1 Evaluating models and algorithms

1. **Cross validation**: partition data into n subsamples; iteratively leave one subsample out for the test set and train on the remaining data
2. **Confusion matrices**:

	actual class +	actual class −
predicted class +	TP	FP
predicted class −	FN	TN

$accuracy = \frac{TP+TN}{TP+FP+FN+TN}$ $\quad error = \frac{FP+FN}{TP+FP+FN+TN}$

$TPR = \frac{TP}{TP+FN}$ $\quad FPR = \frac{FP}{TN+TP}$

3. **Common plots**:
   a) Learning curves: accuracy v. sample size
   b) Receiver operating characteristic (ROC) curve for binary classification task: TPR v. FPR as a threshold on the confidence of an instance being positive is varied; best result: Γ-shaped curve
   c) Precision recall (PR) curve for binary classification task:
      i. recall = TPR = $\frac{TP}{TP+FN}$ $\quad$ precision = positive predictive value = $\frac{TP}{TP+FP}$
      ii. P v. R is plotted as a threshold on the confidence of an instance being positive is varied
      iii. Best result: curve looks like Γ folded over vertical axis
4. **Confidence interval**: with approximately C% probability, the true error lies in the interval
$$error_s(h) + z_c\sqrt{\frac{error_s(h)(1-error_s(h))}{n}}$$
where z_C is a constant that depends on C, e.g. $z_c = 1.96$ for 95% confidence. (When $n \geq 30$ and p is not too extreme, the normal distribution is a good approximation to the binomial distribution.)
5. **Paired t testing**:
   a) Calculate the sample mean, $\bar{\delta} = \frac{1}{n}\sum_{i=1}^{n}\delta_i$
   b) Calculate the t-statistic, $t = \frac{\bar{\delta}}{\sqrt{\frac{1}{n(n-1)}\sum_{i=1}^{n}(\delta_i-\bar{\delta})^2}}$
   c) Use look-up table to relate t-statistic to p-value; reject null hypothesis (H_o) if $p \leq 0.05$
   d) Note: one-tailed t-test evaluates whether system A is better than system B; two-tailed t-test evaluates whether the accuracy of the two systems are different

2 Decision trees

1. **Entropy**: $\mathbb{H}(X) \triangleq -\sum_{k=1}^{K} p(X=k)\log_2 p(X=k)$
2. **Cross entropy**: $\mathbb{H}(p,q) \triangleq -\sum_k p_k \log q_k$
3. **Conditional entropy**: $\mathbb{H}(Y|X) = \sum_{x\in X} P(X=x)\mathbb{H}(Y|X=x)$
where $\mathbb{H}(Y|X=x) = -\sum_{y\in Y} P(Y=y|X=x)\log_2 P(Y=y|X=x)$
4. **Information gain**: $\mathbb{I}(X,Y) = \mathbb{H}(X) - \mathbb{H}(X|Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X)$
5. **Gain ratio**: $GainRatio(D,S) = \frac{InfoGain(D,S)}{\mathbb{H}_D(s)} = \frac{\mathbb{H}_D(y)-\mathbb{H}_D(Y|S)}{\mathbb{H}_D(s)}$
6. **Pseudo-code learning procedure**: MakeSubTree, DetermineCandidateSplits, StoppingCriteriaMet, FindBestSplit, MakeSubTree
7. **Regression trees**: leaves have functions that predict numeric values; CART algorithm minimizes squared error for determining splits
8. **Implementation details**:
   a) For numeric features, can use midpoint of each considered interval as the threshold
   b) Methods to avoid overfitting: early stopping, post-pruning (with validation data set)

3 Instance-based methods, e.g. k-Nearest Neighbor (k-NN)

1. **k-NN classification**: approximate a discrete-valued function f: $\mathbb{R}^n \rightarrow V$, given a query instance x_q to be classified
   a) Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
   b) Return: $\hat{f}(x_q) \leftarrow \underset{v\in V}{argmax} \sum_{i=1}^{k}\delta(v, f(x_i))$ where $\delta(a,b) = 1$ if $a = b$ and $\delta(a,b) = 0$ otherwise
   c) For distanced-weighted k-NN, return: $\hat{f}(x_q) \leftarrow \underset{v\in V}{argmax} \sum_{i=1}^{k} w_i\delta(v, f(x_i))$
   where $w_i \equiv \frac{1}{d(x_q,x_i)^2}$
2. **k-NN regression**: approximate a continuous-valued function f: $\mathbb{R}^n \rightarrow \mathbb{R}$, given a query instance x_q to be classified
   a) Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
   b) Return: $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$
   c) For distance-weighted k-NN, return: $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{w_i}$
3. **Distance metrics for continuous features**:
   a) Euclidean distance: $d(x^{(i)}, x^{(j)}) = \sqrt{\sum_f (x_f^{(i)} + x_f^{(j)})^2}$
   b) Manhattan distance: $d(x^{(i)}, x^{(j)}) = \sum_f |x_f^{(i)} + x_f^{(j)}|$
4. **Computational speed improvements**:
   a) Edited-NN: don't retain every training instance; use *incremental delection* or *incremental growth*
   b) k-d tree: use a data structure to look up nearest neighbors
5. **Locally weighted regression**: Approximate the target function f near x_q using a linear function of the form $\hat{f}(x) = w_o + w_1 a_1(x) + \ldots + w_n a_n(x)$
   a) Optimal loss function: $E_3(x_q) \equiv 1/2 \sum_{x\in\, k-NNs\ of\ x_q} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$
   b) Corresponding gradient descent training rule:
   $\Delta w_j = \eta \sum_{x\in\, k-NNs\ of\ x_q} K(d(x_q, x))(f(x) - \hat{f}(x))a_j(x)$

4 Bayesian networks

1. **Model expression**: directed acyclic graph (DAG)
2. **Joint probability distribution via chain rule**: $P(X_1, \ldots, X_N) = P(X_1)\prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1})$
3. **Compact, Bayesian representation a joint probability distrubtion**: $P(X_1, \ldots, X_N) = P(X_1)\prod_{i=1}^{n} P(X_i|Parents(X_1))$
4. **Spare Candidate algorithm** (Friedman, 1999):
   a) Purpose: unsupervised method to learn the structure of a Bayes network
   b) Overview: given data set D, initial network B_0, and parameter k; *restrict* step; *maximize* step; ... repeat until convergence
   c) Mutual information: $I(X,Y) = \sum_{x\in values(X)}\sum_{y\in values(Y)} P(x,y)\log_2\frac{P(x,y)}{P(x)P(y)}$
   d) Distance between distr. P and Q is evaluated using Kullback-Leibler (KL) divergence: $D_{KL}(P(X)\|Q(x)) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$
5. **Practical consideration: missing data**:
   a) Given: data with some missing values; model structure with initial model parameters
   b) Repeat expectation maximization (EM) method until convergence
      i. Expectation (E) step: using current model, compute expectation over missing values
      ii. Maximization (M): update model parameters with those that maximize probability of the data, i.e. use MLE or MAP
6. **Chow-Liu algorithm**: learns a Bayes network with a tree structure that maximizes the likelihood of the training data
   a) Compute weight $I(X_i, X_j)$ of each possible edge (X_i, X_j)
   b) Find maximum weight spanning tree (MSE); use Prim or Kruskal algorithms, for example
   c) Assign edge directions in MST

5 Naive Bayes

1. **Bayes rule**: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
2. **Conditional independence**: X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y, given the value of Z, e.g. $P(X|Y,Z) = P(X|Z)$
3. **Naive Bayes assumption**: $P(X_1 \ldots X_n|Y) = \prod_i P(X_i|Y)$
4. **Naive Bayes classifier** (Mitchell, 1997): $v_{NB} = P(v_j)\prod_i P(a_i|v_j)$ $\quad v_j \in V$
5. **Naive Bayes algorithm for discrete-valued feature labels**:
   a) Training: (1) for each value y_k, estimate $\pi_k \equiv P(Y = y_k)$, (2) for each value x_{ij} of each attribute X_i, estimate $\theta_{ijk} \equiv P(X_i = x_{ij}|Y = y_k)$
   b) Classification (X^{new}):
   $$Y^{new} \leftarrow \underset{y_k}{argmax}\ P(Y=y_k)\prod_i P(X_i^{new}|Y=y_k)$$
   $$\implies Y^{new} \leftarrow \underset{y_k}{argmax}\ \pi_k \prod_i \theta_{ijk}$$
6. **Naive Bayes algorithm for discrete-valued feature labels**:
   a) Training: (1) for each value y_k, estimate $\pi_k \equiv P(Y = y_k)$, (2) for each attribute X_i, estimate class conditional mean μ_{ik} and variance σ_{ik}
   b) Classification (X^{new}):
   $$Y^{new} \leftarrow \underset{y_k}{argmax}\ P(Y=y_k)\prod_i P(X_i^{new}|Y=y_k)$$
   $$\implies Y^{new} \leftarrow \underset{y_k}{argmax}\ \pi_k \prod_i Normal(X_i^{new}, \mu_{ik}, \sigma_{ik})$$
7. **Tree-augmented Naive Bayes (TAN)**:
   a) Implementation algorithm:
      i. Compute weight $I(X_i, X_j|Y)$ for each possible edge (X_i, X_j) between features
      ii. Find maximum weight spanning tree (MST) for graph over $X_1 \ldots X_n$; use Prim's algorithm, for example
      iii. Assign edge directions in MST
      iv. Construct a TAN model by adding node for Y and an edge from Y to each X_i
   b) Conditional mutual information:
   $$I(X_i, X_j|Y) = \sum_{x_i\in val(X_i)}\sum_{x_j\in val(X_j)}\sum_{y\in val(y)} P(x_i, x_j, y)\log_2\frac{P(x_i,x_j|y)}{P(x_i|y)P(x_j|y)}$$
   c) Laplace estimates ("add-1 smoothing"):
      i. For homework assignment 4: $P_{LAP,K}(x_i|y) = \frac{n_{rows}(X_i=x_i, Y=y)+K}{n_{rows}(Y=y)+K|X_i|}$ where K=1 for homework assignment 4, and $|X_i|$ is the number of distinct values for feature X_i.
      ii. Extension 1: $P_{LAP}(x_i, x_j|y) = \frac{n_{rows}(X_i=x_i, X_j=x_j, Y=y)+1}{n_{rows}(Y=y)+|X_i||X_j|}$
      iii. Extension 2: $P_{LAP}(x_i, x_j, y) = \frac{n_{rows}(X_i=x_i, X_j=x_j, Y=y)+1}{n_{rows,total}+|X_i||X_j||Y|}$

6 Linear regression

1. **Problem statement**:
   a) Given training data $(x^{(i)}, y^{(i)}) : 1 \leq i \leq m$ i.i.d. from distribution D, find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{m}\sum_{i=1}^{m}(w^T x^{(i)} - y^{(i)})^2$
   b) Setting the gradient of $\hat{L}(f_w) = \frac{1}{m}\sum_{i=1}^{m}(w^T x^{(i)} - y^{(i)})^2 = \frac{1}{m}|Xw - y|_2^2$ yields the ordinary least squares solution: $w = (X^T X)^{-1} X^T y$
2. **Special cases**:
   a) Linear regression with bias: find $f_{w,b}(x) = w^T x + b = (w')^T(x')$ to minimize the loss
   b) Linear regression with lasso penalty: find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{m}\sum_{i=1}^{m}(w^T x^{(i)} - y^{(i)})^2 + \lambda|w|_1$
3. **Evaluation metrics**: root mean squared error (RMSE), mean absolute error (MAE; average l_1 error), R^2
   a) R^2: Evaluate and square $r = \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\frac{1}{N}(x_i-\bar{x})^2}\sqrt{\frac{1}{N}(y_i-\bar{y})^2}}$

7 Logistic regression

1. **MLE with sigmoid**: $\hat{L}(w) = -\frac{1}{m}\sum_{y^{(i)}=1}\log\sigma(w^T x^{(i)}) - \frac{1}{m}\sum_{y^{(i)}=0}\log[1-\sigma(w^T x^{(i)})]$
has no closed-form solution; must use gradient descent
2. **Properties of the sigmoid function**
   a) Bounded: $\sigma(a) = \frac{1}{1+exp(-a)} \in (0,1)$
   b) Symmetric: $1 - \sigma(a) = \frac{exp(-a)}{1+exp(-a)} = \frac{1}{1+exp(a)} = \sigma(-a)$
   c) Gradient: $\sigma'(a) = \frac{exp(-a)}{(1+exp(-a))^2} = \sigma(a)(1-\sigma(a))$
3. **Hypothesis class for multiclass logistic regression**:
$p(y=i|x) = \frac{exp((w^i)^T x + b^i)}{\sum_j exp((w^J)^T x + b^J)}$

8 Discriminative v. generative learning

1. **Discriminative approach**
   a) Hypothesis $h \in H$ directly predicts the label given the features $y = h(x)$, i.e. $p(y|x) = h(x)$
   b) Then, define a loss function $L(h)$ and find the hypothesis with minimum loss
   c) In other words, learn $P(Y|X_1, \ldots, X_n)$ directly
   d) Example: linear regression $\rightarrow h_\theta(x) =< x, \theta >$ and $L(h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
   e) More examples: logistic regression, SVM, some neural network topologies
2. **Generative approach**
   a) Hypothesis $h \in H$ specify a generative story for how the data was created, i.e. $h(x,y) = p(x,y)$
   b) Then, pick a hypothesis by maximum likelihood estimation (MLE) or maximum a posteriori (MAP) estimation
   c) In other words, estimate $P(Y)$ and $P(X1, \ldots X_n|Y)$ for learning, and use Bayes' rule to compute $P(Y|X_1, \ldots, X_n)$ for classification
   d) Examples: Naive Bayes, Bayesian networks
3. **Naive Bayes vs. logistic regression**
   a) Logistic regression (discr.): $f(x) = \frac{1}{1+exp(-(w_0+\sum_{i=1}^{n} w_i x_i))}$
   b) Naive Bayes (gen.): $P(Y=1|x_1, \ldots x_n) = \frac{1}{1+exp(-\ln\frac{P(Y=1)}{P(Y=0)} - \sum_{i=1}^{n}\frac{P(x_i|Y=1)}{P(x_i|Y=0)})}$
   c) Generally, Naive Bayes has lower/higher predictive error than logistic regression when training sets are small/large; parameters converge to their asymptotic values more quickly in generative classifiers
4. **Mathematical formulas**
   a) $\theta^{MLE} = \underset{\theta}{argmax}\prod_{i=1}^{N} p(x^{(i)}|\theta)$
   e.g. $\hat{\theta}^{MLE} = \frac{\alpha_1}{\alpha_1+\alpha_0}$ where α_1 is the number of flipped heads and α_0 is the number of flipped zeros
   b) $\theta^{MAP} = \underset{\theta}{argmax}\prod_{i=1}^{N} p(x^{(i)}|\theta)p(\theta)\ldots$ this includes a prior probability!
   e.g. $\hat{\theta}^{MAP} = \frac{\alpha_1+\beta_1}{(\alpha_1+\beta_1)+(\alpha_0+\beta_0)}$ where β_1 is a hallucinated number of flipped heads and β_0 is a hallucinated number of flipped tails

9 Neural networks

1. **Activation functions**:
   a) Sigmoid: $\sigma(x) = \frac{1}{1+exp(x)} \implies \sigma'(x) = \sigma(x)[1-\sigma(x)]$
   b) Hyperbolic tangent: $\sigma(x) = \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)} \implies \sigma'(x) = 1 - \sigma(x)^2$
2. **Cost functions**:
   a) For regression, with K outputs, the negative log-likelihood is given by the squared error: $J(\theta) = -\sum_n\sum_k(\hat{y}_{nk}(\theta) - y_{nk})^2$
   b) For classification, with K classes, the negative log-likelihood is given by the cross entropy: $J(\theta) = -\sum_n\sum_k y_{nk}\log\hat{y}_{nk}(\theta)$
3. **Perceptron algorithm**

a) If including an input of 1 (and w_0 weight), $y = sign(\sum_{i=0}^{h} w_i x_i) \iff y = 1$ for $s > 0$ and $y = 0$ for $s < 0$
b) Example - AND function, i.e. $(X_1 \bigwedge X_2)$: $y = sign(X_1 + X_2 - 1.5)$

4. **Training procedure for one-layer network**
a) Initialize $w(0)$ with random values
b) For each j, calculate $\hat{y}^{(j)} = sign(x^{(j)T} w(t))$
c) Update: $w(t+1) = w(t) + \eta[y^{(j)} - \hat{y}^{(j)}]x^{(j)}$ where η is the learning rate
5. **Derivation of the general training procedure for multi-layer network, including back propagation**
a) Back propagation description: chain rule applied to the cost function
b) Stochastic gradient descent (SGD) method: $w_i(t+1) = w_i(t) - \eta \frac{\partial J(w)}{\partial w_i}$ where $J(w)$ is a cost function
c) Since, typically, $J(w) = \frac{1}{2}[\hat{y}^{(t)} - y^{(t)}]^2$, and $\hat{y}^{(t)} = \sigma(\sum_{i=1}^{k} w_j x_j)$
$w_i(t+1) = w_i(t) - \eta[\hat{y}^{(t)} - y^{(t)}]\sigma'(\sum_{i=1}^{k} w_j x_j)x_i \implies w_i(t+1) = w_i(t) - \eta[\hat{y} - y](1 - \hat{y}^2)x_i = w_i(t) - \eta\delta x_i$
d) The appearance of δ indicates the "δ rule"
6. **Expanded training procedure for multi-layer networks**
a) Consider a multi-layer network with input states x_i, second layer states p_j, third layer states q_k, and output layer states y_l
i. $p_j = \sigma(\sum_i \gamma_{ji} x_i)$; $\gamma \in \mathbb{R}^{Nxn}$
ii. $q_k = \sigma(\sum_j \beta_{kj} p_j)$; $\beta \in \mathbb{R}^{mxN}$
iii. $y_l = \sigma(\sum_k \alpha_{lk} q_k)$; $\alpha \in \mathbb{R}^{mxM}$
b) Update α: $\alpha(t+1) = \alpha(t) - \eta\delta_l \hat{q}^T$ where $\delta_l = (\hat{y} - y) .* (1 - \hat{y}.^2)$ (MATLAB notation)
c) Update β: $\beta(t+1) = \beta(t) - \eta\delta_k \hat{p}^T$ where $\delta_k = \delta_l \alpha_{lk}(1 - \hat{q}_k^2)p_j$
d) Update γ: ...

10 Computational learning theory, e.g. probably approximately correct (PAC) models

1. **Motivation**:
a) Can we infer something about generalization error from training error?
b) Can we infer how many training instances are enough?
c) Do not require that $error_D(h) = 0$; instead, bind the error of a learned hypothesis by some constant $\epsilon \implies$ the probability of the learner failing to learn an accurate hypothesis is bounded by a constant δ
2. **True error** ($error_D(h)$): defined as the probability that h will misclassify an instance drawn at random according to D;
$error_D(h) \equiv \Pr_{x \in D}[c(x) \neq h(x)]$
3. **PAC learnability**: Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is *PAC learnable* by L using H if for all $c \in C$, distributions D over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $error_D(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n, and $size(c)$.
4. **Sample complexity for finite hypothesis space**: Consider a hypothesis space H, target concept c, instance distribution D, and set of training examples D of c. The version space $VS_{H,D}$ is said to be ϵ-exhausted with respect to c and D, if every hypothesis h in $VS_{H,D}$ has error less than ϵ with respect to c and D; $(\forall h \in VS_{H,D}) error_D(h) < \epsilon$
5. **Vapnik-Chervonenkis (VC) dimension**: $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$. This metric essentially measures the complexity of hypothesis space H.
6. **Upper bounds on number of training examples sufficient for PAC learning**
a) PAC learning model: $m \geq \frac{1}{\epsilon}(\ln(1/\delta) + \ln(H))$
b) Agnostic learning model: $m \geq \frac{1}{2\epsilon^2}(\ln(1/\delta) + \ln(H))$
c) Upper bound in terms of the VC dimension: $m \geq \frac{1}{\epsilon}(4\log_2(2/\delta) + 8VC(H)\log_2(13/\epsilon))$
d) Lower bound in terms of VC dimension: $m \geq max[\frac{1}{\epsilon}\log(1/\delta), \frac{VC(C)-1}{32\epsilon}]$
7. **Mistake bound model**: use to analyze the number of training examples a learner will misclassify before it exactly learns the target concept. For arbitrary concept class C, the best worst-case algorithm will make $Opt(C)$ mistakes, where $VC(C) \leq Opt(C) \leq \log_2(|C|)$
8. **Weighted majority algorithm**: combines the weighted votes of multiple prediction algorithms to classify new instances; the weights are learned for each prediction algorithm based on errors made over a sequence of examples

11 Kernal methods and the support vector machine (SVM)

1. **Kernel methods**:
a) Design requirements: it is not required to design $\phi(\cdot)$; only need to design $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$
b) Common types of kernels:
i. Polynomial with fixed degree d and constant c: $k(x, x') = (x^T x' + c)^d$
$\implies K(x, x') = (x_1 x'_1 + x_2 x'_2 + c)^2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \\ \sqrt{2}cx_1 \\ \sqrt{2}cx_2 \\ c \end{bmatrix} \begin{bmatrix} x'^2_1 \\ x'^2_2 \\ \sqrt{2}x'_1 x'_2 \\ \sqrt{2}cx'_1 \\ \sqrt{2}cx'_2 \\ c \end{bmatrix}$
ii. Radial basis function (RBF)/Gaussian with fixed bandiwidth σ: $k(x, x') = exp(-\|x - x'\|^2/2\sigma^2)$
A. Non-normalized version: $k'(x, x') = exp(x^T x'/\sigma^2)$
B. Power series expansion: $k'(x, x') = \sum_i^{+\infty} \frac{(x^T x')^i}{\sigma^i i!}$
c) Kernel trick: implicitly use high-dimensional mappings without explicitly computing them
d) Kernel algebra allows the construction of new kernels, e.g. $k(x, x') = 2k_1(x, x') + 3k_2(x, x')$
e) Can integrate kernel functions into neural networks
2. **Mathematical, optimization background for the SVM**:
a) Simplified optimization objective: $\min_{w,b} \frac{1}{2}\|w\|^2$ and $y_i(w^T x_i + b) \geq 1, \forall i$
b) Lagrange multiplier method
i. Consider the optimization problem: $\min_w f(w)$, where $g_i(w) \leq 0, \forall 1 \leq i \leq k$, and $h_j(w) = 0, \forall 1 \leq j \leq l$
ii. The generalized Lagrangian is: $\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_i \alpha_i g_i(w) + \sum_j \beta_j h_j(w)$ where the α_i and β_i are Lagrangian multipliers
iii. Lagrange duality - the primal problem: $p^* := \min_w f(w) = \min_w \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta)$
iv. Lagrange duality - the dual problem: $d^* := \max_{\alpha,\beta:\alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$
v. It is always true that $d^* \leq p^*$; under Karush-Kuhn-Tucker (KKT) conditions, $d^* = p^*$
c) Using quadratic programming and enforcing KKT conditions reduces the optimization to the dual problem:
$\mathcal{L}(w, \beta, \alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j \implies \sum_i \alpha_i y_i = 0$ and $\alpha_i \geq 0$
d) Since $w = \sum_i \alpha_i y_i x_i$, the result is $w^T x + b = \sum_i \alpha_i y_i x_i^T x + b$ where $x_i^T x$ terms only depend on inner products
e) The final solution is a sparse linear combination of the training instances; instances with $\alpha_i > 0$ are called *support vectors*
f) Minimizing the VC dimension $\rightarrow$ improves the error bound, maximizes the margin
3. **Variants of the classic SVM**: hard/soft-margin SVM, support vector regression (SVR)

12 Ensemble methods

1. **Definition of ensemble**: a set of learned models whose individual decisions are combined in some way to make predictions for new instances; recommended for almost any practical learning problem
2. **Ensemble approaches**: bagging, random forests, boosting, skewing, stacking, error correcting output codes
a) Bagging: choose different subsamples of the training set; works best with unstable learning methods, i.e. those that tend to overfit training data
i. Learning: $D^{(i)} \leftarrow m$ instances randomly drawn from D with replacement; $h_i \leftarrow$ model learned using L on $D^{(i)}$
ii. Classification: given test instance x, predict y via $PluralityVote(h_1(x)...h_T(x))$
iii. Regression: given test instance x_i, predict y as $mean(h_1(x)...h_T(x))$
b) Boosting, skewing: use different probability distributions over the training instances, e.g. *Adaboost* from (Freund and Schapire, 1997)
i. *Adaboost* procedure, given learner L, stages T, and training set D: initialize instance weights, normalize weights, calculate weighted error, lower error $\rightarrow$ smaller β_t, downweigh correct examples, and finally return $h(x) = \underset{y}{argmax} \sum_{t=1}^{T} (\log \frac{1}{\beta_t})\delta(h_t(x), y)$
c) Random forests: choose different features and subsamples

13 Feature selection

1. **List of methods**: filtering-based, e.g. information gain and Markov blanket filtering, frequency pruning, wrapper-based, forward selection, backward elimination, L_1 and L_2 penalties, lasso and ridge regression, dimensionality reduction
2. **Markov blanket approach**: a Markov blanket M_i for a variable X_i is a set of variables such that all other variables are conditionally independent of X_i given M_i; try to find and remove features that minimize a criterion, e.g. if Y is conditionally independent on feature X_i given a subset of other features, we should be able to omit X_i
3. **Frequency pruning**: remove features whose distributions are highly skewed, e.g. remove very high and low frequency words in text document spam filtering
4. **Wrapper-based feature selection**: evaluate each feature set by using the learning method to score it, i.e. "how accurate of a model can be learned with it?"
5. **Forward selection**: uses hill climbing search to score feature set G by learning model(s) with learning method L and assessing model accuracy
6. **Shrinking/regularization**: bias the learning process toward a small number of features
a) Example: linear regression
i. Lasso regression adds the L_1 norm of the weights as a penalty term: $E(w) = \sum_{d \in D}(y^{(d)} - w_0 - \sum_{i=1}^{n} x_i^{(d)} w_i)^2 + \lambda \sum_{i=1}^{n} |w_i|$
ii. Ridge regression adds the L_2 norm of the weights as a penalty term: $E(w) = \sum_{d \in D}(y^{(d)} - w_0 - \sum_{i=1}^{n} x_i^{(d)} w_i)^2 + \lambda \sum_{i=1}^{n} w_i^2$
iii. Other related regression methods: Elastic net, group lasso, fused lasso

14 Dimensionality reduction

1. **Introduction**: Powerful, unsupervised learning techniques, for extracting lower dimensional structure from high dimensional datasets, e.g. PCA, kernel PCA, and ICA, are useful for visualization, resource utilization increases, better generalization properties, noise removal, and further processing by machine learning algorithms
2. **Principal component analysis (PCA)**: an orthogonal projection or transformation of the data into a potentially lower dimensional subspace so that the variance of the projected data is maximized; extracts variance structure from high dimensional data sets
a) The first principal component (PC) represents the direction of greatest variability in the data; the second (orthogonal; uncorrelated) PC represents the direction of second greatest variability
b) Critical computations: eigen-decomposition, singular value decomposition (SVD)
c) Find a vector that maximizes the sample variance of projected data: $\frac{1}{n}\sum_{i=1}^{n}(v_T x_i)^2 = v^T X X^T v \implies ... \implies (XX^T)v = \lambda v$
i. The first PC v is the eigenvector of sample correlation/covariance matrix XX^T
ii. The eigenvalue λ denotes the amount of variability captured along the PC direction; in general $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq ...$
d) Interpretations:
i. Maximum variance direction: the first PC is a vector v such that projection onto this vector captures maximum variance in the data considering all of the possible one-dimensional projections
ii. Minimum reconstruction error: the first PC is a vector v such that projection onto this vector yields minimum mean squared error (MSE) reconstruction; can be described with Pythagorean theorem
e) Practical usage: only keep data projections onto PCs with large eigenvalues; components of smaller significance can be neglected
f) Applications: k-means clustering
g) Limitations: second-order statistical model, linear projections only

15 Reinforcement learning

1. **Description of the procedure**: (1) sense world, (2) reason, (3) choose an action to perform, (4) get feedback (usually reward = 0), (5) learn
2. **Pseudo-procedure**: given a set of states S and actions A - at each time t, the agent observes state $s_t \in S$, then chooses action $a_t \in A$; then, the agent receives a reward r_t and indexes to state s_{t+1}
3. **Value function for a policy**:
a) Given a policy $\pi : S \rightarrow A$, define $V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t E[r_t]$; an optimal policy π^* is desired where $\pi^* = \underset{\pi}{argmax}\, V^\pi(s)$ for all s
b) Key assumption: value iteration for learning $V^\pi(s)$ assumes a model of the world, i.e. $P(s_t|s_{t-1}, a_{t-1})$, is possessed
4. Q **learning introduction**:
a) Define a new function, closely related to V^*, i.e.: define $Q(s, a) \leftarrow E[r(s, a)] + \gamma E_{s'|s,a}[V^*(s')]$ when $V^*(s) \leftarrow E[r(s, \pi^*(s))] + \gamma E_{s'|s,\pi^*(s)}[V^*(s')]$
b) If the agent knows $Q(s, a)$, it can choose optimal action without knowing $P(s'|s, a)$, i.e.: $\pi^*(s) \leftarrow \underset{a}{argmax}\, Q(s, a)$ and $V^*(s) \leftarrow \underset{a}{argmax}\, Q(s, a)$
5. Q **learning for deterministic worlds**:
a) for each s, a, inialize table entry $\hat{Q}(s, a) \leftarrow 0$
b) observe current state s
c) do forever: select an action a and execute it; receive immediate reward r; observe the new state s'; update table entry using $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}'(s', a')$; $s \leftarrow s'$