Data Mining & Machine Learning

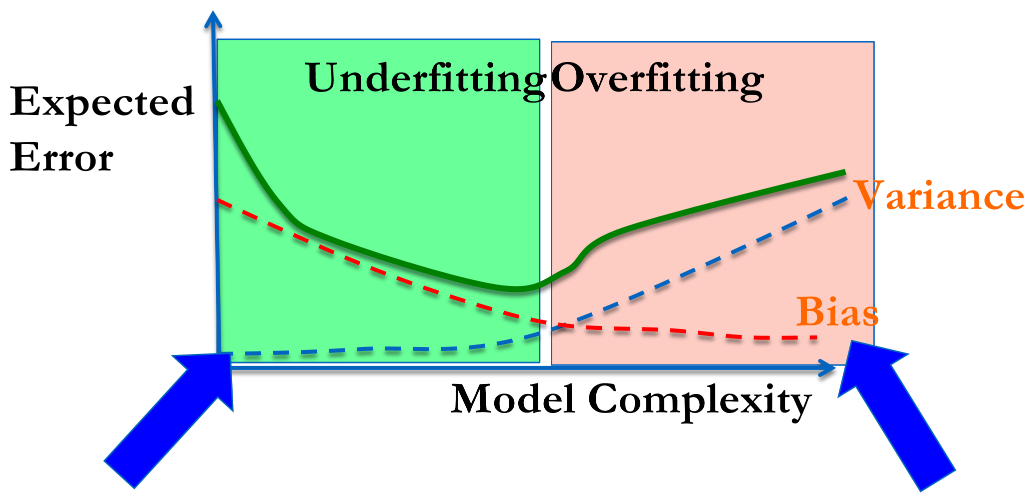
CS37300
Purdue University

Oct 4, 2023

Bias/Variance (contd)

- Bias:
  - Trust the data less
  - Simpler models
  - Tendency to underfit
  - Could mean need more features
  - "Stable" models
- Variance
  - Trust the data more
  - More complex models
  - Tendency to overfit
  - Add more data points
  - "Unstable" models – model (and prediction) can change a lot

Model Complexity



Simple models:
High bias and low variance

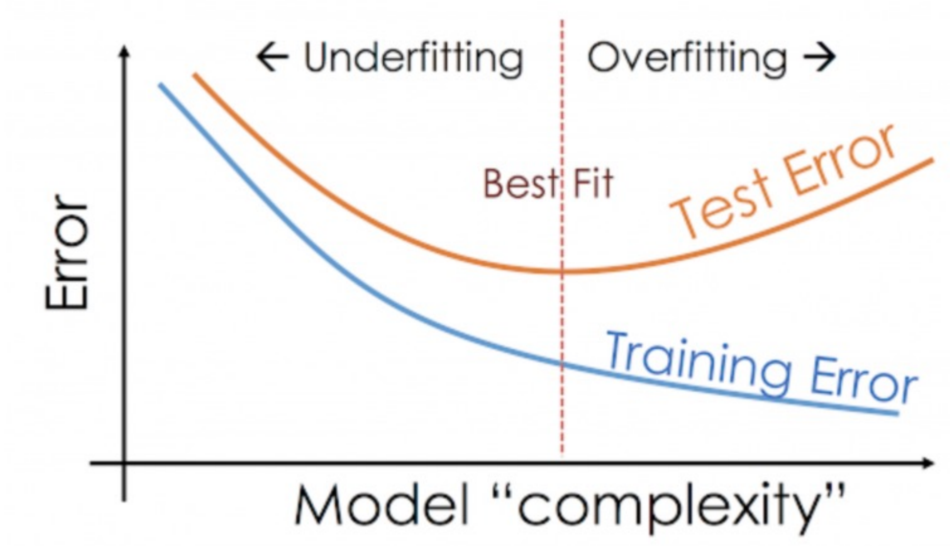
Complex models:
High variance and low bias

Expected Error ≈ Bias + Variance

Bias-Variance Tradeoff in Practice

- We saw that the classification error can be (informally) expressed in terms of bias and variance
- Reducing the bias and variance can reduce expected error!
- Different scenarios can lead to different actions for reducing the error
  - **High bias**: add more features
  - **High variance**: simplify the model, add more examples
- *How can we diagnose each one of these scenarios?*

Overfitting

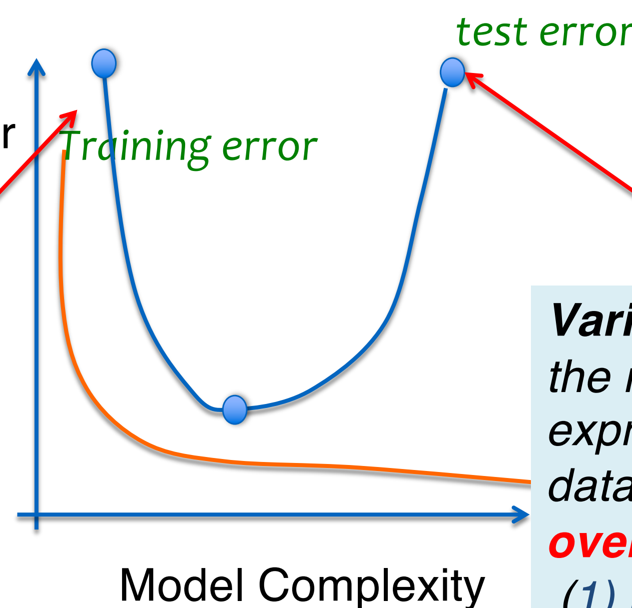


- Question 1: How do we detect overfitting?

- Question 2: How do we prevent / correct overfitting?

Bias-Variance Analysis

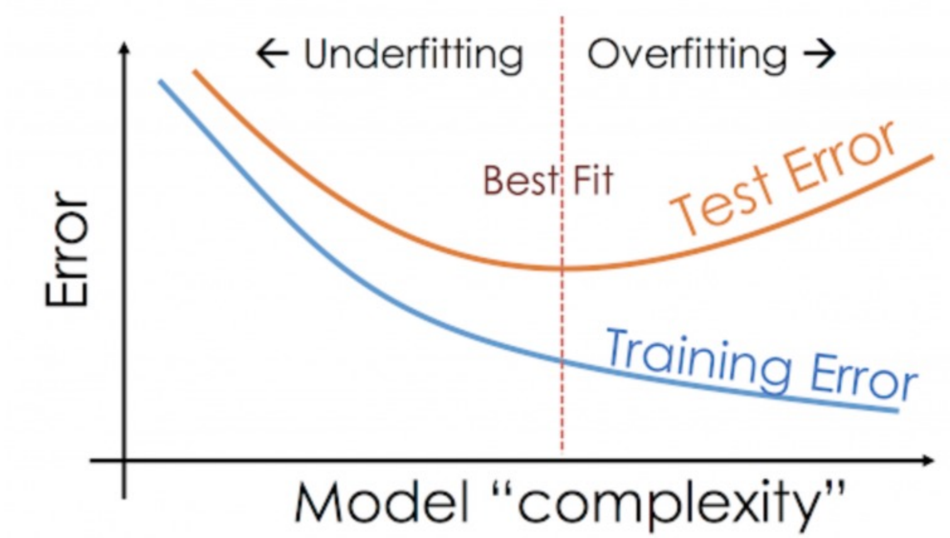
Interpolating over the points: two curves that we can use for ***diagnosis***



Bias problem : *the model is not expressive enough.*
Underfitting
(1) high training error
(2) test error ≈ training error

Variance problem:
the model is too expressive for the data set that we have
overfitting
(1) low training error,
(2) test error > training error.

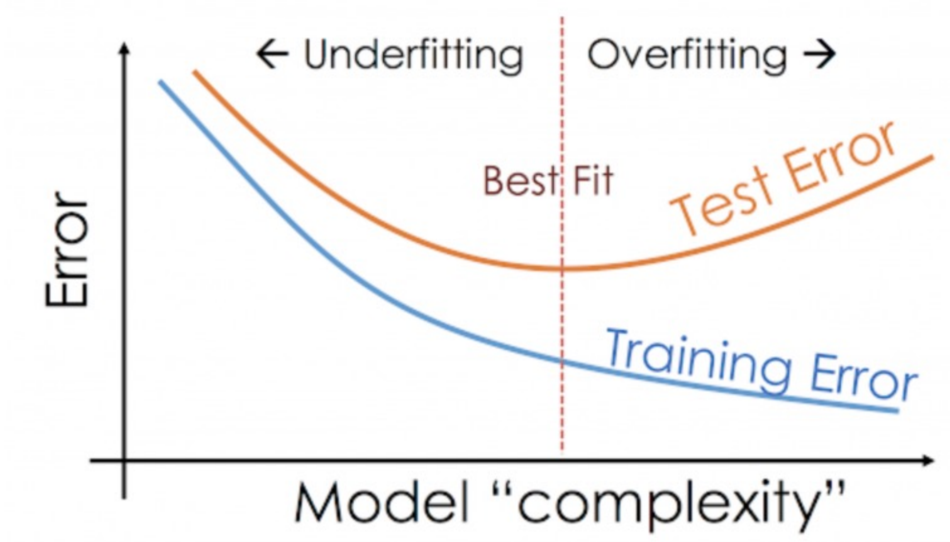
Overfitting



- Question 1: How do we detect overfitting?

- **Question 2: How do we prevent / correct overfitting?**

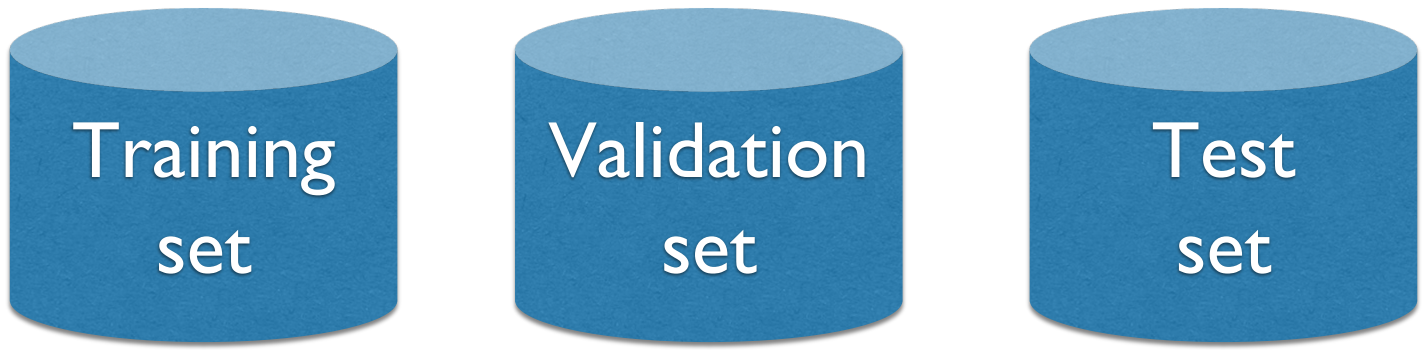
Preventing / Correcting Overfitting



- Model Selection: Find a model that gives good test error
  - Often this is just tuning a "hyper-parameter" (e.g., k in kNN, bandwidth in kernel regression, C in Soft-SVM)
  - Sometimes more involved: e.g., post-pruning in decision trees
- A few ways to do this:
  - Validation set (or cross-validation)
  - Theoretical approaches [Structural Risk Minimization]

Training, Validation, Testing

- Split data set into **three** data sets: **training**, **validation**, **test**

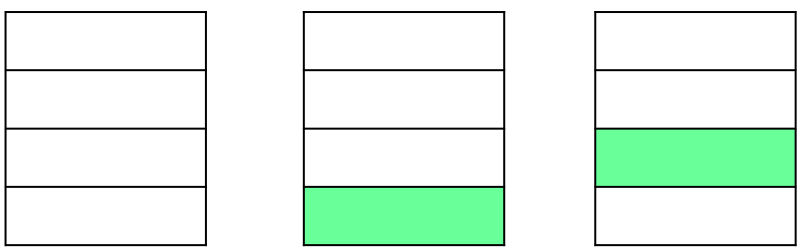


Try different hyper-parameters
(for instance: C=0.1, C=1, C=10 for SVM, or k=1,2,3,…n for kNN)

Report test error rate for the hyper-parameter value
that gave smallest validation error rate

Cross-validation

- Problem: What if the training set is an "unlucky" distribution
  - Error on the test set doesn't match real data
- Solution: Use *all* of the data as test data
  - But then we don't have any data to train on!
- Instead, cross-validation
  - Multiple training/test runs
  - Each uses a different subset as test data



Cross-validation

- **K-fold cross-validation**:
- Randomly partition the training data into K equal-size subsets $S_1, \ldots, S_K$
- For each i
  - Train on $S_1 \cup \cdots \cup S_{i-1} \cup S_{i+1} \cup \cdots \cup S_K$ (all the data except S_i)
  - Call this classifier $\hat{h}_i$
  - Evaluate error rate on S_i : $\mathrm{error}_{S_i}(\hat{h}_i)$
- Return the average: Cross-validation error = $\frac{1}{K}\sum_{i=1}^{K} \mathrm{error}_{S_i}(\hat{h}_i)$
- We can use this to tune hyper-parameters
  - For each setting of the hyper-parameters
  - Run K-fold cross validation
  - Choose the hyper-parameter values with lowest cross-validation error
  - Retrain on the **entire** training set, using the chosen hyper-parameter values

Cross-validation

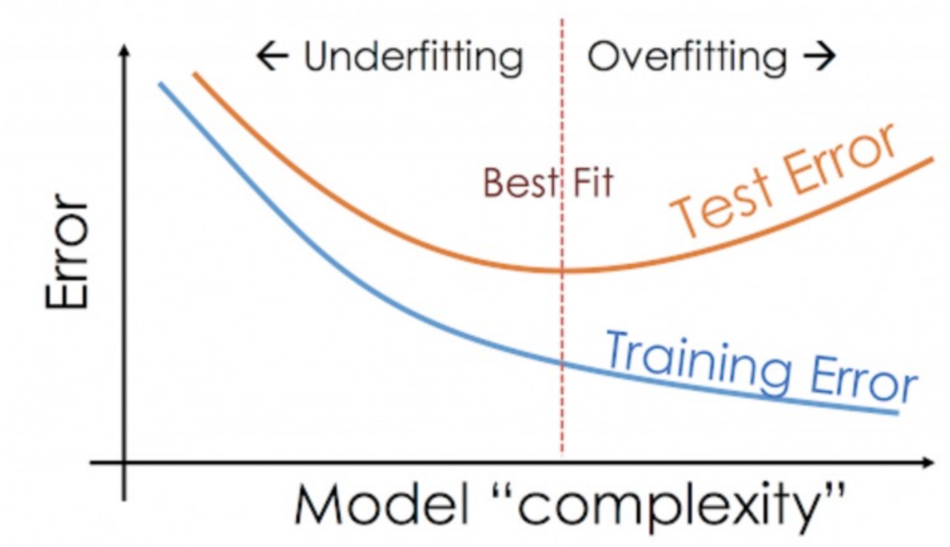
- How do we pick K?

- Most popular in practice: K=10

- If K=n, it's called "leave one out" cross-validation:
  - Every training point (x_i,y_i) gets its own fold $S_i=\{(x_i,y_i)\}$
  - But this is computationally expensive
  - Also can sometimes have higher variance

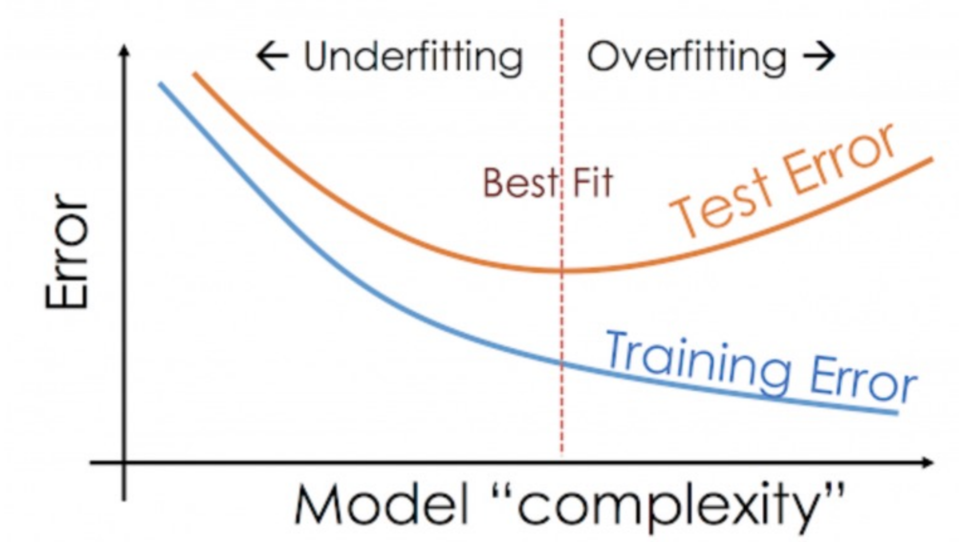
- To estimate the error rate in the end,
  we would still need a separate held-out test set

Preventing / Correcting Overfitting



- Other approaches:
  - Regularization (e.g., minimizing ||w|| in SVM optimization)
  - Dimensionality reduction / feature selection

Preventing / Correcting Overfitting



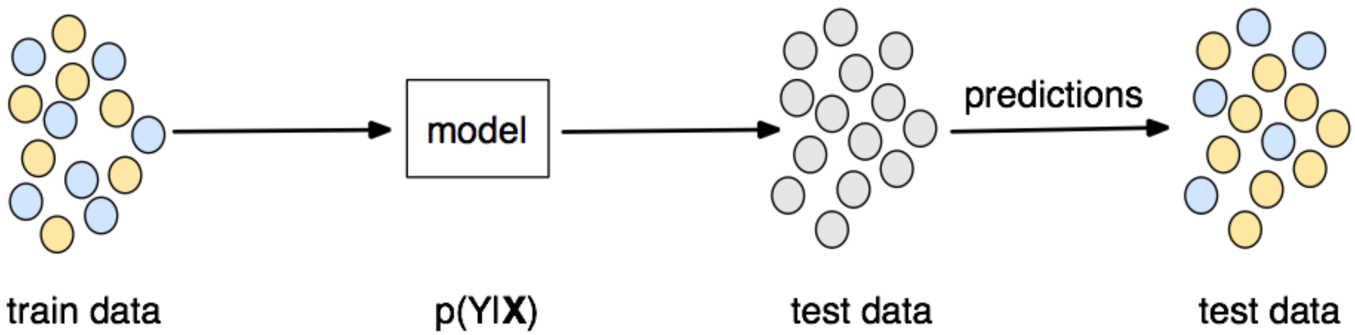
- Other approaches:
  - Regularization (e.g., minimizing ||w|| in SVM optimization)
  - Dimensionality reduction / feature selection
- Note: sometimes the appropriate units for x-axis aren't easy to identify.
  - Sometimes large neural networks overfit less than smaller
  - Possibly because the optimization finds good solutions more easily

Ensemble methods

Ensemble methods

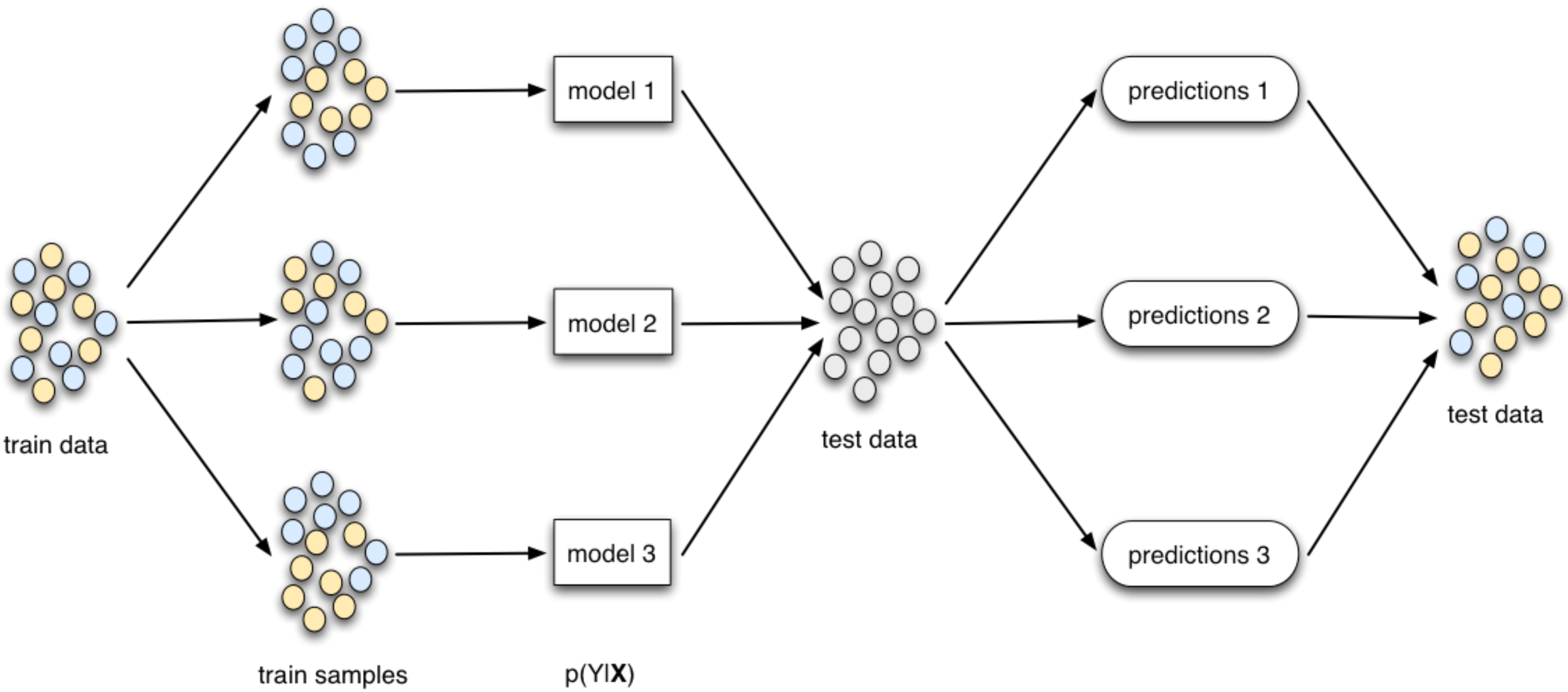
- Motivation
  - Too difficult to construct a single model that optimizes performance (why?)
- Approach
  - Construct many models on different versions of the training set and combine them during prediction
- Goal: reduce bias and/or variance

Conventional classification



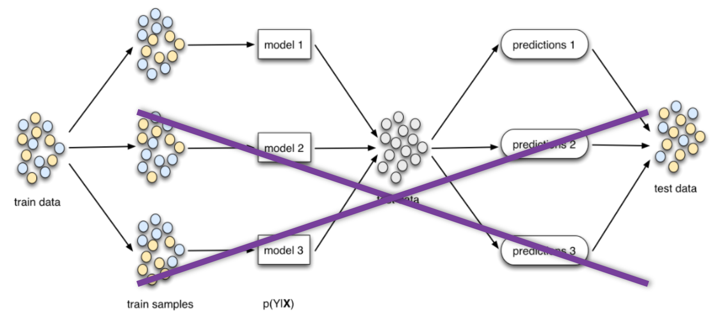
X: attributes
Y: class label

Ensemble classification

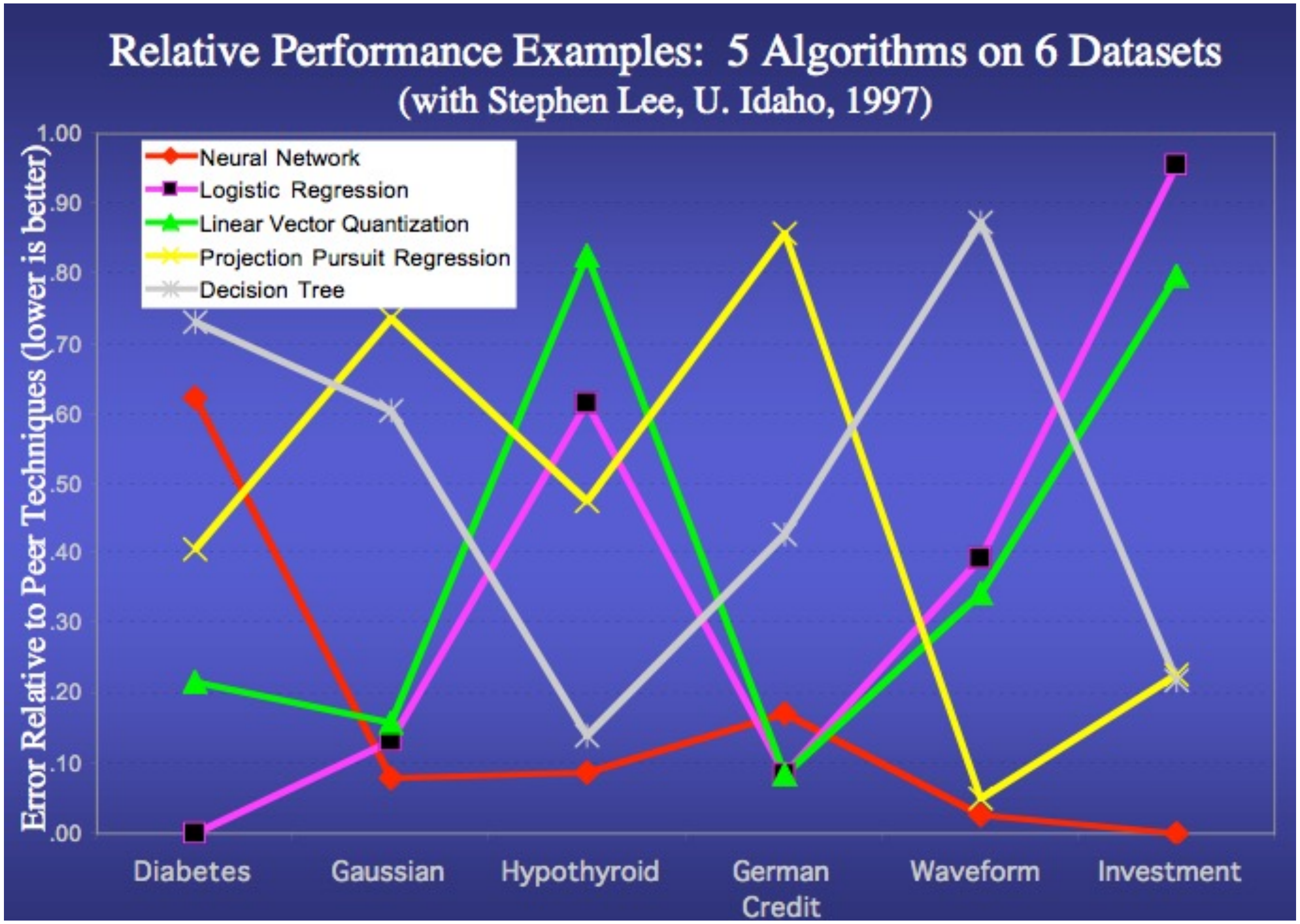


Why not choose the best classifer?

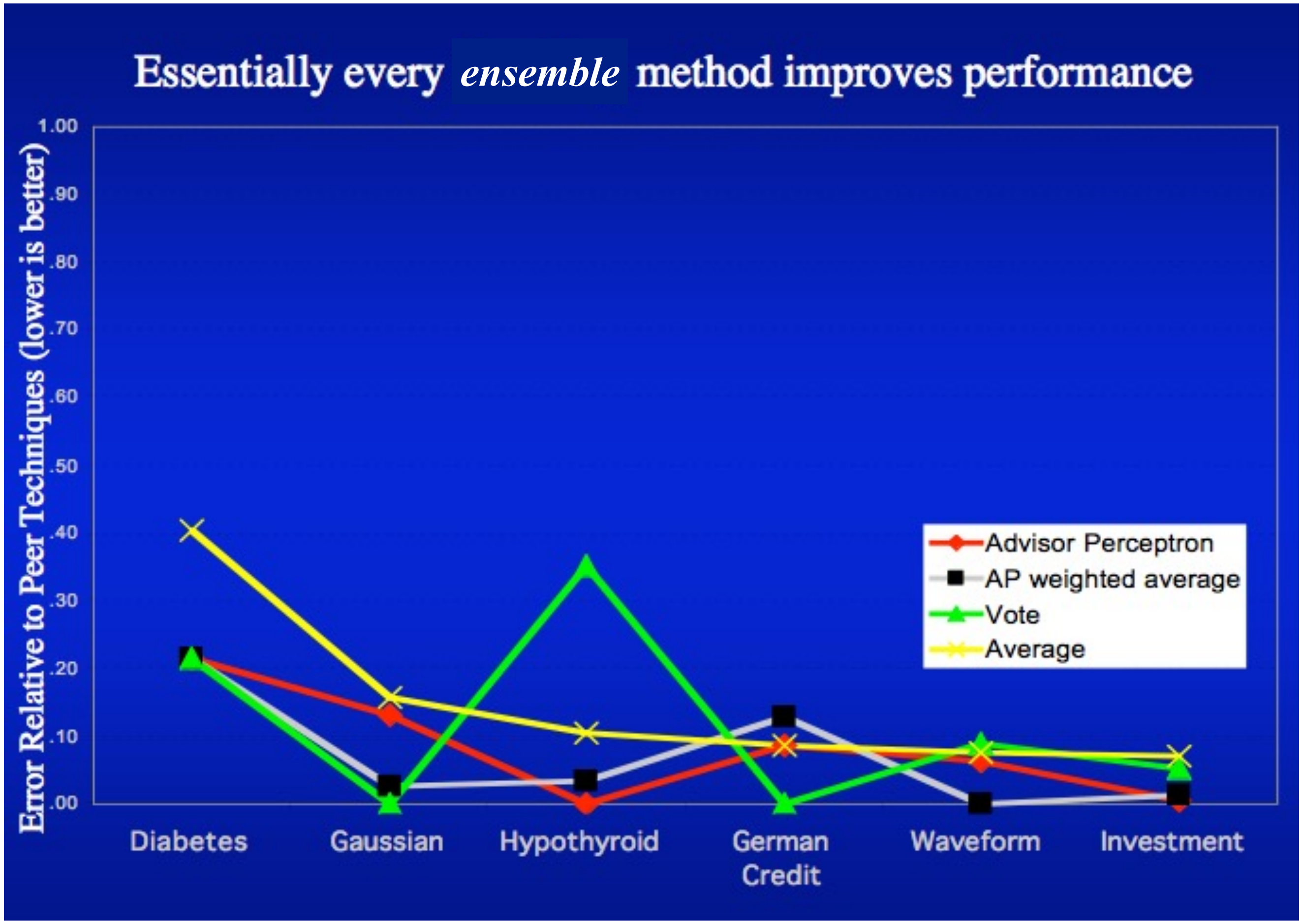
- Best on what?
  - Training data?
  - Test data?
  - After cross-validation?



- *Think of as "multiple hypotheses, not sure which is best*
  - Ensembles: use them all
- We can formally state this in terms of bias/variance reduction
  - Bias: Erroneous assumptions in the model
  - Variance: Sensitivity to small fluctuations in training data

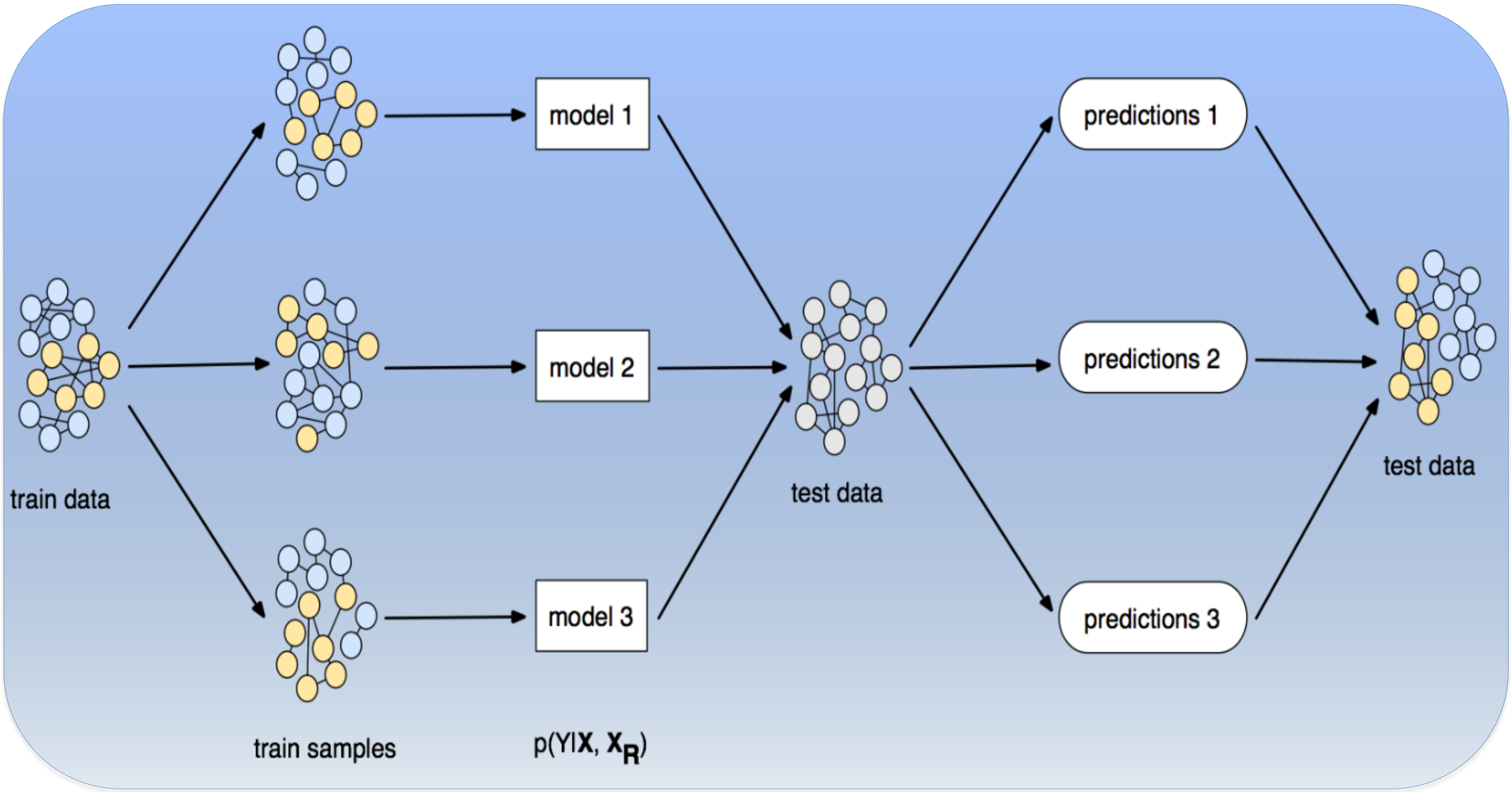


source: *Top Ten Data Mining Mistakes, John Edler, Edler Research*)

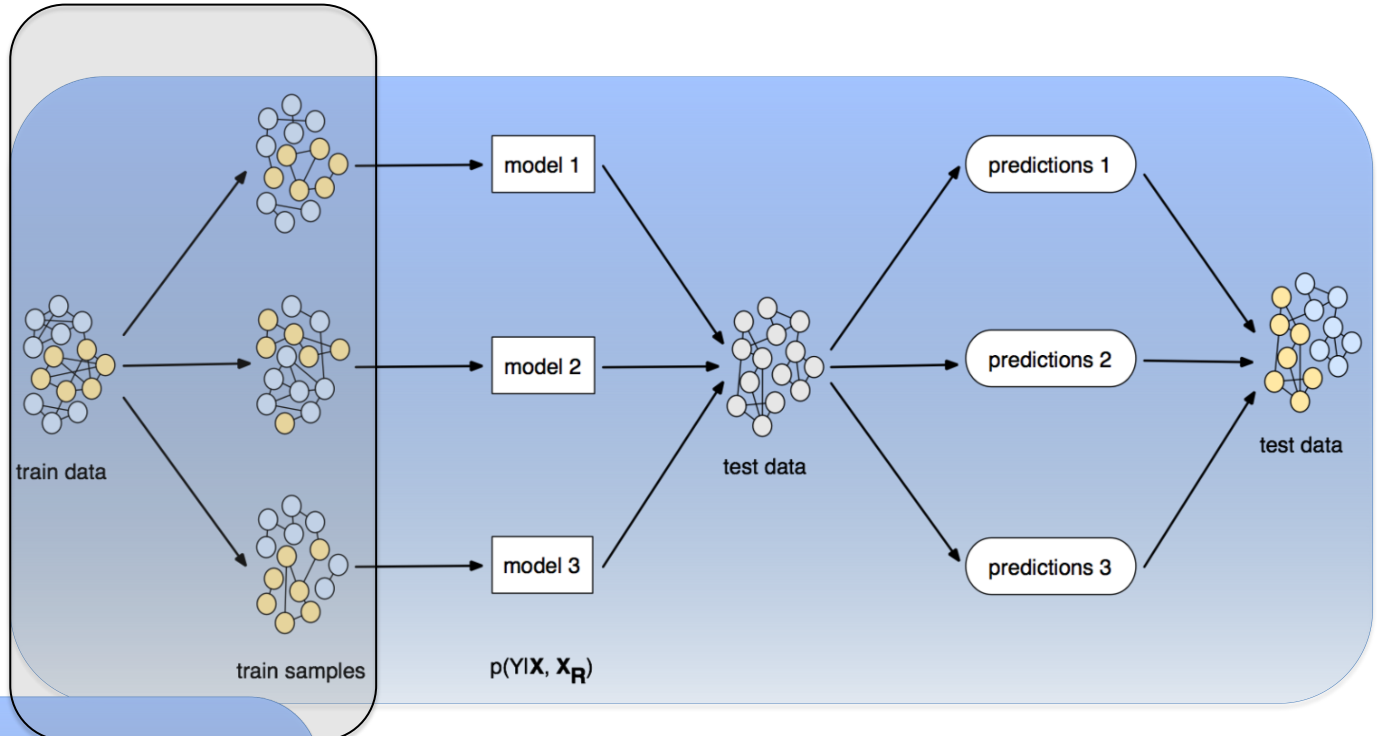


source: *Top Ten Data Mining Mistakes, John Edler, Edler Research*)

Ensemble design



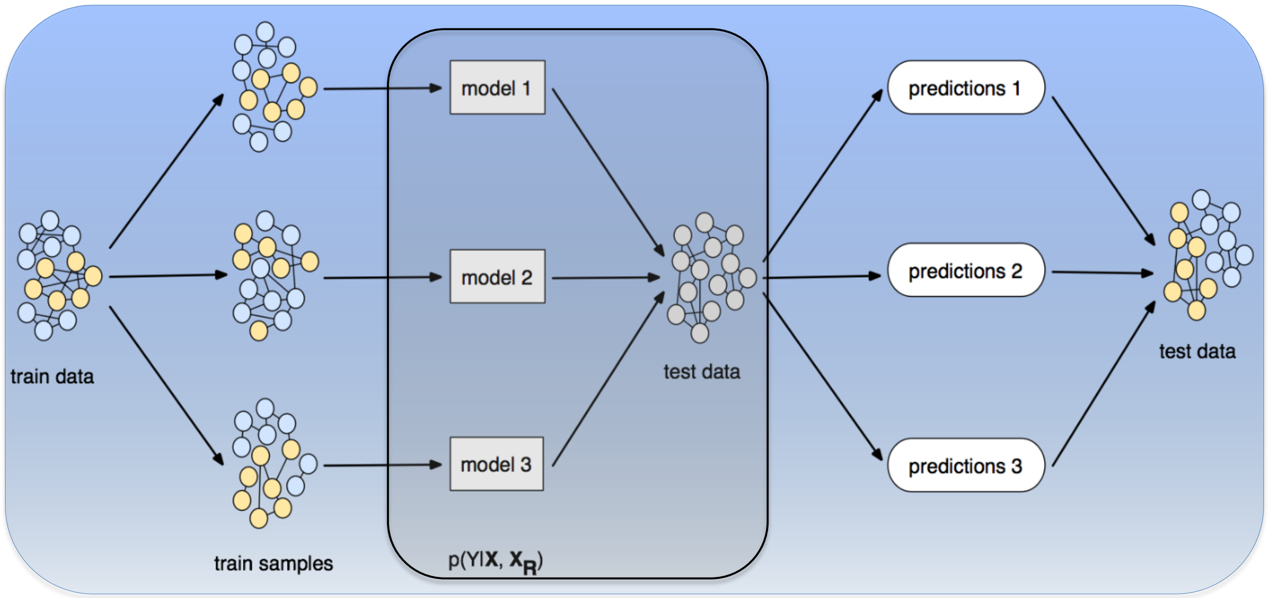
Ensemble design



TREATMENT OF INPUT DATA

- *sampling*
- *variable selection*

Ensemble design



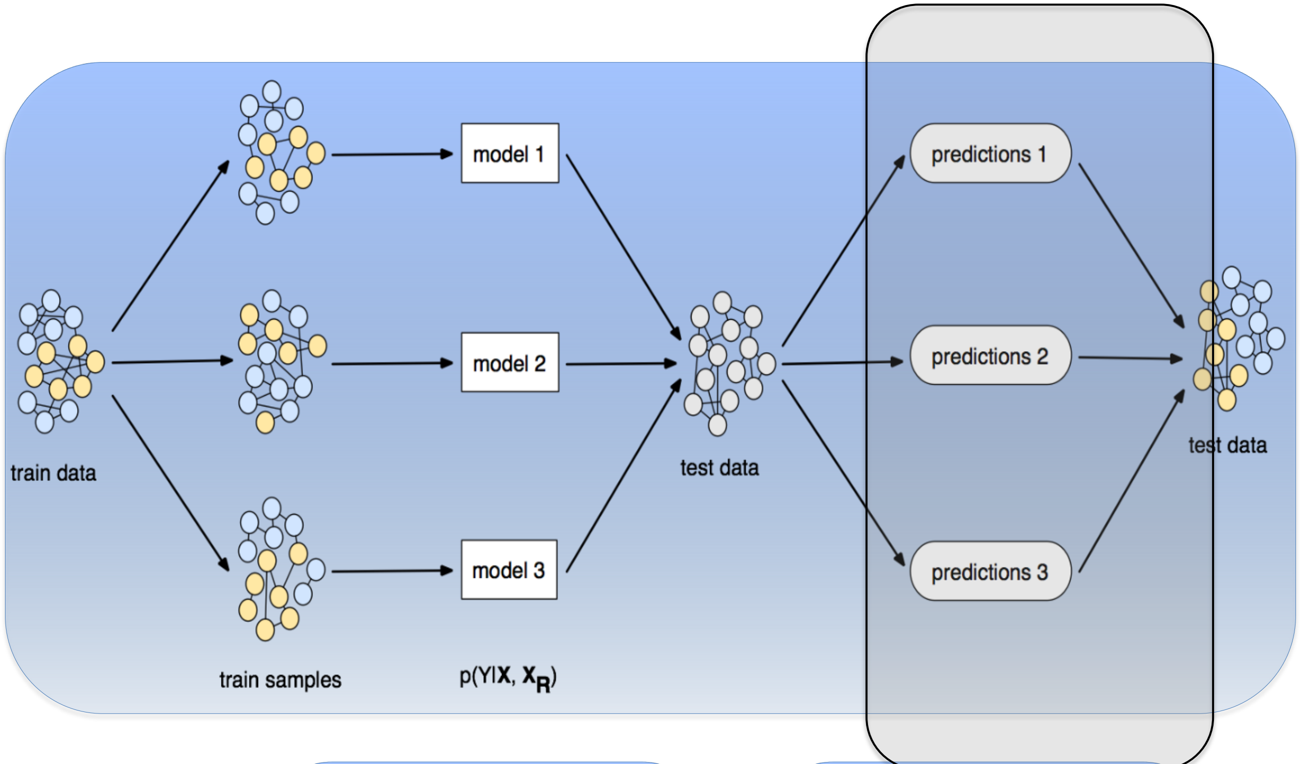
TREATMENT OF INPUT DATA

- *sampling*
- *variable selection*

CHOICE OF BASE CLASSIFIER

- *decision tree*
- *perceptron*
- *…*

Ensemble design



TREATMENT OF INPUT DATA

- *sampling*
- *variable selection*

CHOICE OF BASE CLASSIFIER

- *decision tree*
- *perceptron*
- …

PREDICTION AGGREGATION

- *averaging*
- *weighted vote*
- …

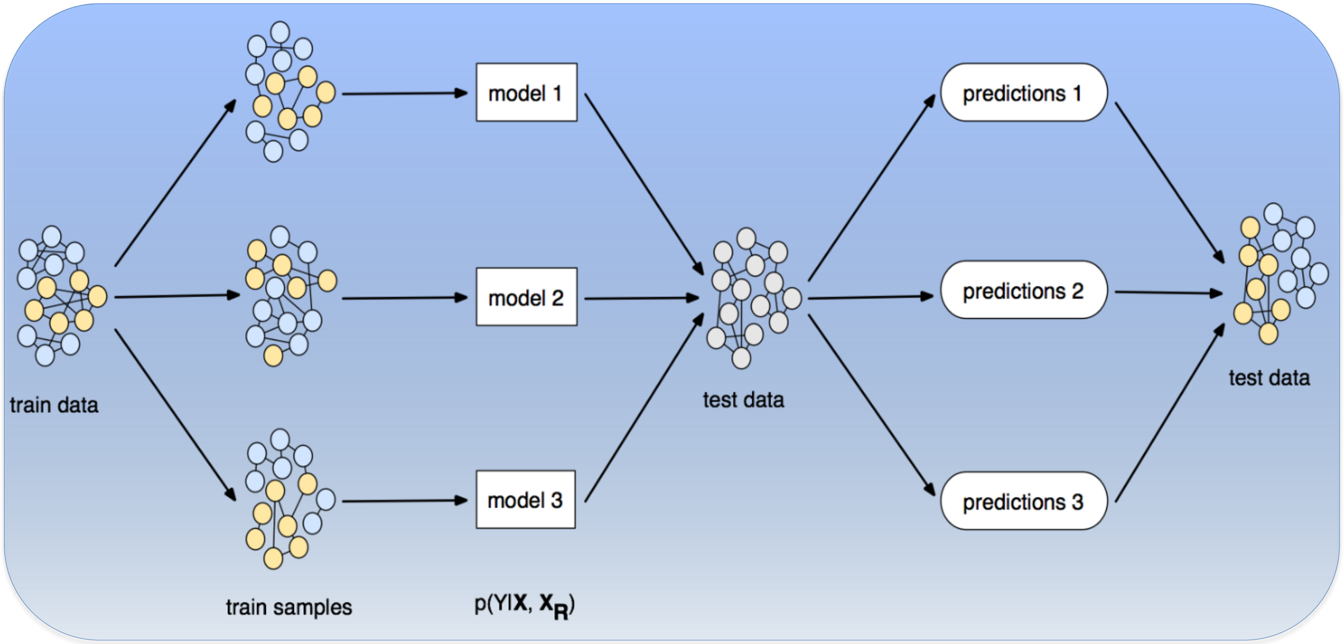
Bootstrap Sampling

- Generate multiple training sets from original training data
    - Sound familiar?
    - *Cross-validation*
- Key distinction: Sample *with replacement*

Bagging

- **B**ootstrap **agg**regat**ing**
- Main assumption
  - Combining many *unstable* predictors in an ensemble produces a *stable* predictor (i.e., reduces variance)
  - Unstable predictor: small changes in training data produces large changes in the model (e.g., trees)
- Model space: non-parametric, can model any function if an appropriate base model is used

Bagging



TREATMENT OF INPUT DATA

- *sample with replacement*

CHOICE OF BASE CLASSIFIER

- *unstable predictor e.g., decision tree*

PREDICTION AGGREGATION

- *averaging*

Bagging

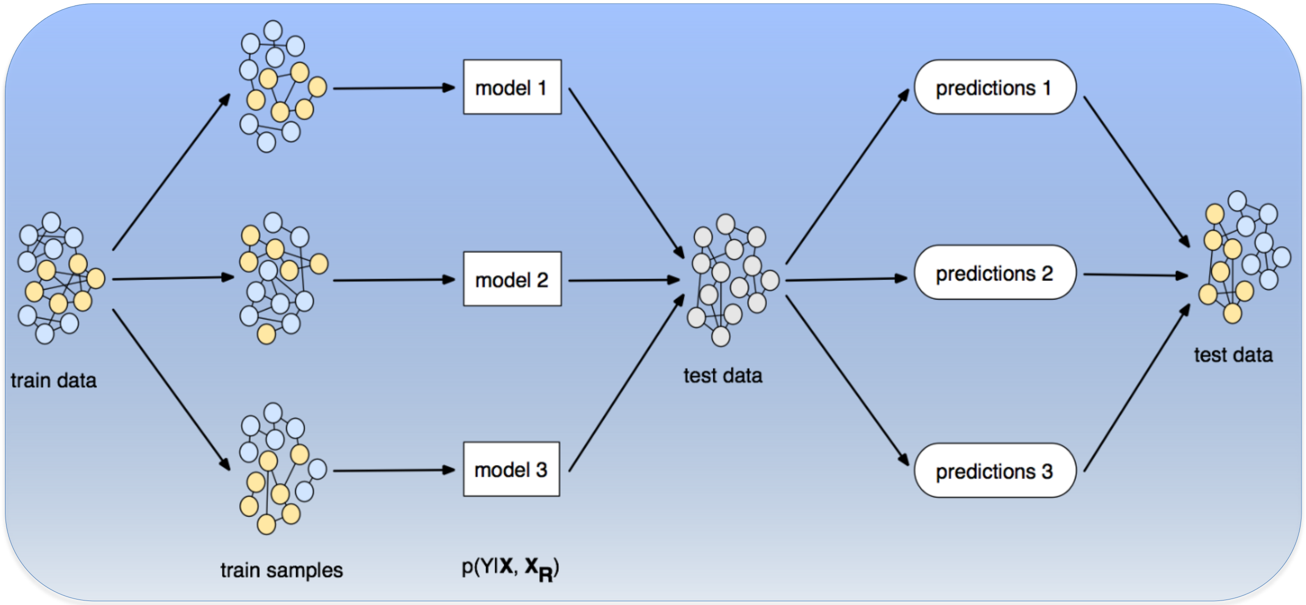
- Given a training data set $D=\{(x_1,y_1),..., (x_N,y_N)\}$
- For m=1:M
  - Obtain a bootstrap sample D_m by drawing N instances ***with replacement*** from D
  - Learn model M_m from D_m
- To classify test instance *t*, apply each model M_m to *t* and use majority prediction or average prediction
- Models have uncorrelated errors due to difference in training sets (each bootstrap sample has ~68% of D)

Sample to create altered training data

Boosting

- Main assumption
  - Combining many *weak* (but stable) predictors in an ensemble produces a *strong* predictor (i.e., reduces bias)
  - Weak predictor: only weakly predicts correct class of instances (e.g., tree stumps, 1-R)
- Model space: non-parametric, can model any function if an appropriate base model is used

Boosting



TREATMENT OF INPUT DATA

- *reweight examples*

CHOICE OF BASE CLASSIFIER

- *weak predictor e.g., decision stump*

PREDICTION AGGREGATION

- *weighted vote*

Boosting: Adaboost

- Assign every example in D an equal weight (1/N)
- For m=1:M
  - Learn model M_m with D_m
  - Calculate the error of M_m and up-weight the examples that are incorrectly classified to form D_{m+1}

Reweight to create altered training data

  - Normalize weights in D_{m+1} to sum to 1
  - Set $\alpha_m = \log((1-err_m)/err_m)$
- To classify test instance *t*, apply each model M_m to *t* and take weighted vote of predictions (ie. using α_m)

Adaboost Algorithm

- Takes training data (x_i, y_i) (y -1 or 1), weights w_i
  - Initialize weights to $1/n$
- For $m=1..M$
  - Learn classifier f_m
    - $Error_m = \sum_{i=1}^{n} w_i^m \mathbf{I}\{f_{m(x_i)} \neq y_i\}$
  - Computer classifier coefficient $\alpha_m = \frac{1}{2}\log\frac{1-Error_m}{Error_m}$
  - Update weights $w_i^{m+1} = \frac{w_i^m \exp(-\alpha_m y_i f_m(x_i))}{\sum_{j=1}^{n} w_j^m \exp(-\alpha_m y_i f_m(x_i))}$
- Final classifier $f^*(x) = \operatorname{sign}(\sum_{m=1}^{M} \alpha_m f_m(x))$

Boosting Caveats

- While theoretically sound, Adaboost not that robust to noisy labels
  - Weights of mislabeled data grow until classifier fits the noise
- **Must** use *weak* classifiers
  - Otherwise easily overfits training data

Random Forests

- Problem: Decision Trees prone to overfitting
- Solution: Decision tree on fewer features
- Ensemble idea
    - Randomly select subsets of features
    - Choose best candidate split from just within subset
- Algorithm the same as standard decision tree, except instead of applying information gain / gini index / …, first randomly select subset, then apply
    - All features (except the one use) passed to the next level

Ensemble summary

- Two approaches for Ensemble learning:
  - Boosting – reduce bias
  - Bagging – reduce variance
- Applicable in different situations