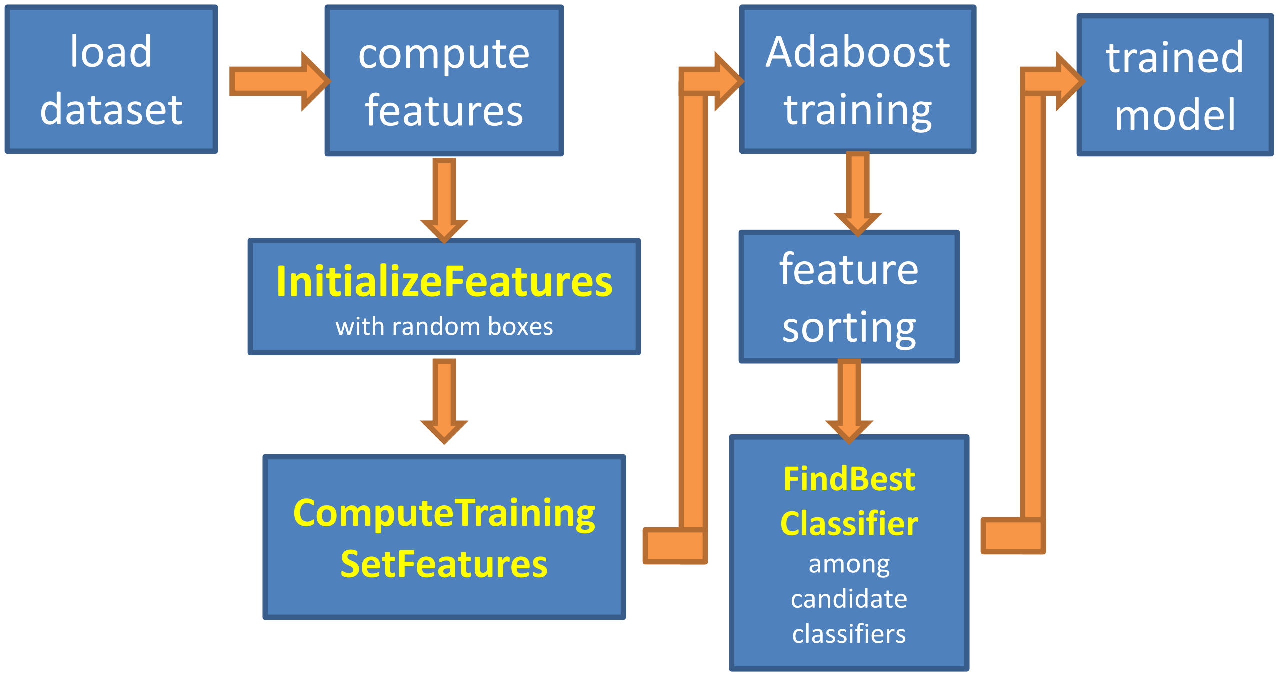
Assignment 4

Face Detection

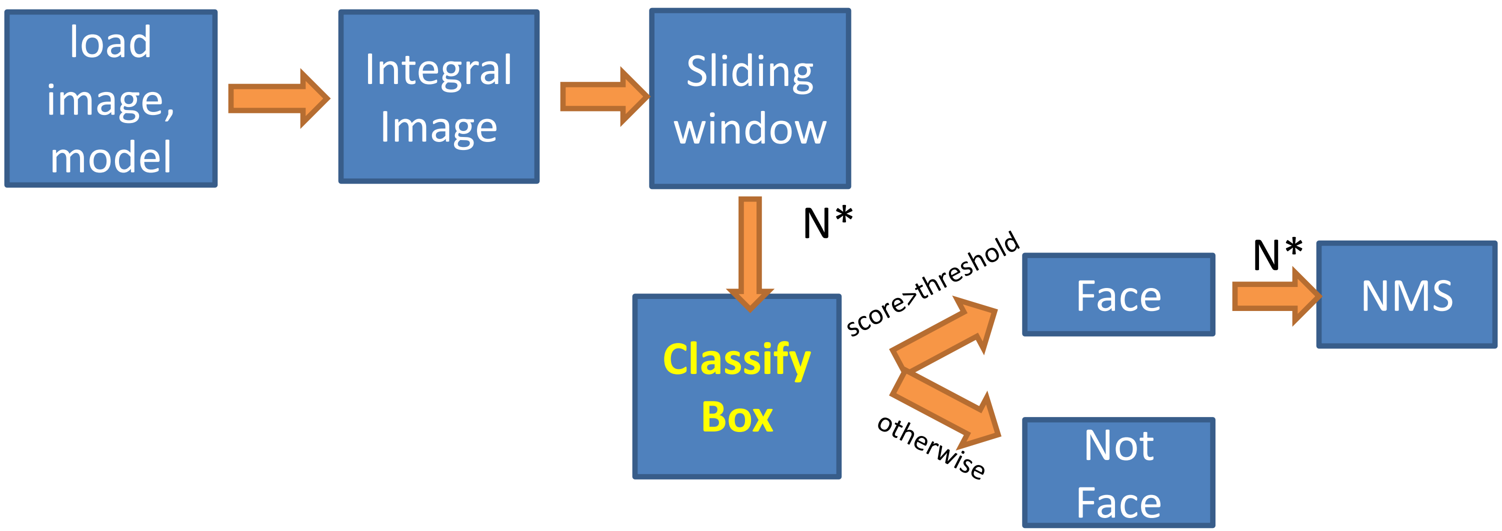
Overview

- Large number of initial weak classifiers.
- Each weak classifier computes one rectangular feature.
- The program computes the best threshold and polarity for each weak classifier. (<, >)
- Adaboost selects a subset of these classifiers and assigns a weight to each one
- Final classifications of boxes in test images are based on a combination of the selected ones.

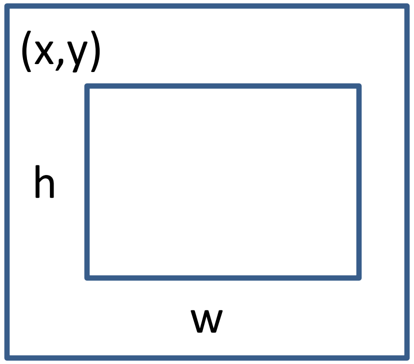
Training pipeline



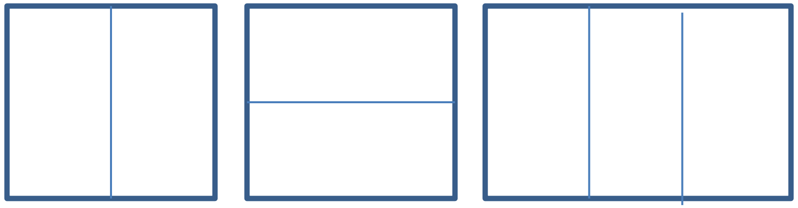
Testing pipeline



Initializefeatures



- Given in the code base
- Initializes all weak classifiers
- Chooses the upper left corner (x,y) and the height and width h and w randomly (but from 0 to 1)
- Chooses type of box
  - vertical 2-box
  - horizontal 2-box
  - vertical 3-box



- Sets areas
- Assigns values (1 and -1 for 2-box; 1, -2, 1 for 3-box)

ComputeTrainingSetFeatures

- Given in the code base as a shell
- Calls two methods that you code
  - IntegralImage: computes the integral image for each training patch (double array in, double array out)
  - ComputeFeatures: uses the integral image for each training patch to compute features for that patch, one for each weak classifier, and puts them in an array called features.

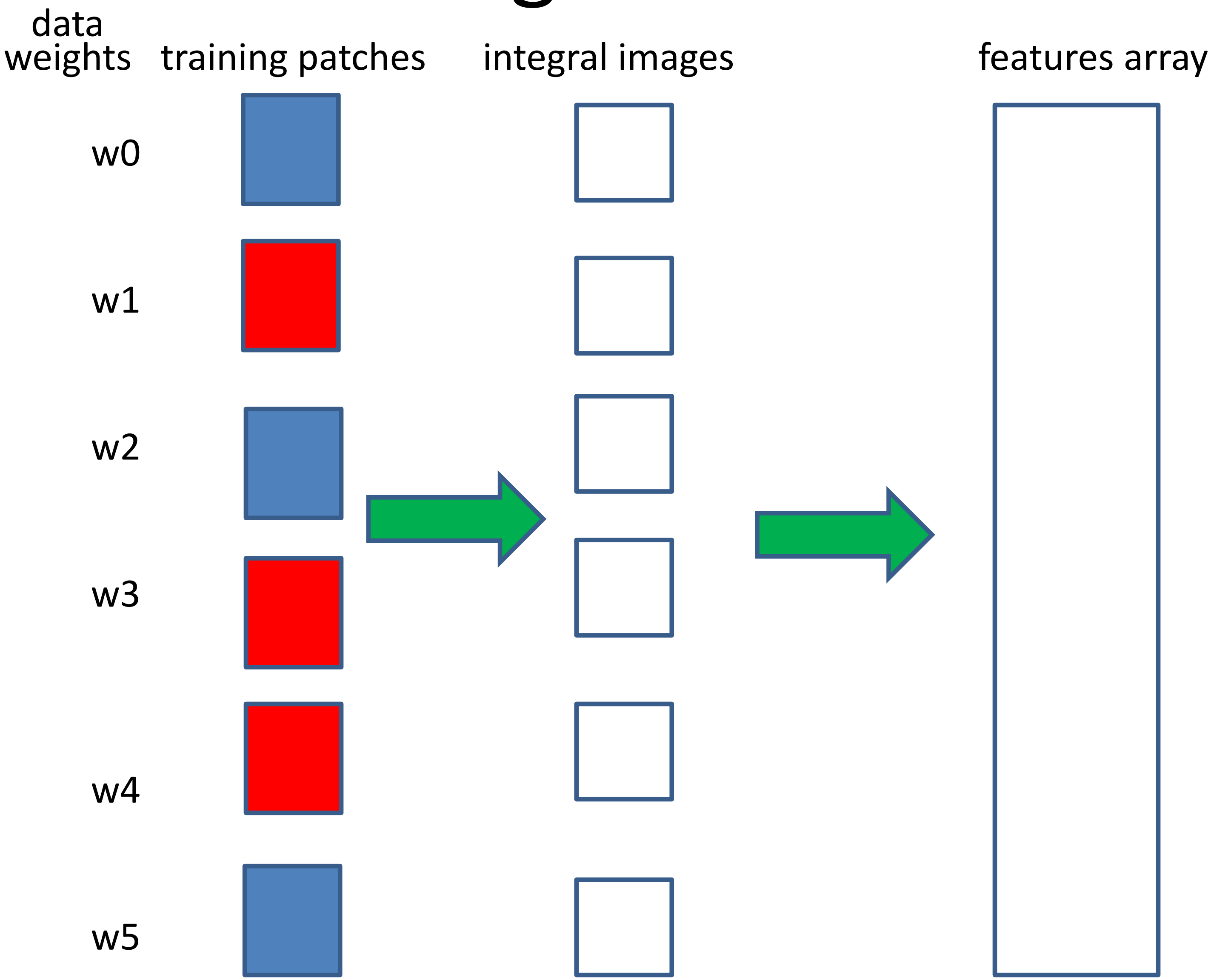
```
void MainWindow::ComputeTrainingSetFeatures(double *trainingData, double *features,
                              int numTrainingExamples, int patchSize, CWeakClassifiers
*weakClassifiers, int numWeakClassifiers)
{
   int i;
   double *integralImage = new double [patchSize*patchSize];

   for(i=0;i<numTrainingExamples;i++)
   {
      // Compute features for training examples

      // First compute the integral image for each patch
      IntegralImage(&(trainingData[i*patchSize*patchSize]), integralImage, patchSize,
patchSize);

      // Compute the Haar wavelets
      ComputeFeatures(integralImage, 0, 0, patchSize, &(features[i*numWeakClassifiers]),
weakClassifiers, numWeakClassifiers, patchSize);
   }
   // We shouldn't need the training data anymore so let's delete it.
   delete [] trainingData;

   delete [] integralImage;
}
```

ComputeFeatures

- For each weak classifier i
  - For each separate box j of that weak classifier
    - Use the integral image to efficiently find the sum of the values in the corresponding subimage of the patch

    - Multiply that by the box value
- Sum and normalize by size of window

Training Data and Features



holds feature values for each (feature/patch) combo.

Initializing Features

- It’s important to understand how the features are initialized and how they are stored.
- They are stored in big arrays but in two different orders
  – Initially, all the features for the first training example are kept together in one block
  – Next, the organization changes so that all the features for one weak classifier are together in one block. In this order they are sorted and indexed for use, but only one classifier at a time.

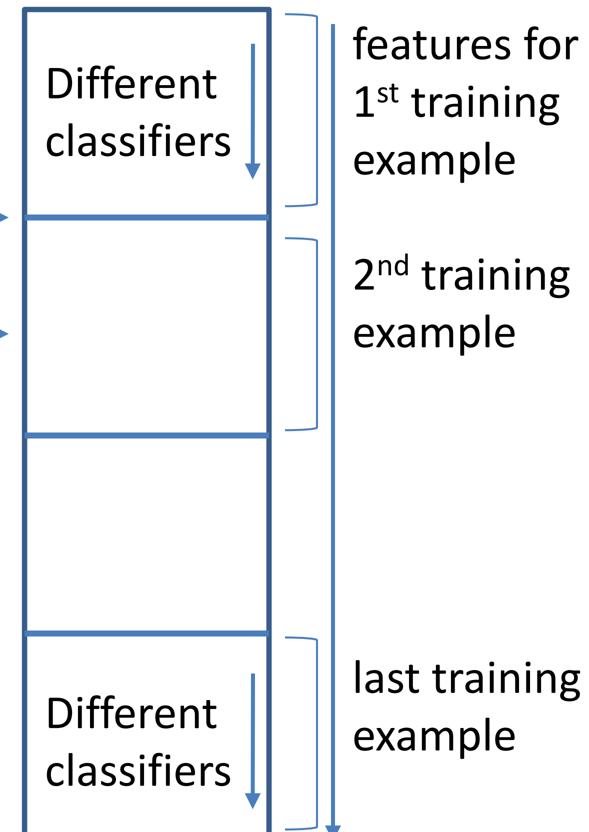
Initializing Features: First Step

Function ComputeTrainingSetFeatures

```
for(i=0;i<numTrainingExamples;i++)
  {         .....
        ComputeFeatures(integralImage, 0, 0, patchSize,
        &(features[i*numWeakClassifiers]), weakClassifiers, numWeakClassifiers,
        patchSize);
  }     feature offset1: i * numWeakClassifiers
```

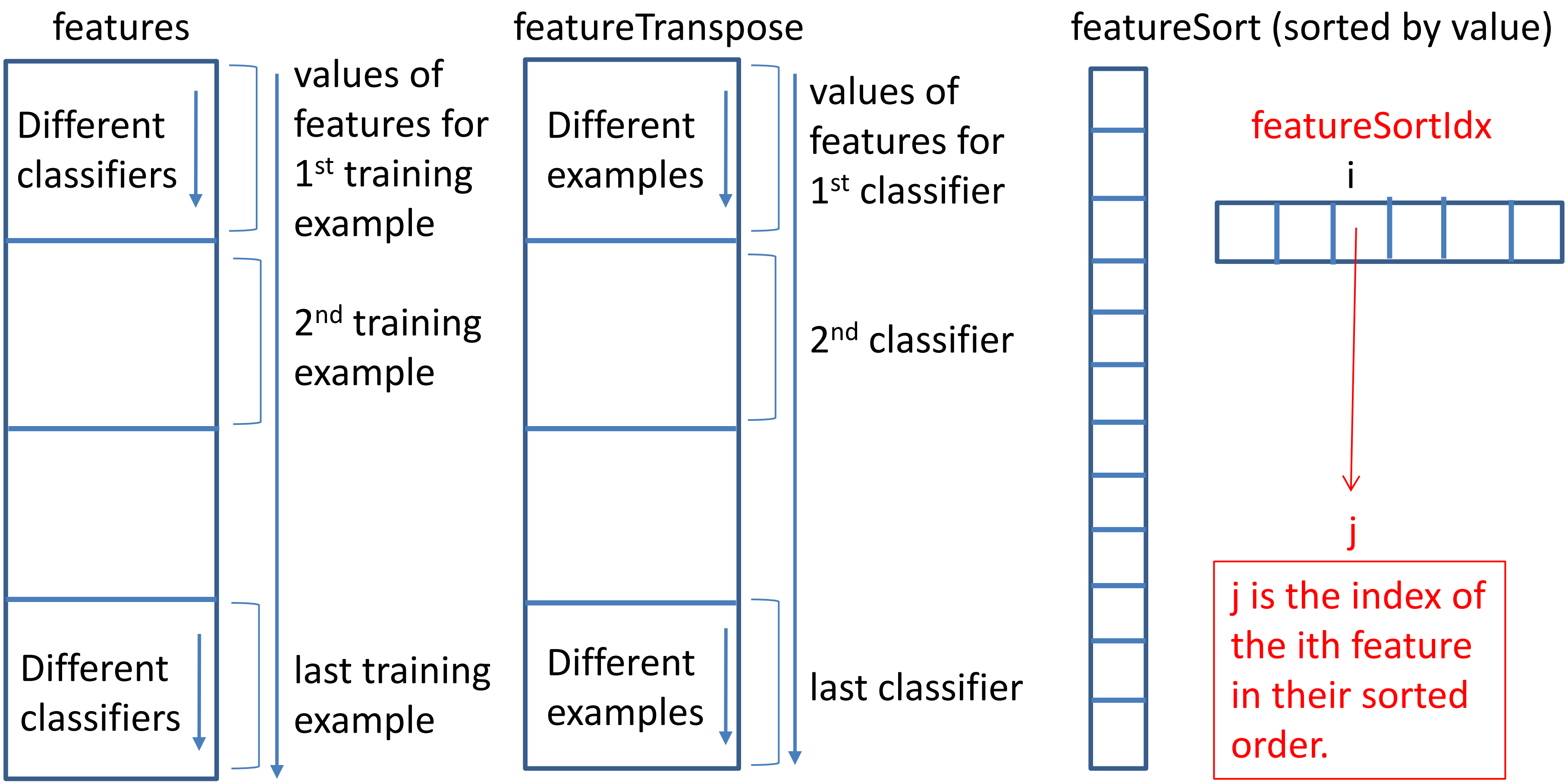
Function ComputeFeatures

```
for(i=0;i<numWeakClassifiers;i++)
  {         ......
       features[i] += weakClassifiers[i].m_BoxSign[j]*
          sum/((double) (size*size));
  }     feature offset2:  offset1 + i
```



features iterates over classifiers first, and then training examples.

Feature Sorting



featureSort is only for ONE classifier at a time.
It produces featureSortIdx, which is what is USED.

findBestThreshold

- you write it
- It is called by AdaBoost with a candidate classifier
- It is given the sort index which indexes into
  - features
  - weights
  - training labels
- Use it to go through the training samples (in sorted order), compute error for the classifier using the formula from the lecture.
- Return threshold, classifier weight, and polarity

Using the Sort Index: Example

samples	0	1	2	3	4
labels	F	B	F	B	B
features	6	3	10	2	1
weights	1/5	1/5	1/5	1/5	1/5

The feature values are for one particular feature (classifier).

index	4	3	1	0	2

The index tells you the sorted order of the features.

REVIEW: Picking the threshold for the best classifier

The features for the training samples are actually **sorted** in the code according to numeric value!

Algorithm:

1. find AFS, the sum of the weights of all the face samples
2. find ABG, the sum of the weights of all the background samples
3. set to zero FS, the sum of the weights of face samples so far
4. set to zero BG, the sum of the weights of background samples so far
5. go through each sample s in a loop IN THE SORTED ORDER

At each sample, add weight to FS or BG and compute:

$$e = \min(BG + (AFS - FS), FS + (ABG - BG))$$

Find the minimum value of e, and use the feature value of the corresponding sample as the threshold.

Setting the Polarity

error = min (BG + (AFS – FS), FS + (ABG –BG))

left right

- left is the number of background patches so far plus the number of faces yet to be encountered.
- right is the number of faces so far plus the number of background patches yet to be encountered.

- When left < right, set polarity to 0
- Else set polarity to 1

Threshold and Polarity Example

error = min (BG + (AFS – FS), FS + (ABG –BG))

samples	0	1	2	3	4
labels	F	B	F	B	B
features	6	3	10	2	1
weight	1/5	1/5	1/5	1/5	1/5
index	4	3	1	0	2

initialize
AFS = 0
ABG = 0
besterr = 999999

AFS becomes sum of face sample weights = 2/5; ABG = 3/5

step 0: idx = 4; FS stays 0; BG = 1/5
error = min(1/5 + (2/5-0), 0 + (3/5-1/5))= 2/5
besterr = 2/5; bestpolarity = 1; bestthreshold=1

step 1: idx = 3; FS stays 0; BG = 2/5
error = min(2/5 + (2/5-0), 0 + (3/5-2/5))= 1/5
besterr = 1/5; bestpolarity = 1; bestthreshold=2

Threshold and Polarity Example

error = min (BG + (AFS – FS), FS + (ABG –BG))

samples	0	1	2	3	4
labels	F	B	F	B	B
features	6	3	10	2	1
weight	1/5	1/5	1/5	1/5	1/5
index	4	3	1	0	2

initialize
AFS = 0
ABG = 0
besterr = 999999

step 2: idx = 1; FS stays 0; BG = 3/5
error = min(3/5 + (2/5-0), 0 + (3/5-3/5))= 0
besterr = 0; bestpolarity = 1; bestthreshold=3

step 3: idx = 0; FS = 1/5; BG = 3/5
error = min(3/5 + (2/5-1/5), 1/5 + (3/5-3/5))= 1/5
NO CHANGE

step 4: idx = 2; FS = 2/5; BG = 3/5
error = min(3/5 + (2/5-2/5), 2/5+ (3/5-3/5))= 2/5
NO CHANGE

RESULT

1 2 3 | 6 10

$\theta > 3$

AdaBoost

- Given in the code base BUT YOU NEED TO UNDERSTAND
- Starts with uniform weights on training patches
- For each weak classifier
  - sorts the feature values in ascending order
  - results of sort go in featureSort and featureSortIdx
  - selects numWeakClassifiers weak classifiers through calling **FindBestClassifier** for all candidates and selecting the ones with lowest errors
- updates weights on patches in dataWeights
- computes current total error for the training data and scores for each sample for debug purposes

Updating the Weights

- Suppose a weak classifier i has error err_i.
- The weight alpha for this classifier is

  $\alpha = \ln((1-err_i)/err_i)$
- The updating formula for the weight w_i for classifier i is given as

  $w_{t+1,i} = w_{t,i}\,\beta_t^{1-err_i}$

where $err_i = 0$ if example x_i is classified correctly else 1.

- And $\beta_t = \exp(-\alpha_t)$ which is $err_i/(1-err_i)$
- After updating weights, be sure to normalize by the sum of all of them.

ClassifyBox

- ClassifyBox uses the final set of weak classifiers to produce a score for a given box x on the image. (Called by FindFaces)
- The score it returns is NOT just zero or one.
- It should be

$$\sum_t \alpha_t h_t(x) - .5 \sum_t \alpha_t$$

- The value of each h_t depends on its polarity, threshold, and computed value on the box.

NMS (nonmaxima supresions)

- First read the comments at the top of the code carefully.
- Also read section 5.6 of the Viola Jones paper
- There is no “correct” method.
- You can do clustering and keep one per cluster.
- You can Sort the boxes according to score in descending order and for each box, remove those that overlap a significant percentage (iteratively)
- You can make up your own method so that you don’t get too many detections.