for each $p(x_j|c)$.

E.g. First multivariate Gaussians model $p(x|c=0), p(x|c=1)$ then $p(c=1|x) = \frac{p(x|c=1)p(c=1)}{p(x|c=1)p(c=1)+p(x|c=0)p(c=0)}$

Gaussian Discriminant Analysis generally assumes $p(x|c)$ is distributed as multivariate Gaussian $p(x|c=k) = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}}\exp\left[-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right]$, where $|\Sigma_k|$ denotes determinant. And each class k has associated mean vector μ_k (D par) and covariance matrix Σ_k ($\mathcal{O}(D^2)$ par, hard to estimate).

PDF of univariate Gaussian $\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, multivariate $\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}}\exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]$

$\mu = \mathbb{E}[x] = (\mu_1, \ldots, \mu_D)^T$, $\Sigma = \mathbb{E}[(x-\mu)(x-\mu)^T] = (\sigma_{ij})$

$$\ell(\mu,\Sigma) = \log\prod_{i=1}^{N}[\ldots] = \sum_{i=1}^{N} -\log(2\pi)^{\frac{d}{2}} - \log|\Sigma|^{\frac{1}{2}}$$

$$-\frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu)$$

Where first term is constant, μ only affects last term

Maximize the log-likelihood by setting der to 0:

$$\frac{\partial\ell}{\partial\mu_j} = -\frac{\partial}{\partial\mu_j}\left(\sum_{i=1}^{N}\frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu)\right)$$

$$\nabla_\mu\ell = -\sum_{i=1}^{N}\Sigma^{-1}(x^{(i)}-\mu) = 0$$

This gives $\hat\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$

Similarly, $0 = \nabla_\Sigma\ell \implies \hat\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)}-\hat\mu)(x^{(i)}-\hat\mu)^T = \frac{1}{N}(X - \mathbb{1}\hat\mu^T)(X - \mathbb{1}\hat\mu^T)^T$

Positive semidefinite if $v^TAv \geq 0$ for all v, definite if > 0.

Since $p(c|x) = \frac{p(c)p(x|c)}{\sum_{c'} p(c')p(x|c')}$, this gives $\log p(c_k|x) = \log p(x|c_k) + \log p(c_k) - \log p(x) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) + \log p(c_k) - \log p(x)$

Decision Boundary of GDA is a conic section since quadratic w.r.t. x.

GDA makes stronger modeling assumption - assumes class-conditional data is multivariate Gaussian (asymptotically efficient if true, otherwise bad prediction), LR beats GDA if data is not Gaussian

GDA can handle easily missing features

Similar with shared covariance

If x high dimensional, Σ_k too large, then assume features are independent (Σ is diagonal). Gaussian Naive Bayes assumes the likelihoods are Gaussian

---

Clustering: unsupervised, group data points (multimodal distribution) into clusters

K-means assumes K clusters, and each point is close to cluster center: randomly initialize cluster centers, then alter between assignment step (assign each data point to closest cluster) and refitting step (move cluster centers to mean of assigned points)

Find cluster center m_k and assignments $r^{(n)}$ (one-hot) to minimize

$$\min_{\{m_k\},\{r^{(n)}\}}\sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}\left|\left|m_k - x^{(n)}\right|\right|^2$$

Find exact is NP-hard; Assignment step = minimize w.r.t. $r^{(n)}$; Refitting step = minimize w.r.t. m_k

Assignment Step: Fix m_k, then $r_k^{(n)} = 1$ if $k = \arg\min_j\left|\left|x^{(n)} - m_j\right|\right|^2$, else 0

Refitting Step: Fix $r^{(n)}$, then $0 = \nabla_{m_\ell}(\Sigma\Sigma\ldots)$ gives $m_\ell = \frac{\sum_n r_\ell^{(n)}x^{(n)}}{\sum_n r_\ell^{(n)}}$

Called alternating minimization, or block coordinate descent

K-means give superpixels for images

May change ont-hot r to softmax: $r_k^{(n)} = \frac{\exp[-\beta\left|\left|m_k - x^{(n)}\right|\right|^2]}{\sum_{j=1}^{K}\exp[-\beta\left|\left|m_j - x^{(n)}\right|\right|^2]}$, m_k remains same

Bayes classifer: fit $p(t)$ and $p(x|t)$ using labelled data.

For Clustering: Sample t from $p(t)$, then sample x from $p(x|t)$

Since t never observed, called **latent variable** or **hidden variable**, denote z instead; x called observables

Then $p(x) = \sum_z p(x,z) = \sum_z p(x|z)p(z)$, called **latent variable model**

If $p(z)$ is categorial distribution (multinomial), this is a **mixture model**, different values of z correspond to different components

Gaussian Mixture Model GMM: If $p(x|z)$ is Gaussian, then $p(x) = \sum_{k=1}^{K}\pi_k\mathcal{N}(x|\mu_k,\Sigma_k)$ with $\sum_{k=1}^{K}\pi_k = 1, \pi_k \geq 0\forall k$.

GMMs are universal approximators of densities (same as MLPs)

No closed form solution of GMM Maximum log-likelihood, not identifiable

To maximize $\log p(X;\theta) = \sum_{i=1}^{N}\log\sum_{k=1}^{K}\pi_k\mathcal{N}(x^{(i)}|\mu_k,\Sigma_k)$

Expectation-Maximization algorithm:

Expectation step: Compute posterior probability over z given current modle - how much do we think each Gaussian generates each data point

Assign responsibility $r_k^{(i)}$ of component k for data point i using $r_k^{(i)} = P(z^{(i)} = k|x^{(i)};\theta)$

Maximization step: Assume data really was generated this way, change parameters of each Gaussian to maximize probability that it would generate the data responsible for

$$\pi_k = \frac{1}{N}\sum_{i=1}^{N} r_k^{(i)}, \mu_k = \frac{\sum_{i=1}^{N} r_k^{(i)}x^{(i)}}{\sum_{i=1}^{N} r_k^{(i)}}$$

$$\Sigma_k = \frac{1}{\sum_{i=1}^{N} r_k^{(i)}}\sum_{i=1}^{N} r_k^{(i)}(x^{(i)}-\mu_k)(x^{(i)}-\mu_k)^T$$

Principal Component Analysis PCA is a **linear** dimensionality reduction technique

Choose an orthonormal basis $\{u_i\}$ of the subspace S, then $\text{Proj}_S(x) = \sum_{i=1}^{k}(u_i^Tx)u_i = Uz$ where $z = U^Tx$

Let $\hat\mu$ be the origin, then $z = U^T(x-\hat\mu)$, $\tilde{x} = Uz + \hat\mu$ where $\tilde{x}$ called reconstruction of x and z called representation or code

capping to a space that is easier to visualize is called representation, learning called representation learning

$\hat\mu$ is the empirical mean of the data, U can be constructed by minimizing the reconstruction error or maximizing the variance of the reconstruction $\min_U \frac{1}{N}\sum_{i=1}^{N}\left|\left|x^{(i)} - \hat{x}^{(i)}\right|\right|^2$, $\max_U \frac{1}{N}\sum_{i=1}^{N}\left|\left|\hat{x}^{(i)} - \hat\mu\right|\right|^2$ (equiv.)

Since $U^TU = I$, the last is same as $||z||^2$

Choosing a subsapce to maximize the projected variance, or minimize the reconstruction error, is called **PCA**

Consider the empirical covariance matrix $\hat\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)}-\hat\mu)(x^{(i)}-\hat\mu)^T = Q\Lambda Q^T$ where Q is orthogonal and Λ is diagonal; then the optimal PCA subspace is spanned by the top (largest) K eigenvectors of $\hat\Sigma$

Eigenvectors are called **principal components**

For $K = 1$, u a unit vector, z scalar

$$\frac{1}{N}\sum_i\left|\left|\tilde{x}^{(i)} - \hat\mu\right|\right|^2 = \frac{1}{N}\sum_i[z^{(i)}]^2$$

$$= \frac{1}{N}\sum_i(u^T(x^{(i)}-\hat\mu))^2 = \frac{1}{N}\sum_i u^T(x^{(i)}-\hat\mu))(x^{(i)}$$

$$= u^T\hat\Sigma u = u^TQ\Lambda Q^Tu = \sum_{j=1}^{D}\lambda_j a_j^2, a = Q^Tu$$

Naive Bayes: Let $p(c=1) = \pi$ be a Bernuolli, then let θ_{ij} denote $p(x_i = 1|c = j)$ for $i \in [3], j \in \{0,1\}$

$p(c = 0|x_1 = 1) = \frac{p(c=0)}{p(x_1=1)}p(x_1 = 1|c = 0) = \frac{1-\pi}{p(x_1=1|c=1)p(c=1)+p(x_1=1|c=0)p(c=0)}\theta_{10}$

$p(x_1 = 1|c = 1) = \frac{p(x_1=1,c=1)}{p(c=1)}$

For $p(c=1) = \pi$ Bernuolli, $p(c) = \pi^c(1-\pi)^{1-c}$

If for MLP, a is sent to z_1, z_2, then $\frac{\partial\mathcal{L}}{\partial a} = \frac{\partial\mathcal{L}}{\partial z_1}\frac{\partial z_1}{\partial a} + \frac{\partial\mathcal{L}}{\partial z_2}\frac{\partial z_2}{\partial a}$

For linear separability, consider the intersection to show false