Linear Classifiers 1

Christian Hardmeier

(based in part on materials by Joakim Nivre and Ryan McDonald)

2018-04-05

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

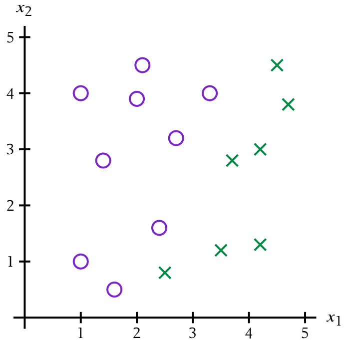
$$\mathbf{f}(x) : \mathcal{X} \to \mathbb{R}^m$$

- To keep notation simple, we'll write $\mathbf{f}(x)$ as $\mathbf{x} \in \mathbb{R}^m$ most of the time.
- For any vector $\mathbf{v} \in \mathbb{R}^m$, let v_j be the jth value.

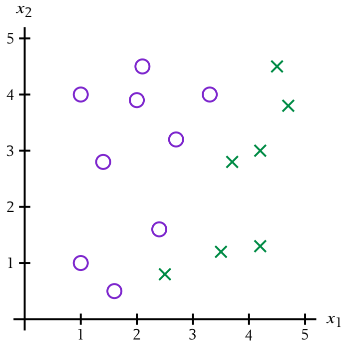
Features and classes

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

A data set in 2D space



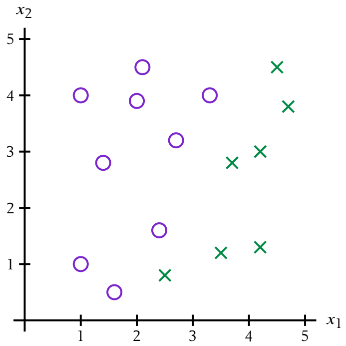
How will the next point be classified?

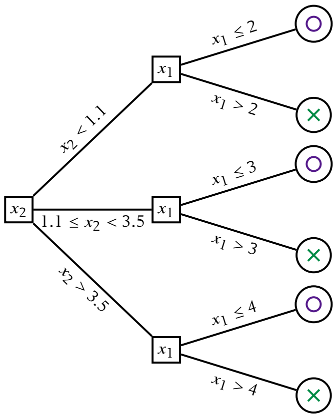


1-NN classifier

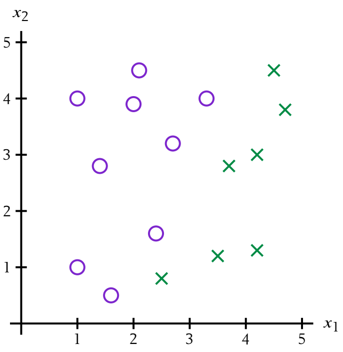
How will the next point be classified?

Decision tree





How will the next point be classified?



Linear classifier

Decision boundary of a linear classifier

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

$$
\begin{aligned}
w_1x_1 + w_2x_2 + \cdots + w_mx_m + b &= 0 \\
\sum_{i=1}^{m} w_i x_i + b &= 0 \\
\mathbf{w} \cdot \mathbf{x} + b &= 0
\end{aligned}
$$

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.

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

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 $\mathcal{T} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
- Feature representation: $\mathbf{f} : \mathcal{X} \to \mathbb{R}^m$.
- Output: A vector $\mathbf{w}$ 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 $\mathbf{w}^{(0)}$.
- Process a single example (on-line) or
  a batch of examples at a time.
- In each step, apply a correction to the initial guess:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \Delta\mathbf{w}$$

- 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

$\mathbf{w} \leftarrow \mathbf{0}, b \leftarrow 0$
for a fixed number of iterations **do**
    for all $(\mathbf{x}, y) \in \mathcal{T}$ **do**
        $a \leftarrow \mathbf{w} \cdot \mathbf{x} + b$
        if $ya \leq 0$ **then**
            $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$
            $b \leftarrow b + y$
        end if
    end for
end for

Why does the perceptron update work?

For misclassified examples, the weights are updated as

$$\begin{aligned} \mathbf{w}' &\leftarrow \mathbf{w} + y\mathbf{x} \\ b' &\leftarrow b + y \end{aligned}$$

After the update, the new perceptron activation is $a' = \mathbf{w}' \cdot \mathbf{x} + b$.
We want to have

- $a' > a$ if $y = 1$, and
- $a' < a$ if $y = -1$.

Perceptron update

$$\begin{aligned} a' &= \mathbf{w}' \cdot \mathbf{x} + b' \\ &= (\mathbf{w} + y\mathbf{x}) \cdot \mathbf{x} + (b + y) \\ &= \sum_{i=1}^{m} (w_i + y x_i) x_i + (b + y) \\ &= \sum_{i=1}^{m} w_i x_i + b + y \sum_{i=1}^{m} x_i^2 + y \\ &= a + y \left(\sum_{i=1}^{m} x_i^2 + 1 \right) \end{aligned}$$

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.

Margin of a point

The margin of a data point $\mathbf{x}$ is its distance from the decision boundary.

Assuming a unit-length weight vector with $||\mathbf{w}|| = \sqrt{\sum_i w_i^2} = 1$, then:

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

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:

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

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

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

Perceptron convergence theorem

Suppose the perceptron algorithm is run on a linearly separable data set $\mathcal{T}$ with margin $\gamma > 0$.

Assume that $||\mathbf{x}|| \leq 1$ for all $\mathbf{x} \in \mathcal{T}$.

Then the algorithm will converge after at most $\frac{1}{\gamma^2}$ updates.

Outline

- Since the data set is separable with margin γ, we know that there exists a unit-length weight vector $\mathbf{w}^*$ that realises this margin.
- Let $\mathbf{w}^{(k)}$ be the weight vector after k updates. We show that $\mathbf{w}^* \cdot \mathbf{w}^{(k)}$ increases with every update.
- Since $\mathbf{w}^* \cdot \mathbf{w}^{(k)} = ||\mathbf{w}^*|| \, ||\mathbf{w}^{(k)}|| \cos\theta$, this could have two reasons:
  1. $\mathbf{w}^{(k)}$ just gets longer and longer. We show that this is not the main reason.
  2. The angle θ between $\mathbf{w}^{(k)}$ and $\mathbf{w}^*$ 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. $\mathbf{w}^* \cdot \mathbf{w}^{(k)}$ increases.

- Suppose the kth update happens on example $(\mathbf{x}, y)$.
- Now

$$\begin{aligned}
\mathbf{w}^* \cdot \mathbf{w}^{(k)} &= \mathbf{w}^* \cdot \left(\mathbf{w}^{(k-1)} + y\mathbf{x}\right) \\
&= \mathbf{w}^* \cdot \mathbf{w}^{(k-1)} + y\mathbf{w}^* \cdot \mathbf{x} \\
&\geq \mathbf{w}^* \cdot \mathbf{w}^{(k-1)} + \gamma
\end{aligned}$$

- We get an increase by $y\mathbf{w}^* \cdot \mathbf{x} \geq \gamma$ every time we update, so after k updates

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

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

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

$$y\mathbf{w}^{(k-1)} \cdot \mathbf{x} \leq 0$$

- So

$$\begin{aligned}
||\mathbf{w}^{(k)}||^2 &= ||\mathbf{w}^{(k-1)} + y\mathbf{x}||^2 \\
&= ||\mathbf{w}^{(k-1)}||^2 + 2y\mathbf{w}^{(k-1)} \cdot \mathbf{x} + y^2||\mathbf{x}||^2 \\
&\leq ||\mathbf{w}^{(k-1)}||^2 + 0 + 1
\end{aligned}$$

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

$$||\mathbf{w}^{(k)}||^2 \leq k$$

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 $k \leq \dfrac{1}{\gamma^2}$.

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.