Data Mining (W4240 Section 001)
Boosting

Giovanni Motta

Columbia University, Department of Statistics

November 25, 2015

Outline

Outline

Model Fitting

What we have done:

- get data
- fit some models
- evaluate results
- choose best model, apply to test data

Last time and today we consider an alternative approach:

- get data
- fit some models
- evaluate results
- combine models, apply to test data

Approaches of this type are generally called *ensemble methods*

Ensemble Methods

Ensembles methods use collections of models to get better predictive performance than any single model

- get a collection of predictive models
- the ensemble predictor is an average of the underlying models
- we introduced bagging
- we introduced random forests

Why should this work?

- often easy to fit simple models well
- if we average lots of *different* simple models, we can fit these well and have a large model space
- and we can reduce the variance of the estimator

And the gun fires (foreshadowing from last time)

- Bagging: we average all our estimators together
- Don't we believe that some will be more useful than others?
- Could we create a weighted average of estimators?
- Can we learn better estimators as we go?
- This time...

Outline

Strong Learners vs. Weak Learners

Setting: classification with 2 classes (here, -1 and 1)

A **strong learner** is a method that can learn a decision rule arbitrarily well.

A **weak learner** is a simple method that does better than guessing, but cannot learn a decision rule arbitrarily well.

Example: trying to decide whether an email is ham or spam.

- strong learner example: method that uses words, syntax, etc as features, and fits a high-accuracy decision rule
- weak learner example: "If the phrase 'lose weight' is in the email, then predict it is spam"

Boosting

Can we combine weak learners to make a strong learner? If so, how can we do it in a computationally efficient manner?

Boosting

Can we combine weak learners to make a strong learner? If so, how can we do it in a computationally efficient manner?

Answer: yes, we can combine weak learners to make a strong learner with *boosting*

Boosting:

- start with a method for finding weak learners
- call this method repeatedly, each time with *new* subsets of the data
- The ith subset is a random sample of the data with some weight (ex: $p_1 = \frac{1}{2n},\ p_2 = \frac{1}{1.5n}, ...$)
- new predictor is a weighted average of the weak learners
- (note the implicit idea that we are educating ourselves as we go, and weighting more as we learn more)

Boosting

Assuming we have a good way to generate weak learners (success rate more than 50%), we still have some problems:

- how should we reweight the data each round?
- how should we combine the weak rules into a single (strong?) rule?

Want:

- want higher weights on data that have been previously misclassified
- prediction to be a simple weighted majority of rules

Outline

AdaBoost

AdaBoost (**Ada**ptive **Boost**ing) is one way to do this.

Data: $(x_i, y_i)_{i=1}^{n}$, x_i is a vector and $y_i \in \{-1, 1\}$

Inputs: data, number of rounds T, weak learner $\hat{y} = h_t(x)$

Output: decision rule H

Weights:

- start with $\mathcal{D}_1(i) = \frac{1}{n}$ for $i = 1, \ldots, n$
- for $t = 1, \ldots, T$:

=1 || =-1 (i.e. prod=1)

$$\begin{aligned} \mathcal{D}_{t+1}(i) &= \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \text{ (smaller weights for easy examples)} \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \text{ (larger weights for hard examples)} \end{cases} \\ &= \frac{\mathcal{D}_t(i)}{Z_t} e^{-y_i \alpha_t h_t(x_i)} \end{aligned}$$

(here Z_t is a normalization constant so weights sum to 1)
(who can tell me why the second equality holds?)

AdaBoost

Weights: this is an exponential weighting scheme. Easy examples are downweighted, hard examples are upweighted.

What about the predictor, H? It is a weighted linear combination of the weak learners,

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

The weights assigned are directly related to how well h_t performed on the weighted training set:

$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

where

$$\epsilon_t = \mathbb{P}[h_t(x_i) \neq y_i] \approx \sum_{i=1}^{n} \mathcal{D}_t(i)\mathbf{1}_{\{h_t(x_i)\neq y_i\}}$$

AdaBoost: Forward stagewise additive modeling

Consider the problem of fitting a basis function of the form

Final Goal is sgn(f)
Use Fourier Transform
Update everyday!
But T=30~40

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

- data set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $y_i \in \{-1, 1\}$
- set of weak classifiers $\{h_1, \ldots, h_T\}$ each of which outputs a classification $h_t(x_i) \in \{-1, 1\}$ for each item
- At iteration $t - 1$, our boosted classifier is the solution of

minimize loss function

$$(\alpha_t, h_t) \quad = \quad \arg\min_{\alpha,h} \sum_{i=1}^{n} L[y_i,\ f_{(t-1)}(x_i) + \alpha h(x_i)]$$

- At the t-th iteration, we want to extend this to a better boosted classifier:

$$f_t(x_i) = f_{(t-1)}(x_i) + \alpha_t h_t(x_i)$$

AdaBoost: Forward stagewise additive modeling

Choose the exponential loss function: $L[y, f(x)] = \exp[-y\, f(x)]$

- At iteration $t-1$, our boosted classifier is the solution of

$$\begin{aligned} (\alpha_t, h_t) &= \arg\min_{\alpha,h} \sum_{i=1}^{n} \exp\left\{-y_i[f_{(t-1)}(x_i) + \alpha h(x_i)]\right\} \\ &= \arg\min_{\alpha,h} \sum_{i=1}^{n} \mathcal{D}_t(i) \exp\left\{-\alpha y_i h(x_i)\right\} \qquad (1) \end{aligned}$$

  where $\mathcal{D}_t(i) = \exp[-y_i f_{(t-1)}(x_i)]$.
- At the t-th iteration

$$f_t(x_i) = f_{(t-1)}(x_i) + \alpha_t h_t(x_i)$$

The solution to (1) is

$$\begin{aligned} h_t &= \arg\min_{h} \sum_{i=1}^{n} \mathcal{D}_t(i) \mathbf{1}_{y_i \neq h(x_i)} \\ \alpha_t &= \tfrac{1}{2} \ln\left(\tfrac{1-\epsilon_t}{\epsilon_t}\right) \end{aligned}$$

AdaBoost

Algorithm:

- Initialize weights $\mathcal{D}_1(i) = \frac{1}{n}$ for $i = 1, \ldots, n$
- For each round $t = 1, \ldots, T$:
  - draw a "sufficiently large" sample i.i.d. from $\mathcal{D}_t$
  - run weak learning algorithm on this sample to produce rule h_t
  - set $\epsilon_t = \sum_{i=1}^{n} \mathcal{D}_t(i) \mathbf{1}_{\{h_t(x_i) \neq y_i\}}$
  - set $\alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - update distribution by setting

$$\boxed{\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} e^{-\alpha_t y_i h_t(x_i)}}$$

- Output function H where

$$\boxed{H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)}$$

AdaBoost

Why AdaBoost?

- no tunable parameters Unlike Ridge and Lasso has λ
- don't need to know how well weak learners do
- works with any weak learner
- computationally reasonable
- tends to avoid overfitting
- has some nice theoretical guarantees

Pause: **A hugely important method, running all over the world**

Getting Learners

So how can we get weak learners?

- use weak learners that are actually pretty strong—they come from another learning algorithm
  - decision trees
  - naive Bayes
  - k-nn
  - generalized linear models
  - splines
  - etc

Example: decision trees

Each round, AdaBoost gives the algorithm a new set of data and asks it to come up with a new tree with low error. Each tree is an h_t.

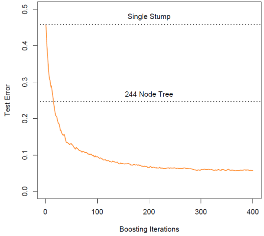
Outline

A simple example

- 10d data: $X_1, ..., X_{10} \sim_{iid} \mathcal{N}(0, 1)$

$$Y = \begin{cases} +1 & if \quad \sum_{i=1}^{10} X_j^2 > \chi^2_{10}(0.5) = 9.34, \\ -1 & \text{otherwise} \end{cases}$$

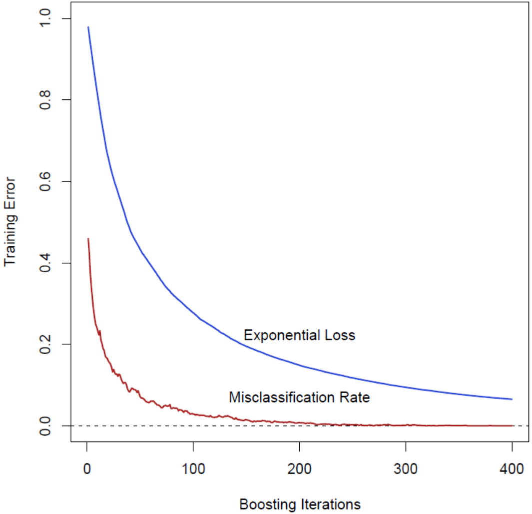
- Classifier is a one-level tree (stump)
- 2000 training points (roughly 50/50), and 10,000 test points
- Error by boosting iterations:



- (let's interpret this classifier...)

A simple example

- Red: misclassification error rate on the training set (red)
- Blue: average exponential loss $\frac{1}{n}\sum_{i=1}^{n}\exp[-y_i f(x_i)]$.



Boosting in R

There are many, many packages that implement boosting in R

- `ada`: stumpy trees from CART
- `gmb`: boosting for regression with trees
- `mboost`: model-based boosting for regression and classification with a variety of methods, including GLMs and splines
- many more....

Boosting in R

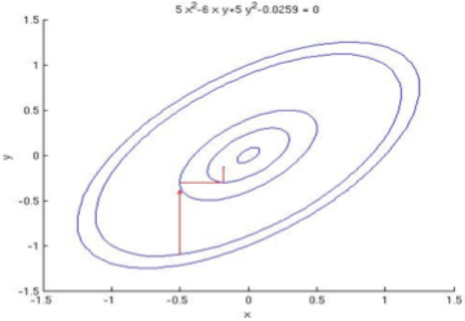
Let's use the package ada on the kyphosis dataset (from rpart).

```
> library(rpart)
> library(ada)
> dim(kyphosis)
> ind.train <- sample(1:81,60)
> ind.test <- setdiff(1:81,ind.train)
> # iter is number of iterations
> # nu is a shrinkage parameter
> ky.ada <-ada(Kyphosis~.,data=kyphosis[ind.train,],iter=20,nu=1,type="discrete")
> # add testing data set
> ky.ada.test < addtest(ky.ada,kyphosis[ind.test,-1],kyphosis[ind.test,1])
> plot(ky.ada.test,TRUE,TRUE)
> ky.ada.test2 <- predict(ky.ada,kyphosis[ind.test,-1],type="vector")
> # Make a tree
> ky.rpart <- rpart(Kyphosis~.,data=kyphosis[ind.train,])
> ky.rpart.test <- predict(ky.rpart,kyphosis[ind.test,-1],type="class")
> cbind(kyphosis[ind.test,1],ky.rpart.test,ky.ada.test2)
```

Outline

A Statistical View of Boosting

Simplified version: AdaBoost is a coordinate descent algorithm



Coordinate descent:

- have function f; take gradient
- move in the *coordinate direction* with largest change
- length of move is α_t; repeat

A Statistical View of Boosting

Misclassification error for some function f:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{\{y_i f(x_i)\leq 0\}}$$

The misclassification error is upper bounded by the exponential loss:

$$\frac{1}{n}\sum_{i=1}^{n}e^{-y_i f(x_i)}$$

...and we want f to be a linear combination of classifiers,

$$f(x)=\sum_{j=1}^{m}\lambda_j h_j(x)$$

So we will minimize the exponential loss with respect to the λ_j's.

A Statistical View of Boosting

Here are the details:

- define an $n \times m$ matrix M with $M_{ij} = y_i h_j(x_i) \in \{\pm 1\}$

$$M = \begin{array}{cc} & \text{weak classifiers } j \\ \text{examples } i & \begin{bmatrix} & & \\ & \pm 1 & \\ & & \end{bmatrix} \end{array}$$

- then

$$y_i f(x_i) = \sum_j \lambda_j y_i h_j(x_i) = \sum_j \lambda_j M_{ij} = (M\lambda)_i$$

- which has the exponential loss

$$R^{train}(\lambda) = \frac{1}{n} \sum_i e^{-y_i f(x_i)} = \frac{1}{n} \sum_i e^{-(M\lambda)_i}$$

A Statistical View of Boosting

Let's do coordinate descent on the exponential loss, R^{train}:

- each iteration, choose a coordinate of λ, called j_t, and move α_t in the j^{th} direction (classifier gives direction, α_t length of step)
- find direction with steepest gradient: (set $\mathbf{e}_j$ as vector of 0's with 1 in j^{th} place)

$$
\begin{aligned}
j_t &\in \arg\max_j \left[- \left. \frac{\partial R^{train}(\lambda_t + \alpha \mathbf{e}_j)}{\partial \alpha} \right|_{\alpha=0} \right] \\
&= \arg\max_j \left[- \left. \frac{\partial}{\partial \alpha} \left[\frac{1}{n} \sum_{i=1}^{n} e^{-(M(\lambda_t + \alpha \mathbf{e}_j))_i} \right] \right|_{\alpha=0} \right] \\
&= \arg\max_j \left[- \left. \frac{\partial}{\partial \alpha} \left[\frac{1}{n} \sum_{i=1}^{n} e^{-(M\lambda_t)_i - \alpha (M\mathbf{e}_j)_i} \right] \right|_{\alpha=0} \right] \\
&= \arg\max_j \left[- \left. \frac{\partial}{\partial \alpha} \left[\frac{1}{n} \sum_{i=1}^{n} e^{-(M\lambda_t)_i - \alpha M_{ij}} \right] \right|_{\alpha=0} \right] \\
&= \arg\max_j \left[\frac{1}{n} \sum_{i=1}^{n} M_{ij} e^{-(M\lambda_t)_i} \right]
\end{aligned}
$$

A Statistical View of Boosting

To deal with

$$j_t \in \arg\max_j \left[\frac{1}{n} \sum_{i=1}^{n} M_{ij} e^{-(M\lambda_t)_i} \right],$$

create a probability distribution

$$\mathcal{D}_t(i) = e^{-(M\lambda_t)_i}/Z_t, \text{ where } Z_t = \sum_{i=1}^{n} e^{-(M\lambda_t)_i}$$

Since we are just multiplying by constants,

$$j_t \in \arg\max_j \sum_{i=1}^{n} M_{ij}\mathcal{D}_t(i) = \arg\max_j (\mathcal{D}_t^T M)_j$$

So that's how we choose the classifier at step t.

A Statistical View of Boosting

So now that we have chosen j_t, how far should we move in that direction? (i.e., how much of that classifier should we add in to the large classifier?)

$$
\begin{aligned}
0 &= \left. \frac{\partial R^{train}(\lambda_t + \alpha \mathbf{e}_{j_t})}{\partial \alpha} \right|_{\alpha_t} \\
&= -\frac{1}{n} \sum_{i=1}^{n} M_{ij_t} e^{-(M\lambda_t)_i - \alpha_t M_{ij_t}} \\
&= -\frac{1}{n} \sum_{i: M_{ij_t}=1} e^{-(M\lambda_t)_i} e^{-\alpha_t} - \frac{1}{n} \sum_{i: M_{ij_t}=-1} -e^{-(M\lambda_t)_i} e^{\alpha_t} \\
&= \sum_{i: M_{ij_t}=1} \mathcal{D}_t(i) e^{-\alpha_t} - \sum_{i: M_{ij_t}=-1} \mathcal{D}_t(i) e^{\alpha_t} \\
&=: d_+ e^{-\alpha_t} - d_- e^{\alpha_t}
\end{aligned}
$$

$$
e^{2\alpha_t} = \frac{d_+}{d_-}, \qquad \alpha_t = \frac{1}{2} \log \frac{d_+}{d_-} = \frac{1}{2} \log \frac{1 - d_-}{d_-}
$$

A Statistical View of Boosting

The new coordinate descent algorithm is:

- set $\mathcal{D}_1(i) = \frac{1}{n}$ for $i = 1, \ldots, n$; $\lambda_1 = 0$
- for $t = 1, \ldots, T$:
  - set $j_t \in \arg\max_j (\mathcal{D}_t^T M)_j$
  - set $d_- = \sum_{M_{ij_t}=-1} \mathcal{D}_t(i)$
  - set $\alpha_t = \frac{1}{2} \log \frac{1-d_-}{d_-}$
  - set $\lambda_{t+1} = \lambda_t + \alpha_t \mathbf{e}_{j_t}$
  - set $\mathcal{D}_{t+1}(i) = e^{-(M\lambda_{t+1})_i} / Z_t$ for each i
  - set $Z_t = \sum_{i=1}^{n} e^{-(M\lambda_{t+1})_i}$
- set $f(x) = \sum_{j=1}^{m} \lambda_j h_j(x)$

A Statistical View of Boosting

The new coordinate descent algorithm is:

- set $\mathcal{D}_1(i) = \frac{1}{n}$ for $i = 1, \ldots, n$; $\lambda_1 = 0$
- for $t = 1, \ldots, T$:
  - set $j_t \in \arg\max_j (\mathcal{D}_t^T M)_j$
  - set $d_- = \sum_{M_{ij_t}=-1} \mathcal{D}_t(i)$
  - set $\alpha_t = \frac{1}{2} \log \frac{1-d_-}{d_-}$
  - set $\lambda_{t+1} = \lambda_t + \alpha_t \mathbf{e}_{j_t}$
  - set $\mathcal{D}_{t+1}(i) = e^{-(M\lambda_{t+1})_i}/Z_t$ for each i
  - set $Z_t = \sum_{i=1}^n e^{-(M\lambda_{t+1})_i}$
- set $f(x) = \sum_{j=1}^m \lambda_j h_j(x)$

Is this AdaBoost?

A Statistical View of Boosting

How does λ_t relate to α_t?

- $\lambda_{T,j}$ is the sum of the α_t's where the chosen direction is j

$$\lambda_{T,j} = \sum_{t=1}^{T} \alpha_t \mathbf{1}_{\{j_t = j\}}$$

The output function f is

$$\begin{aligned} f(x) &= \sum_{j=1}^{m} \sum_{t=1}^{T} \lambda_{t,j} h_j(x) \\ &= \sum_{j=1}^{m} \sum_{t=1}^{T} \alpha_t \mathbf{1}_{\{j_t = j\}} h_j(x) \\ &= \sum_{t=1}^{T} \alpha_t \sum_{j=1}^{m} h_j(x) \mathbf{1}_{\{j_t = j\}} \\ &= \sum_{t=1}^{T} \alpha_t h_{j_t}(x) \end{aligned}$$

A Statistical View of Boosting

OK, so the output function has a similar form. What about the weights?

Weights for AdaBoost:

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)e^{-M_{ij_t}\alpha_t}}{Z_t} \qquad = \frac{\prod_t e^{-M_{ij_t}\alpha_t}}{n\prod_t Z_t}$$

$$= \frac{e^{-\sum_t M_{ij_t}\alpha_t}}{n\prod_t Z_t} \qquad = \frac{1}{n\prod_t Z_t} e^{-\sum_j M_{ij}\lambda_{t,j}}$$

The denominator must be $\sum_i e^{-\sum_j M_{ij}\lambda_{t,j}}$ since the weights sum to 1.

Therefore the weights are the same, as long as the j_t's and α_t's are the same.

A Statistical View of Boosting

Let's check the chosen directions, the j_t's. AdaBoost chooses the classifier with the lowest error. In terms of the matrix M:

$$
\begin{aligned}
j_t \in \arg\min_j \sum_i \mathcal{D}_t(i) \mathbf{1}_{\{h_j(x_i) \neq y_i\}} \\
&= \arg\min_j \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) \\
&= \arg\max_j \left[- \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) \right] \\
&= \arg\max_j \left[1 - 2 \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) \right] \\
&= \arg\max_j \left[\left[\sum_{i:M_{ij}=1} \mathcal{D}_t(i) + \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) \right] - 2 \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) \right] \\
&= \arg\max_j \sum_{i:M_{ij}=1} \mathcal{D}_t(i) - \sum_{i:M_{ij}=-1} \mathcal{D}_t(i) = \arg\max_j (\mathcal{D}_t^T M)_j
\end{aligned}
$$

A Statistical View of Boosting

OK, so we have the same weights and the same chosen direction. Are the step sizes the same? AdaBoost error rate:

$$
\begin{aligned}
\epsilon_t &= \sum_i \mathcal{D}_t(i) \mathbf{1}_{\{h_{j_t}(x_i) \neq y_i\}} \\
&= \sum_{i : h_{j_t}(x_i) \neq y_i} \mathcal{D}_t(i) \\
&= \sum_{i : M_{ij_t} = -1} \mathcal{D}_t(i) = d_-
\end{aligned}
$$

Working our way from the AdaBoost stepsize:

$$
\begin{aligned}
\alpha_t &= \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \\
&= \frac{1}{2} \log \frac{1 - d_-}{d_-}
\end{aligned}
$$

Conclusion: AdaBoost minimizes exponential loss using coordinate descent.

Outline

AdaBoost and Logistic Regression

Logistic regression has a logistic loss function. How is it similar to AdaBoost?

- AdaBoost approximately minimizes exponential loss over the whole distribution
- We can use this idea to get label probabilities from AdaBoost

Lemma (Hastie, Friedman, Tibshirani, 2001)

$$\mathbb{E}_{Y\sim\mathcal{D}(x)} e^{-Yf(x)}$$

is minimized at

$$f(x) = \frac{1}{2}\log\frac{\mathbb{P}(Y=1\,|\,x)}{\mathbb{P}(Y=-1\,|\,x)};$$

AdaBoost and Logistic Regression

Proof.

$$
\begin{aligned}
\mathbb{E}e^{-Yf(x)} &= \mathbb{P}(Y=1\,|\,x)e^{-f(x)} + \mathbb{P}(Y=-1\,|\,x)e^{f(x)} \\
0 = \frac{d\,\mathbb{E}(e^{-Yf(x)}\ \ x)}{d\,f(x)} &= -\mathbb{P}(Y=1\,|\,x)e^{-f(x)} + \mathbb{P}(Y=-1\,|\,x)e^{f(x)} \\
\mathbb{P}(Y=1\,|\,x)e^{-f(x)} &= \mathbb{P}(Y=-1\,|\,x)e^{f(x)} \\
\frac{\mathbb{P}(Y=1\,|\,x)}{\mathbb{P}(Y=-1\,|\,x)} &= e^{2f(x)} \\
f(x) &= \frac{1}{2}\log\frac{\mathbb{P}(Y=1\,|\,x)}{\mathbb{P}(Y=-1\,|\,x)}
\end{aligned}
$$

$\square$

AdaBoost and Logistic Regression

How does this compare with logistic regression?

Logistic regression:

$$f(x) = \log \frac{\mathbb{P}(Y = 1 \,|\, x)}{\mathbb{P}(Y = -1 \,|\, x)}$$

so we are off by a factor of 2.

Use this fact and the previous lemma to get probabilities out of AdaBoost by solving for $p = \mathbb{P}(Y = 1 \,|\, x)$.

AdaBoost and Logistic Regression

$$
\begin{aligned}
f(x) &= \frac{1}{2} \log \frac{\mathbb{P}(Y = 1 \,|\, x)}{\mathbb{P}(Y = -1 \,|\, x)} =: \frac{1}{2} \log \frac{p}{1-p} \\
e^{2f(x)} &= \frac{p}{1-p} \\
e^{2f(x)} - pe^{2f(x)} &= p \\
e^{2f(x)} &= p(1 + e^{2f(x)}) \\
p = \mathbb{P}(Y = 1 \,|\, x) &= \frac{e^{2f(x)}}{1 + e^{2f(x)}}
\end{aligned}
$$

Even though AdaBoost minimizes a different objective function, the probability of success is similar to that of logistic regression (remove the 2's).

Outline

Boosting

What I want you to get from this:

- connection between weak learners and strong learners
- what the algorithm is (you will be implementing it in HW)
- general idea of why it works