Boosting

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Step 1: train a new weak (weighted) classifier

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Step 2: collect predictions; compute weighted error rate

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Step 3: compute classifier weight: the log-odds of a (weighted) error

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Step 4: update the weights, focus more weight on those where Y*Yhat = -1

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Step 5: weight normalization

Adaboost

```
for i=1:niter
  C[i] = NewWeakClassifier()
  C[i].train(X, Y, wts)
  Yhat = C[i].predict(X)
  e = wts * (Y == Yhat)
  alpha[i] = 0.5 * log( (1-e) / e)
  wts *= exp(-alpha[i] * Y * Y_hat)
  wts = wts / sum(wts)
end
# Final classifier
sum([alpha[i] * C[i].predict(X_test) for i in 1:niter]) > 0
```

Finally, the classifier does a weighted prediction; using alpha as the weights