Ensemble Learning

Bagging, Random Forests, Boosting

build diff experts, let them vote, and predict, at the end, combine results and figure

often perform predictive performance than a hypo by itself, as now have a higher population of them - also makes it harder to know what it is doing as lots more hypo to track

Combining Multiple Models

Basic idea:

- build different “experts”, let them vote

Advantage:

- often improves predictive performance

Disadvantage:

- usually produces output that is very hard to analyze
- but: there are approaches that aim to produce a single comprehensible structure

Majority voting

accuracy = probs of being correct
all right 0.7^5 + [0.7^4 * 0.3] + [0.7^3 * 0.3^2] roughly >= 0.8

which is significantly larger than 0.7

- Idea: collect a number of independent classifiers where every classifier has an accuracy higher than 0.5; use the majority vote of the collection (ensemble).
- Example: suppose we have 5 completely independent classifiers each having 70% accuracy. What is the accuracy of the majority vote?

each classifier have an accuracy of 0.5 in a balance dataset. majority voting, emsemble will be better than 0.5. so larger the number of hypo, the higher the accuracy

0.3 ^ 5, is prob of all the 5 classifiers have the wrong prediction

so 1 - (0.3^5)

binomial distrubition

Bagging equal weight - treat all equally

The idea

- Given a collection of hypotheses (classification/regression models):
  - Combine predictions by voting/averaging
  - Each model receives equal weight
- Ideal version:
  - Sample several training sets of size n
    (instead of just having one training set of size n)
  - Build a classifier for each training set
  - Create a wrapper that combines the classifiers' predictions

Bagging

dataset of size n, sample

Challenges

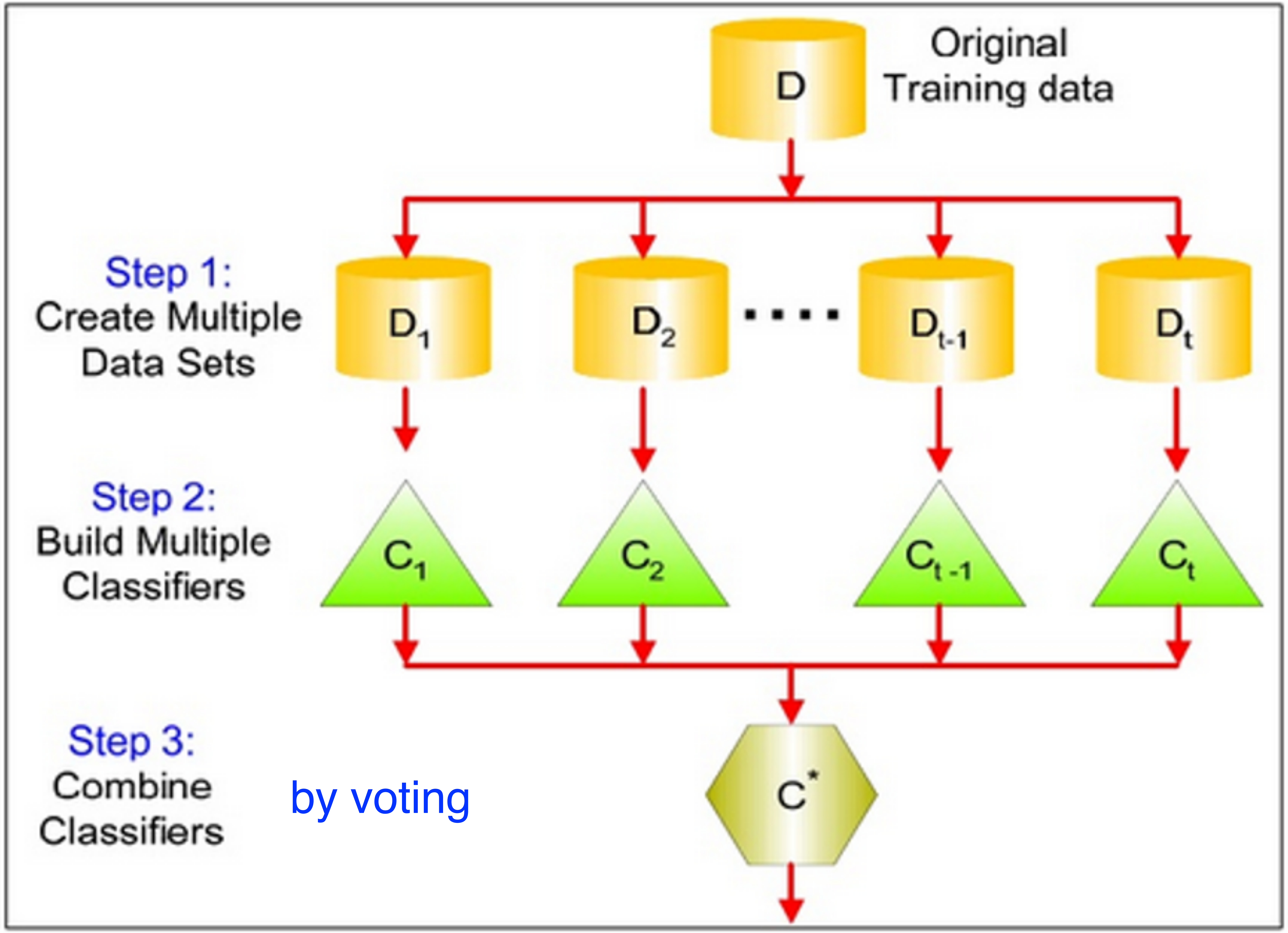
Problem: we only have one dataset!

Solution: generate new ones of size *n* by sampling from it with replacement

- Can help if data is produced by a stochastic process

Other solutions:

- Use a learning algorithm that is stochastic (same training data can yield different models)
- Use an *unstable* learner: small change in training data can make big change in the produced model


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2:
Build Multiple
Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3:
Combine
Classifiers
by voting
C^*

Conditions independent and diverse

- Training the same classifier on the same training data several times would give the same result for most machine learning algorithms
  - Combining these classifiers would make no sense
  - Exception: methods where the training involves some randomness
- Classifier combination gives the best result if
  - The classifier outputs for the same input are diverse
  - The classifiers operate independently (or at least partially independently)
  - The classifiers have some unique or “local” knowledge
    - E.g. they are trained on different (or at least partially different) training data sets
- We also need some formalism to combine (aggregate) the opinions of the various classifiers

How to produce diverse classifiers?

- We can combine different learning algorithms ("hybridization")
  - E.g. we can train a *GMM*, an *SVM*, a *k-NN,…* over the same data, and then combine their output
- We can combine the same randomised learning algorithm trained several times over the same data
  - This works only if there is some random factor in the training method
  - example: neural networks trained with different random initial weights (and converging to different values each time)
- We can combine the same learning algorithm trained over different subsets of the training data
  - We can also try using different subsets of the features
  - Or different subsets of the target classes (multi-class task, lot of classes)

Randomisation of decision trees

creatie diversity with randomisation. sampling data with random subspaces
algorithm randomisation to introduce diff max depth, greediness etc.

- The decision tree is a very popular choice for combination-based methods.
  - Small decision trees cannot create complex classifications.
  - But their training is very simple and fast, so it is very easy to create an ensemble learner from a huge set of small decision trees.
- The decision tree algorithm is deterministic, how can we modify it to produce different learners at different runs?
  - Data randomisation
  - Random subspaces
  - Algorithm randomisation

Aggregation methods

weighted majority voting, more commonin boosting than bagging.
output numeric. min max mean

- There are several methods to combine (aggregate) the outputs of the various classifiers
- When the output is a class label:
  - Majority voting
  - Weighted majority voting (e.g we can weight each classifier by its reliability (which also has to be estimated somehow, of course…)
- When the output is numeric (e.g. a probability estimate for each class):
  - We can combine the scores by taking their (weighted) mean, product, minimum, maximum, …
- Stacking not covered here, but an idea in this domain
  - Instead of using the above simple aggregation rules, we can train yet another classifier on the output values of the base classifiers

Bagging creating lots of data and aggregating

- Bagging = **B**ootstrap + **agg**regat**ing**
- It uses bootstrap resampling to generate *L* different training sets from the original training set.
- On the *L* training sets it trains *L* base learners.
- For prediction it aggregates the L learners by taking their average (using uniform weights for each classifiers), or by majority voting.
- The diversity or complementarity of the base learners is not controlled in any way, it is left to chance and to the instability of the base learning method.
- The ensemble model is almost always better than the unique base learners if the base learners are *unstable*.

Bootstrap resampling

- Suppose we have a training set with *n* examples.
- We would like to create *L* different training sets from this.
- Bootstrap resampling takes random examples from the original set with replacement.
- Randomness is required to obtain different sets for *L* rounds of resampling.
- Allowing replacement is required to be able to create sets of size *n* from the original data set of size *n*.
- As the L training sets are different, the result of the training over these set will also be more or less different, independent of what kind of training algorithm we use.
  - Although, it works better with unstable learners.

Text

Bagging

summary

n = number of rows
t different models
for loop, sample with replacement from training set and apply hidden algorithm

Model generation

```
Let n be the number of instances in the training data
For each of t iterations:
      Sample n instances from training set
            (with replacement)
      Apply learning algorithm to the sample
      Store resulting model
```

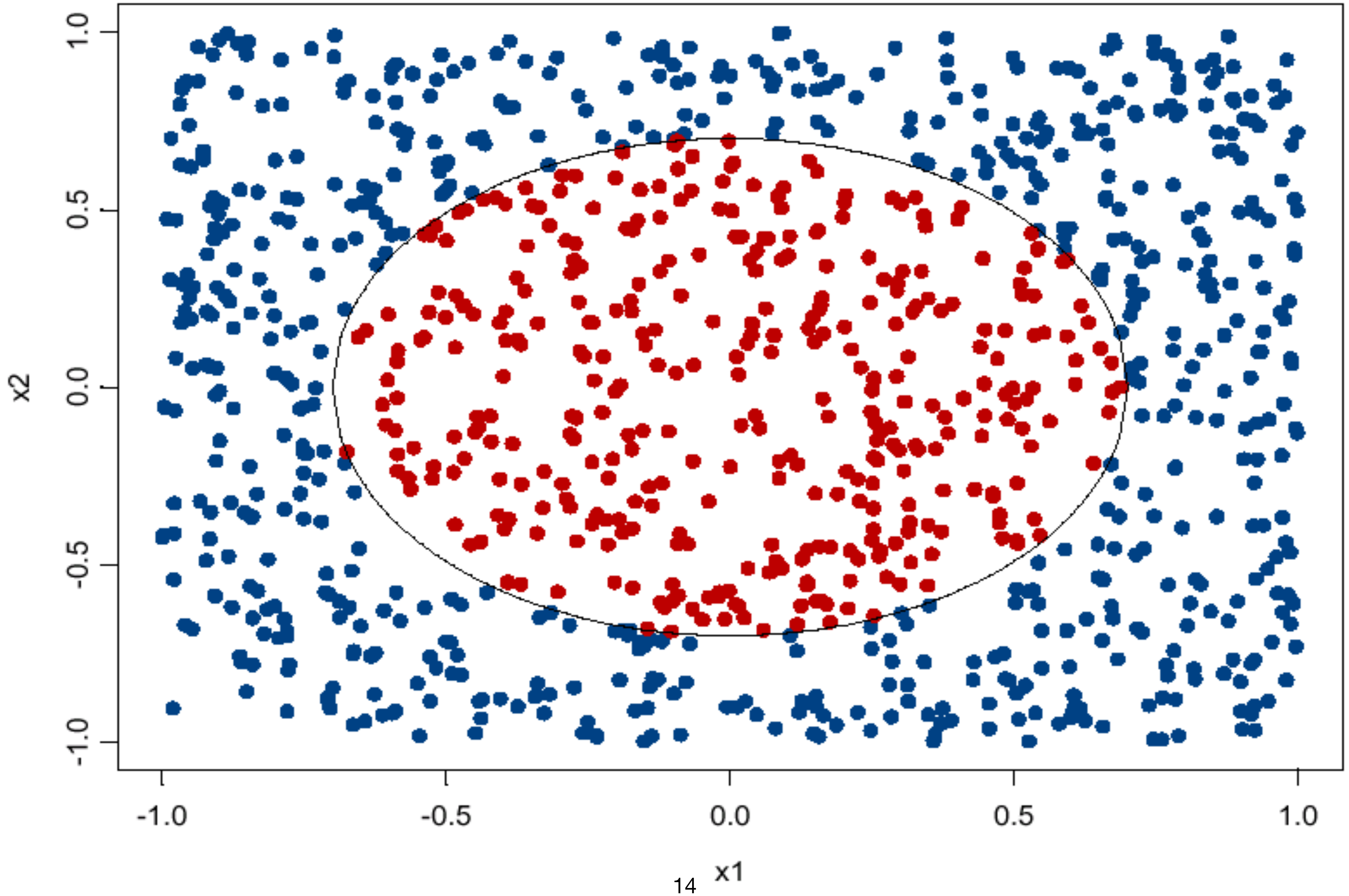
Classification

```
For each of the t models:
      Predict class of instance using model
Return class that is predicted most often
```

Bagging Example

training data

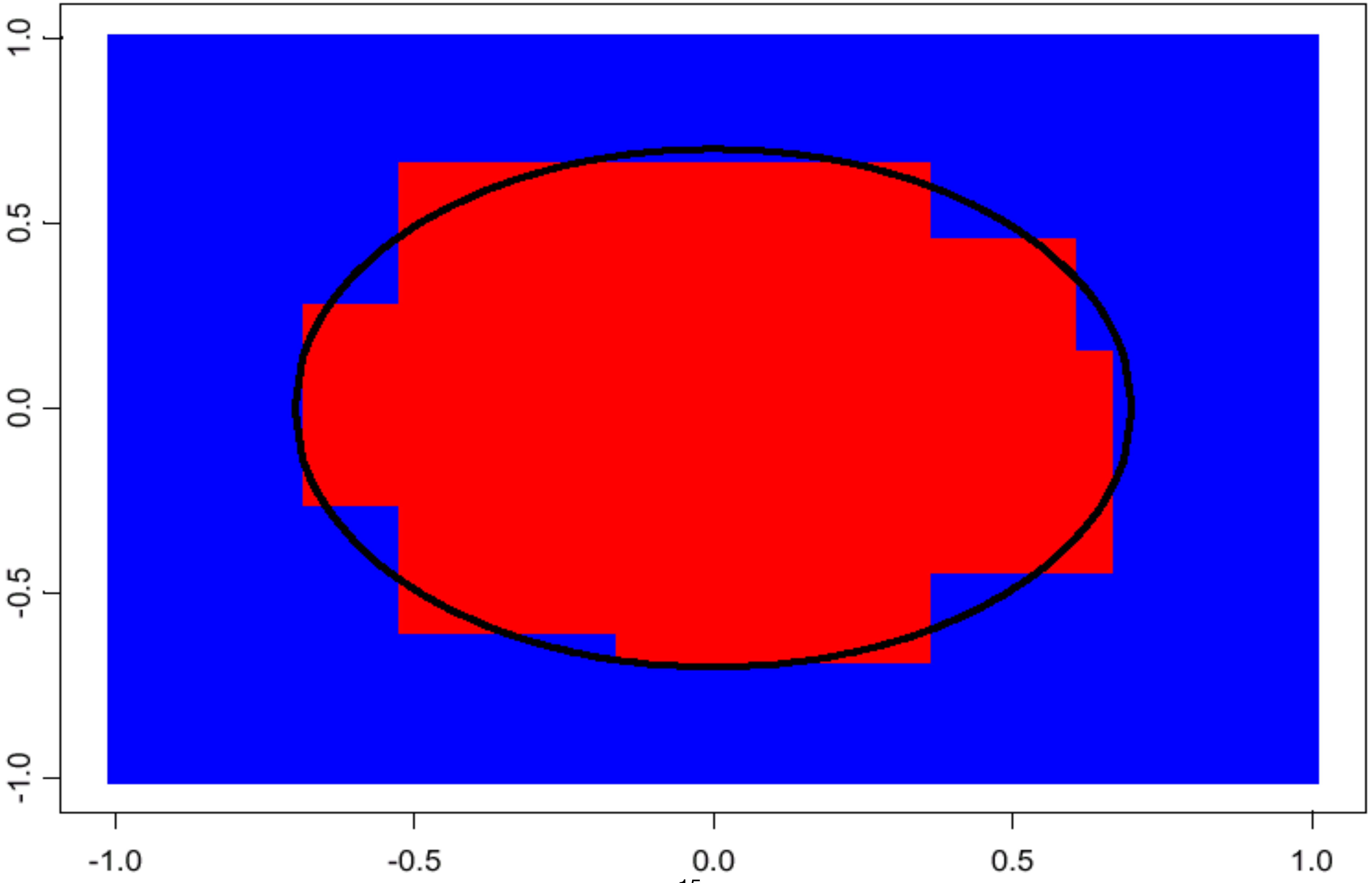
true decision boundary



Bagging Example

a single decision tree

decision boundary are piece wise and
perpendicular
single feature with a threshold

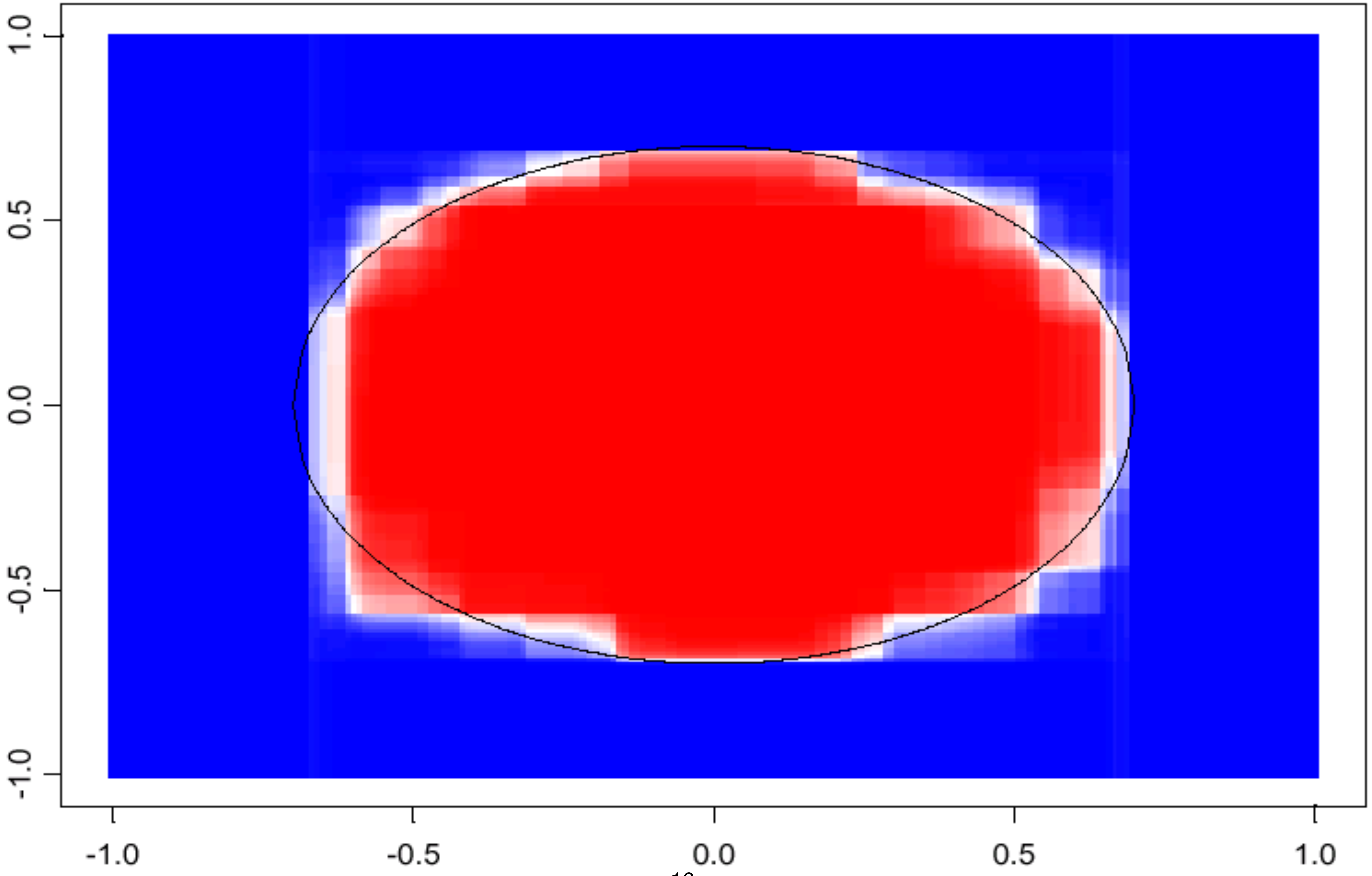


Bagging Example

100 aggregated decision trees

bagging or boosting model are shallow weak models
decision stumps of depth 1, one single decision
dont want it complex, make simple decision but accuracy > 0.5 so aggregated can be useful

aggregation is a slightly more defined decision boundary, more tree finer boundary



not hellpful 36.00 wed 10 may TODO

baseline for regression is intercept. feature list learning,

very least can learn intercept without learning features, alike learning classification , balance between the two classes of positive and negative, want to be better than 50%

advantage
multiple machine running in parralell - given that these tree are not that complex, these solutions can be not big data - tabular data. working with probelms that are not huge, so being run in parralell wont be a game changer

cannot overfit by sampling more, may stop improving but cannot lead to over fitting - bagging not booster.

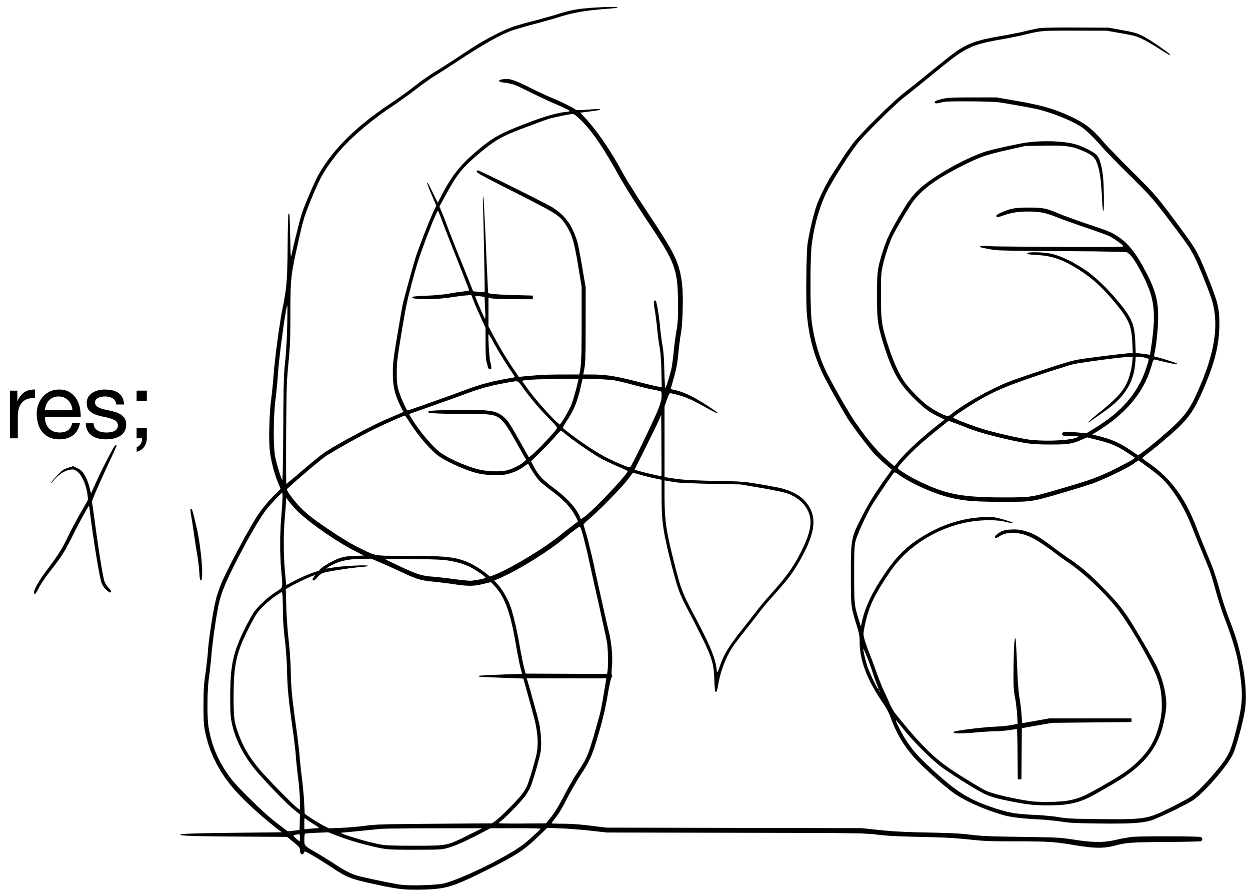
Random Forests

Learning

1. Choose *T*, the number of trees to grow.

selecting m at random can be problematic - disadvantage. can make a selection line that has m dataset that has an importNT FEATURE missing. some interactions between features too.

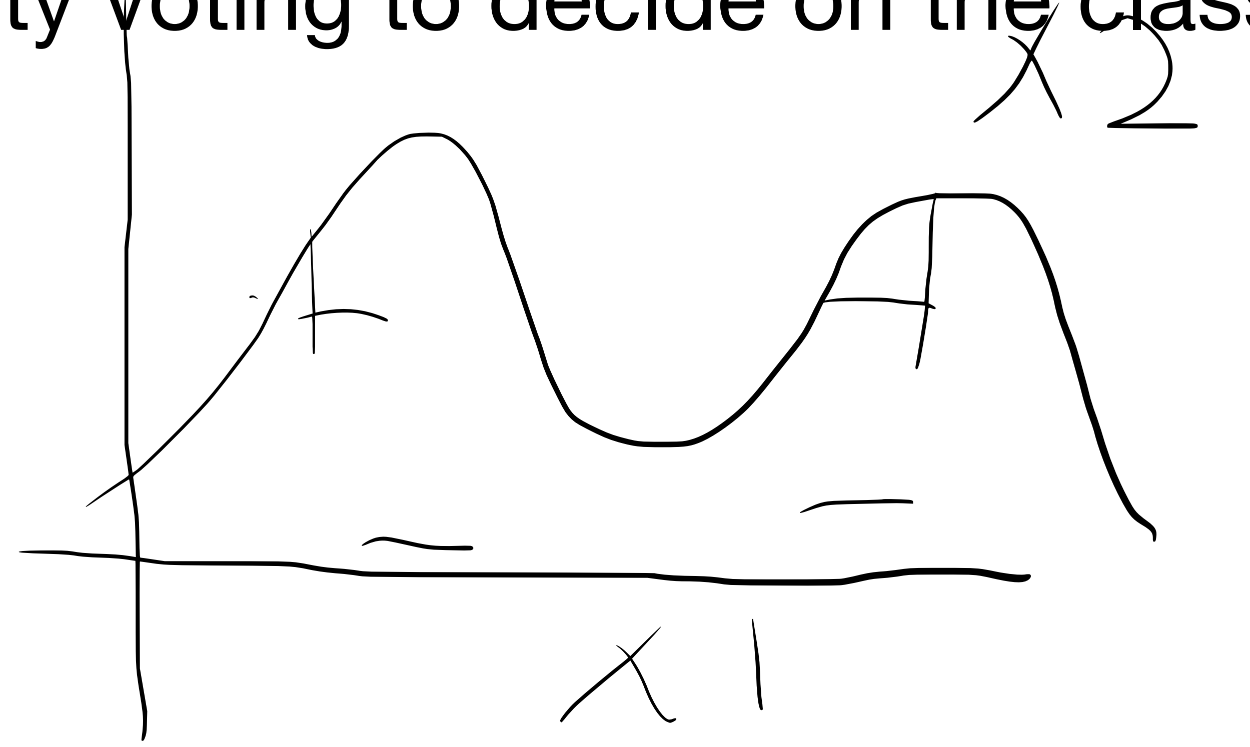
2. Choose *m*, the number of features to use (where *m* is smaller than the total number of features)
3. Repeat T times:
    - select *m* feature at random;
    - create a bootstrap sample of data only containing the *m* selected features;
    - construct a decision tree using this sample and store it in a collection;



Prediction aggregating

To classify point **x**, collect votes from every tree in the forest and then use majority voting to decide on the class label.

cannot say which feature more important , just you need both to separate the data - another problem when selecting the m random data



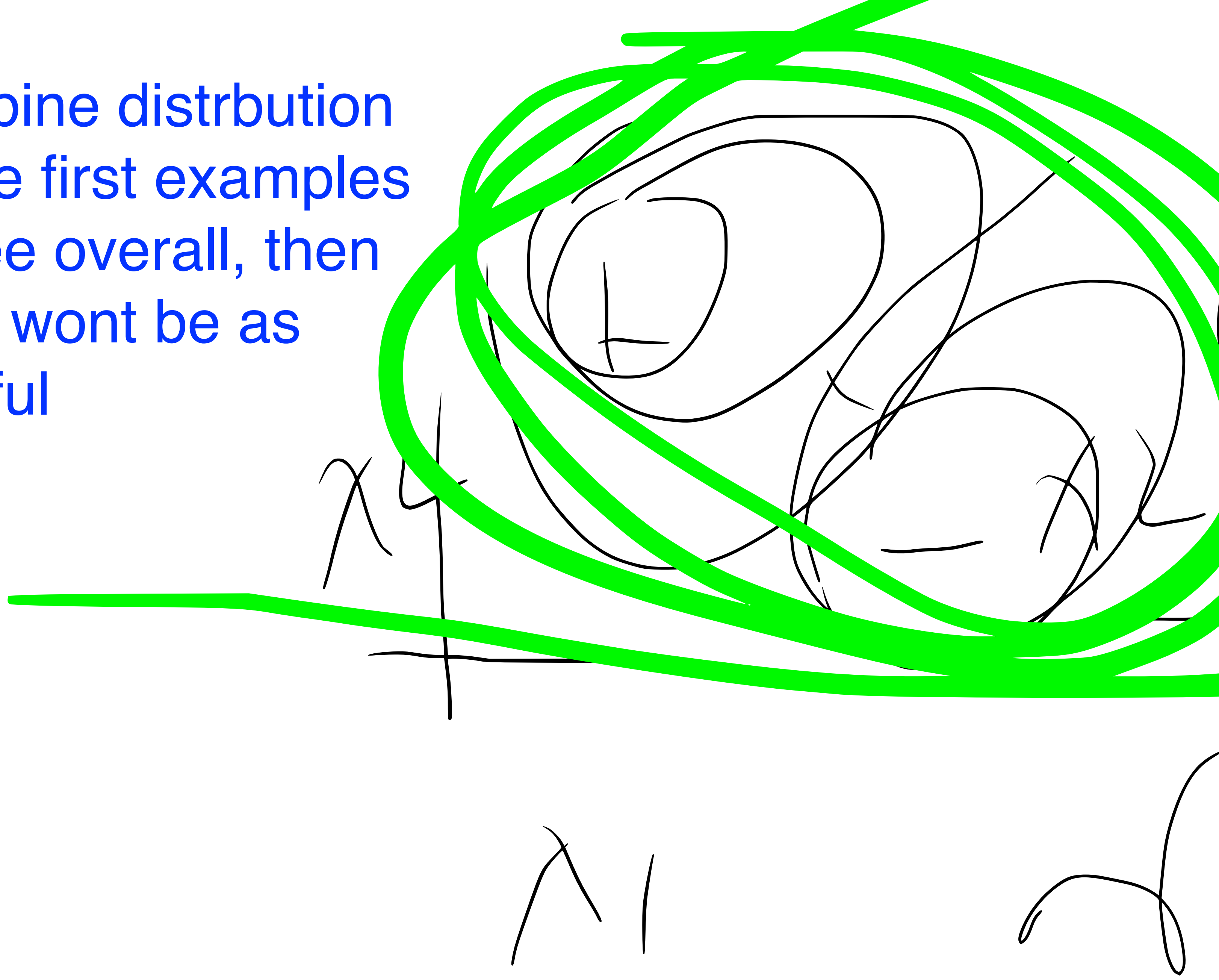
What are the advantages and disadvantages?

feature selection also a topic in machine learning

Rotation Forests

The idea

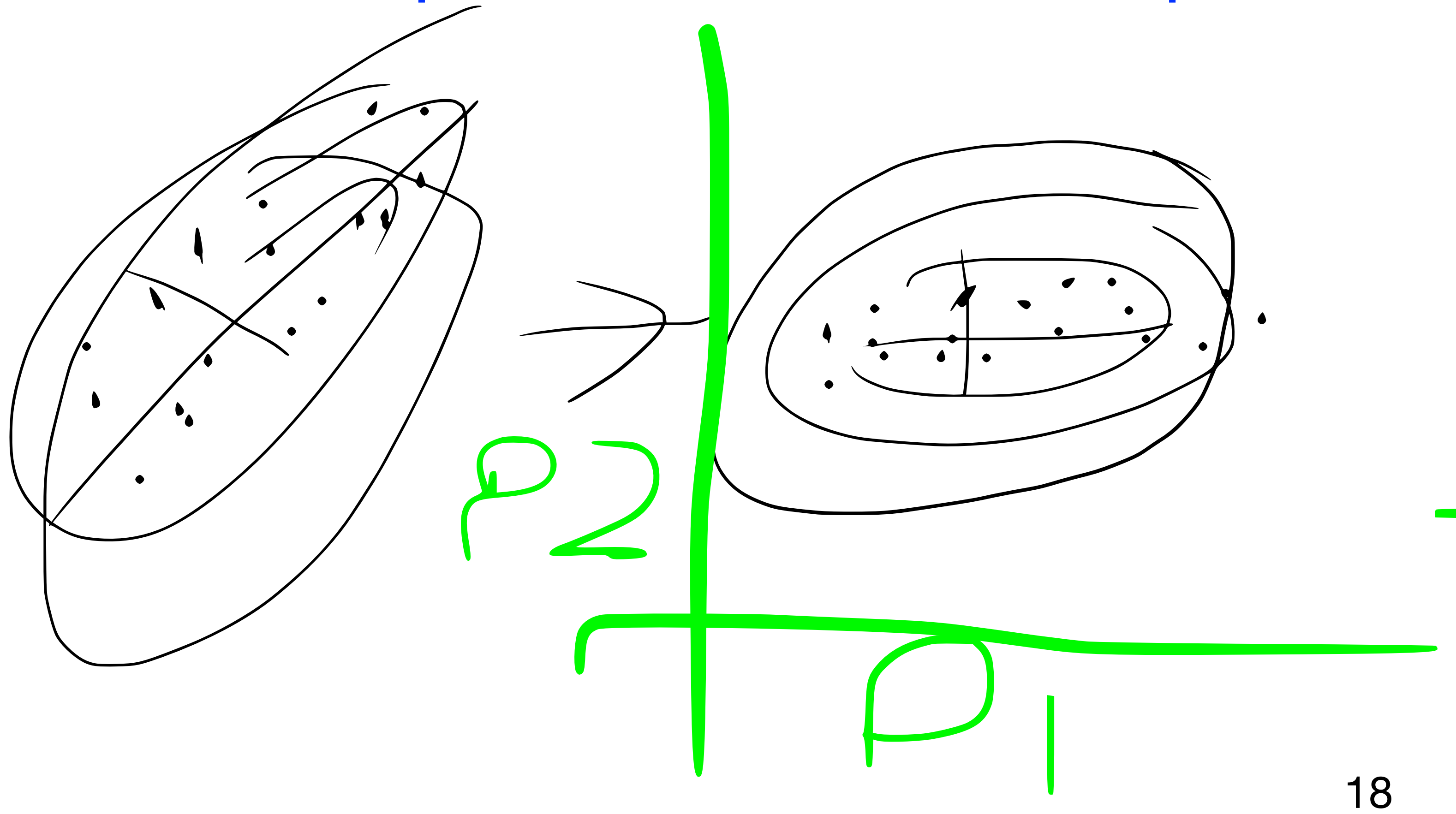
combine distrbution
of the first examples
to see overall, then
PCA wont be as
helpful

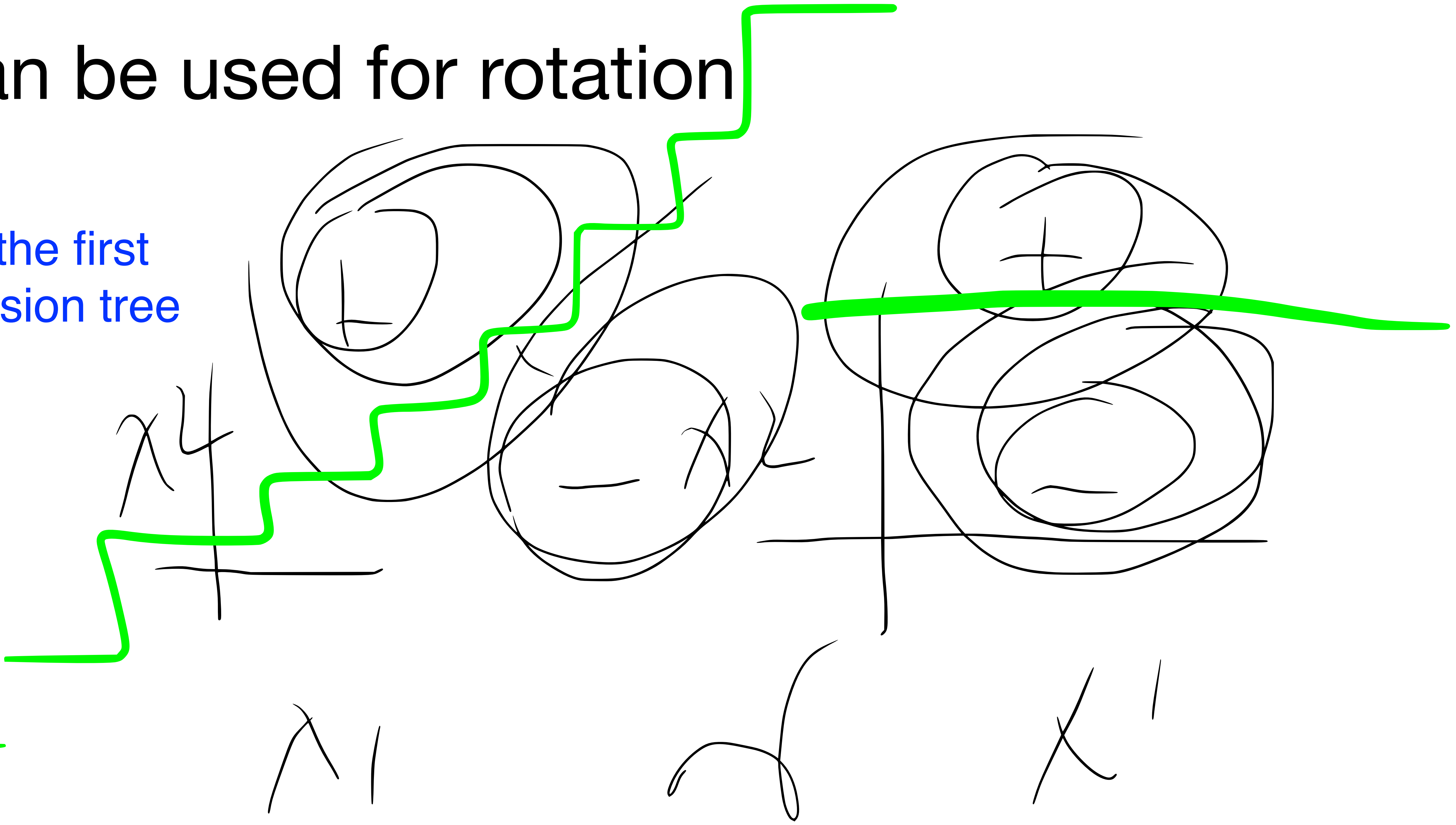


- As with random forests create subspaces for diversity
- Rotating a subspace can be useful
- PCA (Principal Component Analysis) can be used for rotation

the second example would be nicer for a decision tree to be applied as the first
example has many steps to create half space and will have deeper decision tree

PCA can achieve is





Rotation Forests - learning

Given

- X: the objects in the training data set (an $N \times n$ matrix)
- Y: the labels of the training set (an $N \times 1$ matrix)
- L: the number of classifiers in the ensemble
- K: the number of subsets
- $\{\omega_1, \ldots, \omega_c\}$: the set of class labels

For $i = 1 \ldots L$

- Prepare the rotation matrix R_i^a:
  - Split $\mathbf{F}$ (the feature set) into K subsets: $\mathbf{F}_{i,j}$ (for $j = 1 \ldots K$)
  - For $j = 1 \ldots K$
    * Let $X_{i,j}$ be the data set X for the features in $\mathbf{F}_{i,j}$
    * Eliminate from $X_{i,j}$ a random subset of classes
    * Select a bootstrap sample from $X_{i,j}$ of size 75% of the number of objects in $X_{i,j}$. Denote the new set by $X'_{i,j}$
    * Apply PCA on $X'_{i,j}$ to obtain the coefficients in a matrix $C_{i,j}$
  - Arrange the $C_{i,j}$, for $j = 1 \ldots K$ in a rotation matrix R_i as in equation (1)
  - Construct R_i^a by rearranging the the columns of R_i so as to match the order of features in $\mathbf{F}$.
- Build classifier D_i using (XR_i^a, Y) as the training set

Rotation Forests - prediction

- For a given $\mathbf{x}$, let $d_{i,j}(\mathbf{x}R_i^a)$ be the probability assigned by the classifier D_i to the hypothesis that $\mathbf{x}$ comes from class ω_j. Calculate the confidence for each class, ω_j, by the average combination method:

$$\mu_j(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L} d_{i,j}(\mathbf{x}R_i^a), \quad j = 1, \ldots, c.$$

- Assign $\mathbf{x}$ to the class with the largest confidence.

go thro l models and see their
confirdence and add it up

which class has the highest confidence
and pick that one

Option Trees

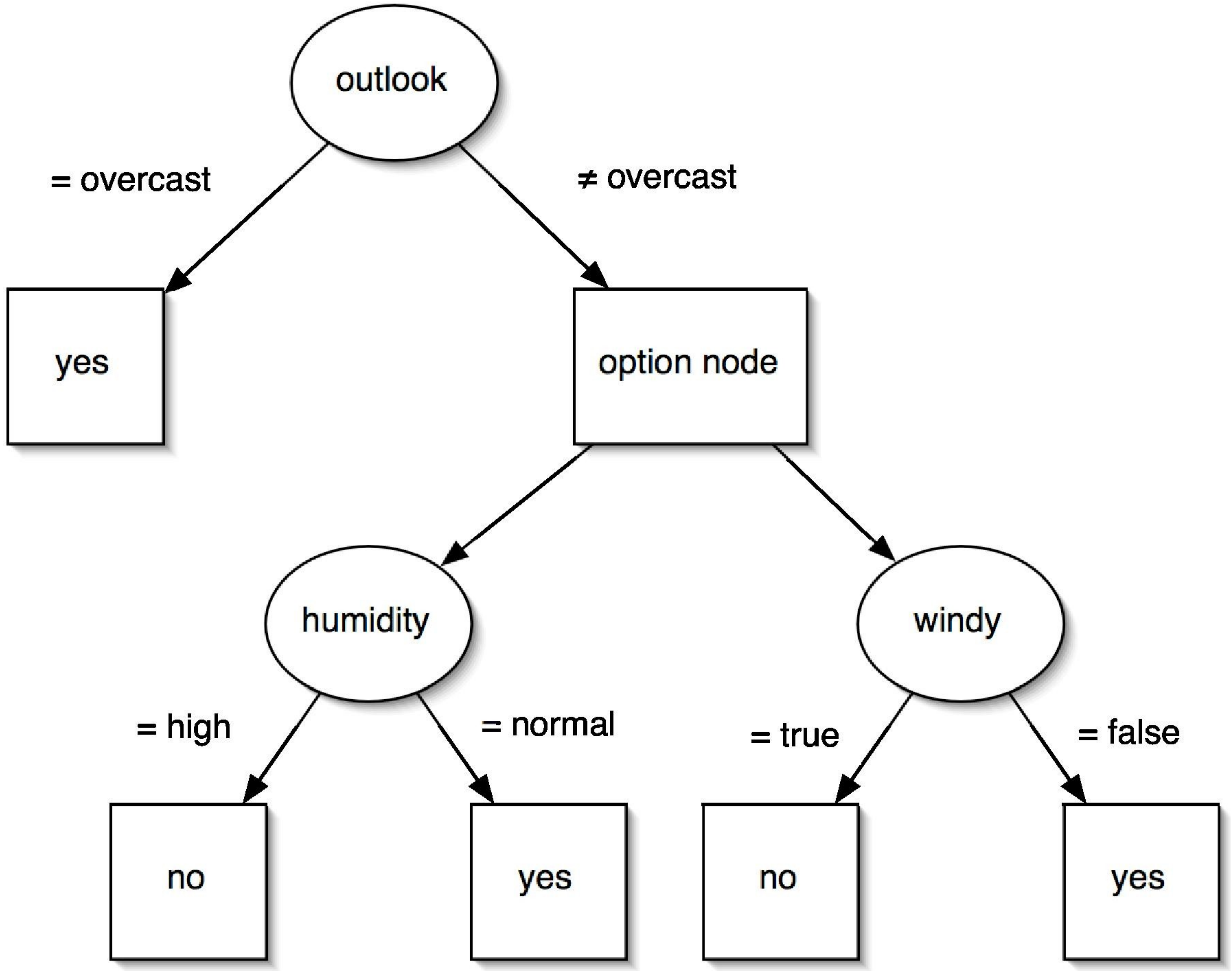
random generate inputs,

train a decision tree or model based on generative data

the other thing is that data is not ensemnle.

some deicison can be conflicting or need to be creative. an option node. you allow branching, more human trained

- Ensembles are not interpretable
- Can we generate a single model?
  - One possibility: "*cloning*" the ensemble
  - Another possibility: generating a single structure that represents ensemble in a compact fashion
- Option tree: decision tree with option nodes
  - Idea: follow all possible branches at option node
  - Predictions from different branches are merged using voting or by averaging probability estimates



Boosting
the idea

instead of making indepdendant models, you are making dependant models.

want to make in sequence. one after another, what is shared among the algorithm in this family is that iteration i focus on iteration i-1 - trying to incrementally improve the performance.

gradient boosting :
$y = h1(x) + e1 = h1(x) + h2(x) + e2$
$e1 = h2(x) + e2$

- Bagging creates diversity but we do not have direct control over the usefulness of the newly added classifiers

- The basic idea of boosting is to generate a series of base learners which complement each other

  - For this, we will force each learner to focus on the mistakes of the previous learner

  - We iteratively add new base learners, and iteratively increase the accuracy of the combined model

Boosting
role of weights

use weights to say some examples are more important than the other, so next iteration the weight make the misclassified have higher chance of correcting itself.

- Weights are used both for examples in the training data and for the learnt models

take performance and error into account and wieght them accordingly

- We represent the importance of each example by assigning a weight
  - Correctly classified: smaller weights
  - Misclassified: larger weights

sometimes weight can be integrated - eg into the loss function. if treated like black boxes, can put weight on different samples. two instances based on their weight.

higher weight higher chance of being repeated in the boostrp sample.

- The weights can influence the algorithm in two ways
  - Boosting by sampling: the weights influence the resampling process
    - This is a more general solution
  - Boosting by weighting: the weights influence the learner
    - Works only with certain learners
- Boosting also makes the aggregation process more clever: We will aggregate the base learners using weighted voting
  - Better weak classifier gets a larger weight

AdaBoost
model generation

maybe need weighted boostratping depending on if the algrithm has weights incorporated.

1.35 wed may 10

equal weight to add up to 1, and remain fixed

```
Assign equal weight to each training instance
For t iterations:
  Apply learning algorithm to weighted dataset,
     store resulting model
  Compute model's error e on weighted dataset
  If e = 0 or e ≥ 0.5:
    Terminate model generation
  For each instance in dataset:
    If classified correctly by model:
       Multiply instance's weight by e/(1-e)
  Normalize weight of all instances
```

0 < e < 1

effect is reducing the rate

important to reduce by
ratio 1-e'

normalize weight
when we assign equal weight
need to add up to 1, a constant
that remains fixed.

AdaBoost
classification

t or less coz could terminate

which class this model is predicting, so add the confidence of prediction which is -log(e/(1-e)) , so adding a positive values, a large value the closer you are to 0

-log(e/(1-e)) is negative coz itll turn into a positive value and this is a value between 0-1

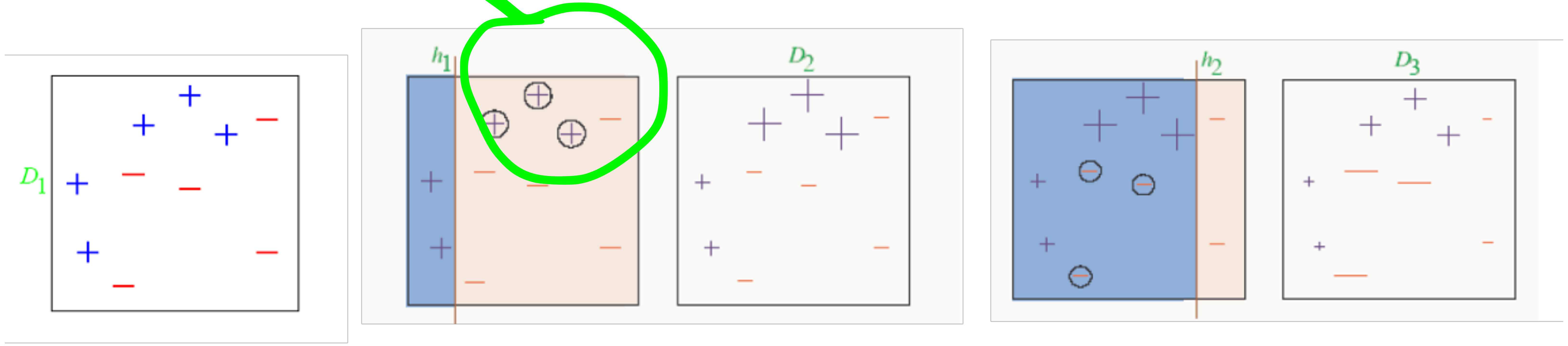
t or less because of there may be termination if e == 0 or e>= 0.5

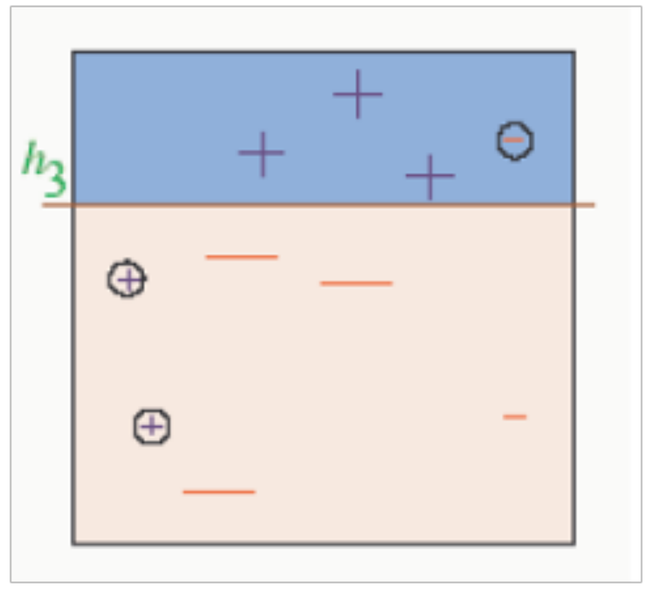
```
Assign weight = 0 to all classes
For each of the t (or less) models:
  For the class this model predicts
    add -log e/(1-e) to this class's weight
Return class with highest weight
```

AdaBoost

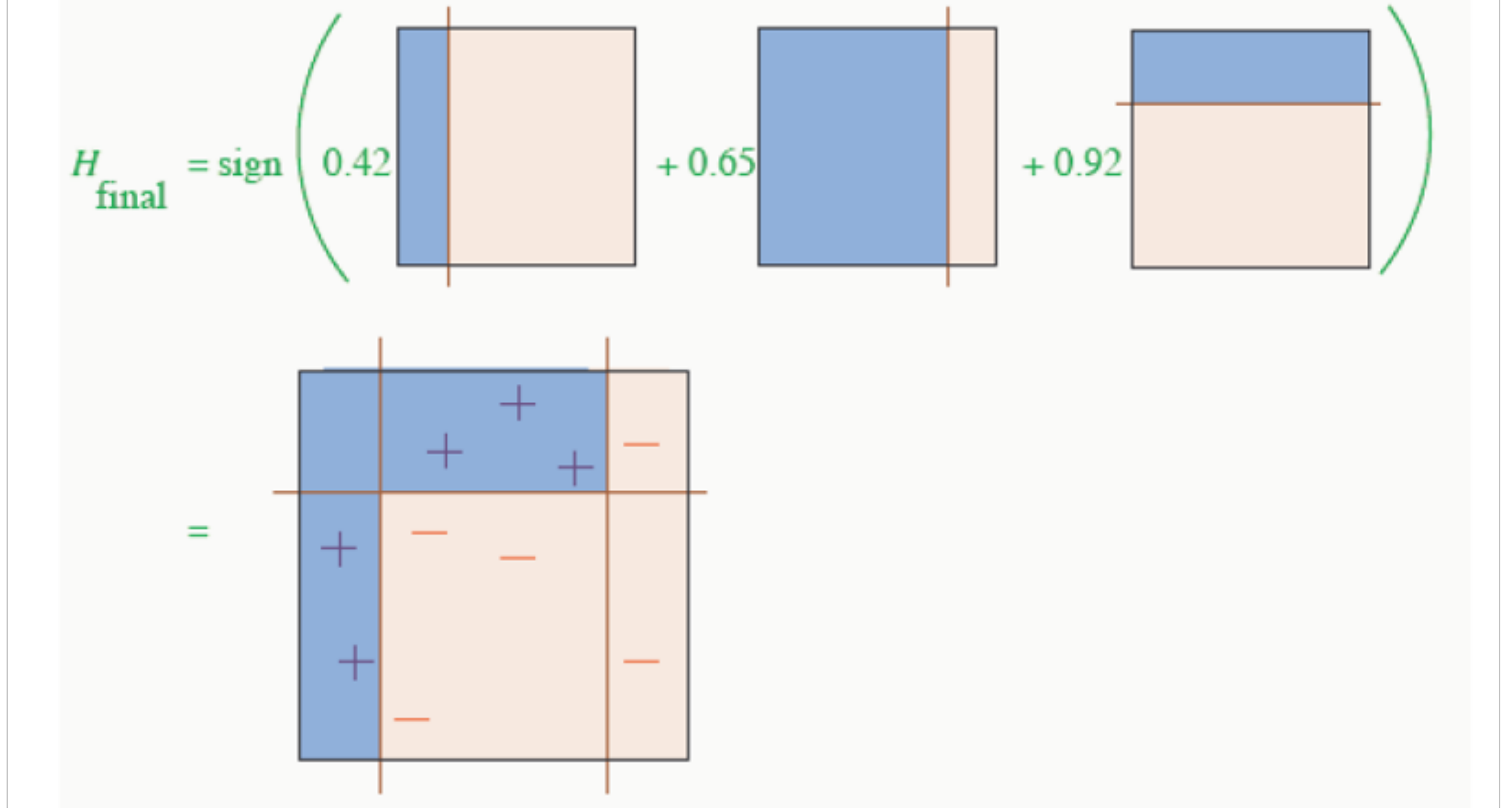
example

misclassification error can be multipled, misclassifified will be larger after normalization, so will I become





Text



Boosting

types of boosting

AdaBoost

- Adaptive Boosting
- One of the originals

Gradient Boosting

- Uses gradient descent to create new learners
- The loss function is differentiable

XGBoost

- “eXtreme Gradient Boosting”
- Type of gradient boosting
- Has become very popular in data science competitions

Boosting

pros and cons

Advantages

- Fast
- Good performance
- A lot of available software
- Can also be used for regression (e.g. additive trees and boosted regression trees)

Disadvantages

- Can easily overfit the data
- Not (immediately) explainable