COMP 6321
Machine Learning

Instructor: Adam Krzyżak
Email: krzyzak@cs.concordia.ca

Lecture 5: Ensemble classifiers. Bagging. Boosting. Perceptron. Large Margin Classifiers. Linear Support Vector Machines

- Bagging
- Idea of boosting
- AdaBoost algorithm (Freund and Schapire)
- Why does boosting work?
- Perceptrons
  - Definition
  - Perceptron learning rule
  - Convergence
- Margin & max margin classifiers
- (Linear) support vector machines
  - Formulation as optimization problem
  - Generalized Lagrangian and dual
  - Allowing for noise (soft margins)
  - Solving the dual: SMO

Ensemble learning in general

- Ensemble learning algorithms work by running a *base learning algorithm* multiple times, then *combining* the predictions of the different hypotheses obtained using some form of voting
- One approach is to construct several classifiers *independently*, then combine their predictions. Examples include:
  - Bagging
  - Randomizing the test selection in decision trees
  - Using a different subset of input features to train different neural nets
- A second approach is to *coordinate* the construction of the hypotheses in the ensemble.

Extremely randomized trees (Geurts et al, 2005)

- Construct M decision trees
- Instead of searching exhaustively for the best test at a node, pick K attributes at random (without replacement) and pick a random test involving each attribute
- All tests get evaluated (using a normalized information gain metric) and the best one is installed
- The smaller K is, the more randomized the trees are
- The process continues until a desired depth or a desired number of instances at the leaf is reached
- Very reliable method in both classification and regression, small K is best, especially with large levels of noise.
- The smaller the K, the more bias and less variance we get in each tree, but the variance gets washed out by averaging over trees.

Measuring bias and variance in practice

- Recall that bias and variance are both defined as expectations:

$$Bias(\mathbf{x}) = E_P[f(\mathbf{x}) - \overline{h}(\mathbf{x})]$$
$$Var(\mathbf{x}) = E_P[(h(\mathbf{x}) - \overline{h}(\mathbf{x}))^2]$$

- To get expected values we *simulated* multiple data sets, by drawing with samples with replacement from the original data set
- This gives a set of hypothesis, whose predictions can be *averaged* together

Bootstrap replicates

- Given data set D, construct a *bootstrap replicate* of D, called D_b, which has the same number of examples, by drawing samples from D *with replacement*
- Use the learning algorithm to construct a hypothesis h_b by training on D_b
- Compute the prediction of h_b on each of the *remaining* points, from the set $T_b = D - D_b$
- This process is repeated B times, where B is typically a few hundred
- If D is very large, the replicates should contain $m < |D|$ points (still drawn with replacement)

Estimating bias and variance

- For each point, we have a set of estimates $h_1(\mathbf{x}), ...h_K(\mathbf{x})$, with $K \leq B$
- The average prediction, determined empirically, is:

$$\overline{h}(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} h_k(\mathbf{x})$$

- We will estimate the bias as:

$$y - \overline{h}(\mathbf{x})$$

- We estimate the variance as:

$$\frac{1}{K-1}\sum_{k=1}^{K}(\overline{h}_k(\mathbf{x}) - h_k(\mathbf{x}))^2$$

Approximations

- Bootstrap replicates are not real data
- We typically ignore the noise
- If we had multiple points with the same x value, we can estimate the noise
- Alternatively, we can do an estimation using similar points, if this appropriate

Bagging: Bootstrap aggregation

- If we did all the work to get the hypotheses h_b, why not use all of them to make a prediction?
- All hypotheses can have a vote, in the classification case, and we pick the majority class
- For regression, we can average all the predictions
- Which hypotheses classes would benefit most from this approach?

Estimated bias and variance of bagging

- According with our way of estimating variance and bias, bagging eliminates variance altogether!
- In practice, bagging tends to reduce variance and increase bias
- Hence, the main benefit is for unstable learners, i.e., learners with high variance.
- This includes complex hypotheses classes, e.g. decision trees (even unpruned), neural networks, nearest-neighbor-type methods

Additive models

- In an ensemble, the output on any instance is computed by averaging the outputs of several hypotheses, possibly with a different weighting.
- Hence, we should choose the individual hypotheses and their weight in such a way as to provide a good fit
- This suggests that instead of constructing the hypotheses independently, we should construct them such that new hypotheses focus on instances that are problematic for existing hypotheses.
- *Boosting* is an algorithm implementing this idea

Main idea of boosting

- Instead of always treating all data points as equal, component classifiers should *specialize* on certain examples.
- In particular, *if an example is difficult, more components should focus on it*
- Algorithm outline
  - Examine the training set
  - Derive some rough "rule of thumb"
  - *Re-weight* the examples of the training set, concentrating on "hard" cases for the previous rule
  - Derive a second rule of thumb
  - And so on... (repeat this T times)
  - *Combine* the rules of thumb into a single, accurate predictor
- Questions:
  - How do we re-weight the examples?
  - How do we combine the rules into a single classifier?

Notation

- Assume that examples are drawn independently from some probability distribution P on the set of possible data $\mathcal{D}$
- Notation: $J_P(h)$ is the expected error of h when data is drawn from P:

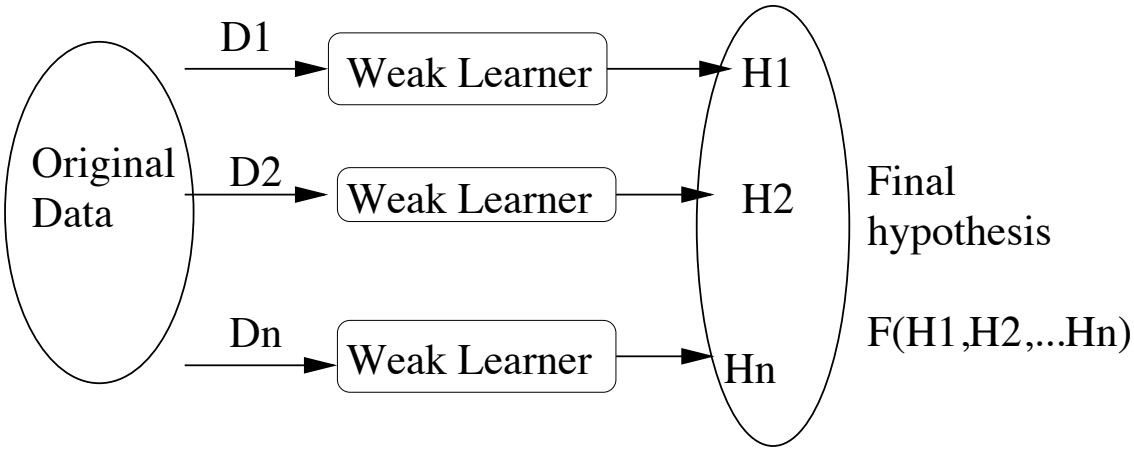
$$J_P(h) = \sum_{\langle \mathbf{x}, y \rangle} J(h(\mathbf{x}), y) P(\langle \mathbf{x}, y \rangle)$$

where $J(h(\mathbf{x}), y)$ could be squared error, or $0/1$ loss

Weak learners

- Assume we have some "weak" binary classifiers (e.g., decision stumps: $x_i > t$)
- "Weak" means $J_P(h) < 1/2 - \gamma$ where $\gamma > 0$ (i.e., the true error of the classifier is better than random).

Boosting classifier



AdaBoost (Freund & Schapire, 1995)

1. Input N training examples $\{(\mathbf{x_1}, y_1), \ldots (\mathbf{x_N}, y_N)\}$, where $\mathbf{x_i}$ are the inputs and y_i is the desired class label
2. Let $D_1(\mathbf{x_i}) = \frac{1}{N}$ (we start with a uniform distribution)
3. Repeat T times:
   (a) Construct D_{t+1} from D_t (details in a moment)
   (b) Train a new hypothesis h_{t+1} on distribution D_{t+1} by minimizing empirical error (1)
4. Construct the final hypothesis:

$$h_f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right),$$

Constructing the new distribution

- We want data on which we make mistakes to be emphasized:

$$D_{t+1}(\mathbf{x_i}) = \frac{1}{Z_t} D_t(\mathbf{x_i}) \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\mathbf{x_i}) = y_i \\ \exp(\alpha_t), & \text{otherwise} \end{cases}$$

  where Z_t is a normalization factor set such that probabilities $D_{t+1}(x_i)/Z_t$ sum to 1.

- Construct the final hypothesis:

$$h_f(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

How to choose α_t?

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

where

$$\epsilon_t = \frac{\sum_{i=1}^{N} D_t(x_i) I_{(h_t(x_i) \neq y_i)}}{\sum_{i=1}^{N} D_t(x_i)} \quad (1)$$

and I_A is an indicator function of set A, i.e., $I_A(x) = 1$ whenever $x \in A$ and $I_A(x) = 0$ otherwise.

- This allows the error to get squashed exponentially
- Note that

$$\epsilon_t < 1/2 \Rightarrow \frac{1 - \epsilon_t}{\epsilon_t} > 1 \Rightarrow \alpha_t > 0$$

so D_{t+1} is increasing on misclassified samples ($y_i h_t(X_i) \leq 0$) and decreasing on correctly classified samples ($y_i h_t(X_i) > 0$)

Bounds on empirical error

Theorem. *The empirical error of the AdaBoost classifier satisfies*

$$\hat{R}(h) \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right).$$

If for all $t \in [1,T], \gamma \leq (1/2-\epsilon_t)$ *then*

$$\hat{R}(h) \leq \exp\left(-2\gamma^2 T\right).$$

Proof

- Let $g_t(x) = \sum_{s=1}^{t} \alpha_s h_s(x)$.

$$
\begin{aligned}
D_{t+1}(x_i) &= \frac{D_t(x_i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \\
&= \frac{D_{t-1}(x_i)\exp(-\alpha_{t-1} y_i h_{t-1}(x_i))\exp(-\alpha_t y_i h_t(x_i))}{Z_{t-1}Z_t} \\
&= \frac{D_1(x_i)\exp(-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i))}{\prod_{s=1}^{t} Z_s} \\
&= \frac{\exp(-y_i g_t(x_i))}{N\prod_{s=1}^{t} Z_s}, i = 1, \ldots, N \\
\Rightarrow \quad & \exp(-y_i g_t(x_i)) = N \prod_{s=1}^{t} Z_s D_{t+1}(x_i)
\end{aligned}
$$

Proof cont.

- Note that $I_{u \leq 0} \leq e^{-u}$.

$$
\begin{aligned}
\hat{R}(h) &= \frac{1}{N} \sum_{i=1}^{N} I_{y_i g_T(x_i) \leq 0} \\
&\leq \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i g_T(x_i)) \\
&= \frac{1}{N} \sum_{i=1}^{N} \left[N \prod_{t=1}^{T} Z_t \right] D_{T+1}(x_i) \\
&= \prod_{t=1}^{T} Z_t \sum_{i=1}^{N} D_{T+1}(x_i) \\
&= \prod_{t=1}^{T} Z_t
\end{aligned}
$$

Proof cont.

- Consider normalizing factor Z_t.

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{N} D_t(x_i) \exp(-\alpha_t y_i h_t(x_i)) \\
&= \sum_{i:y_i h_t(x_i)=+1} D_t(x_i) \exp(-\alpha_t) + \sum_{i:y_i h_t(x_i)=-1} D_t(x_i) \exp(\alpha_t) \\
&= (1-\epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) \\
&= (1-\epsilon_t) \exp\left(-\frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t}\right) + \epsilon_t \exp\left(\frac{1}{2} \log \frac{(1-\epsilon_t)}{\epsilon_t}\right) \\
&= (1-\epsilon_t) \sqrt{\frac{\epsilon_t}{(1-\epsilon_t)}} + \epsilon_t \sqrt{\frac{(1-\epsilon_t)}{\epsilon_t}} \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}
$$

Proof cont.

- Thus

$$\begin{aligned}
\prod_{t=1}^{T} Z_t &= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&= \prod_{t=1}^{T} \sqrt{4\epsilon_t(1-\epsilon_t)} \\
&= \prod_{t=1}^{T} \sqrt{1-4(1/2-\epsilon_t)^2}
\end{aligned}$$

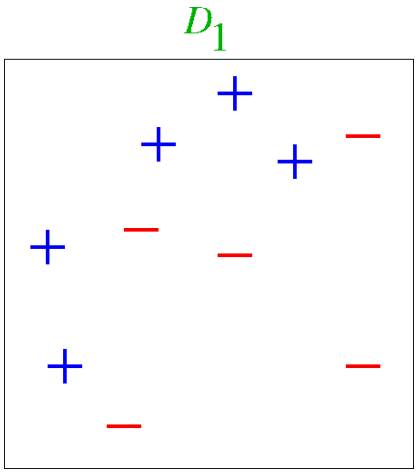
Proof cont.

- Hence

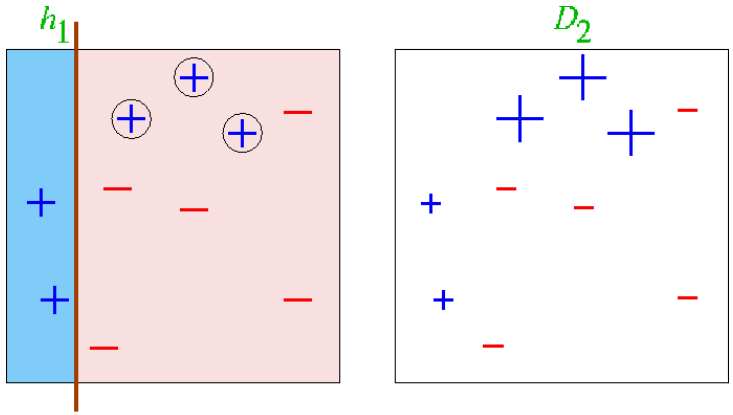
$$\begin{aligned}
\prod_{t=1}^{T} Z_t &= \prod_{t=1}^{T} \sqrt{1 - 4(1/2 - \epsilon_t)^2} \\
&\leq \prod_{t=1}^{T} \left(\exp\left(-4(1/2 - \epsilon_t)^2\right)\right)^{1/2} \\
&= \prod_{t=1}^{T} \exp\left(-2(1/2 - \epsilon_t)^2\right) \\
&\leq \exp(-2\gamma^2 T).
\end{aligned}$$

- Note that α minimizing $g(\alpha) = (1 - \epsilon_t)\exp(-\alpha) + \epsilon_t \exp(\alpha)$ is $\alpha_t = 1/2 \log \frac{(1-\epsilon_t)}{\epsilon_t}$

Toy example



Toy example: First step



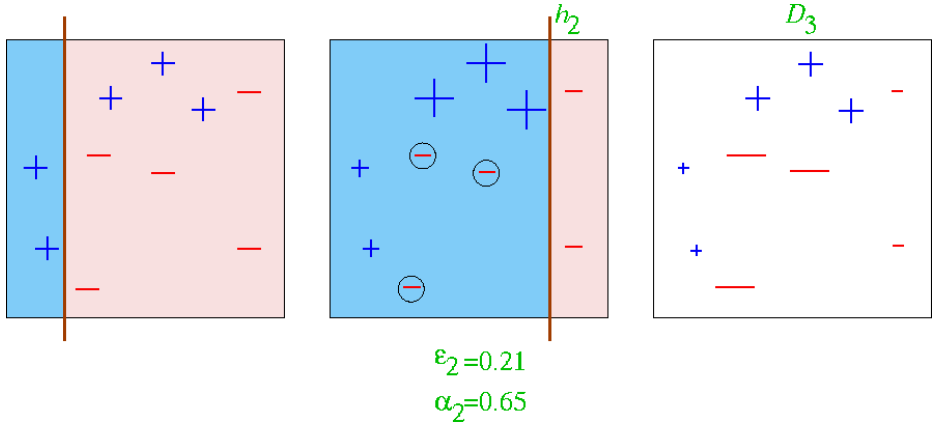
$$D_1(x_i) = \frac{1}{10}, i = 1, \ldots, 10$$

Choose h^* minimizing (1)

$$\Rightarrow \epsilon_1 = \frac{\sum_{i=1}^{10} \frac{1}{10} I_{(h_1(x_i) \neq y_i)}}{\sum_{i=1}^{10} \frac{1}{10}} = 0.3$$

$$\alpha_1 = \frac{1}{2} \log \frac{1-0.3}{0.3} = 0.42$$

Toy example: Second step



$$D_2'(x_i) = \frac{1}{10}\exp(0.42) = 0.15 \quad \text{(3 increased weights)}$$

$$D_2'(x_j) = \frac{1}{10}\exp(-0.42) = 0.066 \quad \text{(7 decreased weights)}$$

$$Z_2 = 3 * 0.15 + 7 * 0.066 = 0.912$$

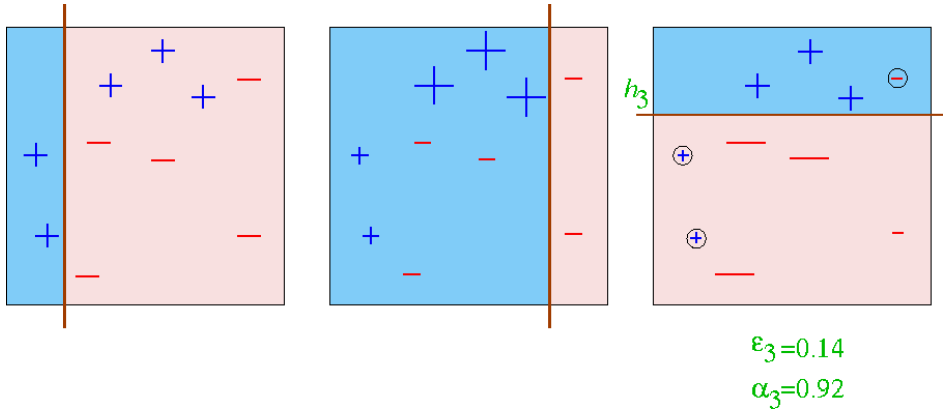
$$D_2(x_i) = D_2'(x_i)/Z_2 = 0.16, D_2(x_j) = D_2'(x_j)/Z_2 = 0.072 \quad \text{(normalized weights)}$$

Choose h^* minimizing (1)

$$\Rightarrow \epsilon_2 = \frac{\sum_{i=1}^{10} D_2(x_i) I_{(h_2(x_i) \neq y_i)}}{1.0} = 3 \cdot 0.072 = 0.21$$

$$\alpha_2 = \frac{1}{2}\log\frac{1-0.21}{0.21} = 0.65$$

Toy example: Third step



$$D_3'(x_i) = 0.072 \exp(0.65) = 0.14 \quad \text{(3 increased small weights)}$$

$$D_3'(x_j) = 0.072 \exp(-0.65) = 0.038 \quad \text{(4 decreased small weights)}$$

$$D_3'(x_k) = 0.17 \exp(-0.65) = 0.089 \quad \text{(3 decreased large weights)}$$

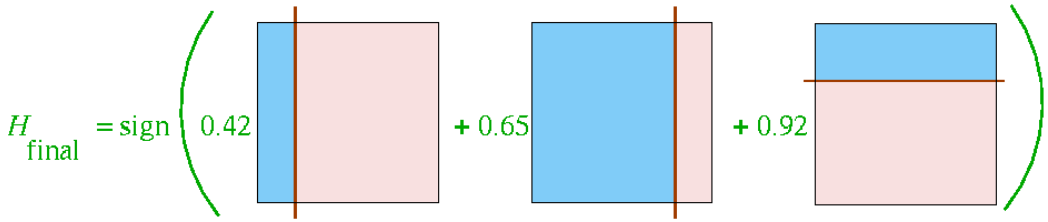
$$D_3(x_i) = 0.17, D_3(x_j) = 0.045, D_3(x_k) = 0.11 \quad \text{(normalized weights)}$$

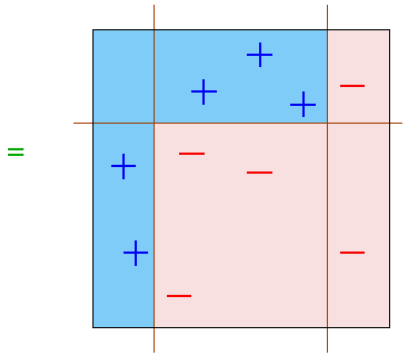
Choose h^* minimizing (1)

$$\Rightarrow \epsilon_3 = \frac{\sum_{i=1}^{10} D_3(x_i) I_{(h_3(x_i) \neq y_i)}}{1.0} = 3 \cdot 0.045 = 0.14$$

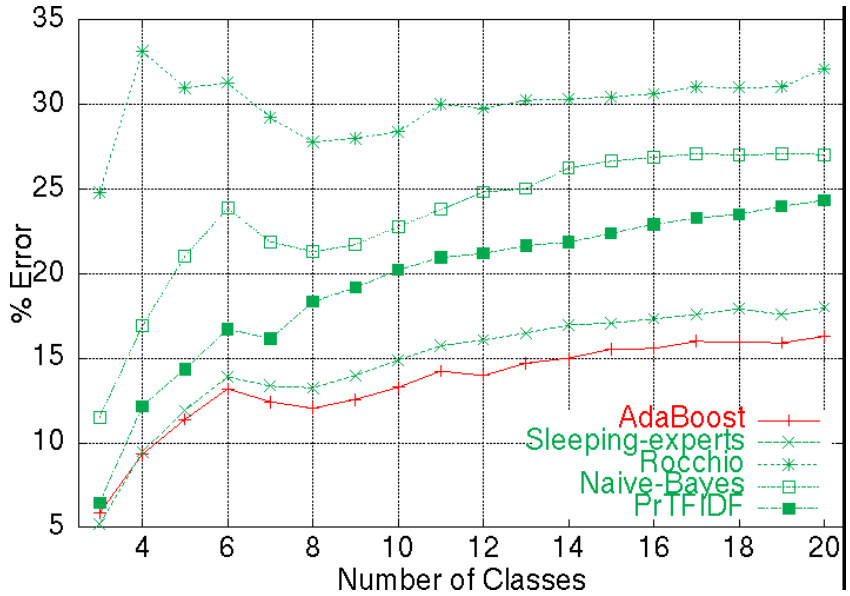
$$\alpha_3 = \frac{1}{2} \log \frac{1-0.14}{0.14} = 0.92$$

Toy example: Final hypothesis

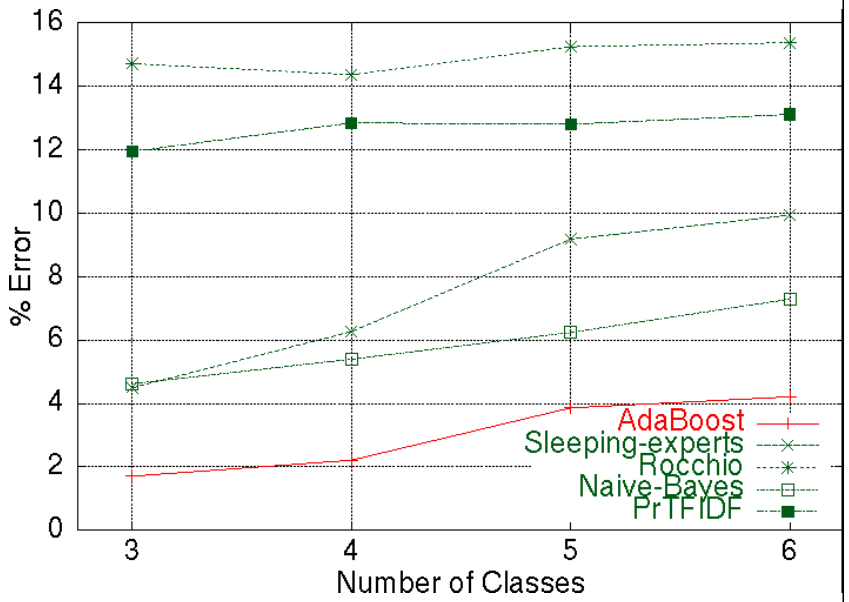




Real data set: Text Categorization

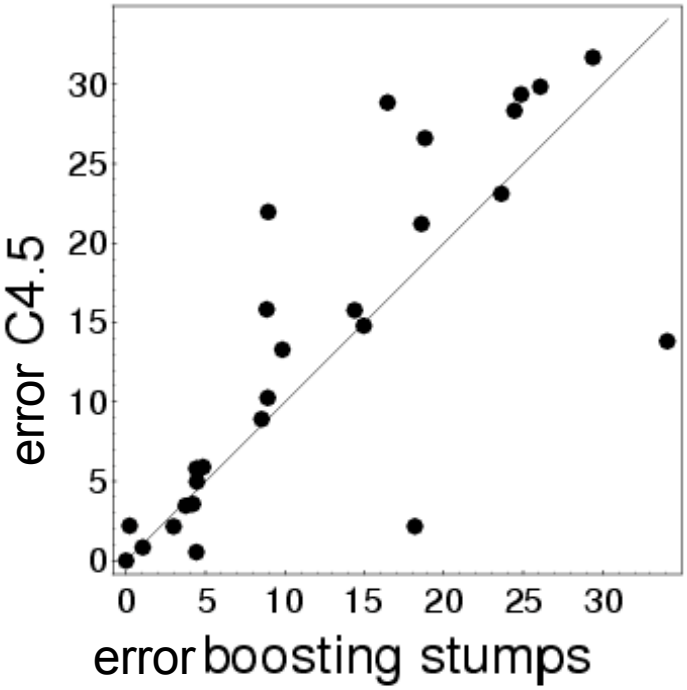


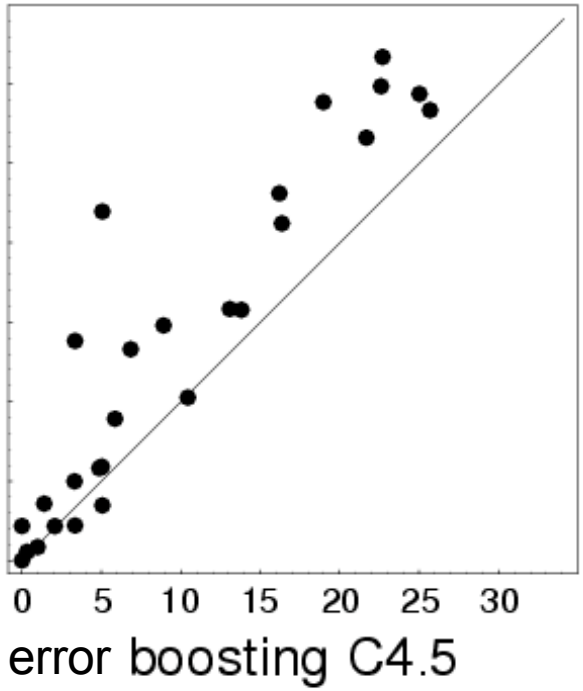
database: AP



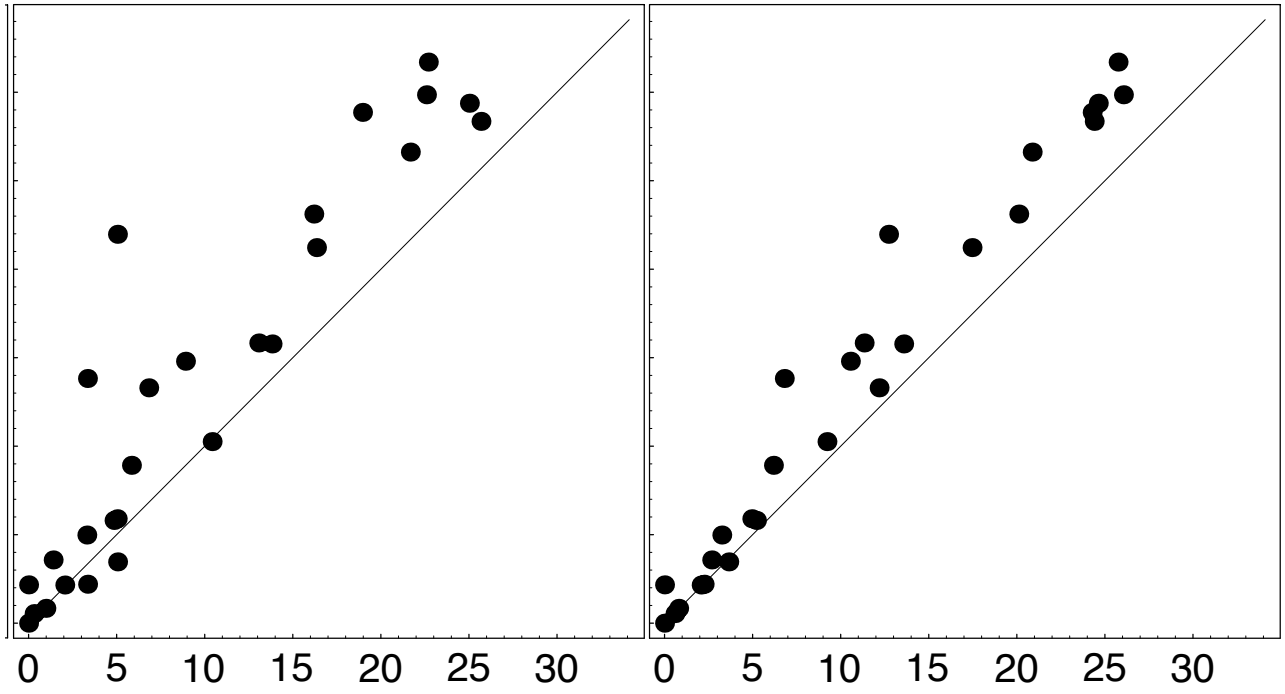
database: Reuters

Boosting empirical evaluation





Bagging vs. Boosting



boosting C4.5 bagging C4.5

Parallel of bagging and boosting

- Bagging is typically faster, but may get a smaller error reduction (not by much)
- Bagging works well with "reasonable" classifiers
- Boosting works with very simple classifiers

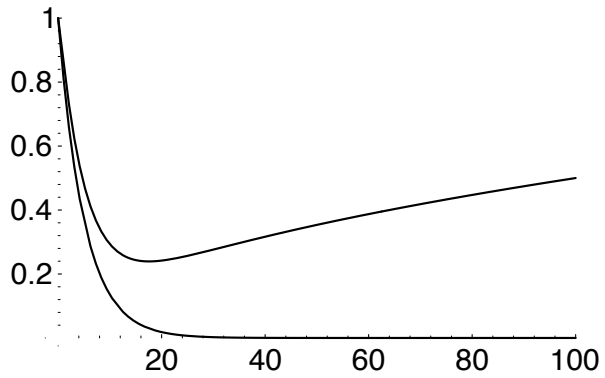
  E.g., Boostexter - text classification using decision stumps based on single words
- Boosting may have a problem if a lot of the data is mislabeled, because it will focus on those examples a lot, leading to overfitting.

Why does boosting work?

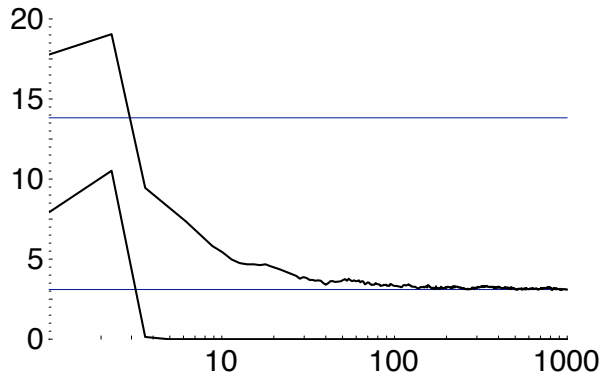
- Weak learners have high bias
- By combining them, we get more expressive classifiers
- Hence, boosting is a *bias-reduction technique*
- What happens as we run boosting longer?
  Intuitively, we get more and more complex hypotheses
- How would you expect bias and variance to evolve over time?

A naive (but reasonable) analysis of generalization error

- Expect the training error to continue to drop (until it reaches 0)
- Expect the test error to *increase* as we get more voters, and h_f becomes too complex.



Actual typical run of AdaBoost



- Test error *does not increase* even after 1000 runs! (more than 2 million decision nodes!)
- Test error *continues to drop* even after training error reaches 0!
- These are consistent results through many sets of experiments!

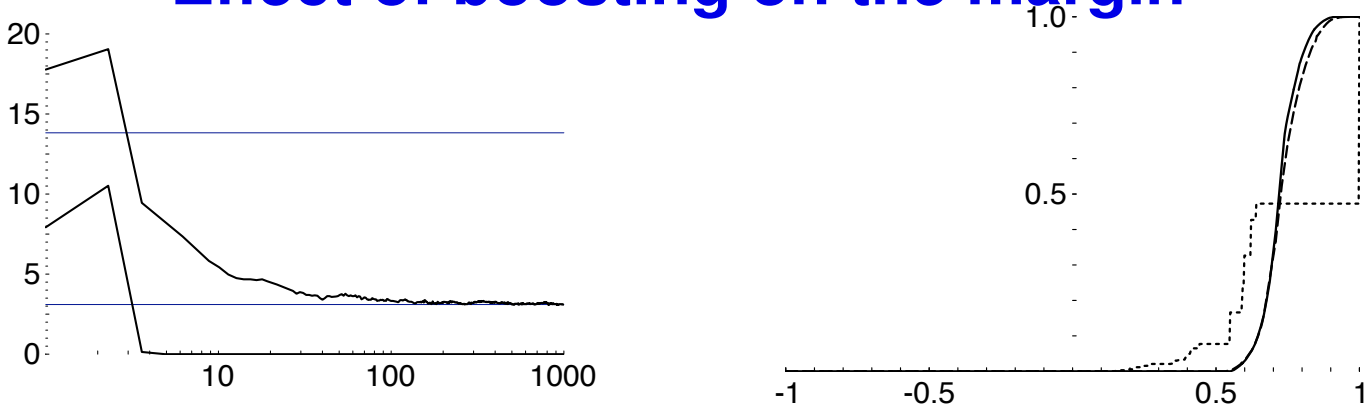
Classification margin

- Boosting constructs hypotheses of the form $h_f(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$
- The classification of an example is correct if $\text{sign}(f(\mathbf{x})) = y$
- The *margin* of a training example is defined as:

$$\text{margin}(f(\mathbf{x}), y) = y \cdot f(\mathbf{x})$$

- The margin tells us how close the decision boundary is to the data point
- The *minimum margin* over the data set gives an idea of how close the training points are to the decision boundary
- A higher margin on the training set should yield a lower generalization error
- Intuitively, increasing the margin is similar to lowering the variance

Effect of boosting on the margin



- Between rounds 5 and 10 there is no training error reduction
- But there is a *significant shift in margin distribution*!
- There is a formal proof that boosting increases the margin
- Next time: classifiers that explicitly aim to construct a large margin.

Summary

- Ensemble methods combine several hypotheses into one prediction
- They work better than the best individual hypothesis from the same class because they reduce bias or variance (or both)
- Bagging is mainly a variance-reduction technique, useful for complex hypotheses
- Main idea is to sample the data repeatedly, train several classifiers and average their predictions.
- Boosting focuses on harder examples, and gives a weighted vote to the hypotheses.
- Boosting works by reducing bias and increasing classification margin.

Perceptrons

- Consider a binary classification problem with data $\{\mathbf{x}_i, y_i\}_{i=1}^m$, $y_i \in \{-1, +1\}$.
- A *perceptron* is a classifier of the form:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + w_0) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

  Here, $\mathbf{w}$ is a vector of weights, "$\cdot$" denotes the dot product, and w_0 is a constant offset.

- The decision boundary is $\mathbf{w} \cdot \mathbf{x} + w_0 = 0$.
- Perceptrons output a class, not a probability
- An example $\langle \mathbf{x}, y \rangle$ is classified correctly if and only if:

$$y(\mathbf{w} \cdot \mathbf{x} + w_0) > 0$$

A gradient descent-like learning rule

- Consider the following procedure:
  1. Initialize $\mathbf{w}$ and w_0 randomly
  2. While any training examples remain incorrectly classified
     (a) Loop through all misclassified examples
     (b) For misclassified example i, perform the updates:

$$\mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \qquad w_0 \leftarrow w_0 + \gamma y_i$$

  where γ is a step-size parameter.

- The update equation, or sometimes the whole procedure, is called the *perceptron learning rule*.
- Intuition: For positive examples misclassified as negative, increase $\mathbf{w} \cdot \mathbf{x}_i + w_0$, and vice versa

Gradient descent interpretation

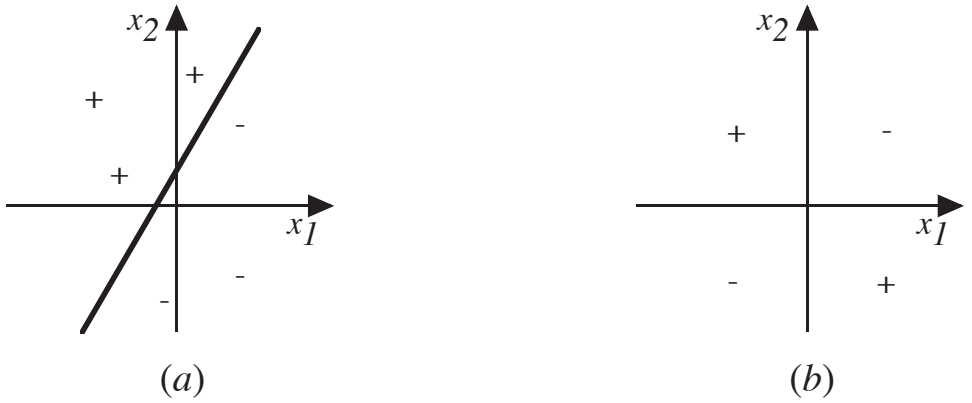
- The perceptron learning rule can be interpreted as a gradient descent procedure, but with the following *perceptron criterion function*

$$J(\mathbf{w}, w_0) = \sum_{i=1}^{m} \begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 0 \\ -y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) < 0 \end{cases}$$

- For correctly classified examples, the error is zero.
- For incorrectly classified examples, the error is by how much $\mathbf{w} \cdot \mathbf{x}_i + w_0$ is on the wrong side of the decision boundary.
- J is piecewise linear, so it has a gradient almost everywhere; stochastic gradient descent gives the perceptron learning rule.
- J is zero if and only if all examples are classified correctly – just like the 0-1 loss function.

Linear separability

- The data set is *linearly separable* if and only if there exists $\mathbf{w}$, w_0 such that:
  - For all i, $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) > 0$.
  - Or equivalently, the 0-1 loss is zero for some set of parameters $(\mathbf{w}, w_0)$.



Perceptron convergence theorem

- *Let subsets of training vectors $\mathcal{C}_1$ and $\mathcal{C}_2$ be linearly separable, and let training vectors be non-trivial and of finite length. Then the perceptron converges after at most*

$$i_0 = \left\lfloor \frac{\max_i ||\mathbf{x}_i||^2 ||\hat{\mathbf{w}}||^2}{\min_i [\mathbf{x}_i \cdot \hat{\mathbf{w}}]^2} \right\rfloor$$

  iterations, where $\hat{\mathbf{w}}$ is a solution vector.
- The *perceptron convergence theorem* states that if the perceptron learning rule is applied to a linearly separable data set, a solution will be found after some finite number of updates.
- The number of updates depends on the data set, and also on the step size parameter.
- If the data is not linearly separable, there will be oscillation (which can be detected automatically).

Proof of perceptron convergence theorem

- It is more convenient to work with $(n+1)$-dimensional augmented input vector

$$\mathbf{x}_i = [+1, x_{1,i}, x_{2,i}, \ldots, x_{n,i}]^T$$

and with the augmented weight vector

$$\mathbf{w}_i = [w_{0,i}, w_{1,i}, w_{2,i}, \ldots, w_{n,i}]^T.$$

- Here the bias w_0 is treated as a synaptic weight driven by a fixed input $x_0 = 1$.
- Then the linear combiner output (local field) is simply

$$v_i = \mathbf{w}_i \cdot \mathbf{x}_i = \sum_{k=0}^{n} w_{k,i} x_{k,i}$$

- The perceptron functions properly if the two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ are *linearly separable* by a hyperplane.
- To simplify the analysis multiply samples from $\mathcal{C}_2$ by -1 and combine it with $\mathcal{C}_1$ yielding set $\mathcal{C}$.
- The perceptron learning algorithm adjusts the weight vector $\mathbf{w}$ until a separating hyperplane is found.
- If the i-th training vector $\mathbf{x}_i$ is correctly classified by the weight vector $\mathbf{w}_i$ computed at the i-th iteration, no correction is made:

$$\mathbf{w}_{i+1} = \mathbf{w}_i, \qquad \text{if } \mathbf{w}_i \cdot \mathbf{x}_i > 0 \text{ and } \mathbf{x}_i \in \mathcal{C}$$

  otherwise the weight vector $\mathbf{w}_i$ is updated

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \gamma_i \mathbf{x}_i \qquad \text{if } \mathbf{w}_i \cdot \mathbf{x}_i \leq 0 \text{ and } \mathbf{x}_i \in \mathcal{C}$$

- After convergence, the weight vector $\hat{\mathbf{w}}$ satisfies the conditions

$$\hat{\mathbf{w}} \cdot \mathbf{x} > 0 \text{ for every input vector } \mathbf{x} \in \mathcal{C}$$

- Initialize $\mathbf{w}$ randomly and assume that $\mathcal{D}$ contains misclassified samples $\mathbf{x}_i, i = 1, \ldots, M$ and $\gamma = 1$.
- Let $\mathbf{x}_i \in \mathcal{D}$, $\hat{\mathbf{w}}$ be any solution vector, so $\hat{\mathbf{w}} \cdot \mathbf{x}_i > 0$, and α be a positive scale factor.

$$\begin{aligned} \mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}} &= (\mathbf{w}_i - \alpha\hat{\mathbf{w}}) + \mathbf{x}_i \\ ||\mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}}||^2 &= ||(\mathbf{w}_i - \alpha\hat{\mathbf{w}})||^2 + 2(\mathbf{w}_i - \alpha\hat{\mathbf{w}}) \cdot \mathbf{x}_i + ||\mathbf{x}_i||^2 \end{aligned}$$

- Because $\mathbf{x}_i$ was misclassified $\mathbf{w}_i \cdot \mathbf{x}_i \leq 0$ then

$$||\mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}}||^2 \leq ||(\mathbf{w}_i - \alpha\hat{\mathbf{w}})||^2 - 2\alpha\hat{\mathbf{w}} \cdot \mathbf{x}_i + ||\mathbf{x}_i||^2$$

- As $\hat{\mathbf{w}} \cdot \mathbf{x}_i$ is strictly positive the second term will dominate the third if α is sufficiently large.
- Let β be the maximum length of a pattern vector

$$\beta^2 = \max_i ||\mathbf{x}_i||^2$$

and let γ be the smallest inner product of the solution vector with any pattern vector

$$\gamma = \min_i [\hat{\mathbf{w}} \cdot \mathbf{x}_i] > 0.$$

- Then

$$||\mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}}||^2 \leq ||(\mathbf{w}_i - \alpha\hat{\mathbf{w}})||^2 - 2\alpha\gamma + \beta^2$$

- Choosing

$$\alpha = \frac{\beta^2}{\gamma}$$

we obtain

$$||\mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}}||^2 \leq ||(\mathbf{w}_i - \alpha\hat{\mathbf{w}})||^2 - \beta^2.$$

- Thus the squared distance $||(\mathbf{w}_i - \alpha\hat{\mathbf{w}})||^2$ is reduced by at least β^2 at each correction, so after i corrections we obtain

$$||\mathbf{w}_{i+1} - \alpha\hat{\mathbf{w}}||^2 \leq ||(\mathbf{w}_1 - \alpha\hat{\mathbf{w}})||^2 - i\beta^2$$

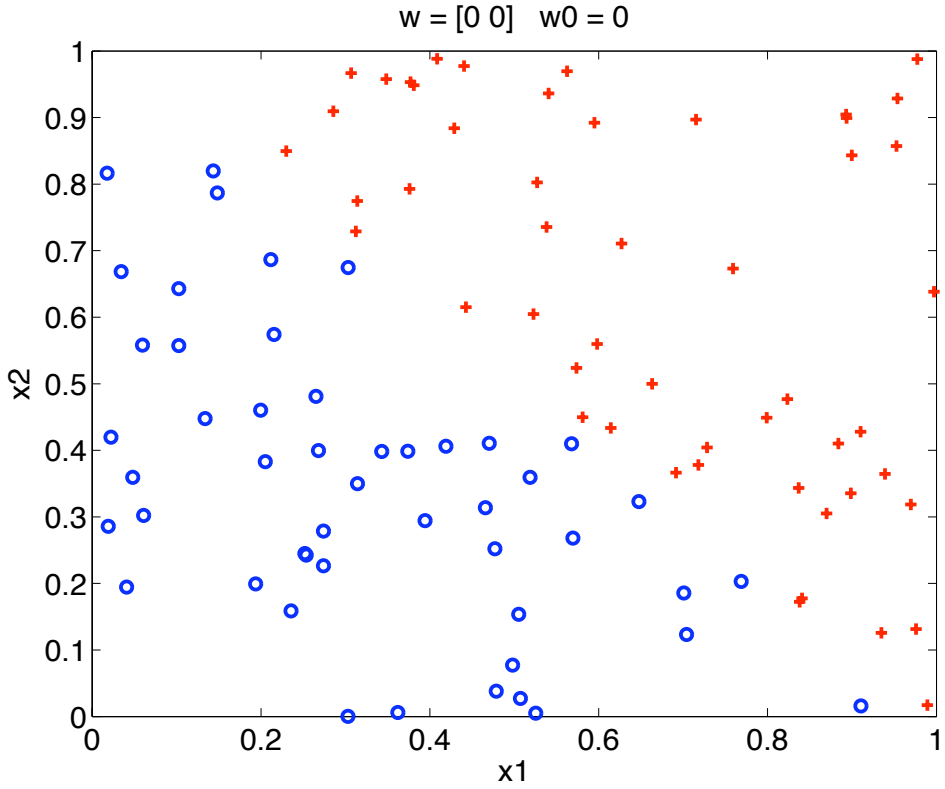
- Because the squared distance cannot become negative the sequence of corrections must terminate after no more than i_0 iterations, where

$$i_0 = \left\lfloor \frac{||(\mathbf{w}_1 - \alpha\hat{\mathbf{w}})||^2}{\beta^2} \right\rfloor$$

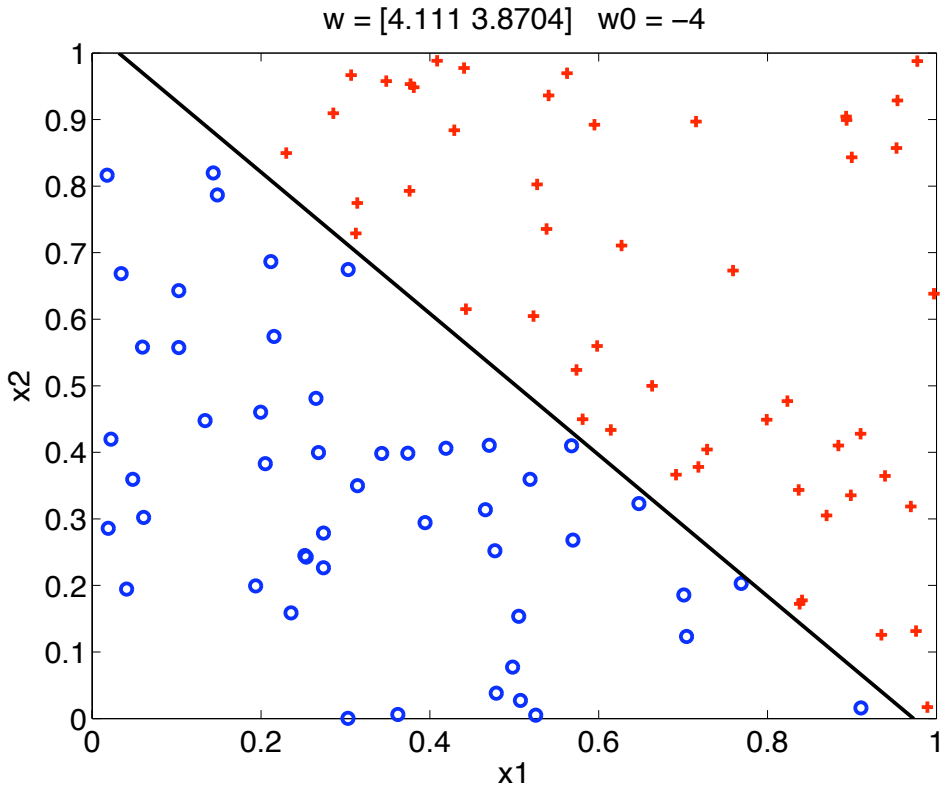
- If $\mathbf{w}_1 = 0$ then

$$i_0 = \left\lfloor \frac{||\alpha\hat{\mathbf{w}}||^2}{\beta^2} \right\rfloor = \left\lfloor \frac{\beta^2||\hat{\mathbf{w}}||^2}{\gamma^2} \right\rfloor = \left\lfloor \frac{\max_i ||\mathbf{x}_i||^2||\hat{\mathbf{w}}||^2}{\min_i[\mathbf{x}_i \cdot \hat{\mathbf{w}}]^2} \right\rfloor.$$

Perceptron learning example–separable data



Perceptron learning example–separable data



Weight as a combination of input vectors

- Recall percepton learning rule:

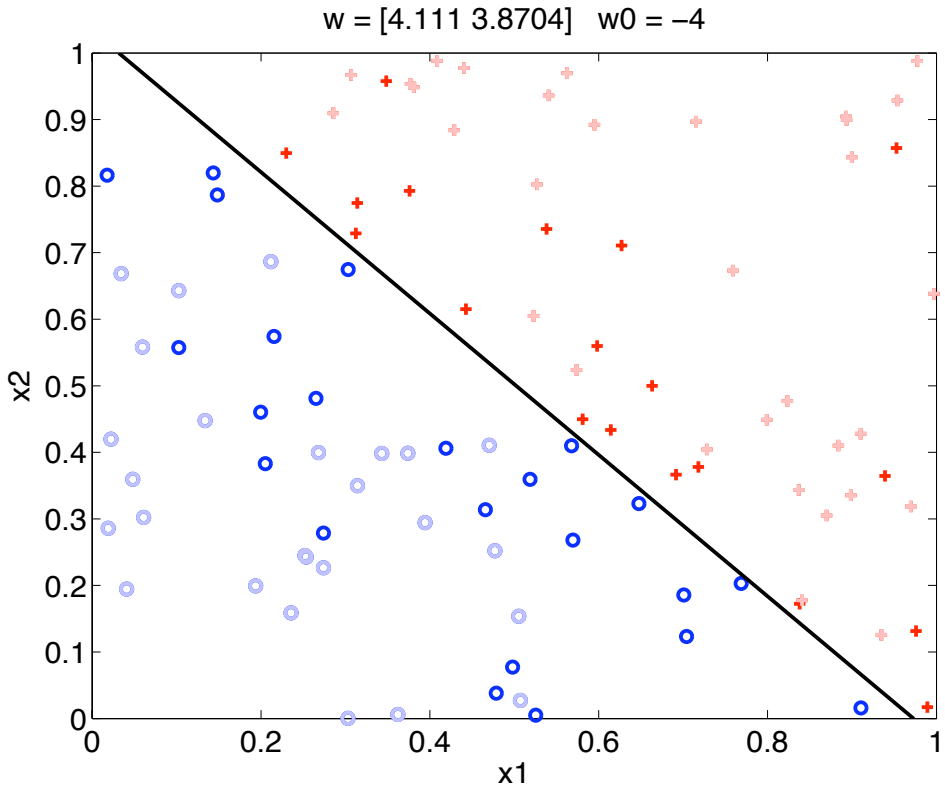
$$\mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \qquad w_0 \leftarrow w_0 + \gamma y_i$$

- If initial weights are zero, then at any step, the *weights are a linear combination of feature vectors of the examples*:

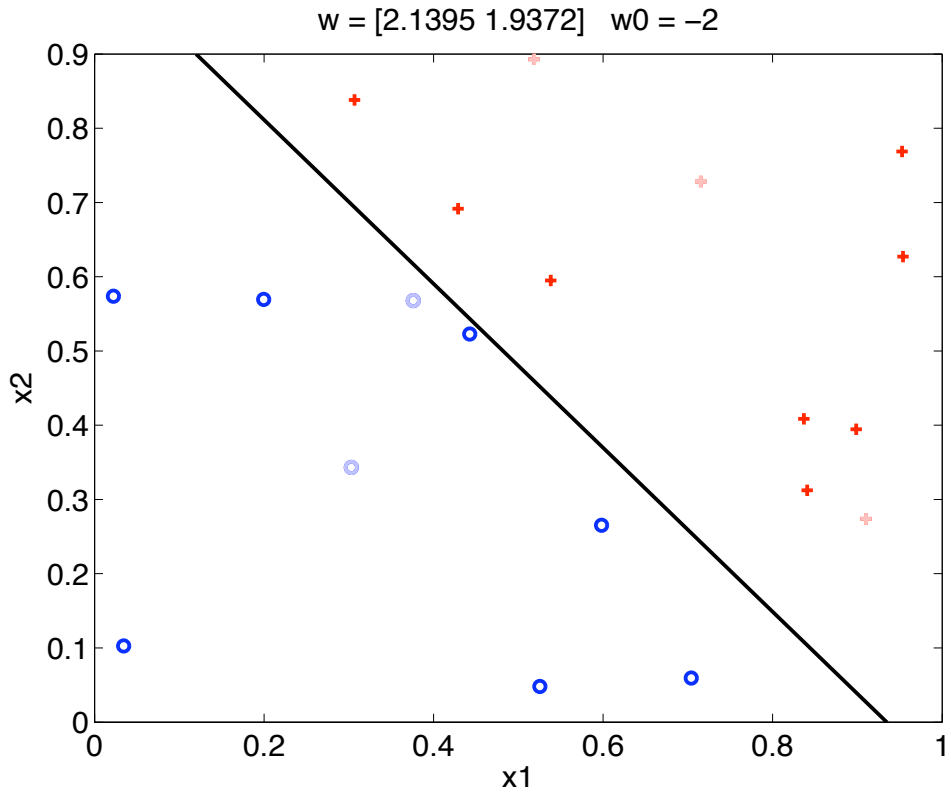
$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \qquad w_0 = \sum_{i=1}^{m} \alpha_i y_i$$

  where α_i is the sum of step sizes used for all updates based on example i.
- This is called the *dual representation* of the classifier.
- Even by the end of training, some example may have never participated in an update, so the corresponding $\alpha_i = 0$.

Example used (bold) and not used (faint) in updates



Comment: Solutions are nonunique



Solutions depend on the set of instances and the order of sampling in updates

Perceptron summary

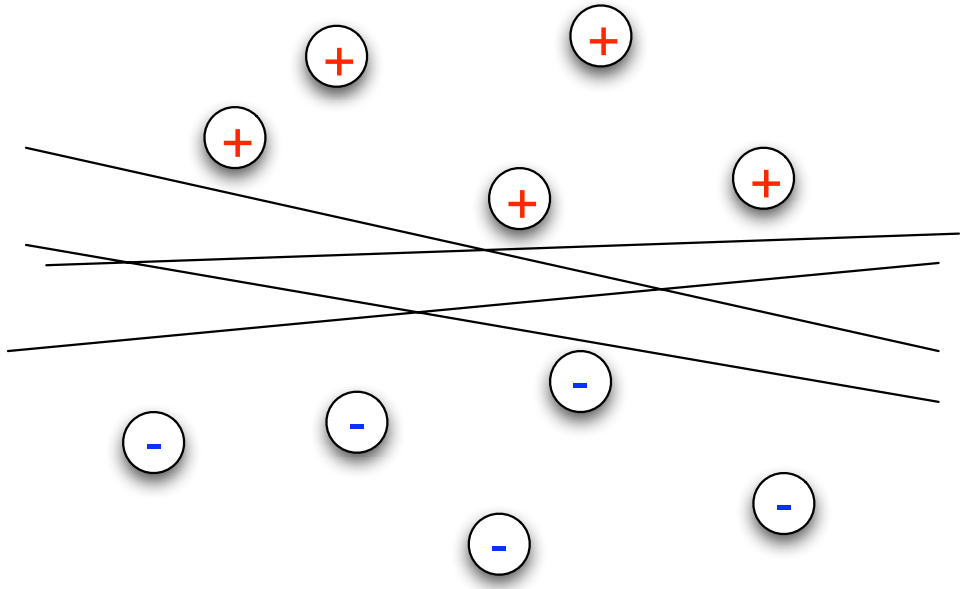
- Perceptrons can be learned to fit linearly separable data, using a gradient descent rule.
- There are other fitting approaches – e.g., formulation as a linear constraint satisfaction problem / linear program.
- Solutions are non-unique.
- Logistic neurons are often thought of as a "smooth" version of a perceptron
- For non-linearly separable data:
  - Perhaps data can be linearly separated in a different feature space?
  - Perhaps we can relax the criterion of separating all the data?

Support Vector Machines

- Support vector machines (SVMs) for binary classification can be viewed as a way of training perceptrons
- There are three main new ideas:
  - An alternative optimization criterion (the "margin"), which eliminates the non-uniqueness of solutions and has theoretical advantages
  - A way of handling nonseparable data by allowing mistakes
  - An efficient way of operating in expanded feature spaces – the "kernel trick"
- SVMs can also be used for multiclass classification and regression.

Returning to the non-uniqueness issue

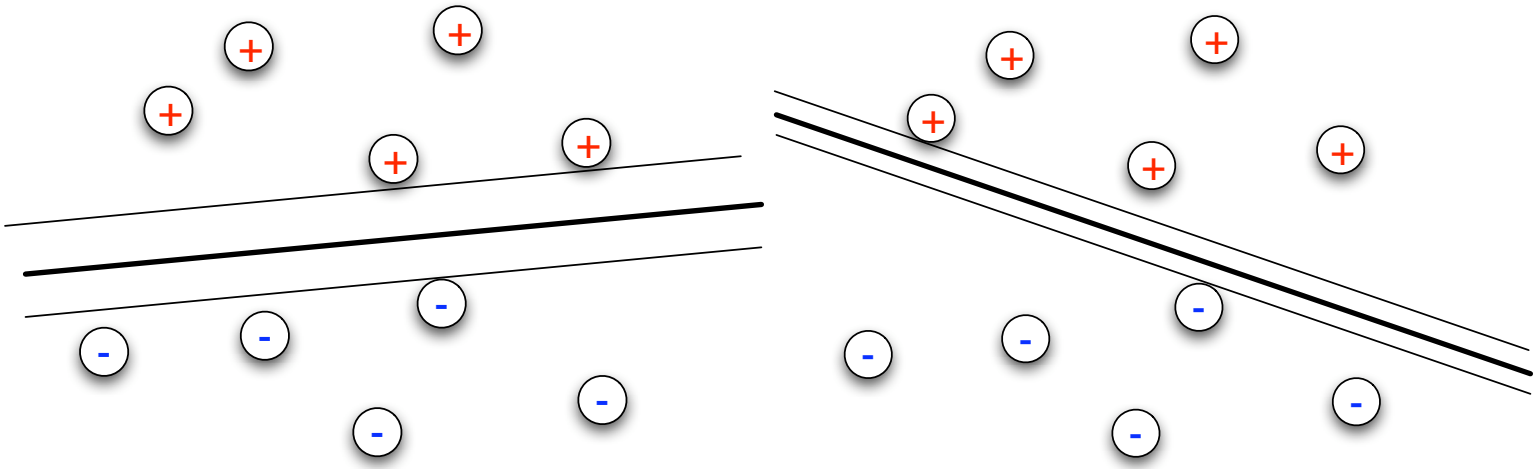
- Consider a linearly separable binary classification data set $\{\mathbf{x}_i, y_i\}_{i=1}^m$.
- There is an infinite number of hyperplanes that separate the classes:



- Which plane is best?
- Relatedly, for a given plane, for which points should we be most confident in the classification?

The margin, and linear SVMs

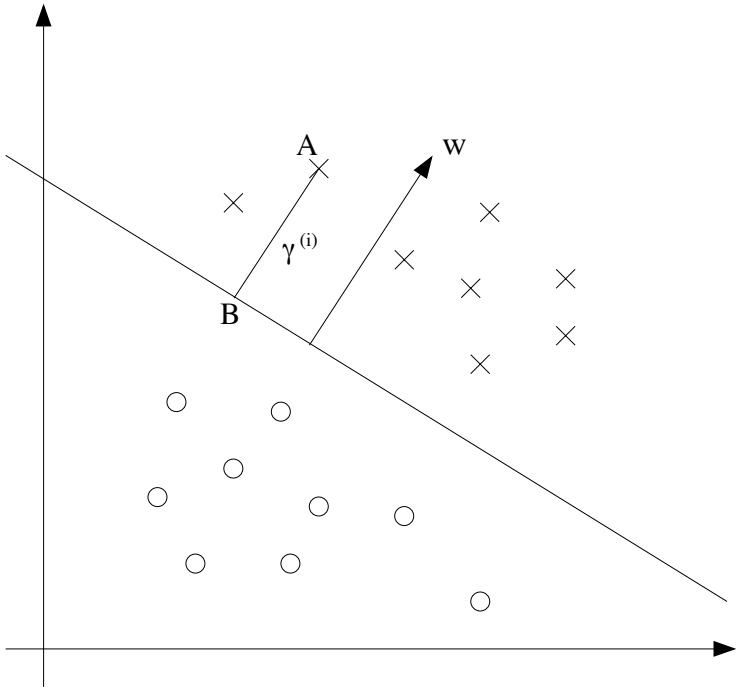
- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.



- It is the width of the "strip" around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose $\mathbf{w}, w_0$ so that margin is maximized

Distance to the decision boundary

- Suppose we have a decision boundary that separates the data.



- Let γ_i be the distance from instance $\mathbf{x}_i$ to the decision boundary.
- How can we write γ_i in term of $\mathbf{x}_i, y_i, \mathbf{w}, w_0$?

Distance to the decision boundary (II)

- The vector $\mathbf{w}$ is normal to the decision boundary. Thus, $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit normal.
- The vector from the B to A is $\gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- B, the point on the decision boundary nearest $\mathbf{x}_i$, is $\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- As B is on the decision boundary,

$$\mathbf{w} \cdot \left(\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||} \right) + w_0 = 0$$

- Solving for γ_i yields, for a positive example:

$$\gamma_i = \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||}$$

The margin

- The *margin of the hyperplane* is $2M$, where $M = \min_i \gamma_i$
- The most direct statement of the problem of finding a maximum margin separating hyperplane is thus

$$\begin{aligned} & \max_{\mathbf{w}, w_0} \min_i \gamma_i \\ \equiv \quad & \max_{\mathbf{w}, w_0} \min_i y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \right) \end{aligned}$$

- This turns out to be inconvenient for optimization, however...

Treating the γ_i as constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x_i} + \frac{w_0}{\|\mathbf{w}\|} \right) \qquad \forall i$$

- This suggests:

$$\begin{array}{rl} \text{maximize} & M \\ \text{with respect to} & \mathbf{w}, w_0 \\ \text{subject to} & y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{w_0}{\|\mathbf{w}\|} \right) \geq M \text{ for all } i \end{array}$$

Treating the γ_i as constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x_i} + \frac{w_0}{\|\mathbf{w}\|} \right) \qquad \forall i$$

- This suggests:

$$\begin{array}{rl} \text{maximize} & M \\ \text{with respect to} & \mathbf{w}, w_0 \\ \text{subject to} & y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{w_0}{\|\mathbf{w}\|} \right) \geq M \text{ for all } i \end{array}$$

- Problems:
  - $\mathbf{w}$ appears nonlinearly in the constraints.
  - This problem is underconstrained. If $(\mathbf{w}, w_0, M)$ is an optimal solution, then so is $(\beta\mathbf{w}, \beta w_0, M)$ for any $\beta > 0$.

Adding a constraint

- Let's try adding the constraint that $\|\mathbf{w}\|M = 1$.
- This allows us to rewrite the objective function and constraints as:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\| \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This is really nice because the constraints are linear.
- The objective function $\|\mathbf{w}\|$ is still a bit awkward.

Final formulation

- Let's maximize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (Taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This we can solve! How?

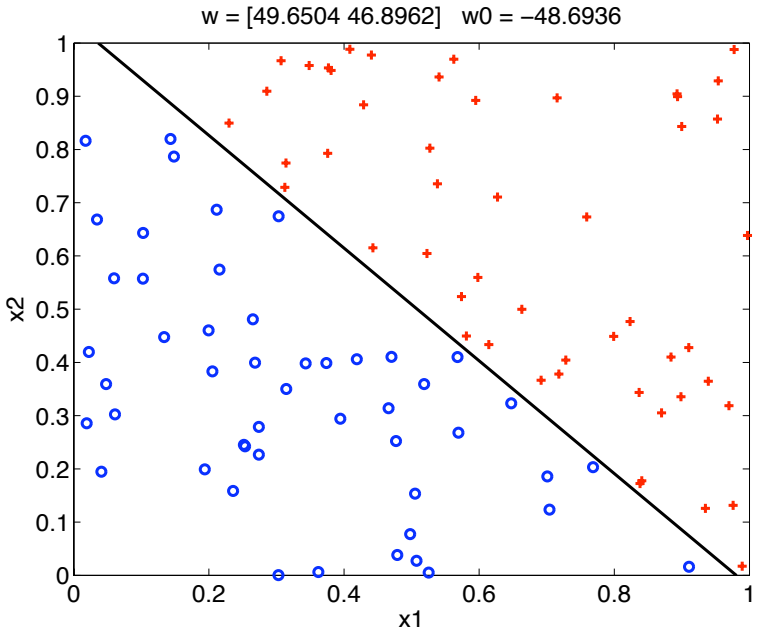
Final formulation

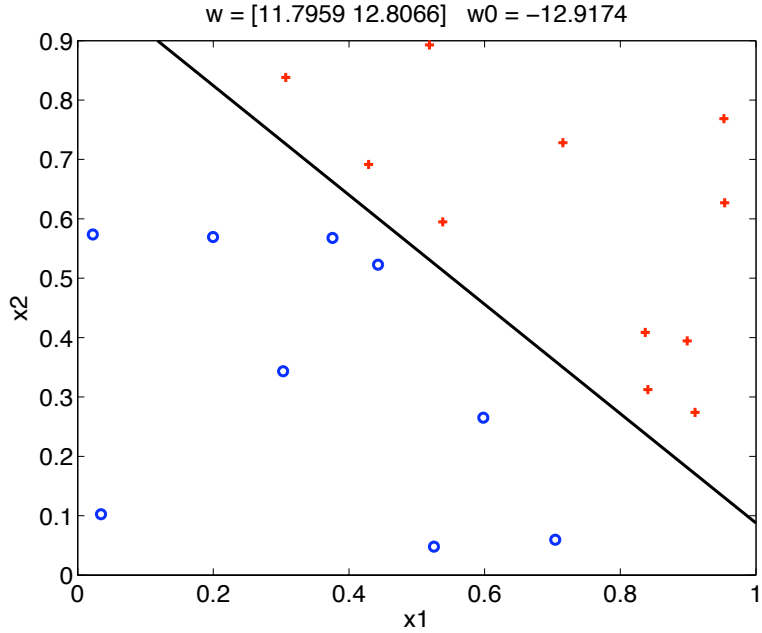
- Let's maximize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (Taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This we can solve! How?
  - It is a *quadratic programming* (QP) problem—a standard type of optimization problem for which many efficient packages are available.
  - Better yet, it's a convex (positive semidefinite) QP

Example





We have a solution, but no support vectors yet...

Lagrange multipliers for inequality constraints (revisited)

- Suppose we have the following optimization problem, called *primal*:

$$\begin{aligned} &\min_{\mathbf{w}} f(\mathbf{w}) \\ &\text{such that } g_i(\mathbf{w}) \quad \leq \quad 0, \;\; i = 1 \ldots k \end{aligned}$$

- We define the *generalized Lagrangian*:

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}), \tag{2}$$

where $\alpha_i, \;\; i = 1 \ldots k$ are the Lagrange multipliers.

A different optimization problem

- Consider $\mathcal{P}(\mathbf{w}) = \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$
- Observe that the following is true. Why?

$$\mathcal{P}(\mathbf{w}) = \begin{cases} f(\mathbf{w}) & \text{if all constraints are satisfied} \\ +\infty & \text{otherwise} \end{cases}$$

- Hence, instead of computing $\min_{\mathbf{w}} f(\mathbf{w})$ subject to the original constraints, we can compute:

$$p^* = \min_{\mathbf{w}} \mathcal{P}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$$

Dual optimization problem

- Let $d^* = \max_{\alpha:\alpha_i \geq 0} \min_{\mathbf{w}} L(\mathbf{w}, \alpha)$ (max and min are reversed)
- We can show that $d^* \leq p^*$.
  - Let $p^* = L(w^p, \alpha^p)$
  - Let $d^* = L(w^d, \alpha^d)$
  - Then $d^* = L(w^d, \alpha^d) \leq L(w^p, \alpha^d) \leq L(w^p, \alpha^p) = p^*$.

Dual optimization problem

- If f, g_i are convex and the g_i can all be satisfied simultaneously for some $\mathbf{w}$, then we have equality: $d^* = p^* = L(\mathbf{w}^*, \alpha^*)$
- Moreover $\mathbf{w}^*, \alpha^*$ solve the primal and dual if and only if they satisfy the following conditions (called Karush-Kunh-Tucker):

$$\frac{\partial}{\partial w_i} L(\mathbf{w}^*, \alpha^*) = 0, \; i = 0 \ldots n \quad (3)$$

$$\alpha_i^* g_i(\mathbf{w}^*) = 0, \; i = 1 \ldots k \quad (4)$$

$$g_i(\mathbf{w}^*) \leq 0, \; i = 1 \ldots k \quad (5)$$

$$\alpha_i^* \geq 0, \; i = 1 \ldots k \quad (6)$$

Back to maximum margin perceptron

- We wanted to solve (rewritten slightly):

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \leq 0 \end{aligned}$$

- The Lagrangian is:

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i (1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0))$$

- The primal problem is: $\min_{\mathbf{w}, w_0} \max_{\alpha : \alpha_i \geq 0} L(\mathbf{w}, w_0, \alpha)$
- We will solve the dual problem: $\max_{\alpha : \alpha_i \geq 0} \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \alpha)$
- In this case, the optimal solutions coincide, because we have a quadratic objective and linear constraints (both of which are convex).

Solving the dual

- From KKT (2), the derivatives of $L(\mathbf{w}, w_0, \alpha)$ wrt $\mathbf{w}, w_0$ should be 0
- The condition on the derivative wrt w_0 gives $\sum_i \alpha_i y_i = 0$ (*)
- The condition on the derivative wrt $\mathbf{w}$ gives:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i} \tag{7}$$

$\Rightarrow$ Just like for the perceptron with zero initial weights, the optimal solution for $\mathbf{w}$ is a linear combination of the $\mathbf{x}_i$, and likewise for w_0.

- The output is

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

$\Rightarrow$ Output depends on weighted dot product of input vector with training examples

Solving the dual (II)

- By plugging (7) back into the expression for L and using (*), we get:

$$\begin{aligned} L(\mathbf{w}, w_0, \alpha) &= \tfrac{1}{2}\| \textstyle\sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i}\|^2 + \sum_{i=1}^{n} \alpha_i (1 - y_i (\sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i} \cdot \mathbf{x}_i + w_0)) \\ &= \tfrac{1}{2} \textstyle\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i - \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ &\qquad + \textstyle\sum_{i=1}^{n} \alpha_i y_i w_0 \\ &= \textstyle\sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j). \end{aligned}$$

- Hence we get dual problem

$$\max_{\alpha} \left(\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \right)$$

with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

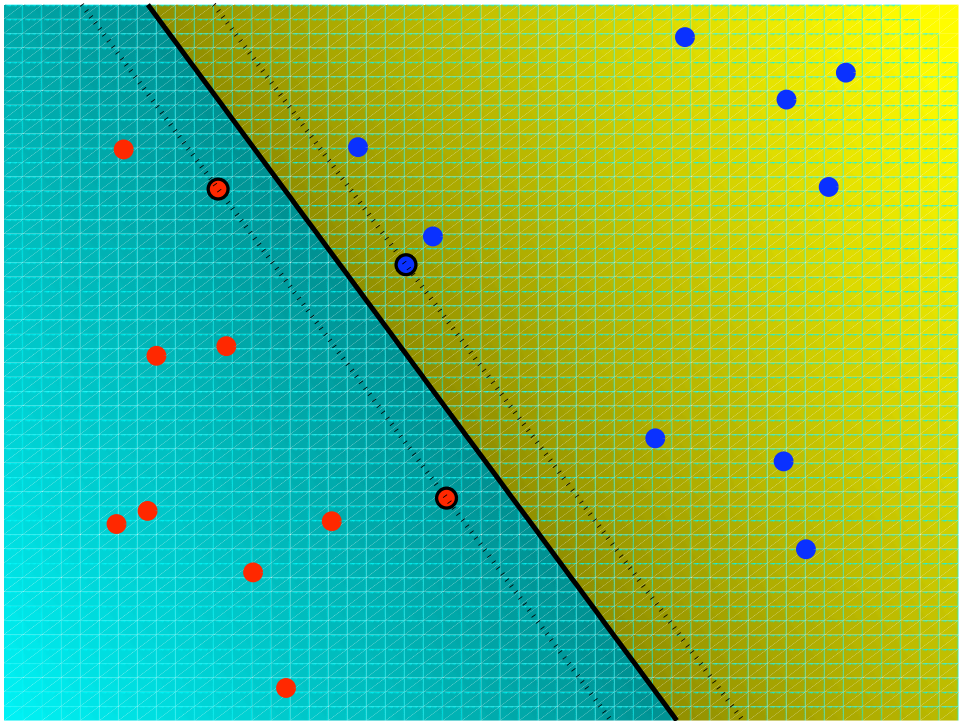
The support vectors

- Suppose we find optimal α's (e.g., using a standard QP package)
- The α_i will be > 0 only for the points for which $1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 0$
- These are the points lying on the edge of the margin, and they are called *support vectors*, because they define the decision boundary
- The output of the classifier for query point $\mathbf{x}$ is computed as:

$$\text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

Hence, the output is determined by computing the *dot product of the point with the support vectors*!

Example



Support vectors are in bold

Soft margin classifiers

- Recall that in the linearly separable case, we compute the solution to the following optimization problem:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- If we want to allow misclassifications, we can relax the constraints to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i$$

- If $\xi_i \in (0, 1)$, the data point is within the margin
- If $\xi_i \geq 1$, then the data point is misclassified
- We define the *soft error* as $\sum_i \xi_i$
- We will have to change the criterion to reflect the soft errors

New problem formulation with soft errors

- Instead of:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

  we want to solve:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{w.r.t.} \quad & \mathbf{w}, w_0, \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i,\ \xi_i \geq 0 \end{aligned}$$

- Note that soft errors include points that are misclassified, as well as points within the margin
- There is a linear penalty for both categories
- The choice of the *constant C controls overfitting*

A built-in overfitting knob

$$
\begin{array}{rl}
\text{min} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\
\text{w.r.t.} & \mathbf{w}, w_0, \xi_i \\
\text{s.t.} & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \xi_i \\
 & \xi_i \geq 0
\end{array}
$$

- If C is 0, there is no penalty for soft errors, so the focus is on maximizing the margin, even if this means more mistakes
- If C is very large, the emphasis on the soft errors will cause decreasing the margin, if this helps to classify more examples correctly.
- Cross-validation is a good way to choose C appropriately

Lagrangian for the new problem

- Like before, we can write a Lagrangian for the problem and then use the dual formulation to find the optimal parameters:

$$\begin{aligned} L(\mathbf{w}, w_0, \alpha, \xi, \mu) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ &+ \sum_i \alpha_i \left(1 - \xi_i - y_i(\mathbf{w}_i \cdot \mathbf{x}_i + w_0)\right) + \sum_i \mu_i \xi_i \end{aligned}$$

- All the previously described machinery can be used to solve this problem
- Note that in addition to α_i we have coefficients μ_i, which ensure that the errors are positive, but do not participate in the decision boundary
- Next time: an even better way of dealing with non-linearly separable data

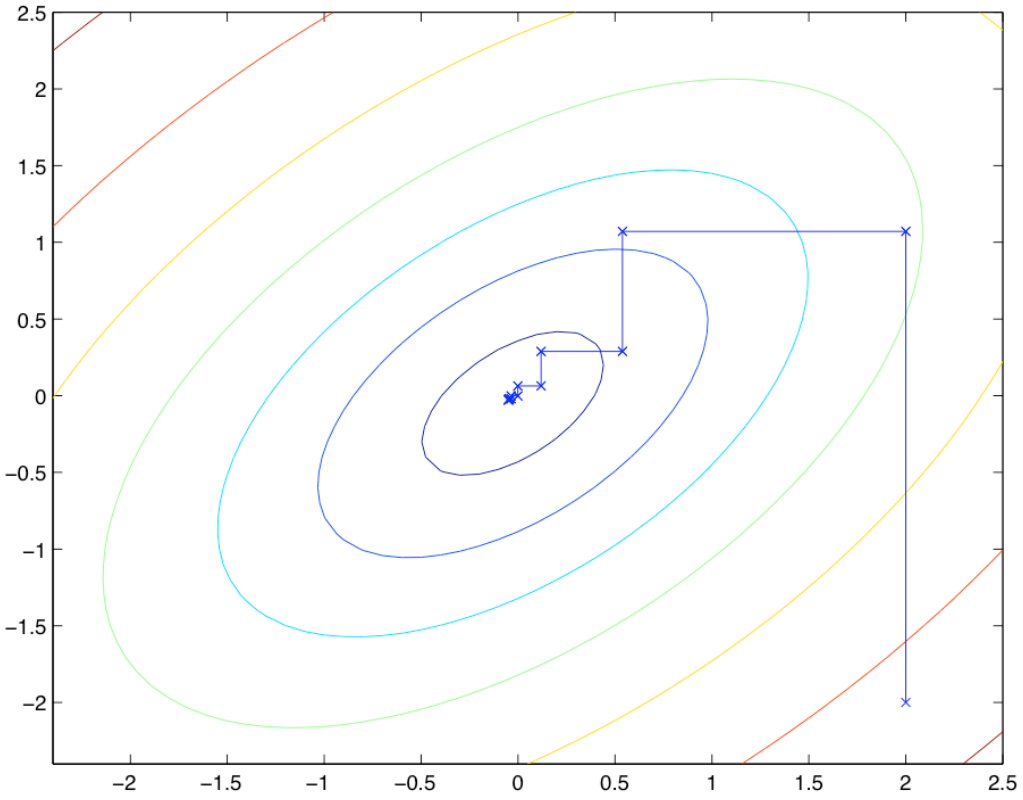
Solving the quadratic optimization problem

- Many approaches exist
- Because we have constraints, gradient descent does not apply directly (the optimum might be outside of the feasible region)
- Platt's algorithm is the fastest current approach, based on *coordinate ascent*

Coordinate ascent

- Suppose you want to find the maximum of some function $F(\alpha_1, \ldots \alpha_n)$
- Coordinate ascent optimizes the function by repeatedly picking an α_i and optimizing it, while all other parameters are fixed
- There are different ways of looping through the parameters:
  - Round-robin
  - Repeatedly pick a parameter at random
  - Choose next the variable expected to make the largest improvement
  - ...

Example



Our optimization problem

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

with constraints: $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0$

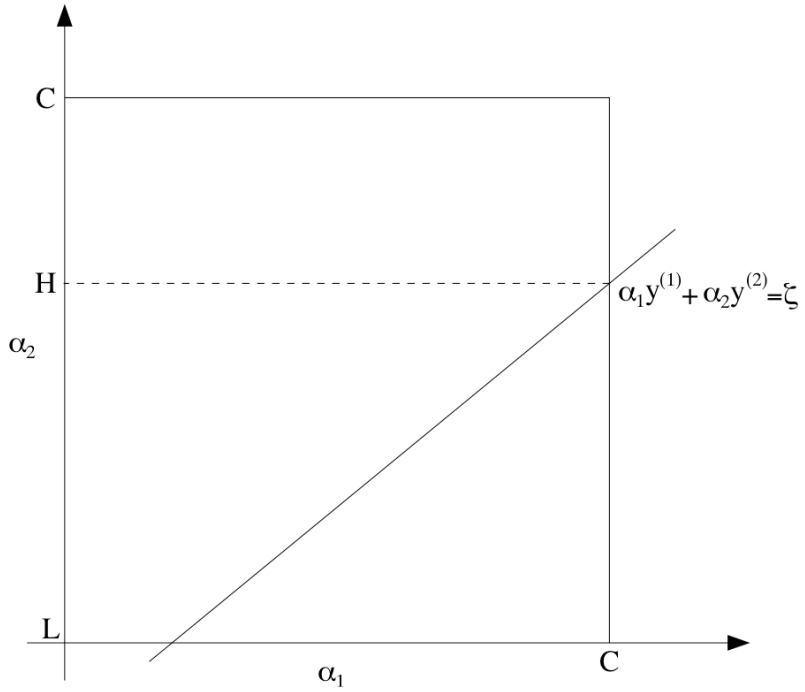
- Suppose we want to optimize for α_1 while $\alpha_2, \ldots \alpha_n$ are fixed
- We cannot do it because α_1 will be completely determined by the last constraint: $\alpha 1 y_1 + \sum_{i=2} \alpha_i y_i = 0$, hence $\alpha_1 = -y_1 \sum_{i=2}^{m} \alpha_i y_i$ (because y_1 is either $+1$ or -1 so $y_1^2 = 1$)
- Instead, we have to optimize *pairs of parameters* α_i, α_j together (Sequential Minimal Optimization (SMO) - Platt, 1999)

SMO

- Suppose that we want to optimize α_1 and α_2 together, while all other parameters are fixed.
- We know that $y_1\alpha_1 + y_2\alpha_2 = -\sum_{i=1}^{m} y_i\alpha_i = \xi$, where ξ is a constant
- So $\alpha_1 = y_1(\xi - y_2\alpha_2)$
- This defines a line, and any pair α_1, α_2 which is a solution has to be on the line

SMO (II)

- We also know that $0 \leq \alpha_1 \leq C$ and $0 \leq \alpha_2 \leq C$, so the solution has to be on the line segment inside the rectangle below



SMO(III)

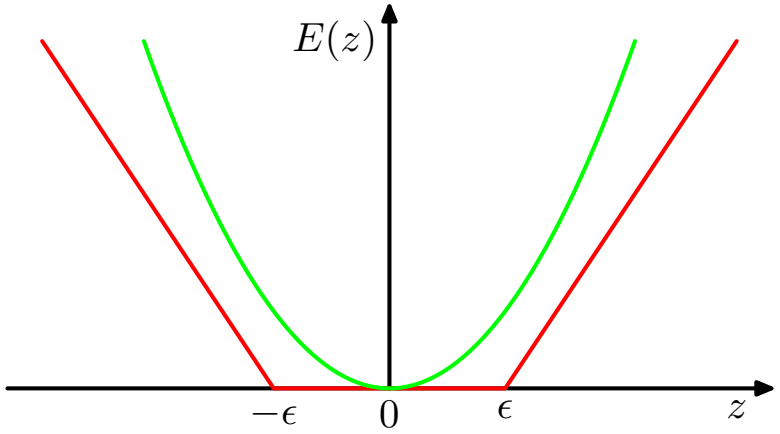
- By plugging α_1 back in the optimization criterion, we obtain a quadratic function of α_2, whose optimum we can find exactly
- If the optimum is inside the rectangle, we take it.
- If not, we pick the closest intersection point of the line and the rectangle
- This procedure is very fast because all these are simple computations.

SVM Regression

- To obtain sparse SVM regression we instead minimizing the regularized error function

$$\frac{1}{2}\sum_{i=1}^{m}(w^T\phi(x_i)+b-y_i)^2+\frac{\lambda}{2}||w||^2$$

we minimize the regularized error function replacing quadratic error term with ϵ-sensitive error function E_ϵ(Vapnik, 1995)



SVM Regression

$$E_\epsilon(x) = \begin{cases} 0 & \text{if} |x| < \epsilon \\ |x| - \epsilon, & \text{otherwise} \end{cases}$$

SVM Regression

- The regularized error function becomes

$$C\sum_{i=1}^{m} E_\epsilon(w^T\phi(x_i) + b - y_i) + \frac{\lambda}{2}||w||^2$$

- One can show using dual Lagrangian formulation that solution of the minimization problem has the form

$$y(x) = \sum_{i=1}^{m}(a_i - \hat{a}_i)K(x, x_i) + b$$

where $k(x, x') = \phi(x)^T\phi(x')$, $0 \leq a_i, \hat{a}_i \leq C$, $b = y_n - \epsilon - w^T\phi(x_n)$, $0 < a_n < C$

SVM Regression

- a_i is non-zero iff the data point (x_i, y_i) lies on or above the upper boundary of the ϵ-tube. $\hat{a}_i$ is non-zero iff the data point (x_i, y_i) lies on or below the lower boundary of the ϵ-tube.