https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec03.pdf

CSC 311: Introduction to Machine Learning

Lecture 3 - Linear Classifiers, Logistic Regression, Multiclass Classification

Roger Grosse   Chris Maddison   Juhan Bae   Silviu Pitis

University of Toronto, Fall 2020

Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target
- Examples of binary classification
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent

Overview

Binary linear classification

- **classification:** given a D-dimensional input $\mathbf{x} \in \mathbb{R}^D$ predict a discrete-valued target
- **binary:** predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples. Sorry.
  - $t \in \{0, 1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- **linear:** model prediction y is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Some Simplifications

Eliminating the threshold

- We can assume without loss of generality (WLOG) that the threshold $r = 0$:

$$\mathbf{w}^\top \mathbf{x} + b \geq r \quad \Longleftrightarrow \quad \mathbf{w}^\top \mathbf{x} + \underbrace{b - r}_{\triangleq w_0} \geq 0.$$

Eliminating the bias

- Add a dummy feature x_0 which always takes the value 1. The weight $w_0 = b$ is equivalent to a bias (same as linear regression)

Simplified model

- Receive input $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$:

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Examples

- Let's consider some simple examples to examine the properties of our model
- Let's focus on minimizing the training set error, and forget about whether our model will generalize to a test set.

Examples

NOT

x_0	x_1	t
1	0	1
1	1	0

- Suppose this is our training set, with the dummy feature x_0 included.
- Which conditions on w_0, w_1 guarantee perfect classification?
  - When $x_1 = 0$, need: $z = w_0 x_0 + w_1 x_1 \geq 0 \iff w_0 \geq 0$
  - When $x_1 = 1$, need: $z = w_0 x_0 + w_1 x_1 < 0 \iff w_0 + w_1 < 0$
- Example solution: $w_0 = 1, w_1 = -2$
- Is this the only solution?

Examples

AND

x_0	x_1	x_2	t
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2$$

need: $w_0 < 0$

need: $w_0 + w_2 < 0$

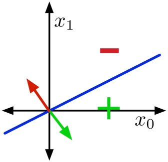
need: $w_0 + w_1 < 0$

need: $w_0 + w_1 + w_2 \geq 0$

Example solution: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$

The Geometric Picture

Input Space, or Data Space for NOT example

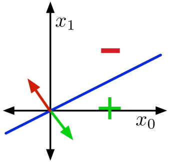


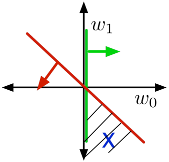
x_0	x_1	t
1	0	1
1	1	0

- Training examples are points
- Weights (hypotheses) $\mathbf{w}$ can be represented by half-spaces $H_+ = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} \geq 0\}$, $H_- = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} < 0\}$
  - The boundaries of these half-spaces pass through the origin (why?)
- The boundary is the decision boundary: $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0\}$
  - In 2-D, it's a line, but in high dimensions it is a hyperplane
- If the training examples can be perfectly separated by a linear decision rule, we say data is linearly separable.

The Geometric Picture

Weight Space





$w_0 \geq 0$

$w_0 + w_1 < 0$

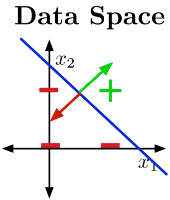
- Weights (hypotheses) $\mathbf{w}$ are points
- Each training example $\mathbf{x}$ specifies a half-space $\mathbf{w}$ must lie in to be correctly classified: $\mathbf{w}^\top \mathbf{x} \geq 0$ if $t = 1$.
- For NOT example:
  - $x_0 = 1, x_1 = 0, t = 1 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 \geq 0\}$
  - $x_0 = 1, x_1 = 1, t = 0 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 + w_1 < 0\}$
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible, otw it is infeasible.

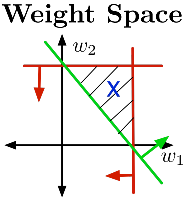
The Geometric Picture

- The **AND** example requires three dimensions, including the dummy one.
- To visualize data space and weight space for a 3-D example, we can look at a 2-D slice.
- The visualizations are similar.
  - Feasible set will always have a corner at the origin.

The Geometric Picture

Visualizations of the **AND** example





- Slice for $x_0 = 1$ and
- example sol: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$
- decision boundary:

$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$
$\implies -1.5 + x_1 + x_2 = 0$

- Slice for $w_0 = -1.5$ for the constraints
- $w_0 < 0$
- $w_0 + w_2 < 0$
- $w_0 + w_1 < 0$
- $w_0 + w_1 + w_2 \geq 0$

Summary — Binary Linear Classifiers

- **Summary:** Targets $t \in \{0, 1\}$, inputs $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$, and model is defined by weights $\mathbf{w}$ and

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- How can we find good values for $\mathbf{w}$?
- If training set is linearly separable, we could solve for $\mathbf{w}$ using linear programming
  - We could also apply an iterative procedure known as the *perceptron algorithm* (but this is primarily of historical interest).
- If it's not linearly separable, the problem is harder
  - Data is almost never linearly separable in real life.

Towards Logistic Regression

Loss Functions

- Instead: define loss function then try to minimize the resulting cost function
  - Recall: cost is loss averaged (or summed) over the training set
- Seemingly obvious loss function: 0-1 loss

$$\mathcal{L}_{0-1}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases}$$
$$= \mathbb{I}[y \neq t]$$

Attempt 1: 0-1 loss

- Usually, the cost $\mathcal{J}$ is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]$$

Attempt 1: 0-1 loss

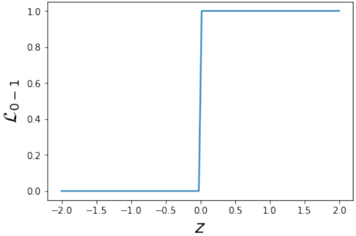
- Problem: how to optimize? In general, a hard problem (can be NP-hard)
- This is due to the step function (0-1 loss) not being nice (continuous/smooth/convex etc)

Attempt 1: 0-1 loss

- Minimum of a function will be at its critical points.
- Let's try to find the critical point of 0-1 loss
- Chain rule:

$$\frac{\partial \mathcal{L}_{0-1}}{\partial w_j} = \frac{\partial \mathcal{L}_{0-1}}{\partial z} \frac{\partial z}{\partial w_j}$$

- But $\partial \mathcal{L}_{0-1}/\partial z$ is zero everywhere it's defined!



  - $\partial \mathcal{L}_{0-1}/\partial w_j = 0$ means that changing the weights by a very small amount probably has no effect on the loss.
  - Almost any point has 0 gradient!

Attempt 2: Linear Regression

- Sometimes we can replace the loss function we care about with one which is easier to optimize. This is known as relaxation with a smooth surrogate loss function.
- One problem with $\mathcal{L}_{0-1}$: defined in terms of final prediction, which inherently involves a discontinuity
- Instead, define loss in terms of $\mathbf{w}^\top \mathbf{x}$ directly
  - Redo notation for convenience: $z = \mathbf{w}^\top \mathbf{x}$

Attempt 2: Linear Regression

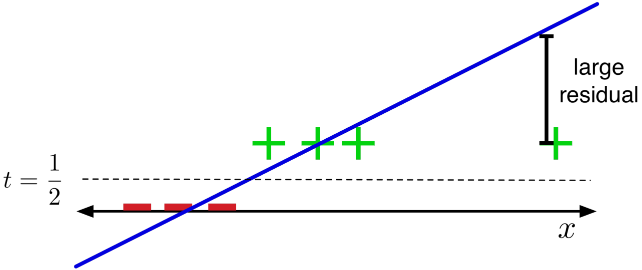
- We already know how to fit a linear regression model. Can we use this instead?

$$z = \mathbf{w}^\top \mathbf{x}$$
$$\mathcal{L}_{\mathrm{SE}}(z, t) = \frac{1}{2}(z - t)^2$$

- Doesn't matter that the targets are actually binary. Treat them as continuous values.
- For this loss function, it makes sense to make final predictions by thresholding z at $\frac{1}{2}$ (why?)

Attempt 2: Linear Regression

The problem:



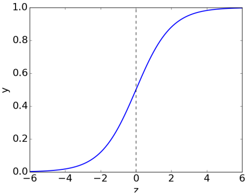
- The loss function hates when you make correct predictions with high confidence!
- If $t = 1$, it's more unhappy about $z = 10$ than $z = 0$.

Attempt 3: Logistic Activation Function

- There's obviously no reason to predict values outside [0, 1]. Let's squash y into this interval.

- The logistic function is a kind of sigmoid, or S-shaped function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



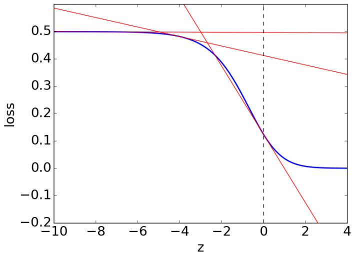
- $\sigma^{-1}(y) = \log(y/(1-y))$ is called the logit.
- A linear model with a logistic nonlinearity is known as log-linear:

$$z = \mathbf{w}^\top \mathbf{x}$$
$$y = \sigma(z)$$
$$\mathcal{L}_{\mathrm{SE}}(y, t) = \frac{1}{2}(y - t)^2.$$

- Used in this way, σ is called an activation function.

Attempt 3: Logistic Activation Function

The problem:
(plot of $\mathcal{L}_{\mathrm{SE}}$ as a function of z, assuming $t = 1$)



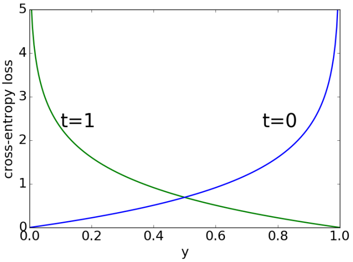
$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_j}$$

- For $z \ll 0$, we have $\sigma(z) \approx 0$.
- $\frac{\partial \mathcal{L}}{\partial z} \approx 0$ (check!) $\implies \frac{\partial \mathcal{L}}{\partial w_j} \approx 0 \implies$ derivative w.r.t. w_j is small $\implies$ w_j is like a critical point
- If the prediction is really wrong, you should be far from a critical point (which is your candidate solution).

Logistic Regression

- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$. If $t = 0$, then we want to heavily penalize $y \approx 1$.
- The pundits who were 99% confident Clinton would win were much more wrong than the ones who were only 90% confident.
- Cross-entropy loss (aka log loss) captures this intuition:

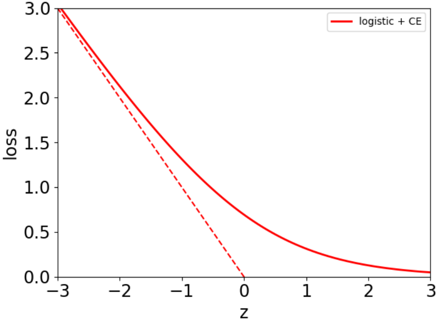
$$\mathcal{L}_{\text{CE}}(y, t) = \begin{cases} -\log y & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases}$$
$$= -t \log y - (1 - t) \log(1 - y)$$



Logistic Regression

Logistic Regression:

$$
\begin{aligned}
z &= \mathbf{w}^\top \mathbf{x} \\
y &= \sigma(z) \\
&= \frac{1}{1 + e^{-z}} \\
\mathcal{L}_{\text{CE}} &= -t \log y - (1 - t) \log(1 - y)
\end{aligned}
$$



Plot is for target $t = 1$.

Logistic Regression — Numerical Instabilities

- If we implement logistic regression naively, we can end up with numerical instabilities.
- Consider: $t = 1$ but you're really confident that $z \ll 0$.
- If y is small enough, it may be numerically zero. This can cause very subtle and hard-to-find bugs.

$$
\begin{aligned}
y &= \sigma(z) && \Rightarrow y \approx 0 \\
\mathcal{L}_{\text{CE}} &= -t \log y - (1-t)\log(1-y) && \Rightarrow \text{ computes } \log 0
\end{aligned}
$$

Logistic Regression — Numerically Stable Version

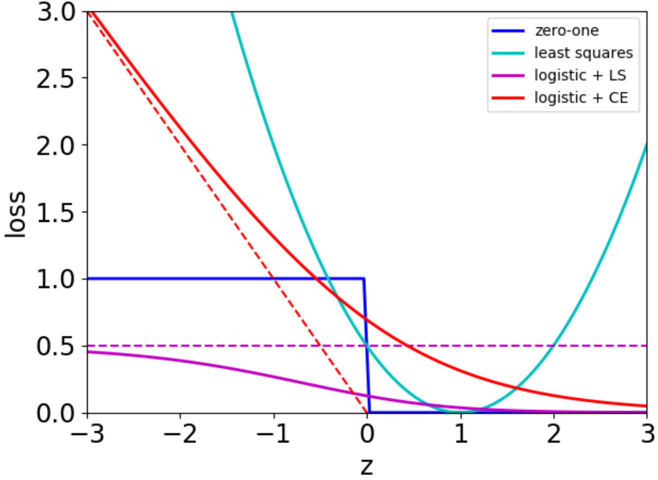
- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function.

$$\mathcal{L}_{\text{LCE}}(z,t) = \mathcal{L}_{\text{CE}}(\sigma(z),t) = t\log(1+e^{-z}) + (1-t)\log(1+e^{z})$$

- Numerically stable computation:

```
E = t * np.logaddexp(0, -z) + (1-t) * np.logaddexp(0, z)
```

Comparison of loss functions: (for $t = 1$)



Gradient Descent for Logistic Regression

- How do we minimize the cost $\mathcal{J}$ for logistic regression? No direct solution.
  - Taking derivatives of $\mathcal{J}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- However, the logistic loss is a convex function in $\mathbf{w}$, so let's consider the gradient descent method from last lecture.
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - A standard initialization is $\mathbf{w} = 0$. (why?)

Gradient of Logistic Loss

Back to logistic regression:

$$\mathcal{L}_{\text{CE}}(y, t) = -\, t \log(y) - (1 - t) \log(1 - y)$$
$$y = 1/(1 + e^{-z}) \quad \text{and} \quad z = \mathbf{w}^\top \mathbf{x}$$

Therefore

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_j} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1 - t}{1 - y} \right) \cdot y(1 - y) \cdot x_j$$
$$= (y - t) x_j$$

(verify this)

Gradient descent (coordinatewise) update to find the weights of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$
$$= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)}$$

Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- Linear regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)}$$

- Logistic regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)}$$

- Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- Notice $\frac{1}{N}$ in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses ($\alpha' = \alpha/N$).

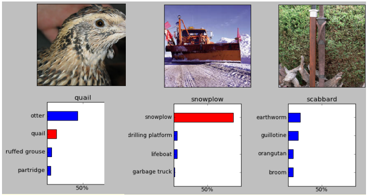
Multiclass Classification and Softmax Regression

Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target
- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal

Multiclass Classification

- Classification tasks with more than two categories:



Multiclass Classification

- Targets form a discrete set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases.
- A linear function of the inputs:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1, 2, ..., K$$

- We can eliminate the bias $\mathbf{b}$ by taking $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$. So, vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \qquad \text{or with dummy } x_0 = 1 \qquad \mathbf{z} = \mathbf{W}\mathbf{x}$$

Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of z_k as an measure of how much the model prefers k as its prediction.
- If we do this, we should set

$$y_i = \begin{cases} 1 & i = \arg\max_k z_k \\ 0 & \text{otherwise} \end{cases}$$

- **Exercise:** how does the case of $K = 2$ relate to the prediction rule in binary linear classifiers?

Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

  - Outputs can be interpreted as probabilities (positive and sum to 1)
  - If z_k is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- The inputs z_k are called the logits.

Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$
\begin{aligned}
\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log y_k \\
&= -\mathbf{t}^\top (\log \mathbf{y}),
\end{aligned}
$$

  where the log is applied elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Softmax Regression

- Softmax regression (with dummy $x_0 = 1$):

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{z})$$
$$\mathcal{L}_{\mathrm{CE}} = -\mathbf{t}^\top(\log \mathbf{y})$$

- Gradient descent updates can be derived for each row of $\mathbf{W}$:

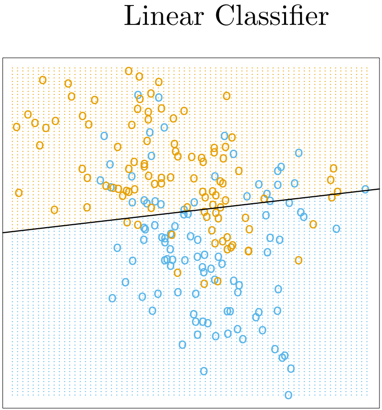
$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$
$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

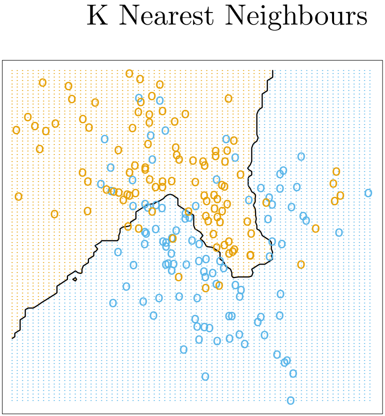
- Similar to linear/logistic reg (no coincidence) (verify the update)

Linear Classifiers vs. KNN

Linear Classifiers vs. KNN

Linear classifiers and KNN have very different decision boundaries:





Linear Classifiers vs. KNN

Advantages of linear classifiers over KNN?

Advantages of KNN over linear classifiers?

A Few Basic Concepts

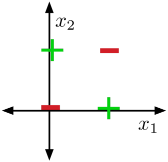
- A hypothesis is a function $f : \mathcal{X} \to \mathcal{T}$ that we might use to make predictions (recall $\mathcal{X}$ is the input space and $\mathcal{T}$ is the target space).
- The hypothesis space $\mathcal{H}$ for a particular machine learning model or algorithm is set of hypotheses that it can represent.
  - E.g., in linear regression, $\mathcal{H}$ is the set of functions that are linear in the data features
  - The job of a machine learning algorithm is to find a good hypothesis $f \in \mathcal{H}$
- The members of $\mathcal{H}$, together with an algorithm's preference for some hypotheses of $\mathcal{H}$ over others, determine an algorithm's inductive bias.
  - Inductive biases can be understood as general natural patterns or domain knowledge that help our algorithms to generalize; E.g., linearity, continuity, simplicity (L_2 regularization) ...
  - The so-called No Free Lunch (NFL) theorems assert that if datasets/problems were not naturally biased, no ML algorithm would be better than another

A Few Basic Concepts

- If an algorithm's hypothesis space $\mathcal{H}$ can be defined using a finite set of parameters, denoted $\boldsymbol{\theta}$, we say the algorithm is parametric.
  - In linear regression, $\boldsymbol{\theta} = (\mathbf{w}, b)$
  - Other examples: logistic regression, neural networks, k-means and Gaussian mixture models
- If the members of $\mathcal{H}$ are defined in terms of the data, we say that the algorithm is non-parametric.
  - In k-nearest neighbors, the learned hypothesis is defined in terms of the training data
  - Other examples: Gaussian processes, decision trees, support vector machines, kernel density estimation
  - These models can sometimes be understood as having an infinite number of parameters

Limits of Linear Classification

Some datasets are not linearly separable, e.g. **XOR**

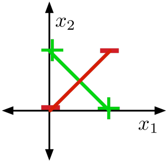


Visually obvious, but how to show this?

Limits of Linear Classification

Showing that XOR is not linearly separable (proof by contradiction)

- If two points lie in a half-space, line segment connecting them also lie in the same halfspace.
- Suppose there were some feasible weights (hypothesis). If the positive examples are in the positive half-space, then the green line segment must be as well.
- Similarly, the red line segment must line within the negative half-space.



- But the intersection can't lie in both half-spaces. Contradiction!

Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for **XOR**:

$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

x_1	x_2	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	t
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable. (Try it!)

Next time...

Feature maps are hard to design well, so next time we'll see how to *learn* nonlinear feature maps directly using neural networks...