Summary for Introduction to Machine Learning 2019

Regression: Predict real valued labels

Linear Regression

$f(x) = w_1x_1 + \cdots + w_dx_d + w_0 = \widetilde{w}^T\widetilde{x}$ with
$\widetilde{w} = [w_1 \cdots w_d,\ w_0]$ and $\widetilde{x} = [x_1 \cdots x_d,\ 1]$
Residual: $r_i = y_i - w^Tx_i,\ x_i \in \mathbb{R}^d,\ y_i \in \mathbb{R}$
Cost / Objective function (is convex):
$\hat{R}(w) = \sum_{i=1}^n r_i^2 = \sum_{i=1}^n (y_i - w^Tx_i)^2$
Optimal weights:
$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^n (y_i - w^Tx_i)^2$
Closed form solution: $w^* = (X^TX)^{-1}X^Ty$
Gradient: $\nabla_w\hat{R}(w) = [\frac{\delta}{\delta w_1}\hat{R}(w) \cdots \frac{\delta}{\delta w_d}\hat{R}(w)] = -2\sum_{i=1}^n r_ix_i^T$
Non-linear functions: $f(x) = \sum_{i=1}^D w_i\phi_i(x)$

Convex function

$f : \mathbb{R}^d \to \mathbb{R}$ is convex $\Leftrightarrow x_1, x_2 \in \mathbb{R}^d, \lambda \in [0,1]$:
$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda)f(x_2)$

Gradient Descent

1. Start at an arbitrary $w_0 \in \mathbb{R}^d$
2. For $t = 1, 2, ...$ do $w_{t+1} = w_t - \eta_t\nabla\hat{R}(w_t)$

Gaussian/Normal Distribution

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

Multivariate Gaussian

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

$\Sigma = \begin{pmatrix}\sigma_1^2 & \sigma_{12}\\ \sigma_{21} & \sigma_2^2\end{pmatrix},\ \mu = \begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}$

Empirical risk minimization

Assumption: Data set generated iid from unknown distribution P: $(x_i, y_i) \sim P(X,Y)$.
True risk: $R(w) = \int P(x,y)(y - w^Tx)^2dxdy = \mathbb{E}_{x,y}[(y - w^Tx)^2]$
Empirical risk:
$\hat{R}_D(w) = \frac{1}{|D|}\sum_{(x,y)\in D}(y - w^Tx)^2$
Generalization error: $|R(w) - \hat{R}_D(w)|$
Uniform convergence:
$\underset{w}{\sup}\ |R(w) - \hat{R}_D(w)| \to 0$ as $|D| \to 0$
In general, it holds that:
$\mathbb{E}_D[\hat{R}_D(\hat{w}_D)] \le \mathbb{E}_D[R(\hat{w}_D)]$, where
$\hat{w}_D = \underset{w}{\operatorname{argmin}}\ \hat{R}_D(w)$.

Cross-validation

For each model m
For i = 1:k
1. Split data: $D = D_{train}^{(i)} \uplus D_{val}^{(i)}$
2. Train model: $\hat{w}_{i,m} = \underset{w}{\operatorname{argmin}}\ \hat{R}_{train}^{(i)}(w)$
3. Estimate error: $\hat{R}_m^{(i)} = \hat{R}_{val}^{(i)}(\hat{w}_{i,m})$

After all iterations, select model:
$\hat{m} = \underset{m}{\operatorname{argmin}}\ \frac{1}{k}\sum_{i=1}^k \hat{R}_m^{(i)}$

Ridge regression

Regularization(corresponds to MAP estimation):
$\underset{w}{\min}\ \frac{1}{n}\sum_{i=1}^n (y_i - w^Tx_i)^2 + \lambda||w||_2^2 = \underset{w}{\operatorname{argmax}}\ P(w)\Pi_iP(y_i|x_iw)$
Sparse regression (L1, convex) encourages coefficients to be exactly 0 - automatic feature selection
Closed form solution: $\hat{w} = (X^TX + \lambda I)^{-1}X^Ty$
Gradient: $\nabla_w(\frac{1}{n}\sum_{i=1}^n(y_i - w^Tx_i)^2 + \lambda||w||_2^2) = \nabla_w\hat{R}(w) + 2\lambda w$

Standardization

Goal: each feature: $\mu = 0,\ \sigma^2 = 1$:
$\tilde{x}_{i,j} = \frac{(x_{i,j} - \hat{\mu}_j)}{\hat{\sigma}_j}$
$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^n x_{i,j},\ \hat{\sigma}_j^2 = \frac{1}{n}\sum_{i=1}^n (x_{i,j} - \hat{\mu}_j)^2$

Dimension Reduction in unsupervised learning

Principal Component Analysis (linear)

Given $D \subseteq \mathbb{R}^d, 1 \le k \le d, \Sigma = \frac{1}{n}\sum_{i=1}^n x_ix_i^T, \mu = \frac{1}{n}\sum_i x_i = 0$ (data is centered)
$(W, z_1, ..., z_n) = \operatorname{argmin} \sum_{i=1}^n ||Wz_i - x_i||_2^2$
where $W \in \mathbb{R}^{d\times k}$ is orthogonal ,$z_1, ...z_n \in \mathbb{R}^k$ is given by $W = (v_1|...|v_k)$and$z_i = W^Tx_i = f(x)$
where $\Sigma = \Sigma_{i=1}^d\lambda_iv_iv_i^T$ where $\lambda_1 \ge ... \ge \lambda_d \ge 0$
The projection is chosen to minimize the reconstruction error, choose k such that most of the variance is explained (like k-means)

Kernel PCA (nonlinear)

For k = 1: Kernel PCA
$\alpha^* = \underset{\alpha^TK\alpha=1}{\operatorname{argmax}}\alpha^TK^TK\alpha$
With $K = \Sigma_{i=1}^n\lambda_iv_iv_i^T$ $(\lambda_1 \ge ... \ge \lambda_d \ge 0)$
$\alpha* = \frac{1}{\sqrt{\lambda_1}}v_1$
For general k: Kernel PCA
The kernel principal components are given by $\alpha^{(1)}, ..., \alpha^{(k)} \in \mathbb{R}^n$
$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}}$ with $K = \Sigma_{i=1}^n\lambda_iv_iv_i^T$
A new point x is projected as z,
$z_i = \Sigma_{j=1}^n\alpha_j^{(i)}k(x, x_j)$
Kernel-PCA corresponds to applying PCA in the feature space induced by the kernel k.
centering a kernel: $K' = K - KE - EK + EKE$
where $E = \frac{1}{n}[1, ..., 1][1, ...1]^T$
- complexity grow with number of data points, requires data specified as kernel

Autoencoders

Goal: learn identity function $x \approx f(x;\theta)$
$f(x;\theta) = f_{dec}(f_{enc}(x;\theta_1);\theta_2)$
NN autoencoders are ANNs where one output unit for each of d input units, nr of hidden units smaller than nr of inputs. Optimize w s.t. output agrees with input.
If activation func. is the identity, fitting NN autoencoder is equivalent to PCA.

Decision Theory

Bayesian Decision Theory

Given: $P(y|x)$, set of actions A and cost function $C : Y \times A \to \mathbb{R}$
$a^* = \underset{a\in A}{\operatorname{argmin}}\mathbb{E}_y[C(y,a)|x]$ (cost for prediction y when true label is a)
for logistic regression:$\underset{y}{\operatorname{argmax}}P(y|x) = sign(w^Tx)$