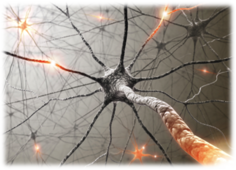
Machine learning: linear classification



- We now present linear (binary) classification, working through a simple example just like we did for linear regression.

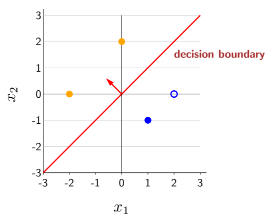
Linear classification framework

training data

	x_1	x_2	y
example	0	2	1
example	-2	0	1
example	1	-1	-1

learning algorithm $\longrightarrow$ $\boxed{f}$ **classifier**

$[2, 0]$ **input** $\downarrow$ $\boxed{f}$ $\downarrow$ -1 **label**



Design decisions:

Which classifiers are possible? **hypothesis class**

How good is a classifier? **loss function**

How do we compute the best classifier? **optimization algorithm**

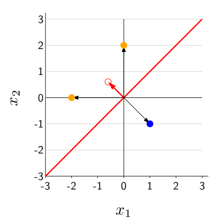
- Here is the linear classification framework.
- As usual, we are given **training data**, which consists of a set of examples. Each **example** consists of an input $x = (x_1, x_2)$ and an output y. We are considering two-dimensional inputs now to make the example a bit more interesting. The examples can be plotted, with the color denoting the label (orange for $+1$ and blue for -1).
- We still want a learning algorithm that takes the training data and produces a model f, which we will call a **classifier** in the context of classification.
- The classifier takes a new input and produces an output. We can visualize a classifier on the 2D plot by its **decision boundary**, which divides the input space into two regions: the region of input points that the classifier would output $+1$ and the region that the classifier would output -1. By convention, the arrow points to the positive region.
- Again, there are the same three design decisions to fully specify the learning algorithm:
- First, which classifiers f is the learning algorithm allowed to produce? Must the decision boundary be straight or can it curve? In other words, what is the **hypothesis class**?
- Second, how does the learning algorithm judge which classifier is good? In other words, what is the **loss function**?
- Finally, how does the learning algorithm actually find the best classifier? In other words, what is the **optimization algorithm**?

An example linear classifier

$$f(x) = \text{sign}(\overbrace{[-0.6, 0.6]}^{\mathbf{w}} \cdot \overbrace{[x_1, x_2]}^{\phi(x)})$$

$$\text{sign}(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \end{cases}$$

x_1	x_2	$f(x)$
0	2	1
-2	0	1
1	-1	-1



$$f([0, 2]) = \text{sign}([-0.6, 0.6] \cdot [0, 2]) = \text{sign}(1.2) = 1$$

$$f([-2, 0]) = \text{sign}([-0.6, 0.6] \cdot [-2, 0]) = \text{sign}(1.2) = 1$$

$$f([1, -1]) = \text{sign}([-0.6, 0.6] \cdot [1, -1]) = \text{sign}(-1.2) = -1$$

Decision boundary: x such that $\mathbf{w} \cdot \phi(x) = 0$

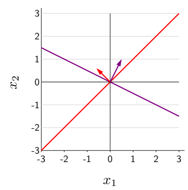
- Before we talk about the hypothesis class over all classifiers, we will start by exploring the properties of a specific linear classifier.
- First take the dot product between a fixed weight vector $\mathbf{w}$ and the identity feature vector $\phi(x)$. Then take the sign of the dot product.
- The **sign** of a number z is $+1$ if $z > 0$ and -1 if $z < 0$ and 0 if $z = 0$.
- Let's now visualize what f does. First, we can plot $\mathbf{w}$ either as a point or as a vector from the origin to that point; the latter will be most useful for our purposes.
- Let's feed some inputs into f.
- Take the first point, which can be visualized on the plot as a vector. Recall from linear algebra that the dot product between two vectors is the cosine of the angle between them. In particular, the dot product is positive iff the angle is acute and negative iff the angle is obtuse. The first point forms an acute angle and therefore is classified as $+1$, which can also be verified mathematically.
- The second point also forms an acute angle and therefore is classified as $+1$.
- The third point forms an obtuse angle and is classified as -1.
- Now you can hopefully see the pattern now. All points in this region (consisting of points forming an acute angle) will be classified $+1$, and all points in this region (consisting of points forming an obtuse angle) will be classified -1.
- Points x which form a right angle ($\mathbf{w} \cdot \phi(x) = 0$) form the **decision boundary**.
- Indeed, you can see pictorially that the decision boundary is perpendicular to the weight vector.

Hypothesis class: which classifiers?

$\phi(x) = [x_1, x_2]$

$f(x) = \text{sign}([-0.6, 0.6] \cdot \phi(x))$

$f(x) = \text{sign}([0.5, 1] \cdot \phi(x))$



General binary classifier:

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

Hypothesis class:

$$\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^2\}$$

- We've looked at one particular red classifier.
- We can also consider an alternative purple classifer, which has a different decision boundary.
- In general for binary classification, given a particular weight vector $\mathbf{w}$ we define $f_{\mathbf{w}}$ to be the sign of the dot product.

Loss function: how good is a classifier?

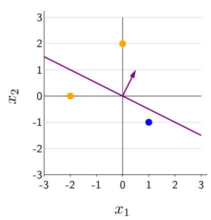
$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$

$\mathbf{w} = [0.5, 1]$

$\phi(x) = [x_1, x_2]$

training data $\mathcal{D}_{\text{train}}$

x_1	x_2	y
0	2	1
-2	0	1
1	-1	-1



$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$ **zero-one loss**

$\text{Loss}([0, 2], 1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [0, 2]) \neq 1] = 0$

$\text{Loss}([-2, 0], 1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [-2, 0]) \neq 1] = 1$

$\text{Loss}([1, -1], -1, [0.5, 1]) = \mathbf{1}[\text{sign}([0.5, 1] \cdot [1, -1]) \neq -1] = 0$

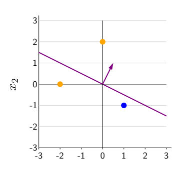
$\text{TrainLoss}([0.5, 1]) = 0.33$

- Now we proceed to the second design decision: the loss function, which measures how good a classifier is.
- Let us take the purple classifier, which can be visualized on the graph, as well as the training examples.
- Now we want to define a loss function that captures how the model predictions deviate from the data. We will define the **zero-one loss** to check if the model prediction $f_{\mathbf{w}}(x)$ disagrees with the target label y. If so, then the indicator function $\mathbf{1}[f_{\mathbf{w}}(x) \neq y]$ will return 1; otherwise, it will return 0.
- Let's see this classifier in action.
- For the first training example, the prediction is 1, the target label is 1, so the loss is 0.
- For the second training example, the prediction is -1, the target label is 1, so the loss is 1.
- For the third training example, the prediction is -1, the target label is -1, so the loss is 0.
- The total loss is simply the average over all the training examples, which yields $1/3$.

Score and margin

Predicted label: $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

Target label: y



Definition: score

The score on an example (x, y) is $\mathbf{w} \cdot \phi(x)$, how **confident** we are in predicting $+1$.

Definition: margin

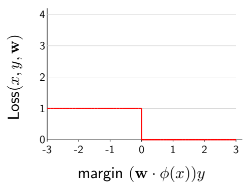
The margin on an example (x, y) is $(\mathbf{w} \cdot \phi(x))y$, how **correct** we are.

- Before we move to the third design decision (optimization algorithm), let us spend some time understanding two concepts so that we can rewrite the zero-one loss.
- Recall the definition of the predicted label and the target label.
- The first concept, which we already have encountered is the **score**. In regression, this is the predicted output, but in classification, this is the number before taking the sign.
- Intuitively, the score measures how confident the classifier is in predicting $+1$.
- Points farther away from the decision boundary have larger scores.
- The second concept is **margin**, which measures how correct the prediction is. The larger the margin the more correct, and non-positive margins correspond to classification errors. If $y = 1$, then the score needs to be very positive for a large margin. If $y = -1$, then the score needs to be very negative for a large margin.
- Note that if we look at the actual prediction $f_{\mathbf{w}}(x)$, we can only ascertain whether the prediction was right or not.
- By looking at the score and the margin, we can get a more nuanced view into the behavior of the classifier.

Zero-one loss rewritten

Definition: zero-one loss

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$
$$= \mathbf{1}[\underbrace{(\mathbf{w} \cdot \phi(x))y}_{\text{margin}} \leq 0]$$



- Now let us rewrite the zero-one loss in terms of the margin.
- We can also plot the loss against the margin.
- Again, a positive margin yields zero loss while a non-positive margin yields loss 1.

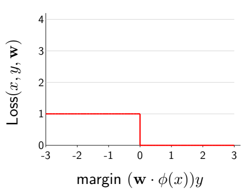
Optimization algorithm: how to compute best?

Goal: $\min_{\mathbf{w}}$ TrainLoss($\mathbf{w}$)

To run gradient descent, compute the gradient:

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \nabla \text{Loss}_{0\text{-}1}(x, y, \mathbf{w})$$

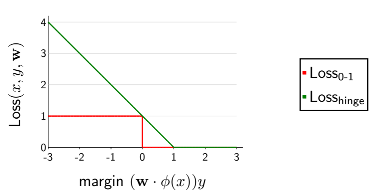
$$\nabla_{\mathbf{w}} \text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \nabla \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$



Gradient is zero almost everywhere!

- Now we consider the third design decision, the optimization algorithm for minimizing the training loss.
- Let's just go with gradient descent. Recall that to run gradient descent, we need to first compute the gradient.
- The gradient of the training loss is just the average over the per-example losses. And then we need to take the gradient of indicator function...
- But this is where we run into problems: recall that the zero-one loss is flat almost everywhere (except at margin $= 0$), so the gradient is zero almost everywhere.
- If you try running gradient descent on a function with zero gradient, you will be stuck.
- One's first reaction to why the zero-one loss is hard to optimize is that it is not differentiable (everywhere). However, that is not really the real reason. The real reason is because it has zero gradients.

Hinge loss

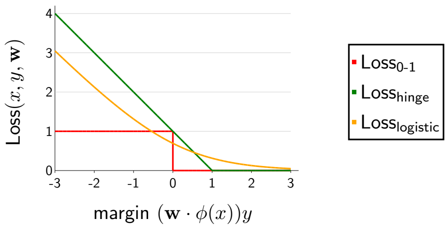


$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$

- To fix this problem, we have to choose another loss function.
- A popular loss function is the **hinge loss**, which is the maximum over a descending line and the zero function. It is best explained visually.
- If the margin is at least 1, then the hinge loss is zero.
- If the margin is less than 1, then the hinge loss rises linearly.
- The 1 is there to provide some buffer: we ask the classifier to predict not only correctly, but by a (positive) margin of safety.
- Aside: Technically, the 1 can be any positive number. If we have regularization, it is equivalent to setting the regularization strength.
- Also note that the hinge loss is an upper bound on the zero-one loss, so driving down the hinge loss will generally drive down the zero-one loss. In particular, if the hinge loss is zero, the zero-one loss must also be zero.

Digression: logistic regression

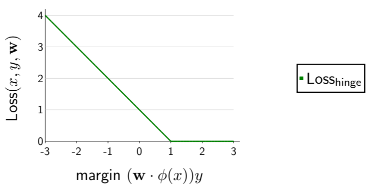
$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w}\cdot\phi(x))y})$$



Intuition: Try to increase margin even when it already exceeds 1

- Another popular loss function used in machine learning is the **logistic loss**.
- The main property of the logistic loss is no matter how correct your prediction is, you will have non-zero loss, and so there is still an incentive (although a diminishing one) to push the loss down by increasing the margin.

Gradient of the hinge loss



$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w}\cdot\phi(x))y, 0\}$$

$$\nabla\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } \{1 - (\mathbf{w}\cdot\phi(x))y\} > \{0\} \\ 0 & \text{otherwise} \end{cases}$$

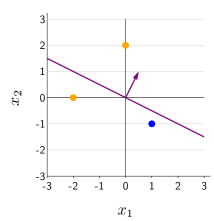
- You should try to "see" the solution before you write things down formally. Pictorially, it should be evident: when the margin is less than 1, then the gradient is the gradient of $1 - (\mathbf{w}\cdot\phi(x))y$, which is equal to $-\phi(x)y$. If the margin is larger than 1, then the gradient is the gradient of 0, which is 0. Combining the two cases: $\nabla_{\mathbf{w}}\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } (\mathbf{w}\cdot\phi(x))y < 1 \\ 0 & \text{if } (\mathbf{w}\cdot\phi(x))y > 1. \end{cases}$
- What about when the margin is exactly 1? Technically, the gradient doesn't exist because the hinge loss is not differentiable there. But in practice, you can take either $-\phi(x)y$ or 0.
- Technical note (can be skipped): given $f(\mathbf{w})$, the gradient $\nabla f(\mathbf{w})$ is only defined at points $\mathbf{w}$ where f is differentiable. However, subdifferentials $\partial f(\mathbf{w})$ are defined at every point (for convex functions). The subdifferential is a set of vectors called subgradients $z \in f(\mathbf{w})$ which define linear underapproximations to f, namely $f(\mathbf{w}) + z\cdot(\mathbf{w}' - \mathbf{w}) \le f(\mathbf{w}')$ for all $\mathbf{w}'$.

Hinge loss on training data

$f_{\mathbf{w}}(x) = \mathbf{w}\cdot\phi(x)$
$\mathbf{w} = [0.5, 1]$
$\phi(x) = [x_1, x_2]$

training data $\mathcal{D}_{\text{train}}$

x_1	x_2	y
0	2	1
-2	0	1
1	-1	-1



$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w}\cdot\phi(x))y, 0\}$$

Loss$([0, 2], 1, [0.5, 1]) = \max\{1 - [0.5, 1]\cdot[0, 2](1), 0\} = 0$ $\quad$ ∇Loss$([0, 2], 1, [0.5, 1]) = [0, 0]$

Loss$([-2, 0], 1, [0.5, 1]) = \max\{1 - [0.5, 1]\cdot[-2, 0](1), 0\} = 2$ $\quad$ ∇Loss$([-2, 0], 1, [0.5, 1]) = [2, 0]$

Loss$([1, -1], -1, [0.5, 1]) = \max\{1 - [0.5, 1]\cdot[1, -1](-1), 0\} = 0.5$ $\quad$ ∇Loss$([1, -1], -1, [0.5, 1]) = [1, -1]$

TrainLoss$([0.5, 1]) = 0.83$ $\quad$ ∇TrainLoss$([0.5, 1]) = [1, -0.33]$

- Now let us revisit our earlier setting with the hinge loss.
- For each example (x, y), we can compute its loss, and the final loss is the average.
- For the first example, the loss is zero, and therefore the gradient is zero.
- For the second example, the loss is non-zero which is expected since the classifier is incorrect. The gradient is non-zero.
- For the third example, note that the loss is non-zero even though the classifier is correct. This is because we have to have a margin of 1, but the margin in this case is only 0.5.

Gradient descent (hinge loss) in Python

[code]

- Let us start from the regression code and change the loss function.
- Note that we don't have to modify the optimization algorithm at all, a benefit of decoupling the objective function from the optimization algorithm.

Summary so far

$$\underbrace{\mathbf{w} \cdot \phi(x)}_{\text{score}}$$

	Regression	Classification
Prediction $f_{\mathbf{w}}(x)$	score	sign(score)
Relate to target y	residual (score $- y$)	margin (score y)
Loss functions	squared absolute deviation	zero-one hinge logistic
Algorithm	gradient descent	gradient descent

- Let us end by comparing and contrasting linear classification and linear regression.
- The score is a common quantity that drives the prediction in both cases.
- In regression, the output is the raw score. In classification, the output is the sign of the score.
- To assess whether the prediction is correct, we must relate the score to the target y. In regression, we use the residual, which is the difference (lower is better). In classification, we use the margin, which is the product (higher is better).
- Given these two quantities, we can form a number of different loss functions. In regression, we studied the squared loss, but we could also consider the absolute deviation loss (taking absolute values instead of squared). In classification, we care about the zero-one loss (which corresponds to the missclassification rate), but we optimize the hinge or the logistic loss.
- Finally, gradient descent can be used in both settings.