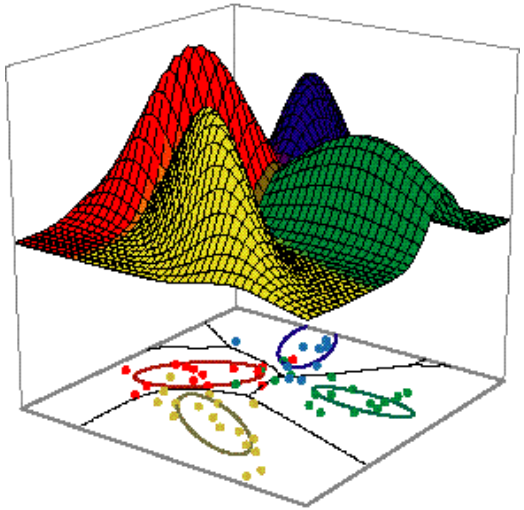
Part 12: Meta Learning



Introduction

Arcing, Bagging, and Boosting

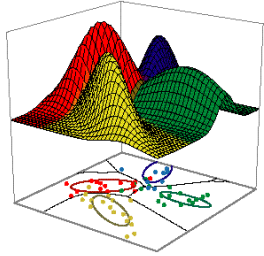
Case study: AdaBoost

Learning with Queries

Combining Classifiers

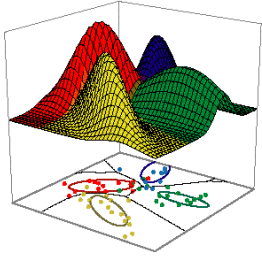
Some materials in these slides were taken from Pattern Classification (2nd ed) by R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, 2000, **Chapter 9.5, 9.7**.

Meta-Learning



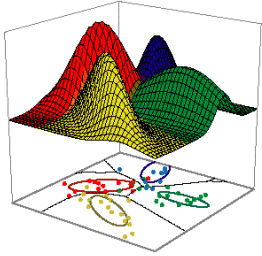
- Until now, have considered how to build the best single classifier
- Possible to achieve greater performance (accuracy, stability, etc) by effective combination of multiple classifiers
  - Irrespective of training algorithm used with each individual classifier
- First consider combinations of classifiers of same type by resampling training data
- Then consider how to combine multiple classifiers, potentially of different types.

Resampling for Classifier Design (9.5)



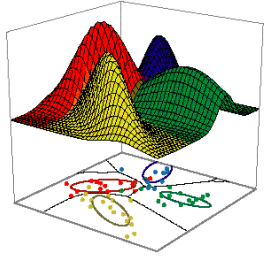
- Arcing: *A*daptive *R*eweighting & *C*ombining
  - "refers to reusing or selecting data in order to improve classification"
  - Train a number of component classifiers on different subsets of training data
  - May choose '*most informative*' subset of data to train next component classifier
    - Normally a function of how well sample is classified by previously added component classifiers.
  - Two most common types: Bagging & Boosting

Bagging



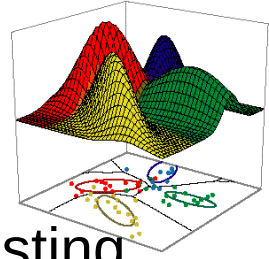
- Bagging = ‘bootstrap aggregation’
- Use multiple versions of the training set
  - each created from n’ < n samples drawn from D with replacement
  - each bootstrap subset used to train a *component classifier*
    - Normally each classifier has the same form
- Bagging improves recognition for unstable classifiers, by averaging over a number of classifiers
  - e.g. decision trees trained using greedy search that change drastically when a single training point is omitted
- Final classification is a simple vote among component classifiers

Boosting



- Goal: improve accuracy of any given learning algorithm
- Outline:
  - Start with a component classifier with ‘better than chance’ accuracy over training data (apparent error)
  - Continue to add component classifiers (each better than chance) to obtain arbitrary performance over training data.
  - Classification performance has been ‘boosted’
  - Each component classifier is trained on the subset of data deemed to be ‘most informative’
  - Final decision derived from component classifier decisions
    - Can use weighted voting – see Adaboost…

Boosting example



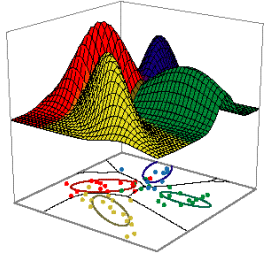
- Example: train 3 classifiers in a 2-class problem using boosting
  - 1) Randomly select subset of training patterns from D (without replacement) → D_1 ($n_1 < n$)
  - 2) Train C_1 over D_1, Need only be a weak learner
    - i.e. better than chance (this is only a minimum requirement)
  - 3) Create subset D_2 ($n_2<n$) where ½ of patterns were correctly classified by C_1, and ½ were incorrectly classified by C_1

To build D_2

    - Test all remaining patterns ($D-D_1$) sequentially using C_1
    - Flip a coin; heads → add the next pattern correctly classified by C_1 to D_2; tails → add the next pattern incorrectly classified by C_1 to D_2
    - Continue until no more patterns can be added in this way

  - 4) Train C_2 on D_2
  - 5) Build subset D_3 by adding all remaining patterns that are not correctly classified by voting of C_1 and C_2
    - Add all remaining patterns for which C_1 and C_2 disagree
  - 6) Train C_3 on D_3
  - 7) Classification now based on voting between $C_{1,2,3}$

Boosting example



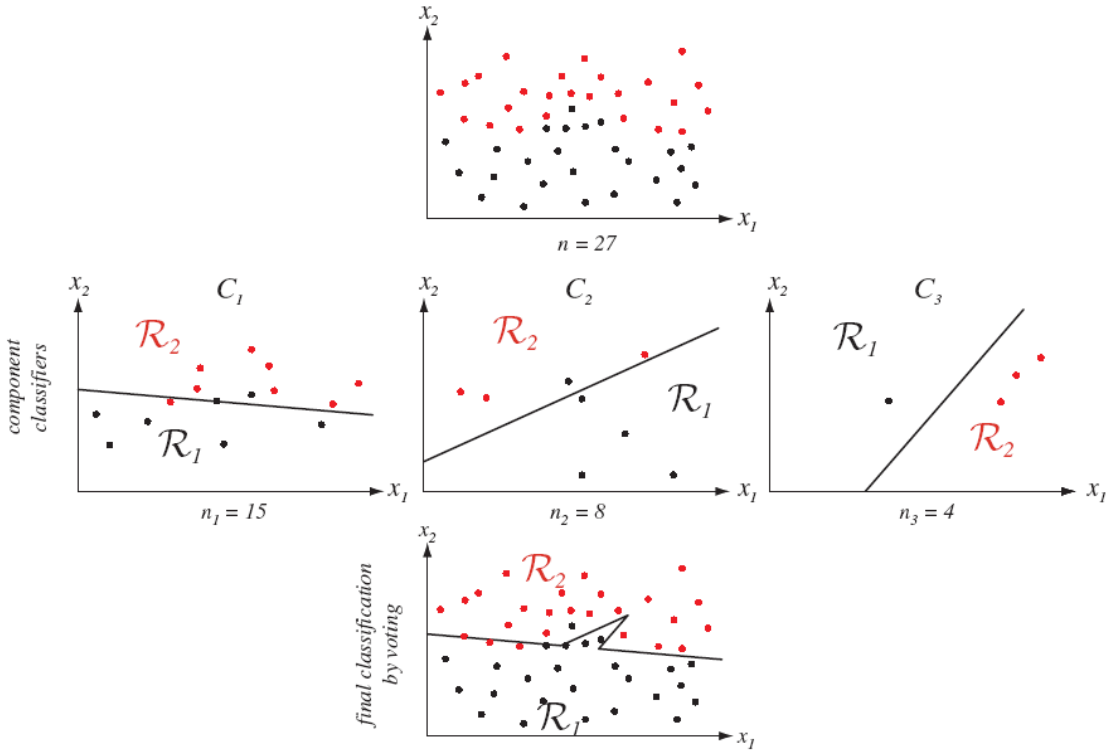
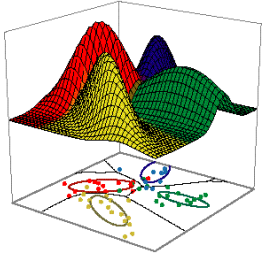


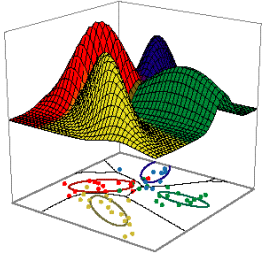
FIGURE 9.6. A two-dimensional two-category classification task is shown at the top. The middle row shows three component (linear) classifiers C_k trained by LMS algorithm (Chapter 5), where their training patterns were chosen through the basic boosting procedure. The final classification is given by the voting of the three component classifiers and yields a nonlinear decision boundary, as shown at the bottom. Given that the component classifiers are weak learners (i.e., each can learn a training set at least slightly better than chance), the ensemble classifier will have a lower training error on the full training set $\mathcal{D}$ than does any single component classifier. Of course, the ensemble classifier has lower error than a single linear classifier trained on the entire data set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Boosting



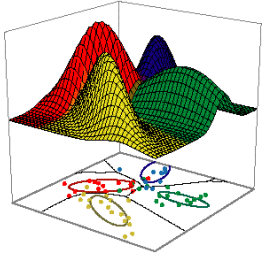
- Difficulty: how to use all training data and have roughly balanced training set sizes ($n_1 \approx n_2 \approx n_3 \approx n/3$)?
  - We only have control over initial size n_1
  - If problem is too easy, C_1 will explain most of data and n_2 & n_3 will be small and not all training data will be used.
  - If problem is too hard, C_1 will perform poorly and n_2 will be unacceptably large. (since C_1 output would be ~rand)
- Solution: rerun boosting a few times with different choices of n_1
  - Some heuristics available…
- Can recursively apply boosting to $C_{1\text{-}3}$ leading to 9 or even 27 component classifiers

AdaBoost



- ‘Adaptive Boosting’ - popular variation on boosting.
- Can continue adding weak learners until arbitrary training accuracy reached.
- Each training pattern receives probability of being included in training subset for next weak learner
  - Increase probability if it is not classified correctly; decrease if existing weak learners already classify it correctly.
  - Allows AdaBoost to ‘focus in’ on informative/difficult patterns.

AdaBoost



- Steps:
  - 1) Initialize weights W_1 across training set to uniform
  - 2) For each iteration, k, draw a random set according to weights W_k, then train classifier C_k
  - 3) Decrease or increase weights in W_{k+1} depending on correct classification of each training pattern by C_k
    - Magnitude of weight change, α_k, depends on weighted training error rate, E_k, (i.e. error rate computed over all samples weighted by W_k)

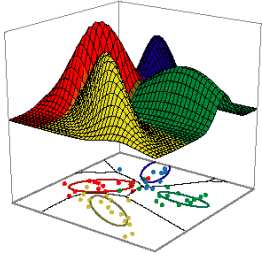
$$\alpha_k = \frac{1}{2}\ln\left[\frac{1-E_k}{E_k}\right] \qquad W_{k+1}(i) = \frac{W_k(i)}{z_k} e^{\pm\alpha_k}$$

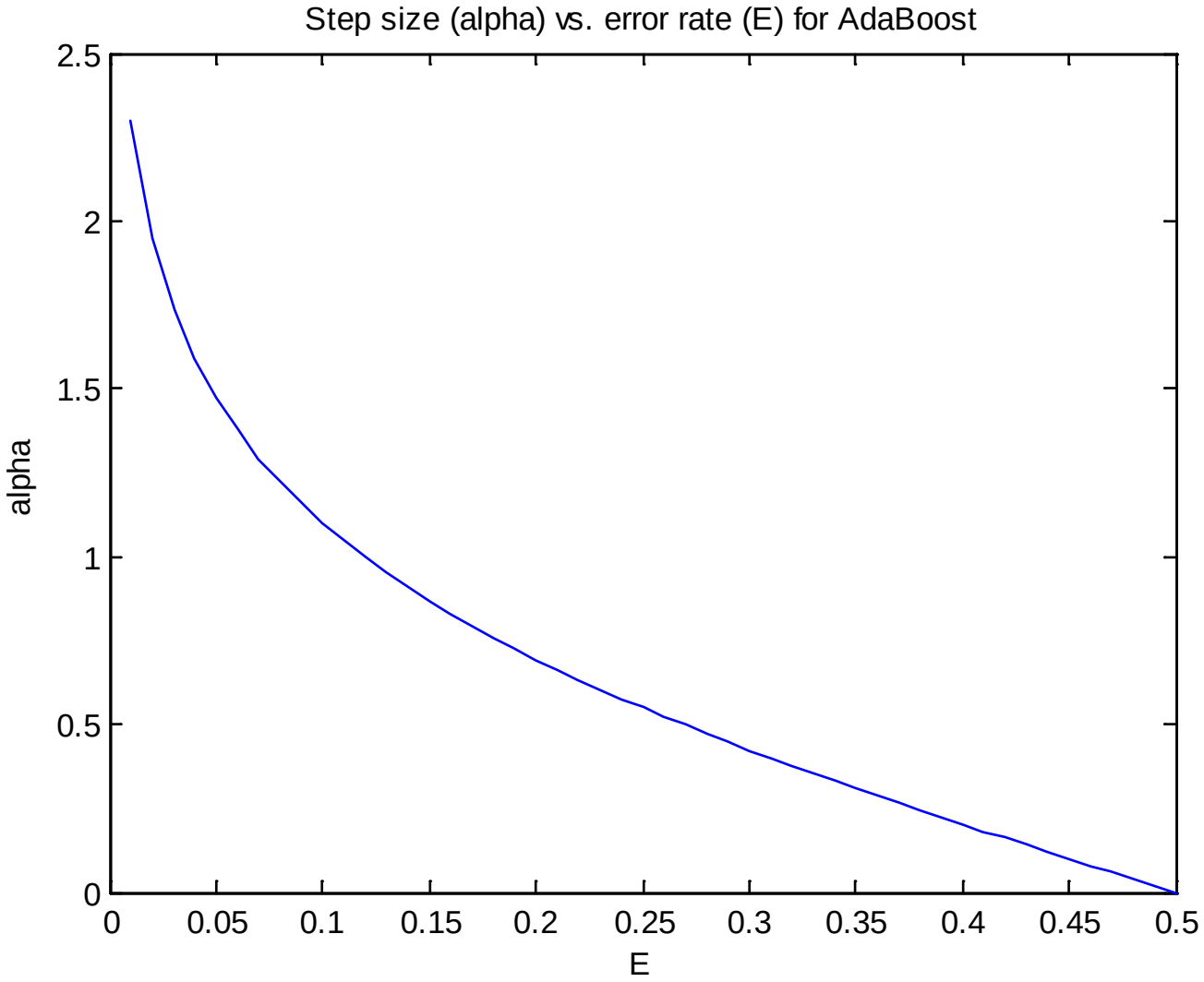
+ if $h_k(x^i)$ incorrect
- if $h_k(x^i)$ correct
h is pred class

scaling constant

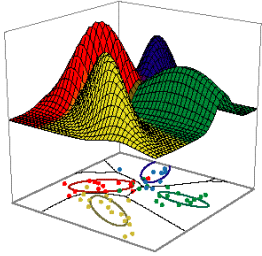
  - 4) Repeat (back to 2) until $k=k_{max}$
  - 5) Return C_k's and α_k's

AdaBoost Step Size (α)





AdaBoost

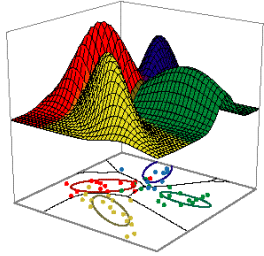


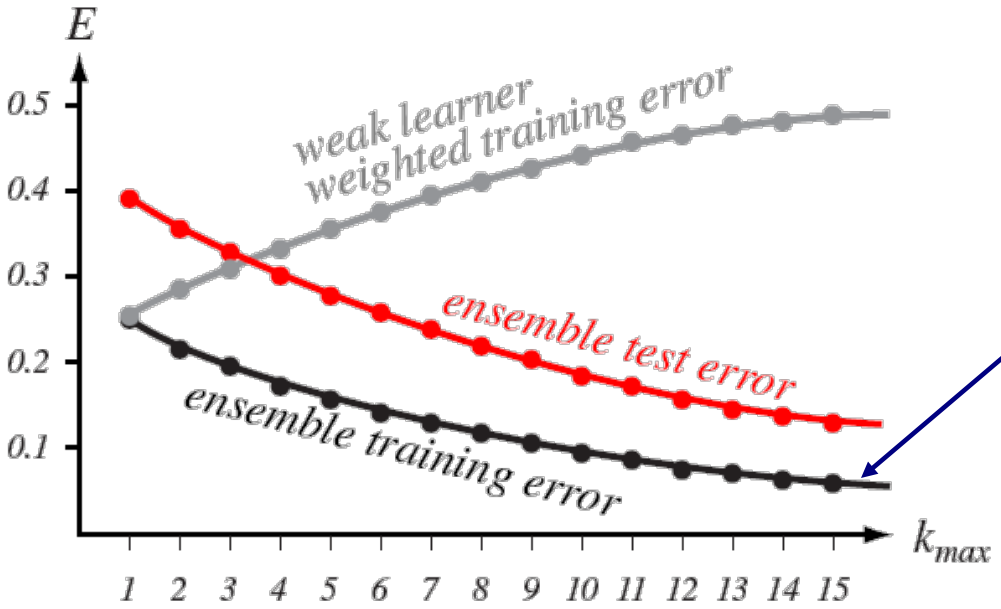
- Final classification vote among k_{max} weak learners C_k, each weighted by α_k

$$g(x) = \sum_{k=1}^{k_{\max}} \alpha_k h_k(x) \qquad \textit{(eq 36)}$$

- As long as each component classifiers is at least a weak learner, training error of ensemble can be made arbitrarily low by increasing k_{max} (exponential reduction)
  - Can often set k_{max} very high without overfitting.
- Does this violate *No Free Lunch Theorem?*
  - 1) Component classifiers may not perform better than chance if we have selected the wrong model (choose another)
  - 2) This only guarantees arbitrary reduction in <u>apparent error</u>
    - However it has shown to be widely useful

AdaBoost



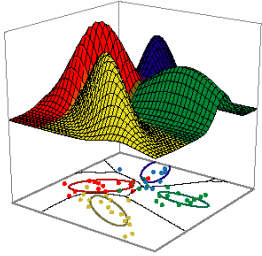


$$E \leq \exp\left(-2\sum_{k=1}^{k_{\max}} G_k^2\right)$$

where $G_k = ½ - E_k$

FIGURE 9.7. AdaBoost applied to a weak learning system can reduce the training error E exponentially as the number of component classifiers, k_{max}, is increased. Because AdaBoost "focuses on" *difficult* training patterns, the training error of each successive component classifier (measured on its own weighted training set) is generally larger than that of any previous component classifier (shown in gray). Nevertheless, so long as the component classifiers perform better than chance (e.g., have error less than 0.5 on a two-category problem), the weighted ensemble decision of Eq. 36 ensures that the training error will decrease, as given by Eq. 37. It is often found that the test error decreases in boosted systems as well, as shown in red. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

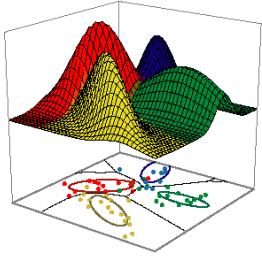
Learning with queries / Active learning



- If patterns are unlabeled, cannot apply bagging/boosting.
- What if there is a (costly) way to label points?
  - How to choose which points to label?
    - How to tell which training patterns are most informative?
  - Assume you have a teacher/oracle who can label points without error
  - Called learning with queries, active learning, interactive learning. Extend to *cost-based learning*

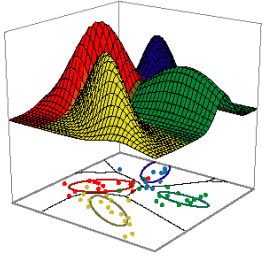
minimize overall cost of accuracy & data collection

Learning with queries / Active learning



- Example: have a large corpus of handwriting
  - Too expensive to label all samples
  - Use teacher/oracle to label a few training patterns
  - Train supervised method
  - Patterns closest to decision boundary clearly candidates for being 'informative' → pass to oracle to label…
- Label patterns according to either:
  - 1) *confidence-based query selection*
    - e.g. for c discriminant functions $g_i(x)$, select points where top two discriminant functions are almost equal
  - 2) *voting-based query selection*
    - e.g. for c component classifiers, select points where vote is closest. Does not require analog discriminant function.
- Can also be used to <u>generate</u> new samples
  - e.g. create surrogate data by distorting existing patterns and passing them to the oracle.

Learning with queries / Active learning



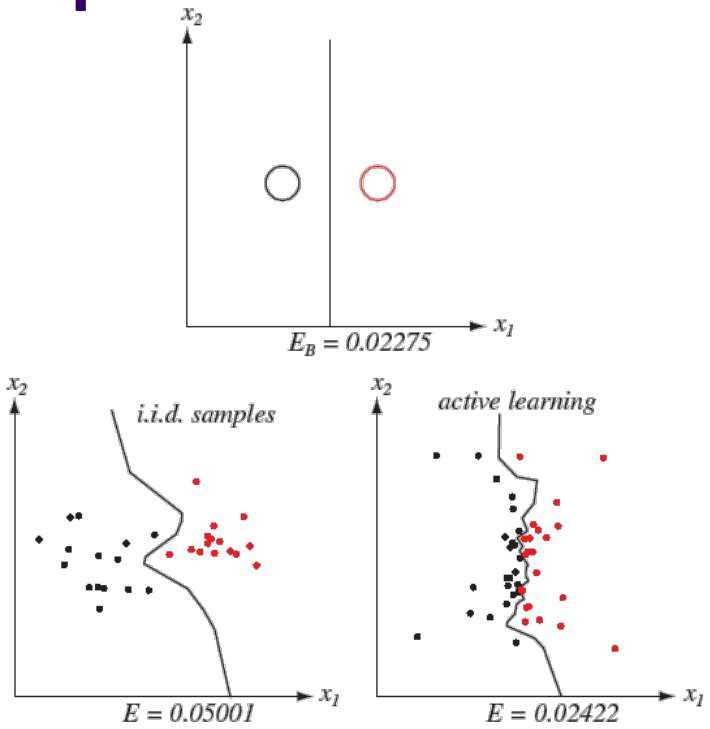
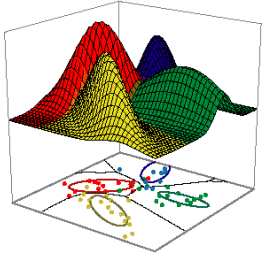


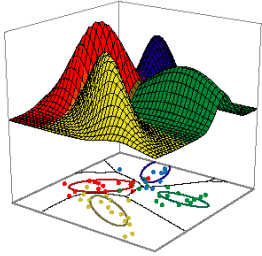
FIGURE 9.8. Active learning can be used to create classifiers that are more accurate than ones using i.i.d. sampling. The figure at the top shows a two-dimensional problem with two equal circular Gaussian priors; the Bayes decision boundary is a straight line and the Bayes error E_B equals 0.02275. The bottom figure on the left shows a nearest-neighbor classifier trained with $n = 30$ labeled points sampled i.i.d. from the true distributions. Note that most of these points are far from the decision boundary. The figure at the right illustrates active learning. The first four points were sampled near the extremes of the feature space. Subsequent query points were chosen midway between two points already used by the classifier, one randomly selected from each of the two categories. In this way, successive queries to the oracle "focused in" on the true decision boundary. The final generalization error of this classifier, $E = 0.02422$, is lower than the one trained using i.i.d. samples, $E = 0.05001$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

An Example: PTM Prediction

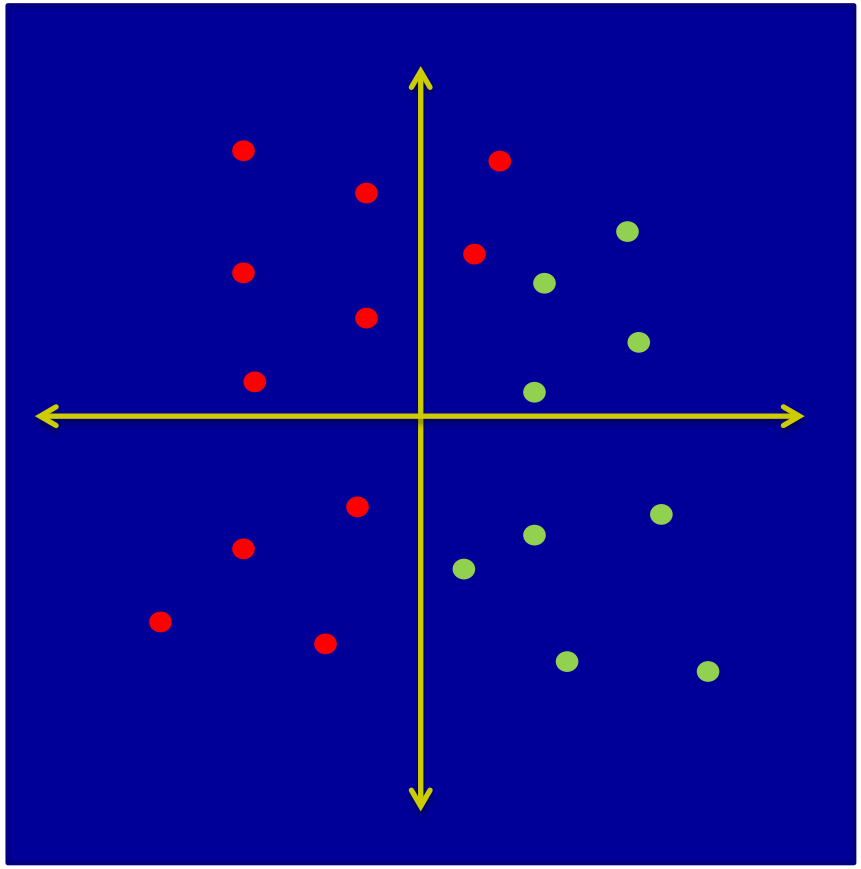


- Known N/D hydroxylation data limited
  - Identified 40 known hydroxylation targets
    - dbPTM & literature review
  - 60 positives sites, 1980 (*presumed*) negatives
    - Extracted windows of ±7 AAs around N/D
    - Eliminated duplicate windows: 47+, 1223-
  - Trained/evaluated SVM using LOO test
    - 92.7% recall; 61.45% precision
- Applied to all 1.3M N/D in human proteins
  - *Now what?*

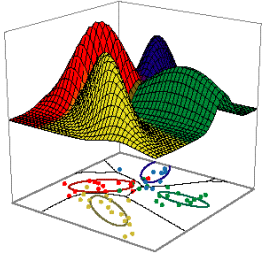
Active Learning



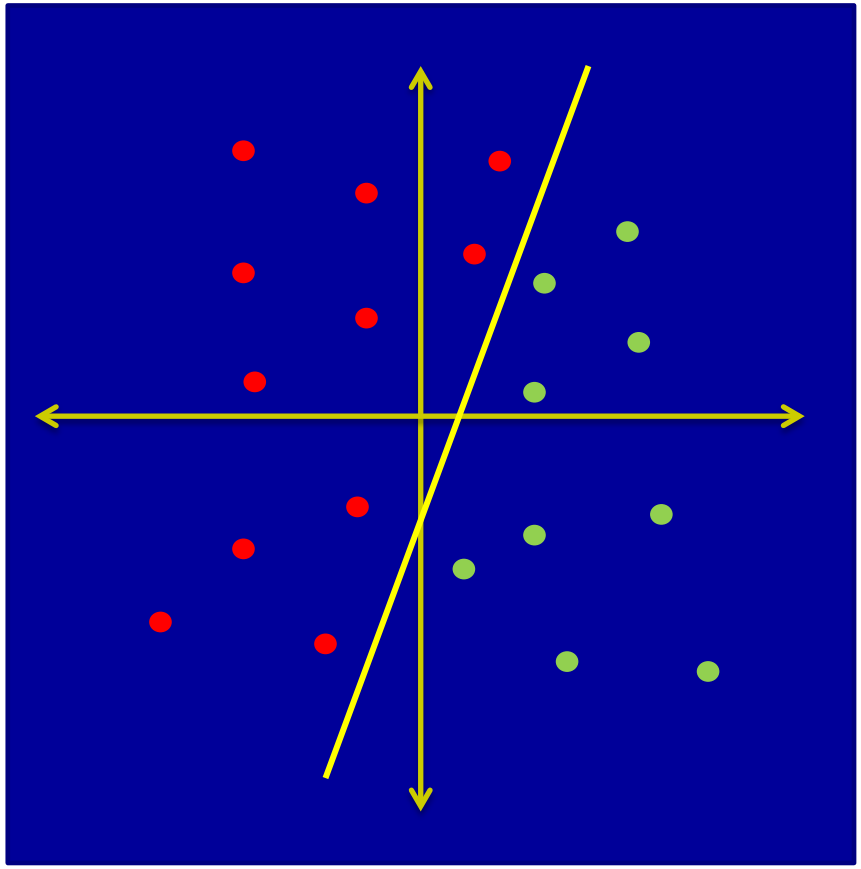
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



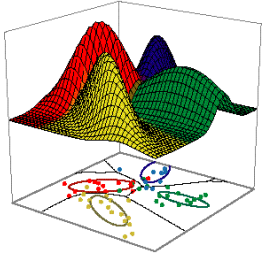
Active Learning



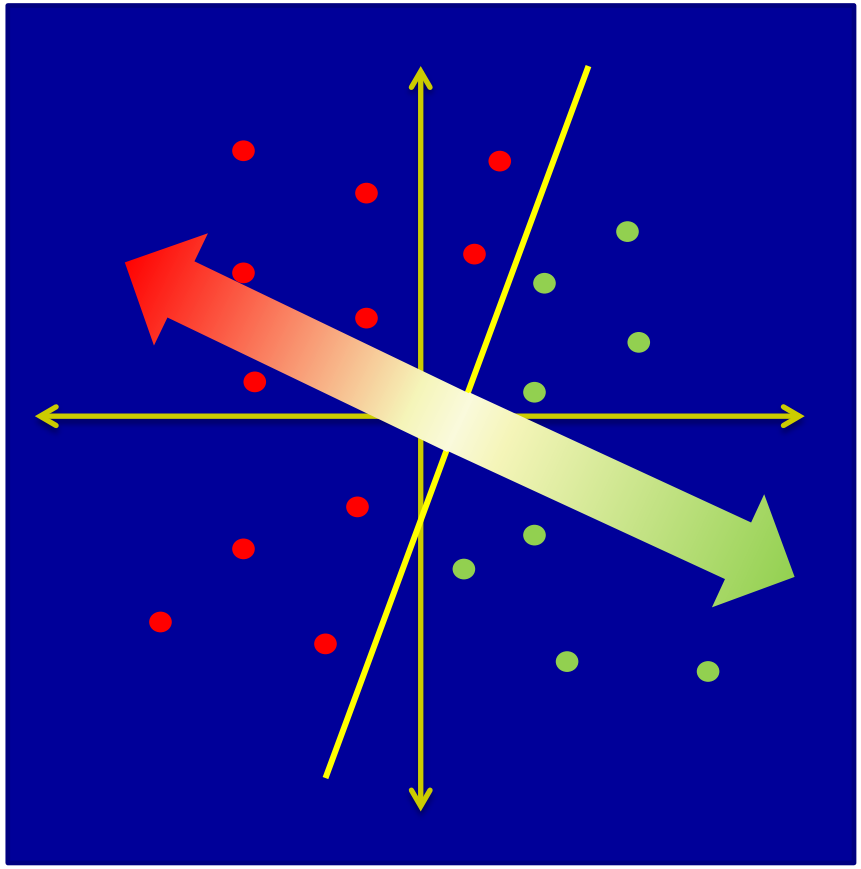
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



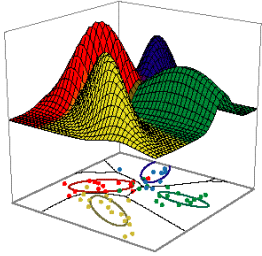
Active Learning



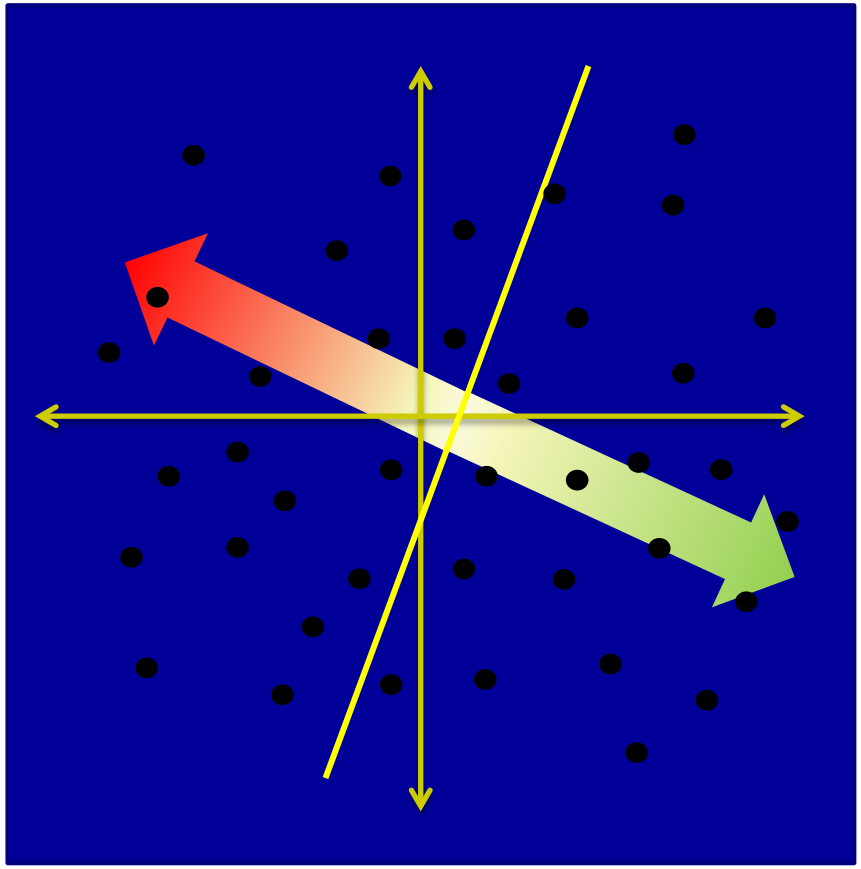
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



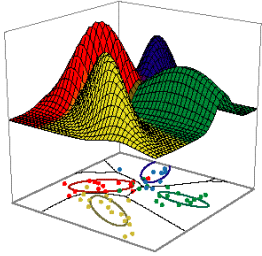
Active Learning



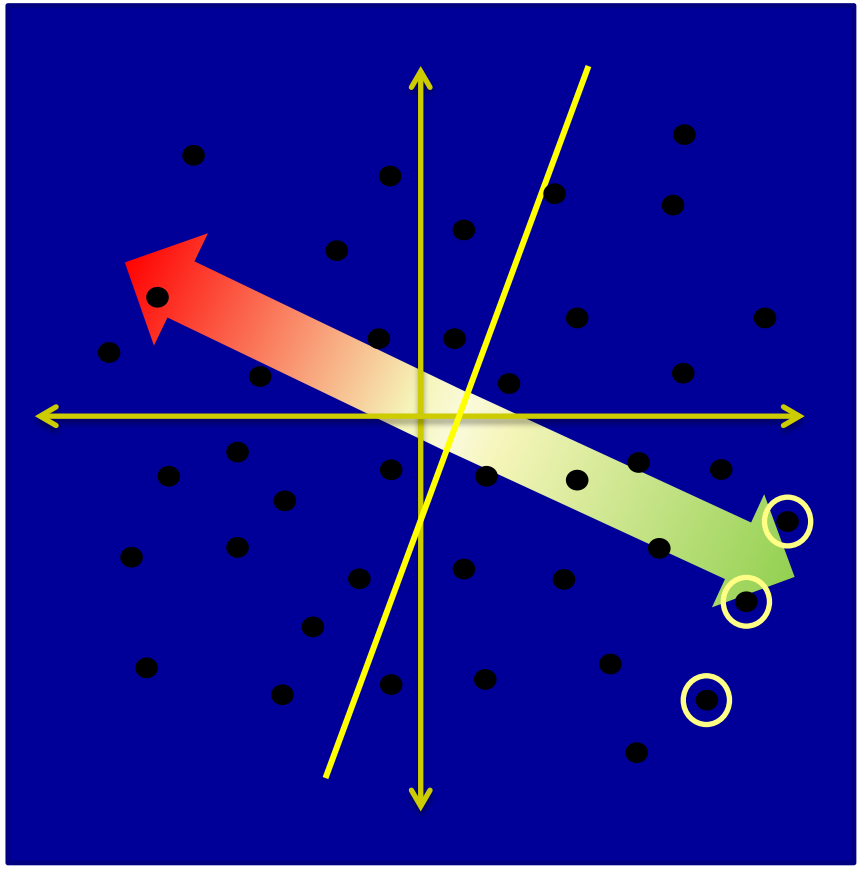
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



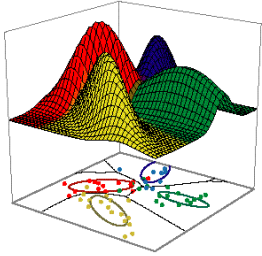
Active Learning



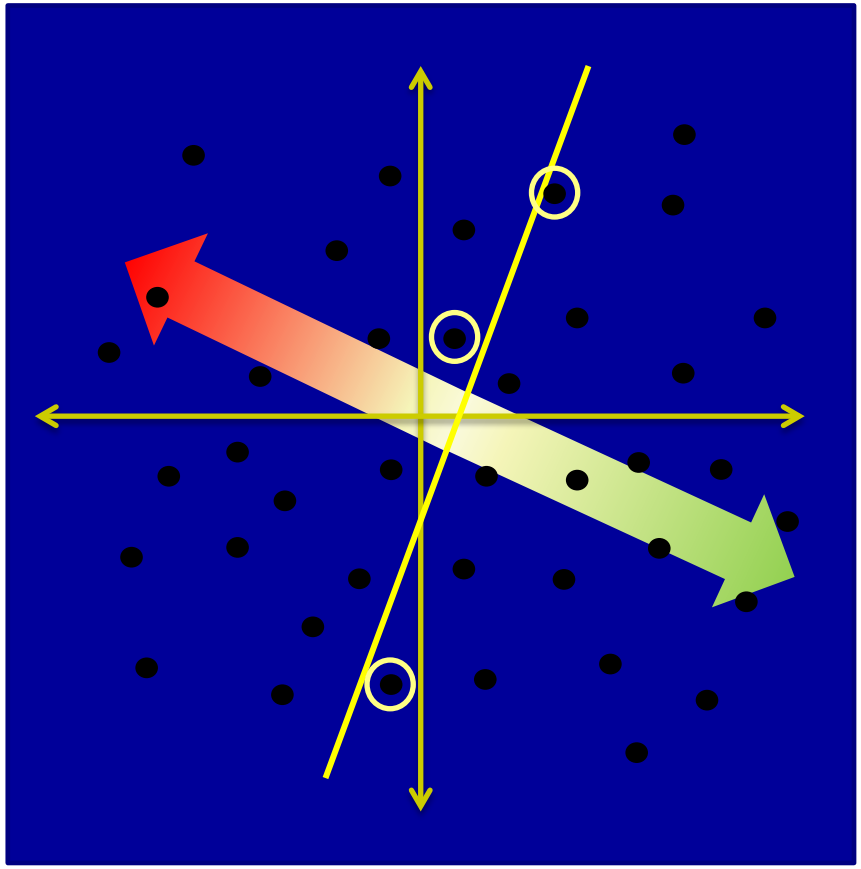
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



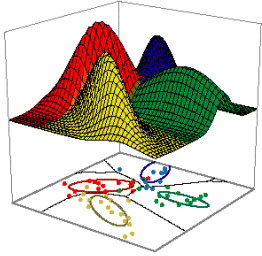
Active Learning



1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



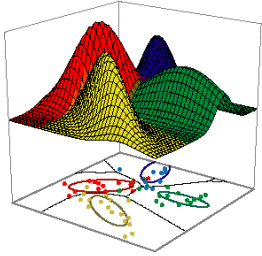
Active Learning



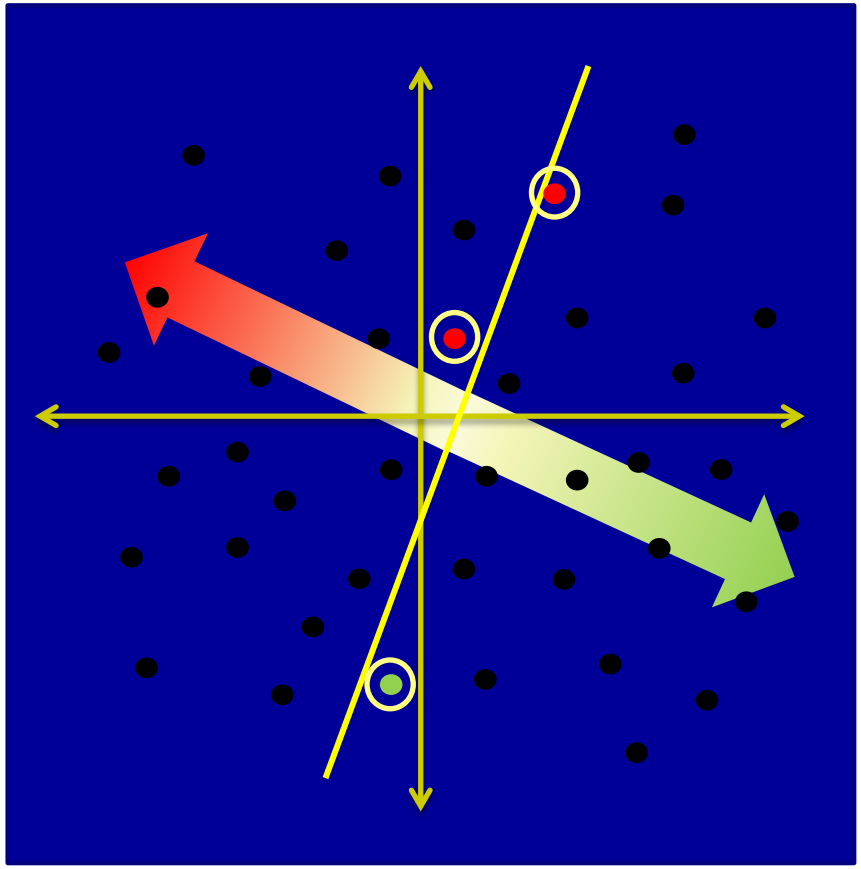
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*

?

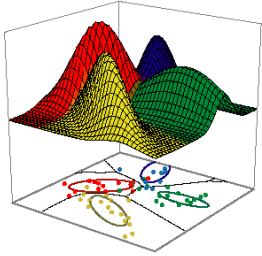
Active Learning



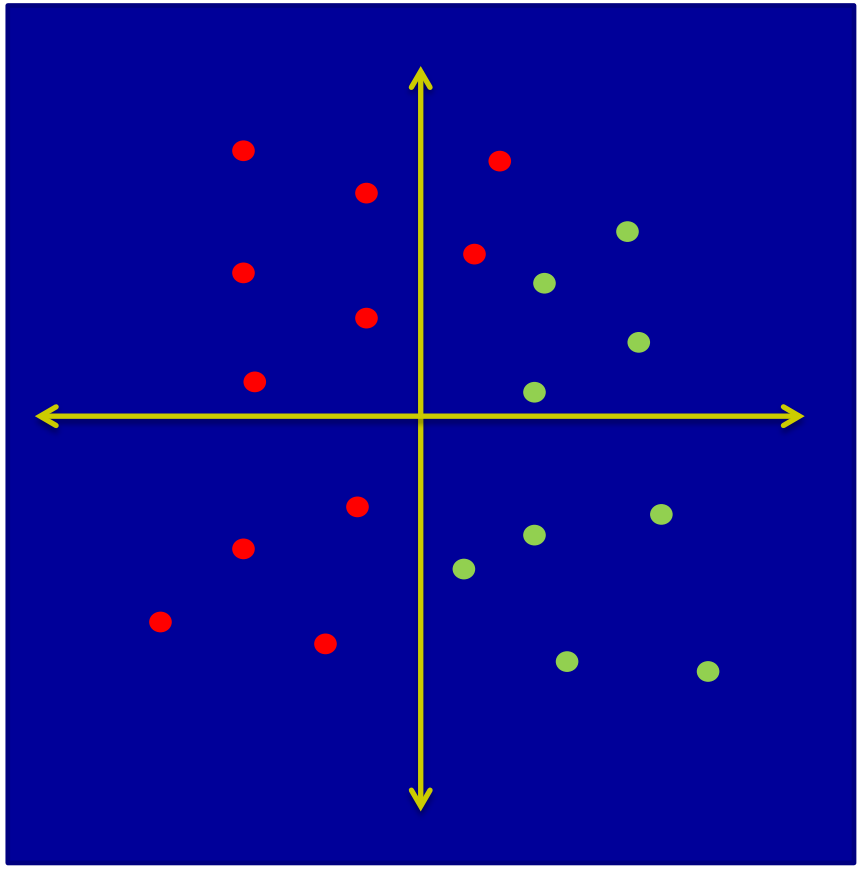
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



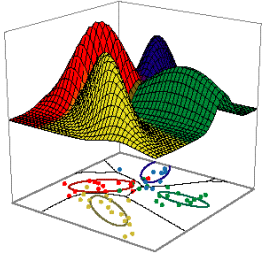
Active Learning



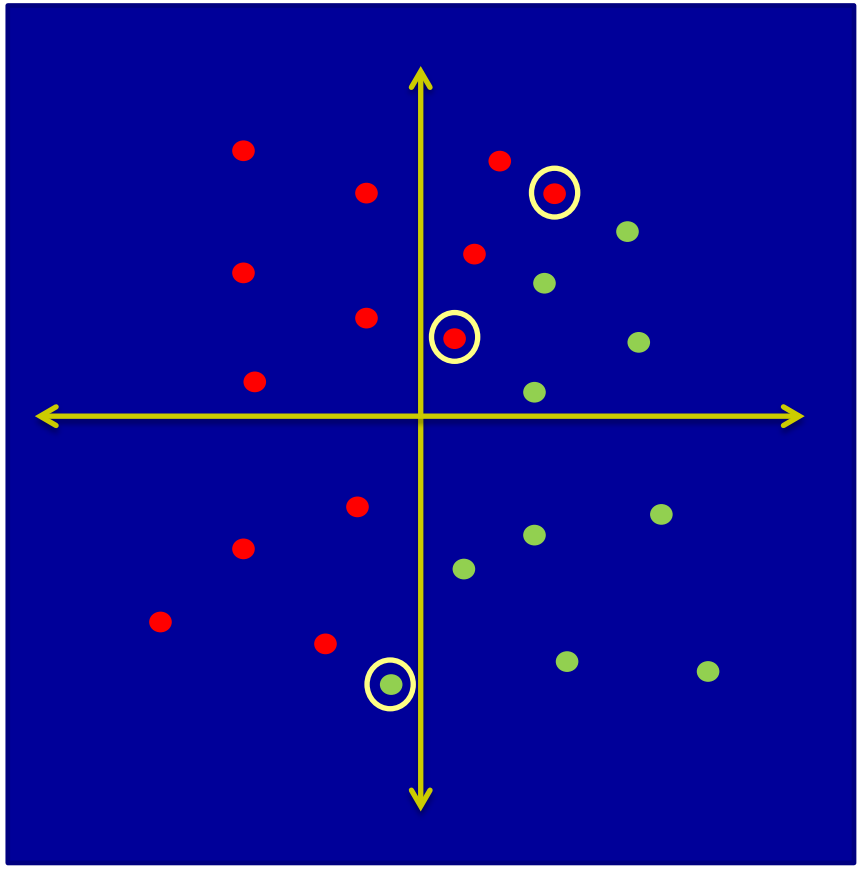
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



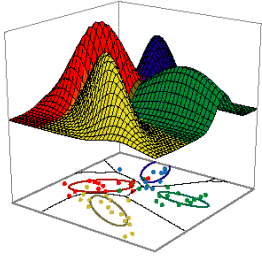
Active Learning



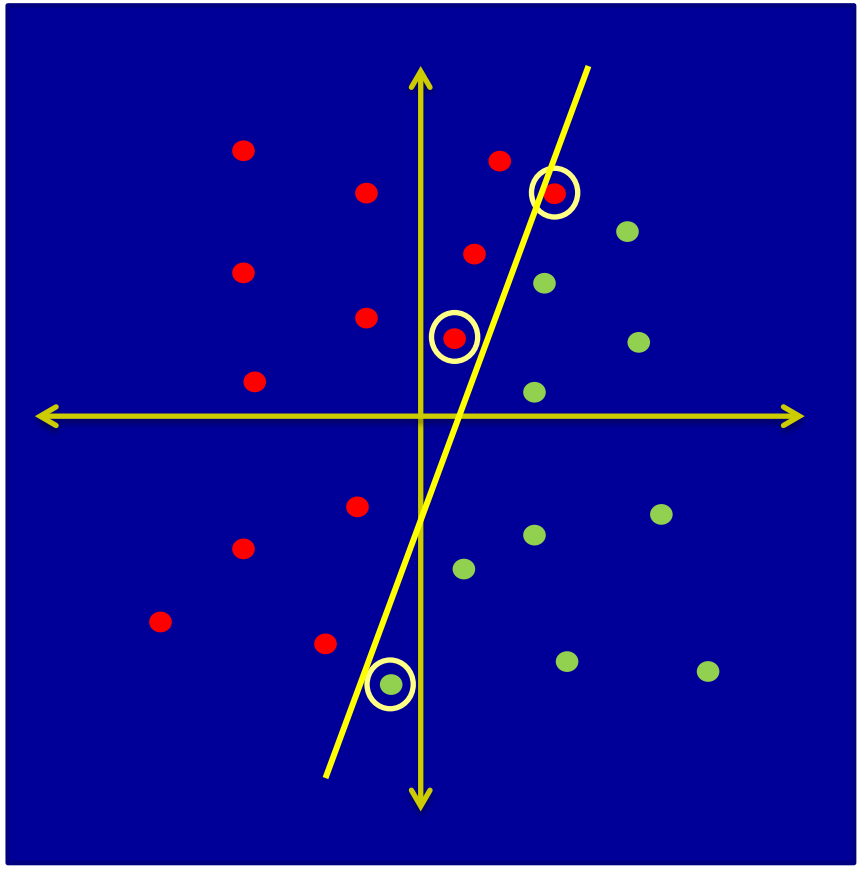
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



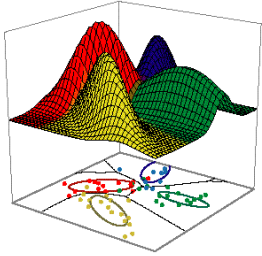
Active Learning



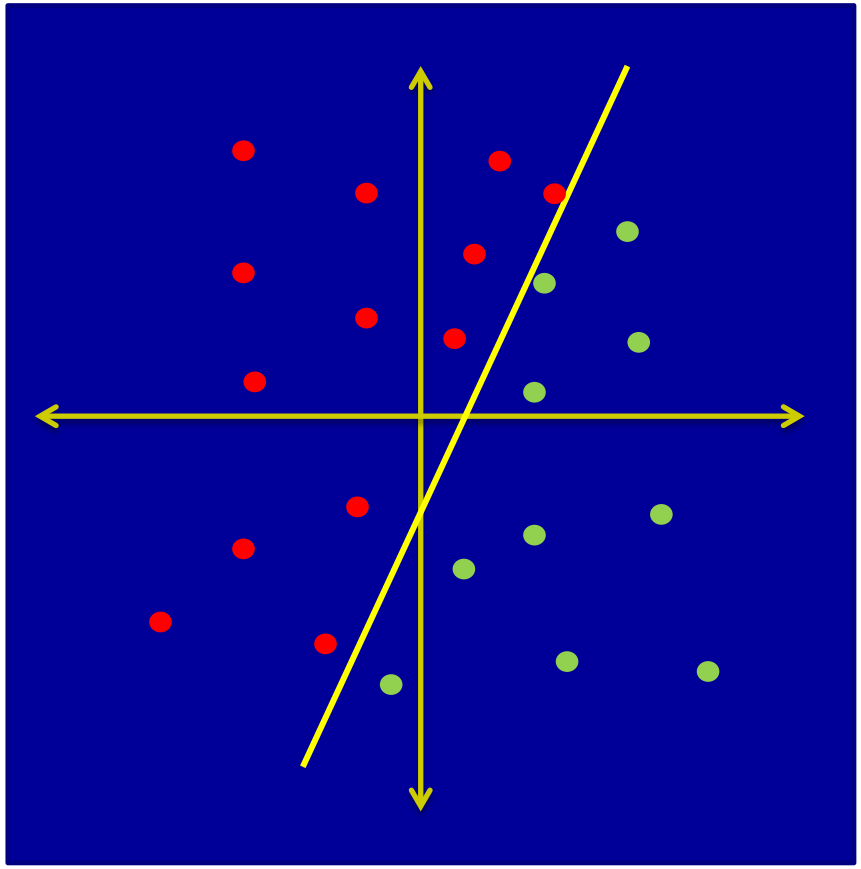
1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*



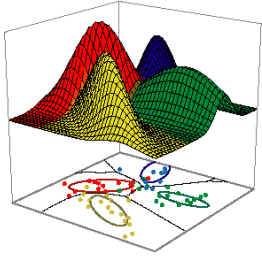
Active Learning



1. *Collect labelled training data*
2. *Train a classifier*
3. *Apply to unlabelled data*
4. *Select points to validate*
5. *Perform wetlab validation*
6. *Add newly labelled samples to training data*
7. *Retrain classifer*

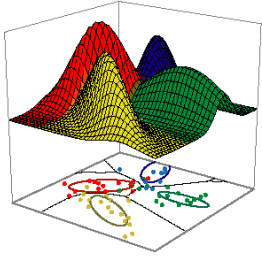


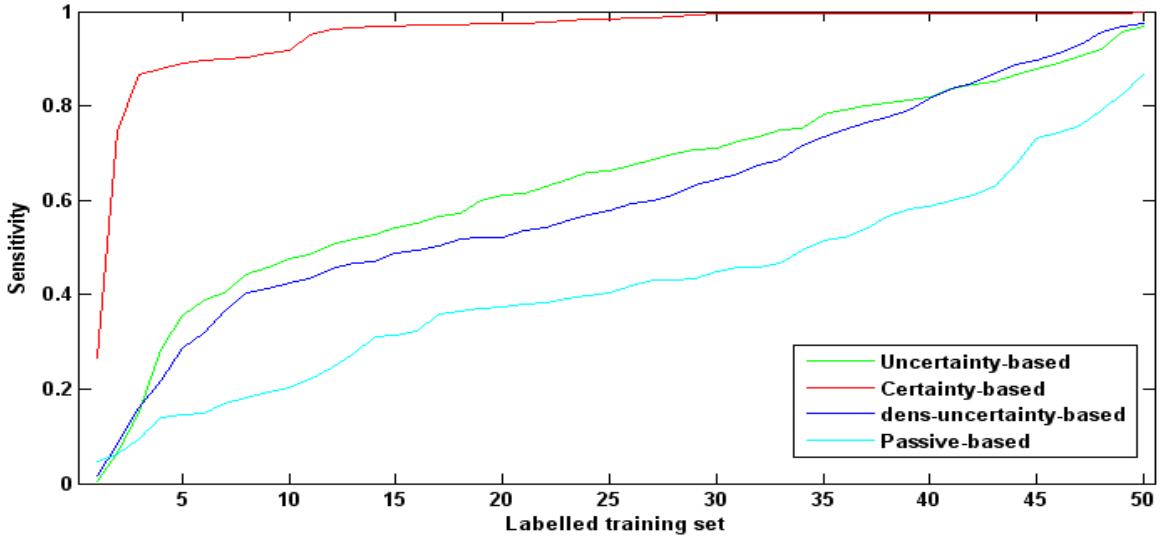
Simulating Active Learning

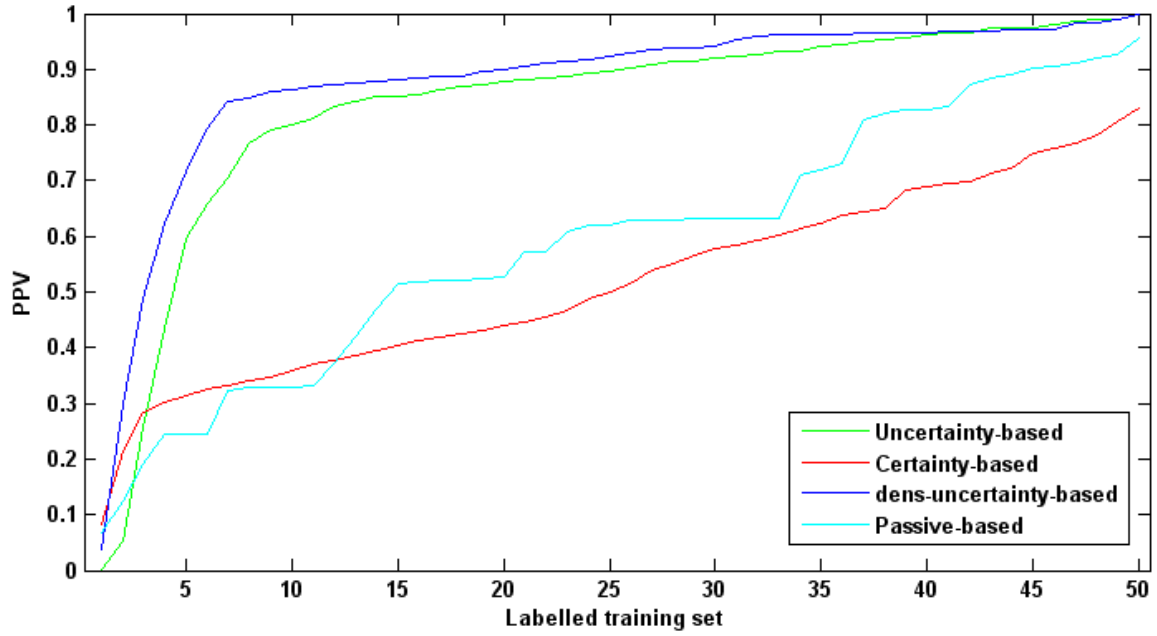


- Simulate guiding PTM validation experiments:
  - Have 51 positives and ~1300 negatives with known class
  - Simulate process of active learning:
    - Create hold-out test set (~25% of data)
    - Pretend that we initially only know the true class of 3 postives, 3 negatives
    - Train classifier, apply to remaining unlabelled data
    - Decide which datum to “validate” next, add to training set, retrain, repeat
    - Track performance vs. # validation experiments
- Simulated 4 strategies for selecting next point to validate (x_n)
  - Random: baseline
    - Select x_n randomly from remaining unlabelled data
  - Confidence-based: experimenter bias
    - Select x_n as point most likely to be positive
  - Uncertainty-based: active learning
    - Select x_n as point whose predicted class is least certain (closest to boundary)
  - Density-uncertainty-based: active learning with initial set
    - Same as above, but select initial 3 points such that they are broadly representative of the data (typical of 6 areas of high density)

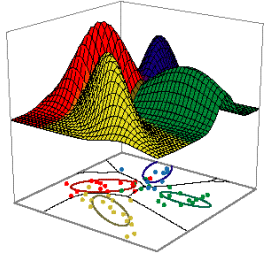
Simulating Active Learning





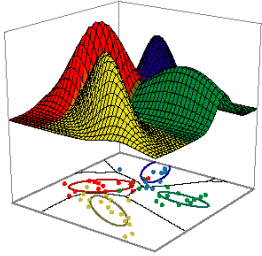


Co-training



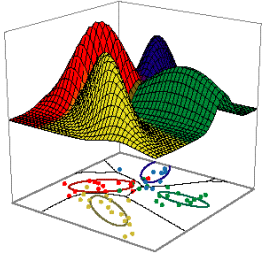
- Assume you have small labelled set, large unlabelled set and no oracle
- Can use co-training (Blum and Mitchell, 1998)
  - Create two disjoint sets of attributes or views to train two separate classifiers on initial labeled data.
    - Can use attribute clustering to select attribute sets
  - Then, alternately each classier classifies the unlabeled data and adds a few of the most confidently labeled samples to the training set.
  - Classifiers are then trained again and the process repeated for a number of times until a stop criterion is met.
  - Semi-supervised approach to multi-view ML

Combining Classifiers (9.7)

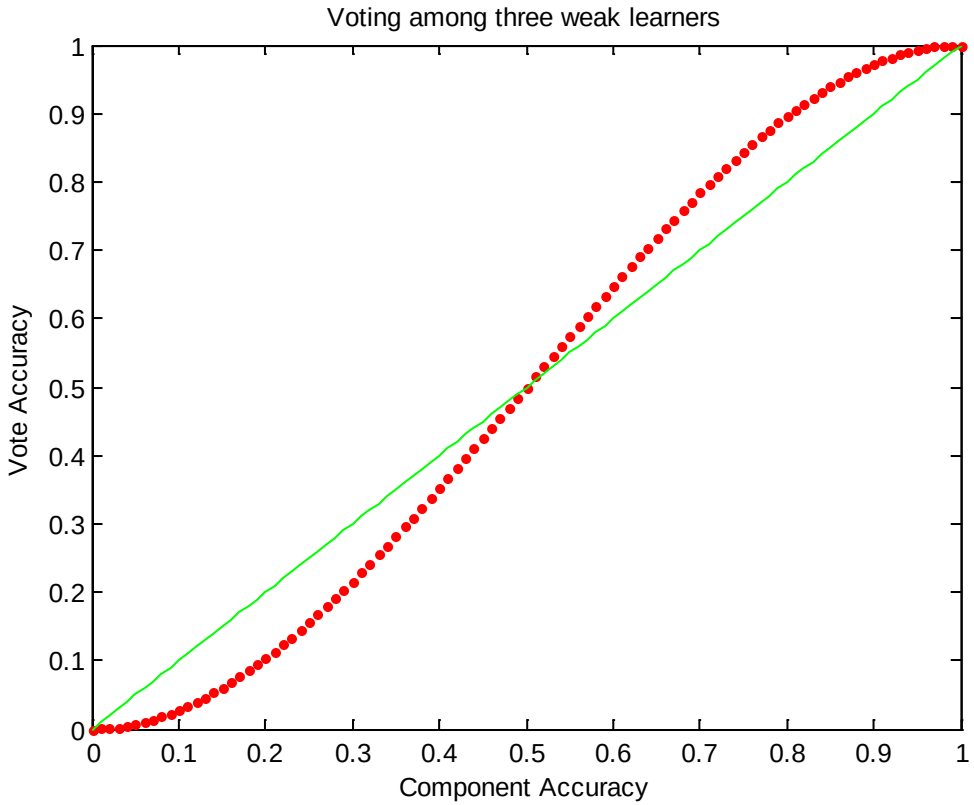


- As long as each component classifier is better than random, adding component classifiers should improve overall accuracy
- Particularly useful when each component classifier is an ‘expert’ in a different region of feature space
- How to combine to achieve a single decision?
  - Simplest approach: voting, weighted voting
    - Natural way to define confidence
  - Can use probabilistic approach
    - Useful when individual component classifier outputs are probability estimates
      - (i.e. discriminant functions which sum to 1)
  - Cascaded classifiers

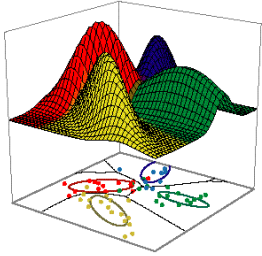
Voting



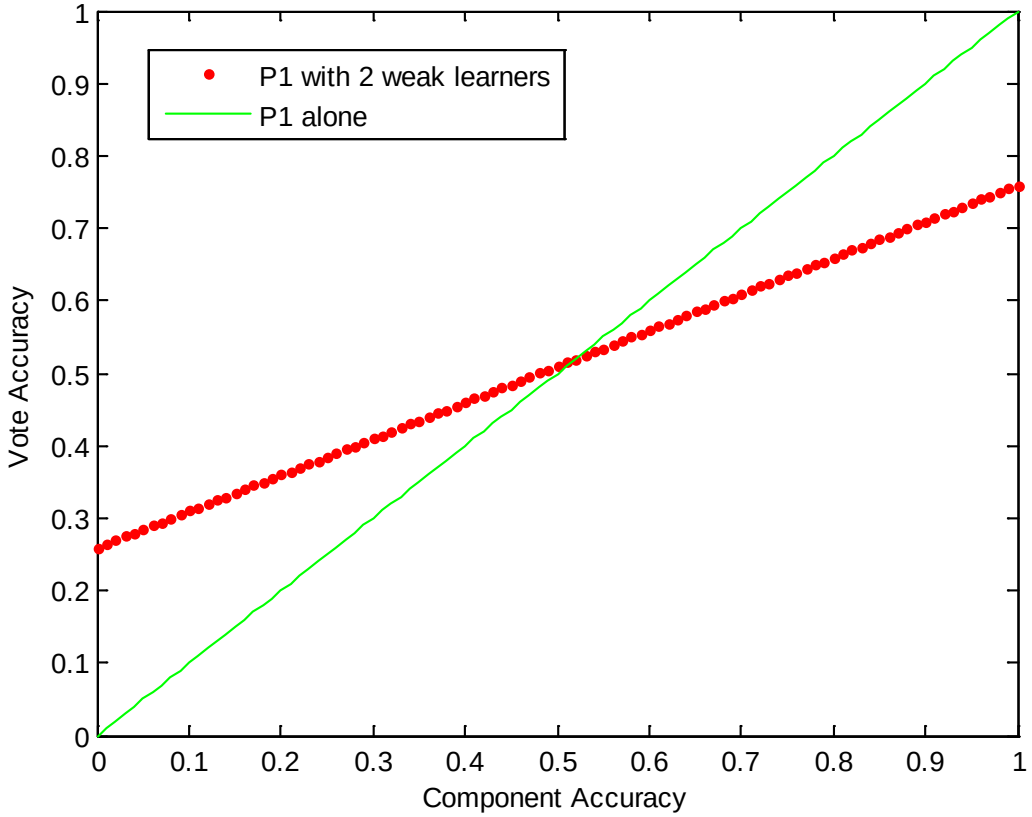
- Here combining three weak learners, where accuracy varies from 0 to 1.
  - Correct when all 3 component classifiers are correct, or when 2 of 3 are correct.



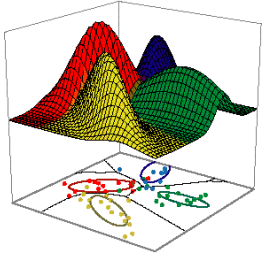
Voting



- Here combining three weak learners, where accuracy of C1 varies from 0 to 1, while accuracy of C2,C3 fixed at 0.51.



Mixture of experts architecture



Final decision:
1) average weighted g_{rj} over r=1:k, choose class j with greatest g
2) winner-takes-all: choose single greatest g_{rj} (no averaging)

w_r scales all g_{rj}

each classifier supplies c discriminant functions sum(g_{rj})=1, j=1..c

Can train entire system using gradient descent or EM

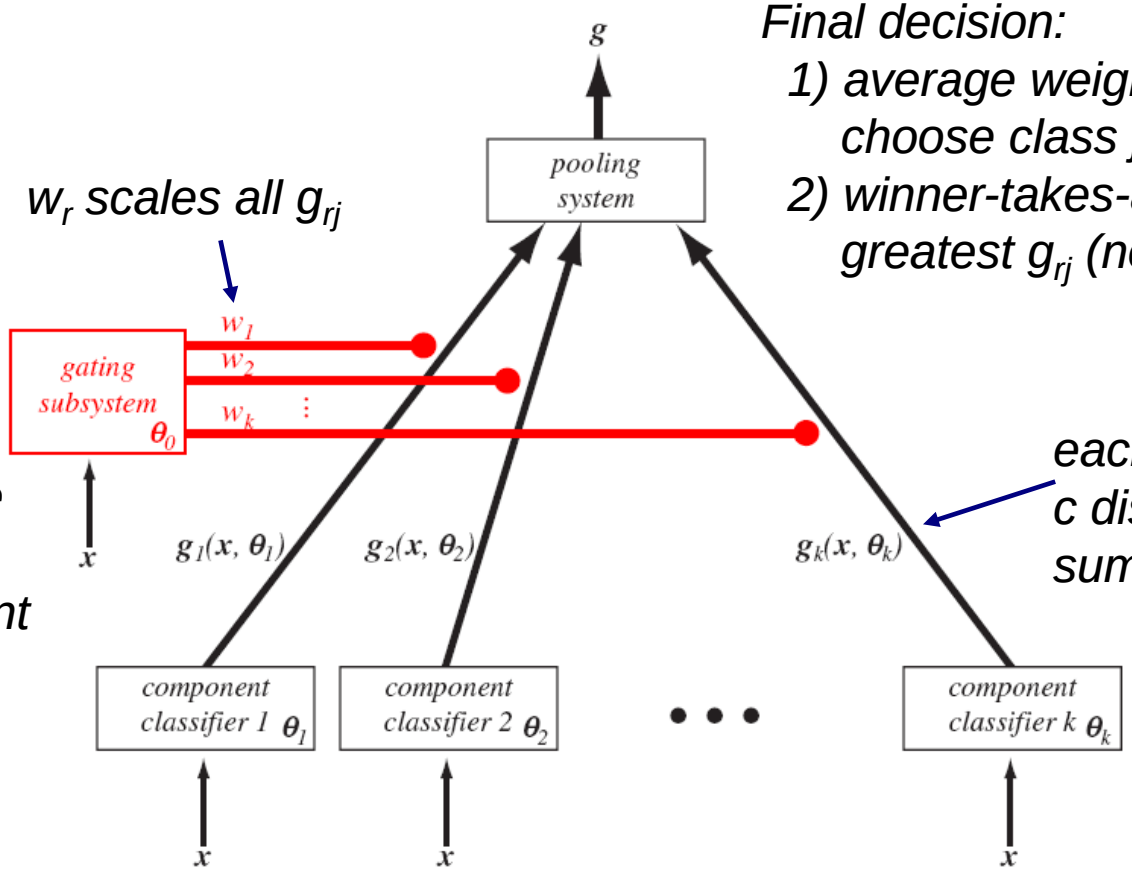
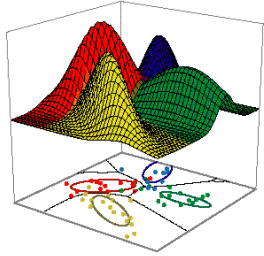


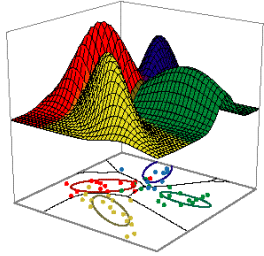
FIGURE 9.19. The mixture-of-experts architecture consists of k component classifiers or "experts," each of which has trainable parameters $\boldsymbol{\theta}_i$, $i = 1, \ldots, k$. For each input pattern $\mathbf{x}$, each component classifier i gives estimates of the category membership $g_{ir} = P(\omega_r|\mathbf{x}, \boldsymbol{\theta}_i)$. The outputs are weighted by the gating subsystem governed by parameter vector $\boldsymbol{\theta}_0$ and are pooled for ultimate classification. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Component classifiers without discriminant functions



- When component classifiers do not supply discriminant functions (which sum to 1), we can use transformations to create proper discriminants, g_i:
- Component classifiers may supply only h_i:
  - 1) Analog outputs (e.g. ANN)
    - Use softmax transformation: $g_i = \frac{e^{h_i}}{\sum_{j=1}^{c} e^{h_i}}$
  - 2) Rank order outputs (e.g. k-NN)
    - Assume rank order is linearly proportional to discriminant function. Resulting g_i should be normalized.
  - 3) One-of-c output (e.g. decision tree)
    - Assign g_i=1.0 where h_i=1, all other g_j=0

Component classifiers without discriminant functions



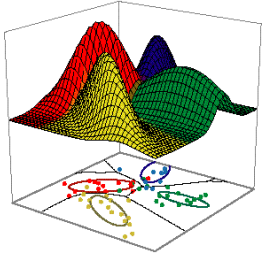
- Example:

Analog	
h_i	g_i
0.4	0.158
0.6	0.193
0.9	0.260
0.3	0.143
0.2	0.129
0.1	0.111

Rank order	
h_i	g_i
3rd	4/21=0.194
6th	1/21=0.048
5th	2/21=0.095
1st	6/21=0.286
2nd	5/21=0.238
4th	3/21=0.143

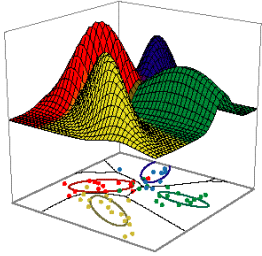
One-of-c	
h_i	g_i
0	0.0
1	1.0
0	0.0
0	0.0
0	0.0
0	0.0

Cascaded Classifiers

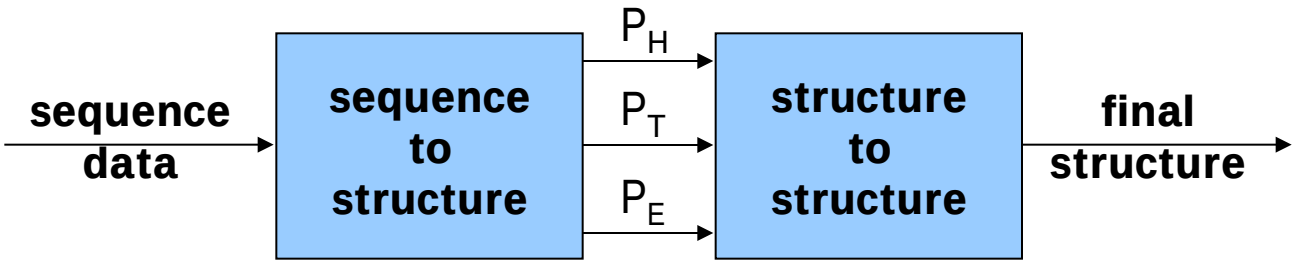


- Can use a classifier to combine the outputs from component classifiers
- Case study:
  - Using parallel cascade identification for predicting protein secondary structure
  - First developed a ‘sequence-to-structure’ classifier that predicted structure from an input window of sequence
  - Then added a cascaded classifier to refine structural prediction by examining a window of predicted structure
  - Then added a cascaded classifier to combine a PCI-based classifier with an ANN-based classifier

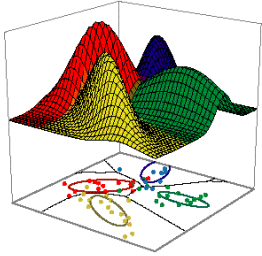
Cascaded classifiers



- First classifier examines a window of sequence and predicts structure of central residue
- Second classifier examines window of predicted structure, and refines structure of central residue

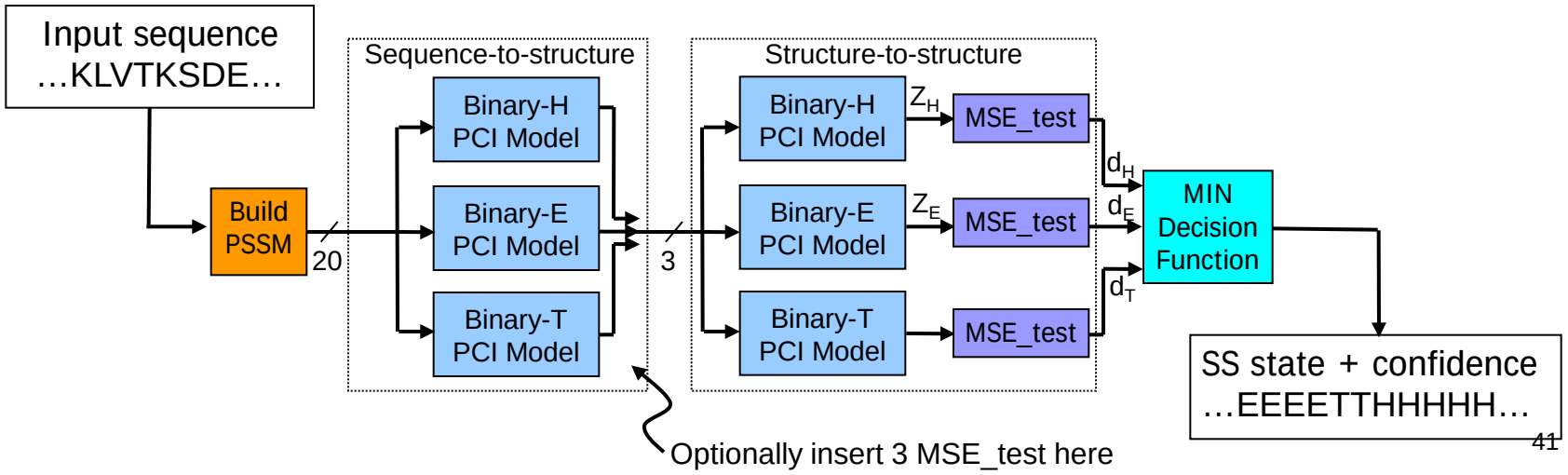


Cascaded Classifiers

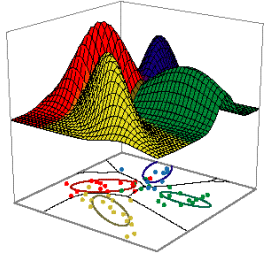


1. Train 3 binary 20-input PCI models on S1 subset (543 proteins)
2. Apply models back to S1 to obtain 3 output vectors
3. Optimize 3 binary 3-input postPCI models (with MSE-test) over S1
4. Train both layers over entire antiTest dataset (2170 chains)
5. Test both layers over S5 test subset (543 chains)

- Try both raw model outputs (PCI-raw) and PCI-MSE outputs



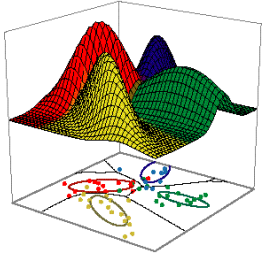
Cascaded Classifiers



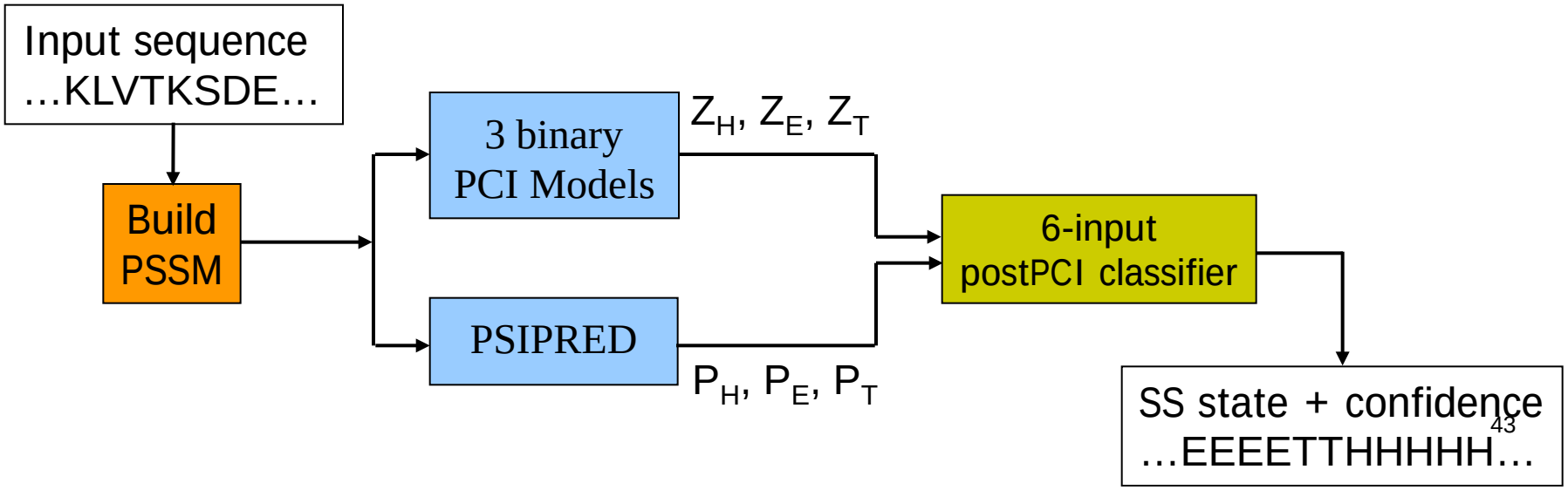
- Results over S5 test subset (543 proteins)
  - 2% increase in accuracy (zero-padding used)
  - Accuracy approaching top contemporary methods

	CC_H	CC_E	CC_T	Q_3	BAD
No post-PCI	0.664	0.579	0.519	73.6	2.43
post-PCI using PCI raw outputs	0.694	0.595	0.548	75.5	1.90
post-PCI using PCI-MSE scores	0.692	0.596	0.548	75.6	1.84

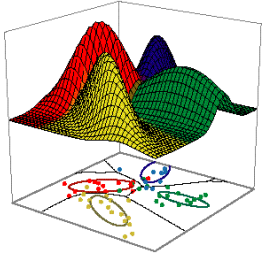
Combination of PSIPRED and PCI



- PSIPRED (Jones 1999):
  - Uses 2 cascaded 3-layer ANN, PSSM input
  - Ranked first in last 2 CAFASP experiments
- Strategy:
  - Combine 3 PSIPRED ANN outputs with 3 raw PCI model outputs or 3 PCI-MSE outputs



Combination of PCI and PSIPRED



- Train PCI on S1-S4, evaluate on S5
- Increase Q3 over PSIPRED alone
- 25% reduction in BAD score

	CC_H	CC_E	CC_T	Q_3	BAD
PSIPRED alone	0.727	0.646	0.585	77.8	1.49
PCI & PSIPRED	0.740	0.647	0.592	78.5	1.12