F14-10601, Machine Learning
Midterm Review Sheet

Dawei Wang
daweiwan@andrew.cmu.edu

October 27, 2014

1 introduction

- supervised, unsupervised, active learning, reasoning under uncertainty.
  - supervised: given a set of features and labels, learn a model, predict;
  - unsupervised: discover patterns in data without labels;
  - reasoning: determine a model either from samples or as going along;
  - active: select not only model but also which examples to use.
- probability: random variable, domain, kolmogorov's axioms (foundation):
  - $0 \leq P(A) \leq 1$;
  - $P(\text{true}) = 1$, $P(\text{false})=0$;
  - $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.

  priors, conditional probability, joint distributions, chain rule. bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_A P(B|A)P(A)} \tag{1}$$

  density estimator, and hence the maximum likelihood principle:

$$\hat{P}(\text{Dataset}|M) = \hat{P}(x_1 \wedge x_2 \wedge \cdots \wedge x_n|M) = \prod_{k=1}^{n} \hat{P}(x_k|M) \tag{2}$$

  the logarithmic maximum likelihood principle:

$$\log \hat{P}(\text{Dataset}|M) = \log \prod_{k=1}^{n} \hat{P}(x_k|M) = \sum_{k=1}^{n} \log \hat{P}(x_k|M) \tag{3}$$

  take ∂ to find the best estimate for binary and guassian distros.

$$\arg\max_q q^{n_1}(1-q)^{n_2} \rightarrow q = \frac{n_1}{n_1 + n_2} \tag{4}$$

  for gaussian, that's the conventional deviation and variance.

2 classification

- bayes decision rule: determine class label using the bayes rule:

$$q_i(x) \equiv P(y = i|x) = \frac{P(x|y = i)P(y = i)}{P(x)} \tag{5}$$

  bayes error (risk) with two classes,

$$R(x) = \min\{P_1(x)P(y = 1), P_0(x)P(y = 0)\} \tag{6}$$

- classifiers: instance based (k nearest neighbors), generative (bayesian networks), discriminative (decision tree), latter two are major types.

- k nearest neighbors (knn): choice of k influences the *smoothness*, but this is determined by the actual distribution, not a pre-defined parameter. :(knn as *approximated* bayes decision rule.

- a probabilistic intrepretation: let V to be volume of the m dimensional ball around z containing the k nearest neighbors for z, then we have

$$p(x)V = P = \frac{K}{N}, \quad p(x) = \frac{K}{NV}, \quad p(x|y = 1) = \frac{K_1}{N_1 V}, \quad p(y = 1) = \frac{N_1}{N} \tag{7}$$

  and using the bayes rule we get $p(y = 1|x) = \frac{K_1}{K}$.

3 decision trees

- discriminative! nodes: attributes, leaves: labels, edges: assignments. split/create subtrees by testing on some attributes.

- determine the best attribute: *entropy*, the definition:

$$H(X) = \sum_c -p(X = c) \log_2 p(X = c) \tag{8}$$

  interpretation: with distribution known, minimum bits required to transmit a value. e.g. $P(X = 1) = 1 \to H = 0$, $P(X = 1) = 0.5 \to H = 1$. *expected bits per symbol.* – its motivation and origin...

- conditional entropy: definition:

$$H(X|Y) = \sum_i P(Y = i)H(X|Y = i) \tag{9}$$

  choose the attribute that produces the most entropy reduction, or the most *information gain*, as defined by

$$IG(X|Y)^* = H(X) - H(X|Y) \tag{10}$$

  which is always greater than 0, by Jensen's inequality.

- to avoid overfitting: tree pruning:
  - split data into train and test set, then build a tree using training set;
  - for all nodes, remove subtree, assign outcome to be the most common, check testing error and decide whether to keep change or restore.
- for continuous values: threshold, recursively.

4 naive-bayes classifier

- generative! naive-bayes assumes that the attributes are conditionally independent given the class label, or mathematically,

$$P(X|y) = \prod_j p_j(x^j|y) \tag{11}$$

and the full classification rule becomes,

$$\begin{aligned} \hat{y} &= \arg\max_v p(y=v|X) = \arg\max_v \frac{p(X|y=v)p(y=v)}{p(X)} \\ &= \arg\max_v \prod_j p_j(x^j|y=v)p(y=v) \end{aligned} \tag{12}$$

and finally the conditional likelihood,

$$L(X_i|y_i = v, \Theta_v) = \prod_j p(x_i^j|y_i = v, \theta_v^j) \tag{13}$$

- for continuous values: covariance matrix $\rightarrow$ diagonal matrix:

$$P(X|y=v) = \prod_j \frac{1}{\sqrt{2\pi}\sigma_v^j} \exp\left[-\frac{(x^j - \mu_v^j)^2}{2(\sigma_v^j)^2}\right] \tag{14}$$

separate means and variances for each class, the estimate would be:

$$\mu_v^j = \frac{1}{k_v} \sum_{X_i:y_i=v} x_i^j, \quad (\sigma_v^j)^2 = \sum_{X_i:y_i=v} \left(x_i^j - \mu_v^j\right)^2 \tag{15}$$

where $(\cdot)_i$ is the instance index, $(\cdot)^j$ feature index. $(\cdot)_v$ class label.

- problems: naive! assumption often violated. also: go through the article classification example in the homework or face annihilation.

5 regression

- linear regression with least squares:

$$w = \arg\min_w \sum_i (y_i - wx_i)^2 \quad \text{where} \quad y = wx + \epsilon \tag{16}$$

  ϵ as noise, y as label, x_i as features, w as weights. taking the derivative

$$\frac{\partial}{\partial w} \sum_i (y_i - wx_i)^2 = 0 \quad \rightarrow \quad w = \frac{\sum_i x_i y_i}{\sum_i x_i^2} \tag{17}$$

  gives the best estimate for w.

- introducing a bias term: $y = w_0 + w_1 x + \epsilon$, then

$$w_0 = \frac{\sum_i (y_i - w_1 x_i)}{n}, \quad w_1 = \frac{\sum_i x_i (y_i - w_0)}{\sum_i x_i^2} \tag{18}$$

- multivariate regression + non-linear basis function, with w, ϕ vectors,

$$y = w_0\phi_0(x) + w_1\phi_1(x) + \cdots + w_k\phi_k(x) = \sum_{j=0}^{n} w_j \phi_j(x) = w^T \phi(x) \tag{19}$$

  and let's minimize the loss-function, with w, ϕ vectors,

$$J(w) = \sum_i \left(y^i - w^T \phi(x^i)\right)^2 \tag{20}$$

$$\frac{\partial}{\partial w} J(w) = 2 \sum_i (y^i - w^T \phi(x^i)) \phi(x^i)^T \Rightarrow 0 \tag{21}$$

  we get, by pesudo-inverse, with j as feature index, and i instance index:

$$w = (\Phi^T \Phi)^{-1} \Phi^T y \quad \text{where} \quad \Phi = \phi_j(x^i) \tag{22}$$

- probabilistic interpretation: mle instead of least squares.

- locally weighted models, use weight function Ω_x and solve

$$\min_w \sum_i \Omega_x(x^i)(y^i - w^T \phi(x^i))^2 \tag{23}$$

  where the weights can be determined, e.g., by gaussian function.

6 logistic-regression

- *discriminative* vs. *generative*.
- regression for classification: sigmoid function as activation function.
- determine parameters using maximum likelihood estimation.

$$L\left(y|X;w\right)=\prod_i\left(\frac{1}{1+\exp w^TX}\right)^{1-y_i}\left(\frac{\exp w^TX}{1+\exp w^TX}\right)^{y_i} \tag{24}$$

$$LL\left(y|X;w\right)=\sum_i\left[y_iw^TX_i-\ln\left(1+\exp w^TX_i\right)\right] \tag{25}$$

$$\frac{\partial}{\partial w^j}LL\left(y|X;w\right)=\sum_i X_i^j\left[y_i-p\left(y^i=1|X_i;w\right)\right] \tag{26}$$

  bad news: *no close form solution*. good news: *concave function*.

- gradient ascent: $z=x(y-g(w;x))$ and its update rule (small ε):

$$w^j\leftarrow w^j+\varepsilon\sum_i X_i^j\left[y_i-(1-g(X_i;w))\right] \tag{27}$$

  until likelihood has no further improvement.

- regularization: additional constraints over w^j, e.g.: gaussian.
  in which case we have a *prior*: $p(y=1,\theta|X)\propto p(y=1|X;\theta)p(\theta)$.

$$LL'\left(y|X;w\right)=\sum_i\left[y_iw^TX_i-\ln\left(1+\exp w^TX_i\right)\right]-\sum_j\frac{(w^j)^2}{2\sigma^2} \tag{28}$$

  and the update rule becomes

$$w^j\leftarrow w^j+\varepsilon\sum_i X_i^j\left[y_i-(1-g(X_i;w))\right]-\varepsilon\frac{w^j}{\sigma^2} \tag{29}$$

- logistic regression for more than two classes: for $i<k$

$$p(y=i|X;\theta)=\frac{\exp z_i}{1+\sum_{j=1}^{k-1}\exp z_j} \tag{30}$$

  and for $i=k$,

$$p(y=i|X;\theta)=1-\sum_{i=1}^{k-1}p(y=i|X;\theta)=\frac{1}{1+\sum_{j=1}^{k-1}\exp z_j} \tag{31}$$

  and the derivative becomes

$$:\frac{\partial}{\partial w_m^j}LL\left(y|X;w\right)=\sum_i X_i^j\left[\delta_m(y_i)-p\left(y^i=m|X_i;w\right)\right] \tag{32}$$

$$w^j\leftarrow w^j+\varepsilon\sum_i X_i^j\left[\delta_m(y_i)-p\left(y^i=m|X_i;w\right)\right] \tag{33}$$

- data transformation: linear $\rightarrow$ generalized.

7 the perceptron

- logistic regression as *soft* linear classifier. `signum` smoothed to `exp`.
- stochastic gradient descent (SGD): one random sample i at a time: [1]

$$w^j \leftarrow w^j + \varepsilon X_i^j \left[y_i - p(y^i = 1 | X_i; w) \right] \tag{34}$$

- *non*-stochastic gradiate descent: taking the average over all samples.

$$w^j \leftarrow w^j + \varepsilon \frac{1}{n} \sum_i X_i^j \left[y_i - p(y^i = 1 | X_i; w) \right] \tag{35}$$

  trying to make expected value of $y|x^j = \cdot$ equal: data $\leftarrow$ predicted.

- linear classifiers (boundary linear): naive bayes, logistic regression.
  questions: learning *easy*? performance?
- perceptron experiment: 1) A $\rightarrow$ B x; 2) B predicts y; 3) A reveals y.
  = compute $\hat{y}_i = sign(\vec{v}_k \cdot \vec{x}_i)$, if wrong, $\vec{v}_{k+1} = \vec{v}_k + y_i \vec{x}_i$, where $y = \pm 1$.
  - rule 1: examples near the origin: $\forall \vec{x}_i$, $|\vec{x}_i|^2 \leq R^2$;
  - rule 2: examples separatable: $\exists \vec{u}$, $|u| = 1$, s.t., $\forall \vec{x}_i$, $(\vec{u} \cdot \vec{x}_i) y_i > \gamma$.

  and two lemmas: $\forall k$: $\vec{v}_k \cdot \vec{u} \geq k\gamma$ (by rule 2); $\forall k$, $|v_k|^2 \leq kR^2$ (by rule 1).
  $\rightarrow k < R^2/\gamma^2$, meaning that the less than R^2/γ^2 mistakes are made.
- the Δ trick. add *label* features to the original features.
- the voted perceptron: online to batch learning. when picking at random, $P(error) = k/m$. where m = number of samples. but keeping $\vec{v}_k$ could be expensive, so: sum of $\vec{v}_k$ weighted by m_k/m.
- sparse vectors.

8 neural networks

- basic unit of a neural net: input: $x^i, w^i \rightarrow y = s(f(w^T X_i))$.
- matrix inversion vs. gradient descent : no iterations, no need to specify parameters, closed form solution; however, not universally applicable.
- assume $f(w^T X_i) = w^T X_i$: back to regression and gradient *descent*:

$$\frac{\partial}{\partial w_j} \sum_i^n \left(y_i - w^T X_i \right)^2 = -2 \sum_i^n x_i^j \left(y_i - w^T X_i \right) \tag{36}$$

[1] See also: `http://leon.bottou.org/publications/psgz/nimes-1991.ps.gz`

- otherwise, assume sigmoid function:

$$\frac{\partial}{\partial w^j}\sum_i \left(y_i - g(w^T X_i)\right)^2 = \sum_i 2\left(y_i - g(w^T X_i)\right) g'(w^T X_i) X_i^j \tag{37}$$

where $g'(x) = -g(x)(1-g(x))$, and $g(x) = 1/(1+\exp x)$.

- learn parameters of two-layer neural networks: *backpropagation*.
take partial derivative for each of the weighted parameters:

$$err_i = (y_i - g(w^{(:)^T} g(w^{(:,j)^T} x_i))^2 \tag{38}$$

where $w^{(:)}$ is the weight vector for the last layer; $w^{(:,j)}$ for the second-last layer, and the inner $g(\cdot)$ produces a vector. for the hidden layer:

$$\frac{\partial}{\partial w^{k,j}} err_i = 2\underbrace{(y_i - g(w^T g(w^{j^T} x_i))(-g'(w^T g(w^{j^T} x_i))(w^j}_{\Delta_i^j\text{: the mystically } set \text{ term on the slides}} g'(w^{j^T} x_i)) x_i^k \tag{39}$$

where w^j is the j-th element of $w^{(:)}$. update rule revision is trivial.
note that for consistency we might sum err over all samples.

- neural network encoding: an example.

- historical background:

  - 1st-gen (1960): perceptrons, hand-coded features;
  - 2nd-gen (1985): error signal + backpropagation.

  back-propagation drawbacks: 1) labeled training data; 2) slow; 3) stuck in poor local optima. solution: 1) iteratively learn different layers; 2) weights adjusted to reproduce input. 3) learn features for unlabeled input.

9 support vector machine

- regression classifiers recalled: possibly many classifiers.
max margin classifiers: boundary that leads to the largest margin from points on both sides, aka, support vector machine.

- boundary: $w^T x + b = 0$: $w^T x + b \geq +1 \to +1$, $w^T x + b \leq -1 \to -1$.
margin m defined in terms of w and b how? observations:

  - $\vec{m}$ is orthogonal to ± 1 planes;
  - for x^+ on $+1$ and x^- closest point to x^+ on -1: $x^+ = \lambda w + x^-$

  together with $w^T x^\pm + b = \pm 1$, $|x^+ - x^-| = m$, $\to m = |\lambda w| = 2/\sqrt{w^T w}$,
  now search w that correctly classifies all points and maximize m.

- quadratic programming: $\min_u \left(\frac{1}{2}u^T R u + d^T u\right)$, constraints: $au \leq b$ and $eu = f$. where u, b, f, d vector, R diagonal, a, e matrix: usable on svm.

- non-linearly separable case: errors involved. if $\min w^T w$ and $\min err$, hard to solve; if $\min \left(w^T w + err\right)$, hard to code in qp. $\rightarrow$ solution: minimize distance ε_i between misclassified points and their correct planes:

$$\min_w \left(\frac{1}{2}w^T w + \sum_i^n C\varepsilon_i\right) \tag{40}$$

  where $\forall x_i \in \text{Class}_{1,2}$: $w^T x + b \geq \pm 1 \mp \varepsilon_i$, C constant, with $\varepsilon_i \geq 0$.

- dual formulation: lagrange multiplier for svm optimization:

$$\min \frac{1}{2}(w^T w) \quad \text{subject to} \quad (w^T x_i + b) y_i \geq 1 \tag{41}$$

$$\min_{w,b} \max_{\alpha} \frac{w^T w}{2} - \sum_i \alpha_i \left[\left(w^T x_i + b\right) y_i - 1\right] \quad \text{subject to} \quad \alpha_i \geq 0 \tag{42}$$

  taking the ∂-wrt w, α and b: $w = \sum_i \alpha_i x_i y_i$, $b = y_i - w^T x_i$, $\sum_i \alpha_i y_i = 0$, then substituting w back in to get the dual formulation:

$$\max_{\alpha} \left(\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j^T\right), \sum_i \alpha_i y_i = 0 \quad \text{subject to} \quad \alpha_i \geq 0 \tag{43}$$

- non-separable training set may be separable with additional dimensions. but: 1) high computation burden; 2) more parameters. svm solves them by 1) kernel tricks 2) dual formulation only assigns parameters to samples.

- quadratic kernels. the features with added dimensions are:

$$\Phi(x) = 1, \underbrace{\sqrt{2}x^1, \ldots, \sqrt{2}x^m}_{m}, \underbrace{(x^1)^2, \ldots, (x^m)^2}_{m}, \underbrace{\sqrt{2}x^1 x^2, \ldots, \sqrt{2}x^{m-1}x^m}_{\frac{1}{2}m(m-1)} \tag{44}$$

  but $\forall i, j$, $\Phi(x_i) \cdot \Phi(x_j)^T$ costs $m + m + \frac{1}{2}m(m-1) = O(m^2)$ operations. however $(x_i \cdot x_j + 1)^2$ produces the same thing with only m operations. it also works for higher order scenarios. just find the correct kernels.
  comments: kernel is a trick! we're just doing projection to higher-dimension space, and happen to find that kernel makes it easier to compute dot-product! or, think of it as refining dot-product!

- for non-linearly separable case: bound on α.

10 model and feature selection

- model selection: features (linear, logistic, svm); parameters (prior, regularization, decision tree, clustering).
- *greed heuristic* model selection algorithm:

  1. start with a empty feature set F_0,
  2. $\forall X$, run alogrithm on $F_0 \cup X$; lowest training error $X_j \rightarrow F_{i+1}$.

  optimal at each stage, but not necessarily global optimal. good for most classifiers so far. but stop when? use an independent data-set. data waste! *better*: **leave-one-out** cross validation. *faster*: **k-fold** cross validation.
- regularization: include all features but punish large parameters (logistic), zero probability outcomes (naive bayes). chi-square test (decision tree). e.g.: bayesian learning: λ selected how? **cross validation!**
- minimum description length (mdl): the information theoretic interpretation of regularization: minimize length of data and length of hypothesis.
- the confidence interval bounds:

$$\text{Accuracy}_s(h) \pm Z_n \sqrt{\frac{\text{Accuracy}_s(h)(1 - \text{Accuracy}_s(h))}{n}} \tag{45}$$

11 computational learning theory

- problem setting: instances X, hypotheses H, target functions C, sequence of training instances, noise-free labels $c(x)$: outputs $h = \arg\min_{h \in H} error_{train}(h)$. $error_{train}(h)$ and $error_{true}(h)$. overfitting. confidence interval bounding.
- some definitions:

  - **consistent**: $\forall (x, c(x)) \in D$, $h(x) = c(x)$;
  - **version space**: all $h \in H$ consistent with D;
  - **ϵ-exhausted**: $\forall h \in \text{VS}_{H,D}, error_{true} < \epsilon$.

  and a theorem:

  - $Pr\left(\exists h \in H, \text{s.t.}, error_{train} = 0 \wedge error_{true} > \epsilon\right) \leq |H| e^{-\epsilon m}$

  how many training examples? $m \geq \epsilon^{-1}\left(\ln |H| - \ln \delta\right)$;
  if $error_{train} = 0$, with at least $1 - \delta$: $error_{true} \leq m^{-1}\left(\ln |H| - \ln \delta\right)$.
- pac-learnable: learning in polynomial time.
- more definitions:

  - **dichotomy**: partition into two disjoint subsets.

- **shattered**: S shattered by H if $\exists$ consistent $h \in H \rightarrow$ dichotomy.
- **vc-dimension**: size of largest finite subset of X shattered by H.

to guarantee $\forall h \in H$ perfectly fits training data, need instances

$$m \geq \frac{1}{\epsilon}\left(4\log_2\frac{2}{\epsilon} + 8\mathrm{VC}(H)\log_2\frac{13}{\epsilon}\right) \tag{46}$$

- to show that $VC(H) = d$, we need to
  - There exists a d-sized subset shatterable with arbitrary labels;
  - There is no subset of size $d+1$ the can be shattered.
- agnostic learning: when the true classifier $c \notin H$, we can still bound the true error by training error, and see how close we are to $h^* \in H$ that has a lowest true error, with a probability at least $1-\delta$.

$$error_{true}(h) \leq error_{train}(h) + \sqrt{\frac{\ln(|H|) + \ln(1/\delta)}{2m}} \tag{47}$$

$$error_{true}(h) \leq error_{train}(h) + \sqrt{\frac{VC(|H|)(\ln\frac{2m}{VC(H)} + 1) + \ln(4/\delta)}{m}} \tag{48}$$

- mistake bounds, optimal mistake bounds, (c.f.: the find-s example),

$$VC(C) \leq \min_A M_A(C) \leq \log_2(|C|) \tag{49}$$

where $M_A(C)$ is the max number of mistakes when learning C using A.

- upper bound of $VC(H)$ as a function of $|H|$.

12 clustering

- finding internal structure, underlying rules, recurring patterns and topics... unsupervised learning: no provided labels. **distance**.
- type i: hierarchical: (bottom-up) repetitively find closest pairs and merge. question: how to compute the distance between clusters?
  - single link: distance of two closest members,
  - complete link: distance of two farthest members,
  - average link: average distance of pairs. (robust against noise)

summary: 1) no need to specify the number of clusters, 2) structure interpreted subjectively, 3) poor scalibility, 4) local optima 5) intuitive. using the dendrogram to detect outliers.

- type ii: partitional: specify number of clusters, and place the instances. (top-down) graph-based clustering: the steps:
  - construct the neighborhood graph,
  - assign weights to edges (similarity),
  - partition the nodes using the graph structure.

  how to find the number of clusters? knee/elbow finding!
- cluster validation:
  - internal validation: coherence. average intra/inter-cluster similarity. requiring the definition of similarity and distance metric.
  - internal validation: stability. subject to minor perturbation. requiring measurement of clusters distance: label matrices. (subsampling)
  - external validation: relation to existent categories. the *p-value*: randomly chosen cluster element exists in some category G.

13 more clustering

- distance: minkowski metric, hamming distance, correlation coefficient,
- k-means algorithm: repeat:
  1. decide k and initialize their centers randomly,
  2. decide memberships by assigning objects to nearest cluster centroid,
  3. move the cluster centers closer to the objects assigned to them.

  parallelized easily (map-reduce); sensitive to starting points.
- mixture models (multi-modal density model): gaussian mixture models where all variables are observable:

$$p(x_n|\mu,\Sigma) = \sum_{i=1}^{k} \pi_i N(x|\mu_i,\Sigma_i) \tag{50}$$

and the likelihood for a bunch of data D would be

$$\begin{aligned} l(\theta;D) &= \log \prod_n p(z_n,x_n) = \log \prod_n p(z_n|\pi)p(x_n|z_n;\mu,\sigma) \\ &= \sum_n \log \prod_k \pi_k^{z_n^k} + \sum_n \log \prod_k N(x_n;\mu_k,\sigma_k)^{z_n^k} \\ &= \sum_n \sum_k z_n^k \log \pi_k - \sum_n \sum_k z_n^k \frac{1}{2\sigma_k^2}(x_n-\mu_k)^2 \end{aligned} \tag{51}$$

which gives the parameter estimates as follows:

$$\hat{\pi} = \sum_i z_i^k/|D|; \quad \hat{\mu} = \frac{\sum_n z_n^k x_n}{\sum_n z_n^k}; \quad \hat{\sigma} = ? \tag{52}$$

- exceptation-maximization (em) algorithm: we're not going through this... but know that K-means is a hard version of em.

14 bias-variance

- as a way to understand overfitting and underfitting: decomposition. *setting*: true function f, noise ε, $h_D(\cdot)$ hypothesis learned from D, error:

$$\text{error} = E_{D,\varepsilon}\left\{\left(f(x) + \varepsilon - h_D(x)\right)^2\right\} \tag{53}$$

  more notations: $f \equiv f(x)+\varepsilon$, $\hat{y} = \hat{y}_D \equiv h_D(x)$, and fact: $h \equiv E_D\left\{h_D(x)\right\}$.

$$\text{error} = E_{D,\varepsilon}\left\{(f-\hat{y})^2\right\} = E_{D,\varepsilon}\left[(f-h)^2\right] + E_{D,\varepsilon}\left[(h-\hat{y})^2\right] \tag{54}$$

  where the first term is $\texttt{BIAS}^2$ (*you tried so hard but still couldn't get to the original f*), second `VARIANCE` (*your each individual attempt may deviate from your best performance.*)

- tradeoffs: *large bias, small variance*: few features, highly regularized, highly pruned decision trees, large k-NN. *large variance, small bias*: many features, less regularization, unpruned trees, small k-NN.

- generalization: (which is not confusing at all):
  - optimal prediction: $y^* = \arg\min_{y'} L(t, y')$;
  - main prediction of **learner**: $y_m = y_{m,D} = \arg\min_{y'} E_D\left\{L(y, y')\right\}$;
  - bias of **learner**: $\texttt{BIAS} = L(y^*, y_m)$;
  - variance of **learner**: $\texttt{VARIANCE} = E_D\left[L(y_m, y)\right]$;
  - noise: $\texttt{NOISE} = E_t\left[L(t, y^*)\right]$

  bias: more expressive (complex)! variance: less expressive (complex)!

- how to approximate $E_D\left\{h_D(x)\right\}$? background: *bootstrap* sampling:
  input: dataset D, **output**: variants: D_1, D_2, ..., D_T.
  algorithm: for each $D_{(\cdot)}$, pick $|D|$ instances uniformly from D.
  then, train on $D_{(\cdot)}$ and test on remaining data to get $h_1(x), h_2(x)$,...
  variance = ordinary variance of $h_{(\cdot)}(x)$, bias = $\bar{h}_{(\cdot)}(x) - y$.

- bagging: bootstrap aggregation. **input**: dataset D, and YFCL (wtf);
  output: a classifier $h_{\text{D,BAG}}$. **algorithm**: get $D_{(\cdot)}$ and train YFCL on it to get $h_{(\cdot)}$, then to classify do majority vote with $h_{(\cdot)}$. generally bagged decision trees outperform linear classifiers if the data is large and clean.

- more details on the confusing generalization: for (x^*, y^*) a single instance.
  - noisy channel: $y_i = \text{noise}(f(x_i))$, may change $y \to y'$;

- learned hypothesis: $h = h_D$ from some dataset D;
- main prediction of **learner**: $y_m(x^*) = \arg\min_{y'} E_{D,P}\left\{L(h(x^*), y')\right\}$;
- bias of **learner**: $B(x^*) = L(y_m(x^*), f(x^*))$;
- variance of **learner**: $V(x^*) = E_{D,P}\left\{L(h_D(x^*), y_m(x^*))\right\}$
- noise: $N(x^*) = L(y^*, f(x^*))$.

and yes, we can do case analysis. but no.

15 boosting

- make weaker classifiers better: lemma ideas:
  - simple: models $\rightarrow$ one with lowest training error, but what if many?
  - stacked learners: train new classifier on predicted and true labels.
- ensemble methods: divide and conquer. a final classifier = a linear combination of votes of different classifiers weighted by their strength.
  weighted data: i-th training example counts as $D(i)$ examples.
- adaboost: given (x_i, y_i) where $x_i \in X$, $y_i \in Y = \{-1, +1\}$, $i = 1, 2, \ldots, m$. initialize $D_1(i) = m^{-1}$, and iterator over $t \in \{1, 2, \ldots, T\}$:
  1. train weak learner using distribution D_t and get $h_t : X \rightarrow \mathbb{R}$;
  2. choose $\alpha_t \in \mathbb{R}$, and update

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t = \sum_{i=1}^m D_t(i)\exp(-\alpha_t y_i h_t(x_i))} \tag{55}$$

  and output the final classifier:

$$H(x) = \text{sign} f(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right) \tag{56}$$

- problem: how to choose α_t? final training error is bounded by:

$$\frac{1}{m}\sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \frac{1}{m}\sum_{i=1}^m \exp(-y_i f(x_i)) = \prod_t Z_t \tag{57}$$

  so minimizing Z_t minimizes the training error. we define that

$$\epsilon_t = \sum_{i=1}^m D_t(i)\delta(h_t(x_i) \neq y_i) \quad \text{s.t.,} \quad Z_t = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \tag{58}$$

  hence take $\partial Z_t / \partial \alpha_t$ tells you that $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$, bang!
- if each classifier is better than random, this gives you zero training error exponentially fast. also: robust to overfitting, testing error decreases after.