Binary classification

- Input: \( x \in \mathcal{X} \)
  - e.g., a document or a sentence with words \( x = w_1 w_2 \ldots w_n \),
  - or a series of previous actions
- Output: \( y \in \mathcal{Y} = \{-1, 1\} \)
- We assume a mapping from inputs pairs \( x \) to a
  high-dimensional feature vector
  \[
  f(x) : \mathcal{X} \rightarrow \mathbb{R}^m
  \]
- To keep notation simple, we’ll write \( f(x) \) as \( x \in \mathbb{R}^m \)
  most of the time.
- For any vector \( v \in \mathbb{R}^m \), let \( v_j \) be the \( j \)th value.

Features and classes

- We want to work in a real vector space.
- All features must be numerical.
  - Numerical features are represented directly as \( f_i(x, y) \in \mathbb{R} \).
  - Boolean features are represented as \( f_i(x, y) \in \{0, 1\} \).
  - Categorical features are translated into sets of
    Boolean features (one-hot representation)
A data set in 2D space

How will the next point be classified?

1-NN classifier

How will the next point be classified?

Decision tree
The decision boundary of a linear classifier is a **hyperplane** in feature space.

What is a hyperplane?
- A hyperplane is a subspace whose dimension is one less than that of its ambient space (Wikipedia).
- A hyperplane splits the space in two halves.
- It is...
  - a **point** in 1-dimensional space,
  - a **line** in 2-dimensional space,
  - a **plane** in 3-dimensional space,
  - an \((m-1)\)-dimensional subspace in \(m\)-dimensional space.

Equation of a hyperplane

A hyperplane can be written as

\[
    w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = 0
\]

\[
    \sum_{i=1}^{m} w_i x_i + b = 0
\]

\[
    \mathbf{w} \cdot \mathbf{x} + b = 0
\]

In a linear classifier, we call \(\mathbf{w}\) the **weight vector** and \(b\) the **bias**.

The bias can also be written as a part of the weight vector if we add a constant feature \(x_0 = 1\).
Decision rule for linear classifiers

Classify examples on one side of the hyperplane as positive, examples on the other side as negative.

\[
\hat{y} = \begin{cases} 
1 & \text{if } w \cdot x + b > 0 \\
-1 & \text{if } w \cdot x + b < 0 
\end{cases}
= \text{sign}(w \cdot x + b)
\]

Linear separability

We cannot always achieve perfect classification of the training data with a linear classifier.

- Find an example that doesn’t work.
- How serious is this problem in practice?
- Can we work around it?

Learning linear classifiers

- Input: Training examples \( \mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N} \).
- Feature representation: \( f : \mathcal{X} \rightarrow \mathbb{R}^m \).
- Output: A vector \( w \) that optimises some important function of the training set:
  - minimise error (Perceptron, SVMs, Boosting)
  - maximise likelihood of data (logistic regression, Naive Bayes)
Learning by optimisation

General procedure for learning by optimisation:
- Start with an initial guess for \( w^{(0)} \).
- Process a single example (on-line) or a batch of examples at a time.
- In each step, apply a correction to the initial guess:
  \[ w^{(t+1)} \leftarrow w^{(t)} + \Delta w \]
- Repeat until a convergence criterion is satisfied.

Perceptron algorithm

- The perceptron algorithm is an on-line algorithm for training linear classifiers.
- It is an error-driven algorithm:
  Updates weights only after making errors.
- Intuition: Whenever you make an error, adjust the weights so they work better for this example.

Perceptron algorithm

1: \( w \leftarrow 0, b \leftarrow 0 \)
2: for a fixed number of iterations do
3:   for all \((x, y) \in \mathcal{T}\) do
4:     \( a \leftarrow w \cdot x + b \)
5:     if \( ya \leq 0 \) then
6:       \( w \leftarrow w + yx \)
7:       \( b \leftarrow b + y \)
8:     end if
9:   end for
10: end for
Why does the perceptron update work?

For misclassified examples, the weights are updated as

\[ w' \leftarrow w + yx \]
\[ b' \leftarrow b + y \]

After the update, the new perceptron activation is \( a' = w' \cdot x + b \).

We want to have

- \( a' > a \) if \( y = 1 \), and
- \( a' < a \) if \( y = -1 \).

Perceptron update

\[
\begin{align*}
a' &= w' \cdot x + b' \\
&= (w + yx) \cdot x + (b + y) \\
&= \sum_{i=1}^{m} (w_i + yx_i)x_i + (b + y) \\
&= \sum_{i=1}^{m} w_i x_i + b + y \sum_{i=1}^{m} x_i^2 + y \\
&= a + y \left( \sum_{i=1}^{m} x_i^2 + 1 \right)
\end{align*}
\]

What does this mean?

We know that, after the perceptron update, the current example will be closer to correct classification than before.

We do not know

- if it will be classified correctly,
- if any of the other examples will be classified correctly, or
- if the activation will still be better when we return to this example after seeing the rest of the training data.
The **margin of a data point** $x$ is its distance from the decision boundary.
Assuming a unit-length weight vector with $||w|| = \sqrt{\sum_{i} w_i^2} = 1$, then:

$$\text{margin}(x, w, b) = y(w \cdot x + b)$$

**Margin of a data set**

The margin of a data set given a separating hyperplane is the margin of the point closest to the decision boundary:

$$\text{margin}(\mathcal{T}, w, b) = \min_{(x, y) \in \mathcal{T}} y(w \cdot x + b)$$

The **best achievable margin of a data set** characterises the difficulty of the classification task:

$$\text{margin}(\mathcal{T}) = \max_{w, b} \text{margin}(\mathcal{T}, w, b)$$

**Perceptron convergence theorem**

Suppose the perceptron algorithm is run on a linearly separable data set $\mathcal{T}$ with margin $\gamma > 0$.
Assume that $||x|| \leq 1$ for all $x \in \mathcal{T}$.
Then the algorithm will converge after at most $\frac{1}{\gamma^2}$ updates.
Since the data set is separable with margin $\gamma$, we know that there exists a unit-length weight vector $w^*$ that realises this margin.

Let $w^{(k)}$ be the weight vector after $k$ updates.
We show that $w^* \cdot w^{(k)}$ increases with every update.

Since $w^* \cdot w^{(k)} = ||w^*|| ||w^{(k)}|| \cos \theta$, this could have two reasons:

1. $w^{(k)}$ just gets longer and longer.
   We show that this is not the main reason.
2. The angle $\theta$ between $w^{(k)}$ and $w^*$ decreases.
   This is in fact what happens.

The bounds on these expressions together imply that we can’t do more than $\frac{1}{\gamma^2}$ updates.

1. $w^* \cdot w^{(k)}$ increases.

- Suppose the $k$th update happens on example $(x, y)$.
- Now

$$w^* \cdot w^{(k)} = w^* \cdot \left(w^{(k-1)} + yx\right)$$

$$= w^* \cdot w^{(k-1)} +yw^* \cdot x$$

$$\geq w^* \cdot w^{(k-1)} + \gamma$$

We get an increase by $yw^* \cdot x \geq \gamma$ every time we update, so after $k$ updates

$$w^* \cdot w^{(k)} \geq k\gamma$$

2. The length of $w^{(k)}$ doesn’t increase very much.

- We’re looking at the situation right after an update, so we know that

$$yw^{(k-1)} \cdot x \leq 0$$

- So

$$||w^{(k)}||^2 = ||w^{(k-1)} + yx||^2$$

$$= ||w^{(k-1)}||^2 + 2yw^{(k-1)} \cdot x + y^2||x||^2$$

$$\leq ||w^{(k-1)}||^2 + 0 + 1$$

- The squared norm of $w^{(k)}$ increases by at most one at every update, so

$$||w^{(k)}||^2 \leq k$$
3. Putting things together

- From step 1, we know that $\mathbf{w}^* \cdot \mathbf{w}^{(k)} \geq k \gamma$.
- From step 2, we know that $||\mathbf{w}^{(k)}||^2 \leq k$, or $\sqrt{k} \geq ||\mathbf{w}^{(k)}||$.
- For all vectors $\mathbf{u}$ and $\mathbf{v}$, we have that $||\mathbf{u}|| ||\mathbf{v}|| \geq |\mathbf{u} \cdot \mathbf{v}|$ (Cauchy-Schwarz inequality).
- Putting things together, we have
  $$\sqrt{k} \geq ||\mathbf{w}^{(k)}|| = ||\mathbf{w}^{(k)}|| ||\mathbf{w}^*|| \geq |\mathbf{w}^* \cdot \mathbf{w}^{(k)}| = \mathbf{w}^* \cdot \mathbf{w}^{(k)} \geq k \gamma$$
- It follows that $\sqrt{k} \geq k \gamma$, and therefore $k \leq \frac{1}{\gamma^2}$.

Averaged perceptron

- During perceptron training, the weights can jump around quite a bit.
- The last example you see has a disproportionate influence.
- It works better to keep a running average of all weight vectors encountered during training.

Structured perceptron

- The perceptron can be extended to deal with structured data.
- Instead of binary classification, predict something like a parse tree, a tag sequence, etc.
- Do this at every step, update weights when the prediction was wrong.
Properties of the perceptron

The perceptron algorithm is appropriate when
- the data are (close to) linearly separable.
- many features
- include feature conjunctions
- you need a simple algorithm that is easy to implement.
- for structured prediction (structured perceptron).

Inductive bias:
- Decision boundary is a hyperplane.

Hyperparameters:
- Feature transformations
- Number of iterations

Next steps

- Next lecture: Tue 10 Apr, 14–16, in Chomsky.
  Reading: CiML, chapters 5–7
- No lecture on Thursday 12 April.
- Assignment 1 due end of next week.