AdaBoost Meta-Learning

葉建華

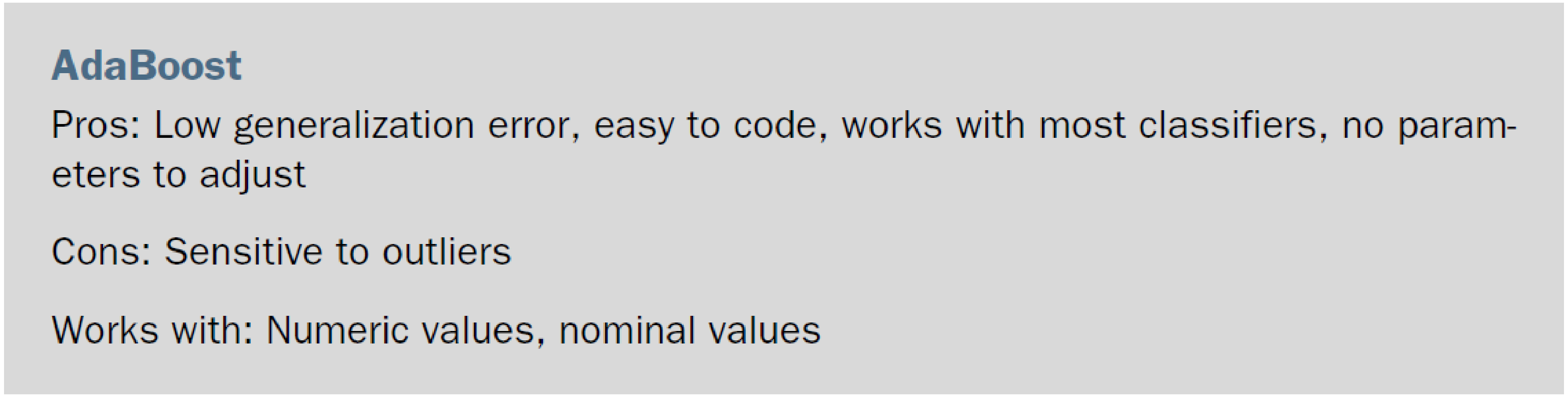
jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

AdaBoost Learning

- Make decision based on multiple experts
- Meta-algorithm: combining other algorithms
- Combining multiple classifiers
  - Also known as ensemble methods

AdaBoost Learning


AdaBoost

Pros: Low generalization error, easy to code, works with most classifiers, no parameters to adjust

Cons: Sensitive to outliers

Works with: Numeric values, nominal values

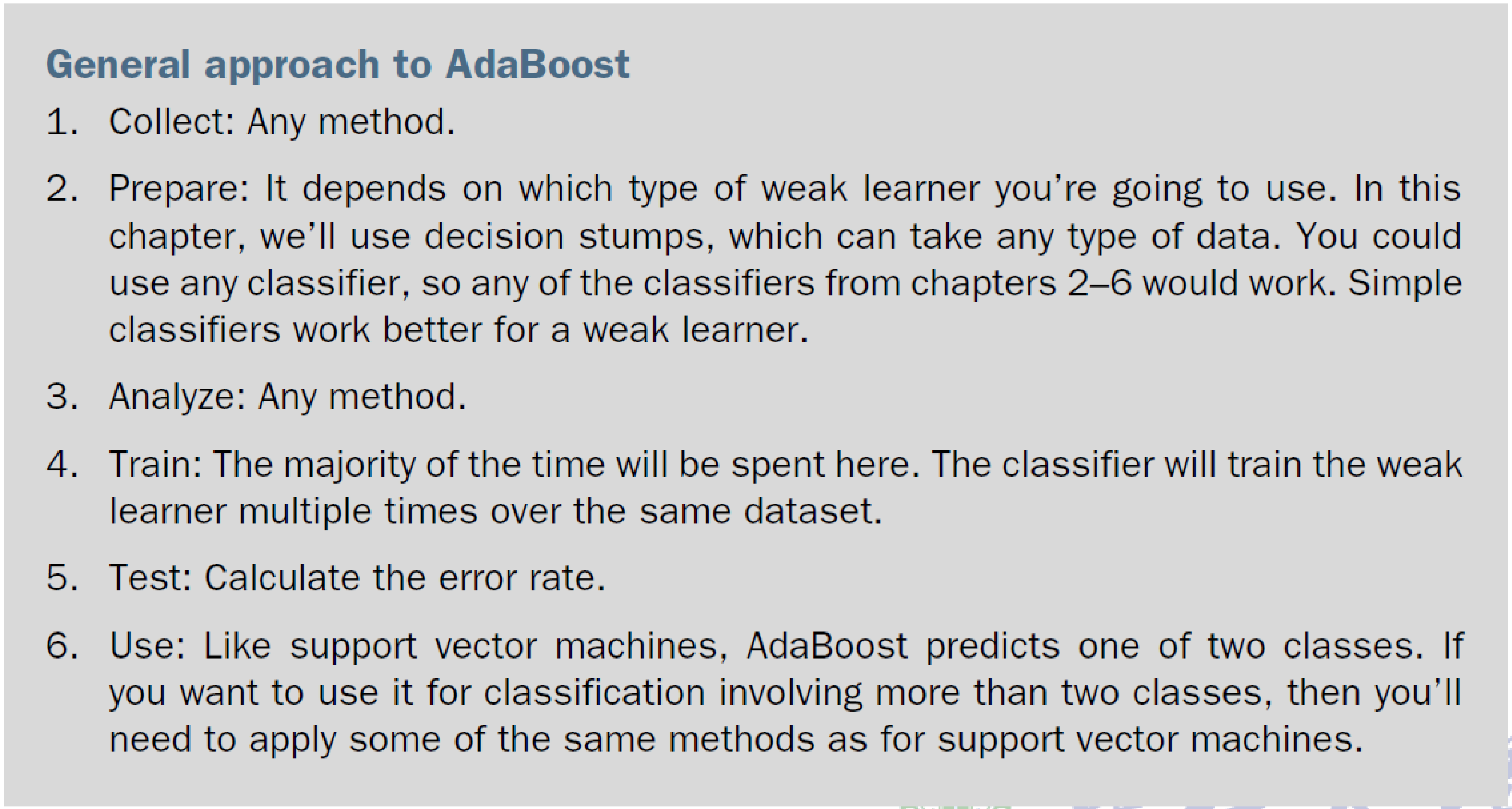
Bagging: Bootstrap Aggregating

- Data is taken from the original dataset S times to make S new datasets
  - Datasets are the same size as the original
  - Each dataset is built by randomly selecting an example with replacement (can select the same example more than once)
- Based on majority vote of classifiers
- Advanced bagging: random forest

Boosting

- Similar to bagging
- Different classifiers are trained sequentially
  - Each new classifier is trained based on the performance of those already trained
  - Makes new classifiers focus on data that was previously misclassified by previous classifiers
  - Output is calculated from a weighted sum of all classifiers
- AdaBoost: adaptive boosting

General Approach



General approach to AdaBoost

1. Collect: Any method.
2. Prepare: It depends on which type of weak learner you're going to use. In this chapter, we'll use decision stumps, which can take any type of data. You could use any classifier, so any of the classifiers from chapters 2–6 would work. Simple classifiers work better for a weak learner.
3. Analyze: Any method.
4. Train: The majority of the time will be spent here. The classifier will train the weak learner multiple times over the same dataset.
5. Test: Calculate the error rate.
6. Use: Like support vector machines, AdaBoost predicts one of two classes. If you want to use it for classification involving more than two classes, then you'll need to apply some of the same methods as for support vector machines.

Train: Improving by Focusing on Errors

- Flow
  - A weight is applied to every example in the training data, weight vector D (initially all equal)
  - A weak classifier is first trained on the training data
  - The errors from the weak classifier are calculated
  - The weak classifier is trained a second time with the same dataset
  - Weights of the training set are adjusted so the examples properly classified the first time are weighted less and the examples incorrectly classified in the first iteration are weighted more

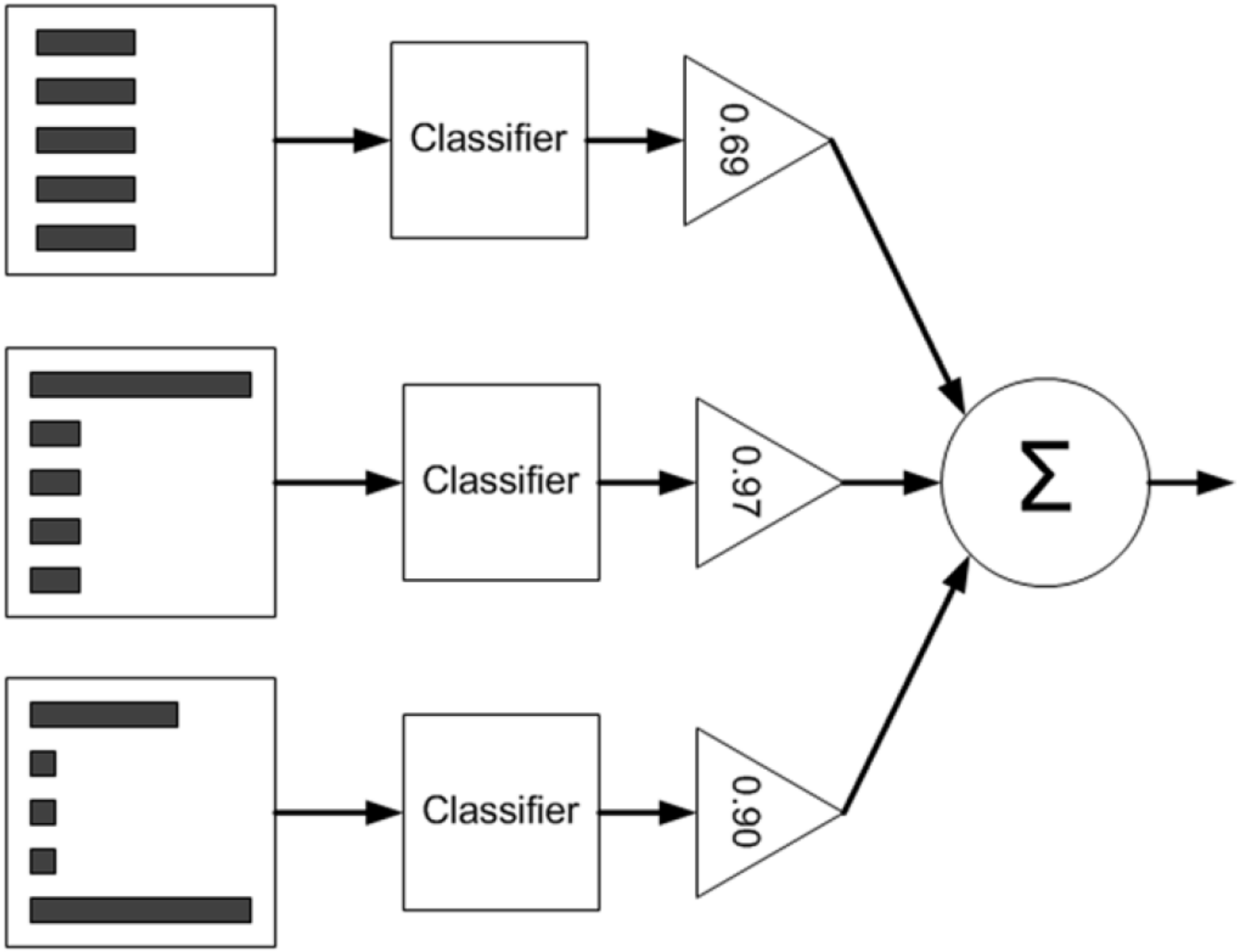
Training

- Calculate answer:
- Assigns α values to each of the classifiers based on the error of each weak classifier

$$\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

$$\varepsilon = \frac{\textit{number of incorrectly classified examples}}{\textit{total number of examples}}$$

Algorithm



Weight Update

- Update training example weight vector D
  - If correctly predicted

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{Sum(D)}$$

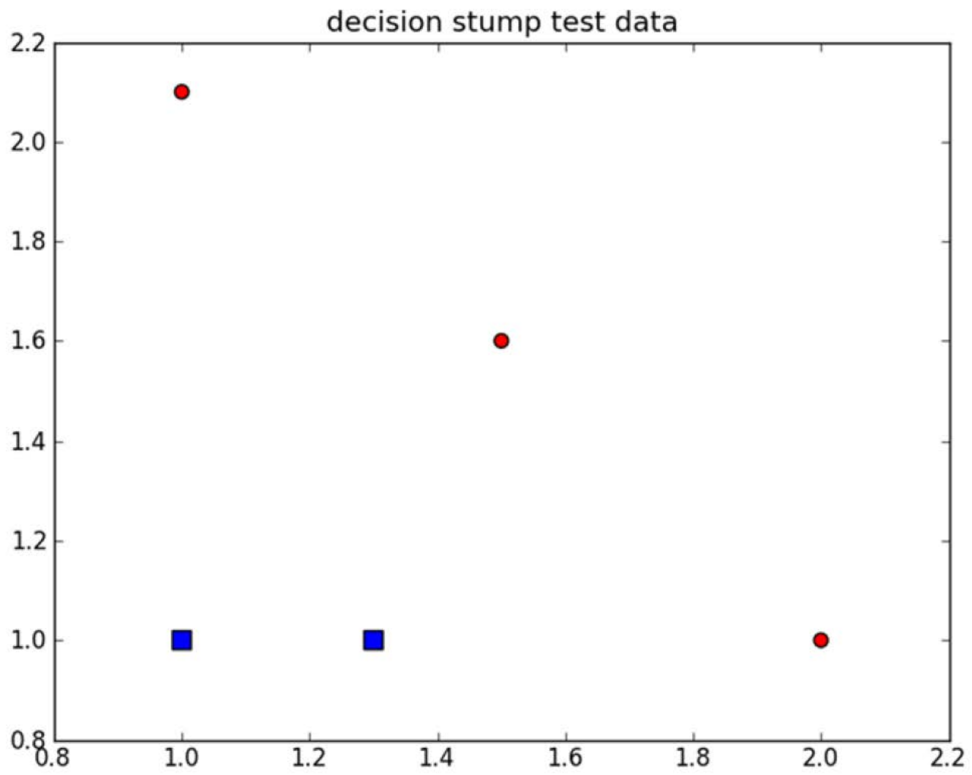
  - If incorrectly predicted

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{Sum(D)}$$

Stopping Condition

- Training error is 0
- The number of weight-adjusting iterations reaches a user-defined value.

Helper Function



```
def loadSimpData():
    datMat = matrix([[ 1. ,  2.1],
        [ 2. ,  1.1],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
  return datMat,classLabels
```

AdaBoost Pseudocode – Part 1

```
Set the minError to $+\infty$
For every feature in the dataset:
    For every step:
        For each inequality:
            Build a decision stump and test it with the weighted dataset
            If the error is less than minError: set this stump as the best stump
Return the best stump
```

Decision Stump–Generating Functions

Listing 7.1 Decision stump–generating functions

```
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = ones((shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray
```

Decision Stump–Generating Functions

```
def buildStump(dataArr,classLabels,D):
    dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
    m,n = shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))
    minError = inf
    for i in range(n):
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps
        for j in range(-1,int(numSteps)+1):
            for inequal in ['lt', 'gt']:
                threshVal = (rangeMin + float(j) * stepSize)
                predictedVals = \
                        stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr = mat(ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                weightedError = D.T*errArr
                #print "split: dim %d, thresh %.2f, thresh ineqal: \
                       %s, the weighted error is %.3f" %\
                       (i, threshVal, inequal, weightedError)
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst
```

Calculate weighted error 1

AdaBoost Pseudocode – Part 2

```
For each iteration:
    Find the best stump using buildStump()
    Add the best stump to the stump array
    Calculate alpha
    Calculate the new weight vector – D
    Update the aggregate class estimate
    If the error rate ==0.0 : break out of the for loop
```

AdaBoost Training

```
def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    weakClassArr = []
    m = shape(dataArr)[0]
    D = mat(ones((m,1))/m)
    aggClassEst = mat(zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)
        print "D:",D.T
        alpha = float(0.5*log((1.0-error)/max(error,1e-16)))
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump)
        print "classEst: ",classEst.T
        expon = multiply(-1*alpha*mat(classLabels).T,classEst)
        D = multiply(D,exp(expon))
        D = D/D.sum()
        aggClassEst += alpha*classEst
        print "aggClassEst: ",aggClassEst.T
        aggErrors = multiply(sign(aggClassEst) !=
                    mat(classLabels).T,ones((m,1)))
        errorRate = aggErrors.sum()/m
        print "total error: ",errorRate,"\n"
        if errorRate == 0.0: break
    return weakClassArr
```

1 **Calculate D for next iteration**

2 **Aggregate error calculation**

AdaBoost Testing

Listing 7.3 AdaBoost classification function

```
def adaClassify(datToClass,classifierArr):
    dataMatrix = mat(datToClass)
    m = shape(dataMatrix)[0]
    aggClassEst = mat(zeros((m,1)))
    for i in range(len(classifierArr)):
        classEst = stumpClassify(dataMatrix,classifierArr[i]['dim'],\
                                 classifierArr[i]['thresh'],\
                                 classifierArr[i]['ineq'])
        aggClassEst += classifierArr[i]['alpha']*classEst
        print aggClassEst
    return sign(aggClassEst)
```

Summary

- Ensemble methods are a way of combining the predictions of multiple classifiers
- Combining multiple classifiers exploits the shortcomings of single classifiers, such as overfitting
- The two types of ensemble methods we discussed are bagging and boosting
- Bagging: datasets the same size as the original dataset are built by randomly sampling examples for the dataset with replacement
- Boosting takes the idea of bagging a step further by applying a different classifier sequentially to a dataset