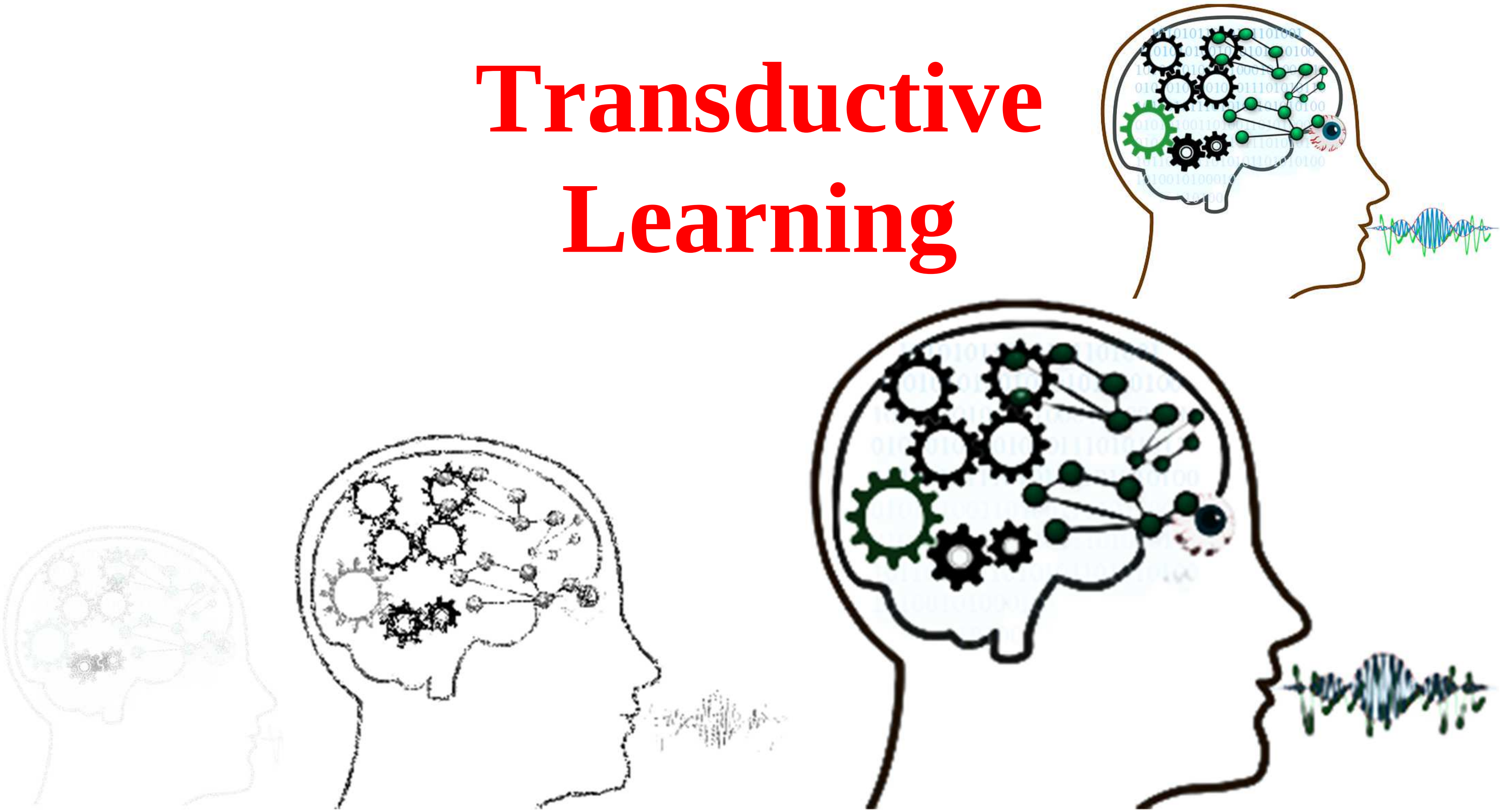
Transductive Learning



Graduate Program in Software
SEIS 763: Machine Learning
Dr. Chih Lai

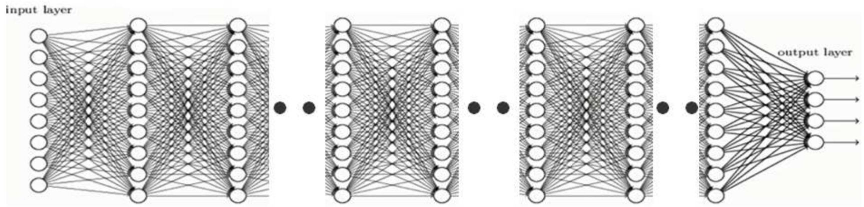
Large Amount of Unlabeled Data

- or, large amount of claim data, but…   no label on fraud or normal

Ws =???

	Age	Gender	Height	Weight	Smoker	HealthStatus	Cancer
1 Smith	38	'Male'	71	176	1	'Excellent'	1
2 Johnson	43	'Male'	69	163	0	'Fair'	0
3 Williams	38	'Female'	64	131	0	'Good'	
4 Jones	40	'Female'	67	133	0	'Fair'	
5 Brown	49	'Female'	64	119	0	'Good'	
6 Davis	46	'Female'	68	142	0	'Good'	
7 Miller	33	'Female'	64	142	1	'Good'	
8 Wilson	40	'Male'	68	180	0	'Good'	
9 Moore	28	'Male'	68	183	0	'Excellent'	
10 Taylor	31	'Female'	66	132	0	'Excellent'	

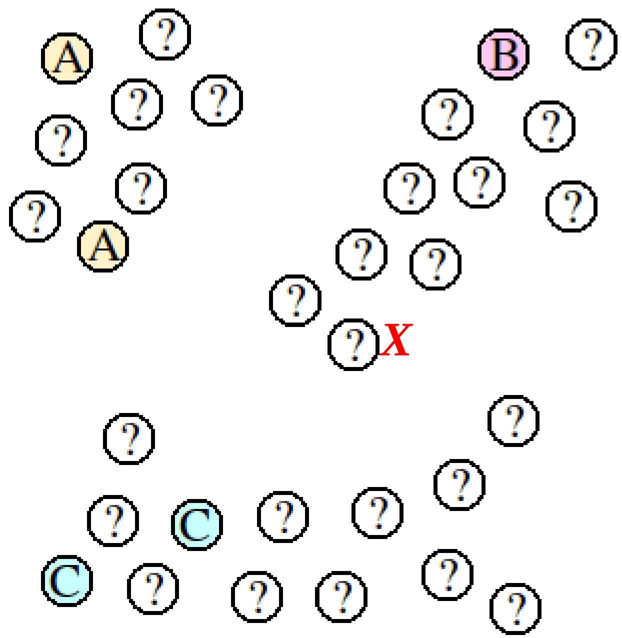
How about Super-Deep NN??



Transductive Learning, <u>**Unsupervised**</u> Learning

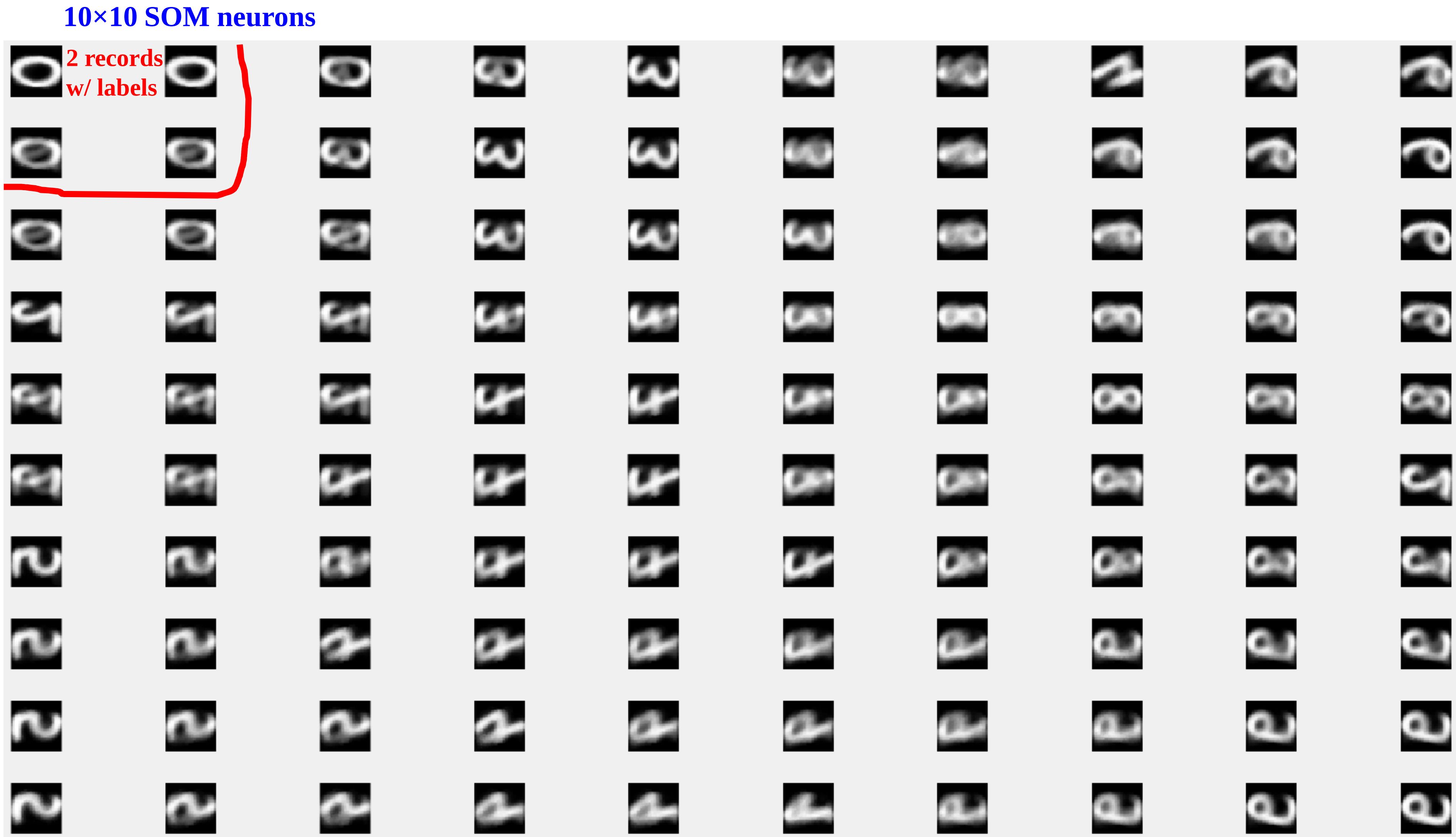
- Supervised learning (general classification) methods won't work➔ too few labeled points.

- Unsupervised learning is one alternative.
  - Clustering data points into ***k*** **(???)** clusters.
  - <u>**Hopefully**</u> point ***X*** is clustered in the group w/ known "**B**" so it will be predicted as **B**.
  - Depends on (1) which cluster method used, and (2) what ***k*** we choose.



https://en.wikipedia.org/wiki/Transduction_(machine_learning)

Self-Organizing Map, **Unsupervised** Learning

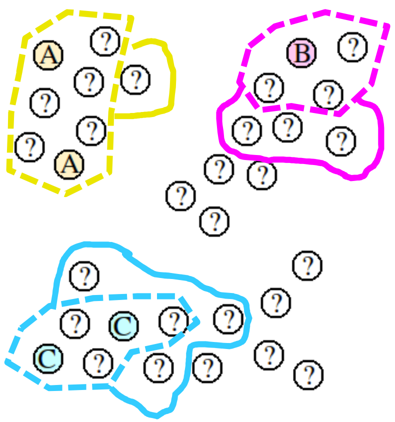


Transductive Learning, <u>**Semi-Supervised**</u> Learning

- How to use ***big*** unlabeled data w/ ***small*** labeled data to improve accuracy?
  - Hard to get fully <u>labeled</u> data.   <u>Unlabeled</u> data is cheaper & easier to get.
  - Train labeled data first, and <u>**then**</u> classify unlabeled data!!
    - Train an initial model ***f*** w/ supervised learning on <u>**labeled**</u> data ***L***.
    - Use ***f*** to predict <u>**unlabeled**</u> data ***Ux***.
    - High confident results are moved from ***U*** to ***L*** w/ class predictions.
    - Retain ***f*** w/ newly augmented ***L***.
    - Assumption: records moved to ***L*** are reliable enough.
    - This approach can work w/ <u>any learning algorithm</u>.

M1	***L1***	***U1***
M2	***L2***	***U2***
M3	***L3***	***U3***

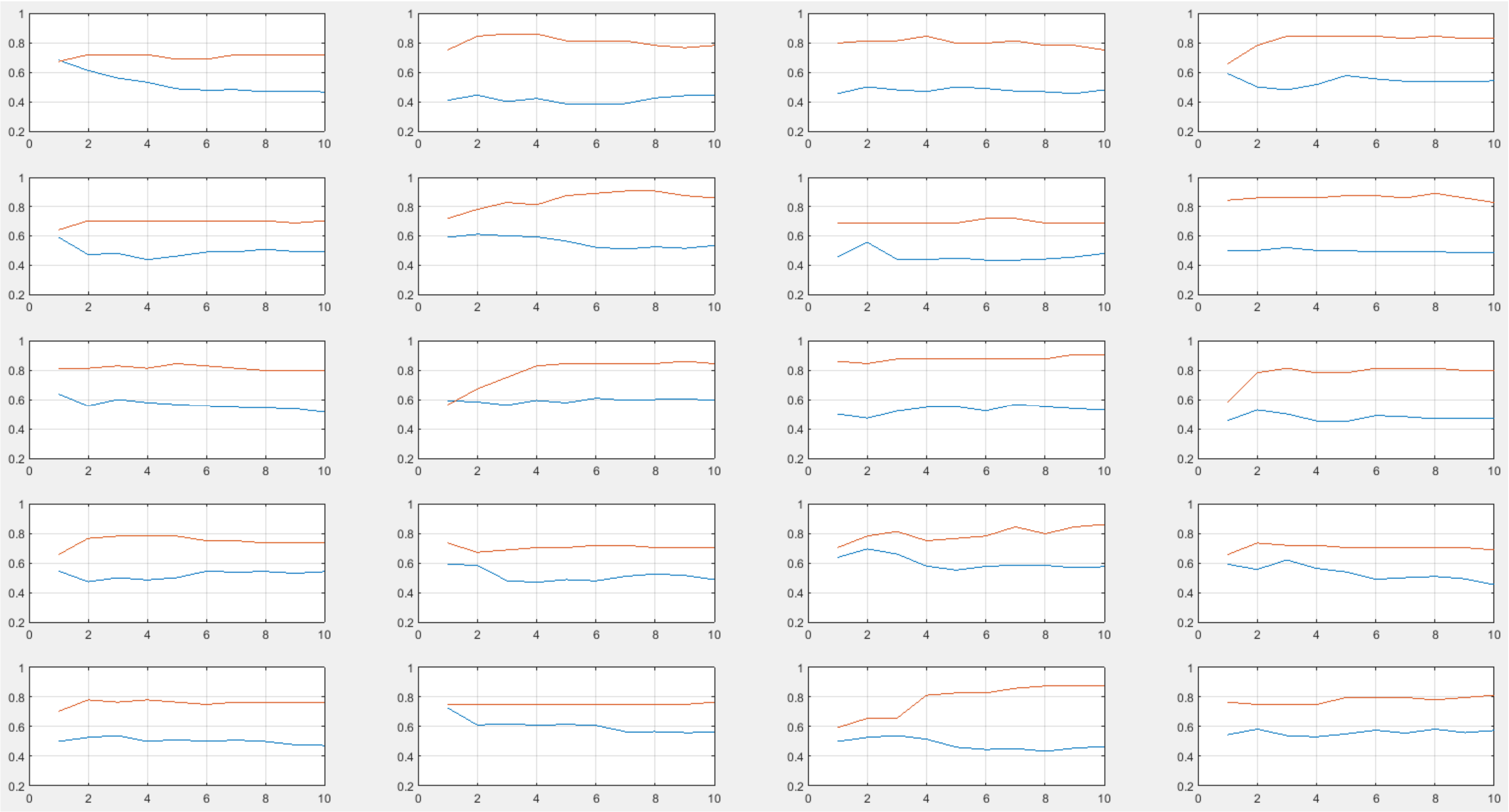
- "***transductive learning***"
  - Infer **correct(??)** labels of the unlabeled data.



- **Any other way???**

Transductive Learning

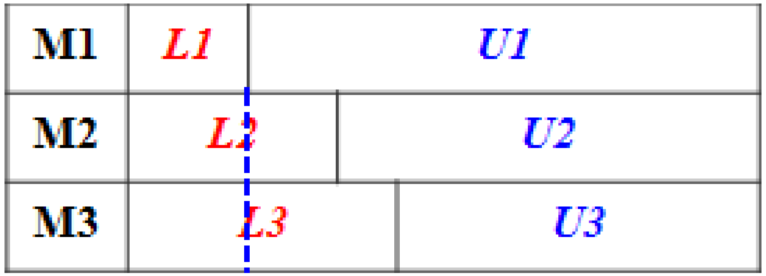
- Try transductive learning 20 times on ovarian cancer dataset: 216 records w/ 4000 features.
  - 64 test records, and 152 candidate records w/ initially only 8 records have known labels.
  - Keep “guessing” more training labels over 10 iterations, and rebuild next model.
  - Apply the new model to the test data.
  - Blue line = training accuracy (avg -4.5%).
  - Red line = test accuracy (avg +7.8%) ➔ models built from transductive learning is getting better.

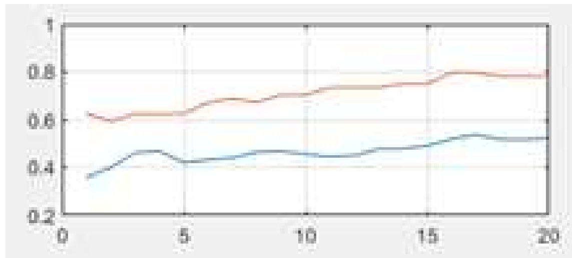


Transductive Learning, Reset Training

- Train labeled data first, and **[then]** classify unlabeled data!!
  - Train an initial model ***f*** w/ supervised learning on **labeled** data ***L***.
  - Use ***f*** to predict all the **originally unlabeled data** ***U1***.
  - High confident results are moved from ***U*** to ***L*** w/ class predictions.
  - Retain ***f*** w/ newly augmented ***L***.
  - Assumption: records moved to ***L*** are reliable enough.
  - This approach can work w/ any learning algorithm.

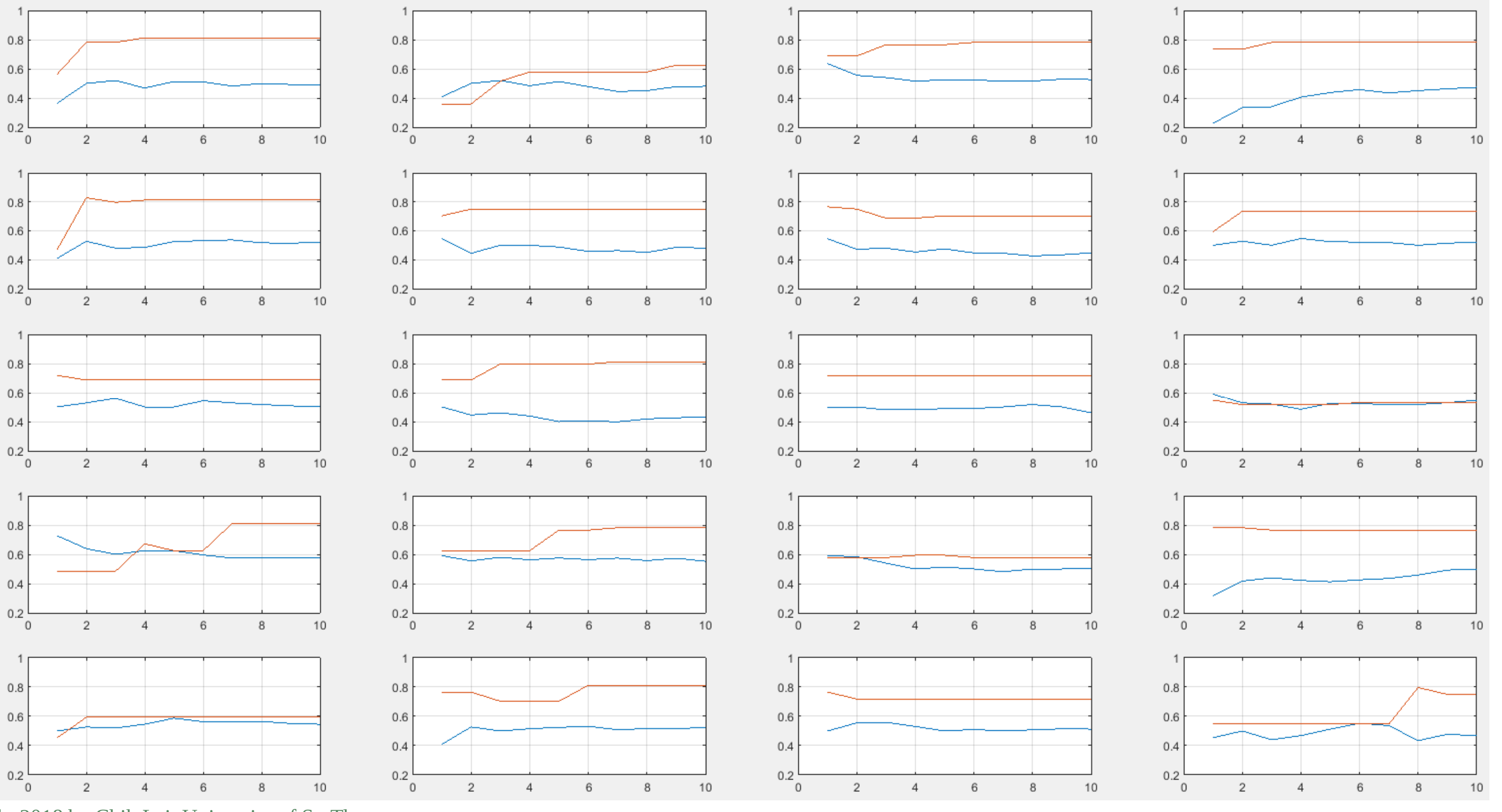
- "***transductive learning***"
  - Infer correct labels of the unlabeled data.





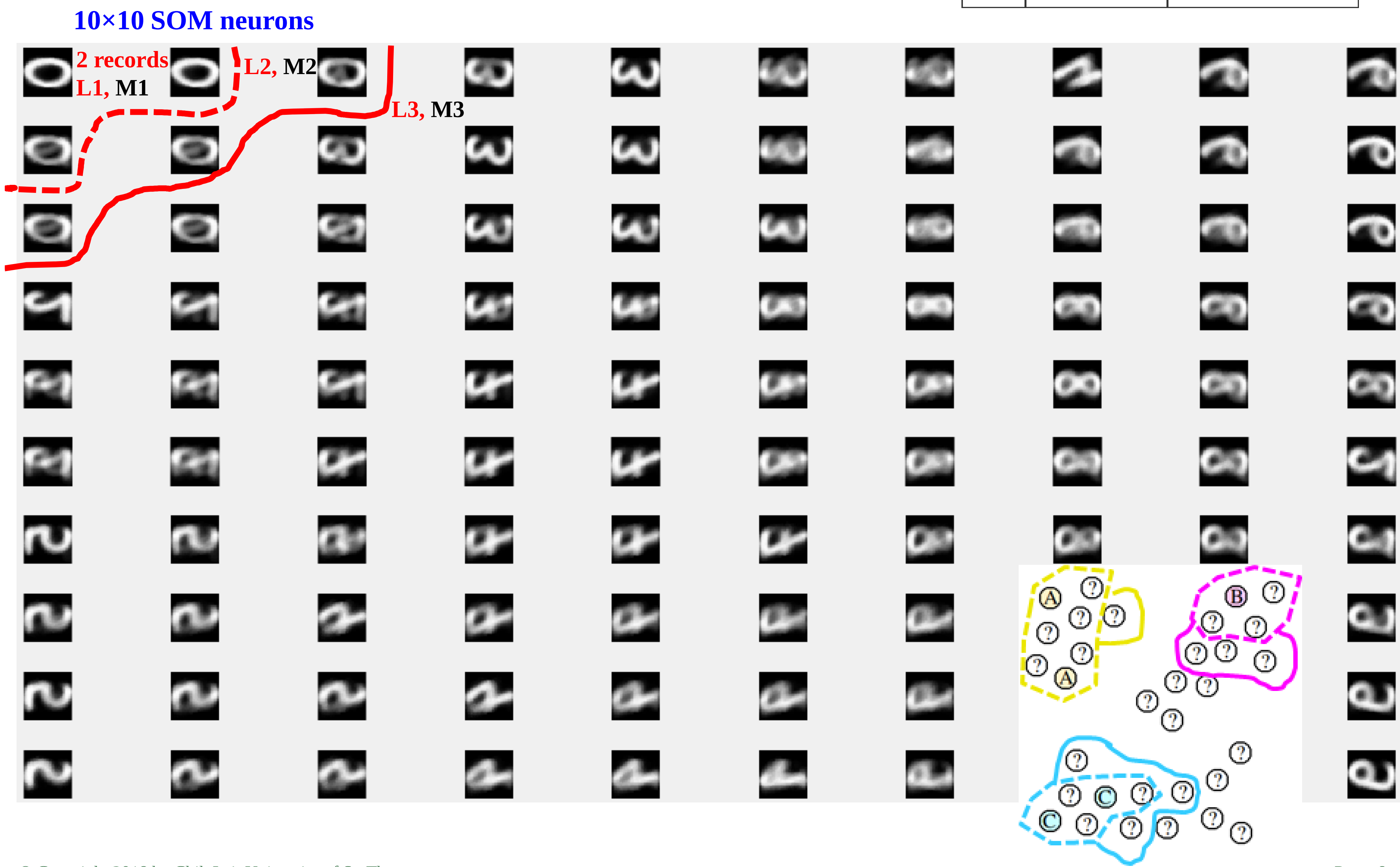
Transductive Learning with … Reset Training

- Try transductive learning 20 times on ovarian cancer dataset: 216 records w/ 4000 features.
  - 64 test records, and 152 candidate records w/ initially only 8 records have known labels.
  - Keep "guessing" more training labels over 10 iterations, and rebuild next model.
  - Apply the new model to the test data.
  - Blue line = training accuracy (avg -1.2%).
  - Red line = test accuracy (avg +10.1%) ➔ models built from transductive learning is getting better.



SOM, Transductive Learning

M1	*L1*	*U1*
M2	*L2*	*U2*
M3	*L3*	*U3*



Ensemble

Graduate Program in Software
SEIS 763: Machine Learning
Dr. Chih Lai

Ensembles

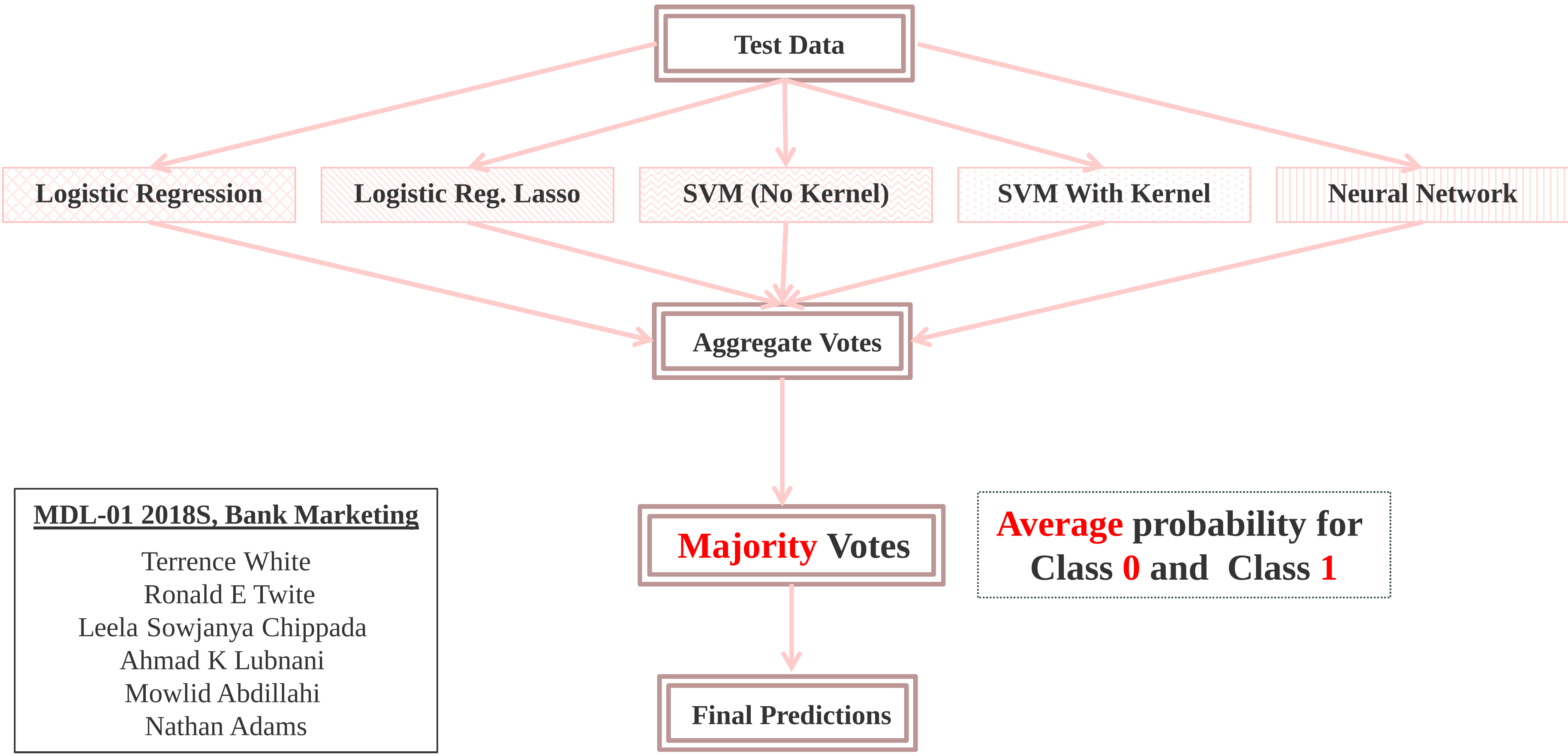
- Combine some weak learners to create one better learner.
  - Wisdom of crowd?!
  - i.e. work yourself vs. ask friends…
  - If we don’t know what kinds of models are best for your prediction problems.
  - More models can be better than a single one.
  - May include same type of model (***bagging*** & ***boosting***) or different types of models.

- Bagging / boosting includes many instances of the same type of model.
  - In regressions, take unweighted / weighted average of predictions.
  - In classifications, take majority vote of predictions or weighted combinations of predictions.

- Weighted method… (***a model of models***)

Ensemble Hard / Soft Voting

- But, the # of models is limited by known # of machine learning methods???
- Can we have more # of models in an ensemble???   i.e. 500+???
- Just repeatedly build ***N*** # of models under each method???   What's wrong???



MDL-01 2018S, Bank Marketing

Terrence White
Ronald E Twite
Leela Sowjanya Chippada
Ahmad K Lubnani
Mowlid Abdillahi
Nathan Adams

Ensembles Methods

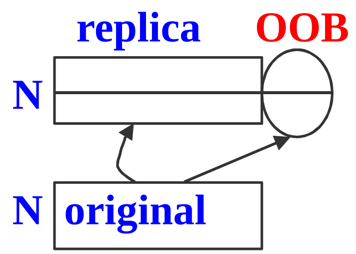
- Different approaches to combine several models into a better one.

- Bagging (**B**ootstrap **Agg**regation)
  - Take bootstrap samples w/ ***replacement*** from data to train **many** weak learners.
  - To decrease variance by selecting training data from original dataset w/ repetitions.
  - May not improve **bias**, but decrease **variance**.
  - Two types of baggings: “data bagging” & "feature bagging“ (random forest TreeBagger()).
    - Combine **both** baggings???

- Boosting
  - Misclassified instances by earlier learners are given more weight.
  - Subsequent learners give more focus to the previously misclassified data.
  - Yield better accuracy than bagging, but also tends to be overfitted.
  - Uses the same data to train each learner sequentially.
  - Most common method **Adaboost**.

"Bagging" = ***B***ootstrap ***Ag***gregation

- Steps…
  - Bootstrapping→ Resample training data several times into ***replica***
  - Build a model from each ***replica***
  - Final predictions = majority classes predicted by Bagger Trees
  - ***Prediction score*** = % trees that classify a sample in each class

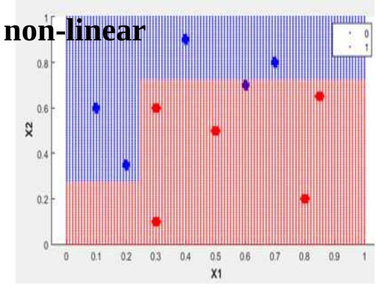


  - Drawing *N* out of *N* samples with replacement for building each tree
  - Every tree is grown on an independently drawn bootstrap ***replica*** of training data.
  - Roughly omits on average 37% (1/*e*) of samples for each decision tree
  - Samples not included in this replica are "**out of bag**" (**OOB**) for this tree.
  - Out-of-bag prediction average is an unbiased estimator of the ***ensemble***
  - **Similar** (**but not same**) to split the data into *training* and *test* subsets ≈ cross validation

"Bagging" Summary

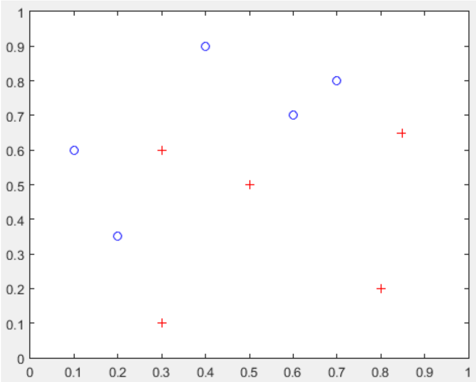
- Bagging steps:
  - Create a random subset of data by sampling w/ **replacement**.
  - Create many training sets (each has size ***n***) from original data size ***N***, ***n*** < ***N*** or ***n*** = ***N***.
  - Train each individual learner based on individual ***replica***.
- Test each learner against its OOB data.
  - For classification, take unweighted or weighted majority vote from each learner.
  - For regression, take average prediction from each learner.
- Bagging impact to overfitting??
  - By training different models using different datasets. (i.e. replica)
  - Each model use only part of data and miss some data (to avoid overfitting).
  - This avoids "overfitting" = A model "**memorize**" too much of training data.
  - Reduce model complexity.
  - Too many models may increase model complexity.
  - Not work well on LR since multi-LRs just average all LRs.

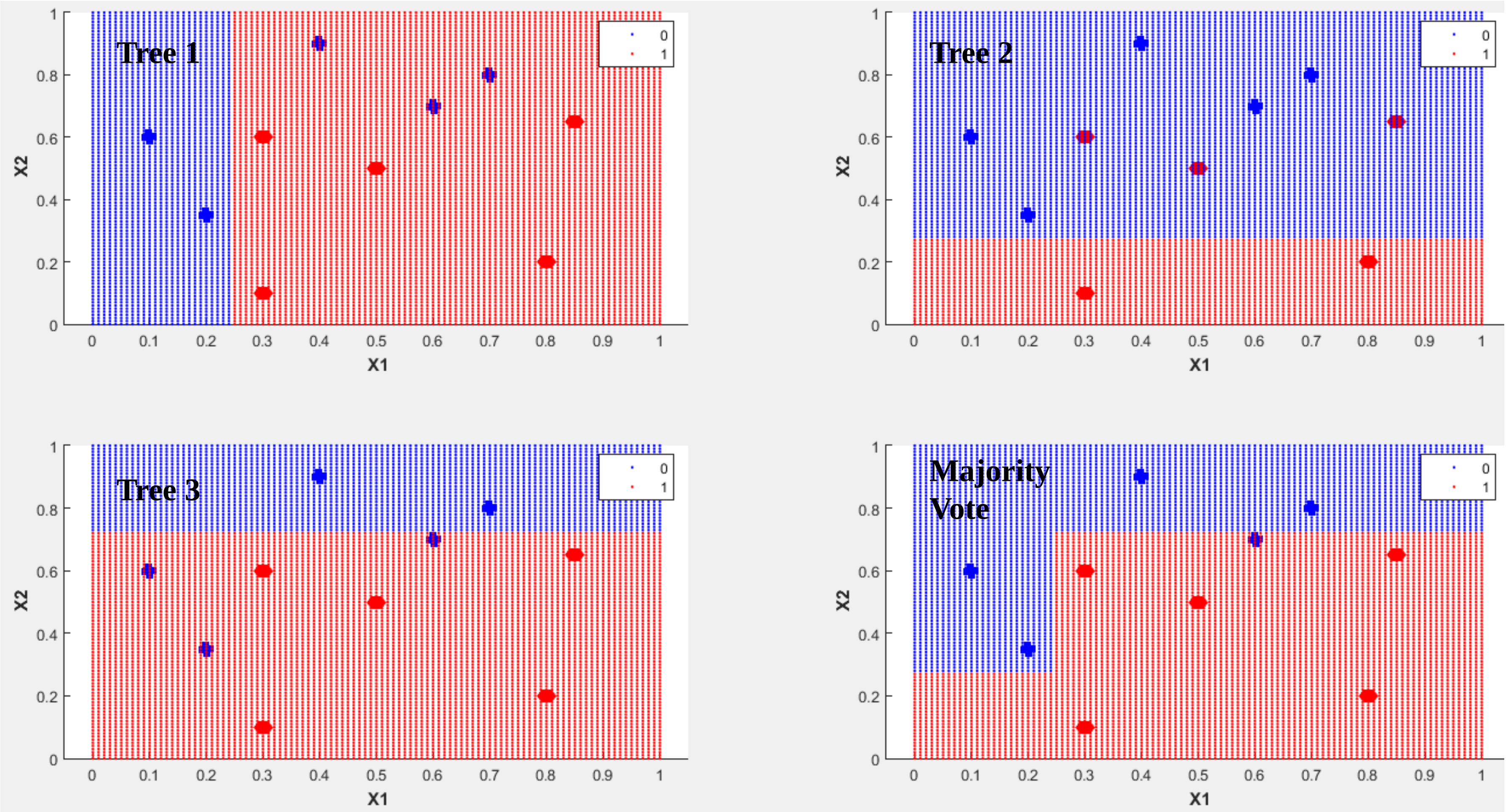
non-linear



Bagging– Visualizing Decision Boundaries

- Create 3 trees with depth 1.
  - Results vary because of **random** sampling in bagging.
  - Majority vote.

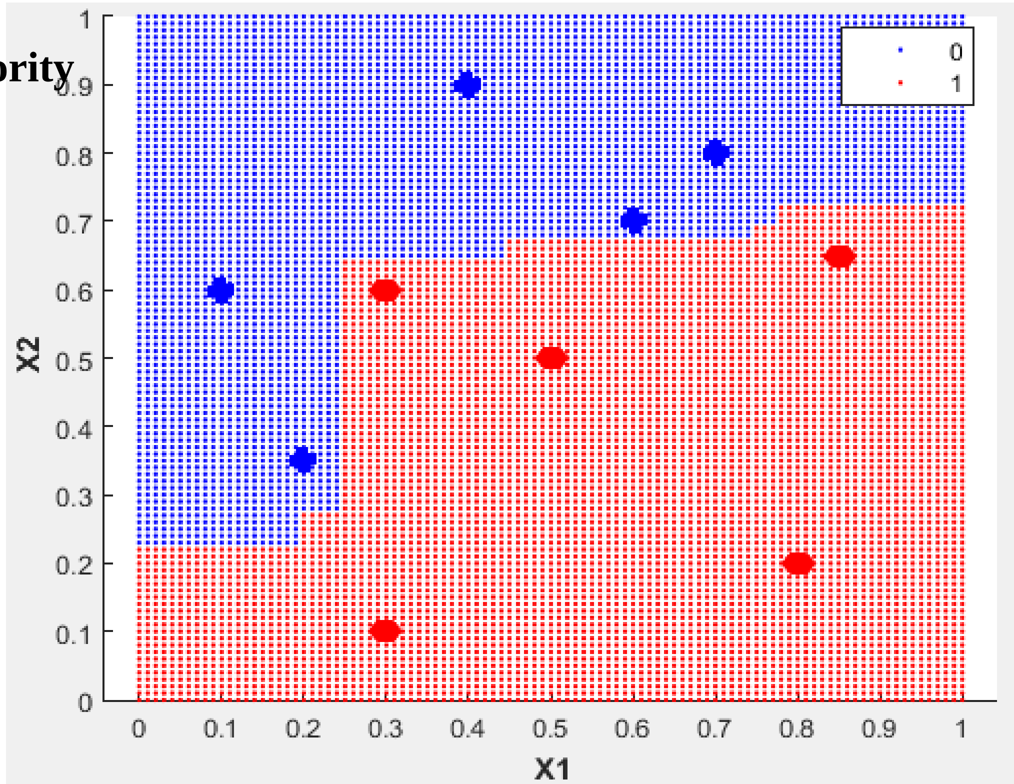




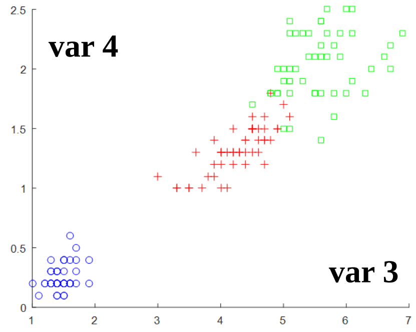
Bagging with 100 Trees

- Bagging for a simple dataset with 100 trees.
  - Create **100** trees with depth 1➔ More complex models for non-linear solutions.
    - **Mdl_All** = fitensemble(X, Y, **'Bag'**, 100, 'Tree', 'Type', 'Classification’)
    - **sklearn.ensemble.BaggingClassifier**
    - **http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html**

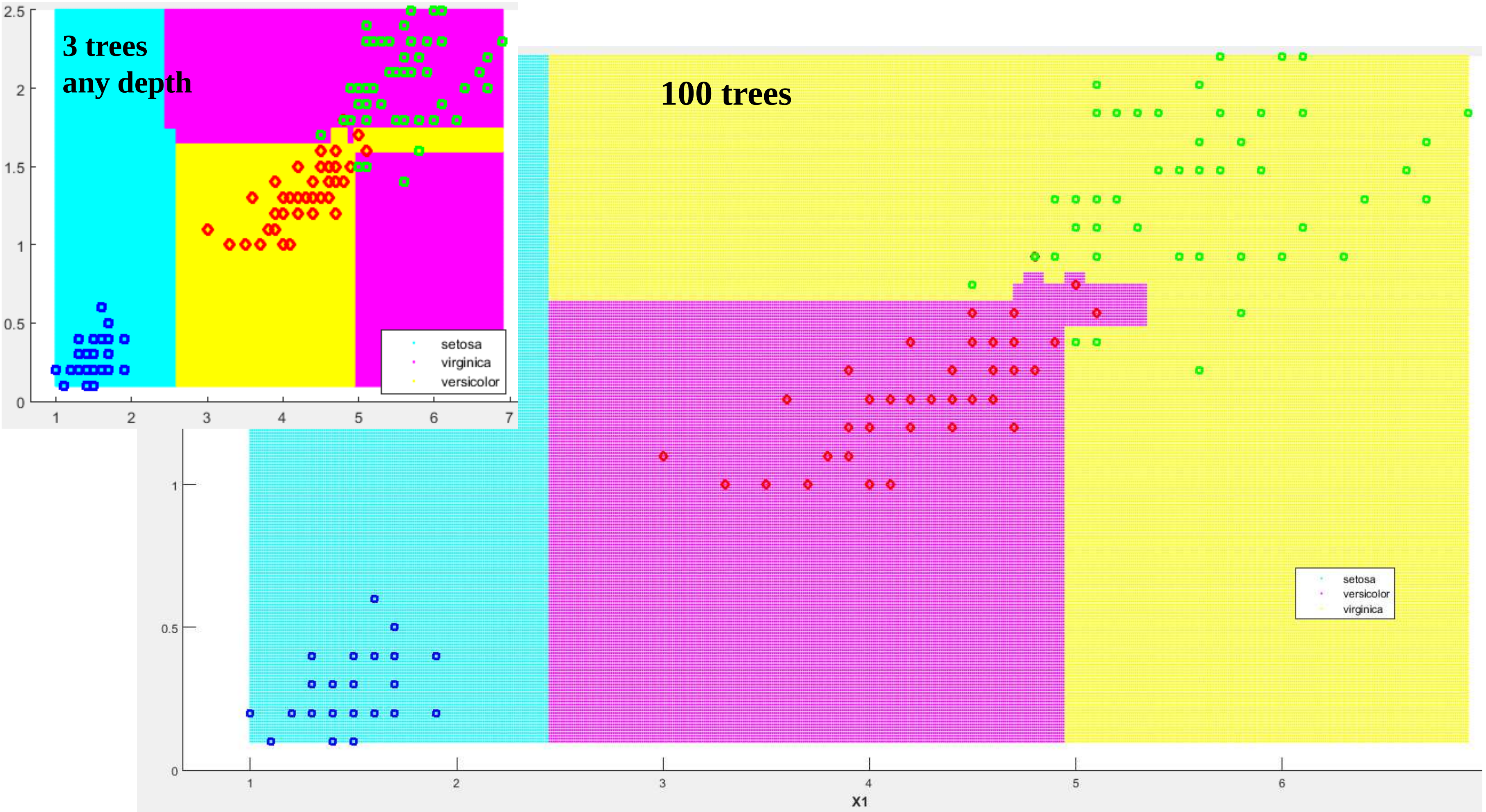
Majority Vote



Bagging, Iris Data, 3 Classes, 100 Trees **(any depth)**




```
load fisheriris
Mdl_All = fitensemble(meas, species, 'Bag', 100, 'Tree', 'Type', 'Classification')
flower = predict(Mdl,mean(meas))
```

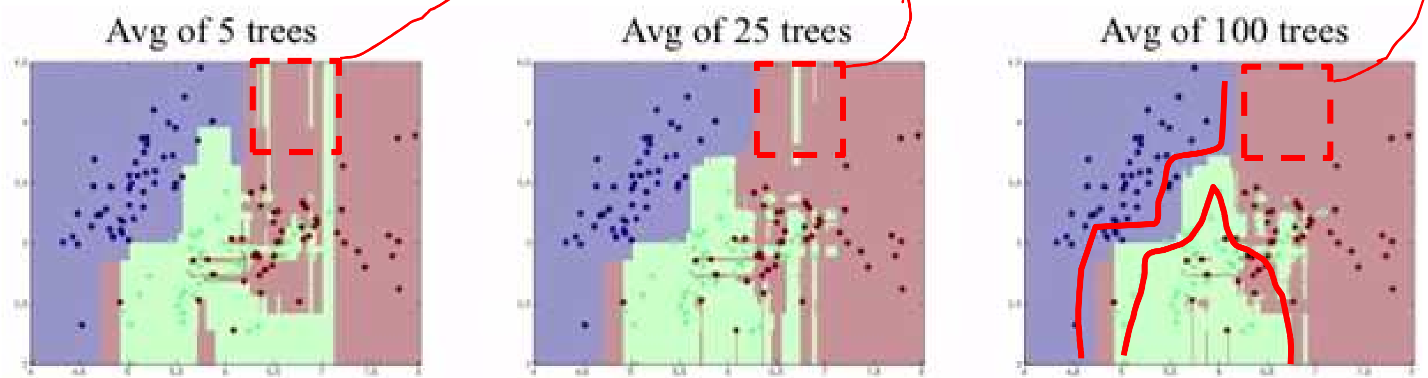


Overfitting??

- Each model was built from different replica of size ***N***.
  - Some data points were sampled multiple times in a model.
  - Every model sees different data points.
  - No model sees all data points.

  - Ensemble model becomes simpler??

green region
begin disappear



Improve Bag Trees with ***Random Forest***

- With large dataset, first few nodes may always be the same on all DTs.
  - When one or few features are very strong predictors, they will be selected on top of many trees.
  - Averaging may not help much.

- From bagging to random forests.
  - Only a random subset of the features is selected when building individual trees.
  - This is "***feature bagging***".

- Random forests randomly select only a subset of features in building trees.
  - To add diversity into forests.
  - Predicting by averaging over all trees (this is same as Bagging).
  - Add "**feature bagging**“ (random forest TreeBagger()) to “**data bagging**”.

- https://en.wikipedia.org/wiki/Random_forest

Boosting

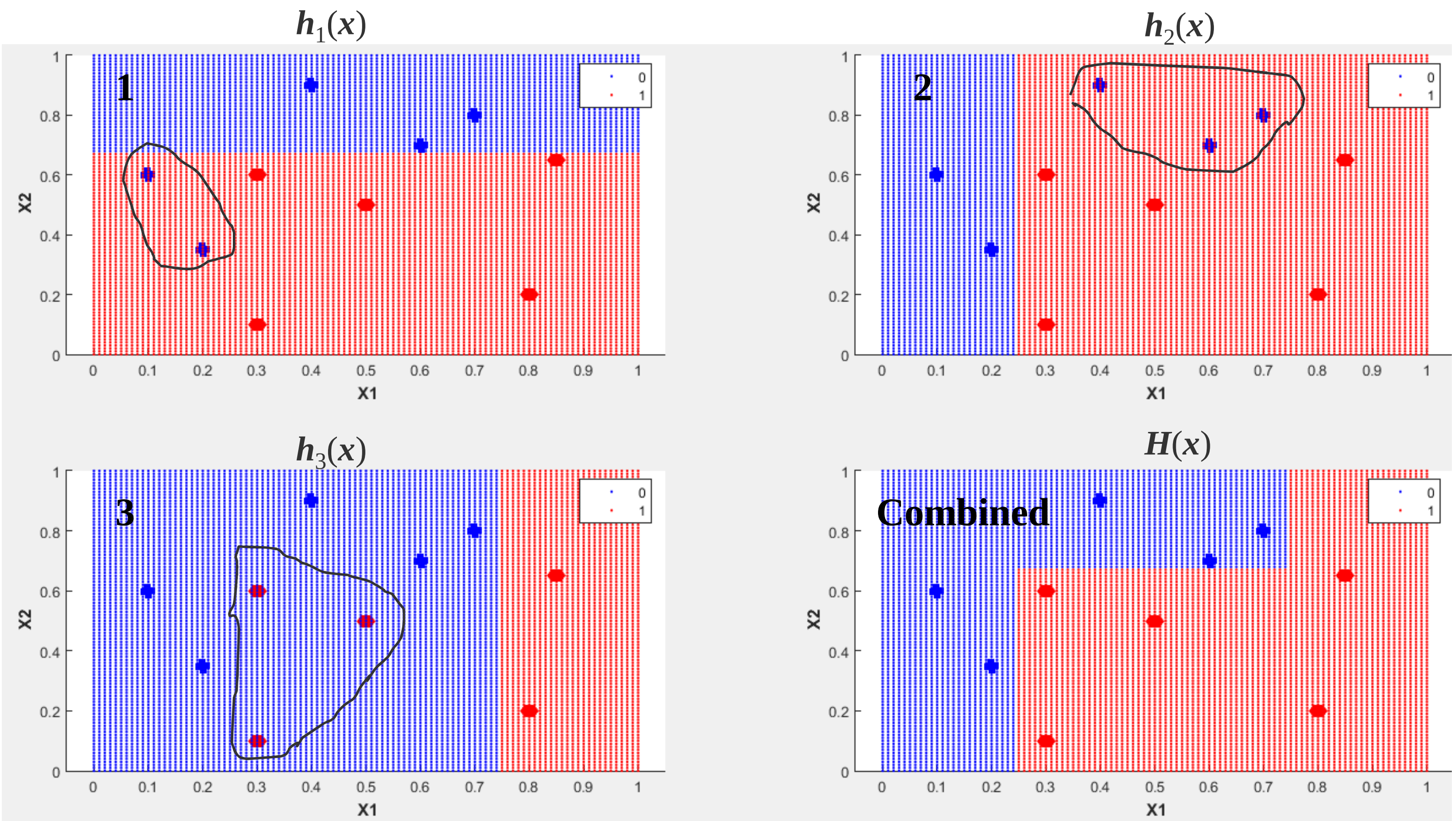
- Gradient boosting contains a collection of models.

- Boosting combine many weak models into a complex model.

- In boosting, learners learn **sequentially**.
  - Early models fit simple models to data, and analyze errors.
  - Errors indicate potential problematic instances of difficult data.
  - Later models focus primarily on those hard data, & try to predict them correctly.
  - All the models are **weighted** and combined to become an overall model.
  - Boosting converts a **sequence** of weak models into a complex model.
  - Model’s complexity may keep increasing.

AdaBoosting– ***Adaptive Boosting***

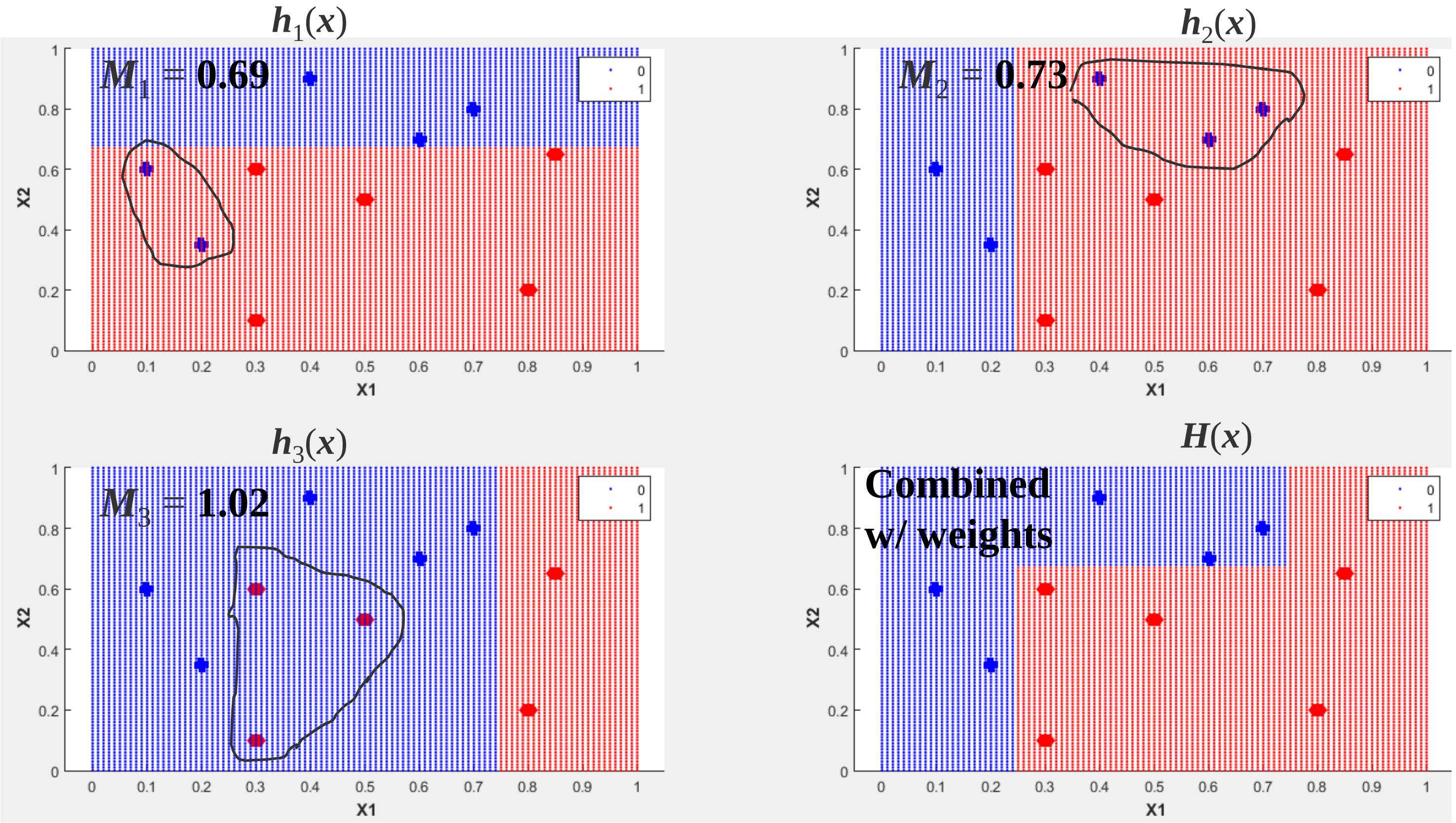
- AdaBoosting for a simple dataset.
  - Iteratively create 3 trees with depth 1.
  - Later models give more weights on previously misclassified data to try to predict them correctly.



Give more weight in each iteration.

AdaBoosting– Visualizing Decision Boundaries

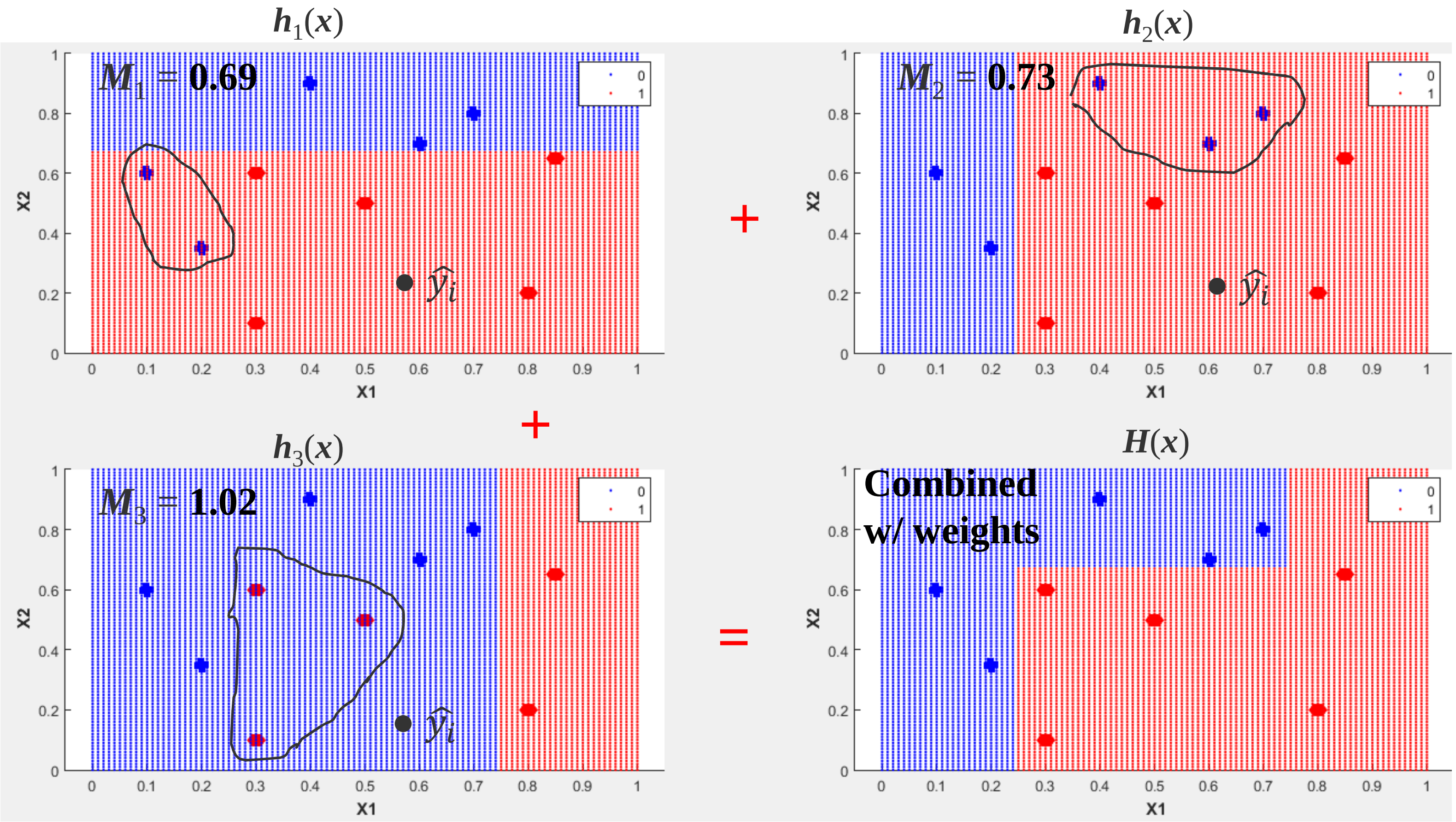
- AdaBoosting for a simple dataset.
  - Iteratively create 3 trees with depth 1.
  - Results are pretty stable (same result in each run).
  - Prediction based on weighted combination from each tree prediction.
  - Access to the weights output by Matlab using “yourModel **.TrainedWeights**”.



Give more weight in each iteration.

Combined Weighted Error for Prediction– AdaBoosting

- Prediction of new data is based on weighted combination from each model.
  - Each model multiply its **±1** class prediction w/ model weight.
  - Let $\boldsymbol{S}$ be the sum of weighted predictions from all classifiers.
  - Predict 1 if $\boldsymbol{S} > 0$. Predict -1 if $\boldsymbol{S} < 0$. $\boldsymbol{S} = \Sigma_i \boldsymbol{M}_i(\boldsymbol{x}) \times \hat{y}_i$



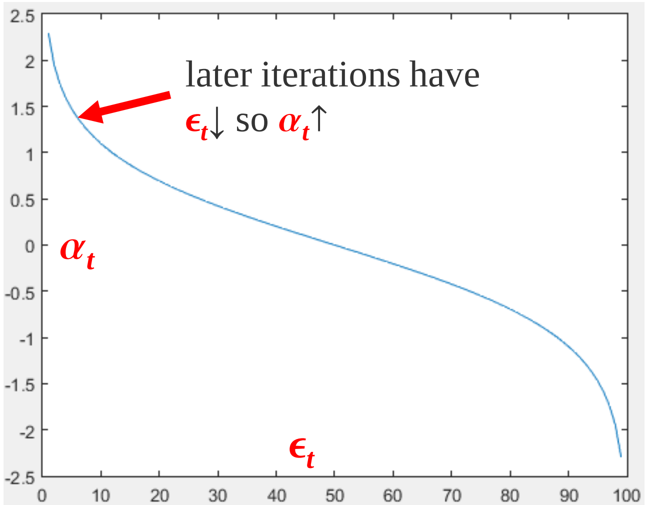
Give more weight in each iteration.

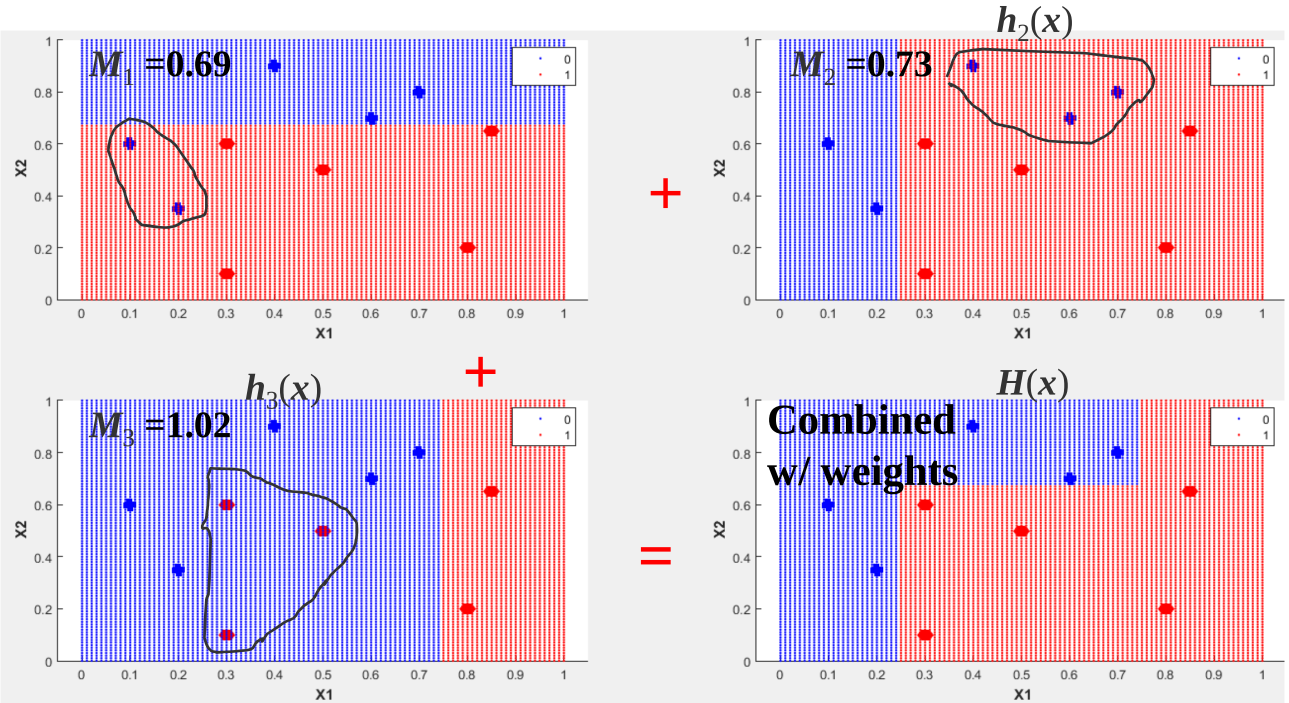
≥ 0

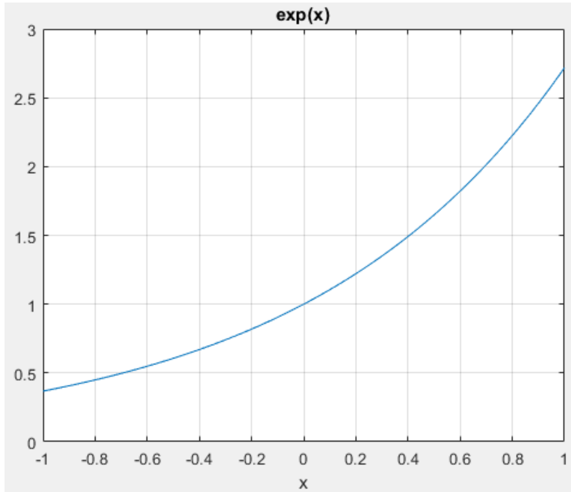
< 0

Data Weights in Iteration t

- $D_{t+1}(i) = \frac{D_t(i)\, exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, Z_t to normalize $\Sigma D_{t+1} = 1$.
  - $D_1(i) = 1 / m$, where m is # of data points.
- $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$, ϵ_t is the error rate in t and $\alpha_t > 0$.
- Final output $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.





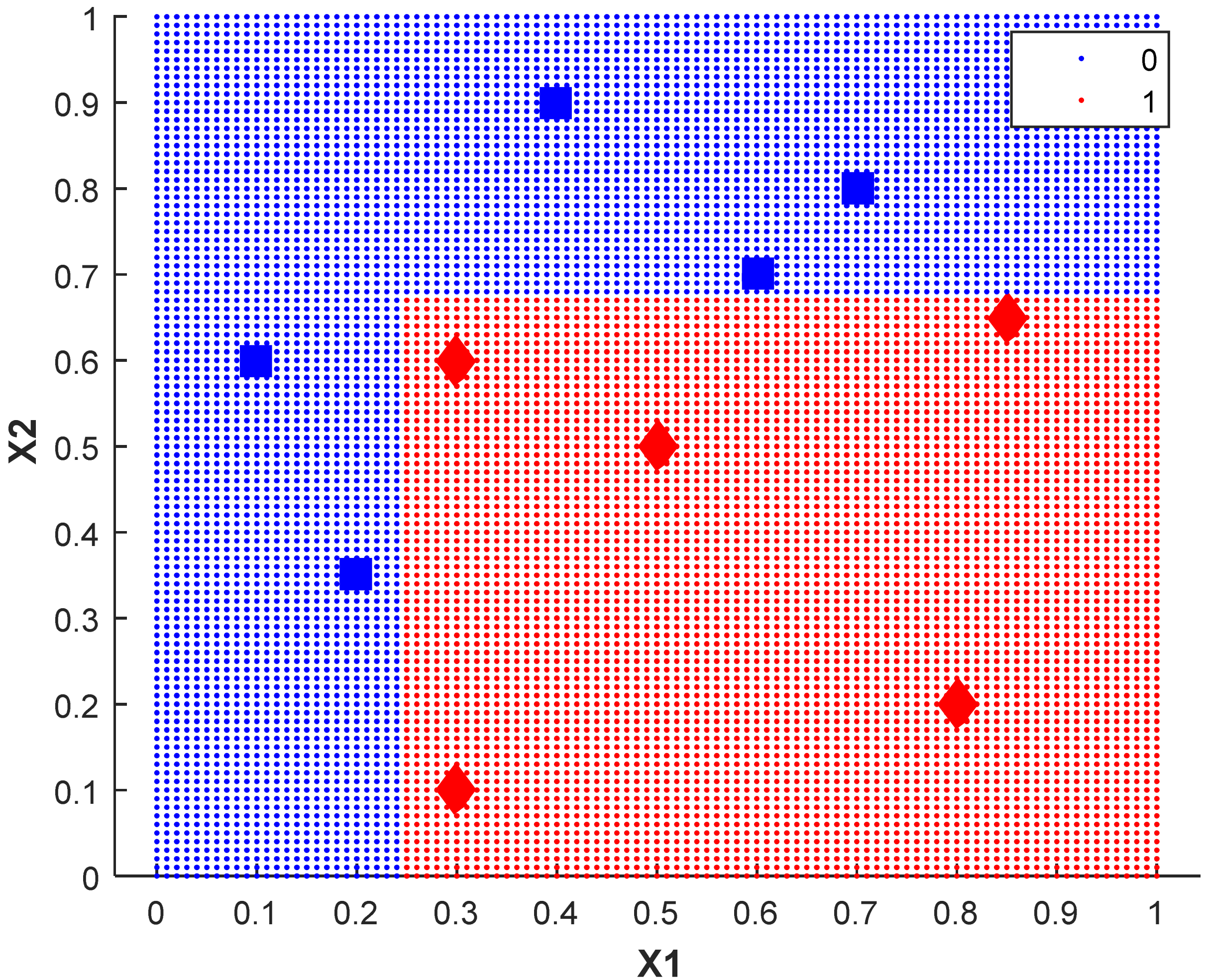


≥ 0

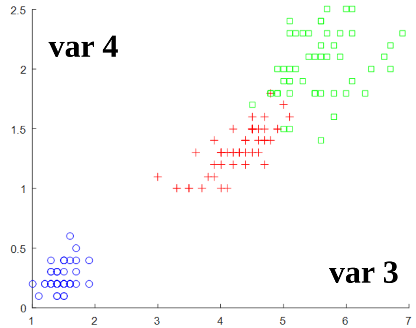
< 0

Decision Boundary with 100 Iterations– AdaBoosting

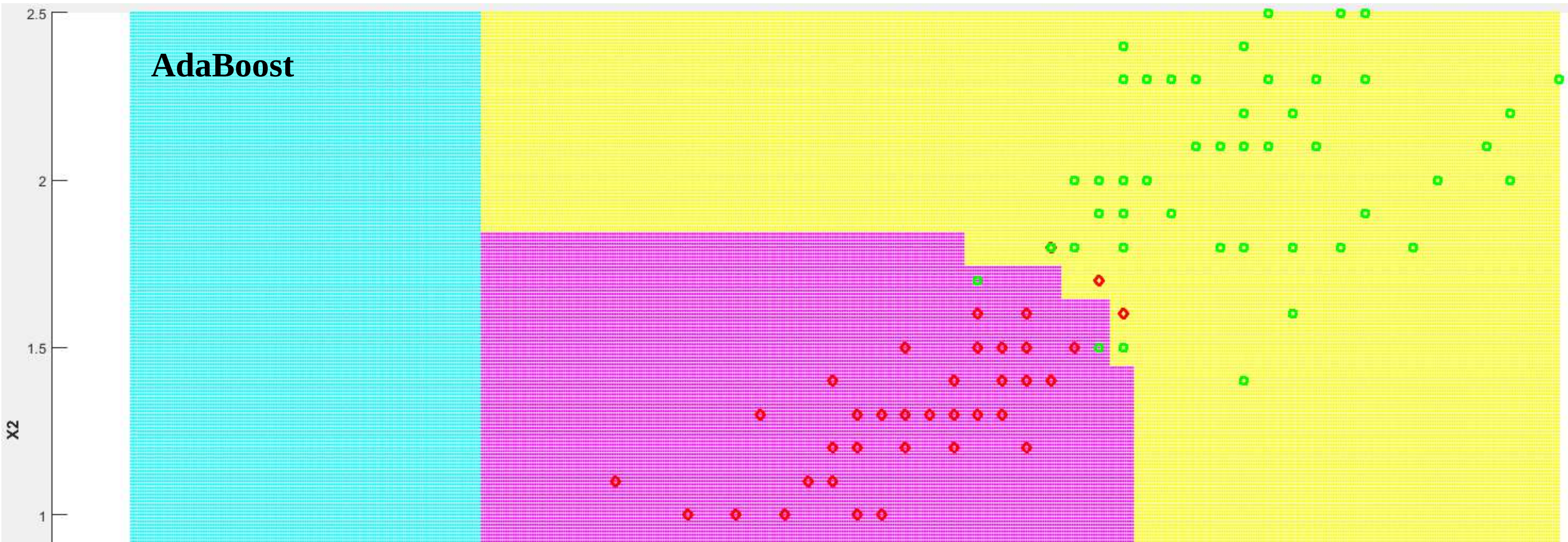
- Less overfitting.
- Results are pretty stable.

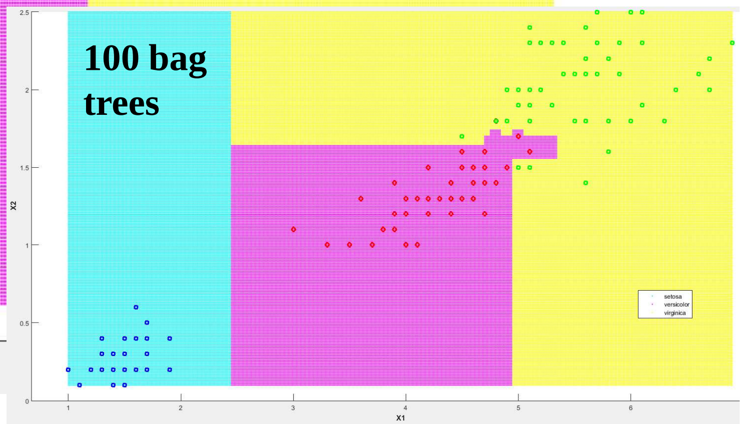


Iris Data, Boosting vs. Bagging **(any depth)**



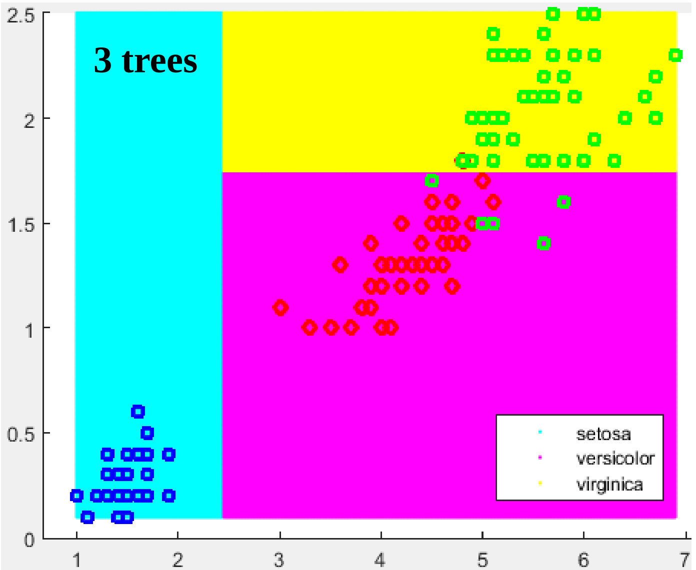

```
load fisheriris
Mdl = fitensemble(meas,species, 'AdaBoostM2', 100, 'Tree')
flower = predict(Mdl,mean(meas))
```

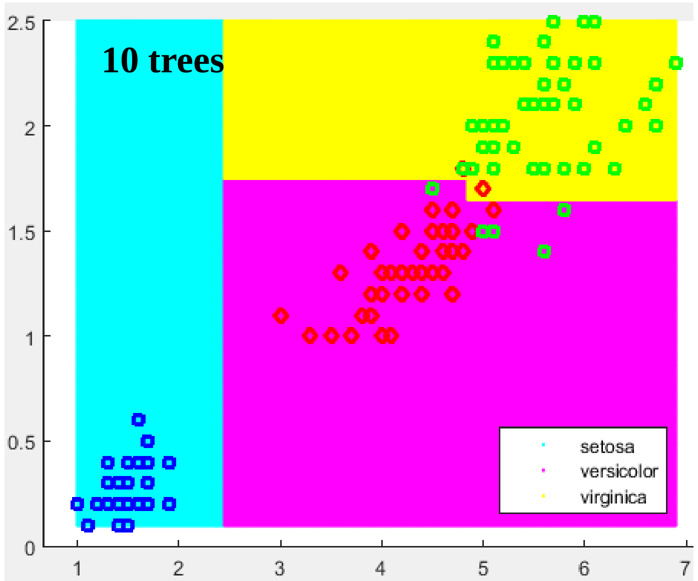


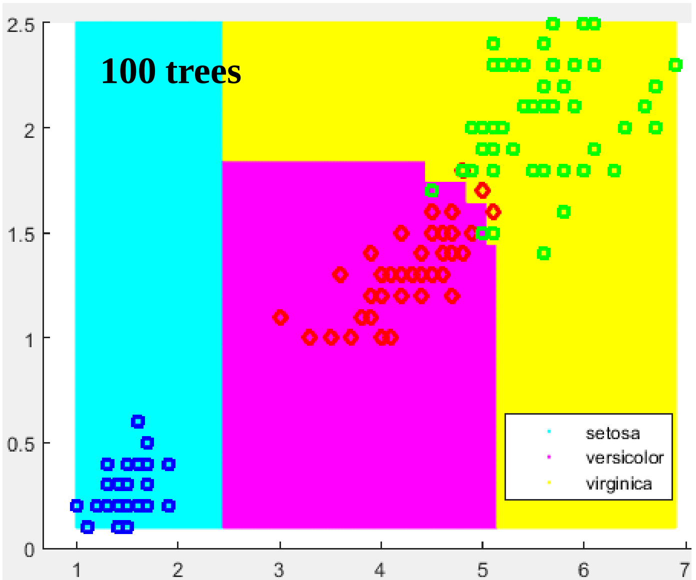


AdaBoosting, Increasing Model Complexity (Iris Data)

- Model's complexity may keep increasing!!!

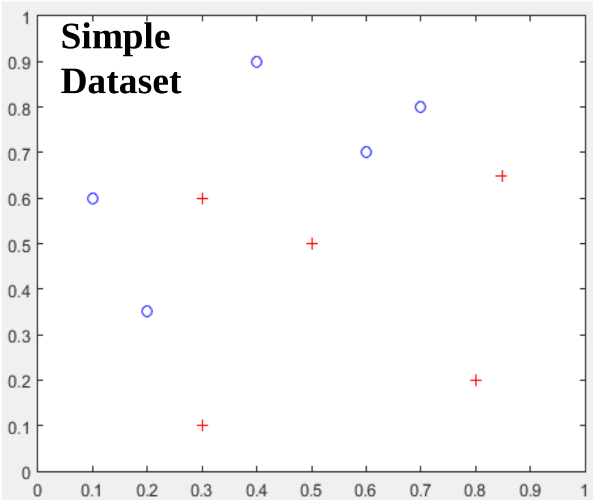


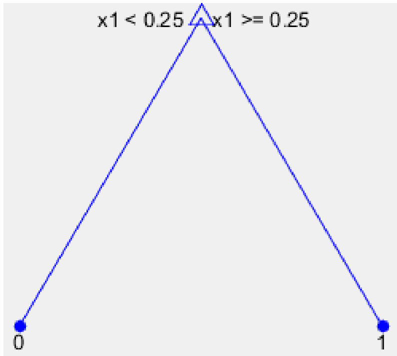


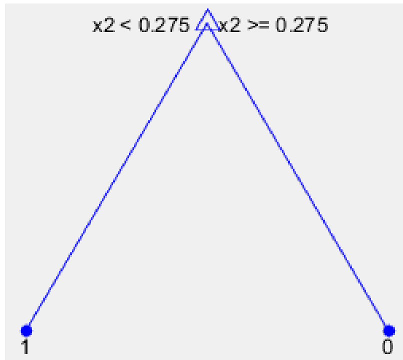


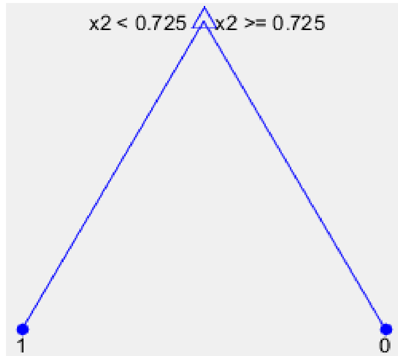
Boosting Program

- Matlab ensemble tree bagging of height 1:
  - **TTP** = **templateTree**('**MaxNumSplits**', 1);
    - **MAX # of branch node = create trees w/ height = 1**
  - ens = **fitensemble**(X, Y, 'AdaBoostM1', 3, **TTP**, 'Type', '**Classification**');









- **Gradient Boosting…**
  - **https://www.mathworks.com/help/stats/fitensemble.html**
  - **http://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regularization.html**

Summary

- Ensembles:
  - Combining predictions from a collection of weak models to improve accuracy.
  - Majority vote or weighted combinations.
- Bagging:
  - Bootstrap aggregation.
  - Resampling data many times for each model.
- Boosting:
  - Train models sequentially; later models focus on mistakes produced in earlier models.
  - Weight “hard” records so later models focus on explaining them more.

Ensembles

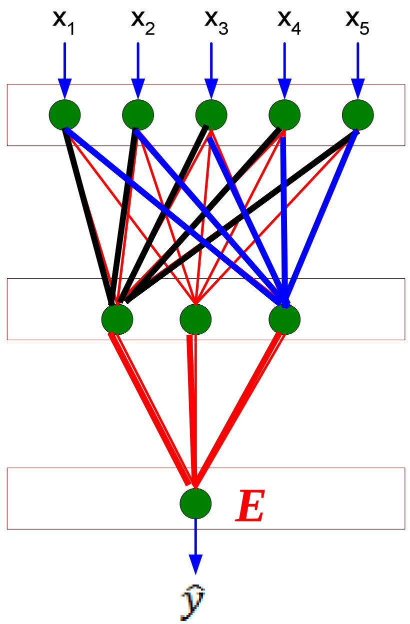
- Combine some weak learners to create one better learner.
  - Wisdom of crowd?!
  - i.e. work yourself   vs. ask friends…
  - If we don't know what kinds of models are best for your prediction problems.
  - More models can be better than a single one.
  - May include same type of model (***bagging*** & ***boosting***) or different types of models.

- Bagging / boosting includes many instances of the same type of model.
  - In regressions, take unweighted / weighted average of predictions.
  - In classifications, take majority vote of predictions or weighted combinations of predictions.

- **Weighted method…**

Weighted Method

- Train ***a model of models***.
  - Prediction from Predictions???
  - Treat the output of each model as a new feature to the final (another) model.
    - Let $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots$ be trained classifiers, $\hat{y}_1 = \boldsymbol{f}_1(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots)$ $\hat{y}_2 = \boldsymbol{f}_2(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots)$ … $\hat{y}_{\boldsymbol{n}} = \boldsymbol{f}_{\boldsymbol{n}}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots)$
    - $\hat{y}_{\boldsymbol{e}} = \boldsymbol{E}(\alpha_1\hat{y}_1,\ \alpha_2\hat{y}_2, \ldots)$, where $\boldsymbol{E}$ is another model to learn weights α_i (weighted vote).
      - Better train ***E*** on different datasets (i.e. validation data)
      - Otherwise, E will be bias to f_i with lowest error which may overfit data.
  - Similar to multi-layer perceptron
    - A multi-layer perceptron trains all layers simultaneously.
    - Outputs from first layer become inputs of next layer.
    - Ensemble trains one model at a time.



Appendix

Matlab Programming

Matlab Framework for Ensemble Learning

- Melding many weak learners' results into high-quality ensemble predictor.
  - Currently only 3 weak learner types in Matlab:
    - Discriminant (recommended for Subspace ensemble)
    - *k*NN (only for Subspace ensemble)
    - Tree (for any ensemble **<u>except</u>** Subspace)

- Create an ensemble with the **fitensemble()** function.
  - ***E*** = fitensemble(***X, Y, M, N, L***)
    - ***X*** is the matrix of data. Each row = one record, each column = one predictor variable.
    - ***Y*** is the vector of responses.
    - ***M*** is a string to name the type of ensemble.
    - ***N*** is the number of weak learners in E from each element of learners.
    - ***L*** is either a string naming a weak learner or a weak learner template.
  - http://www.mathworks.com/help/stats/ensemble-methods.html

Example Weak Learners in Matlab

- Currently only 3 weak learner types in Matlab:
  - Discriminant (recommended for Subspace ensemble)
  - *k*NN (only for Subspace ensemble)
  - Tree (for any ensemble **except** Subspace)

- Syntax:
  - ens = fitensemble(X, Y, 'AdaBoostM2', 50, 'Tree');
  - ens = fitensemble(X, Y, 'Subspace', 50, 'KNN'); **% or 'Discriminant'**
  - For nondefault weak learner options, check http://www.mathworks.com/help/stats/ensemble-methods.html
    - ens = fitensemble(X, Y, 'Bag', 50, **Your_Own_Special_Learner**);

Set the Number of Ensemble Members

- Choosing the size of an ensemble involves balancing speed and accuracy.
  - Larger ensembles take longer to train and to generate predictions.
  - Eensemble algorithms can overfit data when it's too large. (see next slide)

  - Consider starting with several dozen to several hundred members in an ensemble.
  - Build the ensemble, and check the ensemble quality, as in Test Ensemble Quality.
  - If it appears that you need more members, add more using the **resume()** method.
  - Repeat until adding more members does not improve ensemble quality.

  - **LPBoost** & **TotalBoost** are self-terminating & will find appropriate ensemble size.
    - Try setting ***N*** to 500, and they usually terminate with fewer members.

  - ***E*** = fitensemble(***X***, ***Y***, ***M***, ***N***, ***L***)

Applicable Ensemble Methods in Matlab

- fitensemble() uses one of these algorithms to create an ensemble.

http://www.mathworks.com/help/stats/ensemble-methods.html

Algorithm	**Regress.**	**2 classes**	**2-class level predictors**	**3+ classes**	**Skew**	**Stop**	**Sparse**	**tree**	**Comments**
Bag	X	X		X				X	deep trees, oobLoss(), ~big data
AdaBoostM1		X						X	shallow trees,
AdaBoostM2				X				X	shallow trees,
LogitBoost		X	X					X	
GentleBoost		X	X					X	
RobustBoost		X						X	
LPBoost		X		X		**X**	X	X	~big data
TotalBoost		X		X		**X**	X	X	~big data
RUSBoost		X		X	X			X	
LSBoost	X							X	
Subspace		X		X				**LDA**, knn	for many vars,

- Bagging usually uses deep treesss➔ time consuming & memory-intensive.
- Boosting usually uses shallow treesss➔ little time & memory. But, # of ensemble members > **(or >>)** # of bagged trees.
- Except Subspace, all boosting & bagging in Matlab are based on trees.

Test Ensemble Quality– Matlab Diagnosis

- Independent Test Set
  - cvpart = cvpartition(Y,'holdout',0.3);
  - Xtrain = X(training(cvpart),:); Ytrain = Y(training(cvpart),:);
  - Xtest = X(test(cvpart),:); Ytest = Y(test(cvpart),:);
  - **bag** = fitensemble(X, Y, 'Bag', 200, 'Tree', 'Type', 'Classification')
  - **cv** = fitensemble(X, Y, 'Bag', 200, 'Tree', 'type', 'classification', 'kfold', 5)
  - plot(loss(bag,Xtest,Ytest,'mode','cumulative'));
  - plot(kfoldLoss(cv,'mode','cumulative'),'r.');
  - plot(kfoldLoss(cv,'mode','cumulative'),'r.');
  - plot(oobLoss(bag,'mode','cumulative'),'k--');