Basics

Gradient descent

$\mathbf{w}_0 \in \mathbb{R}^d, \mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \hat{R}(\mathbf{w}_t)$ Adaptive step size by line search (optimizing step size every step) or bold driver heuristic (if function decreases, increase / vice-versa). Stochastic (SGD)=Evaluate only one randomly chosen point ($w_{t+1} = w_t - \eta_t \nabla l(w_t, x_1, y_1)$). Minibatch=average over multiple randomly selected points. Momentum:
$a \leftarrow m \cdot a + \eta_t \nabla_{\mathbf{W}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$ and $\mathbf{W} \leftarrow \mathbf{W} - a$

Gaussian distribution

$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$

Multivariate Gaussian

$f(x) = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$

Expectations

$\mathbb{E}[X] = \sum_x p(x) * x$ (discrete) / $\int p(x) * x dx$ (continuous) $\mathbb{E}[f(X)] = \sum_x p(x) * f(x)$ (discrete) / $\int p(x) * f(x) dx$ (continuous)

Expected Error

Generalization: minimize the expected error
$R(w) = \int P(x,y)(y - w^T x)^2 dx dy$
$= \mathbb{E}_{x,y}[(y - w^T x)^2]$

Jensen's Inequality

X is a random variable & φ a convex function then the following holds $\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$

Convex

g(x) is convex $\Leftrightarrow x_1, x_2 \in \mathbb{R}, \lambda \in [0,1]$:
$g(\lambda x_1) + (1-\lambda x_2) \leq \lambda g(x_1) + (1-\lambda) g(x_2) \Leftrightarrow g''(x) > 0$

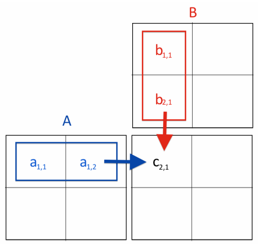
Standardization

$\tilde{x}_{i,j} = \left(x_{i,j} - \hat{\mu}_j\right)/\hat{\sigma}_j$ where $\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^n x_{i,j}$ and $\hat{\sigma}_j^2 = \frac{1}{n}\sum_{i=1}^n \left(x_{i,j} - \hat{\mu}_j\right)^2$

Feature selection

Greedy Forward / Backward: Add / remove until val. error increases. Alternative: L1

Matrix product



Semi-positive-definite Matrices

$M \in \mathbb{R}^{n\times n}$ is SPD $\Leftrightarrow$
$\forall x \in \mathbb{R}^n : x^T M x \geq 0 \Leftrightarrow$
all eigenvalues of M are positive ≥ 0 For a 2×2 matrix, $\lambda_{1,2} = \frac{tr(A) \pm \sqrt{tr(A)^2 - 4det(A)}}{2}$ where $tr(A) = a_{1,1} + a_{2,2}$, $det(A) = a_{1,1} * a_{2,2} - a_{1,2} * a_{21}$

Law of total probability

$\Pr(A) = \sum_n \Pr(A|B_n)\Pr(B_n)$

Bayes rule / Conditional probability

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ and $P(A|B) = \frac{P(A\cap B)}{P(B)}$

Two norm

$\|x\|_2 = \sqrt{x^T x}$

Regression

Problem: $\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^n \left(y_i - \mathbf{w}^T \mathbf{x}_i\right)^2$ Closed form: $\mathbf{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$ or gradient descent with $\eta_t = 0.5, \nabla_w \hat{R}(\mathbf{w}) = -2\sum_{i=1}^n (y_i - \mathbf{w}^T\mathbf{x_i}) \cdot \mathbf{x_i} = 2X^T(\mathbf{Xw} - \mathbf{y})$

Regularization

L2 / Ridge regression: $+\lambda \|\mathbf{w}\|\|_2^2$ $(\equiv +\lambda \sum_i w_i^2)$

(Analytical solution: $\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$)

L1 / Lasso: $+\lambda\|\mathbf{w}\|_1$ (leads to sparse solutions, surrogate for L0 i.e. number of non-zero)

Classification

Perceptron

0/1 loss not convex / differentiable, surrogate: $\ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max\left(0, -y_i \mathbf{w}^T \mathbf{x}_i\right)$ Perceptron algorithm: SGD on Perceptron with $\eta_t = 1$. Will obtain linear separator

Support Vector Machine

Max. margin linear classifier with hinge loss: $\hat{R}(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \lambda\|w\|_2^2$

Imbalanced Data

Solutions: Subsampling (remove examples from majority class) / upsampling (repeat data points from minority class) or cost-sensitive loss function: $\ell_{CS}(\mathbf{w}; \mathbf{x}, y) = c_y \ell(\mathbf{w}; \mathbf{x}, y)$ For evaluation: Accuracy = $\frac{TP+TN}{n}$, Precision = $\frac{TP}{TP+FP}$, Recall (TPR) = $\frac{TP}{TP+FN}$, F1 score = $\frac{2TP}{2TP+FP+FN}$, FPR = $\frac{FP}{TN+FP}$. Higher precision $\Leftrightarrow$ lower recall. ROC curve: TPR against FPR

Multi-class Problems

One-vs-all: Train c binary classifiers, classify using classifier with largest confidence. One-vs-one: Train $c(c-1)/2$ binary classifiers for each class pair. Apply voting scheme (class with highest number of predictions). Multi-class hinge loss: $\max\left(0, 1 + \max_{j\in\{1,\ldots,y-1,y+1,\ldots,c\}} \mathbf{w}^{(j)T}\mathbf{x} - \mathbf{w}^{(y)T}\mathbf{x}\right)$

Kernels

Reformulating Perceptron

Ansatz: $w = \sum_{j=1}^n \alpha_j y_j x_j$
$\min_{w\in\mathbb{R}^d} \sum_{i=1}^n \max[0, -y_i w^T x_i]$
$= \min_{\alpha_{1:n}} \sum_{i=1}^n \max[0, -y_i(\sum_{j=1}^n \alpha_j y_j x_j)^T x_i]$
$= \min_{\alpha_{1:n}} \sum_{i=1}^n \max[0, -\sum_{j=1}^n \alpha_j y_i y_j (x_i^T x_j)]$

Example Kernels

$k_1(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^m$ for all monomials of deg m
$k_2(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^m$ monomials up to deg m
Gaussian / RBF: $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|_2^2 / 2h^2\right)$
Laplacian: $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|_1 / h)$

Kernelized Perceptron

1. Initialize $\alpha_1 = \ldots = \alpha_n = 0$
2. For t do: Pick data $(x_i, y_i) \in_{u.a.r} D$
Predict $\hat{y} = sign(\sum_{j=1}^n \alpha_j y_j k(x_j, x_i))$
If $\hat{y} \neq y_i$ set $\alpha_i = \alpha_i + \eta_t$

Properties of kernel functions

k must be symmetric, the kernel matrix must be SPD.

Kernel Matrix

The kernel matrix K is SPD
$$K = \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \ldots & k(x_n, x_n) \end{bmatrix}$$
$\left(XX^T\right)$ for inner product as kernel.

Kernel Engineering

$k_1(x,y) + k_2(x,y)$
$k_1(x,y) \cdot k_2(x,y)$
$c \cdot k_1(x,y)$ for $c > 0$
$f(k_1(x,y))$, where f is exponential/polynomial with positive coefficents

Parametric to nonparametric linear regression

Ansatz: $w = \sum_i \alpha_i x$
Parametric: $w^* = \underset{w}{\operatorname{argmin}} \sum_i (w^T x_i - y_i)^2 + \lambda\|w\|_2^2$
$= \underset{\alpha_{1:n}}{\operatorname{argmin}} \sum_{i=1}^n (\sum_{j=1}^n \alpha_j x_j^T x_i - y_i)^2 + \lambda \sum_i \sum_j \alpha_i \alpha_j (x_i^T x_j)$
$= \underset{\alpha_{1:n}}{\operatorname{argmin}} \sum_{i=1}^n (\alpha^T K_i - y_i)^2 + \lambda \alpha^T K \alpha$
$= \underset{\alpha}{\operatorname{argmin}} \|\alpha^T K - y\|_2^2 + \lambda \alpha^T K \alpha$
Closed form: $\alpha^* = (K + \lambda I)^{-1} y$
Prediction: $y^* = w^{*T} * x = \sum_{i=1}^n \alpha_i^* k(x_i, x)$

Neural Networks

Parameterize the feature maps and optimize over the parameters:
$\mathbf{w}^* = \arg\min_{\mathbf{w},\theta} \sum_{i=1}^n \ell\left(y_i; \sum_{j=1}^m w_j \phi\left(\mathbf{x}_i, \theta_j\right)\right)$ with $\phi(\mathbf{x}, \theta) = \varphi\left(\theta^T \mathbf{x}\right)$

Activation functions

Sigmoid: $\varphi(z) = \frac{1}{1+\exp(-z)}$ $(\varphi'(z) = \varphi(z)(1 - \varphi(z))$, tanh: $\varphi(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$ $(\varphi'(z) = 1 - \tanh^2(z))$, ReLU: $\varphi(z) = \max(z, 0)$

Backpropagation

For each unit j on the output layer:
- Compute error signal: $\delta_j = \ell_j'(f_j)$
- For each unit i on layer L: $\frac{\partial}{\partial w_{j,i}} = \delta_j v_i$

For each unit j on hidden layer $l = \{L-1, .., 1\}$:
- Error signal: $\delta_j = \phi'(z_j) \sum_{i\in Layer_{l+1}} w_{i,j} \delta_i$
- For each unit i on layer $l-1$: $\frac{\partial}{\partial w_{j,i}} = \delta_j v_i$

(Where v_i is i-th unit of prev. layer)

Overfitting

Early stopping (don't run until convergence), regularization $(+\lambda\|W\|_F^2)$ or Dropout (train: randomly ignore hidden units with prob. p)

Batch Normalization

Normalize input to each layer: $\mu_B = \frac{1}{m}\sum_{i=1}^m x_i$
$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^m (x_i - \mu_B)^2$ $\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
$\text{BN}_{\gamma,\beta}(x_i) = \gamma \hat{x_i} + \beta$

Convolutional neural networks

Convolution layer (filter(s) with learned weights applied as dot-matrix product), pooling layers (subsampling with average / max. value) + fully connected layers. Stride = Amount of shifting of the filter, sometimes padding needed. For a $f \times f$ filter to a $n \times n$ image with padding p, stride s the output height / length is $\frac{n+2p-f}{s} + 1$

Unsupervised learning

Clustering / k-means

Each cluster has center μ_i, goal: $\arg\min_\mu \sum_{i=1}^n \min_{j\in\{1,\ldots,k\}} \left\|\mathbf{x}_i - \mu_j\right\|_2^2$. Algorithm: Initialize centers, while not converged; assign each point to closest center and update center as mean of assigned data points. Initialization with k-Means++: Start with random data point as center, add centers 2 to k randomly, proportionally to squared distance to closest center.

Dimension Reduction / PCA

Given centered data, the solution to the PCA problem
$\arg\min \sum_{i=1}^n \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$ $(\mathbf{W} \in \mathbb{R}^{d\times k}, \mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k)$ is given by $\mathbf{W} = (\mathbf{v}_1 | \ldots | \mathbf{v}_k)$ and $\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$, where $\Sigma = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T = \sum_{i=1}^d \lambda_i \mathbf{v}_i \mathbf{v}_i^T$ For the SVD $X = USV^T$ $(X \in \mathbb{R}^{n\times d}$, $U \in \mathbb{R}^{n\times n}$ and

$V \in \mathbb{R}^{d\times d}$ both orthogonal, $S \in \mathbb{R}^{n\times d}$ diagonal), the top k principal components are the first k columns of V.

Kernel PCA

The Kernel Principal Components are given by $\alpha^{(1)},\dots,\alpha^{(k)} \in \mathbb{R}^n$ where $\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}}\mathbf{v}_i$ and $\mathbf{K} = \sum_{i=1}^n \lambda_i \mathbf{v}_i \mathbf{v}_i^T$. A new point $\mathbf{x}$ is projected as $z_i = \sum_{j=1}^n \alpha_j^{(i)} k(\mathbf{x}_j, \mathbf{x})$

Autoencoders

Try to learn identity function $x \approx f(\mathbf{x};\theta) = f_2(f_1(\mathbf{x};\theta_1);\theta_2)$ where f_1 is the encoder, f_2 the decoder.

Probabilistic Modeling

Goal

Given data $D = \{(x_1,y_1),\dots,(x_n,y_n)\} \subseteq X \times Y$

Want to find the hypothesis with the minimum prediction error (risk). $R(h) = \int P(x,y) l(y;h(x)) dx dy = \mathbb{E}_{x,y}[l(y;h(x))]$

Fundamental assumption: $(x_i, y_i) \in_{iid} X \times Y$

Maximum Likelihood Estimation (MLE)

Choose a particular parametric form $P(Y|X,\theta)$, then optimize the parameters using Maximum Likelihood Estimation.

$\theta^* = \underset{\theta}{\operatorname{argmax}} P(y|x,\theta)$

$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^n P(y_i|x_i,\theta)$ (iid)

$= \underset{\theta}{\operatorname{argmin}} - \sum_{i=1}^n log P(y_i|x_i,\theta)$

Regression

Hypothesis minimizing error for least squares regression: conditional mean $h^*(x) = \mathbb{E}[y|X = x]$ If we assume: $Y = w^T X + \epsilon, \epsilon \in \mathcal{N}(0,\sigma^2) \Leftrightarrow y_i \in \mathcal{N}(w^T x_i, \sigma^2)$

Maximizing the log likelihood:

$\underset{w}{\operatorname{argmax}} P(y|x,\theta) = \underset{w}{\operatorname{argmax}} \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - h(x_i))^2}{\sigma^2}}$

log is monotonic and cancel all constants

$= \underset{w}{\operatorname{argmin}} \sum_i (y_i - w^T x_i)^2$

Bias Variance Tradeoff

Prediction error = $Bias^2 + Variance + Noise$ where Bias = Excess risk of the best model under consideration (compared to lowest risk knowing $P(X,Y)$), Variance = Risk incurred due to estimating model from limited data and Noise = irreducible error. Complex models have a high variance / low bias and vice-versa for simple ones. Want to achieve middle ground.

Maximum a posteriori estimate (MAP)

Introduce bias by expressing assumption through a Bayesian prior $w_i \in \mathcal{N}(0,\beta^2)$ Bayes rule: $P(w|x,y) = \frac{P(w|x)P(y|x,w)}{P(y|x)} = \frac{P(w)P(y|x,w)}{P(y|x)}$, we assume w is independent of x.

Example MAP for lin. Gaussian

$\underset{w}{\operatorname{argmax}} P(w|x,y) =$

$\underset{w}{\operatorname{argmin}} - log P(w) - log P(y|x,w) + const.$

$= \underset{w}{\operatorname{argmin}} \frac{1}{2\beta^2}\|w\|_2^2 + \frac{1}{2\sigma^2}\sum_{i=1}^n (y_i - w^T x_i)^2$

$= \underset{w}{\operatorname{argmin}} \lambda \|w\|_2^2 + \sum_{i=1}^n (y_i - w^T x_i)^2$, $\lambda = \frac{\sigma^2}{\beta^2}$

Logistic Regression

Estimate $P(Y|X)$ by link function $\sigma(w^T x) = \frac{1}{1+exp(-w^T x)}$. Assumption $Y|X \sim Ber(\sigma(w^T x))$ (i.i.d. Bernoulli noise). Learning: $w = \underset{w}{\operatorname{argmax}} P(w|x,y)$ Classification: Use $P(y|x,w) = \frac{1}{1+exp(-yw^T x)}$ and predict most likely class label.

Can use L2 (equiv. to Gaussian prior) / L1 (equiv. to Laplace prior) on logistic loss.

Example: MLE for Logistic Regression

$\underset{w}{\operatorname{argmax}} P(y|x,w) = \underset{w}{\operatorname{argmin}} - \sum_{i=1}^n log P(y|x_i,w)$

$\underset{w}{\operatorname{argmin}} \sum_{i=1}^n log(1 + exp(-yw^T x_i))$ (logistic loss)

Gradient for Logistic Regression

Loss function $l(w) = log(1 + exp(-yw^T x))$

$\nabla_w l(w) = \frac{1}{1+exp(-yw^T x)} exp(-yw^T x)(-yx)$

$= \frac{1}{1+exp(yw^T x)}(-yx)$

$= P(-y|x,w)(-yx)$

Kernelized Logistic Regression

Learning:

$\hat{\alpha} = \arg\min_\alpha \sum_{i=1}^n \log\left(1 + \exp\left(-y_i \alpha^T \mathbf{K}_i\right)\right) + \lambda \alpha^T \mathbf{K} \alpha$ Classification: $\hat{P}(y|\mathbf{x},\hat{\alpha}) = \frac{1}{1+\exp\left(-y\sum_{j=1}^n \alpha_j k(\mathbf{x}_j,\mathbf{x})\right)}$

Multi-class Logistic Regression

$P(Y = i|\mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_c) = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_{j=1}^c \exp(\mathbf{w}_j^T \mathbf{x})}$ Corresponding loss $(-\log P(Y = y|\mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_c))$ is cross-entropy loss.

Bayesian decision theory

Given:

- Conditional distribution over labels $P(y|x)$
- Set of actions $\mathcal{A}$
- Cost function $C : Y \times \mathcal{A} \to \mathbb{R}$

Pick action that minimizes the expected cost:

$a^* = \underset{a\in\mathcal{A}}{\operatorname{argmin}} \mathbb{E}_y[C(y,a)|x] = \sum_y P(y|x) * C(y,a)$

Example: Asymmetric costs

Est. cond. dist: $P(y|x,w) = Ber(\sigma(w^T x))$ Action set: $\mathcal{A} = \{+1,-1\}$ Cost function: $C(y,a) = \begin{cases} c_{FP}, \text{ if } y = -1 \text{ and } a = +1 \\ c_{FN}, \text{ if } y = +1 \text{ and } a = -1 \\ 0, \text{ otherwise} \end{cases}$ The action that minimizes the expected cost is:

$C_+ = \mathbb{E}_y[C(y,+1)|x] = P(y = +1|x) \cdot 0 + (P(y = -1|x) \cdot c_{FP}$

$C_- = \mathbb{E}_y[C(y,-1)|x] = P(y = +1|x) \cdot c_{FN} + P(y = -1|x) \cdot 0$

Predict +1 if $C_+ \leq C_- \Leftrightarrow P(y = +1|x) \geq \frac{c_{FP}}{c_{FP}+c_{FN}}$

Generative Modeling

Discriminative models: aim to estimate $P(y|x)$ generative models: aim to estimate joint distribution $P(y,x)$ $(= P(x|y) * P(y))$

Typical approach:

- Estimate prior on labels $P(y)$
- Estimate conditional distribution for each class y $P(x|y)$
- Obtain predictive distribution using Bayes rule $P(y|x) = \frac{1}{P(x)} P(y)P(x|y)$

Naive Bayes Model

Class labels are modelled as categorical variable ($P(Y = y) = p_y$), features as conditionally independent given Y: $P(X_1,\dots,X_d|Y) = \prod_{i=1}^d P(X_i|Y)$ Gaussian Naive Bayes assumes $P(x_i|y) = \mathcal{N}\left(x_i|\mu_{y,i}, \sigma_{y,i}^2\right)$ GNB with shared variance between the two classes produces a linear classifier and will (if model assumptions met) make same predictions as Logistic Regression.

Gaussian Bayes Classifiers

Class labels are still model as categorical variable, but features generated by multivariate Gaussian: $P(\mathbf{x}|y) = \mathcal{N}\left(\mathbf{x};\mu_y,\Sigma_y\right)$. If $c = 2, p = 0.5$ and the covariances are equal, produces a linear classifier (Fisher's LDA). LDA can be viewed as a projection to a 1-dim. subspace that maximizes ratio of between-class and within-class variances, whereas PCA($k = 1$) maximizes variance of the resulting 1-dim. projection.

Categorical Naive Bayes Classifier

MLE class prior: $P(Y = y) = \frac{Count(Y=y)}{n}$

MLE for feature dist.:

$P(X_i = c|y) = \frac{Count(X_i=c,Y=y)}{Count(Y=y)}$

Overfitting

Prior over parameters $(P(Y = 1) = \theta)$ can be used and compute posterior distribution $P(\theta|y_1,\dots,y_n)$. Pair of prior distributions and likelihood functions is conjugate if the posterior distribution remains in the same familiy as the prior.

Outlier Detection

$P(x) = \sum_y P(x,y) = \sum_y P(y)P(x|y) \leq \tau$

Generative Adversarial Networks

A generator G and a discriminator D is simultaneously trained. Training requires finding a saddle point.

Missing Data / Latent models

Gaussian Mixtures

Convex-combination of Gaussian distributions: $P(\mathbf{x}|\theta) = P(\mathbf{x}|\mu,\Sigma,\mathbf{w}) = \sum_{i=1}^c w_i \mathcal{N}(\mathbf{x};\mu_i,\Sigma_i)$. To fit -> Hard-EM / Soft-EM.

Hard-EM algorithm

Initialize $\theta^{(0)}$, for $t = 1,2,..$: E-step: Predict most likely class for each data point $z_i^{(t)} = \arg\max_z P\left(z|\mathbf{x}_i,\theta^{(t-1)}\right)$ M-step: Compute MLE (with the now complete data). k-Means can be understood as special case (uniform weights over mixture components, identical spherical covariance matrices).

Soft-EM algorithm

Let $\gamma_j(\mathbf{x}) = P(Z = j|\mathbf{x},\Sigma,\mu,\mathbf{w}) = \frac{w_j P(\mathbf{x}|\Sigma_j,\mu_j)}{\sum_\ell w_\ell P(\mathbf{x}|\Sigma_\ell,\mu_\ell)}$. At E-step, calculate $\gamma_j^{(t)}(\mathbf{x}_i)$ using values from prev. iteration. At M-step: $w_j^{(t)} \leftarrow \frac{1}{n}\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)$, $\mu_j^{(t)} \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)}$, $\Sigma_j^{(t)} \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)\left(\mathbf{x}_i - \mu_j^{(t)}\right)\left(\mathbf{x}_i - \mu_j^{(t)}\right)^T}{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)}$

Gaussian-Mixture Bayes Classifiers

Models $P(\mathbf{x}|y)$ as Gaussian mixture model.

Semi-supervised learning

Easy with GMMs, just set $\gamma_j(\mathbf{x}_i)$ to 1 if a label exists and is equal to j.

Theory

EM algorithm is equivalent to calculate the expected complete data log-likelihood (where $z_{1:n}$ are missing data) in the E-Step: $Q\left(\theta;\theta^{(t-1)}\right) = \mathbb{E}_{\mathbf{z}_{1:n}}\left[\log P\left(\mathbf{x}_{1:n},\mathbf{z}_{1:n}|\theta\right)|\mathbf{x}_{1:n},\theta^{(t-1)}\right] = \mathbb{E}_{\mathbf{z}_{1:n}}[\sum_{i=1}^k \log P(x_i,z_i|\theta)|x_{1:n},\theta^{(t-1)}] = \sum_{i=1}^n \mathbb{E}_{z_i}[\log P(x_i,z_i|\theta)|x_i,\theta^{(t-1)}] = \sum_{i=1}^n \sum_{j=1}^k P(z_i = j|x_i,\theta^{(t-1)}) \log P(x_i,z_i = j|\theta)$

And maximize $\theta^{(t)} = \arg\max_\theta Q\left(\theta;\theta^{(t-1)}\right)$ in the M-Step.