Ensemble Techniques
(also known as Committees)

Professor Michael C. Mozer
CSCI 3202

Committees

Idea: Combine outputs of multiple models to achieve higher accuracy

Related ideas

mixture of experts (but really picking 1 of n)

Bayesian approach (but really combining an infinite set of models)

Committee will not help if all members are identical.

Ways of giving each model slightly different expertise:

- different training sets
- different weighting or distribution of training examples (bagging, boosting)
- different subsets of input features
- different classes of models (e.g., neural net, decision trees, KNN)
- different neural network architectures (e.g. vary # hidden units)
- different local optima (e.g., different initial weights)

Committees work best when errors of members are uncorrelated and have zero mean

⇒ choose models with small bias, allow committee to reduce variance by averaging

Combination rules for committees.

1. unweighted averaging

$$y_{com} = \frac{1}{L}\sum_{i=1}^{L} y_i$$

Committee error no greater than avg. error of members:

$$E[(y_{com}-t)^2] \leq E[(y_i - t)^2]$$

But committee response may not be as good as best member.

2. weighted averaging

$$y_{com} = \sum_{i=1}^{L} \alpha_i y_i \qquad \sum \alpha_i = 1 \qquad \alpha_i \geq 0$$

pick $\{\alpha_i\}$ to minimize error of committee on training data

$$\alpha_i = \sum_j (C^{-1})_{ij} \Big/ \sum_k \sum_l (C^{-1})_{kl}$$

$$C_{ij} \cong \frac{1}{P}\sum_{n=1}^{P} (y_i^{(n)} - d^{(n)})(y_j^{(n)} - d^{(n)})$$ correlation matrix

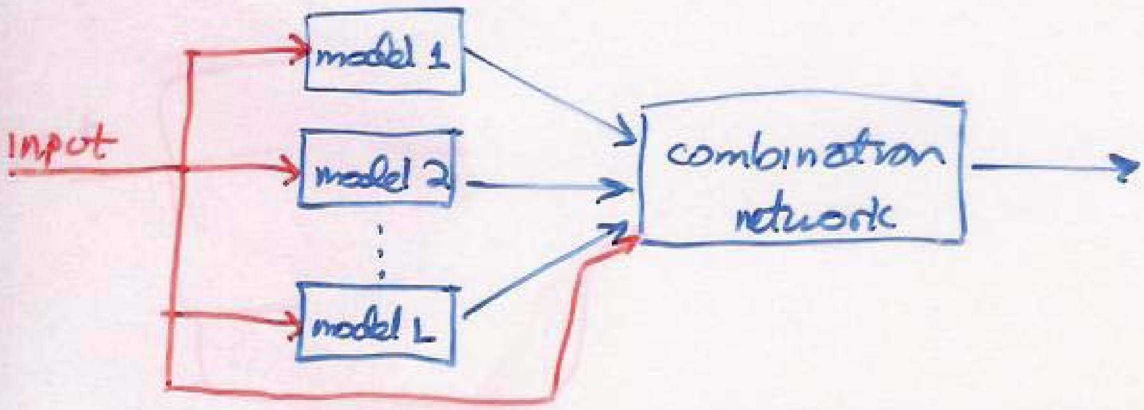
α_i will be large to the extent that member i produces outputs not highly correlated with those of any other member.

Combination rules for committees (cont.)

3. Stacked generalization (Wolpert, 1992)

Perform cross-validation training for each member

Obtain prediction for each member for each validation example



Use validation output of members to produce training set for combination network

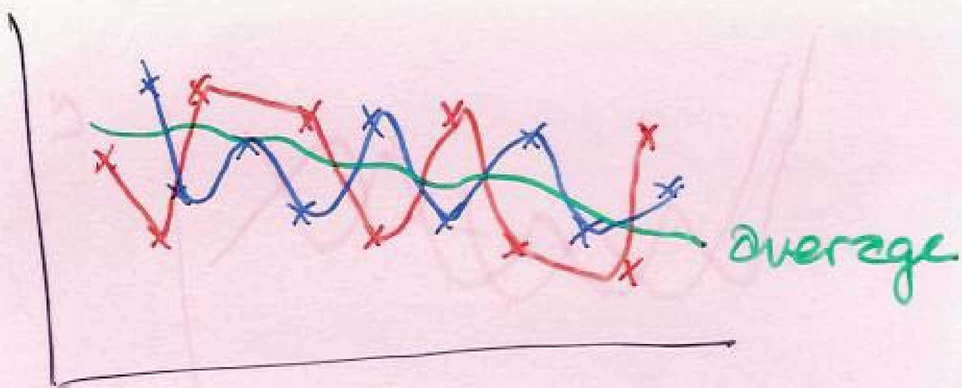
Bagging (Breiman, 1994)

Create multiple training sets by sampling
with replacement from original training set

Train algorithm on selected examples to obtain members

Use unweighted average combination rule

Improves generalization performance by reducing
variance.



AdaBoost (Schapire & Freund, 1996)

Reweights examples for member i based on training performance of member i-1: FOCUS ON HARD EXAMPLES!

$D_t(j)$: weight of example j for member t

$$D_1(j) = \frac{1}{P}$$

$$\hat{D}_{t+1}(j) = D_t(j) \times \begin{cases} \beta_t & \text{if example } j \text{ correctly classified by member } t \\ 1 & \text{otherwise} \end{cases}$$

$$D_{t+1}(j) = \hat{D}_{t+1}(j) \Big/ \sum_k \hat{D}_{t+1}(k)$$

normalize such that $\sum D(j) = 1$

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

where $\varepsilon_t \equiv$ proportion of training examples misclassified by member t, weighted by D_t

Combination rule:

$$\alpha_t = \frac{\ln(1/\beta_t)}{\sum_s \ln(1/\beta_s)}$$

$$y_{com} = \sum \alpha_t y_t$$

AdaBoost guaranteed to lower training error.

Will also lower test error if (a) enough data, & (b) final combination model not too complex

Combination rule is the same as Naive Bayes (i.e., optimal combination rule assuming members are independent of one another)

Boosting has surprising resistance to overfitting.

Boosting seems to improve generalization performance even when training error of committee is reduced to zero!

Why? increases confidence of classification

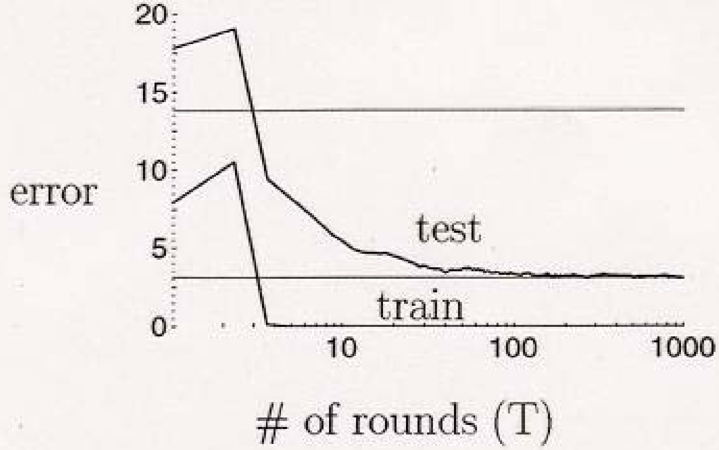
Boosting vs Bagging:

- bagging uses unweighted combination
- bagging does not change distribution from one member to the next

From Schapire & Freund

Actual typical run of **AdaBoost**

(boosting C4.5 on "letter" dataset)



- test error does *not* increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
% train error	0.0	0.0	0.0
% test error	8.4	3.3	3.1

- Similar results reported ([Cortes & Drucker96,Breiman96, Quinlan96])
- Occam's razor predicts "simpler" rule is better
  - clearly wrong in this case!

Comparison: Boosting decision rules vs. C4.5

Percent error on test set.

Name	Boost Stump	C4.5	Boost C4.5	Bag C4.5	Bag Stump
soybean-small	0.2	2.2	3.4	2.2	20.5
labor	9.0	15.8	13.1	11.3	18.9
promoters	9.1	22.0	5.0	12.7	17.2
iris	4.8	5.9	5.0	5.0	7.1
hepatitis	18.3	21.2	16.3	17.5	17.4
sonar	16.8	28.9	19.0	24.3	25.9
glass	29.4	31.7	22.7	25.7	54.2
audiology.stand	23.6	23.1	16.2	20.1	65.7
cleve	18.8	26.6	21.7	20.9	21.9
soybean-large	9.8	13.3	6.8	12.2	74.2
ionosphere	8.5	8.9	5.8	6.2	17.2
house-votes-84	3.7	3.5	5.1	3.6	4.4
votes1	8.9	10.3	10.4	9.2	12.7
crx	14.4	15.8	13.8	13.6	14.5
breast-cancer-w	4.4	5.0	3.3	3.2	6.6
pima-indians-di	24.5	28.4	25.7	24.4	26.0
vehicle	26.1	29.9	22.6	26.1	56.1
vowel	18.2	2.2	0.0	0.0	74.7
german	24.9	29.4	25.0	24.6	30.3
segmentation	4.2	3.6	1.4	2.7	72.5
hypothyroid	1.0	0.8	1.0	0.8	2.2
sick-euthyroid	3.0	2.2	2.1	2.1	5.6
splice	4.4	5.8	4.9	5.2	33.4
kr-vs-kp	4.4	0.5	0.3	0.6	31.3
satimage	14.9	14.8	8.9	10.6	41.6
agaricus-lepiot	0.0	0.0	0.0	0.0	11.3
letter-recognit	34.1	13.8	3.3	6.8	93.7

Bayesian approach

GREATLY OVERSIMPLIFIED
See work by MacKay, Neal

M: "model" (neural net architecture, weights, etc.)

D: "data" (training data)

Bayes rule:
$$\underbrace{P(M_i|D)}_{\text{posterior}} = \frac{\overset{\text{likelihood}}{P(D|M_i)}\,\overset{\text{prior}}{P(M_i)}}{\underbrace{P(D)}_{\text{evidence}}}$$

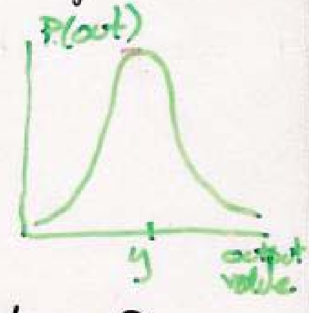
Goal of learning: find i such that $P(M_i|D)$ is maximized

$P(D)$ is independent of M_i & hence is irrelevant for model sel.

$P(D|M_i)$: how likely is network to have produced a given set of outputs in response to a given set of inputs?

Common assumption: Gaussian distribution

$$P(D|M_i) \sim \exp\Big[-\sum_{j,p} (y_j^p - d_j^p)^2\Big]$$



$P(M_i)$: based on prior beliefs about plausibility of different models, e.g., weights come from a particular distribution

Finding i that maximizes $P(M_i|D)$ is equivalent to maximizing

$$\begin{aligned}\log P(M_i|D) &\sim \log\big[P(D|M_i)P(M_i)\big] \\ &= \log P(D|M_i) + \log P(M_i) \\ &= -\sum_{j,p}(y_j^p - d_j^p)^2 + \log P(M_i)\end{aligned}$$

← the usual error function!

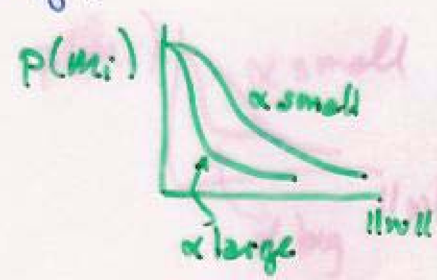
Reinterpreting weight decay in Bayesian terms

Suppose we consider the model class to consist of all the possible parameterizations of a neural net with weights $\vec{w}$.

If we believe that small weights are, a priori, more likely than large weights, we might assume

$$P(M_i) = p(\vec{w}_i) \sim \exp\left(-\frac{\alpha}{2}\|\vec{w}_i\|^2\right)$$

$$= \exp\left(-\frac{\alpha}{2}\sum_j w_{ij}^2\right)$$



and $\log P(\vec{w}_i) \sim -\frac{\alpha}{2}\sum_j w_{ij}^2$

By maximizing $\log P(M_i|D) \sim \log P(D|M_i) + \log P(M_i)$ via gradient descent, we must make weight changes:

$$\Delta w_k \sim \frac{\partial}{\partial w_k}\log P(M_i|D)$$

$$\sim \frac{\partial}{\partial w_k}\log P(D|M_i) + \underbrace{\frac{\partial}{\partial w_k}\log P(M_i)}$$

$$\frac{\partial}{\partial w_k}\left(-\frac{\alpha}{2}\sum_j w_j^2\right) = -\alpha w_k$$

Bayesian approach to learning

Standard learning algs find the set of weights $\vec{w}$ that maximizes

$$p(D \mid \vec{w}) \equiv p(\text{target outputs} \mid \vec{w}, \text{inputs})$$

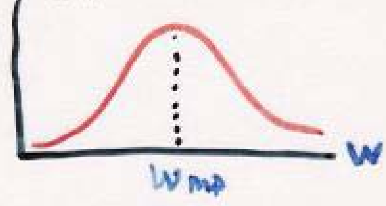
Rather than finding a single set of weights that is most likely to have produced the data (i.e., maximum likelihood), we could try to find

$$p(y \mid D, \vec{x}) = \int_{\vec{w}} p(y \mid D, \vec{x}, \vec{w})\, p(\vec{w} \mid D)\, d\vec{w}$$

I.e., determine prediction of y for every possible $\vec{w}$ and compute weighted average — weighting dependent on how good $\vec{w}$ is given D

Not possible to do in practice
Two approximations:
1) Gaussian approximation to posterior distribution of $\vec{w}$

$p(w \mid D)$

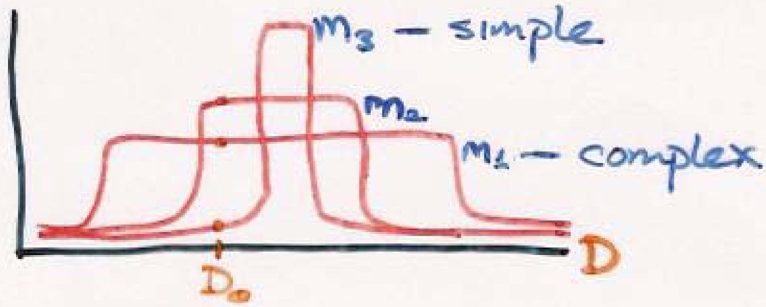


w_{mp}: most probable w, determined by std learning procedure

2) Monte Carlo methods — randomly sample in $\vec{w}$ but use tricks to find more likely $\vec{w}$

Other potential applications of Bayesian approach

1) model selection



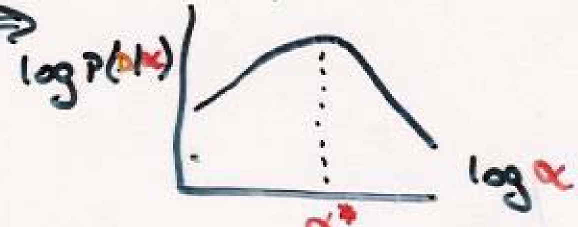
2) hyperparameter selection

$$p(\vec{w}|D) = \int p(\vec{w}, \alpha | D)\, d\alpha = \int p(\vec{w}|\alpha, D)\, \underbrace{p(\alpha|D)}\, d\alpha$$

$$p(\alpha|D) = \frac{p(D|\alpha)\, p(\alpha)}{p(D)}$$

Guess $p(\alpha)$ and estimate $p(D|\alpha)$ by making a few assumptions & computing second derivatives

$\Rightarrow$



3) estimate output uncertainty

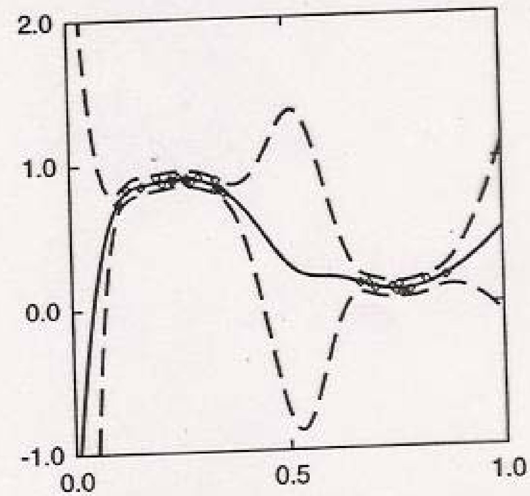


Figure 10.9. A simple example of the application of Bayesian methods to a 'regression' problem. Here 30 data points have been generated by sampling the function (10.35), and the network consists of a multi-layer perceptron with four hidden units having 'tanh' activation functions, and one linear output unit. The solid curve shows the network function with the weight vector set to $\mathbf{w}_{MP}$ corresponding to the maximum of the posterior distribution, and the dashed curves represent the $\pm 2\sigma_t$ error bars from (10.34). Notice how the error bars are larger in regions of low data density.

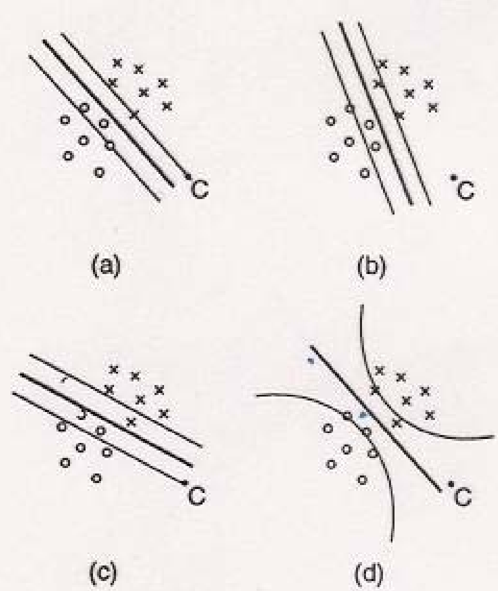


Figure 10.11. Schematic illustration of data from two classes (represented by circles and crosses) showing the predictions made by a classifier with a single layer of weights and a logistic sigmoid output unit. (a) shows the predictions made by the network with the weights set to their most probable values $\mathbf{w}_{MP}$. The three lines correspond to network outputs of 0.1, 0.5 and 0.9. A point such as C, which is well outside the region containing the training data, is classified with great confidence by this network. (b) and (c) show predictions made by the weight vectors corresponding to $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ in Figure 10.10. Notice how the point C is classified differently by these two networks. (d) shows the effects of marginalizing over the distribution of weights given in Figure 10.10. We see that the probability contours spread out in regions where there is little data. The point C is now assigned a probability close to 0.5 as we would expect.