A natural way to train the classifier is to minimize the number of classification errors on the training data, i.e. choosing $w$ so that the training error

$$C(w) = \left| \left\{ (x_i, y_i) \mid y_i \neq \text{sign}(w^T x_i) \right\} \right|$$

is minimized. (We use $|A|$ do denote the cardinality of set $A$.)

However, minimizing $C(w)$ is in the general case computationally intractable (NP-hard).

If it is known that the classes are linearly separable, i.e. that there exists a $w^*$ such that $C(w^*) = 0$, the problem can be solved efficiently.
Perceptron algorithm

- The *perceptron algorithm* is a simple iterative method which can be used to train a linear classifier.

- The algorithm converges if and only if the training data is linearly separable. Of course, the algorithm (or its variations) can be used also for non-separable data.

- Even if the data is linearly separable, the perceptron algorithm is only guaranteed to converge in some finite time. For difficult problems, it may be very slow. A better worst-case behavior is obtained by using *linear programming*.

- The following pseudocode goes through the data repeatedly until a linear classifier with zero training error is found, or until a predefined maximum number of iterations $T$ is reached.
Perceptron algorithm: Pseudocode

\[ w := 0 \]

for round = 1:T
  update := false
  for i = 1:N
    \( \hat{y}_i := \text{sign}(w^T x_i) \)
    if \( \hat{y}_i \neq y_i \)
      \[ w := w + y_i x_i \]
      update := true
    if update == false
      break
  if update == false
    break
return \( w \)
Perceptron algorithm: Main ideas

- The algorithm keeps track of and updates a weight vector $\mathbf{w}$

- Each input item is shown once in a sweep. If a full sweep is completed without any misclassifications then we are done. If $T$ sweeps are reached then we stop (without convergence)

- Whenever $\hat{y}_i \neq y_i$ we update $\mathbf{w}$ by adding $y_i \mathbf{x}_i$. This turns $\mathbf{w}$ towards $\mathbf{x}_i$ if $y_i = +1$, and away from $\mathbf{x}_i$ if $y_i = -1$

- Note on terminology: a full sweep through the data is in this context often called an *epoch*
Perceptron algorithm: Illustration

Current state of $\mathbf{w}$

- training example of class +1
- training example of class −1
Perceptron algorithm: Illustration

Red point classified correctly, no change to $\mathbf{w}$
Perceptron algorithm: Illustration

Green point classified correctly, no change to $\mathbf{w}$
Perceptron algorithm: Illustration

Green point misclassified, will change \( \mathbf{w} \) as follows...
Perceptron algorithm: Illustration

Adding $y_i x_i$ to current weight vector $w$ to obtain new weight vector
Perceptron algorithm: Convergence proof

- Assumption 1:

  The training data is linearly separable with a margin $\gamma > 0$:

  There exists a $\mathbf{w}^* \in \mathbb{R}^n$ for which $||\mathbf{w}^*||_2 = 1$ and $y_i \mathbf{x}_i^T \mathbf{w}^* \geq \gamma$ for all $i = 1, \ldots, N$

  (Note: $||\mathbf{w}||_2 = \sqrt{\mathbf{w}^T \mathbf{w}}$ is the regular Euclidean norm.)
Assumption 2:

The training data fits into a sphere with radius $r$ centered at the origin:

$$\|x_i\|_2 \leq r \text{ for all } i = 1, \ldots, N$$

For a finite number $N$ of points this is of course always satisfied. Let $r$ equal the norm of the datapoint with the largest norm.
Consequence 1:

Each update of $w$ increases the inner product $w^T w^*$ by at least $\gamma$:

Let $w$ denote the weight vector before the update, while $w' = w + y_i x_i$ is the vector after the update. Then we have

$$w'^T w^* = (w + y_i x_i)^T w^* = w^T w^* + y_i x_i^T w^* \geq w^T w^* + \gamma$$

Note that since we started with $w = 0$ at the first iteration, we had $w^T w^* = 0$ at the start, so after $p$ updates we necessarily have $w^T w^* \geq p\gamma$. 
Consequence 2:

Each update of $\mathbf{w}$ increases the squared norm $||\mathbf{w}||_2^2$ of $\mathbf{w}$ by at most $r^2$:

Let $\mathbf{w}$ denote the weight vector before the update, while $\mathbf{w}' = \mathbf{w} + y_i\mathbf{x}_i$ is the vector after the update. Then we have

$$||\mathbf{w}'||_2^2 = (\mathbf{w} + y_i\mathbf{x}_i)^T(\mathbf{w} + y_i\mathbf{x}_i) = \mathbf{w}^T\mathbf{w} + 2y_i\mathbf{x}_i^T\mathbf{w} + y_i^2\mathbf{x}_i^T\mathbf{x}_i,$$

where $y_i\mathbf{x}_i^T\mathbf{w} < 0$ since the example was misclassified, $y_i^2 = 1$, and $\mathbf{x}_i^T\mathbf{x}_i = ||\mathbf{x}_i||_2^2 \leq r^2$ by Assumption 2. Hence $||\mathbf{w}'||_2^2 \leq ||\mathbf{w}||_2^2 + r^2$.

Note that since we started with $\mathbf{w} = \mathbf{0}$ at the first iteration, we had $||\mathbf{w}||_2^2 = 0$ at the start, so after $p$ updates we necessarily have $||\mathbf{w}||_2^2 \leq pr^2$. 
Result:

Thus we obtain

$$1 \geq \frac{\mathbf{w}^T \mathbf{w}^*}{||\mathbf{w}||_2 ||\mathbf{w}^*||_2} \geq \frac{p \gamma}{\sqrt{pr^2}},$$

where $p$ is the number of updates, $||\mathbf{w}^*||_2 = 1$ by Assumption 1, and the first inequality is an instance of the well known inequality

$$\frac{\mathbf{x}^T \mathbf{y}}{||\mathbf{x}||_2 ||\mathbf{y}||_2} \leq 1,$$

for any two vectors $\mathbf{x}$ and $\mathbf{y}$.

Rewriting the inequality we obtain:

$$p \leq \frac{r^2}{\gamma^2}$$
Perceptron algorithm: Non-convergence

- We just proved that if the training data is linearly separable, the perceptron algorithm converges after a finite number of steps.

- However, the finite number can be very large if the margin is not wide.

- If the training data is not linearly separable, the algorithm does not converge, because it simply cannot (by definition) get through a full sweep of the data without performing an update.

- What then?
  - Add some tricks on top of the Perceptron algorithm
  - Use an alternative linear classification algorithm
Other linear classifiers

- Linear Discriminant Analysis (LDA): classic statistical method

- Support Vector Machine (SVM)
  - often used with *kernel function* to allow adding a very large number of new features
  - scaling to extremely large data sets problematic
  - not covered in this class, see Section 5.5 of textbook

- Logistic regression
  - replaces 0-1 loss with a continuous loss, does probabilistic predictions
  - covered a bit later in this class
Perceptron algorithm: dealing with non-convergence

- If the Perceptron algorithm does not converge, how do you pick a final $w$ to use on new data?

- Some methods that are generally useful for improving convergence of iterative algorithms can be useful
  - Use a smaller learning rate, as in $w := w + \lambda y_i x_i$ for misclassified points $(x_i, y_i)$, with $\lambda > 0$ decreasing for each sweep
  - Take a running average of recent weight vectors

- The *Pocket Algorithm* (Gallant 1990) is a simple method to get a final $w$ when iterations don’t converge
The Pocket algorithm

- Run Perceptron algorithm updating $\mathbf{w}$ as usual

- Keep a score for current $\mathbf{w}$ by simply counting how many correct predictions $\mathbf{w}$ has made in a row

- The algorithm has a “pocket” where it stores the weight vector $\mathbf{w}_p$ that has had the highest score thus far, and its score $\text{score}_p$

- Initially $\mathbf{w}_p = 0$ and $\text{score}_p = 0$

- When a full epoch passes without the contents of the pocket changing, the algorithm gives $\mathbf{w}_p$ as its final result and terminates
The Pocket algorithm (2)

Corresponding to each step \(i\) of the Perceptron algorithm

- if the prediction was correct \((\text{sign}(\mathbf{w}^T \mathbf{x}_i) = y_i)\)
  - increase score: \(\text{score} := \text{score} + 1\)

- otherwise (if \((\text{sign}(\mathbf{w}^T \mathbf{x}_i) \neq y_i)\))
  - if the previous high score was beaten, i.e. \(\text{score} > \text{score}_p\), then
    - set \(\mathbf{w}_p := \mathbf{w}\) and \(\text{score}_p := \text{score}\)
  - update \(\mathbf{w} := \mathbf{w} + y_i \mathbf{x}_i\) as usual
  - reset \(\text{score} := 0\).
Logistic regression

- Instead of giving a ‘forced choice’ prediction $\hat{y} = \text{sign}(w^T x)$, we could give probabilistic predictions:

$$P(y = +1 \mid x) = f(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

Here $f$ is the logistic function, and such a classifier is known as logistic regression.

- In what follows, we again use $y \in \{1, 0\}$ for the labels for mathematical simplicity.
The constant-probability surfaces of $P(y = +1 \mid x)$ are hyperplanes, with the .5 hyperplane going through the origin:
In principle, we could use any proper cost function, and minimize the cost on the training set.

In practice, the common choice is to maximize the likelihood, equivalent to minimizing the logarithmic cost. This gives the cost function

\[ C(w) = - \sum_{i=1}^{N} \left[ y_i \log f(w^T x_i) + (1 - y_i) \log f(-w^T x_i) \right] \]

Need to minimize \( C(w) \) with respect to \( w \)

- Recommended: Use existing logistic regression fitting procedures (in Matlab/Octave/R)
- A simple, but not very efficient, alternative: gradient descent (if you want to implement something simple yourself)
Logistic regression: Linearly separable case

- If the training set is linearly separable, the norm of $w$ grows without bound
  - $\Rightarrow$ the cost on the training set approaches zero
  - $\Rightarrow P(y = 1 \mid x') \in \{0, 1\}$ for any new datapoint $x'$

- As in linear regression, we can regularize by adding a term...
  - $\lambda \|w\|_2^2$ (quadratic penalty), or
  - $\lambda \|w\|_1$ (absolute-value penalty, sparser solutions)

...to the cost function.
Logistic regression: Nonlinear features

Again, by first computing features $z = f(x)$ where $f$ is a nonlinear function, and applying the logistic regression to $z$, we can obtain nonlinear structure.
Unsupervised learning
Supervised vs unsupervised learning

- In supervised learning, we are trying to learn a mapping from $\mathcal{X}$ to $\mathcal{Y}$. The training data contains pairs $(x_i, y_i)$, i.e. it includes the target outputs, hence the term \textit{supervised}.

- In \textit{unsupervised} learning, we are only given some data points $x_i$, and the goal is to perform some useful analysis on that data.
  
  - Of course, what is \textit{useful} varies from problem to problem...
  
  - ...so there are many different types of unsupervised learning. (see the examples on the next slides)
Unsupervised learning: Example 1

Clustering search results (http://search.carrot2.org/stable/search):
Unsupervised learning: Example 2

Learning the structure of natural language:

- **Unigrams** (‘bag of words’ model):
  \[
P(\langle \text{word}_1, \ldots, \text{word}_N \rangle) = P(\text{word}_1) \cdots P(\text{word}_N)
\]

- **Bigrams**:
  \[
P(\langle \text{word}_1, \ldots, \text{word}_N \rangle) = P(\text{word}_1) P(\text{word}_2 | \text{word}_1) \cdots P(\text{word}_N | \text{word}_{N-1})
\]

- **n-grams**: Condition on the previous \(n - 1\) words. Lots and lots of n-gram data available from Google!
Example: Text produced with trigram model

Nonsense, Matriona will feed it.
It is always either vodka or brandy.
Yet I am sorry to leave.
You should not handle youthful egoism so roughly, sister.
What did I hurt my poor boy?
No, indeed, are ambition; for the first day.
Yes, they are singing across the water.
It is like a beggar beneath your window.

(see http://www.shiffman.net/teaching/a2z/generate/)

Such a model may be quite helpful in statistical machine translation (because you want to produce good grammar in addition to a decent translation)
Unsupervised learning: Example 3

Process monitoring:

▶ Continuous-valued measurements of a factory process:

▶ We would like to automate the ‘staring at the curves’ job to detect fault conditions

▶ Standard classification problem? \( \mathcal{X} \) is the measurements, \( \mathcal{Y} = \{\text{normal, fault}\} \)
Problem: Lots of measurements from ‘normal’ operating conditions, very few measurements from ‘fault’ conditions. No guarantee that new faults will ‘look like’ old ones...

Solution? Create some form of model for what the ‘normal’ data looks like, and then when monitoring sound the alarm when a new vector is “too far” from any of the normal states (this is anomaly detection)
Unsupervised learning: Example 4

Association rule mining:

- In NBA (U.S. basketball league) very detailed statistics are kept of players’ performance
- This results in a large database that can be searched for dependencies that otherwise may be missed
- The ‘Advanced Scout’ system\(^1\) looks for rules such as ‘when player X is on the field, the accuracy of player Y drops from 75% to 30%’
- Note: Beware of multiple testing effects (these have to be taken into account in the analysis)!

\(^1\)Bhandari et al. (1997): Advanced Scout: datamining and knowledge discovery in NBA data. Data Mining and Knowledge Discovery, 1 (1) 121-125.
Unsupervised learning: Example 5

Solving the “cocktail-party problem”

- $n$ sound sources (e.g. people speaking)
- $n$ microphones at different points in the room
Unsupervised learning: Example 5

Solving the “cocktail-party problem”

- \( n \) sound sources (e.g. people speaking)
  \( n \) microphones at different points in the room

- Each microphone picks up a different combination of all the sound sources. The problem is to separate out the signals.

- Online demo:
  
  http://research.ics.tkk.fi/ica/cocktail/cocktail_en.cgi