CSC 311: Introduction to Machine Learning

Lecture 4 - Linear Models II

Sayyed Nezhadi

University of Toronto, Summer 2023

Outline

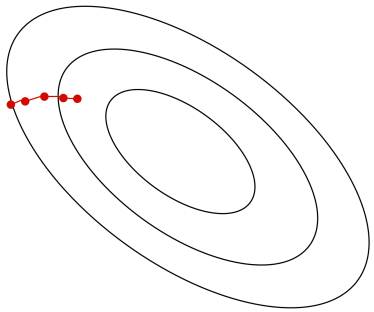
Overview

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

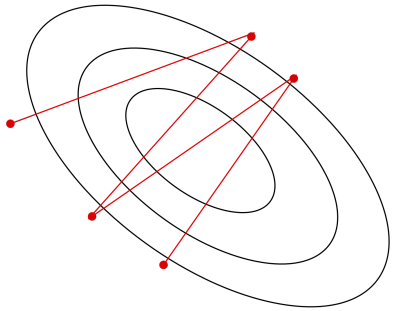
- Classification: predicting a discrete-valued target
  - Binary classification (this week): predicting a binary-valued target
  - Multiclass classification (next week): predicting a discrete-valued target

Impact of Learning Rate on Gradient Descent

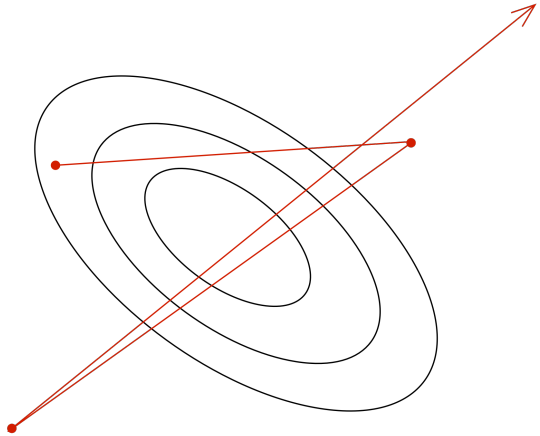
What could go wrong when setting the learning rate?



α too small:
slow progress



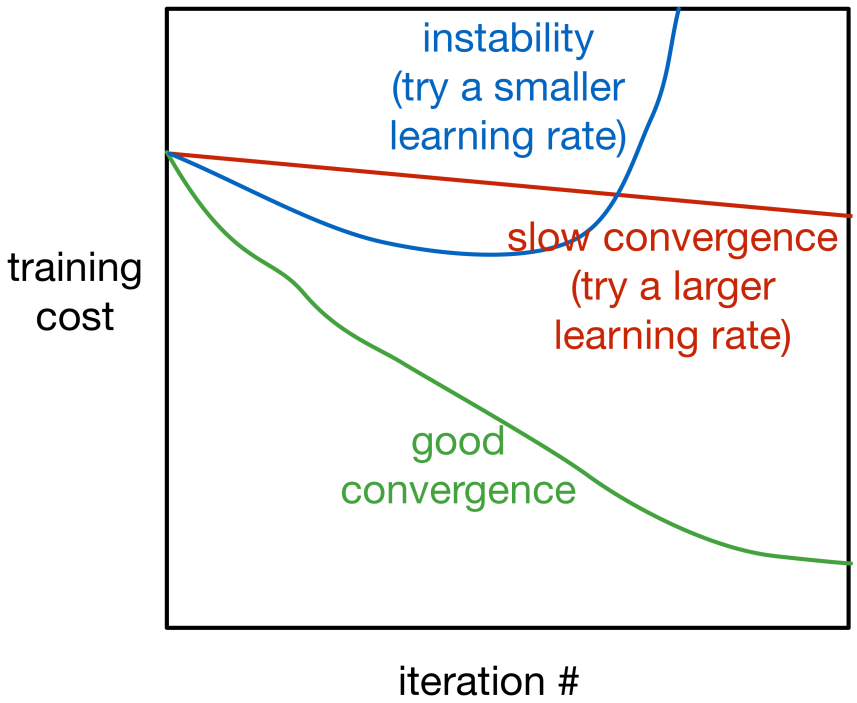
α too large:
oscillations



α much too large:
instability

Finding a Good Learning Rate

- Good values are typically between 0.001 and 0.1.
- Do a grid search for good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).
- Diagnose optimization problems using a training curve.



Main Takeaways on Learning Rate

How can the learning rate affect the behaviour of gradient descent?

How can we determine the best value for the learning rate?

Main Takeaways on Learning Rate

How can the learning rate affect the behaviour of gradient descent?

- Too small, slow convergence.
- Too large, oscillations, misses optimum, instability.

How can we determine the best value for the learning rate?

- Grid search
- Training curves can help diagnose optimization problems.

(Batch) Gradient Descent for a Large Data-set

Computing the gradient for a large data-set is computationally expensive!

Computing the gradient requires summing over all training examples since the cost function is the average loss over all the training examples.

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

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

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

Stochastic Gradient Descent

Updates the parameters based on the gradient for one training example

Repeat

(1) Choose example i uniformly at random,

(2) Perform update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$

Properties of Stochastic Gradient Descent

Benefits:

- Cost of each update is independent of N!
- Make significant progress before seeing all the data!
- Stochastic gradient is an unbiased estimate of the batch gradient given sampling each example uniformly at random.

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

Problems:

- High variance in the estimate
- Can't exploit efficient vectorized operations

A Compromise: Mini-Batch Gradient Descent

- Compute each gradient on a subset of examples.
- Mini-batch: a randomly chosen medium-sized subset of training examples $\mathcal{M}$.
- In theory, sample examples independently and uniformly with replacement.
- In practice, permute the training set and then go through it sequentially. Each pass over the data is called an epoch.

Tradeoff for Mini-Batch Gradient Descent

Trade-off for different mini-match sizes:

Large mini-batch size:

- more computation time
- estimates accurate
- can exploit vectorization

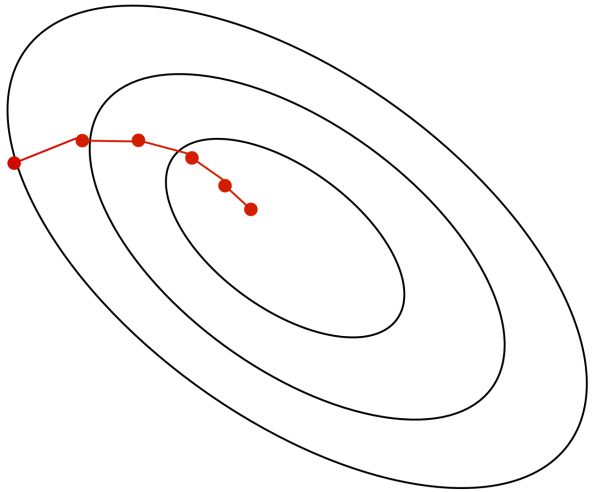
Small mini-batch size:

- faster updates
- estimates noisier
- cannot exploit vectorization

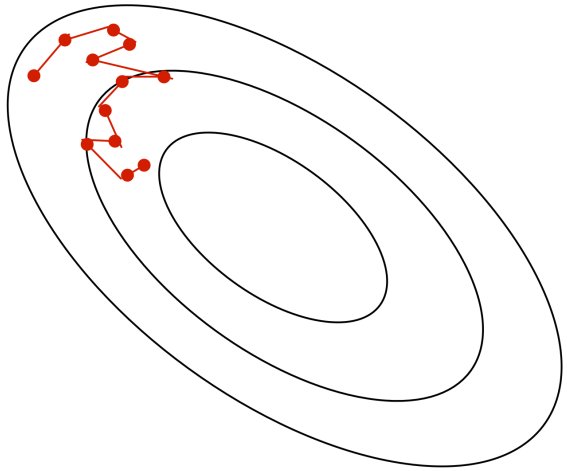
How should we set the mini-batch size $|\mathcal{M}|$?

- $|\mathcal{M}|$ is a hyper-parameter.
- A reasonable value might be $|\mathcal{M}| = 100$.

Visualizing Batch v.s. Stochastic Gradient Descent



Batch GD
moves downhill at each step.



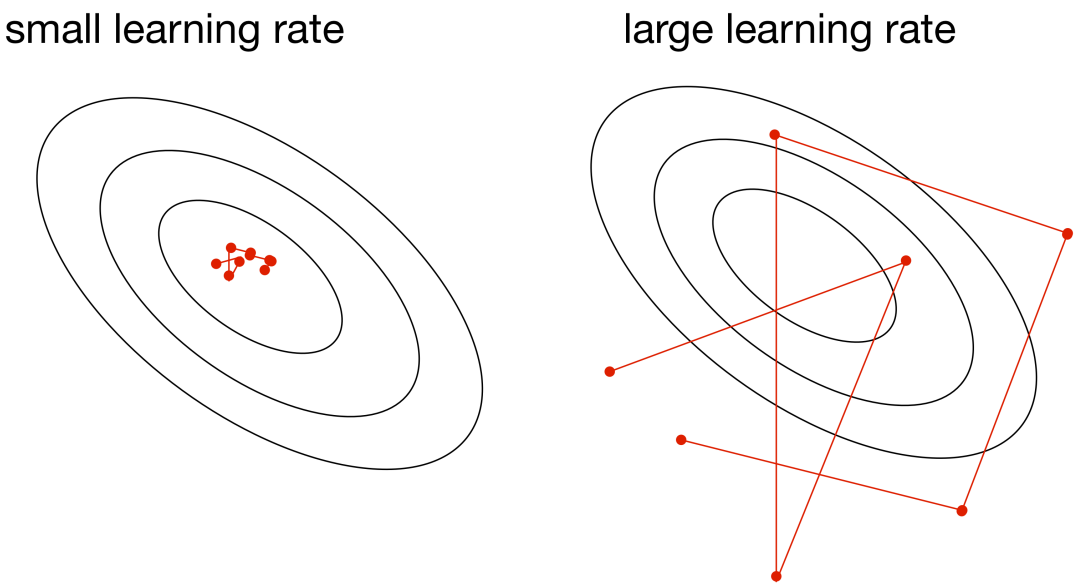
Stochastic GD
moves in a noisy direction,
but downhill on average.

Setting Learning Rate for Stochastic GD

The learning rate influences the noise in the parameters from the stochastic updates.

Typical strategy:

- Start with a large learning rate to get close to the optimum
- Gradually decrease the learning rate to reduce the fluctuations



Main Takeaways on Gradient Descent

Why is batch GD impractical for a large dataset?

How can we modify batch GD to work well with a large dataset?

What is stochastic GD?

What are the benefits and problems of using stochastic GD?

What is mini-batch GD?

What is the trade-off for different mini-batch sizes?

How should we set the learning rate for mini-batch GD?

Main Takeaways on Gradient Descent 1/2

Why is batch GD impractical for a large dataset?

- Computing gradient requires summing over all training examples.
- Too expensive for a large dataset.

How can we modify batch GD to work well with a large dataset?

- Perform update using a gradient for fewer examples.

What is stochastic GD?

- Perform update w/ gradient for one random example.

What are the benefits and problems of using stochastic GD?

- Benefits: faster, can make progress before seeing all data, approximates batch GD.
- Problems: high variance in estimates, can't exploit vectorization.

Main Takeaways on Gradient Descent 2/2

What is mini-batch GD?

- Perform update w/ gradient for a subset of examples.
- In practice, permute examples and go through subsets in sequence.

What is the trade-off for different mini-batch sizes?

- Large mini-batch size: large computational cost, more accurate estimates, and can exploit vectorization.
- Small mini-batch size: estimates more noisy, noisier estimates, and cannot exploit vectorization.

How should we set the learning rate for mini-batch GD?

- Start with a large learning rate to get close to the optimum.
- Gradually decrease the learning rate to get stable estimates.

Introducing Binary Linear Classification

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

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

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

Simplified Model

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

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

Eliminating the threshold r

- Assume that the threshold $r = 0$:

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

Eliminating the bias b

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

Modeling Simple Logical Functions

- Examples: NOT, AND.
- Goal is to minimize the training set error.
- Forget about generalizing to a test set for now.

Modeling NOT

NOT

x_0	x_1	t
1	0	1
1	1	0

$$z = w_0 x_0 + w_1 x_1$$

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

- Derive two sets of values for w_0, w_1 to classify NOT.
- Which conditions on w_0, w_1 guarantee perfect classification?

Modeling AND

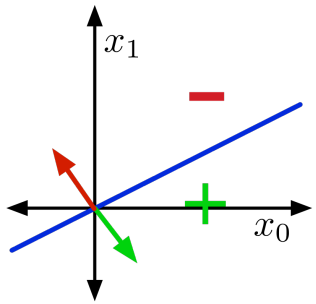
AND

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

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

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

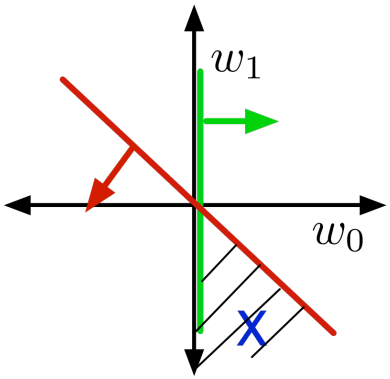
Visualizing NOT in Data Space



x_0	x_1	t
1	0	1
1	1	0

- Each training example is a point in data space.
- The line $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0\}$ defines the decision boundary.
  - A line in 2-D, or a hyper-plane in high dimensions.
  - The boundary passes through the origin (why?)
- Data is linearly separable if a linear decision rule can perfectly separate the training examples.

Visualizing NOT in Weight Space



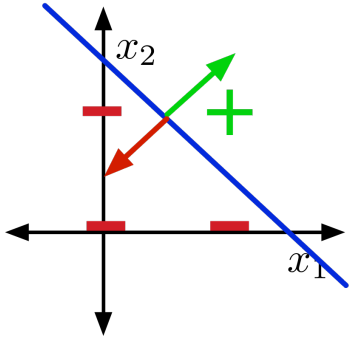
$$w_0 \geq 0$$
$$w_0 + w_1 < 0$$

- Each point is a set of values for the weights $\mathbf{w}$.
- Each training example $\mathbf{x}$ specifies a half-space that $\mathbf{w}$ must lie in to guarantee correct classification.
- The feasible region satisfies all the constraints.
  The problem is feasible if the feasible region is nonempty.

Visualizing AND in Data and Weight Spaces

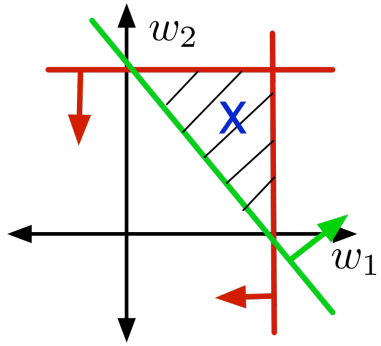
Let's look at a 2-D slice of the 3-D data and weight spaces for AND.

Data Space



- Slice for $x_0 = 1$
- Example Solution:
  $w_0 = -1.5,\ w_1 = 1,\ w_2 = 1$
- Decision Boundary:
  $w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$
  $\implies -1.5 + x_1 + x_2 = 0$

Weight Space



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

Learning the Weights for Linearly Separable Data

Binary Linear Classification

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

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

If data is linearly separable, we can learn the weights

- using linear programming, or
- using the perceptron algorithm (but primarily of historical interest).

Unfortunately, in real life, data is almost never linearly separable.

Main Takeaways on Binary Linear Classification

What is binary linear classification (BLC)?

Describe the simplified model for BLC.

How can we model simple logical functions using BLC?

When visualizing BLC in data or weight space,
what do the points, the lines, and the half-spaces represent?

How can we learn the weights for a linearly separable data-set?

Main Takeaways on Binary Linear Classification 1/2

What is binary linear classification (BLC)?

- Predict a binary target by using a linear function of inputs followed by thresholding.

Describe the simplified model for BLC.

$$z = w^T x, y = 1 \text{ if } z \geq 0, y = 0 \text{ otherwise.}$$

How can we model simple logical functions using BLC?

- Derive weights to represent NOT, AND.

Main Takeaways on Binary Linear Classification 2/2

When visualizing BLC in data or weight space,
what do the points, the lines, and the half-spaces represent?

- In data space: points are training examples. a line represents a set of weights.
- In weight space: points are weights. a half-space represents a training example constraining the weights.

How can we learn the weights for a linearly separable data-set?

- linear programming.
- perceptron algorithm.

Data Isn't Linearly Separable

What if the data-set isn't linearly separable?

- Define a loss function.
- Find weights that minimize the average loss over the training examples.

First Try: 0-1 Loss

Binary linear classification with 0-1 loss:

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

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

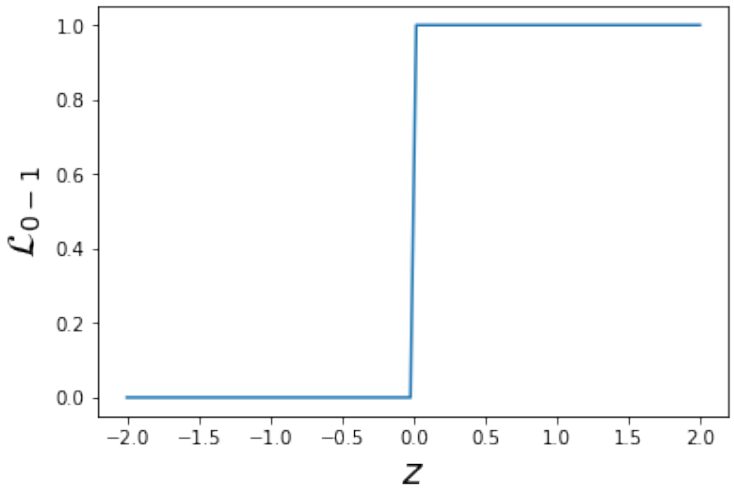
$$\mathcal{L}_{0-1}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases} = \mathbb{I}[y \neq t]$$

The cost $\mathcal{J}$ is the misclassification rate:

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

Problems with 0-1 loss

- To minimize the cost, we need to find a critical point.
- But, the gradient is zero almost everywhere!
  Changing the weights has no effect on the loss.
- Also, 0-1 loss is discontinuous at $z = 0$,
  where the gradient is undefined.



Second Try: Squared Loss for Linear Regression

- Choose an easier to optimize loss function.
- How about the squared loss for linear regression?

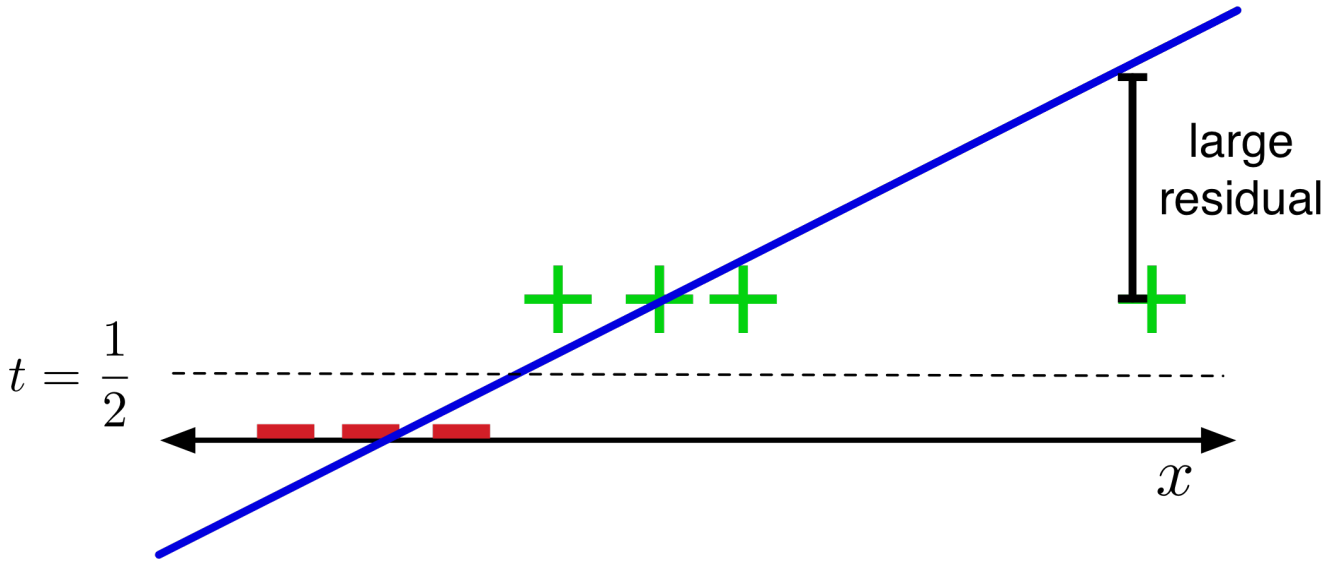
$$z = \mathbf{w}^\top \mathbf{x}$$

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

- Treat the binary targets as continuous values.
- Make final predictions y by thresholding z at $\frac{1}{2}$.

Problems with Squared Loss

- If $t = 1$, a greater loss for $z = 10$ than $z = 0$.
- Making a correct prediction with high confidence should be good, but incurs a large loss.

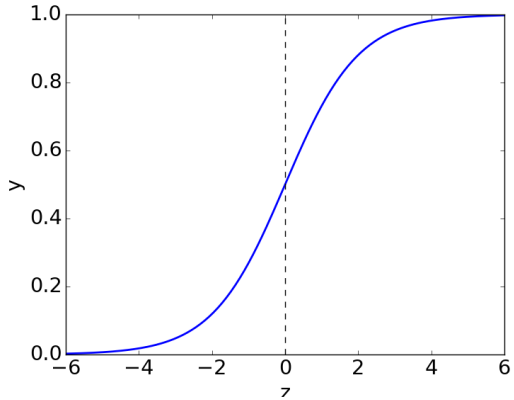


Third Try: Logistic Activation Function

For binary targets, no reason to predict values outside $[0, 1]$.
Let's squash predictions y into $[0, 1]$.

The logistic function is a sigmoid (S-shaped) function.

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



This results in a linear model with a logistic non-linearity:

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

σ is called an activation function.

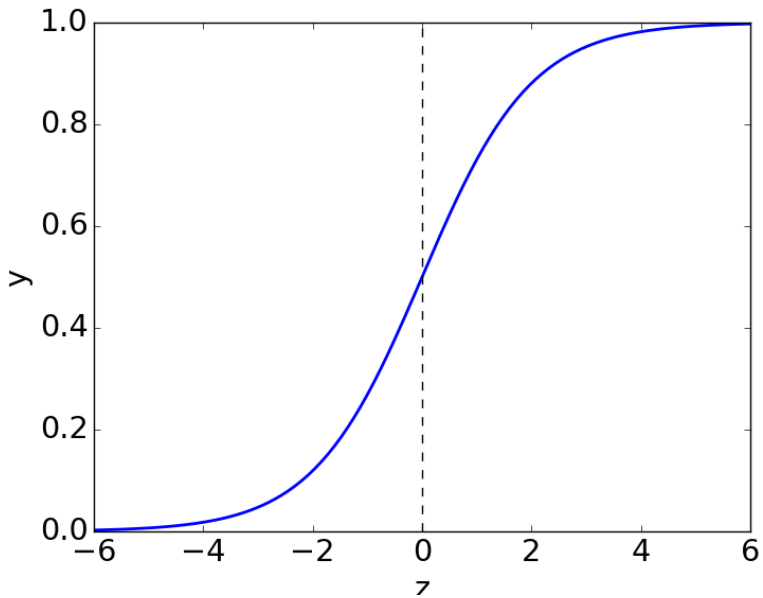
Problems with Logistic Activation Function

Suppose that $t = 1$ and z is very negative ($z \ll 0$).

Then, the prediction $y \approx 0$ is really wrong.

However, the weights appears to be at a critical point:

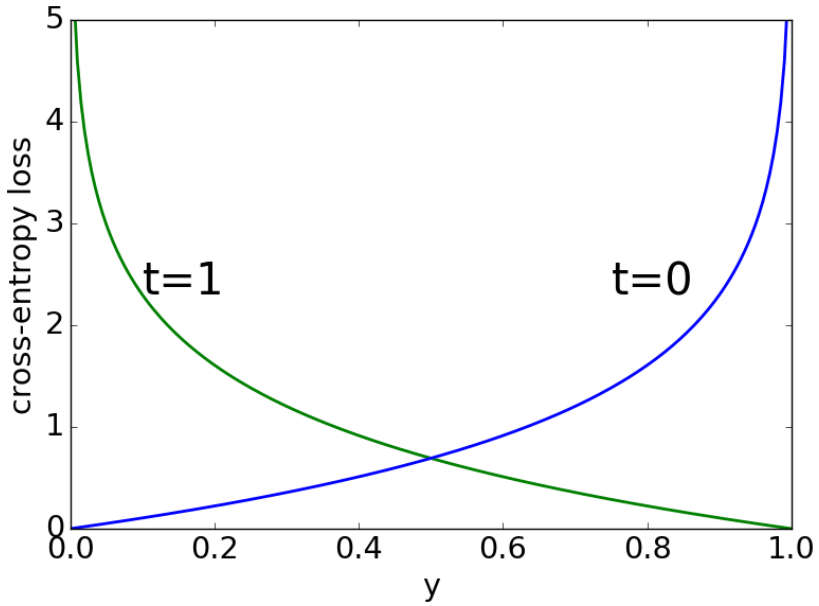
$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_j} \approx 0\frac{\partial z}{\partial w_j} = 0$$



Final Try: Cross-Entropy Loss

- Interpret $y \in [0, 1]$ as the estimated probability that $t = 1$.
- Heavily penalize the extreme mis-classification cases when $t = 0, y = 1$ or $t = 1, y = 0$.
- Cross-entropy loss (a.k.a. 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

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

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

$$\mathcal{L}_{\text{CE}} = -t \log y - (1 - t) \log(1 - y)$$

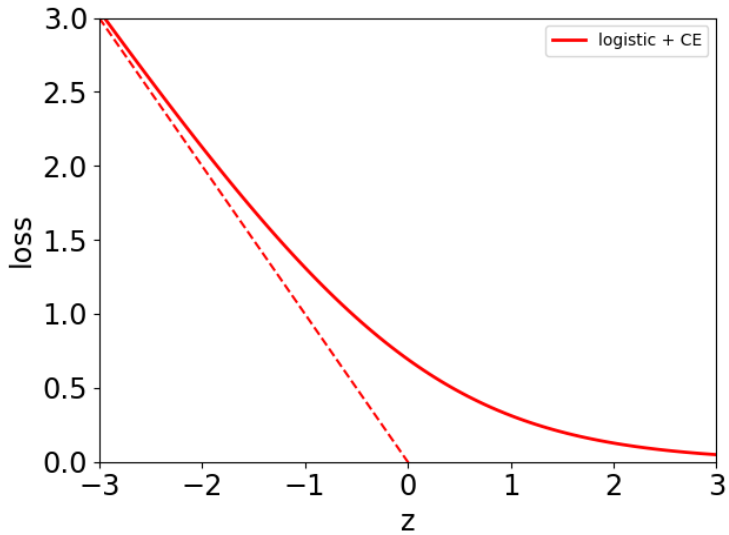


Figure: Cross-Entropy Loss w.r.t z, assuming $t = 1$

Numerical Instabilities

- Implementing logistic regression naively can cause numerical instabilities.
- Suppose that $t = 1$ and $z \ll 0$.
- If y is small enough, it may be numerically zero.
  This can cause very subtle and hard-to-find bugs.

$$z \ll 0 \Rightarrow y = \sigma(z) \approx 0$$
$$\mathcal{L}_{\text{CE}} = -t \log y - (1 - t) \log(1 - y) \approx -1 \log 0$$

Numerically Stable Version

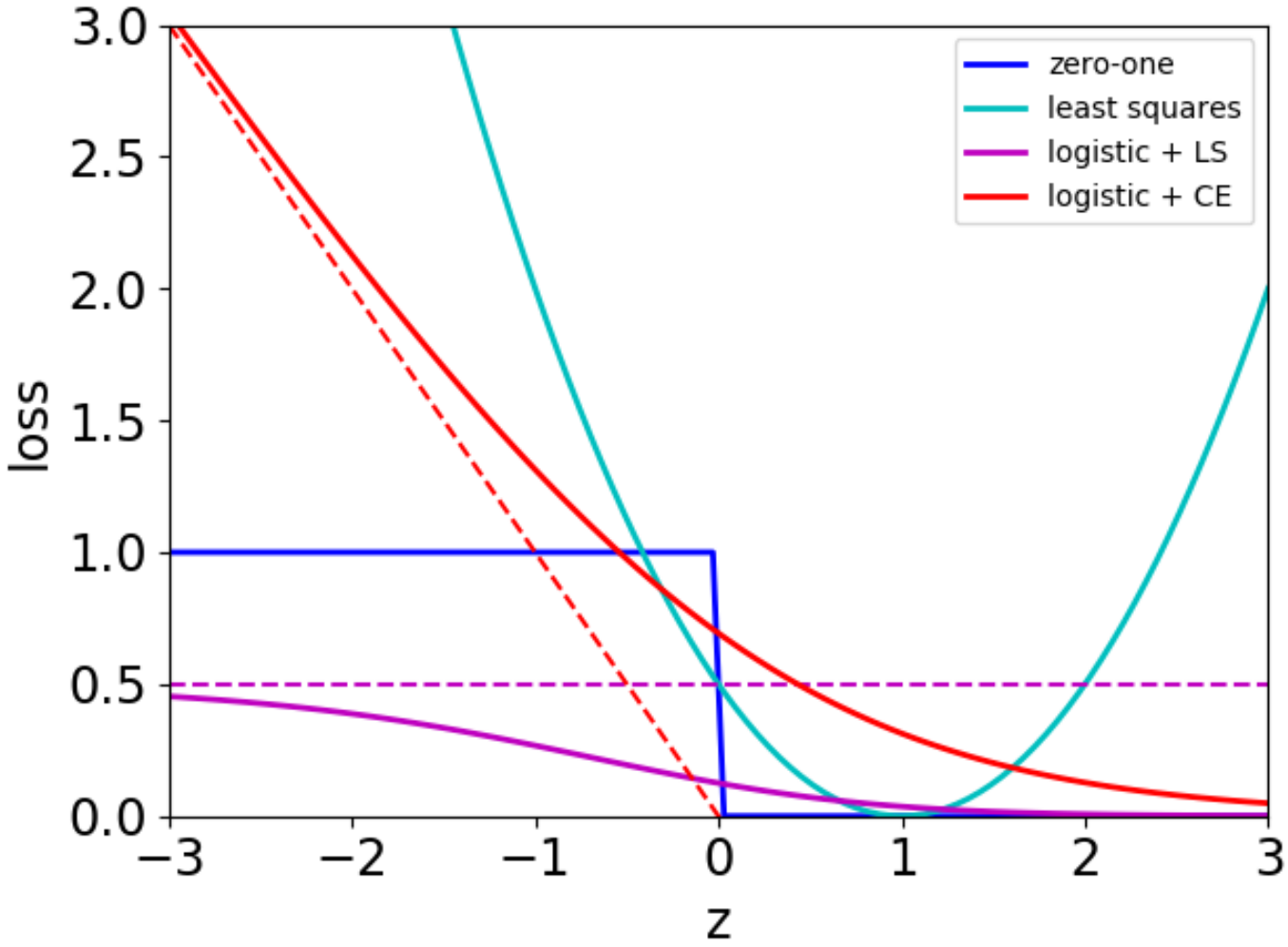
- Instead, we combine the logistic activation function and the cross-entropy loss into a single logistic-cross-entropy function.

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

- Numerically stable computation:

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

Comparing Loss Functions for t = 1



Gradient Descent for Logistic Regression

- How do we minimize the cost $\mathcal{J}$ for logistic regression? Unfortunately, no direct solution.

- Use gradient descent
  since the logistic loss is a convex function in $\mathbf{w}$.
  - 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

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

Gradient descent update for logistic regression:

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

Comparing 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:

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

They are both examples of generalized linear models.

Main Takeaways on Logistic Regression

What is the main motivation for using logistic regression?

Why did we try 0-1 loss first? What's the problem with it?

Why did we try squared loss next? What's the problem with it?

Why did we try logistic activation function next? What's the problem with it?

Why did we try cross-entropy loss next?

How do we apply gradient descent to logistic regression?

Main Takeaways on Logistic Regression 1/2

What is the main motivation for using logistic regression?

- When data isn't linearly separable, cannot classify data perfectly.
- Use a loss function and minimize average loss.

Why did we try 0-1 loss first? What's the problem with it?

- Natural choice for classification.
- Gradient zero almost everywhere. a discontinuity.

Why did we try squared loss next? What's the problem with it?

- Easier to optimize.
- Large penalty for a correct prediction with high confidence.

Main Takeaways on Logistic Regression 2/2

Why did we try logistic activation function next? What's the problem with it?

- Prediction $\in [0, 1]$.
- An extreme mis-classification case appears optimal.

Why did we try cross-entropy loss next?

- Heavily penalizes extreme mis-classification.

How do we apply gradient descent to logistic regression?

- Derive the update rule.

Gradient Checking

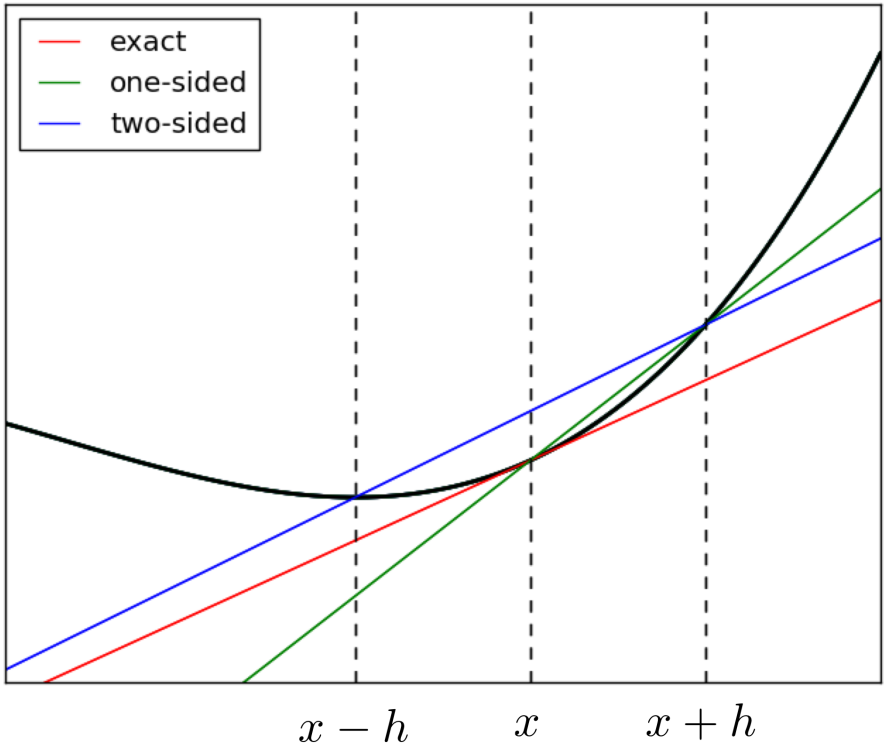
- If we implement derivatives by hand,
  how do we know if the implementation is correct?
- We can perform gradient checking using finite differences.
- The one-sided definition of the partial derivative:

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

- The two-sided definition of the partial derivative:

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

Definitions of Partial Derivatives



Gradient Checking

- Run gradient checks on small, randomly chosen inputs
- Use double precision floats
  (not the default for most deep learning frameworks!)
- Plug in a small value of h, e.g. 10^{-10}.
- Compute the relative error:

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

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

Gradient Checking

- If you implement derivatives by hand,
  gradient checking is the most important thing to do!
- 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.

Main Takeaways on Gradient Checking

Why is gradient checking important?

How can we perform gradient checking w/ finite differences?

Main Takeaways on Gradient Checking

Why is gradient checking important?

- Algorithms often appear to work even if the math is wrong.
- But they can work better if the derivatives are correct.

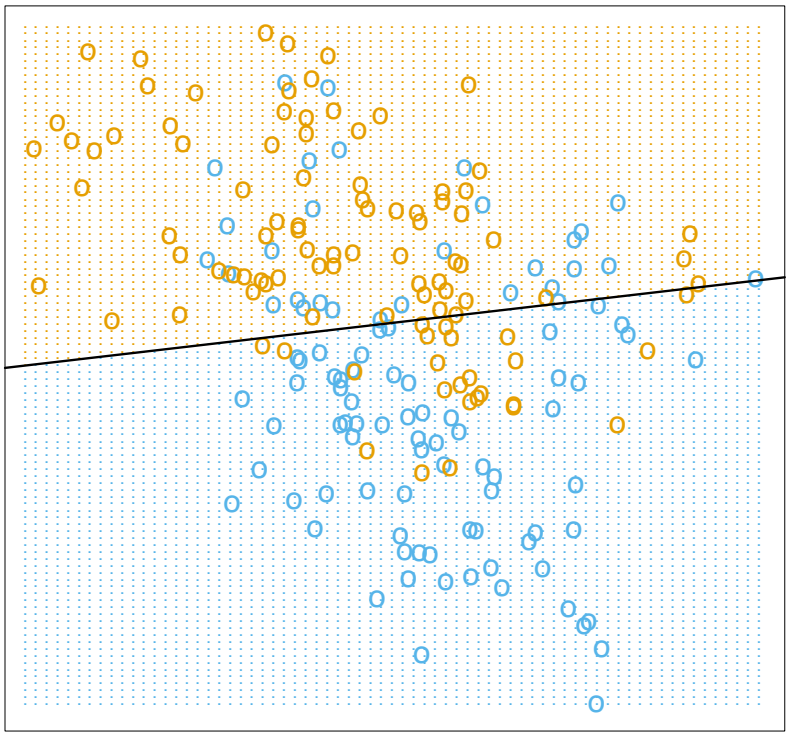
How can we perform gradient checking w/ finite differences?

- Estimate the gradient using finite differences.
- Make sure the relative error is small.

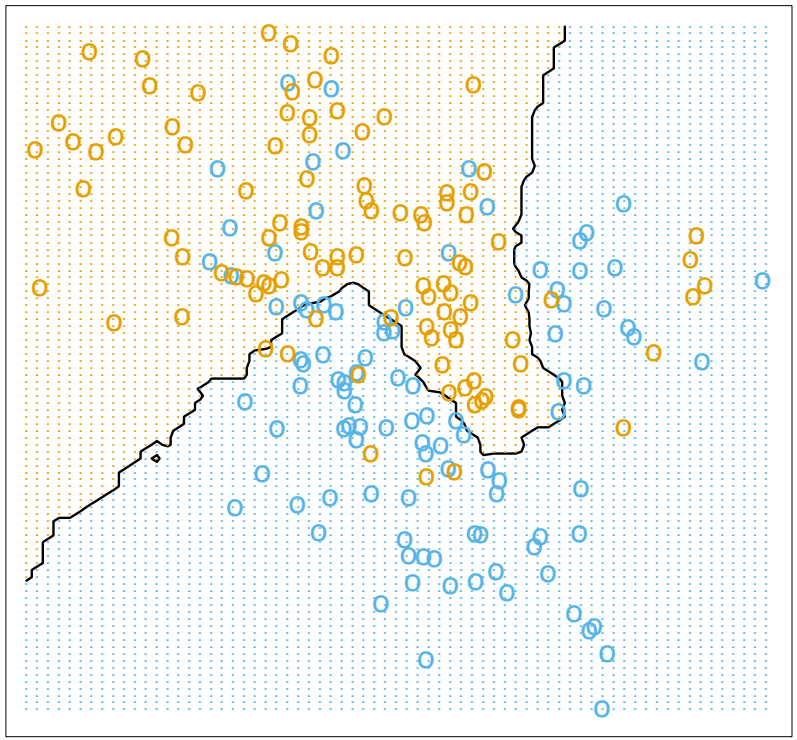
Linear Classifiers vs. KNN

Linear classifiers and KNN have very different decision boundaries:

Linear Classifier



K Nearest Neighbours



Linear Classifiers vs. KNN

Linear Classifiers

K Nearest Neighbours

Hypothesis Space and Inductive Bias

- A hypothesis is a function $f : \mathcal{X} \to \mathcal{T}$ from the input to target space.
- The hypothesis space $\mathcal{H}$ a set of hypotheses.
  - Linear regression, $\mathcal{H}$ is the set of linear functions.
- A machine learning algorithm aims to find a good hypothesis $f \in \mathcal{H}$.
- An algorithm's inductive bias is the members of $\mathcal{H}$ and its preference for some hypotheses of $\mathcal{H}$ over others.
  - General natural patterns or domain knowledge that help our algorithms to generalize; e.g., linearity, continuity, simplicity ...
- No Free Lunch theorem: if datasets/problems were not naturally biased, no machine learning algorithm would be better than another.

Parametric v.s. Non-Parametric Algorithms

- A parametric algorithm:
  the hypothesis space $\mathcal{H}$ is defined using a finite set of parameters.
  - Examples: linear regression, logistic regression.
  - Other examples: neural networks, k-means and Gaussian mixture models.

- A non-parametric algorithm:
  the hypothesis space $\mathcal{H}$ is defined in terms of the data.
  - Examples: k-nearest neighbors, decision trees.
  - Other examples: Gaussian processes, kernel density estimation

Main Takeaways on Basic Concepts

Compare and contrast KNN and Linear Classifiers.

What is a hypothesis, the hypothesis space, and the inductive bias?

Define parametric and non-parametric algorithms. Give examples.

Main Takeaways on Basic Concepts

Compare and contrast KNN and Linear Classifiers.

What is a hypothesis, the hypothesis space, and the inductive bias?

- A hypothesis is a function from the input space to the target space.
- A hypothesis space contains a collection of hypotheses.
- The inductive bias is the hypothesis space and our preference for some hypotheses over others.

Define parametric and non-parametric algorithms. Give examples.

- A parametric algorithm has a finite number of parameters.
  Examples: linear regression, logistic regression.
- A non-parametric algorithm has no parameter is defined in terms of the data.
  Examples: k-nearest-neighbours, decision trees.

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