Training the linear classifier

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

$$C(\mathbf{w}) = \#\{ (x_i, y_i) \mid y_i \neq \text{sign}(\mathbf{w}^T x_i) \}$$

is minimized.

- However, minimizing $C(\mathbf{w})$ is in the general case computationally hard: NP-complete!

- If it is known that the classes are linearly separable, i.e. that there exists a $\mathbf{w}^*$ such that $C(\mathbf{w}^*) = 0$, the problem can be solved easily
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. There are more complicated algorithms that have better worst-case behavior.

- 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 == true
        break
return \( w \)
Perceptron algorithm: Main ideas

- The algorithm keeps track of and updates a weight vector $w$
- 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 $w$ by adding $y_i x_i$. This turns $w$ towards $x_i$ if $y_i = +1$, and away from $x_i$ if $y_i = -1$
Perceptron algorithm: Illustration

Current state of $\mathbf{w}$
Perceptron algorithm: Illustration

Red point classified correctly, no change to $w$
Perceptron algorithm: Illustration

Green point classified correctly, no change to $w$
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.$\,$

- training example of class +1
- training example of class -1
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

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

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:

$$p \leq \frac{r^2}{\gamma^2}.$$
We just proved that if the training data is linearly separable, the perceptron algorithm converges after a finite number of steps. It is obvious that if the training data is not linearly separable, the algorithm does not converge, because it cannot (by definition) get through a full sweep of the data without performing an update.

What then?

- 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.
- Change the 0/1 cost function to something smoother.
Instead of giving a ‘forced choice’ prediction $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$, we could give probabilistic predictions:

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

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.
Logistic regression: Illustration

The constant-probability surfaces of $P(y = +1 \mid x)$ are hyperplanes, with the .5 hyperplane going through the origin:
Logistic regression: Learning

- 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 $\mathbf{w}$ 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...
  - $\lambda \| \mathbf{w} \|^2_2$ (quadratic penalty), or
  - $\lambda \| \mathbf{w} \|^1_1$ (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 *supervised*.

- In *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 *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):

- **Bass Pro Shops Outdoors Online**: Offering the best in Fishing...
  - The world's most exciting sporting goods available online. Truly everything an outdoor enthusiast needs.
  - http://www.basspro.com/ [Entireweb, Google]

- **G.H. Bass & Co. Official Online Store**: Classic and edgy styles of shoes.
  - Official site for GH BASS shoes. Classic and edgy styles of shoes for women, men and children.
  - https://bassshoes.harborgh.com/ [Google]

- **BASS Fishing Membership and Tournament News**: Information for Bass fishermen.
  - JERRY'S BASS BLOG. Jerry McKinnis is one of B.A.S.S.’s co-owners. Check out his Blog.

- **Bass guitar - Wikipedia**: The free encyclopedia.
  - The bass guitar is a stringed instrument played primarily with the fingers or thumb to produce sound.

- **Bass, British Beer, The Original Pale Ale**: A national retailer and brewer.
  - A national retailer and brewer with a large, varied portfolio and company history.
  - http://www.bass.com/ [Entireweb, Google]
Unsupervised learning: Example 2

Learning the structure of natural language:

- **Unigrams (‘bag of words’ model):**

  \[
P(\langle \text{word}_1, \ldots, \text{word}_N \rangle) = P(\text{word}_1) \ldots P(\text{word}_N)
  \]

- **Bigrams:**

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

- **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/generative/)

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:

![Graphs showing continuous-valued measurements]

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

- Standard classification problem? $\mathcal{X}$ is the measurements, $\mathcal{Y} = \{\text{normal}, \text{fault}\}$
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’).
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