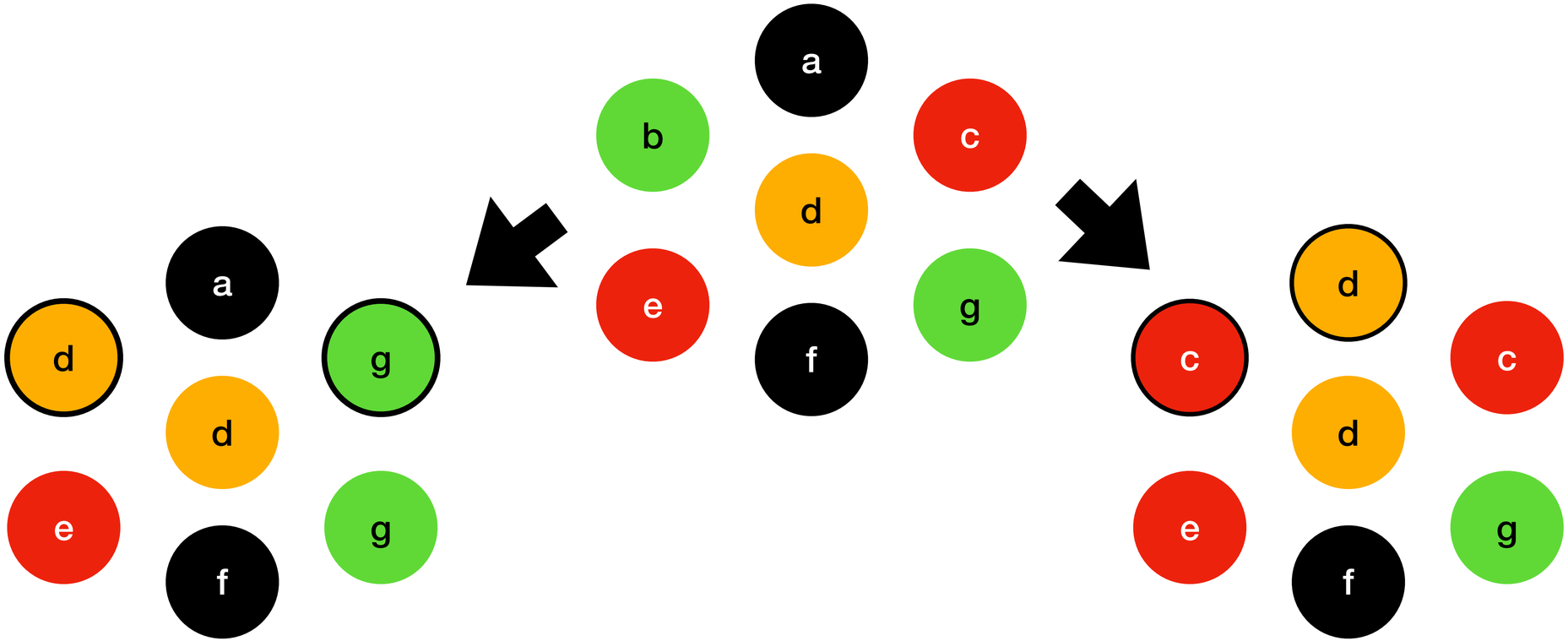
Bagging and Boosting

klgold for 10/26

Bagging and Random Forests

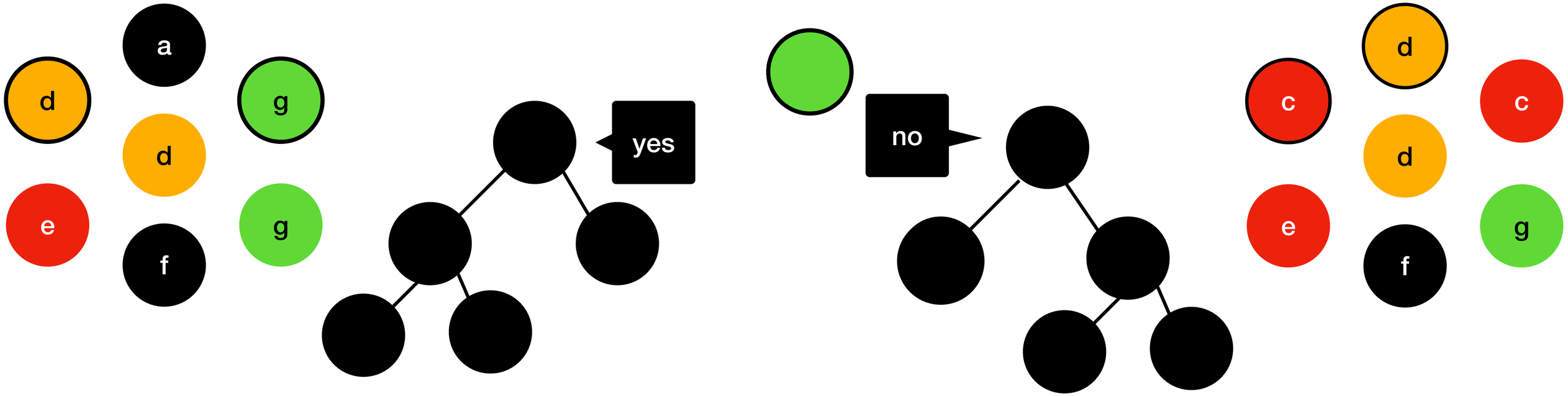
Bagging: Sampling With Replacement

- Instead of training one classifier, train K with K different samples-with-replacement (of the same size as original data)
  - If original dataset had N points, draw N points for each new dataset - but draws are with replacement



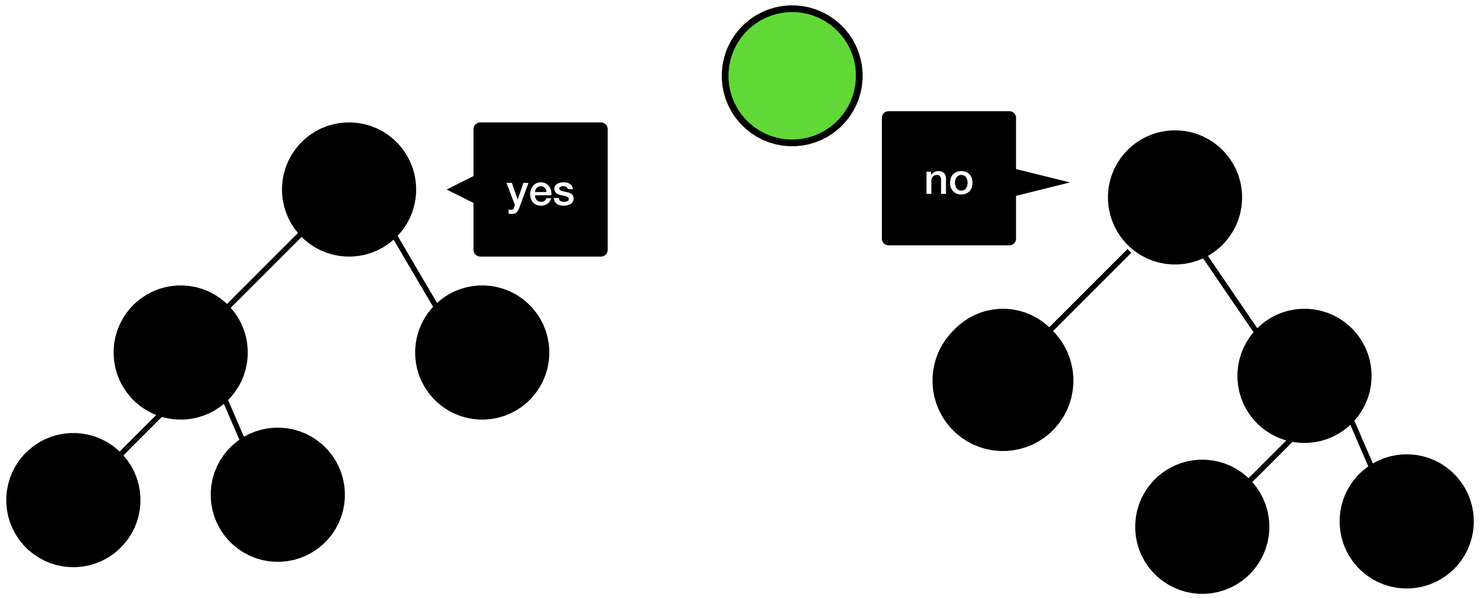
Bagging: Voting

- Instead of training one classifier, train K with K different samples-with-replacement (of the same size as original data)
  - If original dataset had N points, draw N points for each new dataset - but draws are with replacement
- Once trained, each trained classifier votes on new examples



Bagging

- Overall, bagging tends to reduce *variance* in classifier accuracy, and also helps with overfitting
  - In both cases, classification can't depend as heavily on a single attribute



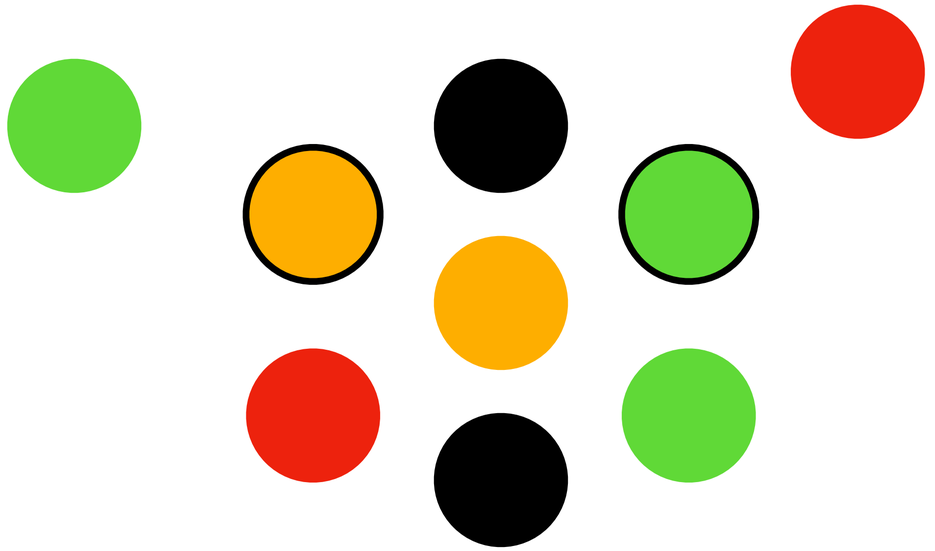
Random Forests

- Decision trees are often still highly correlated in their responses, even after bagging
- To further vary decision tree structure, some fraction of the attributes are used for potential splits
  - Common choice is $\sqrt{F}$ if there were f features before
  - The feature sampling happens at every point in the tree
  - The most informative feature is still selected from these

Random Forests are efficient

- Random forests may sound inefficient...
  - Train K trees instead of one
  - Extra work randomly sampling attributes to split on
- But a few things work to make them fast
  - $\sqrt{F}$ attributes to try instead of F, per node
  - No need to prune the trees - the voting in the end mitigates overfitting
  - Highly parallelizable, since the trees can be trained apart from each other

Out-of-bag error



- One way to measure the error of the forest is:
  - For each training example,
    - Classify the example using the trees that weren't trained with it.
  - Find the average accuracy over all examples
- This is called the *out-of-bag* error, and it efficiently makes use of data points without need for a separate validation set

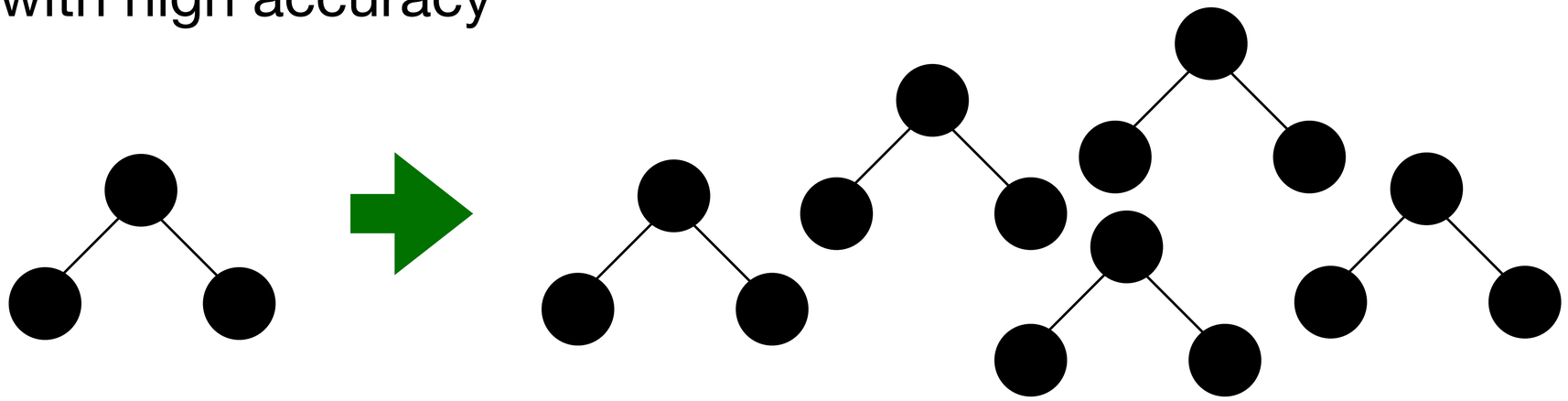
Random forests are a popular ML solution

- They have high accuracy, but we can also ask for the whole forest what attributes are most important
- Random forests have been used for (list from R&N 4e):
  - Credit card default prediction
  - Household income prediction
  - Option pricing
  - Remote sensing
  - Cancer prediction
  - Protein interaction prediction
  - And more ....

Boosting

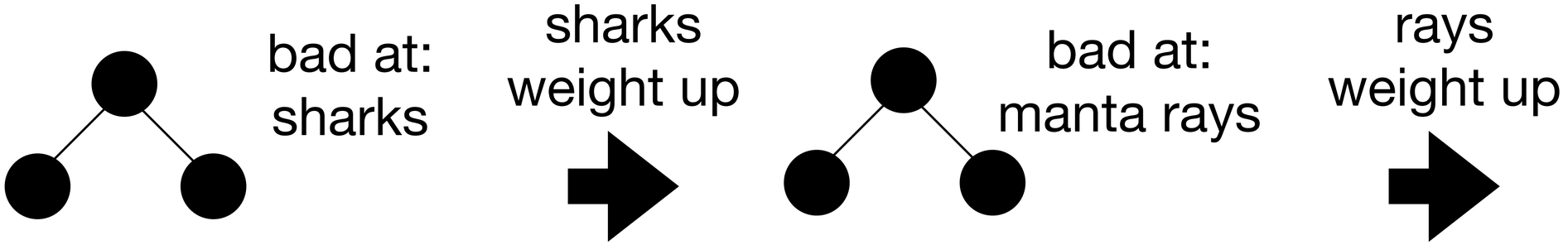
From a weak learner to a strong learner

- A "weak learner" is one that does even a tiny amount better than chance
  - Probability 0.5 + ϵ of correct on Boolean labels
  - Common example: "decision stump," depth 1 decision tree
- Boosting can create an *ensemble* of weak learners that votes on a correct classification with high accuracy

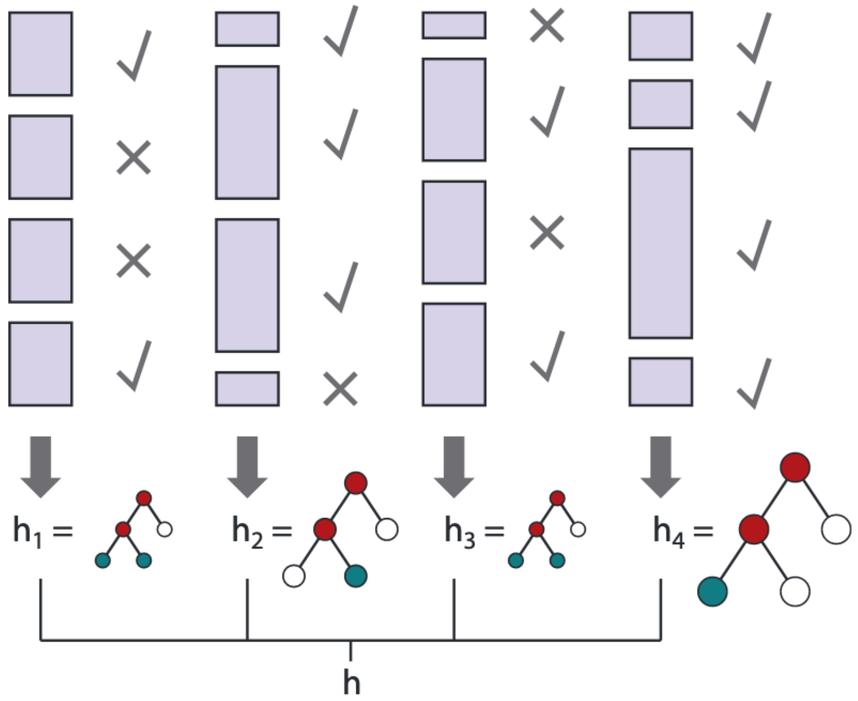


High-level idea

- After training a classifier, weight the examples it got wrong more heavily
  - In decision trees, add *example_weight* to the label count instead of 1 when calculating entropy
- Train the next classifier on the reweighted examples
- Reweight again for the next classifier
- Weight classifiers proportionally to the weight of the examples they get right



Boosting illustration from Russell and Norvig



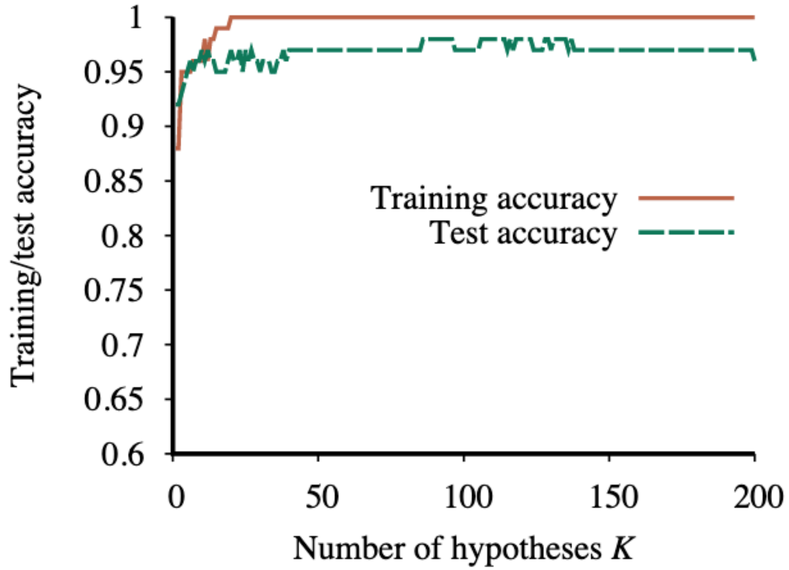
AdaBoost Algorithm - Pseudocode sketch

Initialize weights to equal
For each learner h_k to be trained,
    train classifier h_k on weighted examples
    error = sum of incorrect example weights
    downgrade the weights of correctly classified examples
    give learner a weight learnerWeight based on its error
return all h_k with weights learnerWeight_k

AdaBoost Algorithm - Pseudocode

for all weights
    $weights_i$ = 1/N (where N is the number of examples)
for k = 1 to max_learners do
    train classifier h_k on examples with weights
    error = 0
    for j = 1 to N do
        if $h_k(example_j)$ incorrect,
            error += $weight_j$
    if error > 1/2 break
    for j = 1 to N do
        if $h_k(example_j)$ correct,
            $weight_j$ = $weight_j$ * error/(1-error)
    Normalize(weights)
    $learnerWeight_k$ = 1/2 log ((1-error)/error)
return all h_k with weights $learnerWeight_k$

Test set error improves after training set error reaches zero



- Common effect with boosting - fitting after training set is perfect continues to improve test set performance

Calling from scikit-learn

```
from sklearn.datasets import fetch_lfw_people
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split

faces = fetch_lfw_people(min_faces_per_person = 100)
face_f_train, face_f_test, face_l_train, face_l_test = \
  train_test_split(faces['data'],faces['target'], test_size=0.1, random_state=110)

clf = AdaBoostClassifier(n_estimators=200)
clf.fit(face_f_train, face_l_train)
print(clf.score(face_f_train,face_l_train))
print(clf.score(face_f_test,face_l_test))
```