DSC 140A

Probabilistic Modeling & Machine Learning

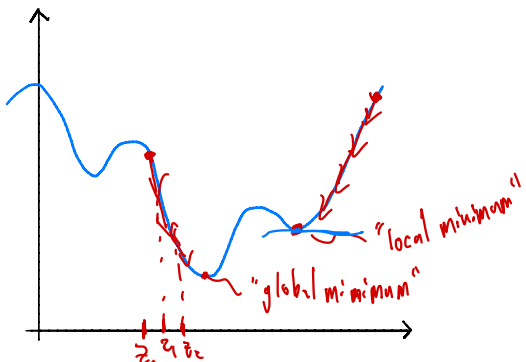
Lecture 6 | Part 1

Convexity

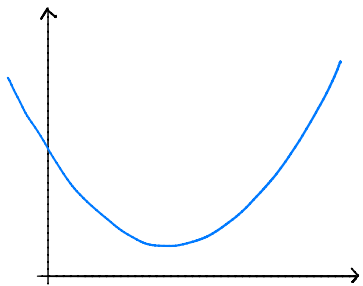
Question

- When is gradient descent guaranteed to work?

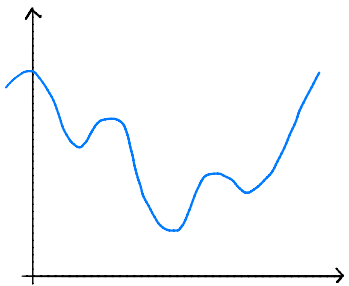
Not here...



Convex Functions



Convex



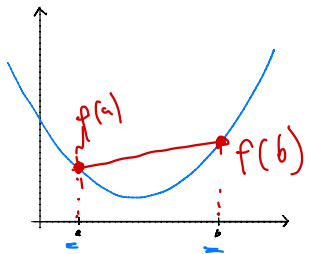
Non-convex

Convexity: Definition

- f is **convex** if for **every** a, b the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of f.

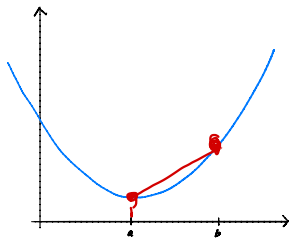


Convexity: Definition

- f is **convex** if for **every** a, b the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of f.

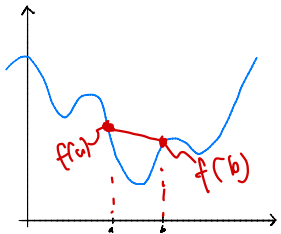


Convexity: Definition

- f is **convex** if for **every** a, b the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of f.

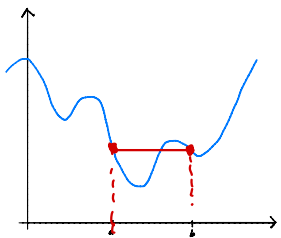


Convexity: Definition

- f is **convex** if for **every** a, b the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

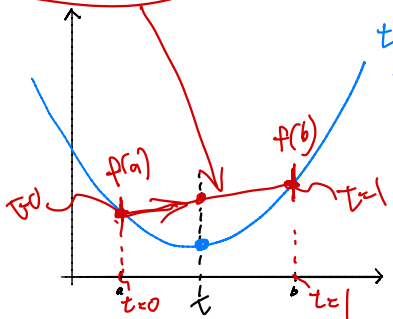
does not go below the plot of f.



Convexity: Formal Definition

- A function $f : \mathbb{R} \to \mathbb{R}$ is **convex** if for every choice of $a, b \in \mathbb{R}$ and $t \in [0, 1]$:

$$(1 - t)f(a) + tf(b) \geq f((1 - t)a + tb).$$



Another View: Second Derivatives

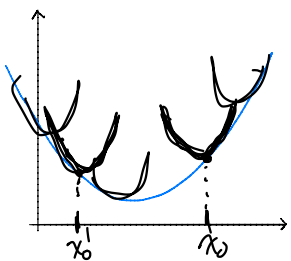
- If $\frac{d^2 f}{dx^2}(x) \geq 0$ for all x, then f is convex.
- Example: $f(x) = x^4$ is convex.
- **Warning!** Only works if f is twice differentiable!

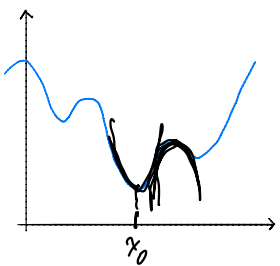
Another View: Second Derivatives

- "Best" straight line at x_0:
  - $f_1(x) = f(x_0) + f'(x_0) \cdot (x - x_0)$
- "Best" parabola at x_0:
  - $f_2(x) = f(x_0) + f'(x_0) \cdot (x - x_0) + \frac{1}{2} f''(x_0) \cdot (x - x_0)^2$
  - Possibilities: upward-facing, downward-facing, flat.

Convexity and Parabolas

- Convex if for **every** x_0, parabola is upward-facing (or flat).
  - That is, $f''(x_0) \geq 0$.





Careful!

- A function can be convex without having a second derivative.

- Example: $f(x) = |x|$ is convex.
  - But can't use the second derivative test to show it.

Proving Convexity Using Properties

Suppose that $f(x)$ and $g(x)$ are convex. Then:

1. $w_1 f(x) + w_2 g(x)$ is convex, provided $w_1, w_2 \geq 0$
   - Example: $3x^2 + |x|$ is convex

2. $g(f(x))$ is convex, provided g is non-decreasing.
   - Example: e^{x^2} is convex

3. $\max\{f(x), g(x)\}$ is convex
   - Example: $\begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$ is convex

Note!

- These properties are useful for proving convexity for functions of **one variable**.
- Some of them will not generalize to higher dimensions.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 2

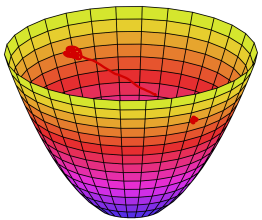
Convexity in Many Dimensions

Convexity: Definition

- $f(\vec{x})$ is **convex** if for **every** $\vec{a}, \vec{b}$ the line segment between

$$(\vec{a}, f(\vec{a})) \qquad \text{and} \qquad (\vec{b}, f(\vec{b}))$$

does not go below the plot of f.



Convexity: Formal Definition

"Mapping from $\mathbb{R}^d$ to $\mathbb{R}$"

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if for every choice of $\vec{a}, \vec{b} \in \mathbb{R}^d$ and $t \in [0, 1]$:

vectors

scalar

$$(1 - t)f(\vec{a}) + tf(\vec{b}) \geq f((1 - t)\vec{a} + t\vec{b}).$$

line segment

function

interpolating btwn $\vec{a}$ and $\vec{b}$

$t=0 \Rightarrow \vec{a}$

$t=1 \Rightarrow \vec{b}$

Checking for Convexity

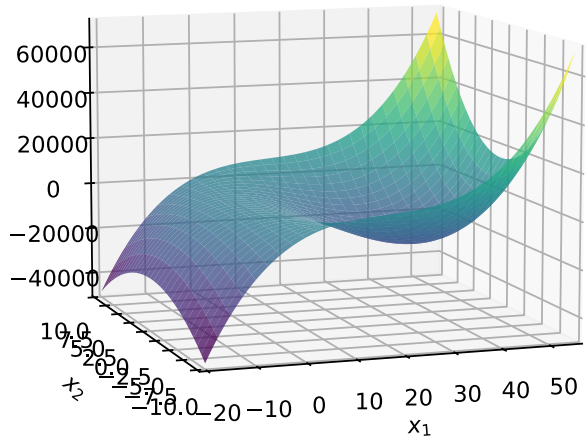
- We can usually go back to the definition to check if a function is convex.
- Example: see discussion.
- Typically, though, there are **easier** ways to check.

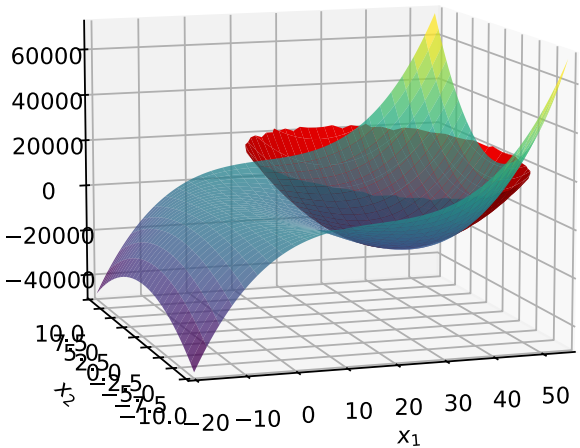
The Second Derivative Test

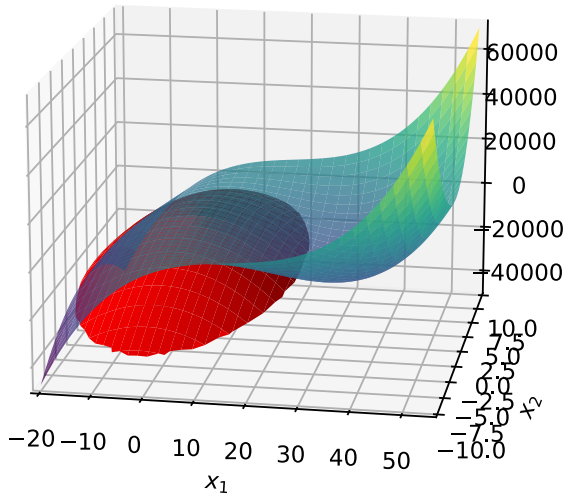
- For 1-dimensions functions:
  - convex if second derivative ≥ 0.
- For *d*-dimensional functions:
  - convex if ???

Second Derivatives in d-Dimensions

- In 2-dimensions, there are 4 second derivatives:
  - $\frac{\partial f^2}{\partial x_1^2}, \frac{\partial f^2}{\partial x_2^2}, \frac{\partial f^2}{\partial x_1 x_2}, \frac{\partial f^2}{\partial x_2 x_1}$
- In d-dimensions, there are d^2:
  - $\frac{\partial f^2}{\partial x_i \partial x_j}$ for all i, j.
- The second derivatives describe the local curvature(i.e. second order approximation) of f.
  - **Convex** if the approximation is always an upward-facing paraboloid or flat.


60000
40000
20000
0
−20000
−40000
−20
−10
0
10
20
30
40
50
x_1
x_2


60000
40000
20000
0
−20000
−40000
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
x_2
−20
−10
0
10
20
30
40
50
x_1


60000
40000
20000
0
−20000
−40000
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
x_2
−20
−10
0
10
20
30
40
50
x_1

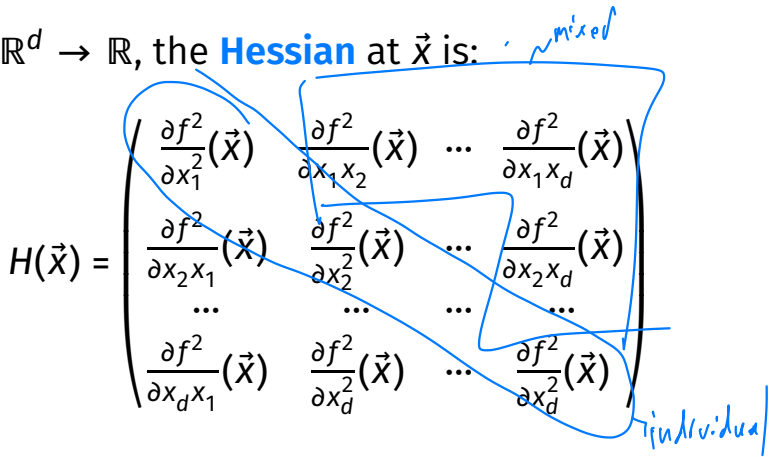
The Hessian Matrix

- In 2-dimensions, there are 4 second derivatives:
  - $\frac{\partial f^2}{\partial x_1^2}, \frac{\partial f^2}{\partial x_2^2}, \frac{\partial f^2}{\partial x_1 x_2}, \frac{\partial f^2}{\partial x_2 x_1}$
- Collect them all into the **Hessian** matrix:
- For $f : \mathbb{R}^2 \to \mathbb{R}$:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \end{pmatrix}$$

In General

- If $f : \mathbb{R}^d \to \mathbb{R}$, the **Hessian** at $\vec{x}$ is:



$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_1 x_d}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_2 x_d}(\vec{x}) \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f^2}{\partial x_d x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) \end{pmatrix}$$

Second Derivative Test

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if for any $\vec{x} \in \mathbb{R}^d$, all **eigenvalues** of the Hessian matrix $H(\vec{x})$ are ≥ 0.

$H(\vec{x})$ is positive semi-definite

For This Class...

- You will not need to compute eigenvalues “by hand”...
- Unless the matrix is diagonal.
  - In which case, the eigenvalues are the diagonal entries.

Example

- The eigenvalues of this matrix are 5, 2, and 1.

$$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Exercise

Is $f(x, y) = e^x + e^y + x^2 - y^2$ convex?

eigenvalues

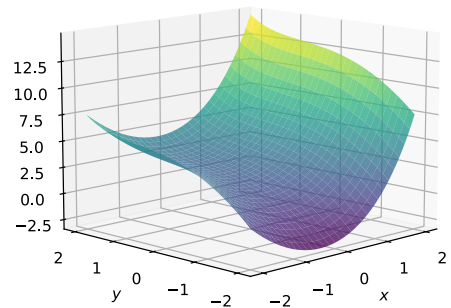
$$H(x,y) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} = \begin{pmatrix} e^x + 2 & 0 \\ 0 & e^y - 2 \end{pmatrix}$$

$y < \log 2$

There is a y s.t. the eigenvalues of H are not all ≥ 0 (i.e. H is not PSD for all y)

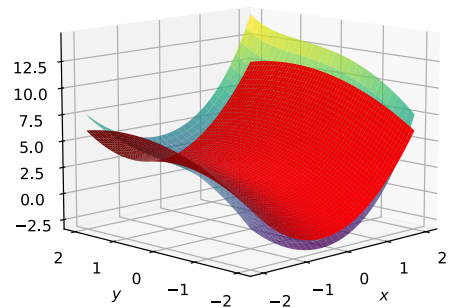
No

- The Hessian at (0,0) has a negative eigenvalue.



No

- The Hessian at (0,0) has a negative eigenvalue.



Exercise

Is $f(\vec{w}) = \|\vec{w}\|^2$ convex?

$$\|\vec{w}\|^2 = \vec{w} \cdot \vec{w} = w_0^2 + w_1^2 + \cdots + w_d^2$$

$$H(\vec{w}) = \begin{pmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 \end{pmatrix}$$

Note

- The second derivative test only works if f is twice differentiable.
- A function can be convex without having a second derivative.

Properties

- We can often prove convexity using properties.

- Two useful properties:
  1. Sums of convex functions are convex.
  2. Affine compositions of convex functions are convex.

Sums of Convex Functions

- Suppose that $f(\vec{x})$ and $g(\vec{x})$ are convex. Then $w_1 f(\vec{x}) + w_2 g(\vec{x})$ is convex, provided $w_1, w_2 \geq 0$.

Affine Composition

- Suppose that $f(x)$ is convex. Let A be a matrix, and $\vec{x}$ and $\vec{b}$ be vectors. Then

$$g(\vec{x}) = f(A\vec{x} + \vec{b})$$

  is convex as a function of $\vec{x}$.

- **Remember:** a vector is a matrix with one column/row.

- Useful!

Exercise

Consider the function

affine function of

$$g(\vec{w}) = (\vec{x} \cdot \vec{w} - y)^2$$

Is this function convex as a function of $\vec{w}$?

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 3

Convex Loss Functions

Empirical Risk Minimization (ERM)

- Step 1: choose a **hypothesis class**
  - We've chosen linear predictors, $H(\vec{x}) = \text{Aug}(\vec{x}) \cdot \vec{w}$.
- Step 2: choose a **loss function**
- Step 3: find $\vec{w}$ minimizing **empirical risk**
  - Some choices of loss function make this **easier**.

Convexity and Gradient Descent

- Convex functions are (relatively) easy to optimize.
- **Theorem**: if $f(x)$ is convex and "not too steep"[1] then (stochastic) (sub)gradient descent converges to a **global optimum** of f *provided* that the step size is small enough[2].

bounded slope everywhere

[1] Technically, c-Lipschitz

[2] step size related to steepness, should decrease like $1/\sqrt{\text{step \#}}$

Convex Loss

- **Recall:** sums of convex functions are convex.
- **Implication:** if loss function is convex as a function of $\vec{w}$, so is the empirical risk, $R(\vec{w})$

$$R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}, y_i)$$

- **Takeaway:** Convex losses make ERM **easier**.

Example: Square Loss

- Recall the square loss for a linear predictor:

$$\ell_{\text{sq}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = (\text{Aug}(\vec{x}) \cdot \vec{w} - y)^2$$

- This is **convex** as a function of $\vec{w}$.

- **Proof**: a few slides ago.

Example: Absolute Loss

- Recall the absolute loss for a linear predictor:

$$\ell_{\text{abs}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = |\,\text{Aug}(\vec{x}) \cdot \vec{w} - y\,|$$

- This is *also* **convex** as a function of $\vec{w}$.

Linear Predictors

- It's also important that we've chosen linear predictors.
- A loss that is **convex** in $\vec{w}$ for linear $H_1(x)$ may be **non-convex** for non-linear $H_2(x)$.
- Example: square loss.
  - If $H_1(x) = w_0 + w_1 x$, then $(w_0 + w_1 x - y)^2$ is **convex**.
  - If $H_2(x) = w_0 e^{w_1 x}$, then $(w_0 e^{w_1 x} - y)^2$ is **non-convex**.

Summary

- By combining 1) linear predictors and 2) a convex loss function, we make ERM **easier**.
- **Many** machine learning algorithms are linear predictors with convex loss functions.
  - As we'll see...

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 4

Appendix: From Theory to Practice

Gradient Descent

- We've spent three lectures on **gradient descent**.
- A powerful optimization algorithm.
- In practice, we use extensions of (stochastic) gradient descent.

Extensions of SGD

- Newton's method
  - Second order optimization, using the Hessian.
  - Can converge in fewer steps.
  - But the Hessian is **expensive** to compute.

- Adagrad, RMSprop, Adam
  - SGD with adaptive learning rates.
  - Used heavily in training of deep neural networks.

Non-Convex Optimization

- So far, we've only seen convex risks.
- But there's an important class of machine learning algorithms that have **non-convex** risks.
- **Namely:** deep neural networks.

Empirical Risk Minimization (ERM)

- Step 1: choose a **hypothesis class**
  - **Deep neural networks**.
- Step 2: choose a **loss function**
- Step 3: find $\vec{w}$ minimizing **empirical risk**

Deep Learning

- A **deep neural network** is a prediction function $H(\vec{x}; \vec{w})$ composed of many layers.
- Typically, H is not linear in $\vec{w}$.
- The risk becomes highly **non-convex**.
  - Even, for example, the square loss.
- How do we minimize the empirical risk?

Answer: SGD

- We use **stochastic gradient descent** (and extensions).
  - Even though the empirical risk is **non-convex**.
  - The optimization problem becomes much harder.
- SGD may not find a global minimum of the risk.
- But often finds a "**good enough**" local minimum.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 5

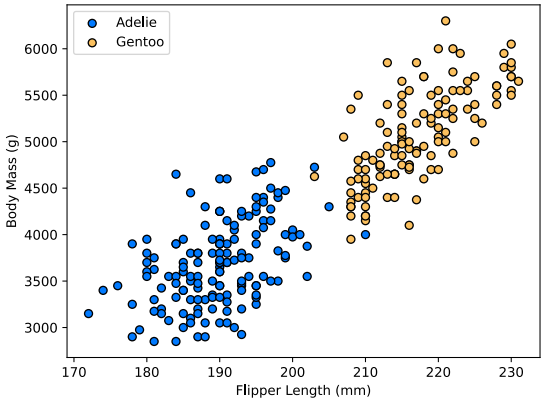
Linear Classification

Classification

- We've been talking about **regression**.
  - Label is a continuous value.
- What about **classification**?
  - Label is a discrete value.

Example: Penguins

- Given a new penguin's measurements, predict its species.



Looking Back

- We know one classification algorithm already.
  - k-Nearest Neighbors.
- But k-NN does not "**learn**", it "**memorizes**".
- Can we use linear predictors for classification?

$$H(\vec{x}) = w_0 + w_1(\text{flipper length}) + w_2(\text{body mass})$$

- Train by minimizing risk?

Linear Classifiers

- **Problem**: output of $H(\vec{x})$ is a real number; we want the output to be a **species**.

$$H(\vec{x}) = w_0 + w_1(\text{flipper length}) + w_2(\text{body mass})$$

- **Idea:** turn species into a number.

Label Encodings

- There are two natural ways to **encode** a label y as a number in **binary classification**.

- $y \in \{0, 1\}$:
  - $y = 0$ for one class, $y = 1$ for the other.
  - **Example:** 0 for Adelie, 1 for Gentoo.

- $y \in \{-1, 1\}$:
  - $y = -1$ for one class, $y = 1$ for the other.
  - **Example:** -1 for Adelie, 1 for Gentoo.

- Unless otherwise specified, we'll use $y \in \{-1, 1\}$.

Linear Classifiers

- Assume the labels are encoded as $y \in \{-1, 1\}$.
- Another **problem**: $H(\vec{x})$ can be any real number.
  - Output is not necessarily –1 or 1.
- We need to turn output of $H(\vec{x})$ into -1 or 1.

Sign Function

- **Idea:** use the **sign function**.

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

Linear Classifiers

- We will still use linear predictors.
  - $H(\vec{x}; \vec{w}) = \text{Aug}(\vec{x}) \cdot \vec{w}$.
- But our final **predicted label** will be $\text{sign}(H(\vec{x}; \vec{w}))$.
  - If $H(\vec{x}) = 0$, predict either 1 or -1 (it's arbitrary).
- $\text{sign}(H(\vec{x}; \vec{w}))$ is called a **linear classifier**.
  - Takes in a feature vector and outputs a discrete label.
  - Sometimes called a **linear decision function.**

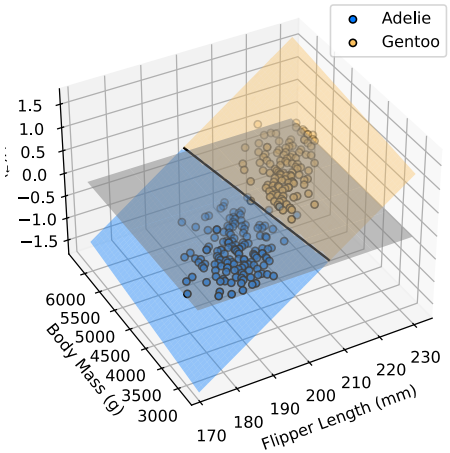
Interpretation: Weighted Vote

- A linear classifier is like a **weighted vote**.
- Each term $w_i x_i$ "votes" on the label.

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

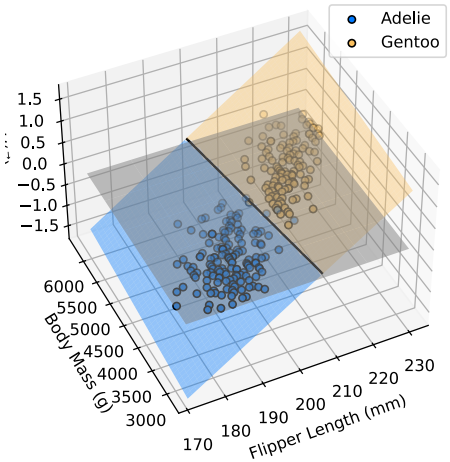
- If the sum is:
  - positive: predict 1.
  - negative: predict –1.
  - zero: toss a coin, it's arbitrary!

The Prediction Surface



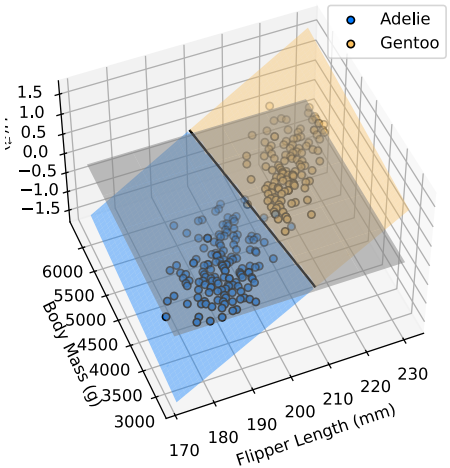
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



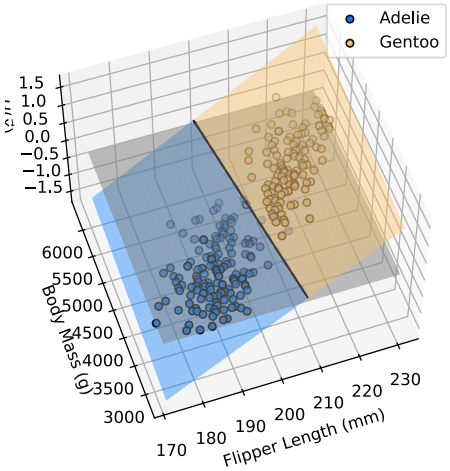
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



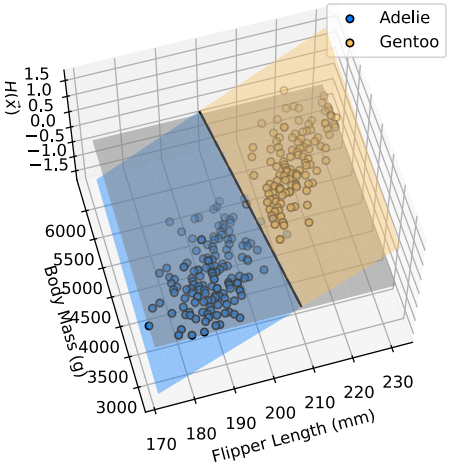
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict −1.

The Prediction Surface



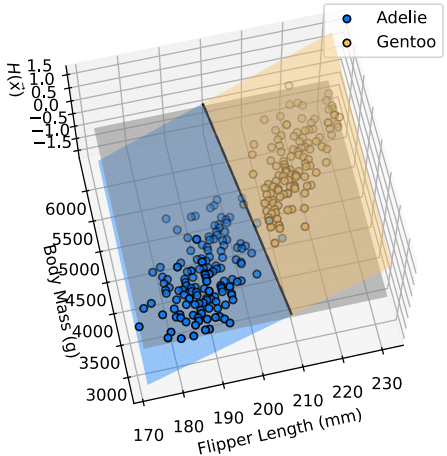
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



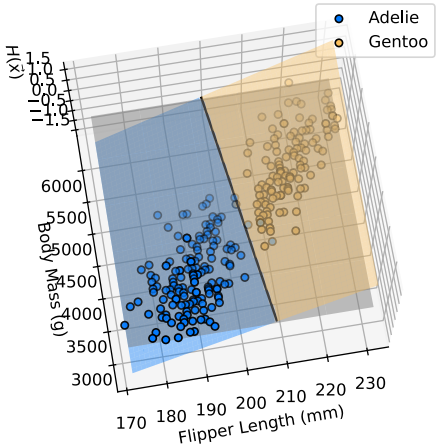
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



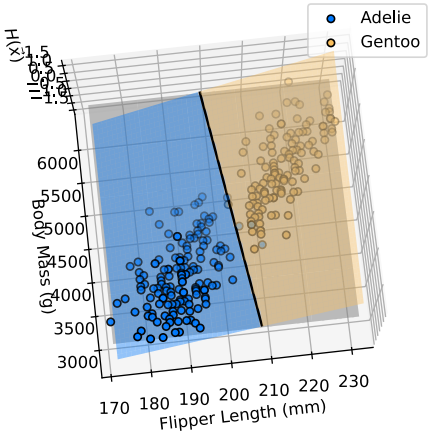
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



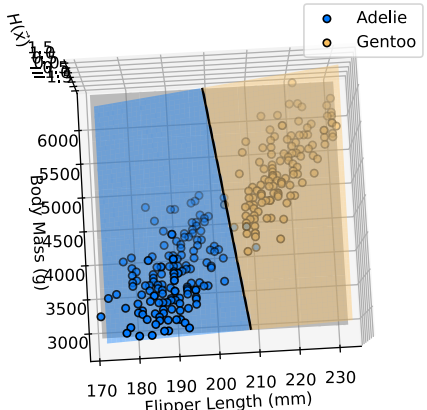
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



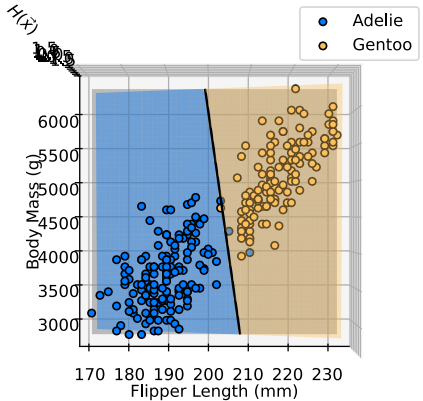
- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

The Prediction Surface



- $H(\vec{x})$ is a (hyper) plane.
- The place where $H(\vec{x}) = 0$ is the **decision boundary**.
- On one side, we predict 1.
- On the other, we predict –1.

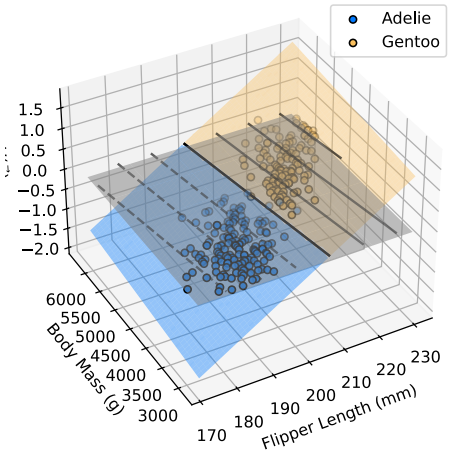
Decision Boundary

- The **decision boundary** is the place where the output of $H(\vec{x})$ switches from "yes" to "no".

- If H is a linear predictor and[3]
  - $\vec{x} \in R^1$, then the decision boundary is just a number.
  - $\vec{x} \in \mathbb{R}^2$, the boundary is a straight line.
  - $\vec{x} \in \mathbb{R}^d$, the boundary is a $d - 1$ dimensional (hyper) plane.

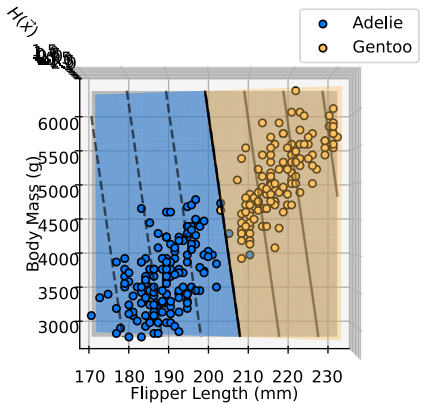
[3] when plotted in the original feature coordinate space!

Magnitude of H



- The magnitude of $H(\vec{x})$ is **proportional** to the distance from the decision boundary.

Magnitude of H

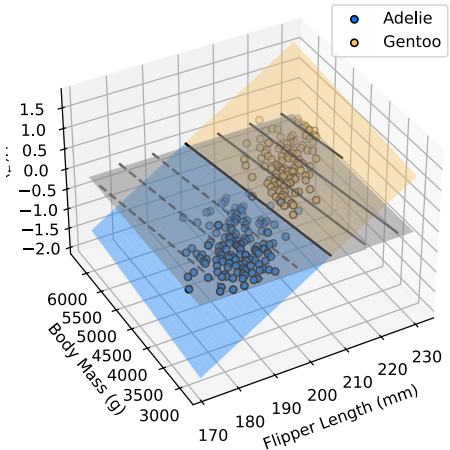


- The magnitude of $H(\vec{x})$ is **proportional** to the distance from the decision boundary.

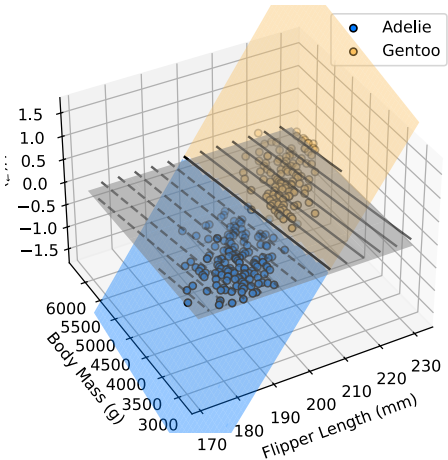
Exercise

True or **False**: it's possible for two different linear prediction functions $H_1(\vec{x})$ and $H_2(\vec{x})$ to have the exact same decision boundary.

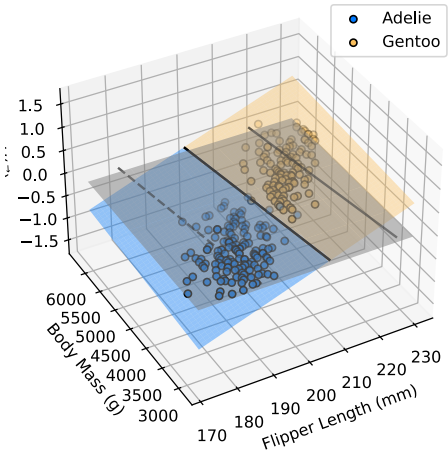
True


Adelie
Gentoo
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
6000
5500
5000
4500
4000
3500
3000
Body Mass (g)
170
180
190
200
210
220
230
Flipper Length (mm)

True


Adelie
Gentoo
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
6000
5500
5000
4500
4000
3500
3000
Body Mass (g)
170
180
190
200
210
220
230
Flipper Length (mm)

True


Adelie
Gentoo
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
6000
5500
5000
4500
4000
3500
3000
Body Mass (g)
170
180
190
200
210
220
230
Flipper Length (mm)

Another Useful Fact

- $\vec{w}$ controls the orientation of the decision boundary.
  - A different $\vec{w}$ gives a different decision boundary.
- Let $\vec{w}' = (w_1, \ldots, w_d)$.
  - In other words, it is $\vec{w}$ without the bias term w_0.
- **Fact**: the decision boundary of $H(\vec{x}) = \text{Aug}(\vec{x}) \cdot \vec{w}$ is orthogonal to $\vec{w}'$.

Finding a Linear Classifier

- How do we find a good linear classifier?

ERM for Classification

- Step 1: choose a **hypothesis class**
  - We've chosen linear classifiers, $\text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w})$.
- Step 2: choose a **loss function**
- Step 3: find H minimizing **empirical risk**
  - In case of linear predictors, equivalent to finding $\vec{w}$.

A First Idea

- Let's try using the same, familiar **square loss**.

DSC 140A

Probabilistic Modeling & Machine Learning

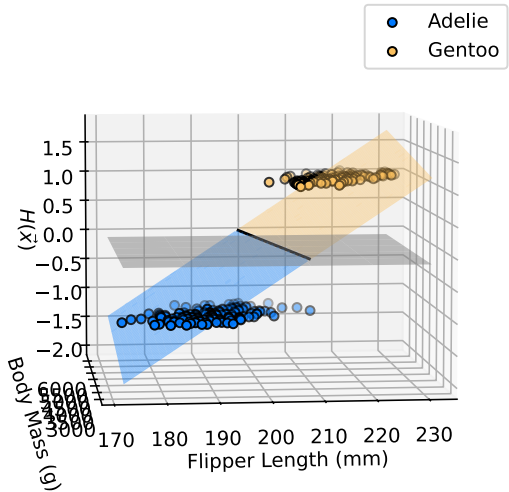
Lecture 6 | Part 6

Least Squares Classifiers

Classification as Regression

- We can think of classification as a special case of regression where the labels are always 1 or -1.

- **Goal**: find a prediction function $H(\vec{x})$ whose output is:
  - close to 1 for points from positive class.
  - close to -1 for points from negative class.


Adelie
Gentoo
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
$H(\vec{x})$
Body Mass (g)
170
180
190
200
210
220
230
Flipper Length (mm)

Least Squares Classifier

- **Idea:** least squares regression can be used for classification, too.

- The resulting algorithm is called the **least squares classifier**.

Linear Least Squares Classification

- **To train:**
  - Given training data $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$, with $y_i \in \{-1, 1\}$.
  1. Construct $n \times (d + 1)$ augmented design matrix, X.
  2. Solve the **normal equations**: $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$.

- **To predict:**
  - Given a new point $\vec{x}$, predict $\text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}^*)$.

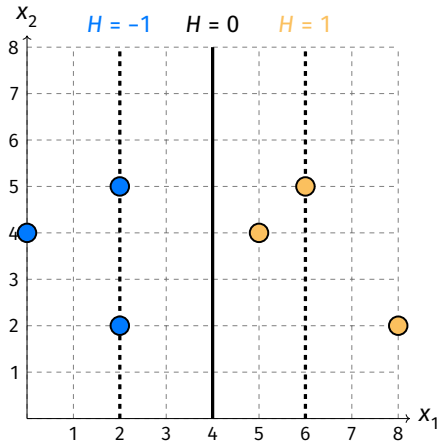
Square Loss for Classification

- We designed square loss for **regression**
- We can use it for **classification**.
- But it might not be the best choice.

Exercise

What is the **total** square loss of the predictor on the data?

Assume ● is class -1 and ● is class 1.



Observation

- The square loss penalizes points that are far from the decision boundary.
- **Even if they are correctly classified!**

Least Squares and Outliers

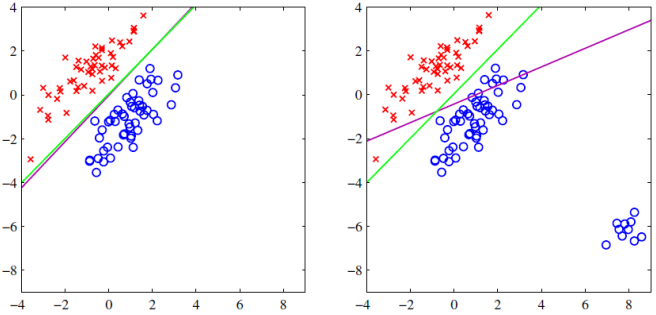


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

[4] Bishop, *Pattern Recognition and Machine Learning*

Another Loss?

- Least squares classifiers can work well in practice.
  - **Easy to implement!**
- But maybe a loss designed for classification will work better.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 7

0-1 Loss

Empirical Risk Minimization

- Step 1: choose a **hypothesis class**
  - Let's assume we've chosen linear predictors
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Another Loss Function

- What about the **0-1 loss**?
  - Loss = 0 if prediction is **correct**.
  - Loss = 1 if prediction is **incorrect**.

- More formally:

$$\ell_{0\text{-}1}(H(\vec{x}^{(i)}), y_i) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

Expected 0-1 Loss

- The expected 0-1 loss (empirical risk) has a nice interpretation:

$$R_{0\text{-}1}(H) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i\end{cases}$$

Exercise

What is it?

Answer

- The empirical risk with respect to the 0-1 loss is the **misclassification rate** of the classifier.
  - That is, (1 - the **accuracy**)

$$R_{0\text{-}1}(H) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i\end{cases}$$

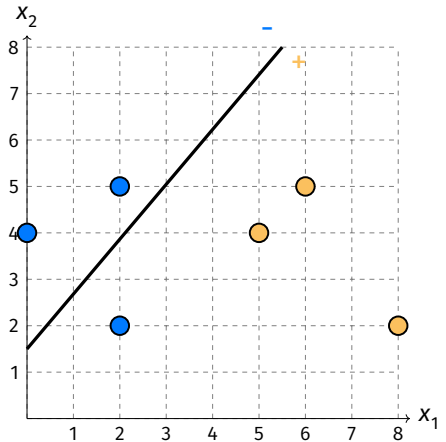
$$= \frac{\#\text{ of } \textbf{incorrect} \text{ predictions}}{n}$$

ERM for the 0-1 Loss

- Minimizing the empirical risk with respect to the 0-1 loss is equivalent to **maximizing the accuracy**.
- That's exactly what we want!
- But there's a **problem**...

Exercise

What is the **gradient** of $R_{0\text{-}1}(H)$ with respect to the current $\vec{w}$?



Answer

- The gradient of $R_{0\text{-}1}$ is $\vec{0}$ almost everywhere.
- In other words, $R_{0\text{-}1}$ is **flat** almost everywhere.
- This is a **problem** because **gradient descent** needs slope information to make progress.

Computationally Difficult

- It is **not feasible** to minimize 0-1 risk in general.
- More formally: **NP-Hard** to optimize expected 0-1 loss in general.[5]

[5] It is efficiently doable if the classes are linearly separable by finding convex hulls of each class. If non-separable, it is difficult.

Main Idea

It is computationally difficult (NP-Hard) to find a linear classifier with maximum accuracy, in general.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 8

Perceptron Loss

Surrogate Loss

- We'd like to use the 0-1 loss, but it's not feasible.
- Instead, we use a **surrogate loss**.
- That is, a loss that is similar in spirit, but leads to easier optimization problems.

A New Loss

- No penalty if point is **correctly classified**.
  - Like the 0-1 loss.
- A penalty that grows with distance to decision boundary if point is **incorrectly classified**.
  - Unlike the 0-1 loss.
  - This will give us a non-zero gradient.

Perceptron Loss

- We call this loss the **perceptron loss**.

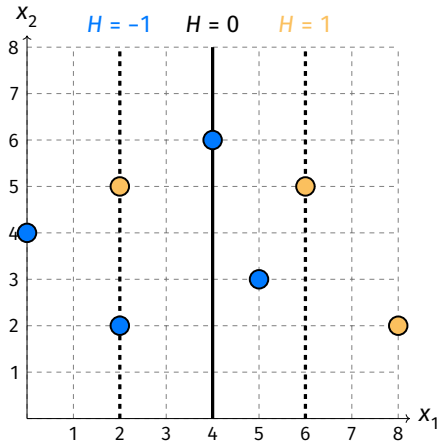
$$\ell_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = y \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq y \end{cases}$$

- Remember, $|H(\vec{x})|$ is **proportional** to distance from decision boundary.

Exercise

What is the **total** perceptron loss of the predictor on the data?

Assume ● is class -1 and ● is class 1.



Convexity?

- Is the perceptron loss **convex** in $\vec{w}$?
- Trick:

$$\ell_{\text{tron}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = \begin{cases} 0, & \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}) = y \\ |\text{Aug}(\vec{x}) \cdot \vec{w}|, & \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}) \neq y \end{cases}$$

$$= \max(0, -y\,\text{Aug}(\vec{x}) \cdot \vec{w})$$

- **Fact:** Max of convex functions **is convex**.

ERM for the Perceptron

- **Goal:** minimize empirical risk w.r.t. perceptron loss for a linear predictor $H(\vec{x}) = \text{Aug}(\vec{x}) \cdot \vec{w}$.

$$
\begin{aligned}
R_{\text{tron}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^{n} \ell_{\text{tron}}(H(\vec{x}^{(i)}), y_i) \\
&= \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 0, & \text{sign}(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}) = y_i \\ |\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}|, & \text{sign}(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}) \neq y_i \end{cases}
\end{aligned}
$$

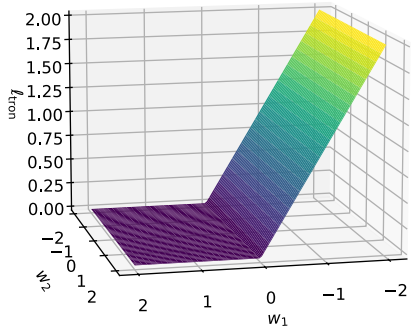
Minimizing Perceptron Risk

- R_{tron} is **not differentiable**.
  - Because of the absolute value.
- But it is **convex**.
  - Since ℓ_{tron} is convex.
- We can minimize using **subgradient descent**.

A Subgradient of the Loss

- We need a subgradient of ℓ_{tron}.

$$\ell_{\text{tron}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = \max(0, -y\,\text{Aug}(\vec{x}) \cdot \vec{w})$$



A Subgradient of the Loss

- We need a subgradient of ℓ_{tron}.

$$\ell_{\text{tron}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = \max(0, -y\,\text{Aug}(\vec{x}) \cdot \vec{w})$$

- If $-y\,\text{Aug}(\vec{x}) \cdot \vec{w} > 0$, the gradient is $-y\,\text{Aug}(\vec{x})$.
- If $-y\,\text{Aug}(\vec{x}) \cdot \vec{w} < 0$, the gradient is $\vec{0}$.
- **Claim:** at $-y\,\text{Aug}(\vec{x}) \cdot \vec{w} = 0$, $\vec{0}$ is a subgradient.

Subgradient of the Loss

- We've found:

$$\text{subgrad}\, \ell_{\text{tron}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = \begin{cases} \vec{0}, & \text{if } -y\,\text{Aug}(\vec{x}) \cdot \vec{w} < 0 \\ -y\,\text{Aug}(\vec{x}), & \text{if } -y\,\text{Aug}(\vec{x}) \cdot \vec{w} > 0 \end{cases}$$

- Or, equivalently:

$$\text{subgrad}\, \ell_{\text{tron}}(\text{Aug}(\vec{x}) \cdot \vec{w}, y) = \begin{cases} \vec{0}, & \text{if } \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}) = y \\ -y\,\text{Aug}(\vec{x}), & \text{if } \text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}) \neq y \end{cases}$$

Subgradient of the Risk

- A subgradient of the risk is then:

$$\text{subgrad } R_{\text{tron}}(\vec{w}) =$$

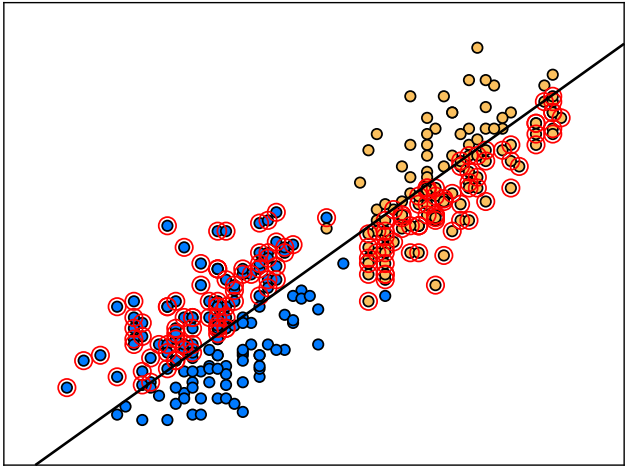
$$\frac{1}{n}\sum_{i=1}^{n}\begin{cases}\vec{0}, & \text{sign}(\text{Aug}(\vec{x}^{(i)})\cdot\vec{w}) = y_i \\ -y_i\,\text{Aug}(\vec{x}^{(i)}), & \text{sign}(\text{Aug}(\vec{x}^{(i)})\cdot\vec{w}) \neq y_i\end{cases}$$

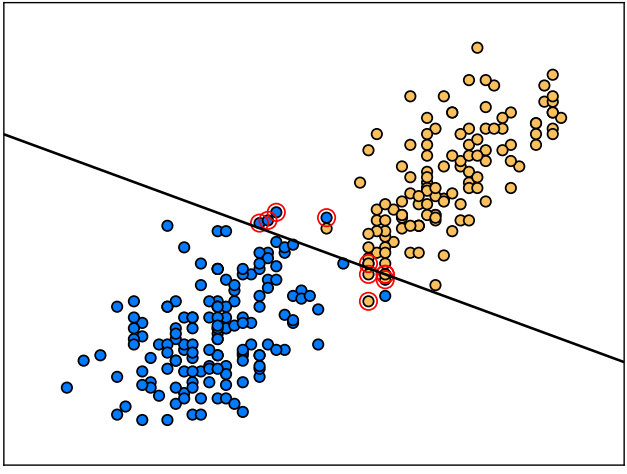
The Perceptron

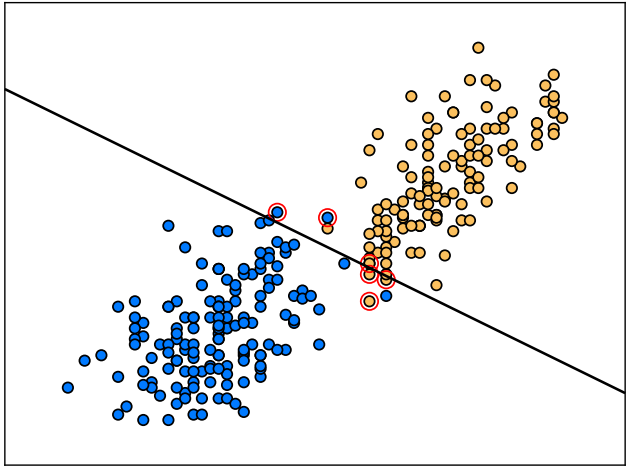
- **To train:**
  - Given training data $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$, with $y_i \in \{-1, 1\}$.
  1. Minimize $R_{\text{tron}}(\vec{w})$ with, e.g., subgradient descent:

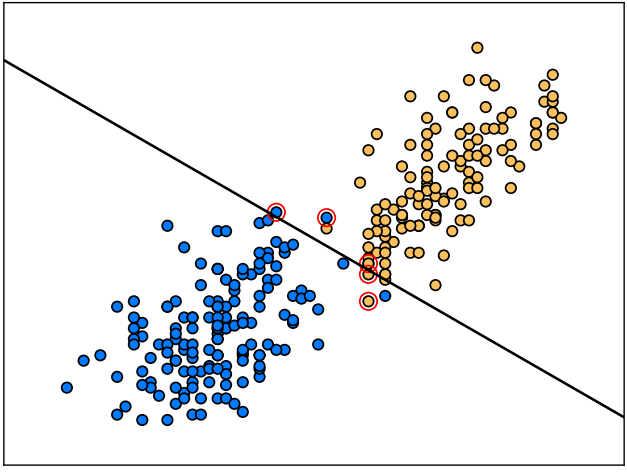
$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta(t) \times \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \vec{0}, & \text{sign}(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}) = y_i \\ -y_i \, \text{Aug}(\vec{x}^{(i)}), & \text{sign}(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}) \neq y_i \end{cases}$$

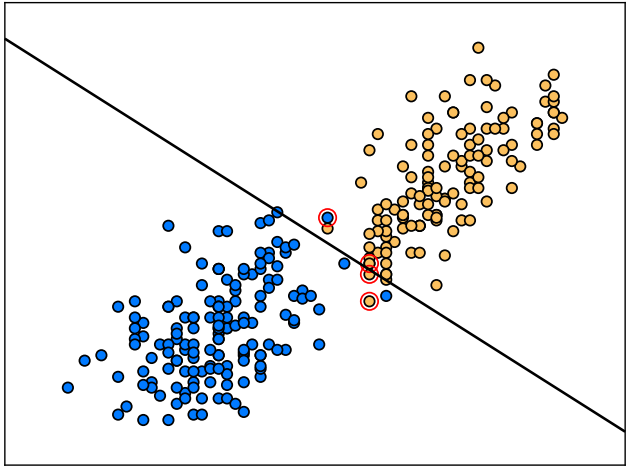
- **To predict:**
  - Given a new point $\vec{x}$, predict $\text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}^*)$.

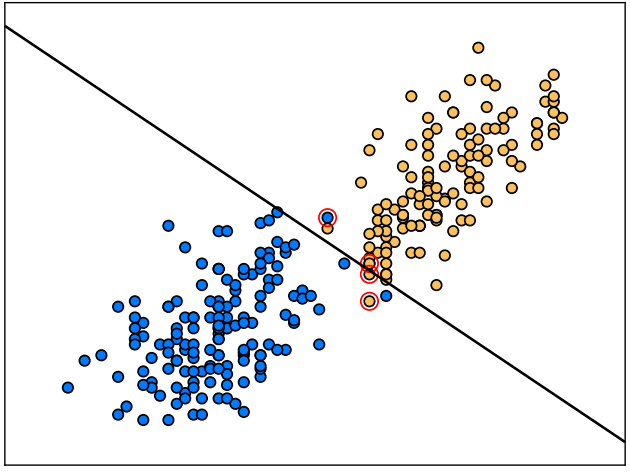


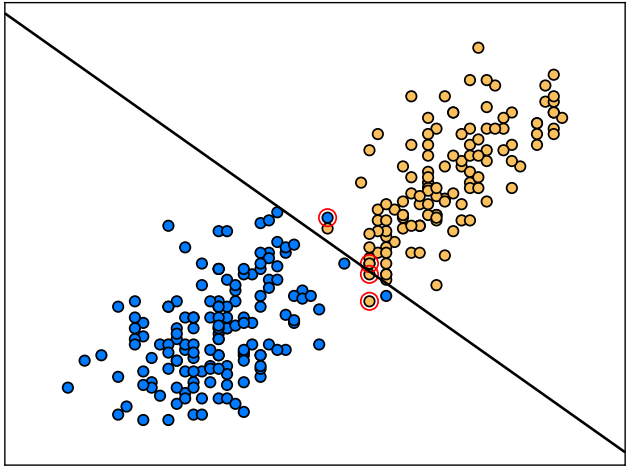


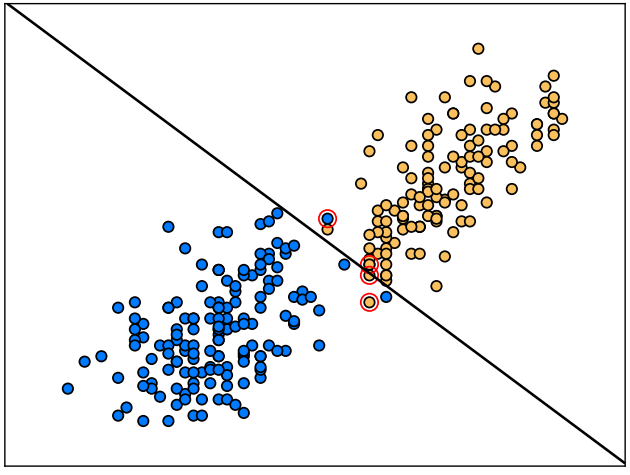


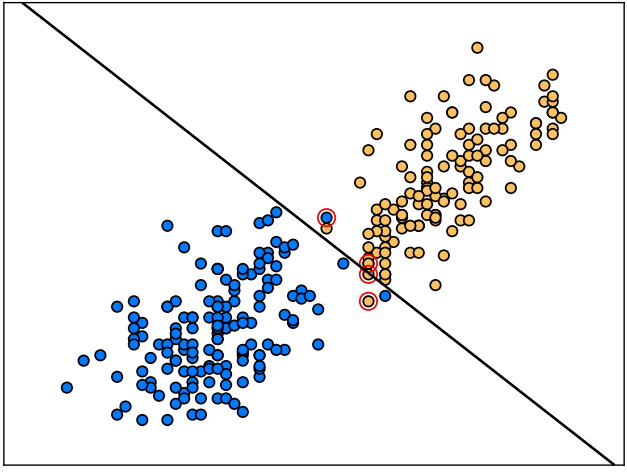


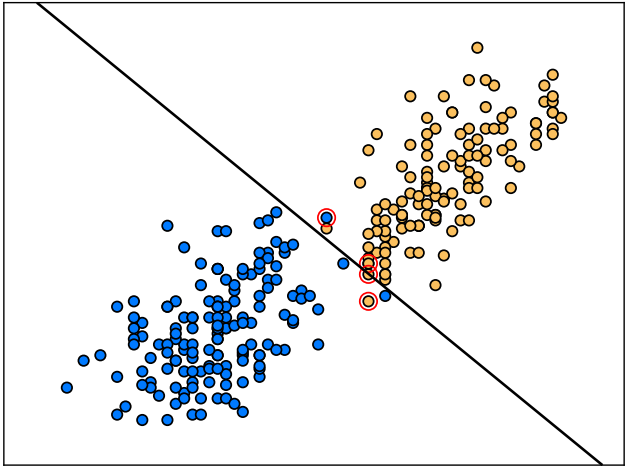


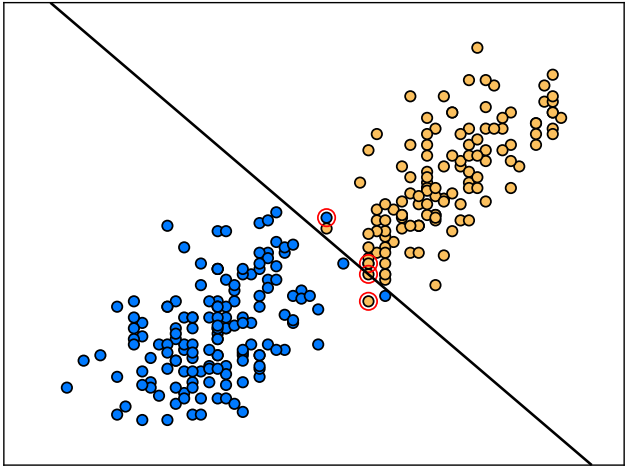


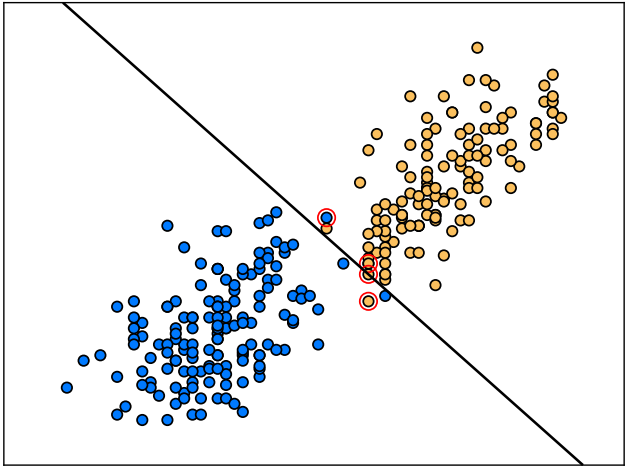


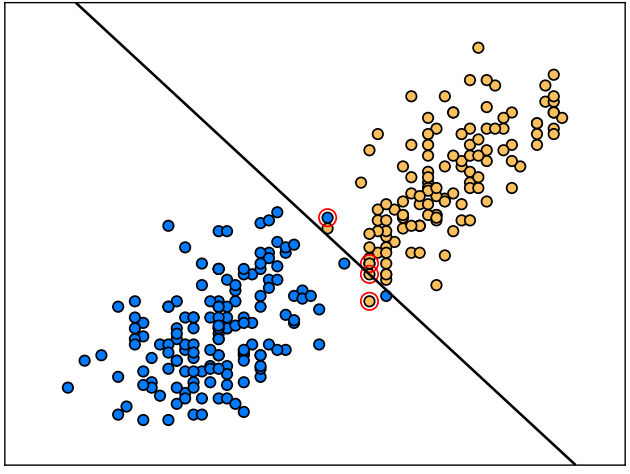


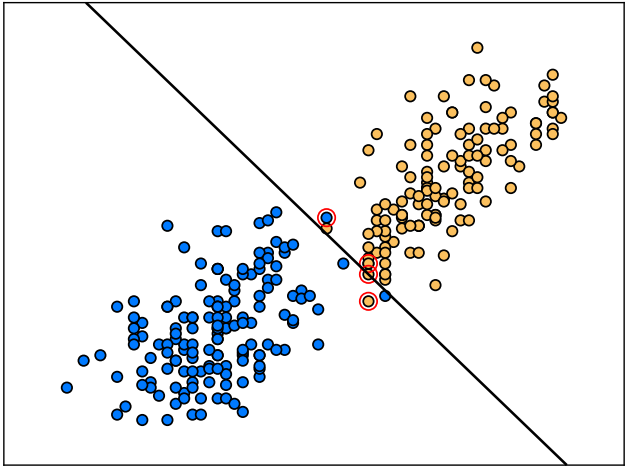


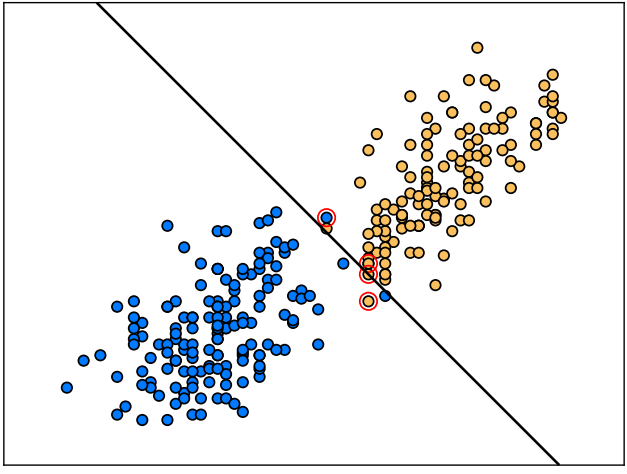


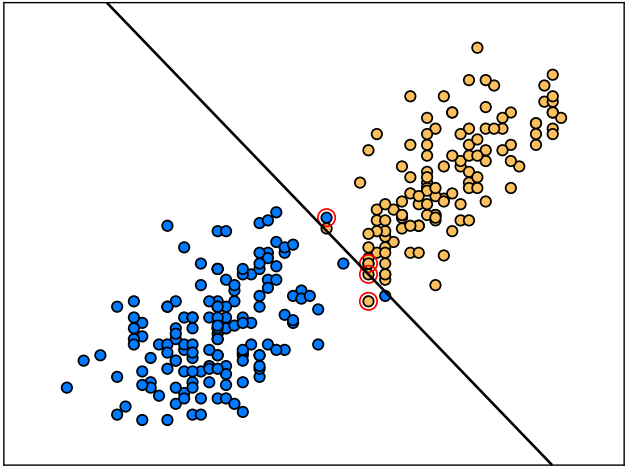


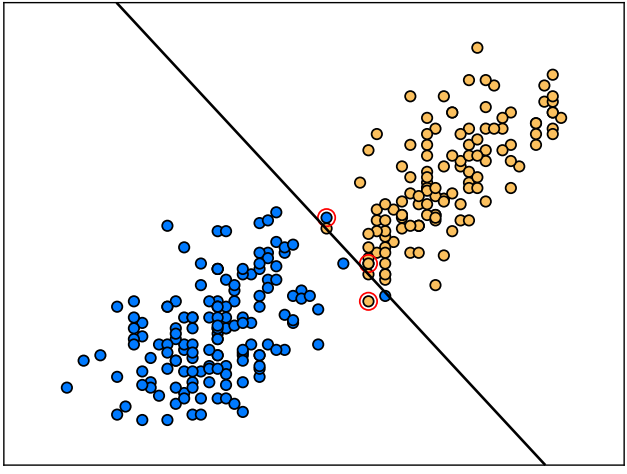


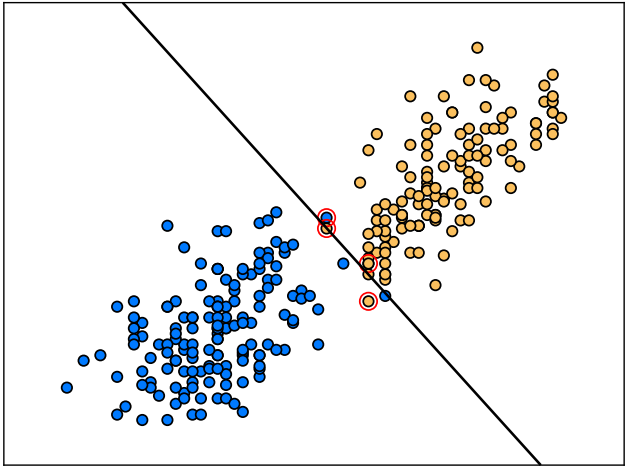


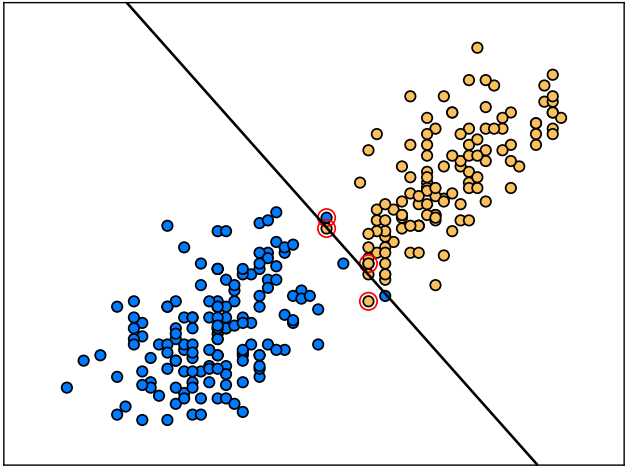


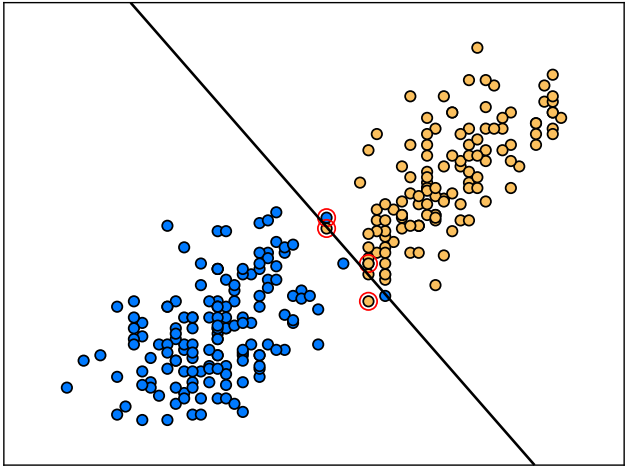












DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 6 | Part 9

Perceptron Demo: MNIST

Demo: MNIST

- MNIST is a classic machine learning data set.
- Many images of handwritten digits, 0-9.
- Multiclass classification problem.
- But we can make it binary: 3 vs. 7.

Example MNIST Digit

- Grayscale
- 28 x 28 pixels

MNIST Feature Vectors

- $28 \times 28 = 784$ pixels
- Each image is a vector in $\mathbb{R}^{784}$
- Each feature is intensity of single pixel
  - black $\rightarrow$ 0, white $\rightarrow$ 255
- A **very** simple representation.

Demo: MNIST

- Use only images of 3s and 7s.
- 4132 training images.
- 680 testing images.
- Some minor tuning.
  - Added random noise for robustness.
  - Picked classification threshold automatically.

Perceptron Learning

- Linear prediction function parameterized by $\vec{w}$.
- In this case, we can “reshape” $\vec{w}$ to be same size as input image.

Weight Vector

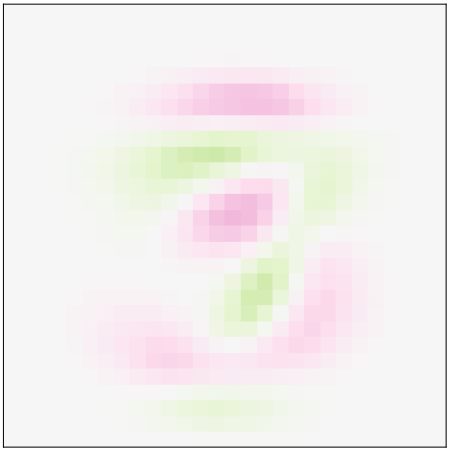
- Recall that the prediction is a **weighted vote**:

$$H(\vec{x}) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \ldots w_{784} x_{784})$$

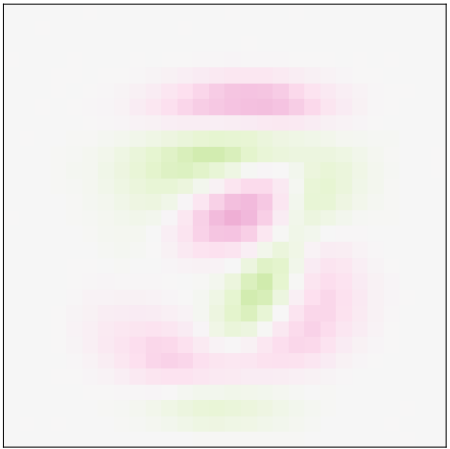
- Positive $\rightarrow$ 7, Negative $\rightarrow$ 3

- w_i is the weight of pixel i
  - positive: if this pixel is bright, I think this is a 7
  - negative: if this pixel is bright, I think this is a 3
  - magnitude: confidence in prediction

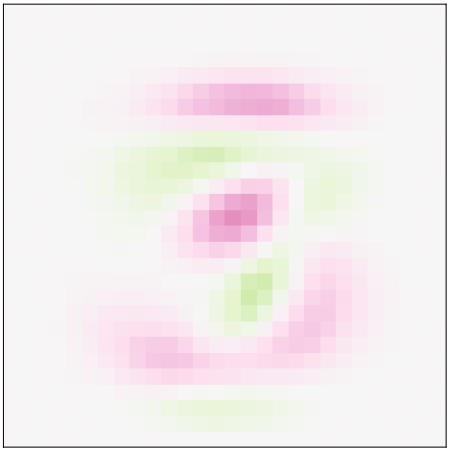
Perceptron Training



Perceptron Training



Perceptron Training



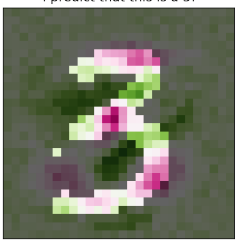
Perceptron Training

Perceptron Training

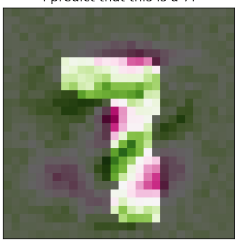
Perceptron Training

Perceptron Weight Vector

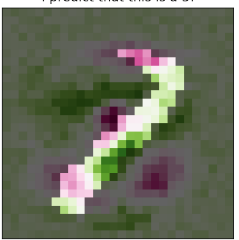
I predict that this is a 3!



I predict that this is a 7!



I predict that this is a 3!



Perceptron Results

- Test accuracy: 97.3%

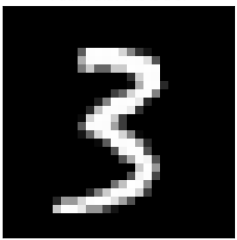
Square Loss for Classification

- What if we use square loss for classification?
- We *can*, but will it work well?

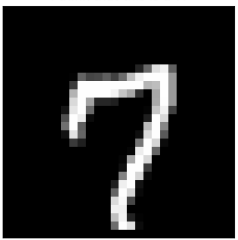
Results: Least Squares

- Test Accuracy: 96.7% (marginally worse)

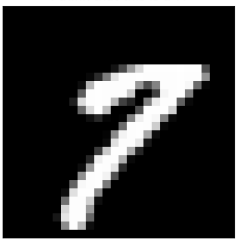
I think that this is a 3.



I think that this is a 7.



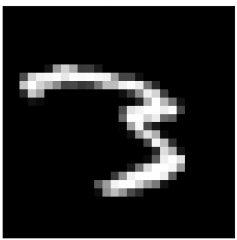
I think that this is a 7.



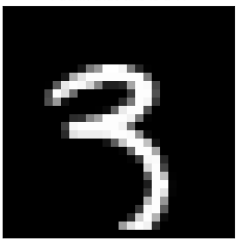
Results: Least Squares

- Misclassifications are telling.

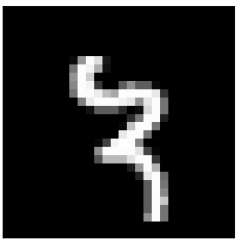
I think that this is a 7.



I think that this is a 7.

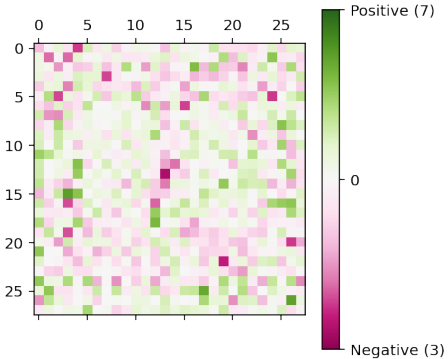


I think that this is a 7.



Least Squares Weight Vector

- Can visualize weight of each pixel as an image.

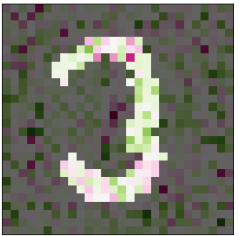


Least Squares Weight Vector

I predict that this is a 7!

I predict that this is a 3!

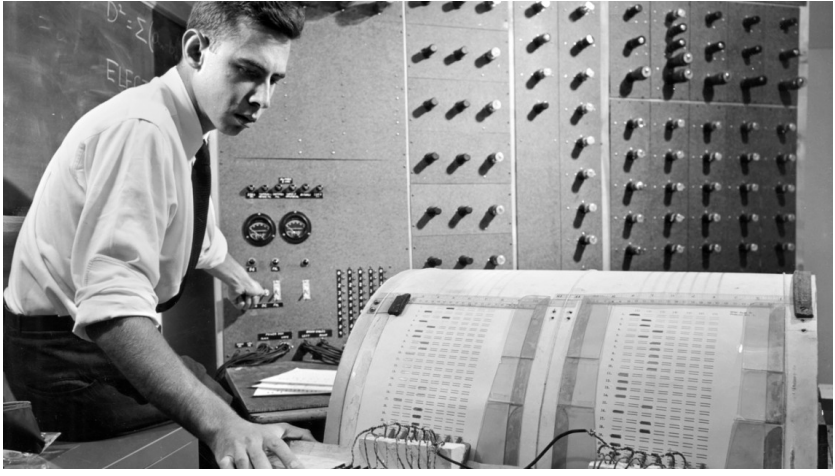
I predict that this is a 7!

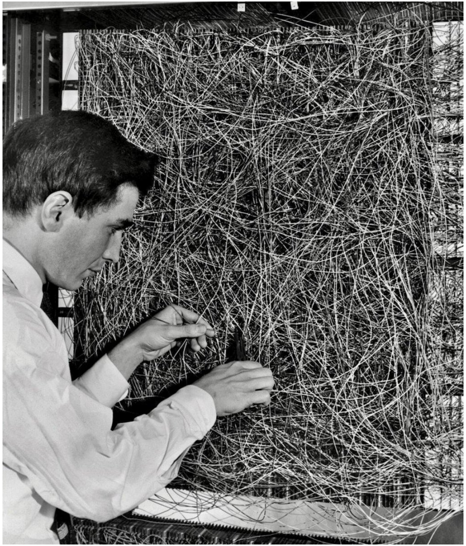


Some History

- Perceptrons were one of the first “machine learning” models.
- The basis of modern neural networks.

Rosenblatt's Perceptron





Next Time

- We “solve” linear classification, once and for all.