Linear Classifiers/SVMs

David Kauchak, CS311, Spring 2013

Admin

- Midterm exam posted
- Assignment 4 due Friday by 6pm
- No office hours tomorrow

Math

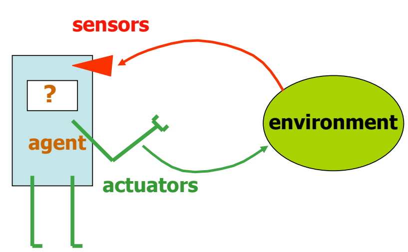
Machine learning often involves a lot of math

– some aspects of AI also involve some familiarity

Don't let this be daunting

– Many of you have taken more math than me
– Gets better over time
– Often, just have to not be intimidated

Learning



As an agent interacts with the world, it should learn about it's environment

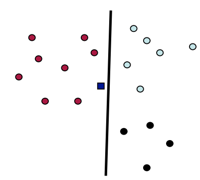
Quick review

- Three classifiers
  - Naïve Bayes
  - k-nearest neighbor
  - decision tree
  - good and bad?
- Bias vs. variance
  - a measure of the **model**
  - where do NB, k-nn and decision trees fit on the bias/variance spectrum?

Separation by Hyperplanes

A strong high-bias assumption is *linear separability*:

- in 2 dimensions, can separate classes by a line
- in higher dimensions, need hyperplanes



Hyperplanes

A hyperplane is line/plane in a high dimensional space

What defines a hyperplane?
What defines a line?

Hyperplanes

A hyperplane in an n-dimensional space is defined by n+1 values

$$0 = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n + w_{n+1}$$

e.g. a line

$$0 = w_1 f_1 + w_2 f_2 + w_3$$ **f(x) = ax+b**

or a plane

$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + w_4$$ **f(x,y) = ax+by + c**

NB as a linear classifier

To classify:

$$\operatorname{argmax}_C P(C \mid f_1, f_2, \ldots, f_n)$$

Another way to view this (for 2 classes):

$$d(f_1, f_2, \ldots, f_n) = \frac{P(c_1 \mid f_1, f_2, \ldots, f_n)}{P(c_2 \mid f_1, f_2, \ldots, f_n)}$$

Given d how would we classify?

NB as a linear classifier

$$d(f_1, f_2, \ldots, f_n) = \frac{P(c_1 \mid f_1, f_2, \ldots, f_n)}{P(c_2 \mid f_1, f_2, \ldots, f_n)}$$

To classify:

$$classify(f_1, f_2, \ldots, f_n) = \begin{cases} c_1 \text{ if } d > 1 \\ c_2 \text{ if } d < 1 \end{cases}$$

We can take the log:

$$classify(f_1, f_2, \ldots, f_n) = \begin{cases} c_1 \text{ if } \log d > 0 \\ c_2 \text{ if } \log d < 0 \end{cases}$$

NB as a linear classifier

$$\log d(f_1, f_2, \ldots, f_n) = \log \frac{P(c_1 \mid f_1, f_2, \ldots, f_n)}{P(c_2 \mid f_1, f_2, \ldots, f_n)}$$

$$= \log \frac{P(f_1 \mid c_1) P(f_2 \mid c_1) \ldots P(f_n \mid c_1) p(c_1)}{P(f_1 \mid c_2) P(f_2 \mid c_2) \ldots P(f_n \mid c_2) p(c_2)}$$

$$= \log P(c_1) - \log P(c_2) + \sum_{i=1}^{n} \log P(f_i \mid c_1) - \log P(f_i \mid c_2)$$

NB as a linear classifier

$$= \log P(c_1) - \log P(c_2) + \sum_{i=1}^{n} \log P(f_i \mid c_1) - \log P(f_i \mid c_2)$$

$$= \underbrace{\log P(c_1) - \log P(c_2)}_{w_{n+1}} + \sum_{i=1}^{n} \underbrace{f_i}_{\text{binary features}} \underbrace{(\log P(f_i \mid c_1) - \log P(f_i \mid c_2))}_{\text{weight for that dimension}}$$

$$\boxed{0 = w_1 f_1 + w_2 f_2 + \ldots + w_n x_n + w_{n+1}}$$

Lots of linear classifiers

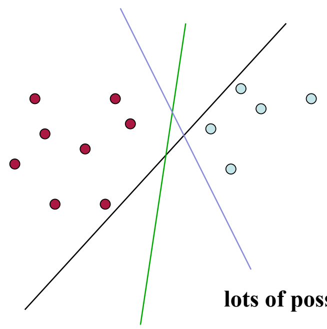
Many common text classifiers are linear classifiers

- Naïve Bayes
- Perceptron
- Rocchio
- Logistic regression
- Support vector machines (with linear kernel)
- Linear regression

Despite this similarity, noticeable performance difference

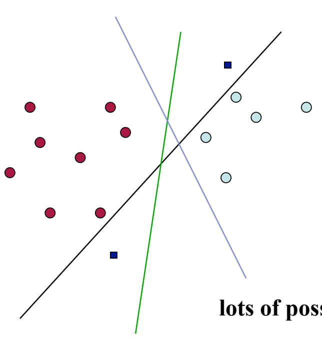
How might algorithms differ?

Which Hyperplane?



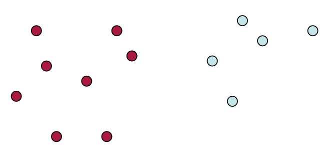
lots of possible solutions

Which Hyperplane?

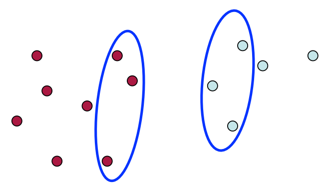


lots of possible solutions

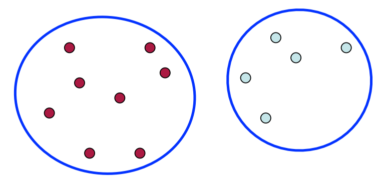
Which examples are important?



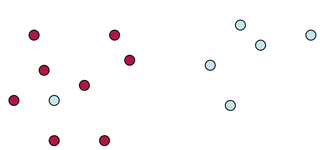
Which examples are important?



Which examples are important?

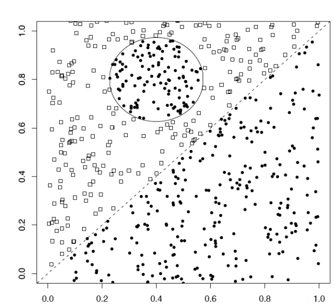


Dealing with noise



linearly separable?

A nonlinear problem



A linear classifier like Naïve Bayes does badly on this task

k-NN will do very well (assuming enough training data)

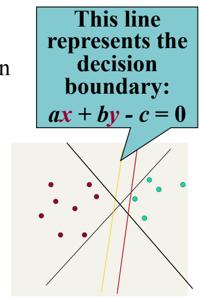
20

Linear classifiers: Which Hyperplane?

Lots of possible solutions for *a,b,c*

Support Vector Machine (SVM) finds an optimal solution

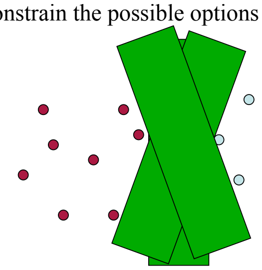
– Maximizes the distance between the hyperplane and the “difficult points” close to decision boundary



Another intuition

Think of it as trying to place a wide separator between the points.

Will constrain the possible options



22

Support Vector Machine (SVM)

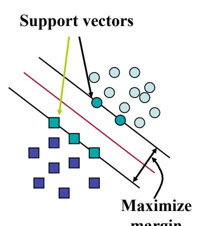
SVMs maximize the *margin* around the separating hyperplane

- aka large margin classifiers

specified by a subset of training samples, *the support vectors*

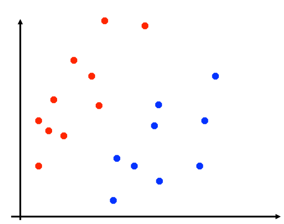
Posed as a *quadratic programming* problem

Seen by many as the most successful current text classification method*

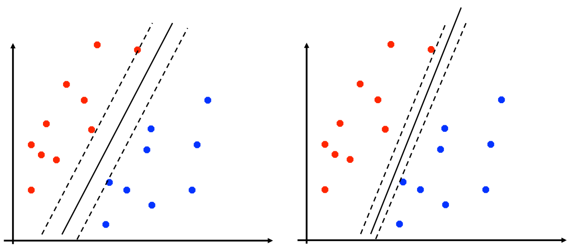


***but other discriminative methods often perform very similarly**

Margin maximization

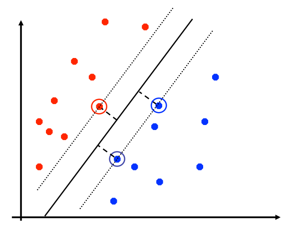


Margin maximization



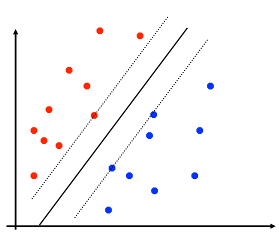
Measuring the margin

The *support vectors* define the hyperplane and the margin



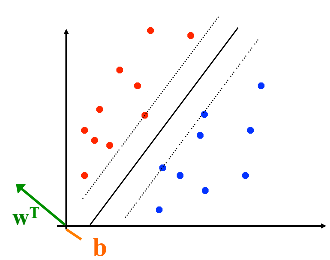
Measuring the margin

How do we classify points given the hyperplane?



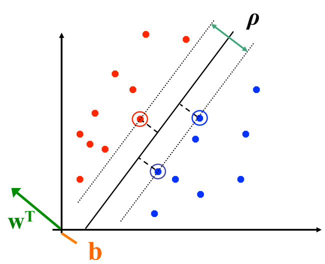
Measuring the margin

$$f(x_i) = \text{sign}(w^T x_i + b)$$



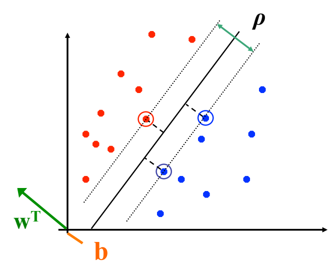
Measuring the margin

How can we calculate margin?



Measuring the margin

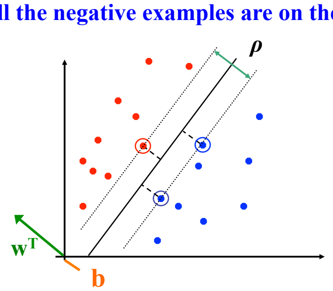
Minimum of the distance from the hyperplane to any point(s) (specifically the support vectors)



Basic SVM setp

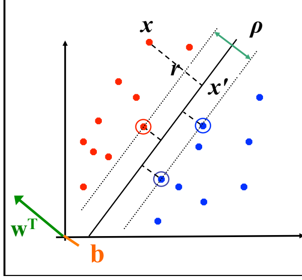
Find the largest margin hyperplane where:

- all the positive examples are on one side
- all the negative examples are on the other side



Measuring the margin

Want to calculate r



x' – x is perpendicular to hyperplane

w/|w| is the unit vector in direction of w

x' = x – rw/|w|

x' satisfies $w^T x' + b = 0$ because it's on w^T

So $w^T(x - rw/|w|) + b = 0$

$w^T x - w^T rw/|w| + b = 0$

$w^T x - w^T rw|w|/|w||w| + b = 0$

$w^T x - w^T rw|w|/w^T w + b = 0$

$w^T x - r|w| + b = 0$

$|w| = \mathrm{sqrt}(w^T w)$

$$\boxed{r = y\frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}}$$

Linear SVM Mathematically

The linearly separable case

Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x}_i, y_i)\}$

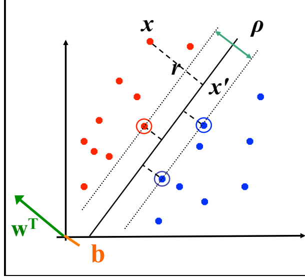
$\mathbf{w}^T\mathbf{x}_i + b \geq 1$ if $y_i = 1$ — positive examples on one side

$\mathbf{w}^T\mathbf{x}_i + b \leq -1$ if $y_i = -1$ — negative examples on the other side

Measuring the margin

$\mathbf{w}^T\mathbf{x}_i + b \geq 1$ if $y_i = 1$

$\mathbf{w}^T\mathbf{x}_i + b \leq -1$ if $y_i = -1$



The support vectors are those that define the hyperplane. They're the "boderline" cases where this weight is exactly 1.

Then, since each example's distance from the hyperplane is

$$r = y\frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

The margin is:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Linear SVMs Mathematically (cont.)

Then we can formulate the *quadratic optimization problem:*

Find w and b such that $\rho = \frac{2}{\|\mathbf{w}\|}$ is maximized; **maximize margin**

for all $\{(\mathrm{x}_i, y_i)\}$

$\mathrm{w}^T\mathrm{x}_i + b \geq 1$ if $y_i=1$; $\mathrm{w}^T\mathrm{x}_i + b \leq -1$ if $y_i = -1$

make sure points are on correct size

Linear SVMs Mathematically (cont.)

A better formulation (min $\|\mathbf{w}\|$ = max $1/\|\mathbf{w}\|$):

Find w and b such that

$\Phi(\mathrm{w}) = \mathrm{w}^T\mathrm{w}$ is minimized;

and for all $\{(\mathrm{x}_i, y_i)\}$: $y_i(\mathrm{w}^T\mathrm{x}_i + b) \geq 1$

Solving the Optimization Problem

Find w and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized;
and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

This is a *quadratic* function subject to *linear* constraints

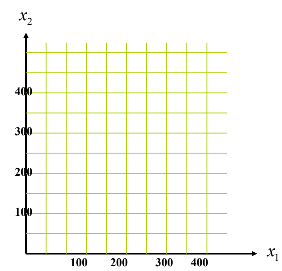
Quadratic optimization problems are well-known

Many ways exist for solving these

An LP example

maximize $x_1 + 6x_2$
subject to

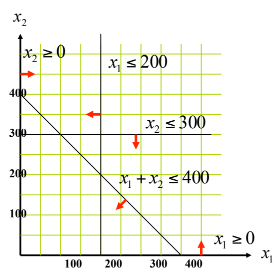
$$\begin{array}{rcl} x_1 & \leq & 200 \\ x_2 & \leq & 300 \\ x_1 + x_2 & \leq & 400 \\ x_1, x_2 & \geq & 0 \end{array}$$



An LP example

maximize $x_1 + 6x_2$
subject to

$$\begin{array}{rcl} x_1 & \leq & 200 \\ x_2 & \leq & 300 \\ x_1 + x_2 & \leq & 400 \\ x_1, x_2 & \geq & 0 \end{array}$$

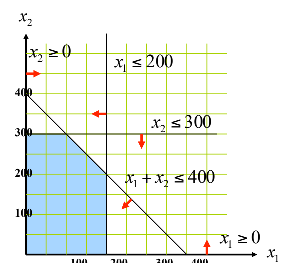


Where is the feasibility region?

An LP example

maximize $x_1 + 6x_2$
subject to

$$\begin{array}{rcl} x_1 & \leq & 200 \\ x_2 & \leq & 300 \\ x_1 + x_2 & \leq & 400 \\ x_1, x_2 & \geq & 0 \end{array}$$

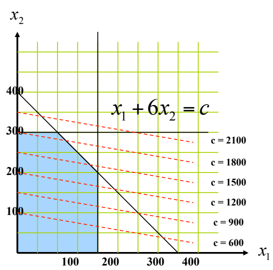


An LP example

maximize $x_1 + 6x_2$

subject to

$$\begin{array}{rcl} x_1 & \leq & 200 \\ x_2 & \leq & 300 \\ x_1 + x_2 & \leq & 400 \\ x_1, x_2 & \geq & 0 \end{array}$$

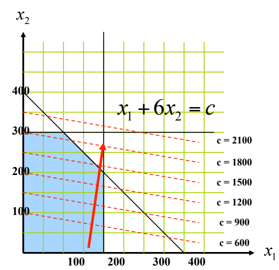


An LP example

maximize $x_1 + 6x_2$

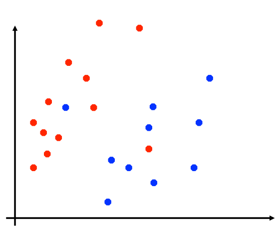
subject to

$$\begin{array}{rcl} x_1 & \leq & 200 \\ x_2 & \leq & 300 \\ x_1 + x_2 & \leq & 400 \\ x_1, x_2 & \geq & 0 \end{array}$$



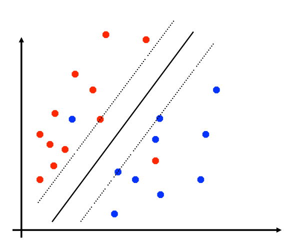
to maximize, move as far in this direction as the constraints allow

Soft Margin Classification



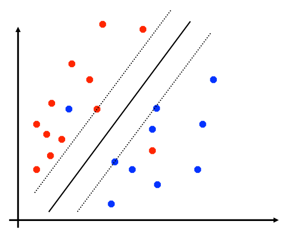
What about this problem?

Soft Margin Classification



Like to learn something like this, but our constraints won't allow it ☹

Soft Margin Classification



Slack variables: allow it to make some mistakes, but penalize it

Soft Margin Classification Mathematically

Old:

Find **w** and b such that
$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$
$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

With slack variables:

Find **w** and b such that
$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$
$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- allows us to make a mistake, but penalizes it
- C trades off noisiness vs. error

Linear SVMs: Summary

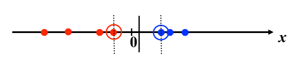
Classifier is a *separating hyperplane*
– large margin classifier: learn a hyperplane that maximally separates the examples

Most "important" training points are support vectors; they define the hyperplane

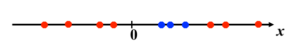
Quadratic optimization algorithm

Non-linear SVMs

Datasets that are linearly separable (with some noise) work out great:

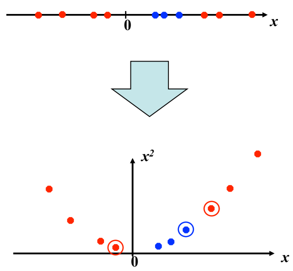


But what are we going to do if the dataset is just too hard?



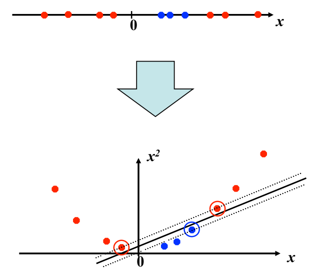
Non-linear SVMs

How about … mapping data to a higher-dimensional space:



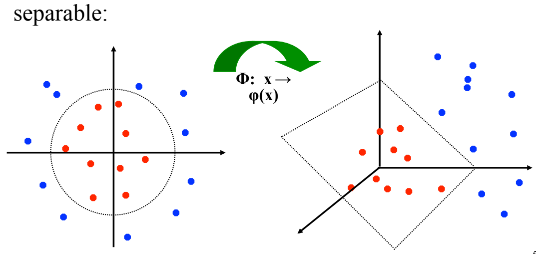
Non-linear SVMs

How about … mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

General idea: map original feature space to higher-dimensional feature space where the training set is separable:



51

The "Kernel Trick"

The linear classifier relies on an inner product between vectors $K(\mathbf{x_i},\mathbf{x_j})=\mathbf{x_i}^\mathbf{T}\mathbf{x_j}$

If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x_i},\mathbf{x_j})= \varphi(\mathbf{x_i})^\mathbf{T}\varphi(\mathbf{x_j})$$

A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

Kernels

Why use kernels?

- Make non-separable problem separable.
- Map data into better representational space

Common kernels

- Linear
- Polynomial $K(x,z) = (1+x^Tz)^d$
  - Gives feature conjunctions
- Radial basis function (infinite dimensional space)

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$$

Demo

http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml

SVM implementations

SVMLight (C)

SVMLib (Java)

Switching gears: weighted examples

Are all examples equally important?

Weak classifiers

Sometimes, it can be intractable (or very expensive) to train a full classifier

However, we can get some information using simple classifiers

A ***weak classifier*** is any classifier that gets more than half of the examples right

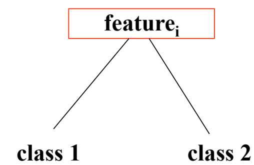
- – not that hard to do
- – a weak classifier does better than random

- Ideas?

Decision stumps

A decision stump is a common weak classifier

Decision stump: 1 level decision tree:



Ensemble methods

If one classifier is good, why not 10 classifiers, or 100?

Ensemble methods combine different classifiers in a reasonable way to get at a better solution

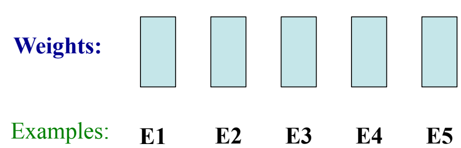
- – similar to how we combined heuristic functions

Boosting is one approach that combines multiple weak classifiers

Boosting

Start with equal weighted examples

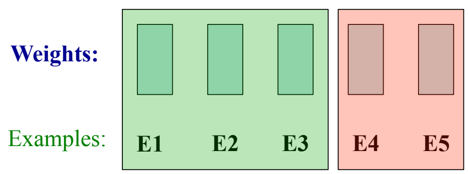
Learn a *weak* classifier



Boosting

Weak$_1$

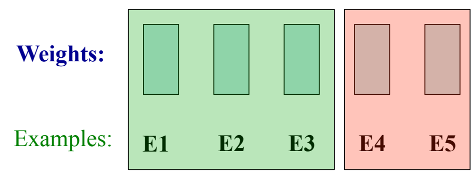
It will do well on some of our training examples and not so well on others



Boosting

Weak$_1$

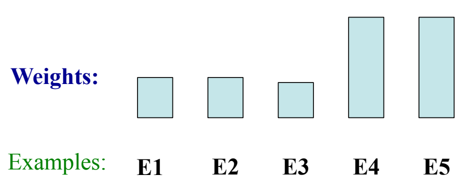
We'd like to reweight the examples and learn another weak classifier. Ideas?



Boosting

Weak$_1$

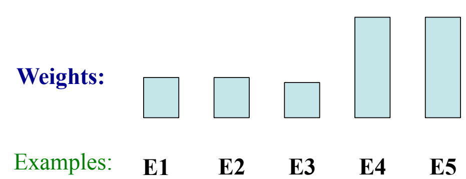
Downweight ones that we're doing well, and upweight those that we're having problems with



Boosting

Weak$_1$

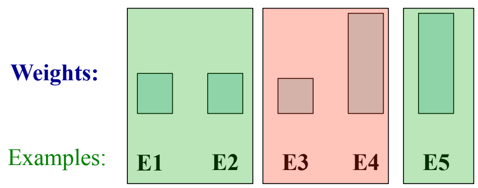
Learn a new classifier based on the new set of weighted examples



Boosting

Weak$_1$ Weak$_2$

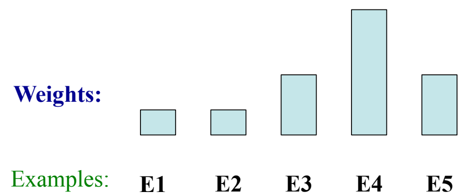
Learn a new classifier based on the new set of weighted examples



Boosting

Weak$_1$ Weak$_2$

Reweight again:



Boosting

Continue this for some number of "rounds"
- at each round we learn a new weak classifier
- and then reweight the examples again

Our final classifier is a weighted combination of these weak classifiers

Adaboost is one common version of boosting
- specifies how to reweight and how to combine learned classifiers
- nice theoretical guarantees
- tends not to have problems with overfitting

http://cseweb.ucsd.edu/classes/fa01/cse291/AdaBoost.pdf

Classification: concluding thoughts

Lots of classifiers out there
- SVMs work very well on broad range of settings

Many challenges still:
- coming up with good features
- preprocessing
- picking the right kernel
- learning hyper parameters (e.g. C for SVMs)

Still a ways from computers "learning" in the traditional sense