CSC 311: Introduction to Machine Learning

Lecture 4 - Linear Models II

Roger Grosse Rahul G. Krishnan Guodong Zhang

University of Toronto, Fall 2021

Overview

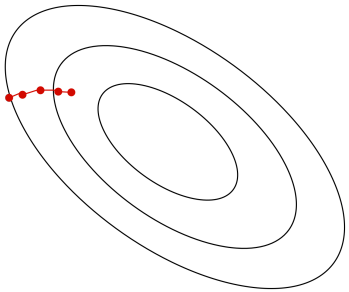
- More about gradient descent
  - Choosing a learning rate
  - Stochastic gradient descent

- Classification: predicting a discrete-valued target
  - Binary classification (this week): predicting a binary-valued target
  - Multiclass classification (next week): predicting a discrete-valued target

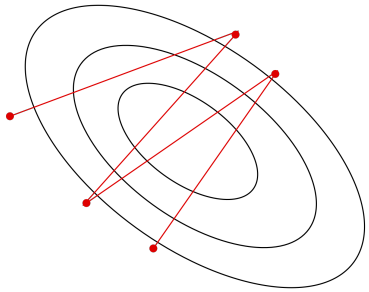
Setting the learning rate

Learning Rate (Step Size)

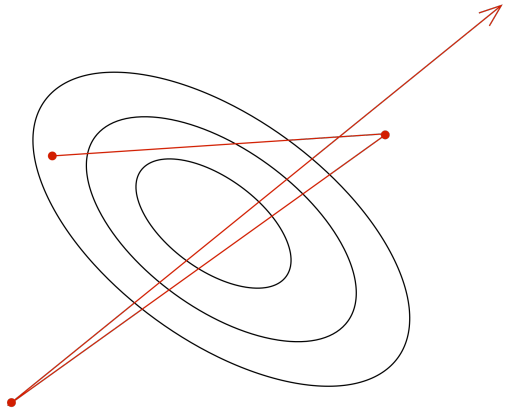
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small:
slow progress



α too large:
oscillations

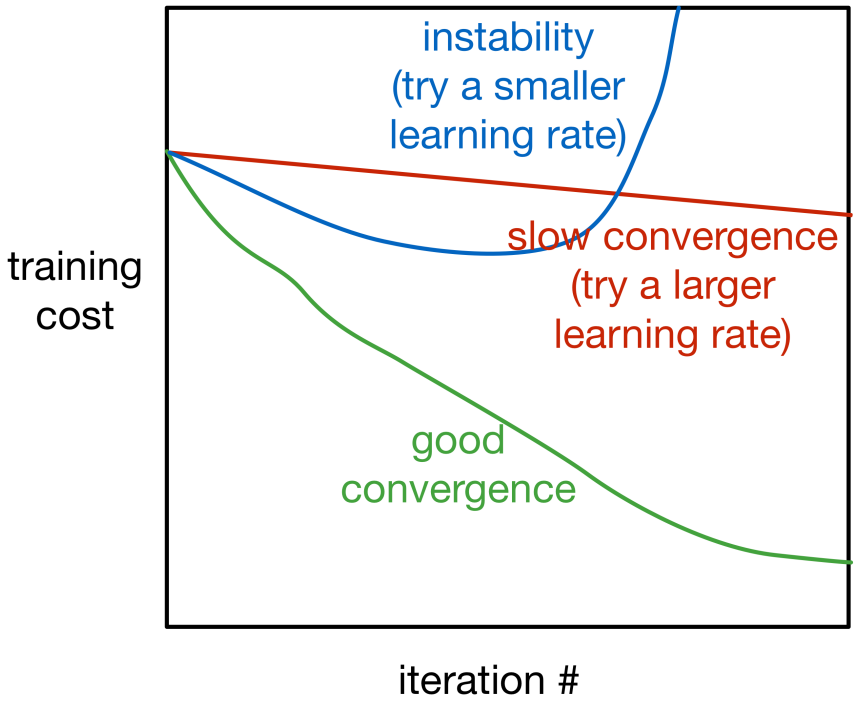


α much too large:
instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

Training Curves

- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.



- Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

Stochastic gradient descent

Stochastic Gradient Descent

- So far, the cost function $\mathcal{J}$ has been the average loss over the training examples:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}^{(i)} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)}).$$

  ($\boldsymbol{\theta}$ denotes the parameters; e.g., in linear regression, $\boldsymbol{\theta} = (\mathbf{w}, b)$)

- By linearity,

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}.$$

- Computing the gradient requires summing over *all* of the training examples. This is known as batch training.
- Batch training is impractical if you have a large dataset $N \gg 1$ (e.g. millions of training examples)!

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) updates the parameters based on the gradient for a single training example:

$$\begin{aligned} &1- && \text{Choose } i \text{ uniformly at random,} \\ &2- && \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} \end{aligned}$$

- Cost of each SGD update is independent of N!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}.$$

Stochastic Gradient Descent

- Problems with using single training example to estimate gradient:
  - Variance in the estimate may be high
  - We can't exploit efficient vectorized operations
- Compromise approach:
  - compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, N\}$, called a mini-batch.
  - For purposes of analysis, we often assume the examples in the mini-batch are sampled independetly and uniformly with replacement.
  - In practice, we typically permute the training set and then go through it sequentially. Each pass over the data is called an epoch.

Stochastic Gradient Descent

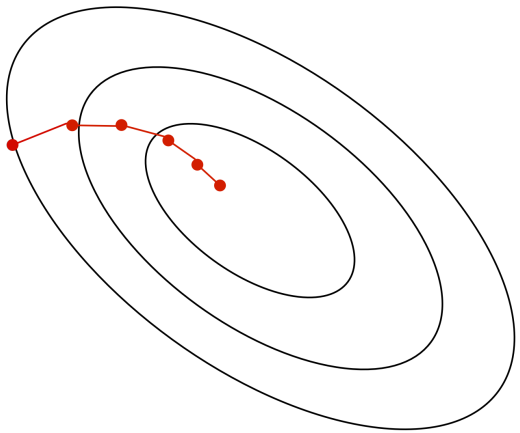
- Stochastic gradients computed on larger mini-batches have smaller variance. This is similar to bagging.
  - If the training examples are sampled independently, we can apply the linearity rule for variance.

$$\text{Var}\left[\frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}}\frac{\partial\mathcal{L}^{(i)}}{\partial\theta_j}\right] = \frac{1}{|\mathcal{M}|^2}\sum_{i\in\mathcal{M}}\text{Var}\left[\frac{\partial\mathcal{L}^{(i)}}{\partial\theta_j}\right] = \frac{1}{|\mathcal{M}|}\text{Var}\left[\frac{\partial\mathcal{L}^{(i)}}{\partial\theta_j}\right]$$

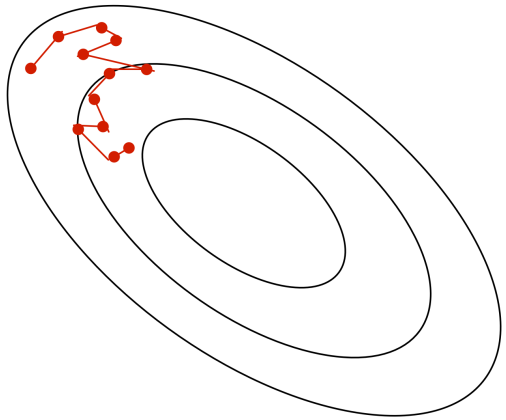
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - Too large: requires more compute; e.g., it takes more memory to store the activations, and longer to compute each gradient update
  - Too small: can't exploit vectorization, has high variance
  - A reasonable value might be $|\mathcal{M}| = 100$.

Stochastic Gradient Descent

- Batch gradient descent moves directly downhill (locally speaking).
- SGD takes steps in a noisy direction, but moves downhill on average.



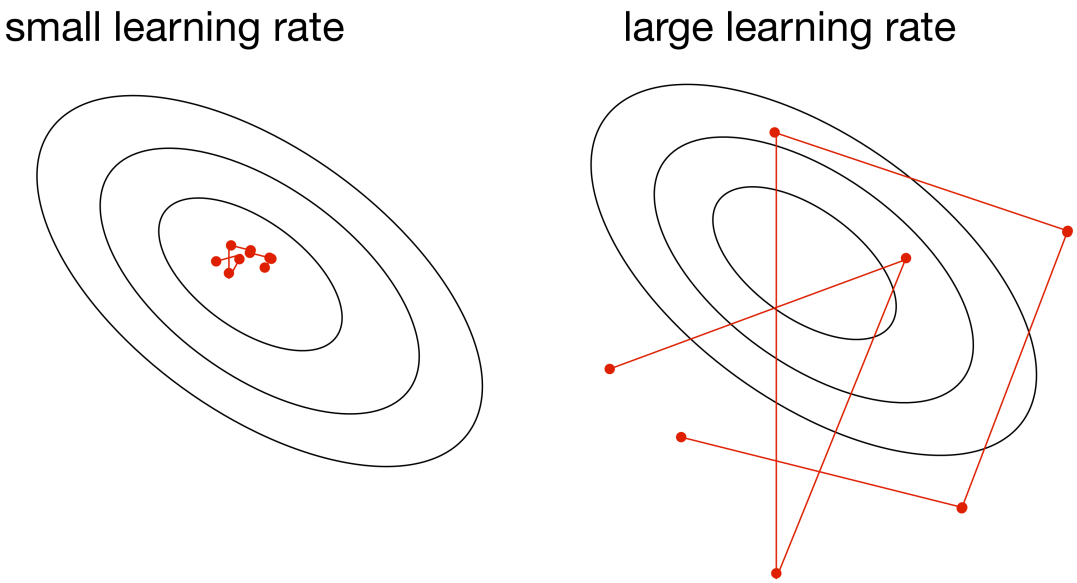
batch gradient descent



stochastic gradient descent

SGD Learning Rate

- In stochastic training, the learning rate also influences the amount of noise in the parameters resulting from the stochastic updates.



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

Binary Linear Classification

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 $r = 0$:

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

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

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

$$\text{need: } w_0 < 0$$

$$\text{need: } w_0 + w_2 < 0$$

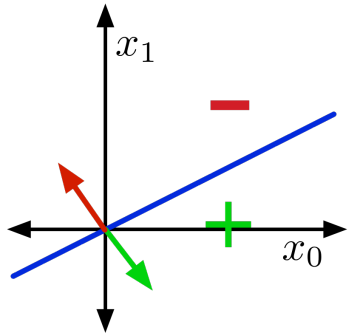
$$\text{need: } w_0 + w_1 < 0$$

$$\text{need: } w_0 + w_1 + w_2 \geq 0$$

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

The Geometric Picture

Input Space, or Data Space for NOT example

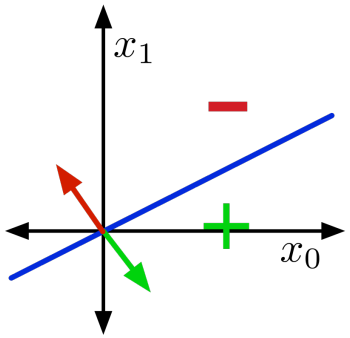


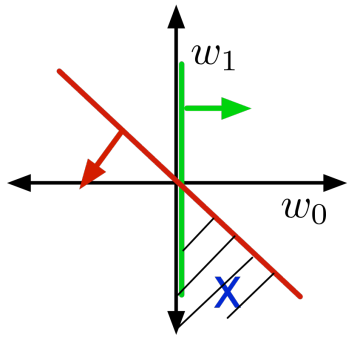
x_0	x_1	t
1	0	1
1	1	0

- 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.

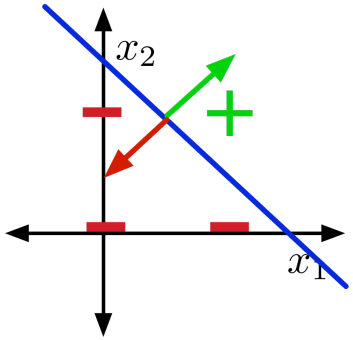
The Geometric Picture

- 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.

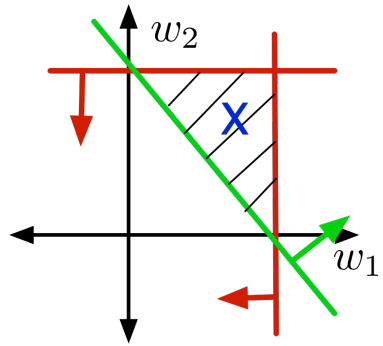
The Geometric Picture

Visualizations of the **AND** example

Data Space



Weight Space



- 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

- What if the dataset isn't linearly separable?
- Define a loss function, and minimize its average over the training set.
- Seemingly obvious loss function: 0-1 loss

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

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

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

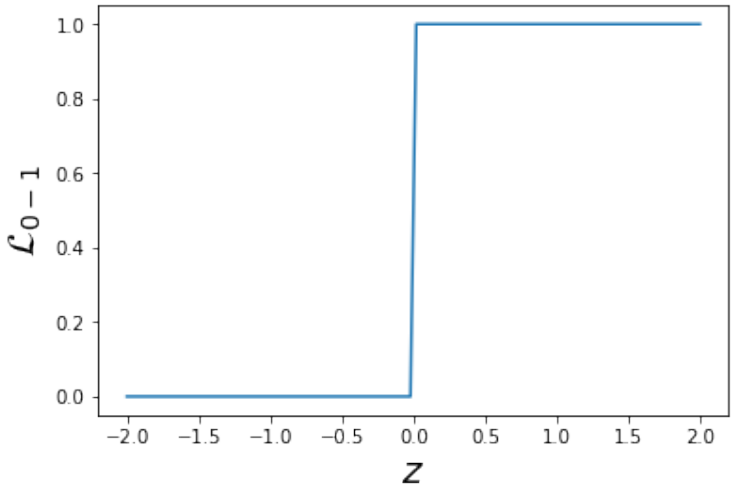
- Can you think of a problem with this approach?

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?

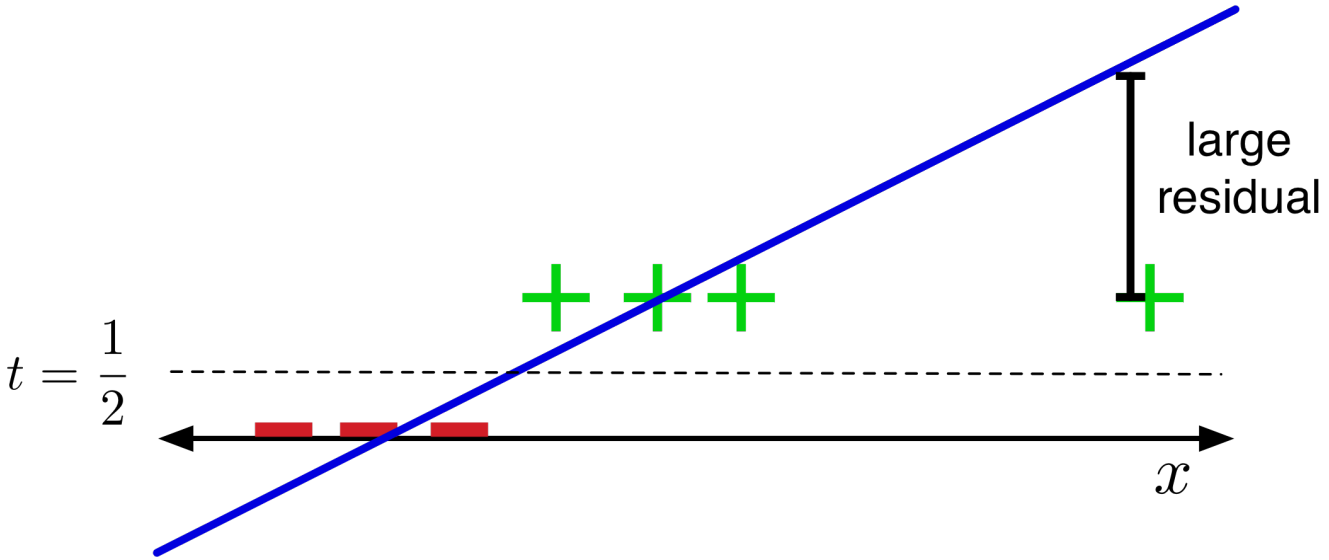
$$z = \mathbf{w}^\top \mathbf{x}$$

$$\mathcal{L}_{\text{SE}}(z, t) = \frac{1}{2}(z - t)^2$$

- 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:



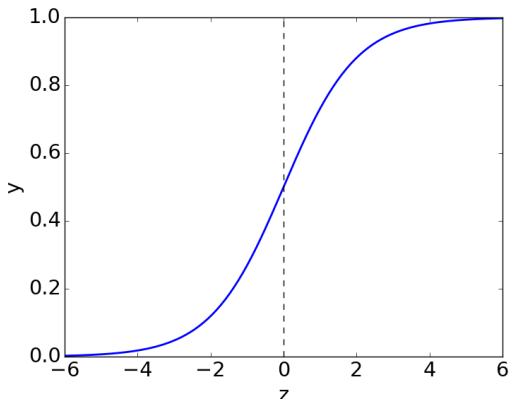
- The loss function hates when you make correct predictions with high confidence!
- If $t = 1$, it's more unhappy about $z = 10$ than $z = 0$.

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:

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



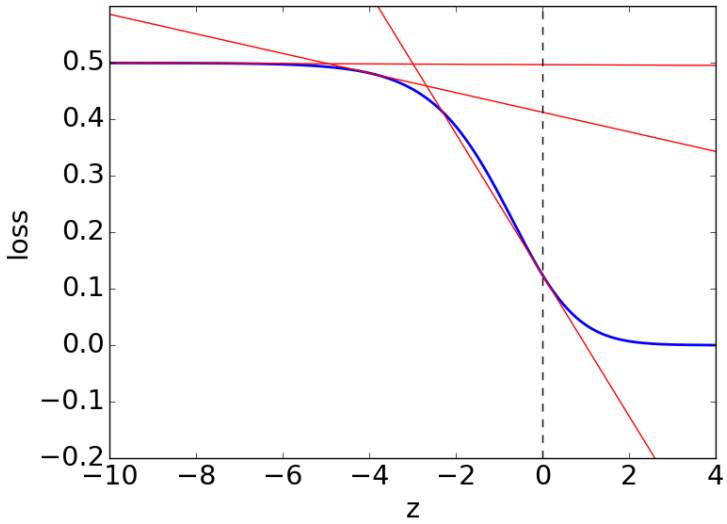
- $\sigma^{-1}(y) = \log(y/(1-y))$ is called the logit.
- A linear model with a logistic nonlinearity is known as log-linear:

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

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

Attempt 3: Logistic Activation Function

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



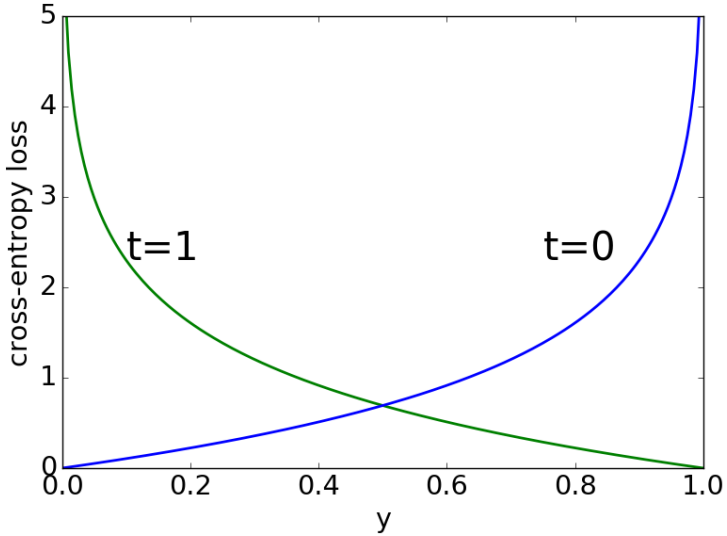
$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_j}$$

- 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

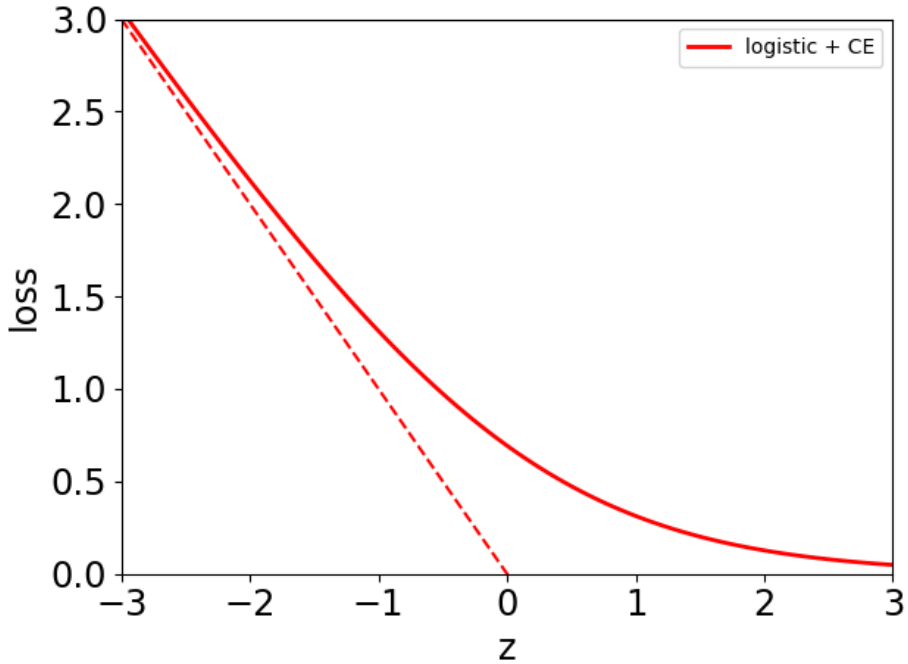
- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$. If $t = 0$, then we want to heavily penalize $y \approx 1$.
- The pundits who were 99% confident Clinton would win were much more wrong than the ones who were only 90% confident.
- Cross-entropy loss (aka log loss)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

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



Logistic Regression:

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



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.

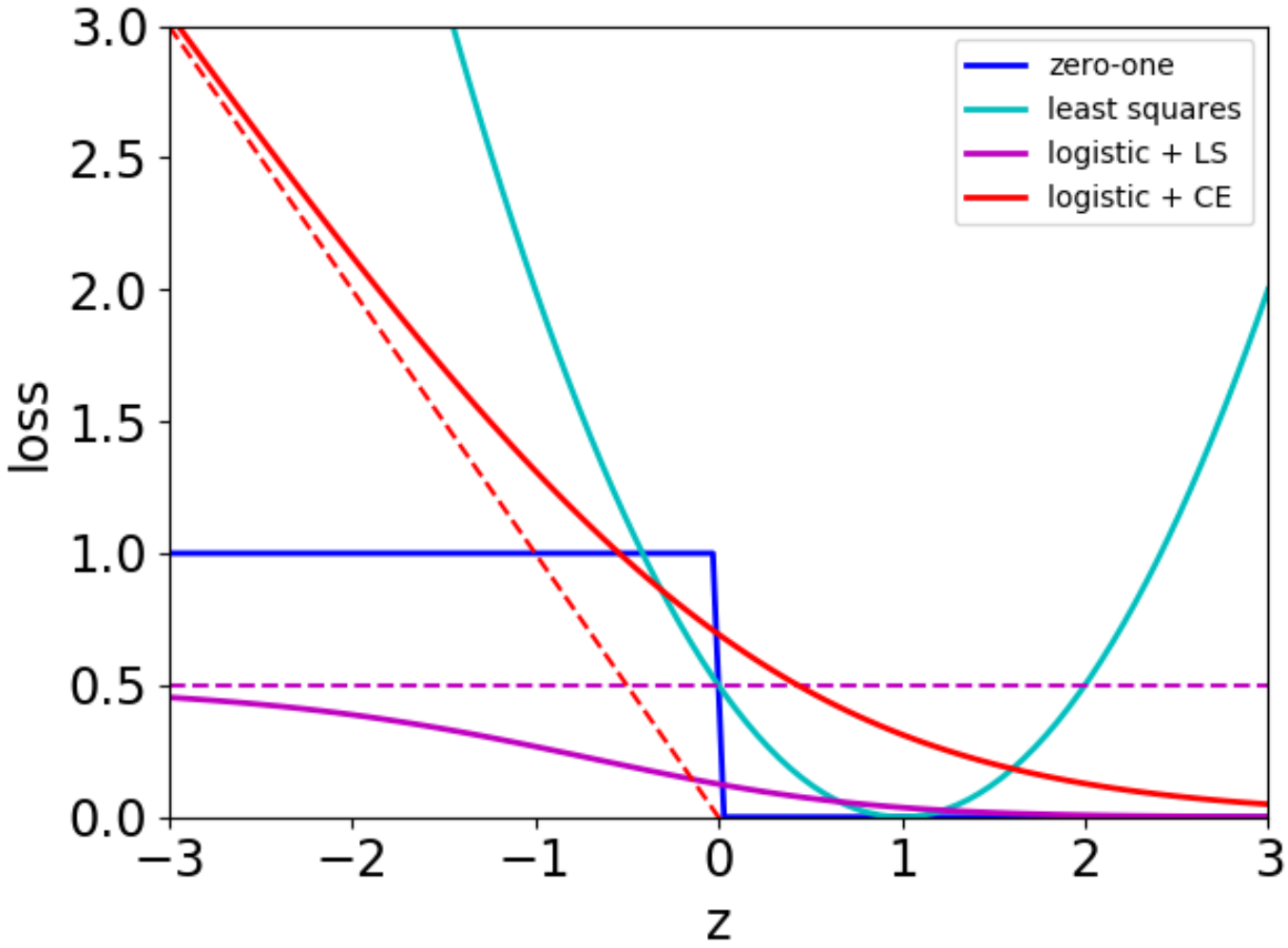
$$\mathcal{L}_{\text{LCE}}(z, t) = \mathcal{L}_{\text{CE}}(\sigma(z), t) = t \log(1 + e^{-z}) + (1 - t) \log(1 + e^{z})$$

- Numerically stable computation:

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

Logistic Regression

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:

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

Therefore

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

(verify this)

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

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

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.

Gradient Checking with Finite Differences

Gradient Checking

- We've derived a lot of gradients so far. How do we know if they're correct?
- Recall the definition of the partial derivative:

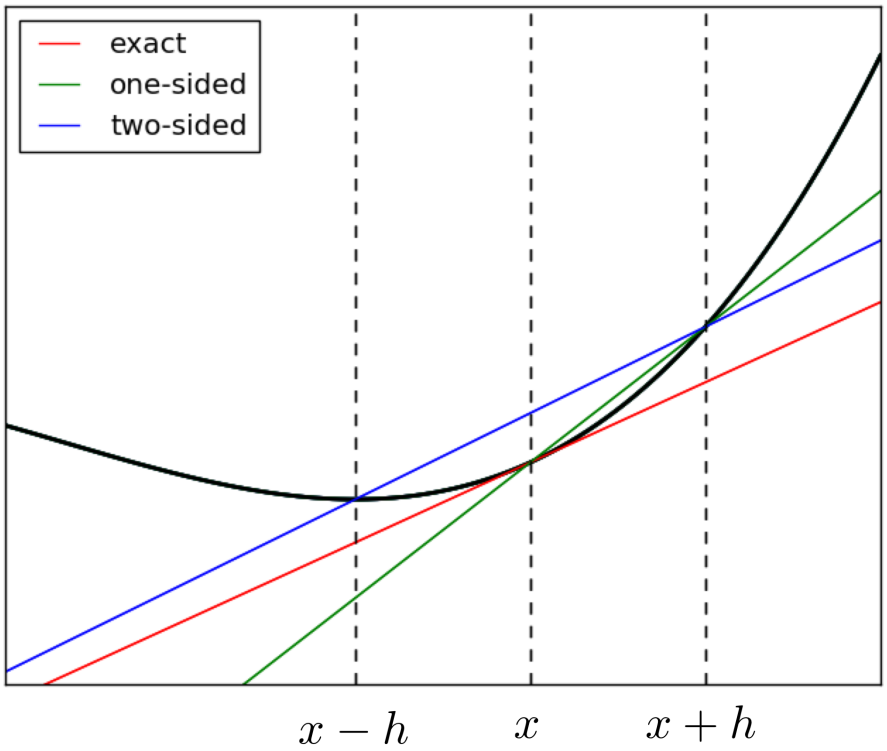
$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i, \ldots, x_N)}{h}$$

- Check your derivatives numerically by plugging in a small value of h, e.g. 10^{-10}. This is known as finite differences.

Gradient Checking

- Even better: the two-sided definition

$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_N) - f(x_1, \ldots, x_i - h, \ldots, x_N)}{2h}$$



Gradient Checking

- Run gradient checks on small, randomly chosen inputs
- Use double precision floats (not the default for most deep learning frameworks!)
- Compute the relative error:

$$\frac{|a-b|}{|a|+|b|}$$

  where a is the finite differences estimate and b is the derivative computed by the function you wrote.
- The relative error should be very small, e.g. 10^{-6}

Gradient Checking

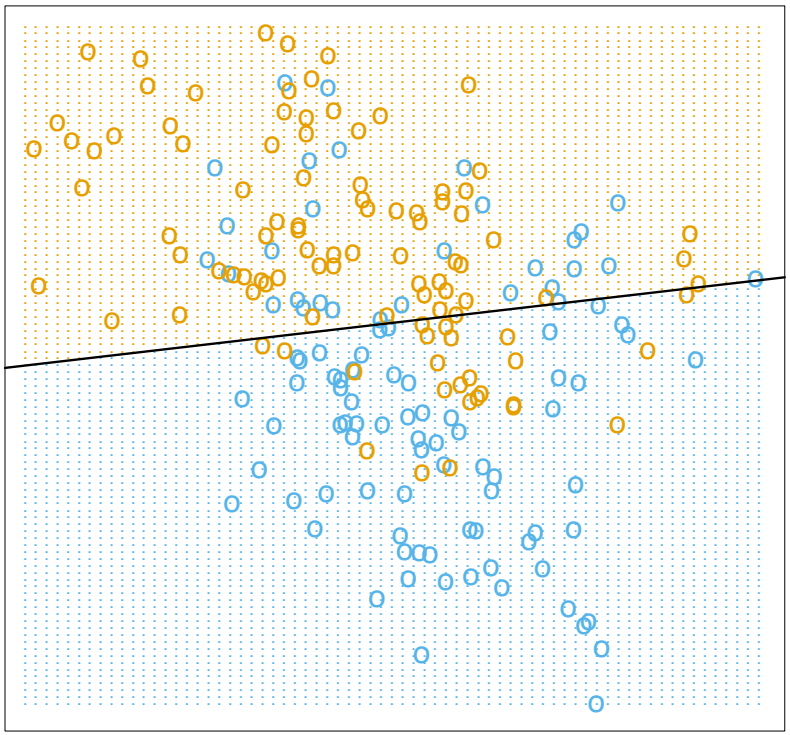
- Gradient checking is really important!
- Learning algorithms often appear to work even if the math is wrong.
- **But:**
  - They might work much better if the derivatives are correct.
  - Wrong derivatives might lead you on a wild goose chase.
- If you implement derivatives by hand, gradient checking is the single most important thing you need to do to get your algorithm to work well.

Linear Classifiers vs. KNN

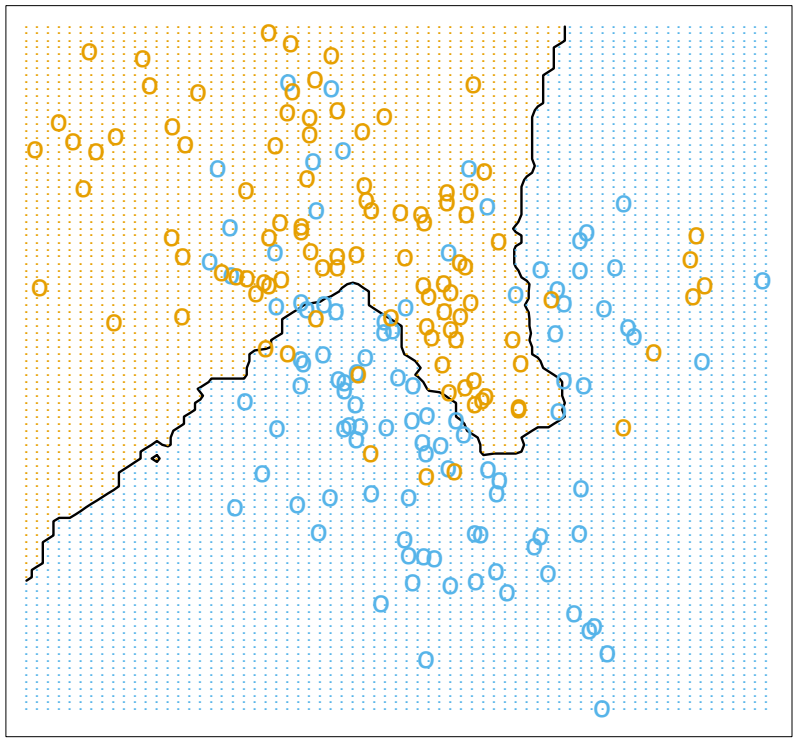
Linear Classifiers vs. KNN

Linear classifiers and KNN have very different decision boundaries:

Linear Classifier



K Nearest Neighbours



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, kernel density estimation
  - These models can sometimes be understood as having an infinite number of parameters

Conclusions

- Introduced logistic regression, a linear classification algorithm.
- Exemplified some recurring themes
  - Can define a surrogate loss function if the one we care about is intractable.
  - Think about whether a loss function penalizes certain mistakes too much or too little.
  - Can be useful to view the classfier's output as probabilities.
  - Learning algorithms can impose inductive biases (in this case, linearity), which can help or hurt depending on the problem.
- Next week: multiclass classification