Ensemble Methods

BUAN 6341 - APPLIED MACHINE LEARNING

LECTURE 07

MUHAMMAD FAROOQ SABIR

Summary

Decision Trees

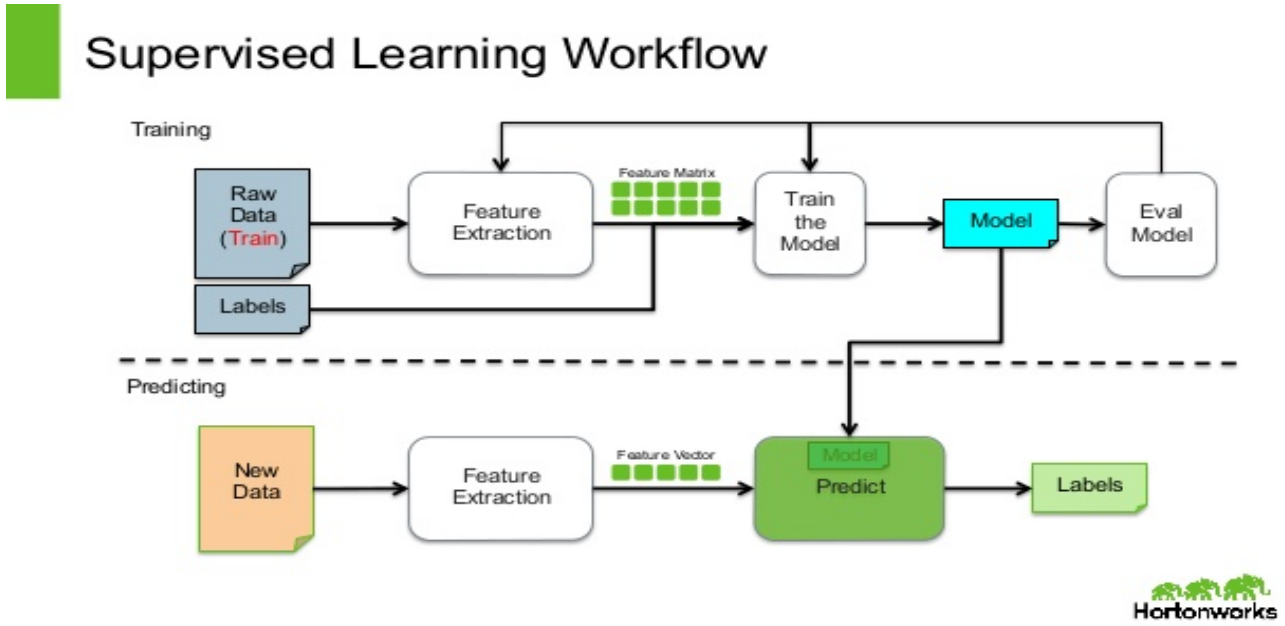
- What are decision trees
- When are decision trees useful
- ID3 algorithm
- Tennis example using ID3
- Best attribute selection:
  - Entropy, information gain
- Limitations and extensions to ID3

Ensemble Learners

Supervised learning

- Search through hypotheses space to find a suitable hypothesis
- Make good predictions on the data set and generalize well

May be difficult to find a good hypothesis



Ensemble Learning - Background

Michael Kearns and Leslie Valiant posed the following question in 1988:

- Can a set of weak learners be combined to create a single strong learner?
- One answer came in 2009
- In 2006, Netflix offered a $ 1 Million prize for coming up with an algorithm that performs 10% better than their algorithm at predicting which movies their customers would like to see
- The prize was awarded in 2009
- Winning algorithm was not a single algorithm but an ensemble of algorithms

Ensemble Learning

Example: Consider email classification task into spam and not spam

Simple rules:

- Spouse: -
- Short: +
- URL: +
- Just an image: +
- Employer: -

The main idea behind ensemble learning is to learn from many simple rules and then combine to come up with a complex rule

Ensemble Learning

Learn over a subset of data and generate some kind of basic rule

Learn over another subset of data to generate another basic rule

Keep doing it for many subsets and rules

Then you take all of these simple rules and combine them into a complex rule

Important: Use only a subset of data for each of these simple rules

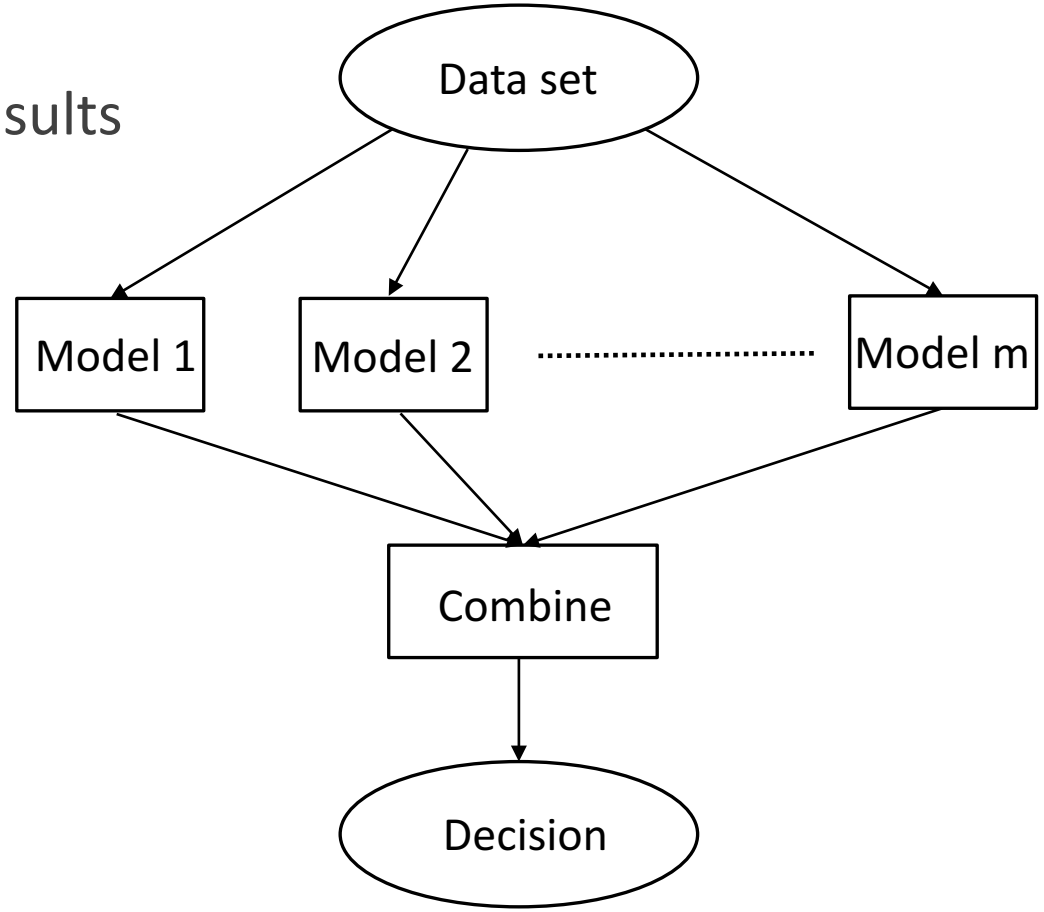
Simple rules:

- Spouse: -
- Short: +
- URL: +
- Just an image: +
- Employer: -

Ensemble Learners

Main idea:

- Use multiple learners to produce better results than what can be achieved using a single learner
- Supervised learning*
- Finite number of models



*Can be used in unsupervised learning as well

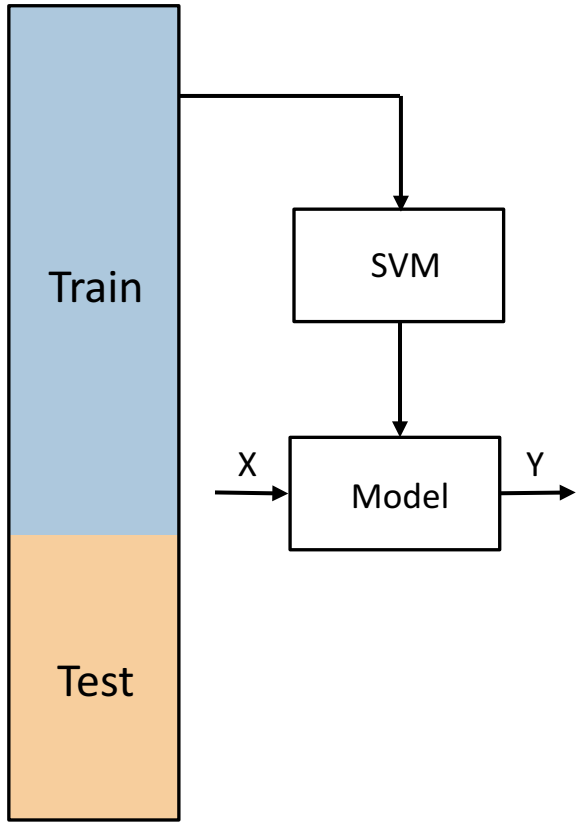
Ensemble Learners

Creating an ensemble of learners is one way to make your learners better

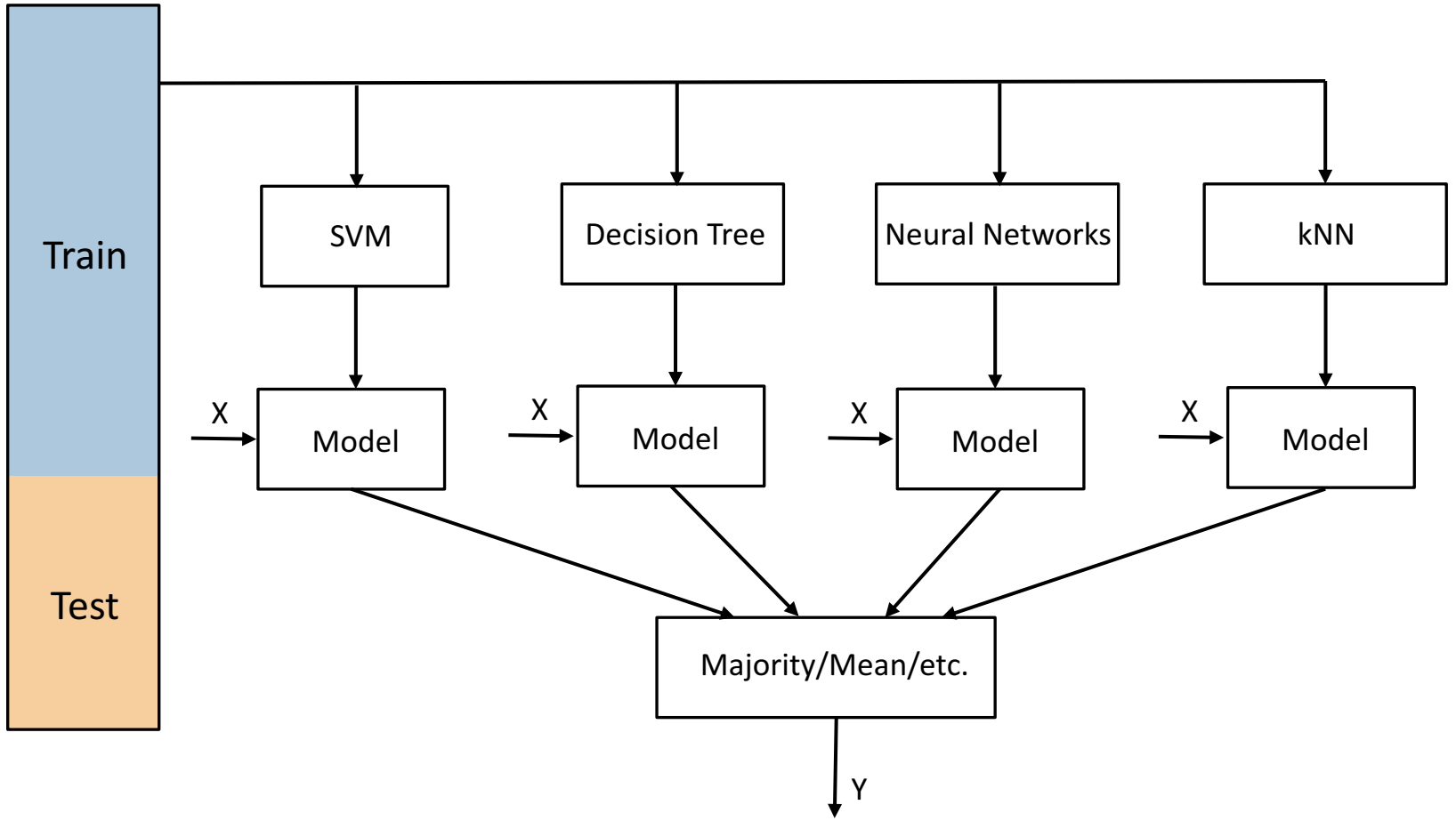
No new algorithms are created

- Several different models are assembled together

Can easily replace your model with an ensemble (same input/output)



Ensemble Learners



Properties of Ensemble Learners

Ensemble method is itself a supervised learning method

- Trained on train set
- Makes predictions

The final hypothesis represents a single hypothesis

- It may not be contained within the hypothesis space of the models it is built with

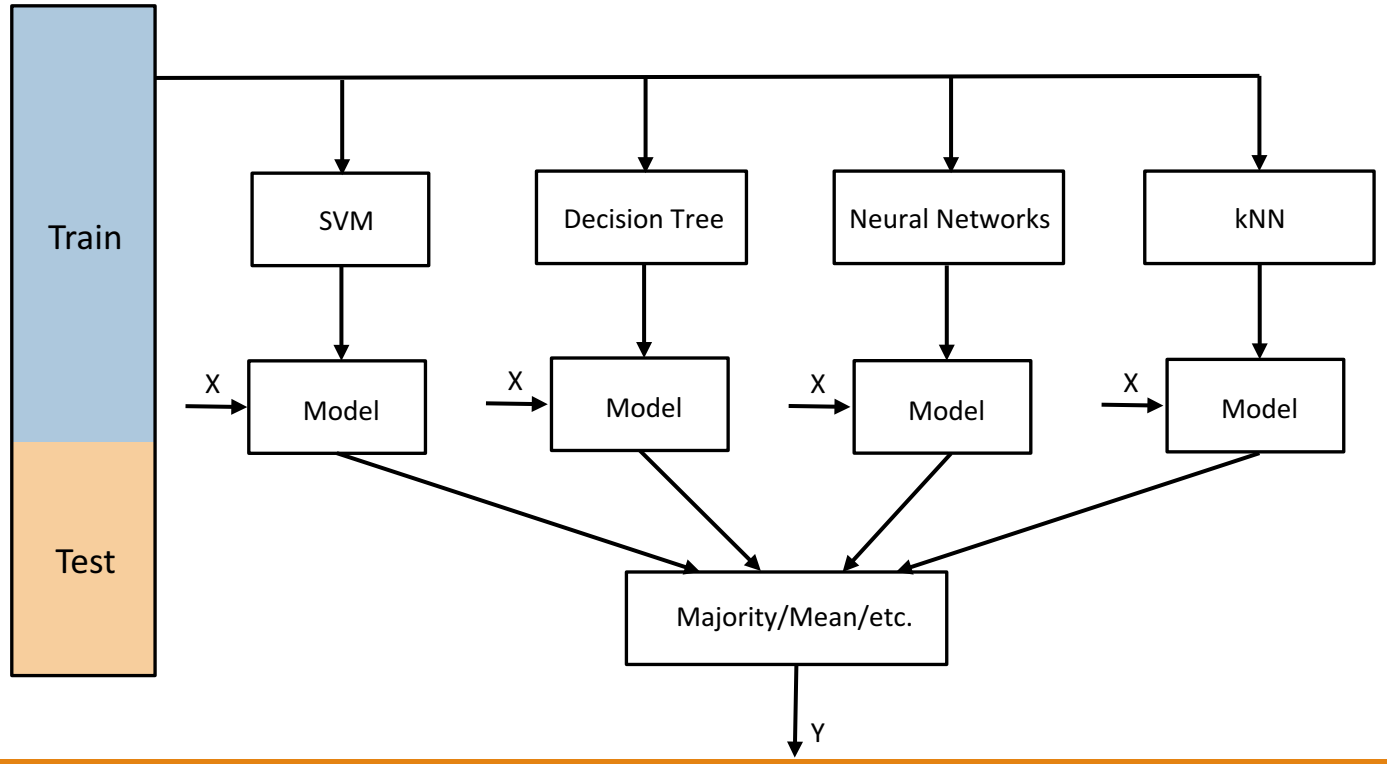
Ensemble methods show more flexibility in the functions they can represent

Properties of Ensemble Learners

Lower error rates than individual models

Less overfitting

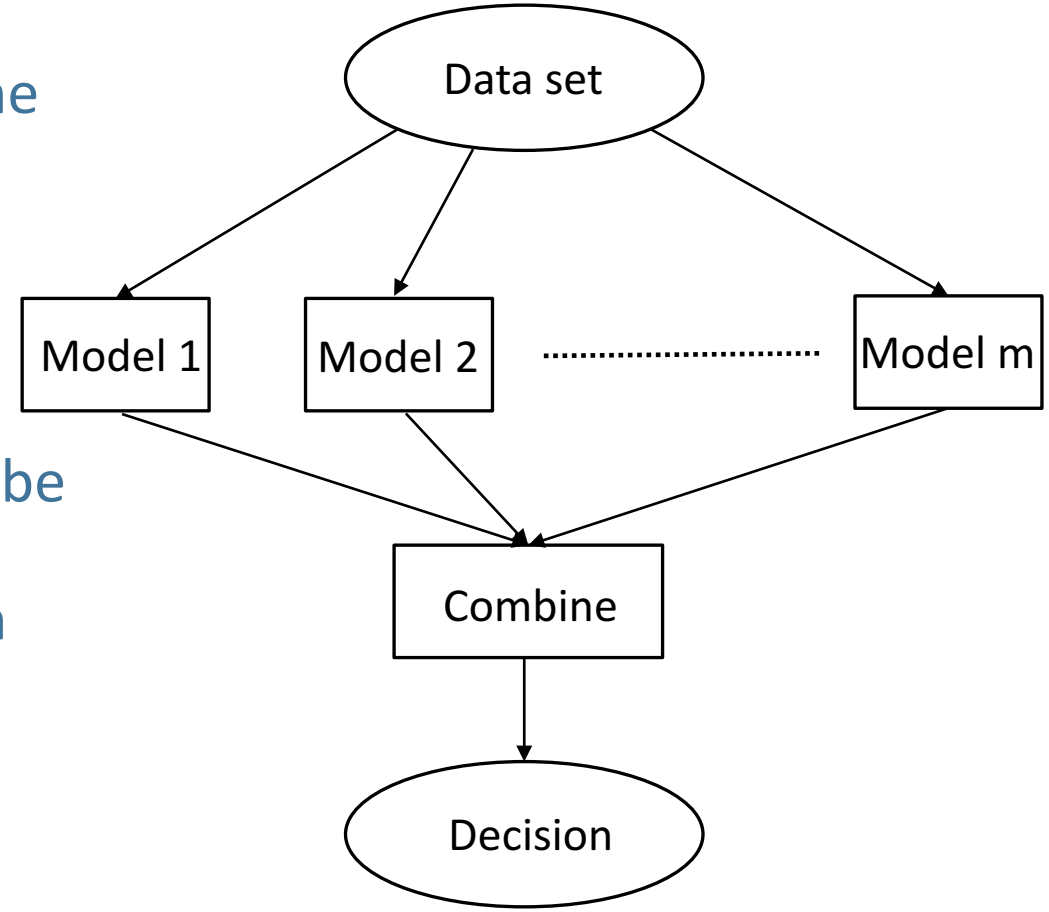
Overall less bias than each individual model's bias



Properties of Ensemble Learners

Typically, ensemble methods tend to produce results when there is more diversity among the underlying models

Random algorithms (e.g. random forests) can be used to produce stronger ensemble than very deliberate algorithms (entropy based decision trees)



Properties of Ensemble Learners

How many models or ensembles?

- No good answer – experimentation
  - Learning curves may be used to find best performance

Evaluation

- Ensemble methods require more computation for evaluating predictions as compared to single learners
- Training many learners may require large amount of time/computation
  - But each individual learner may take much less time/computation
  - e.g. decision trees with fixed depth

Types of Ensemble Learning Methods

Bagging

- Also known as bootstrap aggregating
- Random sampling of train data
- Random forests

Boosting

- Improves on previous misclassifications
- AdaBoost

Stacking

- Final model trained on outputs of various learners

Bagging (Bootstrap Aggregating)

One of the easiest and most common methods to build an ensemble learner

Same learning algorithm is used many times

- Each learner is trained on a different set of data
- Invented by Leo Breiman in 1996 (Bagging Predictors)

Bagging

Consider a data set consisting of n samples

Let's make m "bags" from this data set

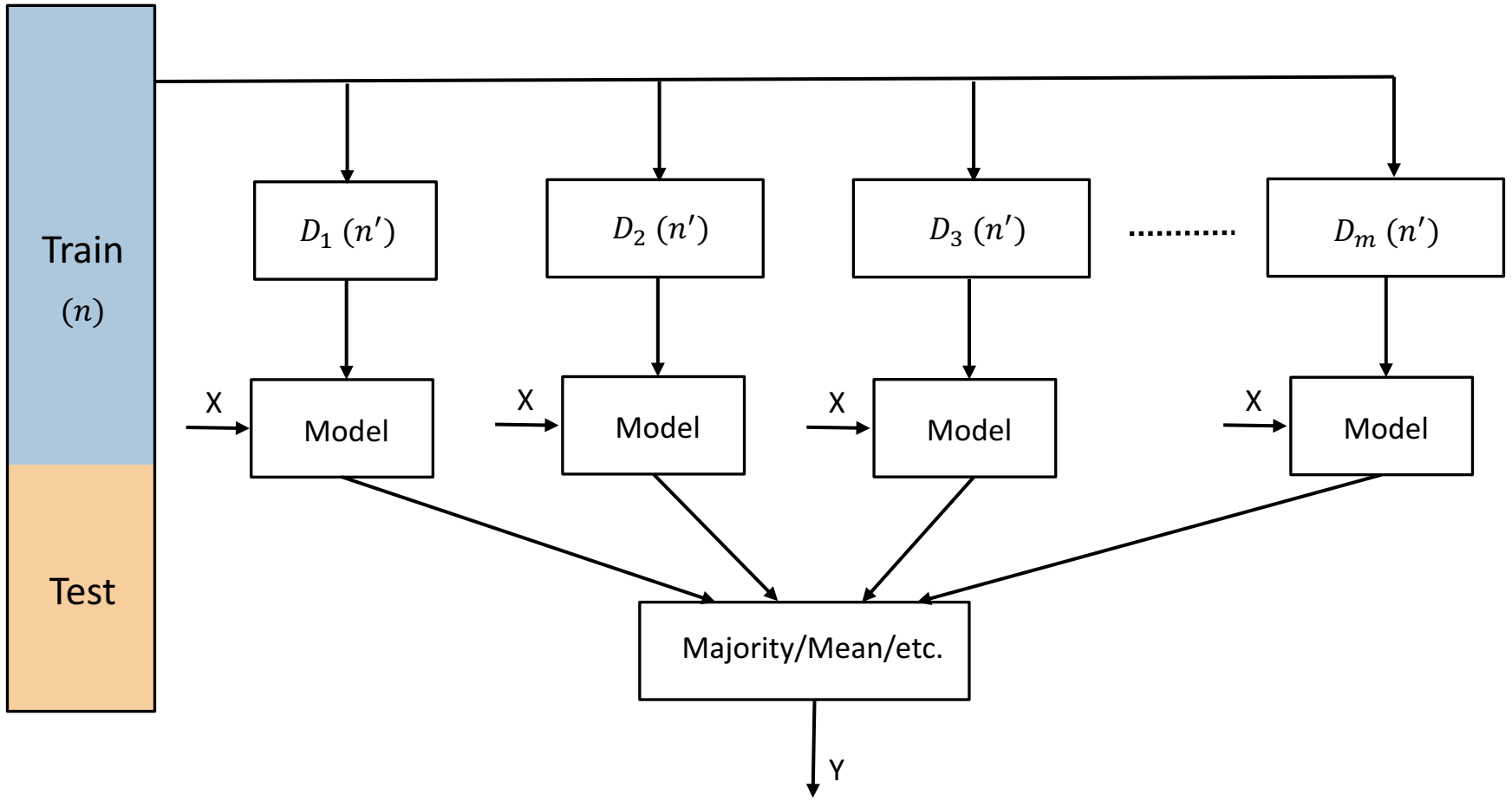
- Each bag consists of n' samples ($D_1, D_2, \ldots D_m$)
- n' <= n
- Bags are constructed using sampling with replacement

Train a model on each of these data sets

- Usually same algorithm

The output is the mean (or majority, etc.) of all the models' outputs

Bagging


Train
(n)
Test
$D_1\ (n')$
$D_2\ (n')$
$D_3\ (n')$
.....
$D_m\ (n')$
X
Model
X
Model
X
Model
X
Model
Majority/Mean/etc.
Y

Properties of Bagging

Very easy to understand and implement

Increases the stability and accuracy of our learning problem

Reduces variance and helps to avoid overfitting

Most commonly used with decision trees

Can be used with any classification or regression method

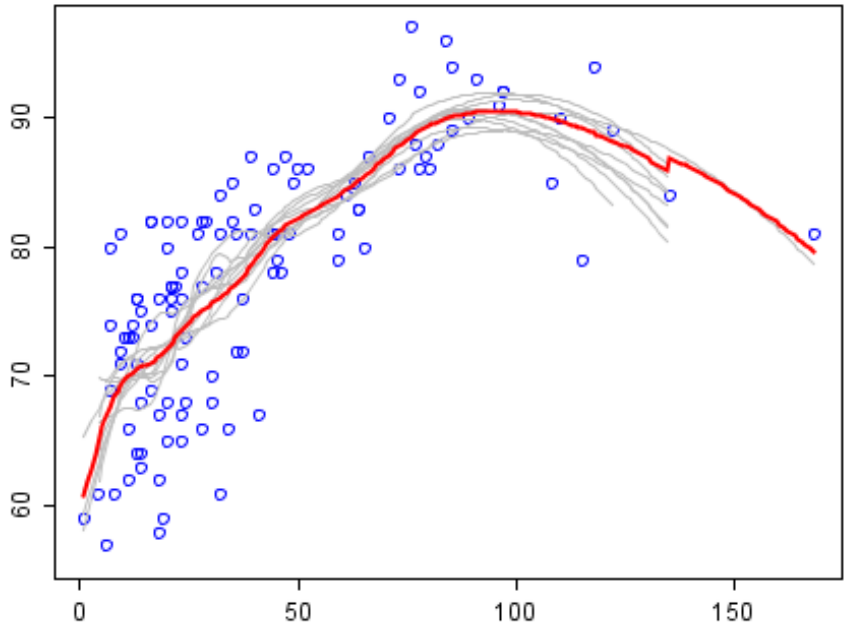
Bagging Example

Each gray curve represents output of an individual bag

Red curve is the mean of the gray outputs

Mean clearly performs better than individual models

- Less overfitting



Boosting

Also based on the question posed by Kearns and Valiant

- Can a set of weak learners be combined to create a single strong learner?

Common boosting algorithms iteratively learn weak classifiers and add them to form a strong classifier

In each step the examples that are misclassified gain weight and those that are classified correctly lose weight

Boosting Algorithms

Common boosting algorithms

- AdaBoost – most popular
- LPBOOST
- TotalBoost
- BrownBoost
- Xgboost – also very popular
- MadaBoost
- LogitBoost

The main difference in various boosting algorithms is how they weigh training data points and hypotheses

Bagging and Boosting

In bagging

- We uniformly randomly sample data to form subsets of data
- Learn over each subset using some learner
- Use some criterion to select the final outcome
  - Mean
  - Most popular (majority)

In boosting

- Instead of randomly choosing the data
- We should take advantage of what we are learning as we go along
  - Pick examples that our model is not performing good on
  - This means pick subset of data with hard examples

Boosting

By hard examples we mean pick the examples that are relatively hard to classify compared to other examples

Example: In the email classification example

- We have learnt rules that classify most of the emails very good but do a bad job on some
- Why do we want to come up with more rules that do a good job on the examples that we are already doing good at?
- Devise rules that focus on examples that we are doing bad at
- This is the main idea behind boosting – find the hard examples

Boosting – Final Hypothesis

Suppose we know how to pick the data subsets for our learners

How do we combine the outputs of these learners

- Weighted mean
- Weighted voting

Weighting is important so that the learner does not lose track of the examples that it has already mastered

Boosting - Error

We compute error rates in our learning algorithms usually as

- Mean squared error
- Classification error (number of mismatches)

Here we define our error over some distribution of samples $\mathcal{D}$ as the probability that our hypothesis will disagree with the true concept

$$\text{Error} = \Pr_{\mathcal{D}}\left[h(x) \neq c(x)\right]$$

- $h(x)$ is the hypothesis
- $\mathrm{c}(x)$ is our true target/concept function

Boosting - Error

How is this different than using classification error rate?

Suppose we have four examples

- Using our learner, we have classified two correctly and two incorrectly
- The error rate is ½

Now suppose the four examples are likely to be seen in different proportions (1/2, 1/20, 4/10, 1/20)

- Correctly classified examples: 1/2, 4/10
- Incorrectly classified examples: 1/20, 1/20
- Now the error rate is: 1/10

Boosting - Error

This example shows that even though we may classify many examples wrong, some examples are more important than others because some are very rare

Therefore it is important to think of error as not the number of distinct mistakes our classifier will make but as the amount of time it will be wrong

- Think of underlying distribution of data

Hard Examples

This notion of error over a distribution is very important for boosting

Our learners will be passed a distribution over the examples

This will tell the learners which examples are important to learn and which are not as important

This way we will try to make our learner to get the harder examples more important to get right than the ones we already know how to solve

- Harder examples will be picked more often by sampling

Weak Learner

Weak learner

- A learner that always performs better than chance no matter what the distribution of the data is
- Error rate is always less than ½
- We can think of it as a learner that is always able to learn something from the data

Boosting Algorithm

Given training data set $\{(x_i , y_i)\}$

y_i are in {-1, +1}

For t = 1 to T

- Construct distribution $\mathcal{D}_t$
- Given the distribution $\mathcal{D}_t$,
  - Find weak classifier $h_t(x)$ with error defined as

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

  - The error ε_t has to be less than ½ (doesn't have to be very small)
- Output H_{final}

Boosting - Distribution

How to construct the distribution $\mathcal{D}_t$

Let us suppose at time t=1 the distribution is uniform

$$\mathcal{D}_1(i) = \frac{1}{n}$$

- Where n is the number of examples
- We have no reason to believe at the beginning that for any given algorithm one example is better or more important or harder than the other examples

Boosting - Distribution

$$\mathcal{D}_1(i) = \frac{1}{n}$$

At every time step t+1, we will construct a new distribution that is given by

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{Z_t}$$

where

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

Boosting - Distribution

$\mathcal{D}_t(i)$ at any given time t tells us how important example i is

This equation is making $\mathcal{D}_t(i)$ bigger or smaller at each step based on how well the current hypothesis performs on this particular example i

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

Boosting - Distribution

We know that

- Our labels y_i are {-1,1}
- Also, our classification outputs $h_t(x_i)$ are {-1,1}

$\propto_t$ is a positive number

- Weak learner: $\varepsilon_t < ½$

z_t is a normalization factor to make the distribution sum to 1 at any given time

So at time $t + 1$, what happens to the distribution?

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

Boosting - Distribution

For the examples for which the label and hypothesis output agree

- $y_i h_t(x_i) = 1$
- $e^{-\propto_t y_i h_t(x_i)} < 1$
  - as long as there is at least one example where the label and hypothesis disagree
- This means that
  - $\mathcal{D}_{t+1}(i) < \mathcal{D}_t(i)$

The algorithm puts less weight on the examples that have correct classification

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

Boosting - Distribution

For the examples for which the label and hypothesis output disagree

- $y_i h_t(x_i)$ = -1
- $e^{-\propto_t y_i h_t(x_i)} > 1$
  - as long as there is at least one example where the label and hypothesis agree
- This means that
  - $\mathcal{D}_{t+1}(i) > \mathcal{D}_t(i)$

The algorithm puts more weight on the examples that have incorrect classification

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

Boosting - Distribution

These equations connect constructing the distribution $\mathcal{D}_t$ at each step to the notion of hardest examples

- More weight on examples that are harder and hence classified incorrectly during the previous iteration
- Less weight on examples that are easier and hence classified correctly during the previous iteration

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t} [h_t(x_i) \neq y_i]$$

Boosting – Final Hypothesis

So how do we get to the final hypothesis H_{final}?

- H_{final} is the sign of the weighted sum of the learner outputs at each iteration
- Weighted by $\propto_t$

$$H_{final}(x_i) = \text{sgn}\left(\sum_t \propto_t h_t(x_i)\right)$$

$$\mathcal{D}_1(i) = \frac{1}{n}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \cdot e^{-\propto_t y_i h_t(x_i)}}{z_t}$$

$$\propto_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$$

$$\varepsilon_t = \Pr_{\mathcal{D}_t}[h_t(x_i) \neq y_i]$$

Boosting – Final Hypothesis

$$H_{final}(x_i) = \text{sgn}\left(\sum_t \propto_t h_t(x_i)\right)$$

- We are using a thresholding function here to decide the final output
- We are weighing the learners with low error rate higher and learners with high error rates lower
  - Basically saying that if a learner performs well then it gets a bigger say in the final output and vice versa
- This way we don't lose track of examples that we classify correctly in each step

Boosting Example

Consider this binary classification problem

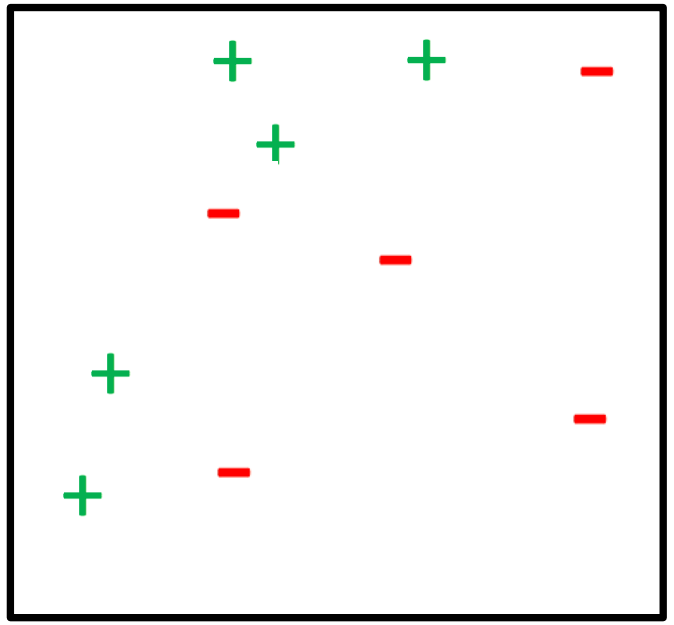
- 5 positive and 5 negative examples

Let our hypothesis space be axis aligned semi planes

- Horizontal or vertical lines only

Initially the distribution is uniform

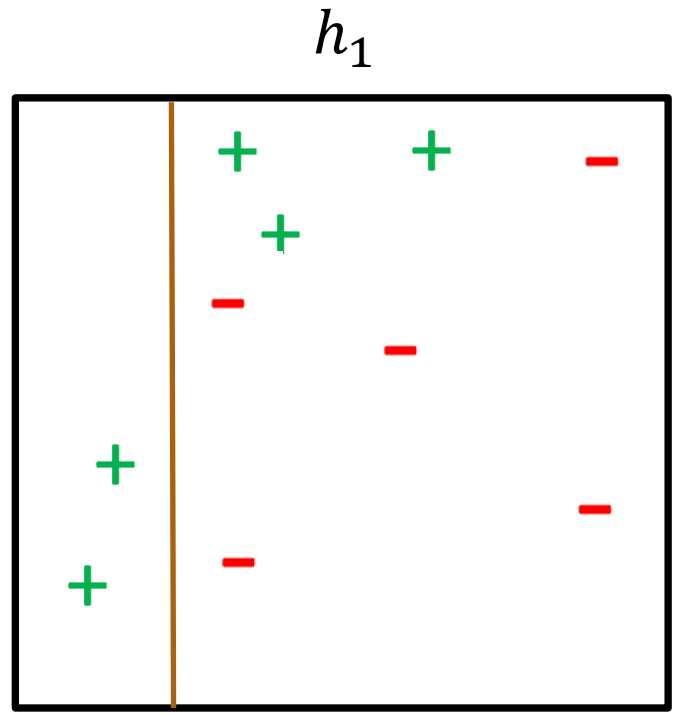
- Each example equally likely



Boosting Example

First iteration:

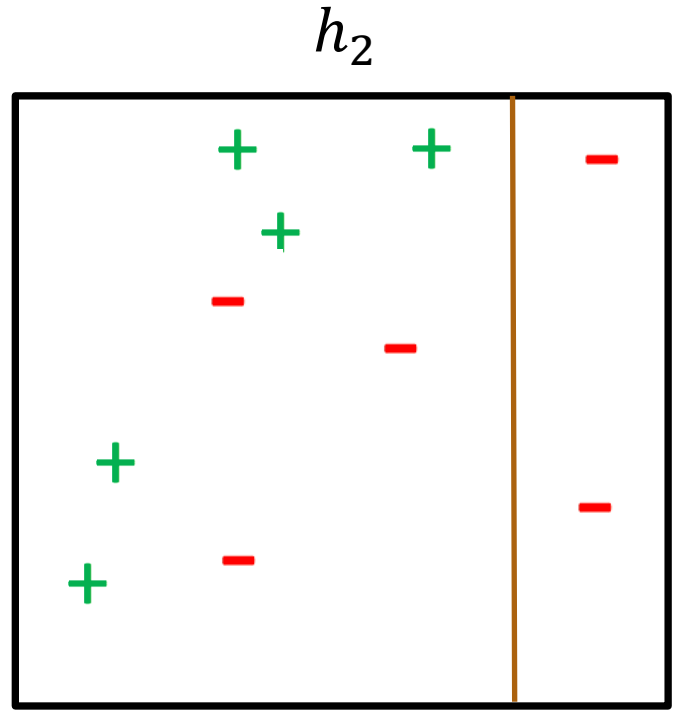
- Our learning draws this decision boundary
- 2/5 positive examples classified correctly
- All negative examples classified correctly
- $\varepsilon_t = 0.3$
- $\propto_t = 0.42$



Boosting Example

Second iteration:

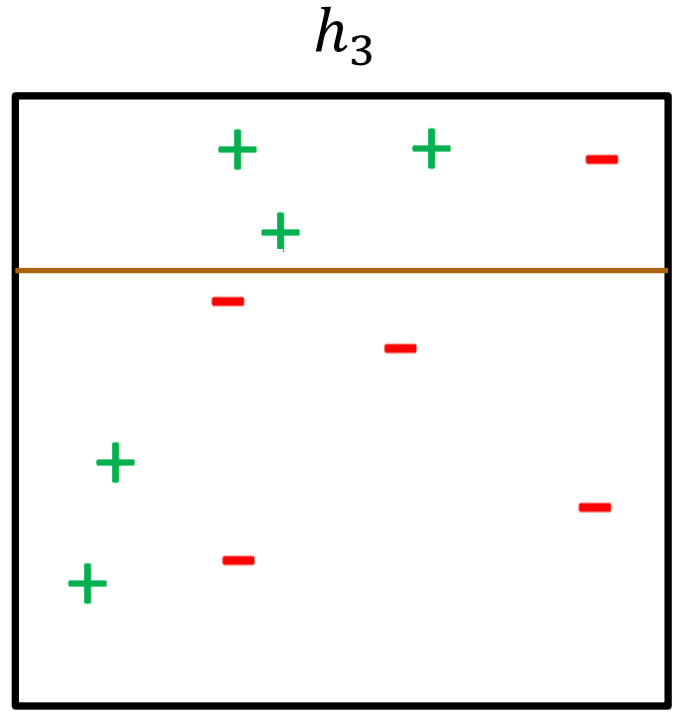
- Top three positive examples get higher weight since they were incorrectly classified in the previous iteration
- Remaining examples get lower weight since they were correctly classified previously
- Decision boundary is drawn such that all positive examples get classified correctly
- 2/5 negative examples get classified correctly
- $\varepsilon_t = 0.21$
- $\propto_t = 0.65$



Boosting Example

Third iteration:

- The three negative examples misclassified in previous iteration get highest weight
- The top three positive examples get lower weight but still higher than the other two positive examples
- Decision boundary is drawn below the top three positive examples
- 2/5 positive and 1/5 negative examples classified incorrectly
- $\varepsilon_t = 0.14$
- $\propto_t = 0.92$



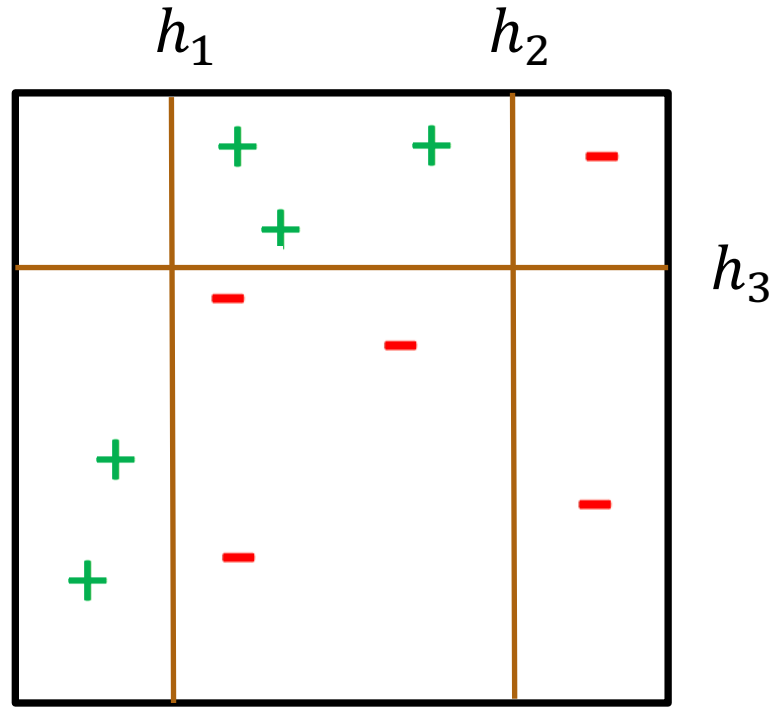
Boosting Example

Notice that error rate ε_t goes down with each iteration

- Our learner is getting better

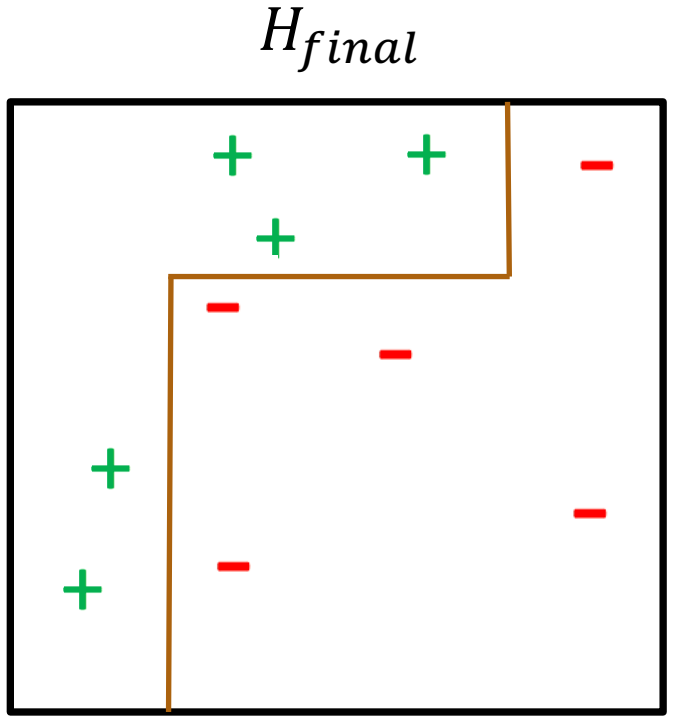
To get the final hypothesis, we need to weigh, add and threshold using

$$H_{final}(x_i) = \text{sgn}\left(\sum_t \propto_t h_t(x_i)\right)$$



Boosting Example

$$H_{final}(x_i) = \text{sgn}\left(\sum_t \propto_t h_t(x_i)\right)$$



After weighing, adding and thresholding, we get this decision boundary as our final hypothesis

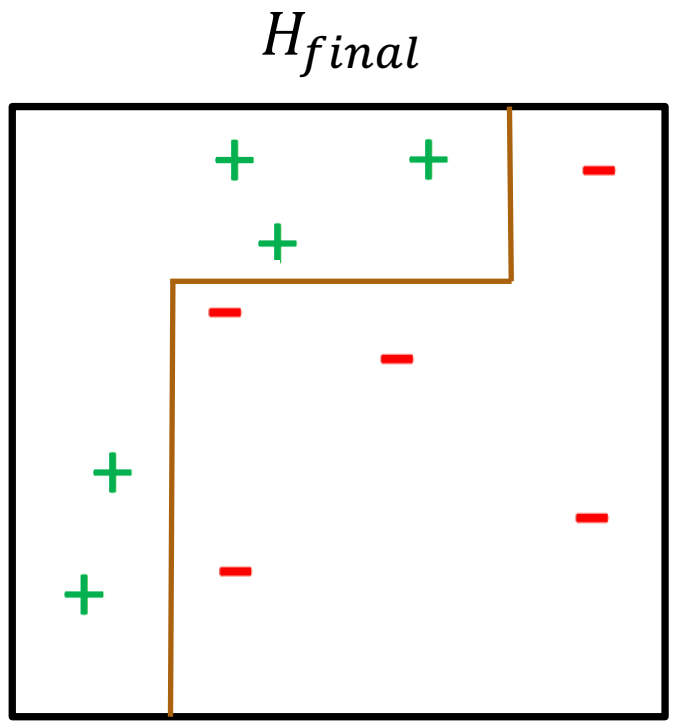
We can make such shapes with many algorithms, such as decision trees, neural networks, kNN, etc.

Boosting Example

We used three simple lines/planes to construct a more complex shaped decision boundary

This is a general feature of ensemble methods

- The hypothesis space contains simple hypotheses (e.g. lines)
- We can combine them in simple ways (in this case weighted sum) to get a more complicated hypothesis
  - Higher flexibility in the final hypothesis than individual hypotheses



Why Does Boosting Do Well?

Main idea: If there are some examples which I am bad at classifying, I am going to re-rate all the examples such that the ones that are classified incorrectly become increasingly important

In future iterations we are forced to make our weak learner classify these previously incorrect examples correctly

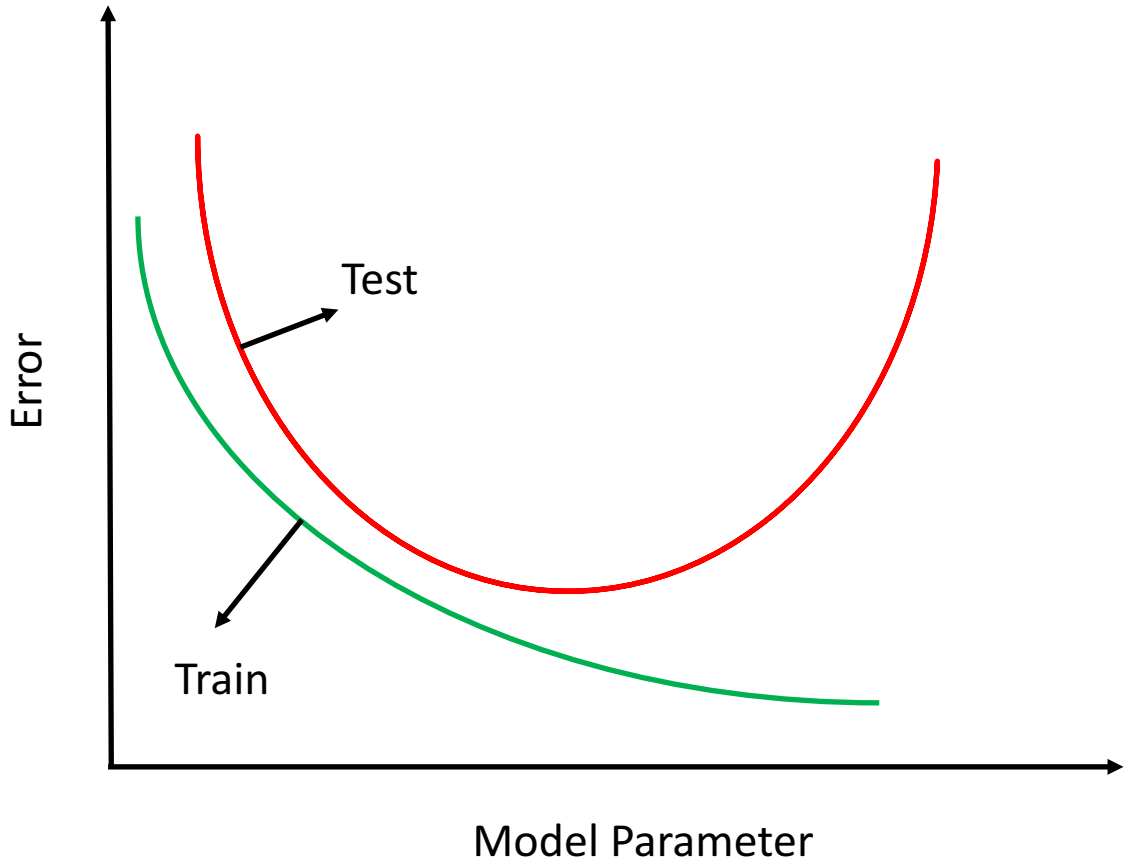
In the end we weigh our individual classifiers based on their ability to classify the examples correctly

Number of Learners

Typically the learning curve for any algorithm looks like this for all parameters

Training error keeps going down

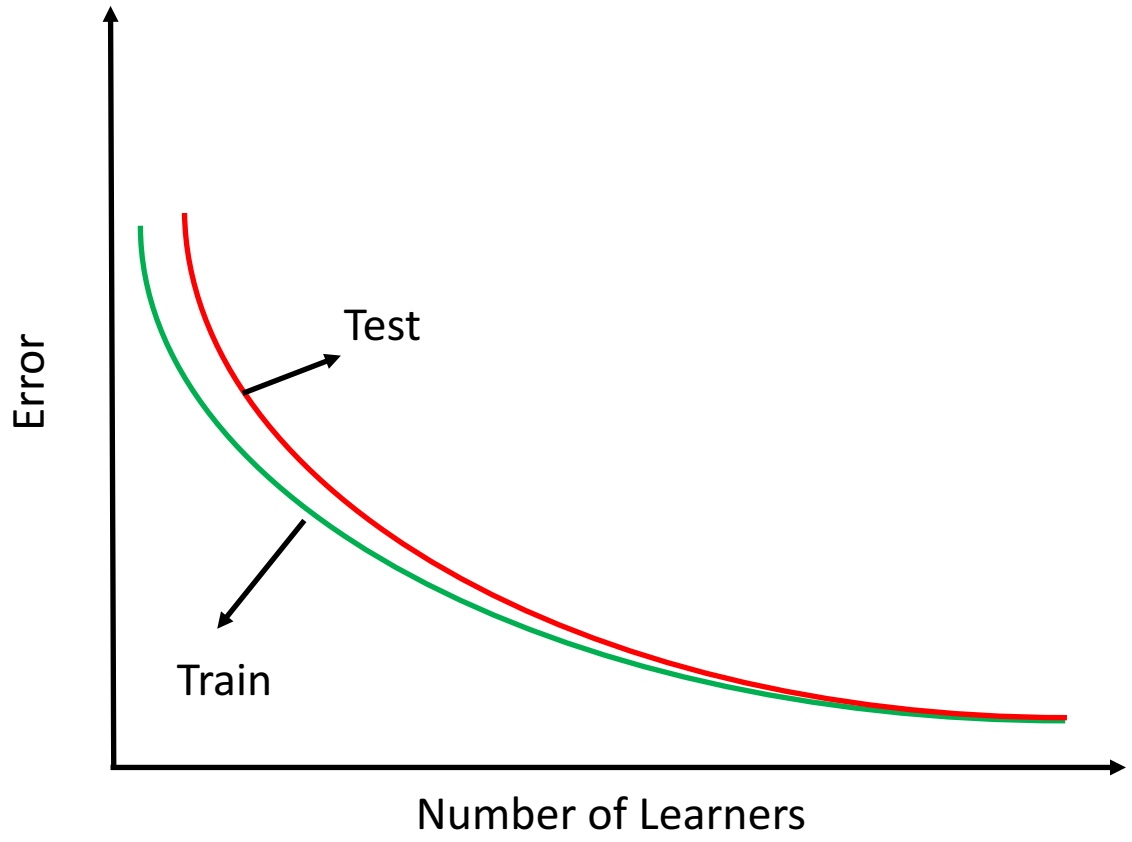
Test error goes down till a point and then starts to increase



Number of Learners

With boosting, test error also continues to decrease like train error as a function of number of learners

Is this too good to be true?



Number of Learners

Why does test error continue to decrease?

How we make the distribution at each iteration and then classify

- When we use more and more learners, not only the error goes down, but also the learners' confidence in decisions goes up
- This in effect causes the points to move away from decision boundary as the number of learners increase
- This results in test error to decrease like train error

Stacking

Another common ensemble learning method

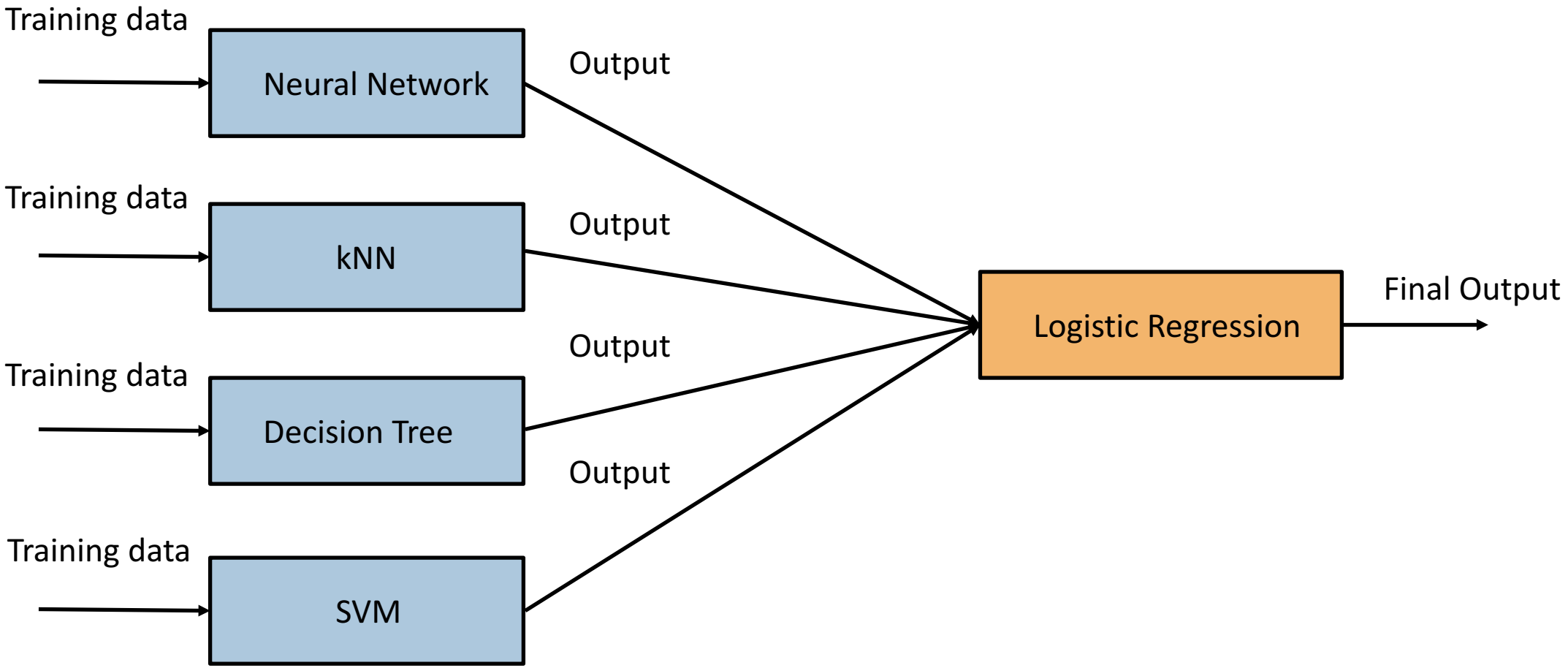
Different than bagging and boosting

Multiple single classifiers are used (bagging and boosting use the same learner)

Example: Many classification algorithms such as decision trees, neural networks, support vector machines, etc. can be used together

A combiner algorithm is used to predict the final outcome using the outcomes of other learners

Stacking


Training data
Neural Network
Output
Training data
kNN
Output
Training data
Decision Tree
Output
Training data
SVM
Output
Logistic Regression
Final Output

Summary

Ensemble Methods

- Bagging
- Boosting
  - Iterative improvement
  - Constructing the distribution
  - Final hypothesis: thresholded weighted mean
- Stacking

References

An Introduction to Statistical Learning – James, Witten, Hastie, Tibshirani

Machine Learning – Tom Mitchell

The Elements of Statistical Learning – Hastie, Tibshirani, Friedman

Charles Isbell and Michael Littman – Machine Learning (Udacity)

Tucker Balch – Machine Learning for Trading (Udacity)