What is Machine Learning?

Program an algorithm to automatically learn from data or experience

main focus → **Supervised Learning** labeled examples of correct behaviour

Unsupervised Learning unlabeled examples, instead we look for "interesting" patterns in the data

Reinforcement Learning learning system (agent) interacts w/ the world/environment to maximize scalar reward signal

Derive algorithm, translate math into code

Vectorize computations (express in matrix/vector) to exploit hardware efficiency. — easily do linear algebra

In supervised learning, we are given a **training set** consisting of **inputs** & corresponding **labels**

Input commonly represented as **input vector** $\in \mathbb{R}^d$

Representation mapping to another space that's easy to manipulate

Mathematically, training set consists of collection of pairs of input vector $x \in \mathbb{R}^d$ & corresponding **label/target** t

Regression $t \in \mathbb{R}$ (continuous), predict a scalar valued target

Classification $t \in$ discrete set $\{1, \dots, C\}$

t can also be a highly structured object → image

Denote the training set by $\{(x^{(1)}, t^{(1)}), \dots, (x^{(N)}, t^{(N)})\}$

Hyperparameter external configuration variables from the model whose value cannot be estimated from data, can be tuned using validation set

Overfit model is sensitive to random behaviours in traing data

Underfit model fails to capture important regularities

K-Nearest Neighbours

Classify a new input x based on its k-nearest neighbours in the training set, usually classification

Can visualize w/ Voronoi diagram

Decision Boundary the boundary between regions of input space associated w/ different categories

However, it can be sensitive to noise or mis-labeled data, so instead of just finding 1 NN, we use kN

Find k examples $\{x^{(i)}, t^{(i)}\}$ closest to test instance x, classification by the majority

$$\underset{\text{prediction for } x}{y^*} = \max_{t^{(z)} \in \text{class labels}} \sum_{i=1}^{k} \mathbb{I}(t^{(z)} = t^{(i)})$$

Hyperparameter k controls # of neighbours used. optimal choice of k depends on # of data points n — general rule of thumb: $k < \sqrt{n}$

Small good at capturing fine-grained patterns, may overfit

Large makes stable predictions by averaging over many examples, may underfit

The Curse of Dimensionality

As the dimensionality increases, the number of data points required for good performance increases exponentially (Hughes Phenomenon)

In high dimensions, "most" points are the same distance — # of irrelevant dimensions that NN will compute on increases

Some datasets have low **intrinsic dimension** lie on a low-dimension manifold

Normalization

Can be sensitive to range/units of features

$$\bar{x}_j = \frac{x_j - M_j}{\sigma_j}$$ (238 hypothesis test stuff)

Simple fix: **normalize** each dimension to be zero mean & unit variance

Computation Cost

Computations at **test time** $O(ND)$ to calculate distances w/ D dimensions, N data points, sort distances takes $O(N \log N)$

DECISION TREES

Make predictions by splitting features according to a tree structure

Internal nodes test a feature

Branching is determined by feature value

Leaf nodes are outputs/predictions

Regression continuous output

leaf value $y^{(m)}$ typically set to mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Classification discrete output

leaf value $y^{(m)}$ typically set to most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Decision Boundary axis-aligned planes

Resort to a greedy heuristic

Start w/ whole training set & empty decision tree

How do we quantify loss?

Pick feature & candidate split that would most reduce a loss, split on that feature, recurse w/ subpartitions

QUANTIFYING UNCERTAINTY

Entropy of a discrete random variable is a # that quantifies the uncertainty inherent in its possible outcomes

Expected information content of a random draw from a probability distribution

High entropy variable has uniform-like outcome distribution, values less predictable

Low entropy distribution more concentrated in a few areas, more predictable values

$$\log_2 p(y|x) = \log_2 \frac{p(x,y)}{p(x)}$$
$$= \log_2 p(x,y) - \log_2 p(x)$$

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

continuous: use integral over Y

Joint Entropy

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

Conditional Entropy

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

Properties

- $H \geq 0$
- Chain rule: $H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
- If X & Y independent, then uncertainty unaffected: $H(Y|X) = H(Y)$
- Knowing Y makes knowledge of Y certain: $H(Y|Y) = 0$
- Knowing X, we can only decrease our uncertainty of Y: $H(Y|X) \leq H(Y)$

Information Gain $IG(Y|X) = H(Y) - H(Y|X)$ measures reduction in entropy after splitting a dataset, measures informativeness of a split

If X completely uninformative about Y: $IG(Y|X) = 0$ completely informative: $IG(Y|X) = H(Y)$

Learn decision tree by greedy, recursive approach, building tree node by node. Pick features that maximize information gain.

What makes a good tree? We desire small trees w/ informative nodes near the root.

Not too small: need to handle important but possibly subtle distinctions in data

Not too big: computational efficiency, careful not to overfit, human intepretability

BIAS-VARIANCE DECOMPOSITION

Consider squared error loss, $L(y,t) = \frac{1}{2}(y-t)^2$. Suppose we knew $p(t|x)$, then $y_* = \mathbb{E}[t|x]$ is the best possible prediction

$$\mathbb{E}[(y-t)^2|x] = \mathbb{E}[y^2 - 2yt + t^2|x]$$
$$= y^2 - 2y\mathbb{E}[t|x] + \mathbb{E}[t^2|x] \qquad \text{Var}[t|x] = \mathbb{E}[t^2|x] - \mathbb{E}[t|x]^2$$
$$= y^2 - 2yy_* + y_*^2 + \text{Var}[t|x] = (y-y_*)^2 + \text{Var}[t|x]$$

Bayes error (noise) inherent unpredictability of target t, best possible algorithm

Treating y as a random variable...

$$\mathbb{E}[(y-t)^2|x] = \mathbb{E}[(y-y_*)^2|x] + \text{Var}(t) = y_*^2 - 2y_*\mathbb{E}[y] + \mathbb{E}[y^2] + \text{Var}(t)$$
$$= \underbrace{(y_* - \mathbb{E}[y])^2}_{\text{Bias}} + \underbrace{\text{Var}(y)}_{\text{variance}} + \underbrace{\text{Var}(t)}_{\text{Bayes error}}$$

Bias the inability for a ML method to capture the true relationship, underfitting

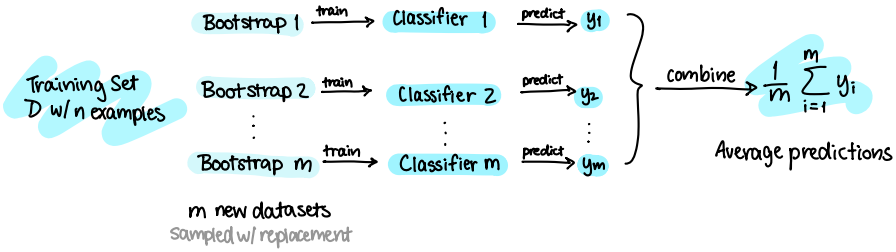
Variance the difference in fits between data sets, if it fits one data set well but not another ⇒ high variance, overfitting

BAGGING

short for bootstrap aggregating

Process by which multiple models of same learning algorithm are trained w/ bootstrapped samples of original dataset

Reduces overfitting by averaging predictions



m new datasets
sampled w/ replacement

Average predictions & apply a threshold

↳ $y_{bagged} = \mathbb{I}\left(\frac{1}{m}\sum_{i=1}^{m} y_i > 0.5\right)$

A bagged classifier can be stronger than average model (↳ "Ask the Audience" vs. "Phone a Friend")

m datasets are NOT independent ⇒ don't get $\frac{1}{m}$ variance reduction ⟶ solution: add more randomness in Random Forests

Weighting members equally might not be the best, weighted ensembling often leads to better results if members are very different

LINEAR MODELS

LINEAR REGRESSION

Linear function of features $x = (x_1, \dots, x_D) \in \mathbb{R}^D$ to make prediction $y \in \mathbb{R}$ of target $t \in \mathbb{R}$

$$y = f(x) = \sum_j w_j x_j + b = w^T x + b$$

↑ b often included in w by padding x w/ 1 (STA302 MLR)

Parameters weights w, bias/intercept b

Want prediction to be close to target, $y \approx t$

Loss function $\mathcal{L}(y,t)$ defines how badly algorithm's prediction fits target t for some sample x.

Often use squared error loss for regression $\mathcal{L}(y,t) = \frac{1}{2}(y-t)^2$

↑ residual/error, want to minimize

Cost function loss function averaged over all training examples (aka. empirical / average loss)

$$J(w,b) = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N}(w^T x^{(i)} + b - t^{(i)})^2$$

We can vectorize algorithms using vectors & matrices to abuse GPU capabilities instead of for loop, `y = np.dot(w, x) + b`

Learn by minimizing cost function J(w) using direct solution, gradient descent, stochastic gradient descent, batch gradient descent

↳ $w^* = (X^T X)^{-1} X^T t$ for squared loss error, check STA302

GRADIENT DESCENT (Batch)

Iterative algorithm, initialize weights and update repeatedly in direction of steepest descent.

$$w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}$$

$0 < \alpha \leq 1$ is learning rate hyperparameter, how fast we learn

minimizing total loss vs. average loss requires smaller learning rate

For linear regression $w \leftarrow w - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})x^{(i)}$

$\frac{\partial J}{\partial w_j} > 0 \Rightarrow$ decreasing J means decreasing w_j

$\frac{\partial J}{\partial w_j} < 0 \Rightarrow$ increasing w_j

L^2 regularized: $J + \lambda R$

$w \leftarrow w - \alpha\frac{\partial}{\partial w}(J + \lambda R)$

$= (1 - \alpha\lambda)w - \alpha\frac{\partial J}{\partial w}$

STOCHASTIC GRADIENT DESCENT

Update the parameters based on gradient for one training example instead of taking the average over all training data

↳ Choose example i uniformly at random, perform update $\theta \leftarrow \theta - \alpha\frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$

Cost of each update is independent of N, approximates batch GD, fast progress before seeing all data.

High variance in estimate, no exploitation of vectorization

MINI-BATCH GRADIENT DESCENT

Set learning rate high to get close to optimum
Gradually decrease to get stable estimates

Mini-batch a randomly chosen medium-sized subset of training examples M

In theory, examples sampled independently & uniformly w/ replacement.

In practice, permute training set, go through sequentially. Each pass over data is an epoch

Split the dataset into mini-batches, compute gradients for each batch, use mean gradient to update parameters

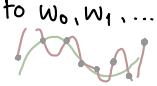
As $|M|$ increases: more compute, more accurate estimates, can exploit vectorization

Controlling Model Complexity

Feature Mapping map input features to another space $\psi(x): \mathbb{R}^D \to \mathbb{R}^d$, treat as input to linear regression

↳ Polynomial feature mapping $\psi(x) = [1, x, x^2, \dots, x^M]^T$

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^{M} w_i x^i = \psi(x)^T w$$ linear to $w_0, w_1, \dots, w_M$



As M increases, magnitude of coefficients tend to increase, can lead to over-fitting

Regularization quantify how much we prefer one hypothesis vs. another w/ a regularizer (function)

↳ Encourage weights to be small w/ L^2 penalty as regularizer

$$R(w) = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}\sum_j w_j^2 \quad \Longrightarrow \quad J_{reg}(w) = J(w) + \lambda R(w) = J(w) + \frac{\lambda}{2}\sum_j w_j^2$$

If model fits training data poorly, J is large. If weights are large in magnitude, R is large.

↳ L^2 Regularized Least Square Regression (Ridge Regression)

$$w_\lambda^{Ridge} = \operatorname*{argmin}_w J_{reg}(w) = \operatorname*{argmin}_w \frac{1}{2N}\|Xw - t\|_2^2 + \frac{\lambda}{2}\|w\|_2^2 = (X^TX - \lambda N I)^{-1}X^Tt$$

Convexity

Critical points found by gradient descent may only be local min, not global.

If a function is convex, every critcal point is a global optimum.

A set S is convex if any line segment connecting two points in S also lies entirely in S

$$\forall x_1, x_2 \in S, \quad \lambda x + (1-\lambda)x_2 \in S \text{ for } 0 \le \lambda \le 1$$

Weighted averages / convex combinations of points in S lie in S.

$$\forall x_1, \dots, x_N \in S, \quad \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_N x_N \in S \text{ for some } \lambda_i > 0, \lambda_1 + \cdots + \lambda_N = 1$$

A function f is convex if the line segment between any two points on f lies above f's graph

$$\forall x_0, x_1 \in \text{domain of } f, \quad f((1-\lambda)x_0 + \lambda x_1) \le (1-\lambda)f(x_0) + \lambda f(x_1)$$

For linear models, $z = w^Tx + b$ is a linear function of w & b.

If the loss function is a convex function of z, then it is also a convex function of w & b.

Binary Linear Classification

$$z = w^Tx + b \qquad y = \begin{cases} 1 & \text{if } z \ge r \\ 0 & \text{if } z < r \end{cases}$$

classification predict a discrete-valued target given input $x \in \mathbb{R}^D$

binary predict a binary target $t \in \{0 = \text{negative}, 1 = \text{positive}\}$

linear prediction y is a linear function of x, followed by threshold r

↳ NOT

x_0	x_1	t
1	0	1
1	1	0

$z = w_0x_0 + w_1x_1$

$$y = \begin{cases} 1 & \text{if } z \ge 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Substitute values in

$1 \cdot w_0 + 0 \cdot w_1 \ge 0$

$1 \cdot w_0 + 1 \cdot w_1 < 0$

→ Simplify

$w_0 \ge 0$

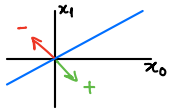
$w_1 < -w_0$

→ test values

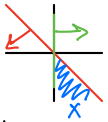
$w_0 = 1$

$w_1 = -2$

In data space, points are training examples, line is a set of weights.



In weight space, points are weights



$w_0 \ge 0$

$w_0 + w_1 < 0$

A half-space represents a training example constraining the weights

Feasible region satisfies all constraints. The problem is feasible if feasible region is nonempty

Data is linearly separable if a linear decision rule can perfectly separate the training examples

Can learn weights using linear programming, preception algorithm.

LOGISTIC REGRESSION

If data isn't linearly separable: define a loss function, find weights that minimize the average loss over training examples

0-1 Loss

$$\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & \text{if } y=t \\ 1 & \text{if } y \neq t \end{cases} = \mathbb{I}[y \neq t]$$

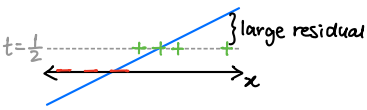
The cost J is the misclassification rate $J = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]$

Problems: ∇ is 0 almost everywhere except at $z=0$ where is undefined

SQUARED Loss

$\mathcal{L}_{SE}(z,t) = \frac{1}{2}(z-t)^2$, treat binary targets as continuous values.

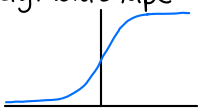
Making a correct prediction w/ high confidence inccurs a large loss



LOGISTIC ACTIVATION FUNCTION

Squash predictions y to $[0,1]$ using logistic function, sigmoid shape

activation function $\sigma(z) = \frac{1}{1+e^{-z}}$



$y = \sigma(z)$

$\mathcal{L}_{SE}(y,t) = \frac{1}{2}(y-t)^2$

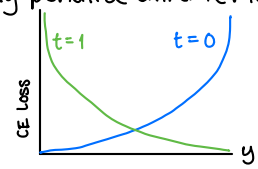
An extreme misclassification case appears optimal

CROSS-ENTROPY Loss

Interpret $y \in [0,1]$ as estimated probability that $t=1$, heavily penalize extreme misclassification cases

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

$$= -t\log y - (1-t)\log(1-y)$$



We combine logistic activation function & cross-entropy into a logistic-cross-entropy function for numerical stability

$$\mathcal{L}_{LCE}(z,t) = \mathcal{L}_{CE}(\sigma(z),t) = t\log(1+e^{-z}) + (1-t)\log(1+e^{-z})$$

Use gradient descent to minimize cost J since logistic loss is convex in w.

$$\mathcal{L}_{CE}(y,t) = -t\log(y) - (1-t)\log(1-y), \quad y = \frac{1}{1+e^{-z}}, \quad z = w^T x$$

$$\frac{\partial \mathcal{L}_{LCE}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y}\cdot\frac{\partial y}{\partial z}\cdot\frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1-t}{1-y}\right)\cdot y(1-y)\cdot x_j = (y-t)x_j$$

$$w_j \leftarrow w_j - \alpha\frac{\partial J}{\partial w_j} = w_j - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})x_j^{(i)}$$

oh wow this is just linear regression gradient descent
who would have thought

GRADIENT CHECKING

Algorithms may appear to work even if math is wrong, but obviously work better @ math is correct.

Perform gradient checking using finite differences

one-sided vs two-sided

$$\frac{\partial}{\partial x_i} f(x_1,\ldots,x_N) = \lim_{h\to 0}\frac{f(x_1,\ldots,x_i+h,\ldots,x_N) - f(x_1,\ldots,x_i,\ldots,x_N)}{2h}$$

$f(x_1,\ldots,x_i+h,\ldots,x_N)$

Check w/ small value of h (10^{-10}), compute relative error $\frac{|a-b|}{|a|+|b|}$

a = finite diff

b = implementation derivative

↑ should be really small (10^{-6}?)

HYPOTHESIS SPACE & INDUCTIVE BIAS

Hypothesis a function $f: \mathcal{X} \to \mathcal{T}$ from input to target space. A ML algorithm aims to find a good hypothesis $f \in \mathcal{H}$.

The hypothesis space $\mathcal{H}$ is a set of hypotheses (Linear regression $\mathcal{H}$ is set of linear functions)

Inductive bias an algorithm's members of $\mathcal{H}$ & its preference for some hypotheses over others.

No Free Lunch Theorem if datasets/problems were not naturally biased, no ML algorithm would be better than the other

PARAMETRIC VS. NON-PARAMETRIC ALGORITHMS

Parametric hypothesis space $\mathcal{H}$ defined using finite set of parameters

↳ linear & logistic regression, neural networks, k-means, Gaussian mixture, decision trees

Non-Parametric hypothesis space $\mathcal{H}$ defined in terms of data, no parameter defined in terms of data

↳ kNN, Guassian processes, kernel density estimation

Multi-class Classification

the k^{th} entry is 1

Targets form a discrete set $\{1, \dots, K\}$ OR represent targets as one-hot vectors / one-of-K encoding $t = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K$

Inputs represented linearly

W : $K \times D$ matrix
x : $D \times 1$ vector
b : $K \times 1$ vector
z : $K \times 1$ vector

Vectorized: $z = Wx + b$, $z = Wx$ w/ dummy $x_0 = 1$

Non-vectorized: $z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k$ for $k = 1, 2, \dots, K$

Interpret z_k as how much the model prefers the k-th prediction

$$y_i = \begin{cases} 1, & \text{if } i = \operatorname{argmax}_k z_k \\ 0, & \text{otherwise} \end{cases}$$

↳ $z = (-3, -2, 5, 12)$
↳ $y = (0, 0, 0, 1)$

Softmax Regression

Softens the predictions for optimization using softmax function, natural activation function

$$y_k = \text{softmax}(z_1, \dots, z_K) = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

Inputs z_k are logits, interpret outputs as probabilities

If z_k is much larger than others, $\text{softmax}(z)_k \approx 1$, behaves like argmax.

Use cross-entropy as loss function, combine w/ softmax $\Rightarrow$ softmax-cross-entropy function (log applied element-wise)

Tracking Model Performance

Track progress during learning by plotting training curves.

For binary classification, we can track accuracy

$$\text{Acc} = \frac{TP + TN}{P + N}$$

Prediction \ True class	0	1
0	TP	FN
1	FP	TN

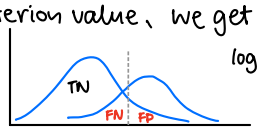
FN false negative, type II error

FP false positive, type I error

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$ true positive rate

$$\text{Specificity} = \frac{TN}{TN + FP}$$ true negative rate

As we increase criterion value, we get a tradeoff

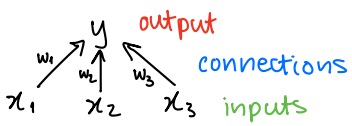


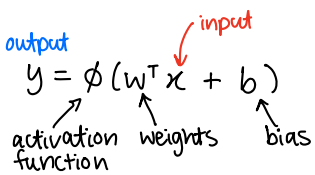
logistic, $p(y = 1 \mid x) \geq r$

Neural Networks

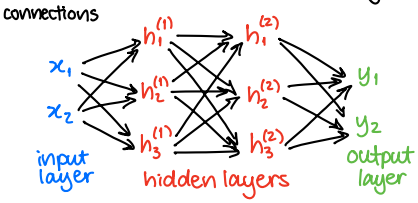
In the brain, neurons receive input signals & accumulate voltage. They fire after reaching some threshold

For neural nets, each "neuron" model is called a unit





Feed-forward Neural Network a directed acyclic graph, units are grouped into layers



A multi-layer network consists of fully connected layers, where all input units are connected to all output units

Each hidden layer i connects N_{i-1} input units to N_i output units, weight matrix is $N_i \times N_{i-1}$

Outputs are a function of input units $y = f(x) = \phi(Wx + b)$, ϕ applied component wise

↳ Identity, ReLU, soft ReLU, threshold, logistic, tanh

Each layer computes a function → $h^{(i)} = f^{(i)}(h^{(i-1)}) = \phi(w^{(i)} h^{(i-1)} + b^{(i)})$, overall computes $y = f^{(L)} \circ \cdots \circ f^{(1)}(x)$

↳ regression $y = f^{(L)}(h^{(L-1)}) = (w^{(L)})^T h^{(L-1)} + b^{(L)}$

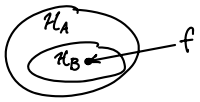
binary $y = f^{(L)}(h^{(L-1)}) = \sigma((w^{(L)})^T h^{(L-1)} + b^{(L)})$

Each hidden unit in first layer acts as a feature detector

EXPRESSIVITY

Consider two models A & B w/ hypothesis spaces $\mathcal{H}_A$, $\mathcal{H}_B$. If $\mathcal{H}_B \subseteq \mathcal{H}_A$, then A is more expressive than B.

A can represent any function f in $\mathcal{H}_B$



PROBABILISTIC MODELS

LOG LIKELIHOOD

Coin flip outcome x can be a Bernoulli RV for a currently unknown parameter $\theta \in [0,1]$

Assume outcomes $\{x_1, \dots, x_N\}$, $x_i = 1$ means heads, are independently & identically distributed

→ Joint probability of outcome is

$$p(x_1, \dots, x_N) = \prod_{i=1}^{N} \theta^{x_i} (1-\theta)^{1-x_i}$$

The likelihood function of probability observing the data as function of parameter θ is

$$L(\theta) = \prod_{i=1}^{N} \theta^{x_i} (1-\theta)^{1-x_i} \xrightarrow{\log} \ell(\theta) = \sum_{i=1}^{N} x_i \log\theta + (1-x_i)\log(1-\theta)$$

Maximum likelihood criterion says we should pick parameters that maximize likelihood ℓ

$$\hat{\theta} = \operatorname{argmax}_{\theta \in [0,1]} \ell(\theta)$$

Find optimal solution by setting derivatives to 0.

$$\frac{d\ell}{d\theta} = \frac{d}{d\theta}\left(\sum_{i=1}^{N} x_i \log\theta + (1-x_i)\log(1-\theta)\right) = \frac{\sum x_i}{\theta} - \frac{N - \sum x_i}{1-\theta} = \frac{N_H}{\theta} - \frac{N_T}{1-\theta} = 0$$

$$\hat{\theta}_{ML} = \frac{N_H}{N_H + N_T}$$

DISCRIMINATIVE & GENERATIVE CLASSIFIERS

How do we use tools from probability to model questions in learning?

Discriminative classifiers try to learn mappings directly from space of inputs $\mathcal{X}$ to class labels $\{0, 1, \dots, K\}$

features x ⟶ class probability $p(y|x)$

Model $p(t|x)$ directly (logistic regression models)

Learn mappings from inputs to classes (regression, decision trees, etc.)

How do I separate the classes?

Generative classifiers try to build model of "what data for a class looks like"

class $p(y)$ ⟶ probability of class given label $p(x|y)$

Model $p(x|t)$, distribution of individual classes. What does each class look like?

Apply Bayes Rule to derive $p(t|x)$

NAÏVE BAYES MODELS

Given features $x = [x_1, x_2, \dots, x_D]^T$, want to compute class properties using Bayes Rule

$$\underbrace{p(c|x)}_{p(\text{class given feature})} = \frac{\overbrace{p(x|c)}^{p(\text{feature given class})}\ p(x)}{p(c)} \longrightarrow \text{posterior for class} = \frac{p(\text{feature given class}) \times \text{prior for class}}{p(\text{feature})}$$

Naïve assumption features x_i are conditionally independent given class c $\quad p(x, y|c) = p(x|c)\,p(y|c)$

$$p(c, x_1, \dots, x_D) = p(c)\,p(x_1|c) \cdots p(x_D|c)$$

↳ Classifying spam $p(c=1) = \pi$ (spam), $p(x_j = 1|c) = \theta_j$ (prob of word appearing in spam)

$$\ell(\theta) = \sum_{i=1}^{N} \log p(c^{(i)}, x^{(i)}) = \sum_{i=1}^{N} \log\{p(x^{(i)}|c^{(i)})\,p(c^{(i)})\} = \cdots = \underbrace{\sum_{i=1}^{N} \log p(c^{(i)})}_{\text{log-likelihood of labels}} + \sum_{j=1}^{D} \underbrace{\sum_{i=1}^{N} \log p(x_j^{(i)}|c^{(i)})}_{\text{log-likelihood for feature } x_j}$$

Learn prior (probability distribution) by maximizing $\sum \log p(c^{(i)})$ $\quad \pi = p(c^{(i)}=1)$ $\quad \theta?$

↳ Pr of i^{th} email: $p(c^{(i)}) = \pi^{c^{(i)}}(1-\pi)^{1-c^{(i)}}$

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

$$\frac{c^{(i)}}{\pi} - \frac{N-c^{(i)}}{1-\pi} = 0$$

$$c^{(i)} - \pi c^{(i)} - \pi N + \pi c^{(i)} = 0$$

Max likelihood estimator of prior π: $\hat{\pi} = \frac{\sum \mathbb{1}(c^{(i)}=1)}{N} = \frac{\#\text{ spam in dataset}}{\text{total }\#\text{ samples}}$

Learn posterior by maximizing $\sum \log p(x_j^{(i)} \mid c^{(i)})$ $\quad \theta_{jc} = p(x_j^{(i)}=1 \mid c)$

↳ Pr of i^{th} email: $p(x_j^{(i)} \mid c^{(i)}) = \theta_{jc}^{x_j^{(i)}}(1-\theta_{jc})^{1-x_j^{(i)}}$

$$\sum_{i=1}^{N} \log p(x_j^{(i)} \mid c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \{x_j^{(i)} \log \theta_{j1} + (1-x_j^{(i)}) \log(1-\theta_{j1})\} \quad \text{spam}$$

$$+ \sum_{i=1}^{N} (1-c^{(i)}) \{x_j^{(i)} \log \theta_{j0} + (1-x_j^{(i)}) \log(1-\theta_{j0})\} \quad \text{not spam}$$

$1-0$ since $c^{(i)}=0$ not spam

Max likelihood estimate of θ_{jc}: $\hat{\theta}_{jc} = \frac{\sum \mathbb{1}(x_j^{(i)}=1 \wedge c^{(i)}=c)}{\sum \mathbb{1}(c^{(i)}=c)} = \frac{\#\text{ word } j \text{ appears in class } c}{\#\text{ class } c \text{ in dataset}}$

Predict class by performing inference on model. Apply Bayes' Rule

$$p(c \mid x) = \frac{p(c)\,p(x \mid c)}{\sum_{c'} p(c')\,p(x \mid c')} = \frac{p(c) \prod_{j=1}^{D} p(x_j \mid c)}{\sum_{c'} p(c') \prod_{j=1}^{D} p(x_j \mid c)}$$

For input x, predict c w/ largest $p(c)\prod_{j=1}^{D} p(x_j \mid c)$ most likely class

$$p(c \mid x) \propto p(c) \prod_{j=1}^{D} p(x_j \mid c)$$

Short training time b/c just one pass through data. Also cheap test time b/c of model structure

Analysis easily extends to other probability distributions.

Less accurate in practice compared to discriminative models b/c of naïve independence assumption.

BAYES PARAMETER ESTIMATION

Maximum likelihood can overfit if data is too sparse, we have sparsity problem.

Need to specify prior distribution $p(\theta)$ encodes beliefs about parameters before observing data

likelihood $p(D \mid \theta)$ encodes likelihood of observing data given parameters

↻ we update beliefs based on observations, we compute posterior distribution using Bayes' Rule

$$p(D \mid \theta) = \frac{p(\theta)\,p(D \mid \theta)}{\int p(\theta')\,p(D \mid \theta')\,d\theta'}$$

↳ Coinflip: $L(\theta) = p(D \mid \theta) = \theta^{N_H}(1-\theta)^{N_T}$.

We can use beta distribution for prior $p(\theta; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1}(1-\theta)^{b-1}$, $\mathbb{E}[\theta] = \frac{a}{a+b}$

As $a, b \uparrow$, distribution becomes more peaked. $a=b=1$ uniform

Posterior distribution $p(\theta \mid D) \propto p(\theta)\,p(D \mid \theta) \propto \theta^{a-1+N_H}(1-\theta)^{b-1+N_T}$, $\mathbb{E}[\theta \mid D] = \frac{N_H + a}{N_H + N_T + a + b}$

↻ you have enough observations, the data overwhelms the prior

Maximum A-Posteriori (MAP) Estimation finds most likely parameters under posterior (mode)

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta \mid D) = \arg\max_{\theta} p(\theta)\,p(D \mid \theta) = \arg\max_{\theta} \log p(\theta) + \log p(D \mid \theta)$$

Maximize by finding critical point, $\hat{\theta}_{MAP} = \frac{N_H + a - 1}{N_H + N_T + a + b - 2}$

$\hat{\theta}_{MAP}$ assigns non-zero probabilities as long as $a, b > 1$.

LINEAR ALGEBRA REVIEW

Let B be a square matrix. Vector v is an eigenvector of B if $Bv = \lambda v$ for some eigenvalue $\lambda \in \mathbb{R}$

A $D \times D$ matrix has at most D distinct eigenvalues.

A symmetric matrix A can be factorized w/ Spectral Decomposition $A = Q \Lambda Q^T$

Q is an orthogonal matrix where columns q_i are eigenvectors

Λ is a diagonal matrix where entries λ_i are corresponding eigenvalues $Aq_i = \lambda_i q_i$

Since A has full set of orthonormal eigenvectors $\{q_i\}$, can use as orthonormal basis for $\mathbb{R}^D$.

In the other basis: $x = \tilde{x}_1 q_1 + \cdots + \tilde{x}_D q_D$, $\tilde{x} = Q^T x$, $x = Q\tilde{x}$

A acts as re-scaling. $Ax = \tilde{x}_1 A q_1 + \cdots + \tilde{x}_D A q_D = \lambda_1 \tilde{x}_1 q_1 + \cdots + \lambda_D \tilde{x}_D q_D$

Symmetric matrices represent quadratic forms, $f(v) = v^T A v$

$\forall v \neq 0$, $v^T A v > 0$, A is positive definite $v^T A v < 0$, A is negative definite

≥ 0, A is positive semi-definite ≤ 0, A is negative semi-definite

Claim: A is PSD $\iff$ all of its eigenvalues are non-negative

A positive definite $\implies$ contours of quadratic form are elliptical

A positive definite + diagonal $\implies$ ellipses are axis-aligned

For a PD $A = Q \Lambda Q^T$, contours are elliptical + aligned w/ eigenvectors

↳ $f(v) = v^T Q \Lambda Q^T v = \tilde{v}^T \Lambda \tilde{v} = \sum \lambda_i \tilde{v}_i^2$

By spectral decomp, $A^k = Q \Lambda^k Q^T$. If A invertible: $A^{-1} = Q \Lambda^{-1} Q^T$. If PSD, square root $A^{1/2} = Q \Lambda^{1/2} Q^T$

Determinant of a symmetric matrix is the product of its eigenvalues. $|A| = |\Lambda| = \prod_i \lambda_i$

PSD matrix A, $|A| \geq 0$

MULTIVARIATE GAUSSIAN DISTRIBUTION

Mean $\mu = \mathbb{E}[x] = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_D \end{pmatrix}$ Covariance $\Sigma = \text{Cov}(x) = \mathbb{E}[(x-\mu)(x-\mu)^T] = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_D^2 \end{pmatrix}$

(μ, Σ) uniquely define a multivariate Gaussian (Normal) distribution. $N(x; \mu, \Sigma)$, $N(\mu, \Sigma)$

$$\text{PDF} \quad N(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)^T\right)$$

All univariate Gaussian distributions are shaped like standard normal distribution

To shift multivariate Gaussian distribution, start w/ standard $x \sim N(0, 1)$. Apply linear map $\hat{x} = Sx + b$.

$$\mathbb{E}[\hat{x}] = \mathbb{E}[Sx + b] = b, \quad \text{Cov}(\hat{x}) = \text{Cov}(Sx + b) = SS^T$$

To obtain $N(\mu, \Sigma)$, start w/ $N(0, 1)$ again.

Shift by μ, scale by matrix $\Sigma^{1/2} = Q \Lambda^{1/2} Q$

MAXIMUM LIKELIHOOD

$$l(\mu, \Sigma) = \sum_{i=1}^{N} -\log(2\pi)^{d/2} - \log|\Sigma|^{1/2} - \frac{1}{2}(x^{(i)} - \mu)^T \Sigma^{-1} (x^{(i)} - \mu)$$

Best estimate for μ, $\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$, is sample mean of observed values, empirical mean

Best estimate for Σ, $\hat{\Sigma} = \frac{1}{N}(X - 1\mu^T)^T (X - 1\mu^T)^T$, is the empirical covariance

MORE LINEAR REGRESSION

Given a traing set of inputs & targets $\{(x^{(i)}, t^{(i)})\}_{i=1}^N$, model $y = w^T x$

Squared error loss $\mathcal{L}(y,t) = \frac{1}{2}(t-y)^2$, L_2 reg. $R(w) = \frac{\lambda}{2}\|w\|^2$

closed-form solution: $w = (X^TX + \lambda I)^{-1}X^Tt$, $\quad w \leftarrow (1-\alpha\lambda)w - \alpha X^T(y-t)$

We can give linear regression a probabilistic interpretation.

Assume Gaussian noise model $t \mid x \sim N(w^Tx, \sigma^2)$

Linear regression is just maximum likelihood under this model

$$\frac{1}{N}\sum_{i=1}^N \log p(t^{(i)} \mid x^{(i)}; w, b) = \frac{1}{N}\sum_{i=1}^N \log N(t^{(i)}; w^Tx, \sigma^2)$$

$$= \text{const.} - \frac{1}{2N\sigma^2}\sum_{i=1}^N (t^{(i)} - w^Tx)^2$$

Minimizing MSE yields the MLE for parameters

We can also view L_2 regularizer as MAP inference w/ Gaussian prior.

Assume Gaussian prior, $w \sim N(m, S)$

$$\log p(w) = \log N(w; m, S) = \cdots = \frac{1}{2}(w-m)^T S^{-1}(w-m) + \text{const}$$

Commonly, $m = 0$, $S = \eta I$, so

$$\log p(w) = -\frac{1}{2\eta}\|w\|^2 + \text{const} \quad \text{is } L_2 \text{ regularization.}$$

GAUSSIAN DISCRIMINANT ANALYSIS

Assumes that $p(x \mid t)$ is distributed according to multivariate Gaussian distribution.

$$p(x \mid t = k) = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right] \quad |\Sigma_k| = \text{determinant}$$

Each class k has mean vector μ_k, covariance matrix Σ_k

Learn parameters for each class using likelihood (assume binary classification for now)

↳ $p(t \mid \phi) = \phi^t(1-\phi)^{(1-t)}$

Compute ML estimates ($r_k^{(i)} = \mathbb{I}[t^{(i)} = k]$)

$$\phi = \frac{1}{N}\sum_{i=1}^N r_1^{(i)} \qquad \mu_k = \frac{\sum_{i=1}^N r_k^{(i)} \cdot x^{(i)}}{\sum_{i=1}^N r_k^{(i)}} \qquad \Sigma_k = \frac{1}{\sum_{i=1}^N r_k^{(i)}}\sum_{i=1}^N r_k^{(i)}(x^{(i)}-\mu_k)(x^{(i)}-\mu_k)^T$$

DECISION BOUNDARY

$$\log p(t_k \mid x) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) + \log p(t_k) - \log p(x)$$

Plug in Gaussian $p(t \mid x)$ into Bayes' rule, eventually somehow get decision boundary

$$(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) = (x-\mu_\ell)^T\Sigma_\ell^{-1}(x-\mu_\ell) + \text{const}$$

Quadratic function in x, so decision boundary is conic section

If all classes share same covariance Σ, decision boundary is linear.

For $p(t=1 \mid x)$ under GDA, assuming $\Sigma_0 = \Sigma_1 = \Sigma$, then oh wow, it's logistic regression

GDA makes stronger modeling assumption that class-conditional data is multivar Gaussian

Non-Gaussian stuff, LR beats GDA, but GDA can handle missing features

GAUSSIAN NAIVE BAYES

If x is high dimensional, $\Sigma_k \in \mathcal{O}(D^2 K)$ parameters.

Gaussian Naive Bayes classifier assumes Gaussian likelihood, model same as GDA w/ diag. Σ.

$$\mu_{jk} = \frac{\sum_{i=1}^N r_k^{(i)} x_j^{(i)}}{\sum_{i=1}^N r_k^{(i)}} \qquad \sigma_{jk}^2 = \frac{\sum_{i=1}^N r_k^{(i)} (x_j^{(i)} - \mu_{jk})^2}{\sum_{i=1}^N r_k^{(i)}} \qquad r_k^{(i)} = \mathbb{I}[t^{(i)} = k]$$

ISOTROPIC DECISION BOUNDARIES

Assuming spherical/isotropic $\Sigma = \sigma^2 I$, we only need one parameter

For simplicity, assume uniform $p(t)$.

$$\log p(t_k | x) - \log p(t_\ell = x) = -\frac{1}{2\sigma^2}\left[(x-\mu_k)^T(x-\mu_k) - (x-\mu_\ell)^T(x-\mu_\ell)\right]$$
$$= -\frac{1}{2\sigma^2}\left[\|x-\mu_k\|^2 - \|x-\mu_\ell\|^2\right]$$

PRINCIPAL COMPONENT ANALYSIS

Dimensionality reduction — map data to a lower dimensional space so that only meaningful information is retained, ideally close to intrinsic dimension

Save computation/memory, reduce overfitting, better generalization

Principal component analysis (PCA) is an unsupervised learning algorithm

linear dimensionality reduction, projection

To project onto K-dimensional subspace, choose orthonormal basis $\{u_1, u_2, \dots, u_K\}$ for S

project onto each unit vector individually, then sum

$$\text{Proj}_S(x) = \sum_{i=1}^K z_i u_i \text{ where } z_i = x^T u_i, \qquad \text{Proj}_S(x) = Uz \text{ where } z = U^T x$$

In math terminology, we want to project onto affine spaces, can have arbitrary origin $\hat{\mu}$

So, we instead have $\tilde{x} = Uz + \hat{\mu}$, called the reconstruction of x

(z: representation/code)

If data points $x \in \mathbb{R}^D$ lie close to reconstructions $z \in \mathbb{R}^K$, we can approximate distances

Representation learning is learning a representation to a space that is easier to manipulate & visualize

LEARNING A SUBSPACE

A good subspace has $\hat{\mu}$ is empirical mean of data

$D \times K$ matrix U w/ orthonormal columns

equivalent:

- Minimize reconstruction error $\min_U \frac{1}{N}\sum_{i=1}^N \|x^{(i)} - \tilde{x}^{(i)}\|^2$
- Maximize variance of reconstructions $\max_U \frac{1}{N}\sum_i^N \|\tilde{x}^{(i)} - \hat{\mu}\|^2$

Emperical covariance matrix $\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^N (x^{(i)} - \hat{\mu})(x^{(i)} - \mu)^T$ is symmetric & PSD.

Principal components are eigenvectors, & optimal PCA subspace is spanned by top K of those.

↳ $K=1$, $\frac{1}{N}\sum \|\tilde{x}^{(i)} - \hat{\mu}\|^2 = \cdots = u^T \hat{\Sigma} u = u^T Q \Lambda Q^T u = a^T \Lambda a = \sum_{j=1}^D \lambda_j a_j^2$

Assuming λ_i sorted $(\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D)$, the k^{th} principal component is the k^{th} eigenvector of Σ

MATRIX FACTORIZATION

Assume data $x^{(i)} \approx \tilde{x}^{(i)} = Uz^{(i)}$ centered at $\hat{\mu} = 0$. In matrix form $X \approx ZU^T$

Squared reconstruction error is $\sum_{i=1}^{N} \|x^{(i)} - Uz^{(i)}\|^2 = \|X - ZU^T\|_F^2$ (Frobenius norm)

PCA approximates $X \approx ZU^T$, gives optimal low-rank matrix factorization to matrix X.

Similar to Singular Value Decomposition of $X = U\Sigma V^T$

- U $m \times m$ matrix, orthonormal columns
- V $n \times n$ orthonormal matrix
- Σ $m \times n$ diagonal matrix, $\sigma_1, \sigma_2, \ldots, \sigma_{\min\{m,n\}} \geq 0$

Sparse data matrix $R \approx UZ^T$, need to predict missing entries

Let $O = \{(n,m)$: entry (n,m) of matrix R is observed$\}$, squared error loss $\min_{U,Z} \frac{1}{2} \sum_{(i,j) \in O} (R_{ij} - u_i^T z_j)^2$

ALTERNATING LEAST SQUARES

Fix Z & optimize U, then fix U & optimize Z, until convergence

Decompose cost into sum of independent terms

$$\sum_{(i,j) \in O} (R_{ij} - u_i^T z_j)^2 = \sum_i \sum_{j:(i,j) \in O} (R_{ij} - u_i^T z_j)^2$$

1. Initialize U, Z randomly
2. Repeat until convergence: values stop changing
3. for $i = 1, \ldots, N$:
4. $u_i = \left(\sum_{j:(i,j) \in O} z_j z_j^T\right)^{-1} \sum_{j:(i,j) \in O} R_{ij} z_j$
5. for $j = 1, \ldots, M$:

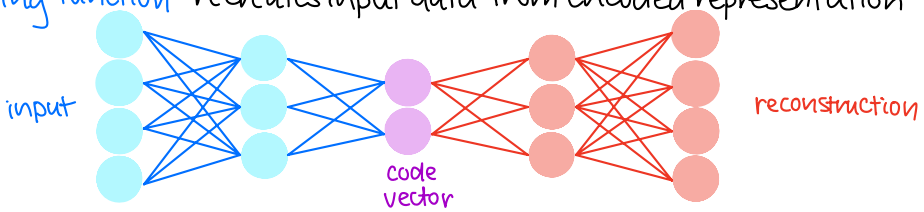
$z_j = \left(\sum_{i:(i,j) \in O} u_i u_i^T\right)^{-1} \sum_{i:(i,j) \in O} R_{ij} u_i$

AUTOENCODER

Autoencoder feed-forward NN whose job is to take input x & predict x (unsupervised)

Learns encoding function that transforms input data, hopefully removes noise

decoding function recreates input data from encoded representation



Why autoencoders?

Map high-dimensional data to 2 dim for visualization,

Learn features in unsupervised way to apply to supervised task

LINEAR

Simpliest autoencoder has 1 hidden layer, linear activations, squared error loss $\mathcal{L}(x, \tilde{x}) = \|x - \tilde{x}\|^2$

Optimal, min, error K-dimensional subspace in terms of reconstruction error is PCA subspace.

$$x \in \mathbb{R}^D \xrightarrow[\text{encoder}]{W_1 = U^T} z \in \mathbb{R}^K \xrightarrow[\text{decoder}]{W_2 = U} \tilde{x} \in \mathbb{R}^D$$

$K < D$ dimensionality reduction

optimal weights are principal components

NONLINEAR

Deep nonlinear autoencoders learn to project data onto a nonlinear manifold

2 units → 100 units → 1 unit → 100 units → 2 units

Nonlinear autoencoders can learn more powerful codes for a given dimensionality vs. linear (PCA)

PCA, matrix factorization, autoencoders are all latent variable models

assume data depends on some latent variables that are never used

high dimension
↓
low dimension

K-MEANS

Clustering grouping data points in clusters w/ no observed labels (unsupervised)

Sometimes data forms clusters, samples within a cluster are similar to each other

Multimodal distribution, multiple **modes** (regions of high probability mass)

In **k-means**, there are K clusters, each point is close to its cluster / mean of cluster

Randomly initialize cluster centers

Alternate between 2 steps until convergence:

Assignment assign each data point to closest center

Refitting move each cluster center to mean of its members

Objective Find cluster centers m & assignments r to minimize sum of squared distances of data points $\{x^{(n)}\}$ to their assigned centers

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

where $r_k^{(n)} = 1$ means $x^{(n)}$ is assigned to cluster k w/ center m_k

$$= \begin{cases} 0 & \text{if } x^{(n)} \text{ in } k^{th} \text{ cluster} \\ \|m_k - x^{(n)}\|^2 & \text{if } x^{(n)} \text{ not in} \end{cases}$$

Assignment $r = \begin{cases} 1 & \text{if } k = \arg\min_j \|m_j - x^{(n)}\|^2 \\ 0 & \text{otherwise} \end{cases}$

minimize w.r.t. $\{r_k^{(n)}\}$

Refitting $m_\ell = \dfrac{\sum_n r_\ell^{(n)} x^{(n)}}{\sum_n r_\ell^{(n)}}$ (sum of points in ℓ / # points in cluster ℓ)

minimize w.r.t. $\{m_k\}$

This is **alternating minimization** / **block coordinate descent**

K-means reduces cost at each iteration. If assignments do not change at assignment step, converged at local min.

Objective J is non-convex, not guarateed to converge at global minimum. Could try many random starting points

SOFT K-MEANS

Make **soft assignments** instead. $r_k^{(n)} = \dfrac{\exp[-\beta\|m_k - x^{(n)}\|^2]}{\sum_j \exp[-\beta\{\|m_k - x^{(n)}\|^2\}_{k=1}^K}$ $= \text{softmax}(-\beta\{\|m_k - x^{(n)}\|^2\}_{k=1}^K)$

Initialization & refitting are the same.

↳ one cluster might have responsibility of 0.7 for a datapoint, another may have 0.3

As $\beta \to \infty$, soft k-means becomes normal k-means.

GUASSIAN MIXTURE MODELS

If $p(z)$ in a **latent variable model** $p(x) = \sum_z p(x,z) = \sum_z p(x|z)\,p(z)$ is a categorical distribution, it is a **mixture model**. Different values of z corresponds to different **components**

Gaussian mixture model (GMM) represents a distribution as

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

w/ the **mixing coefficents** π_k, where $\forall k$, $\sum_{k=1}^K \pi_k = 1$ and $\pi_k \geq 0$.

Defines a density over x, fit parameters using maximum likelihood.

GMM are universal approximators of densities.

For $i = 1, \ldots, N$:

$z^{(i)} \sim \text{categorical}(\pi)$

$x^{(i)}|z^{(i)} \sim \mathcal{N}(\mu_z^{(i)}, \Sigma_{z^{(i)}})$

How do we choose π_k, μ_k, Σ_k?

Given data $D = \{x^{(n)}\}_{n=1}^N$, choose parameters to maximize $\log p(D) = \sum_{n=1}^N \log p(x^{(n)})$

Find $p(x)$ by marginalizing out z: $p(x) = \sum_{k=1}^K p(z=k|x) = \sum_{k=1}^K p(z=k)\,p(x|z=k)$

EXPECTATION-MAXIMIZATION ALGORITHM

TLDR: fitting GMM w/ max likelihood, objective has no closed-form solution. Solution is invariant to permutations.

$$\log p(X;\theta) = \sum_{i=1}^{N} \log p(x^{(i)}|\theta) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} p(x^{(i)}|z^{(i)};\{\mu_k\},\{\Sigma_k\}) p(z^{(i)}|\pi)$$ log outside sum, so cannot simplify

Expection-Maximization algorithm alternates between 2 steps

Expectation (E-step) Compute posterior probability over z given current model. Given θ, compute $z^{(i)}$

How much we think Gaussian generates each datapoint.

Assign responsibility $r_k^{(i)}$ of component k for data point i using posterior probability.

$$r_k^{(i)} = Pr(z^{(i)}=k|x^{(i)};\theta)$$

Maximization (M-step) Change parameters of each Gaussian to maximize probability it would generate data it is currently responsible for. Given $z^{(i)}$, learn θ.

Apply max likelihood updates, where each component is fit w/ weighted dataset. Weights $\propto$ responsibilities.

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

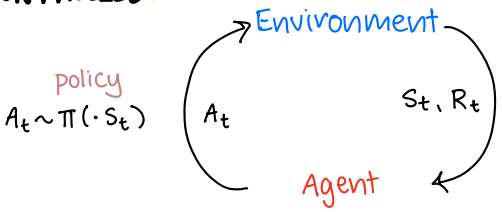
Both k-Means & EM algorithms rely on alternating optimization methods & can suffer from bad local optima.

REINFORCEMENT LEARNING

You better not fuck up this part or else you don't go back

Sequential decision-making Agent chooses sequence of actions which affect future possibilities available to agent.

MARKOV DECISION PROCESS



State current status of agent

$S_{t+1} \sim P(\cdot|S_t, A_t)$ transition probabilities

$R_t \sim R(\cdot|S_t, A_t)$

reward how good current action is based on agent's location

Markov assumption says state at time $t+1$ depends directly on state & action at time t only (not previous)

Agent takes an action according to policy. Environment updates state & gives reward. Rinse & repeat.

Policy how the agent chooses action

Deterministic $A_t = \pi(S_t)$ for some function $\pi: \mathcal{S} \to \mathcal{A}$

Stochastic $A_t \sim \pi(\cdot|S_t)$ for some function $\pi: \mathcal{S} \to P(\mathcal{A})$ set of distributions over actions

distribution over rollouts/trajectories factorizes:

$$p(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\pi(a_1|s_1)P(s_2|s_1,a_1)\pi(a_2|s_2)\cdots P(s_T|s_{T-1},a_{T-1})\pi(a_T|s_T)$$

Each time step, agent receives a reward from distribution that relies on current state & action $R_t \sim R(\cdot|S_t, A_t)$

Return determines how good outcome of an episode is $G_1 = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots$

Discount factor γ $(0 \le \gamma \le 1)$ determines how much we care about future rewards

Goal is to maximize expected return $\mathbb{E}[G_1]$

MDP defined by ($\mathcal{S}$ state space, $\mathcal{A}$ action space, P transition position, R immediate reward distribution, γ discount factor)

FINDING A POLICY

Planning given a fully specified MDP

↳ technically chess (finite states)

Learning agent interacts w/ environment w/ unknown dynamics

environment is black box

↳ Super Mario

Value function V^π for policy π measures expected return if you start in state s and follow π. Measures desirability of s.

$$V^\pi(s) \triangleq \mathbb{E}_\pi[G_t|S_t=s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right]$$

Many value functions satisfy Bellman equation

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}[R_t + \gamma G_{t+1} \mid S_t = s]$$
$$= \sum_a \pi(a|s)\left[r(s,a) + \gamma \sum_{s'} P(s'|a,s) V^\pi(s')\right]$$

Viewing V^π as vector (entries corresponds to states), define Bellman backup operator T^π

$$(T^\pi V)(s) \triangleq \sum_a \pi(a|s)\left[r(s,a) + \gamma \sum_{s'} P(s'|a,s) V(s')\right]$$

Bellman equation can be seen as fixed point of Bellman operator $T^\pi V^\pi = V^\pi$

State-action function Q^π pretty much the same except it's for a specific action

$$Q^\pi(s,a) \triangleq \mathbb{E}_\pi\left[\sum_{k\geq 0} \gamma^k R_{t+k} \mid S_t = s, A_t = a\right] = r(s,a) + \gamma \mathbb{E}_{s'\sim P(s'|s,a)}[Q^\pi(s', \pi(s))]$$

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a), \quad Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|a,s) V^\pi(s')$$

Bellman $Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|a,s) \sum_{a'} \pi(a'|s') Q^\pi(s',a')$

DYNAMIC PROGRAMMING + VALUE ITERATION

If deterministic policy π is optimal, then $\pi(s) = \operatorname{argmax}_a Q^\pi(s,a)$ — always pick action that gives highest expected return

Bellman equation for optimal policy π^*, $Q^* = Q^{\pi^*}$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a') \triangleq (T^* Q^*)(s,a)$$

is sufficient to characterize optimal policy.

Need to solve fixed point equation $T^*Q^* = Q^*$, then we can choose $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$.

Keep iterating backup operator repeatedly

$Q \leftarrow T^\pi Q$ policy eval.

$Q \leftarrow T^* Q$ finding optimal policy

Treat Q^π or Q^* as vector w/ $|S| \cdot |A|$ entries.

Example of dynamic programming

Value iteration start from initial function Q_1. For each $k = 1, 2, \dots$ apply $Q_{k+1} \leftarrow T^* Q_k$

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \max_{a'\in A} Q_k(s',a')$$

A fixed point of this update is exactly the solution of optimal Bellman equation

Bellman operator is contration map, we can just keep repeatedly apply it & we converge to unique fixed point.

Assumes known dynamics, environment is known, $|S|$ & $|A|$ are finite

Q-LEARNING

For optimal Bellman equation, we need to know dynamics to evaluate expectation.

Temporal Difference Learning (TD) updating predictions to match later predictions, once we have more info

Monte Carlo Estimation of an expectation $\mu = \mathbb{E}[X]$: $\mu \leftarrow \mu + \alpha(X - \mu)$ repeatedly sampling & updating

Apply MC to Bellman equation by sampling $s' \sim P(\cdot|s,a)$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)}_{\text{Bellman error}}]$$

If the agent always takes the action w/ highest state-action value (exploitation), some states are never visited

ε-greedy policy picks $\operatorname{argmax}_a Q(s,a)$ w/ probability $1-\varepsilon$, random action w/ probability ε

exploitation — exploration — tradeoff

We did this for chess AI in CSC111

Q-learning combines MC, value iteration, & ε-greedy

Hyperparameters: learning rate α, exploration parameter ε = 0.05 usually

Initialize $Q(s,a)\ \forall (s,a) \in \mathcal{S}, \mathcal{A}$

Start at state s_0.

For each time step $t = 0, 1, \ldots$

Choose A_t according to ε-greedy policy $\quad A_t \leftarrow \begin{cases} \text{argmax}_{a \in \mathcal{A}} Q(s,a) & \text{probability } 1-\varepsilon \\ \text{uniform random action in } \mathcal{A} & \text{probability } \varepsilon \end{cases}$

Take action A_t in environment

Change state to $S_{t+1} \sim P(\cdot \mid S_t, A_t)$

Observe S_{t+1} & R_t (could be $r(S_t, A_t)$ or stochastic)

Update action-value function at state-action (S_t, A_t)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

Q-learning does not use policy anywhere, so it is an off-policy algorithm

Policy gradient is example of on-policy. Harder to encourage exploration

FUNCTION APPROXIMATION

Guess what, we're doing back prop. again.

Impractical to store tabular representation of Q for all but simplest problems, so we approximate Q instead.

Linear approx. $Q(s,a) = w^T \psi(s,a)$, compute Q w/ a neural net.

Update Q using back propagation.

$$t \leftarrow r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)$$
$$\theta \leftarrow \theta + \alpha (t - Q(s,a)) \nabla_\theta Q(s_t, a_t)$$