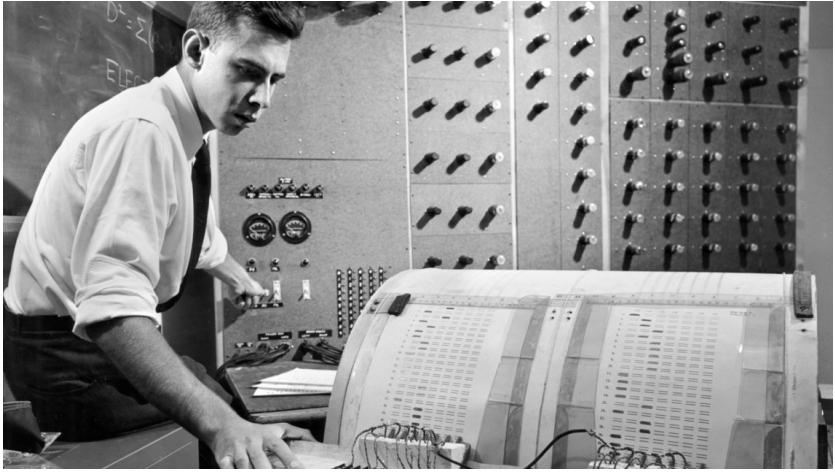
DSC 190

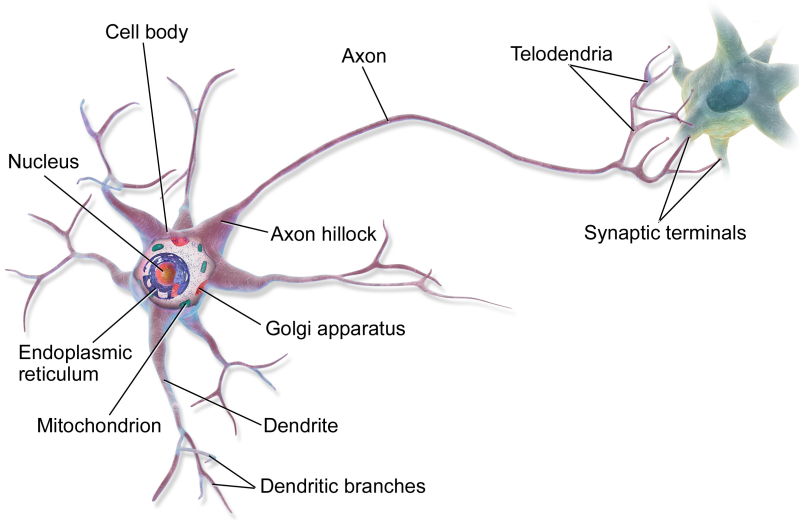
Machine Learning: Representations

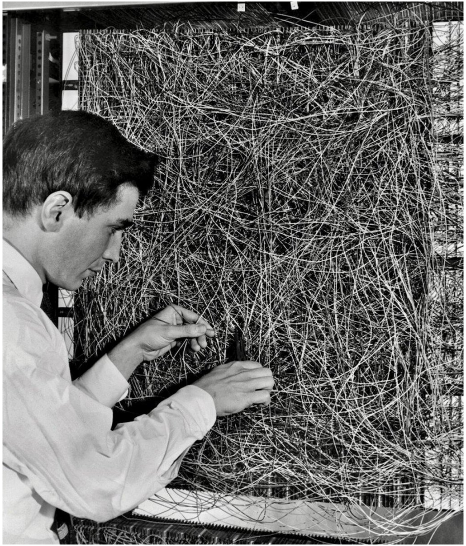
Lecture 2 | Part 1

Learning in the 1950s

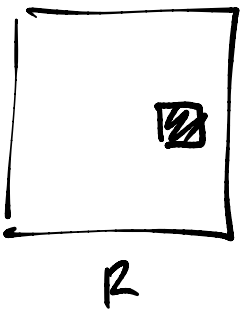
Rosenblatt's Perceptron

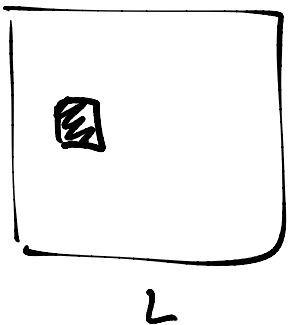



Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches



The Task





1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

By the end of today...

- We'll see how this machine worked.
- We'll build one to recognize images.
- We'll see what it's limitations are.

DSC 190

Machine Learning: Representations

Lecture 2 | Part 2

Learning to Predict

Predicting Opinions

- We often use the opinions of others to predict our own.
- But we don’t hold all opinions equally...

Movie Ratings

- Friend A: “This movie was great!”
  - → I know I’ll like it.
- Friend B: “This movie was great!”
  - → I know I *won’t* like it.
  - Still useful!
- Friend C: “This movie was great!”
  - → I don’t know... they like every movie!
  - Not useful.

Movie Ratings

- Five of your friends rate a movie from 0-10:
  - x_1: 9
  - x_2: 3
  - x_3: 7
  - x_4: 2
  - x_5: 8
- **Task**: What will *you* rate the movie?

Prediction

- **Prediction** is a core ML task.
- **Regression**: output is a number.
  - Example: movie rating, future salary
- **Classification**: output is a **class label**.
  - Example: like the movie? mango is ripe? (yes/no) → **binary**
  - Example: species (cat, dog, mongoose) → **multiclass**

Prediction Functions

- Informally: we think our friends' ratings predict our own.
- Formally: we think there is a function H that takes our friend's ratings $\vec{x} = (x_1, x_2, x_3, x_4, x_5)$ and outputs a good prediction of our rating.

$$H(\vec{x}) \rightarrow \text{prediction}$$

- H is called a **prediction function**.[1]

[1] Or, sometimes, a **hypothesis function**

Prediction Functions

- **Problem**: There are **infinitely many** prediction functions.
  - $H_1(\vec{x}) = -2x_1 + 3x_5$
  - $H_2(\vec{x}) = \sin(x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5)$
  - $H_3(\vec{x}) = \sqrt{x_1 + x_3}(x_1 - x_2 x_5 + 100)$
  - ...[2]
- How do we pick one?

[2] Most can't even be expressed algebraically.

The Fundamental Assumption of Learning

- Informally: The past will repeat itself.
- Formally: A prediction function that made good predictions in the past will continue to make good predictions in the future[3].

[3] This isn't always true!

Picking a Prediction Function

- **Idea:** Use **data** to pick a prediction function that worked well in the past.
- We *hope* it **generalizes** to future predictions.
- A function that did well in the past but does not generalize is said to have **overfit**.

Training Data

Movie	x_1	x_2	x_3	x_4	x_5	You
#1	8	5	9	2	1	6
#2	3	5	7	8	2	8
#3	1	5	2	3	3	9
#4	0	5	3	8	2	?

A Learning Meta-Algorithm

- Given data, how do we choose a prediction function?

- One common strategy is **empirical risk minimization** (ERM).
  - a.k.a., “minimizing expected loss”

Empirical Risk Minimization (ERM)

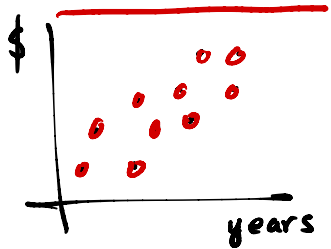
- Step 1: choose a **hypothesis class**
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Hypothesis Classes

- A **hypothesis class** $\mathcal{H}$ is a set of possible prediction functions.
- By choosing a hypothesis class, we are saying something about what the prediction function should look like.
- Examples:
  - $\mathcal{H}$:= linear functions
  - $\mathcal{H}$:= functions of the form $\sin(w_1 x_1 + \ldots x_5 x_5)$
  - $\mathcal{H}$:= decision trees of depth 10
  - $\mathcal{H}$:= neural networks with one layer

Exercise

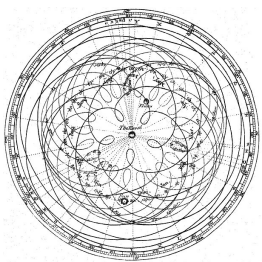
Why not just choose $\mathcal{H}$ to be the set of all possible functions?

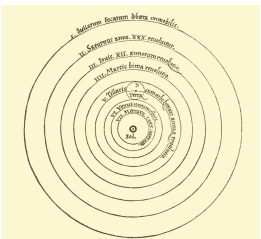


Hypothesis Class Complexity

- The more complex the hypothesis class, the greater the danger of **overfitting**.
  - Think: polynomials of degree 10 versus 2.

- Occam's Razor: assume H is simple.





DSC 190

Machine Learning: Representations

Lecture 2 | Part 3

Linear Regression

A Simple Prediction Function

- We can go a long way by assuming our prediction functions to be **linear**.

A Simple Prediction Function

- Five of your friends rate a movie from 0-10:
  - x_1: 9
  - x_2: 3
  - x_3: 7
  - x_4: 2
  - x_5: 8

- Predict the average: [4]
  $H(\vec{x}) = (x_1 + x_2 + x_3 + x_4 + x_5)/5 = (9 + 3 + 7 + 2 + 8)/5$

[4] There is only one function in this hypothesis class: $(x_1 + x_2 + \ldots + x_5)/5$

Exercise

Why is this a bad prediction function?

A Better Hypothesis Class

- A weighted “vote”:

$$H(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

Exercise

Suppose you are a cynic (you dislike everything). How can the prediction function be changed to take into account the fact that your ratings are likely lower than average across the board?

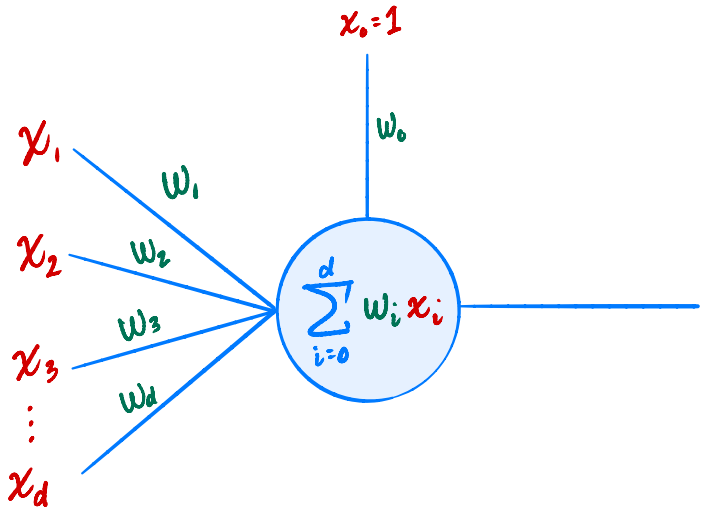
$$H(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 \; - w_0$$

An Even Better Hypothesis Class: Linear Prediction Functions

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

- This is a **linear prediction function**.
- $w_0, w_1, \ldots, w_5$ are the **parameters** or **weights**.
- $\vec{w} = (w_0, \ldots, w_5)^T$ is a **parameter vector**.

Linear Predictors



Class of Linear Functions

- There are infinitely many functions of the form

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

- Each one is completely determined by $\vec{w}$.
  - Sometimes write $H(\vec{x}; \vec{w})$

- Example: $\vec{w} = (8, 3, 1, 5, -2, -7)^T$ specifies

$$H(\vec{x}; \vec{w}) = 8 + 3x_1 + 1x_2 + 5x_3 - 2x_4 - 7x_5$$

“Parameterization”

- A very useful trick.
- Searching all linear functions ≡ searching over $\vec{w} \in \mathbb{R}^6$

In General

- If there are d features, there are $d + 1$ parameters:

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

$$= w_0 + \sum_{i=1}^{d} w_i x_i$$

Linear Prediction and the Dot Product

- The **augmented feature vector** $\text{Aug}(\vec{x})$ is the vector obtained by adding a 1 to the front of $\vec{x}$:

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \qquad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

$w_0 + w_1 x_1 + w_2 x_2 + \dots w_d x_d$

Simplification

- With augmentation, we can write as dot product:

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$$
$$= \text{Aug}(\vec{x}) \cdot \vec{w}$$

$\vec{w}$ $\quad$ $\text{Aug}(\vec{x})$

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix}$$

$$\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} \qquad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

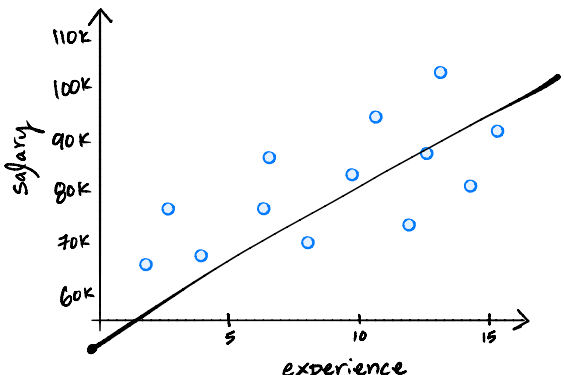
Geometric Meaning

- It can be very useful to **think geometrically** when reasoning about prediction algorithms.

Example

- A linear prediction function for salary.

$$H_1(\vec{x}) = \$50{,}000 + (\text{experience}) \times \$8{,}000$$



Regression

- The **surface** of a prediction function H is the surface made by plotting $H(\vec{x})$ for all $\vec{x}$.

- If H is a linear prediction function, and[5]
  - $\vec{x} \in R^1$, then $H(x)$ is a straight line.
  - $\vec{x} \in \mathbb{R}^2$, the surface is a plane.
  - $\vec{x} \in \mathbb{R}^d$, the surface is a d-dimensional **hyperplane**.

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

Empirical Risk Minimization (ERM)

- 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)**

Step #2: Choose a loss function

- Suppose we assume prediction function is linear.
- There are still infinitely-many possibilities.
- We'll pick one that works well on training data.
- What does "works well" mean?

Example: Movie Ratings

Movie	x_1	x_2	x_3	x_4	x_5	You
#1	8	5	9	2	1	6
#2	3	5	7	8	2	8
#3	1	5	2	3	3	9
#4	0	5	3	8	2	?

Quantifying Quality

- Consider a training example $(\vec{x}^{(i)}, y_i)$
  - Notation: $\vec{x}^{(i)}$ is the "ith training example"
  - $\vec{x}_j^{(i)}$ is the "jth entry of the ith training example"
- The "right answer" is y_i
- Our prediction function outputs $H(\vec{x}^{(i)})$
- We measure the difference using a **loss function**.

Loss Function

- A **loss function** quantifies how wrong a single prediction is.

$$L(H(\vec{x}^{(i)}), y_i)$$

L(prediction for example i, correct answer for example i)

Empirical Risk

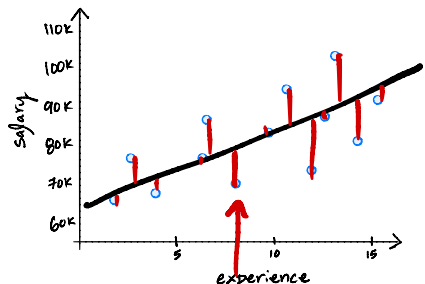
- A good H is good *on average* over entire data set.
- The **expected loss** (or **empirical risk**) is one way of measuring this:

$$R(H) = \frac{1}{n} \sum_{i=1}^{n} L(H(\vec{x}^{(i)}), y_i)$$

- Note: depends on H and the data!

Loss Functions for Regression

- We want $H(\vec{x}^{(i)}) \approx y_i$.
- **Absolute loss**: $|H(\vec{x}^{(i)}) - y_i|$
- **Square loss**: $(H(\vec{x}^{(i)}) - y_i)^2$

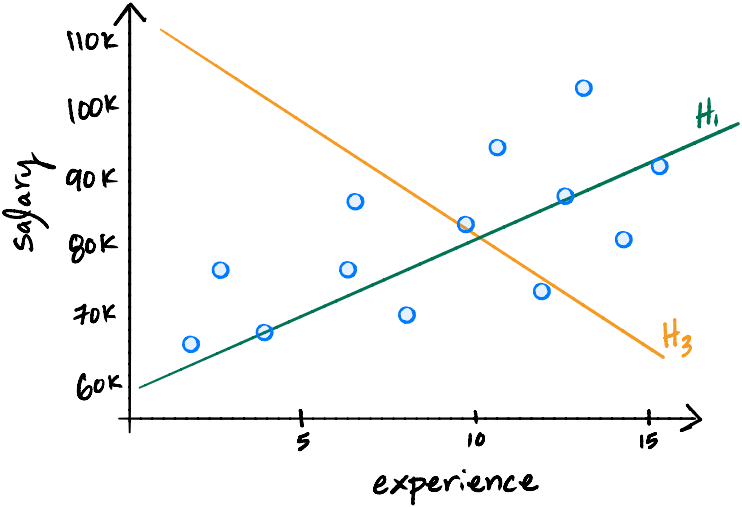


Mean Squared Error

- **Expected square loss** (mean squared error):

$$R_{\text{sq}}(H) = \frac{1}{n}\sum_{i=1}^{n}(H(\vec{x}^{(i)}) - y_i)^2$$

- This is the empirical risk for the square loss.
- Goal: find H minimizing MSE.


110K
100K
90K
80K
70K
60K
salary
5
10
15
experience
H_1
H_3

Step #3: Minimize MSE

- We want to find an H minimizing this:

$$R_{\text{sq}}(H) = \frac{1}{n}\sum_{i=1}^{n}(H(\vec{x}^{(i)}) - y_i)^2$$

- It helps to use linear assumption:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i)^2$$

Calculus

- We want to find $\vec{w}$ that minimizes:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i)^2$$

- Take the gradient, set to $\vec{0}$, solve.

- Solution: the **Normal Equations**, $\vec{w} = (X^tX)^{-1}X^t\vec{y}$

Design Matrix

- X is the **design matrix** X:

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \longrightarrow \\ \text{Aug}(\vec{x}^{(2)}) \longrightarrow \\ \vdots \quad \vdots \\ \text{Aug}(\vec{x}^{(n)}) \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

Note

- There was a closed-form solution!
- This is a direct consequence of using the **mean squared error**.
- Not true if we use, e.g., the **mean absolute error**.

Why linear?

- Easy to work with mathematically.
- Harder to overfit.
- But still quite powerful.

DSC 190

Machine Learning: Representations

Lecture 2 | Part 4

Linear Classification

Movie Ratings

- Five of your friends rate a movie from 0-10:
  - x_1: 9
  - x_2: 3
  - x_3: 7
  - x_4: 2
  - x_5: 8
- **Task**: Will you like the movie? (yes / no)

Classification

- Linear prediction functions can be used in classification, too.

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

- Same ERM paradigm also useful.

Classification

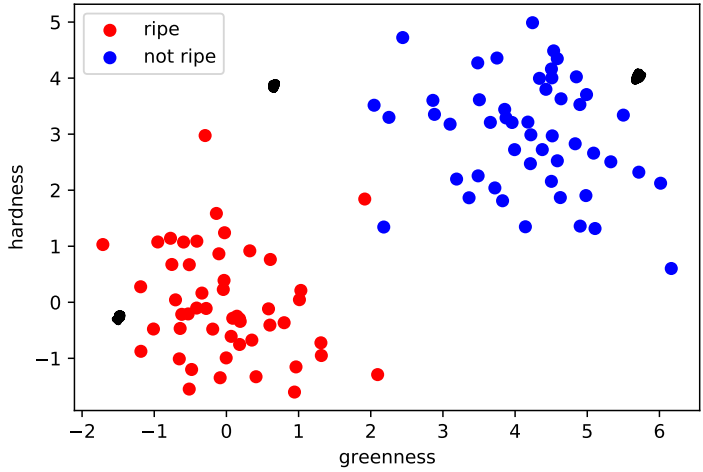
- In classification, the output of the prediction function is a discrete **label**.
- Our linear prediction functions output real numbers.
- But we can use a linear prediction function as a classifier by, e.g., thresholding.

Example: Mango Ripeness

- Predict whether a mango is ripe given greenness and hardness.

- Idea: gather a set of labeled **training data**.
  - Inputs along with correct output (i.e., “the answer”).

Greenness	Hardness	Ripe
0.7	0.9	1
0.2	0.5	-1
0.3	0.1	-1
⋮	⋮	⋮


ripe
not ripe
hardness
greenness
5
4
3
2
1
0
−1
−2
−1
0
1
2
3
4
5
6

A Classifier from a Regressor

- Binary classification can be thought of as regression where the targets are 1 and -1
  - (or 0 and 1, or ...)

- $H(\vec{x})$ outputs a real number. Use the **sign** function to turn it into –1, 1:

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

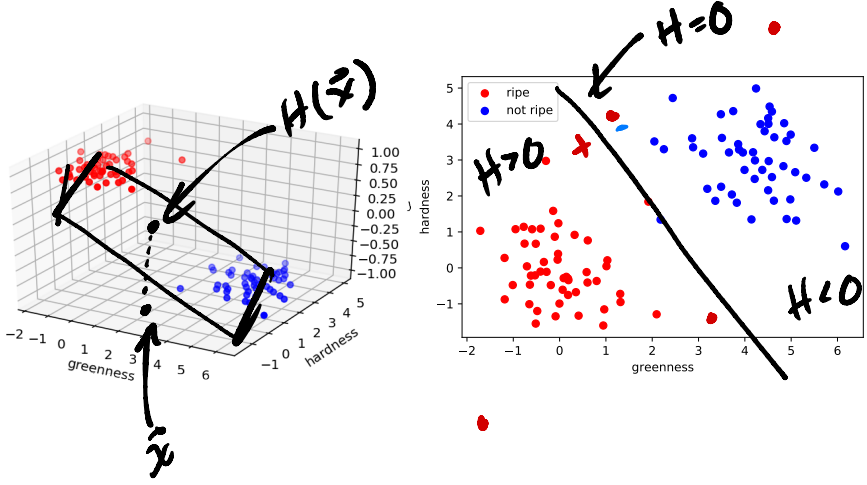
- Final prediction: $\text{sign}(H(\vec{x}))$

Decision Boundary

- The **decision boundary** is the place where the output of $H(x)$ switches from "yes" to "no".
  - If $H > 0 \mapsto$ "yes" and $H < 0 \mapsto$ "no", the decision boundary is where $H = 0$.

- If H is a linear predictor and[6]
  - $\vec{x} \in R^1$, 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.

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


H(x)
x
greenness
hardness
H=0
H>0
H<0
ripe
not ripe
greenness
hardness

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)**

Exercise

Can we use the square loss for classification?

$$(H(\vec{x}^{(i)}) - y_i)^2$$

Least Squares and Outliers

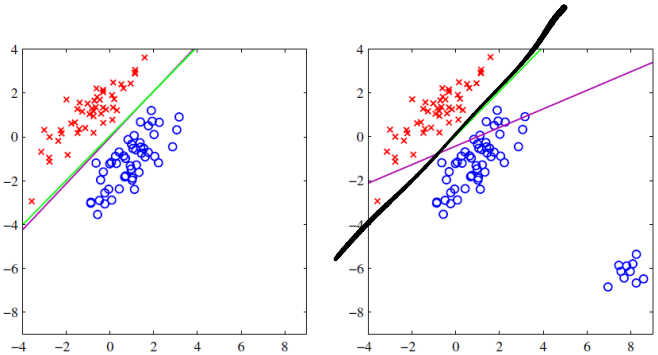


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding 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.

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

Loss Functions for (Binary) Classification

- $H(\vec{x})$ is prediction, $y_i \in \{-1, 1\}$ is correct answer
- Another loss function: **0-1 loss**.

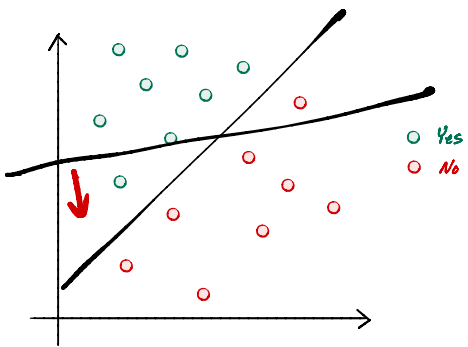
$$L(H(\vec{x}^{(i)}, y_i)) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

- Expected 0-1 loss:

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

0-1 Loss

- The expected 0-1 loss is simply proportion of misclassified points.



Problem

- The 0-1 loss is not differentiable.
- Can’t even use gradient descent...
- In fact, NP-Hard to optimize expected 0-1 loss in general.

Perceptron Loss

- The **perceptron loss** is designed for binary classification.

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

- Empirical expected perceptron loss (risk):

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

Perceptron Training

- There's no closed form solution for the $\vec{w}$ that minimizes expected perceptron loss.
  - Unlike expected square loss.

- Train iteratively using gradient descent.

Perceptrons

- A **perceptron** is a linear prediction function trained using the perceptron loss.

Loss Functions

- There are many different loss functions for classification.
- Each leads to a different classifier:
  - Logistic Regression
  - Support Vector Machine
  - Perceptron
  - etc.
- But that's for another class...

DSC 190

Machine Learning: Representations

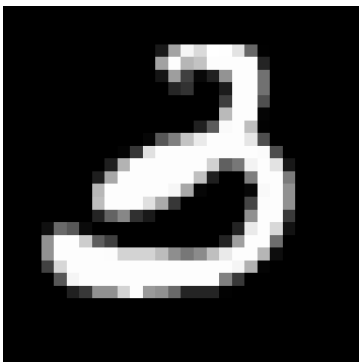
Lecture 2 | Part 5

Classification Demo: MNIST

Demo: MNIST

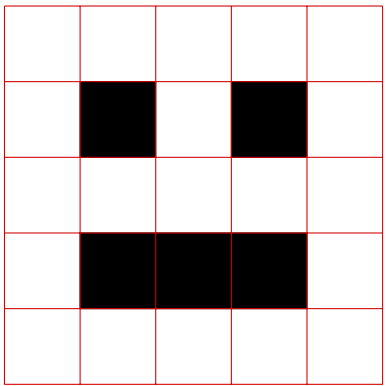
- MNIST is a classic machine learning data set.
- Many images of handwritten digits, 0-9.
- Multiclass classification problem.
- But we can make it binary: 3 vs. 7.

Example MNIST Digit

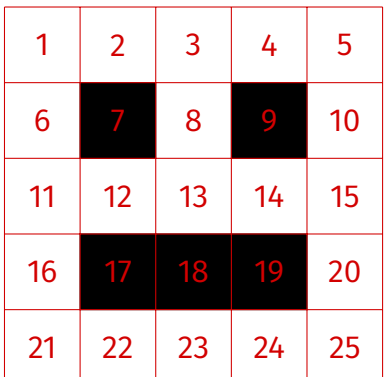


- Grayscale
- 28 x 28 pixels

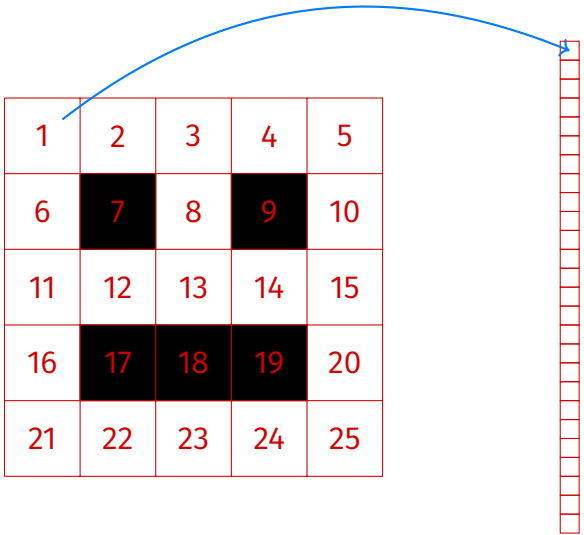
Images as Vectors



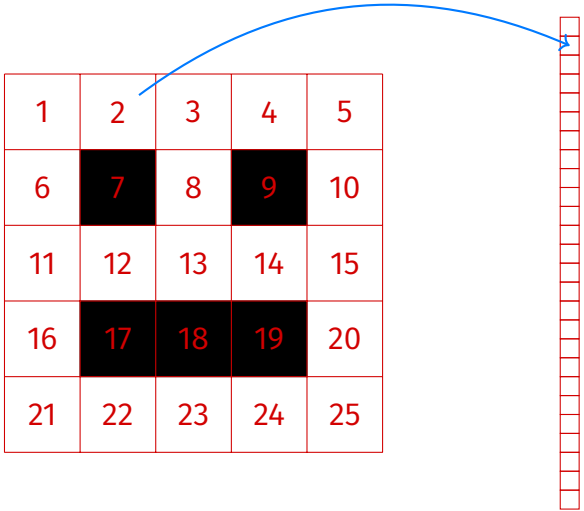
Images as Vectors



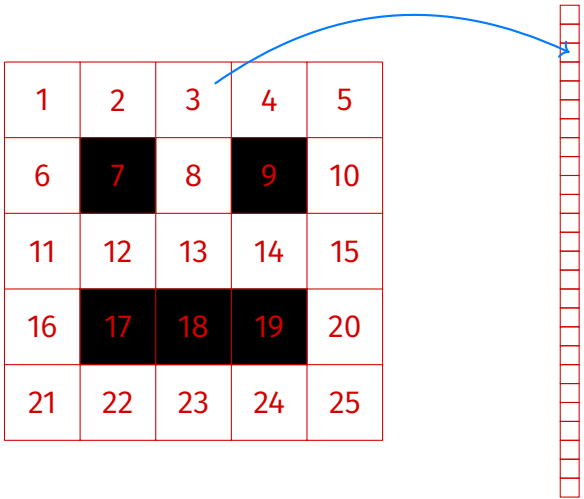
Images as Vectors



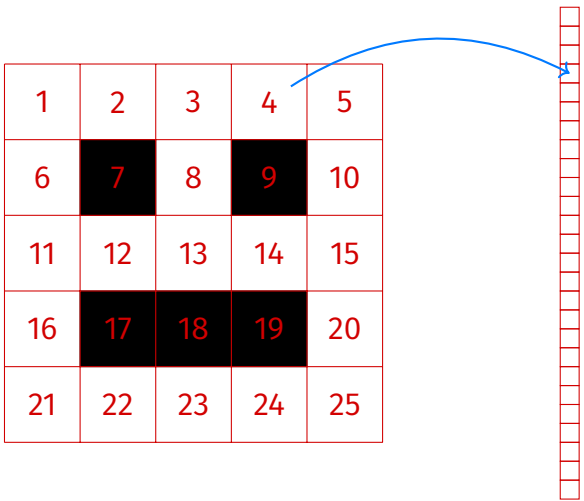
Images as Vectors



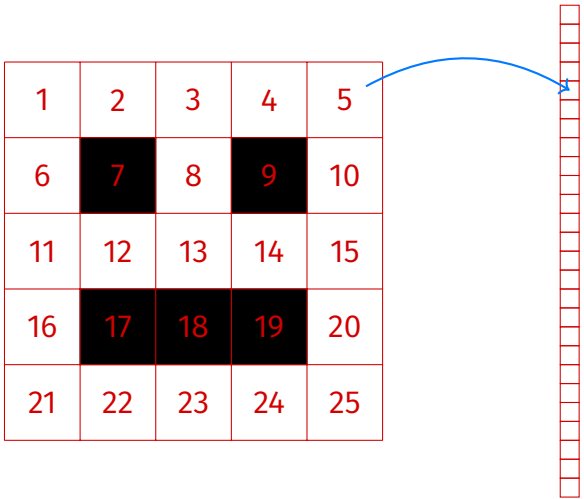
Images as Vectors



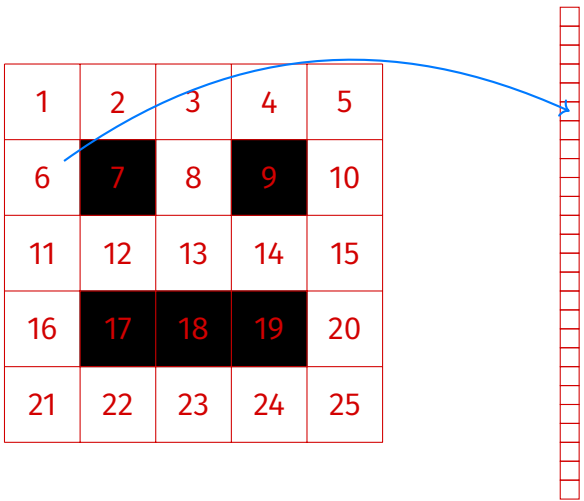
Images as Vectors



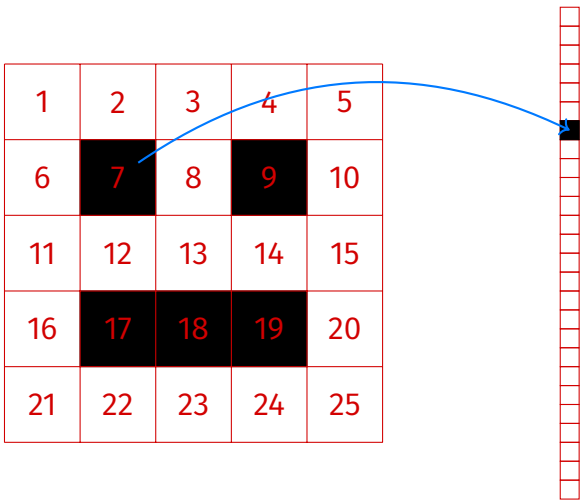
Images as Vectors



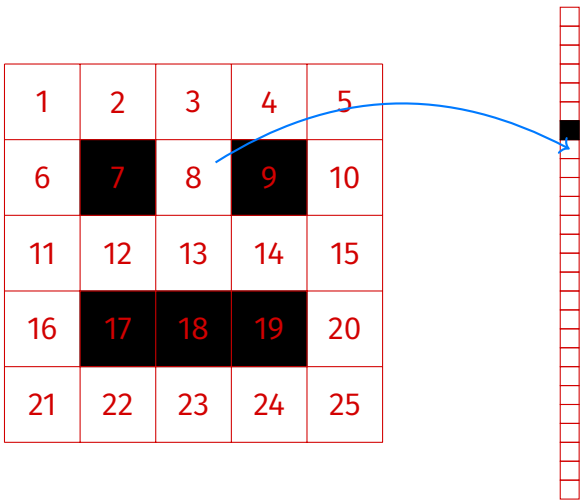
Images as Vectors



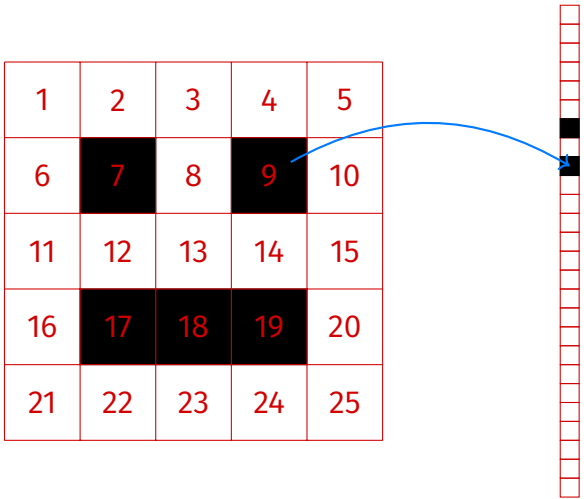
Images as Vectors



Images as Vectors



Images as Vectors



Images as Vectors

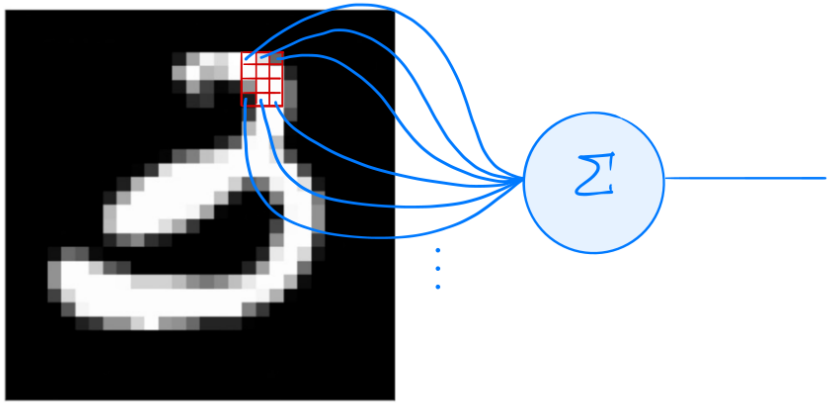
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

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.


Σ

Percepton 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

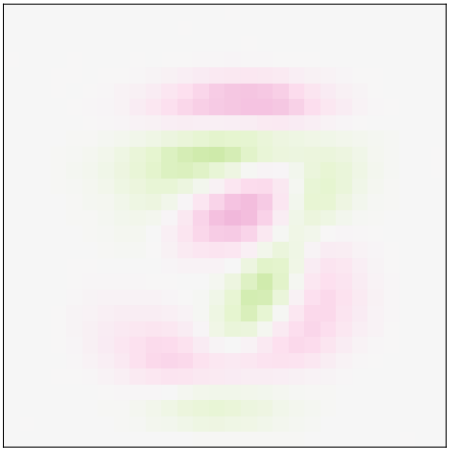
- Recall that the prediction is a **weighted vote**:

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

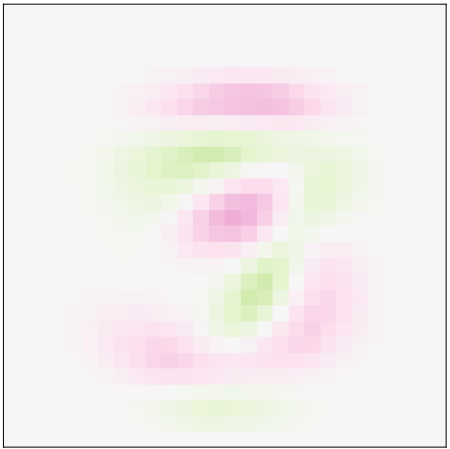
- Positive $\rightarrow$ 7, Negative $\rightarrow$ 3

- 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

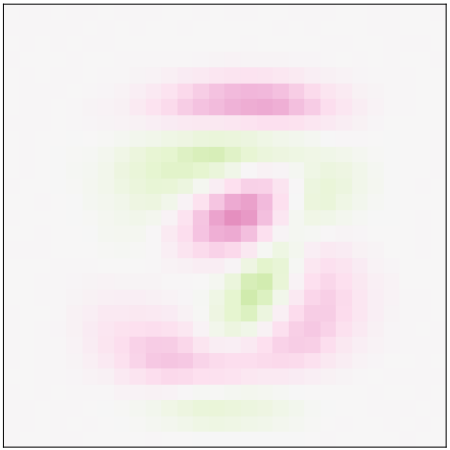
Perceptron Training



Perceptron Training



Perceptron Training



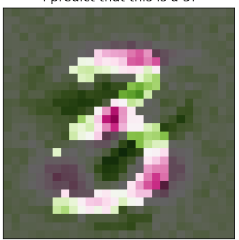
Perceptron Training

Perceptron Training

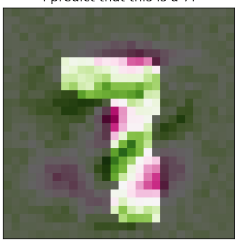
Perceptron Training

Perceptron Weight Vector

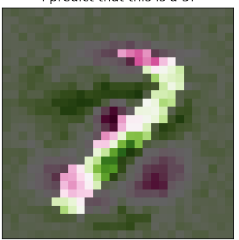
I predict that this is a 3!



I predict that this is a 7!



I predict that this is a 3!



Perceptron Results

- Test accuracy: 97.3%

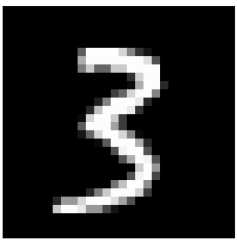
Square Loss for Classification

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

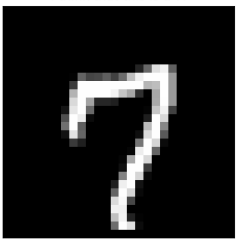
Results: Least Squares

- Test Accuracy: 96.7% (marginally worse)

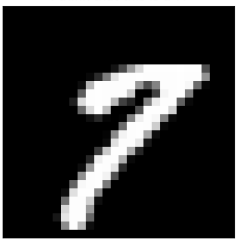
I think that this is a 3.



I think that this is a 7.



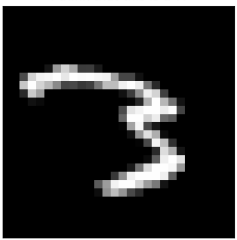
I think that this is a 7.



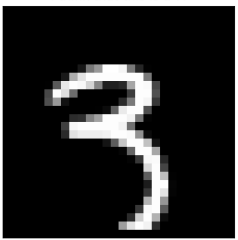
Results: Least Squares

- Misclassifications are telling.

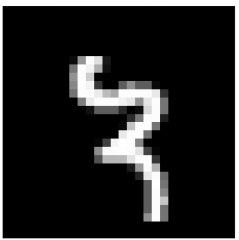
I think that this is a 7.



I think that this is a 7.

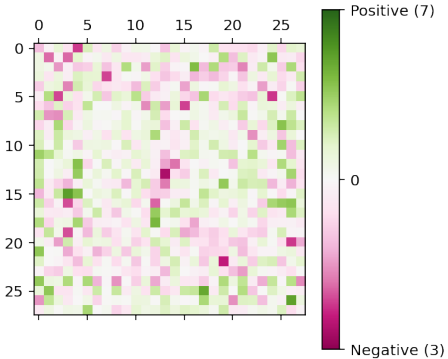


I think that this is a 7.



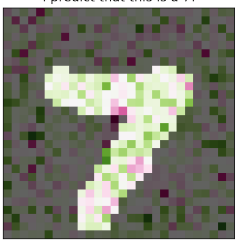
Least Squares Weight Vector

- Can visualize weight of each pixel as an image.



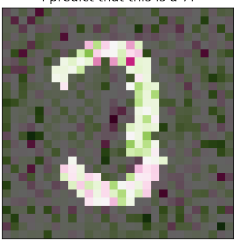
Least Squares Weight Vector

I predict that this is a 7!



I predict that this is a 3!

I predict that this is a 7!

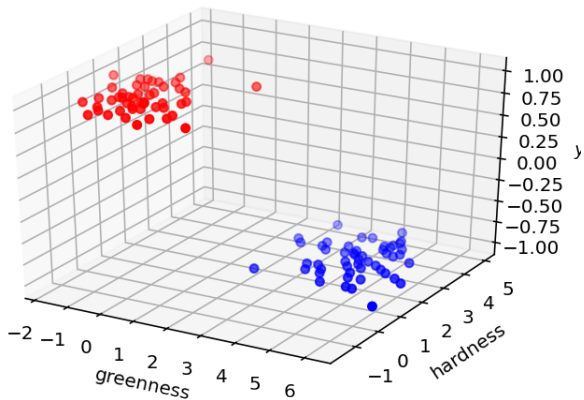


Hack

- Idea: binary classification is just a special case of regression, where output is either -1 or 1.
- We can still minimize mean squared error.
- Somewhat strange: penalizes “very correct” predictions.

$$y_i = 1 \qquad H(\vec{x}^{(i)}) = 3 \qquad H(\vec{x}^{(i)}) = -1$$

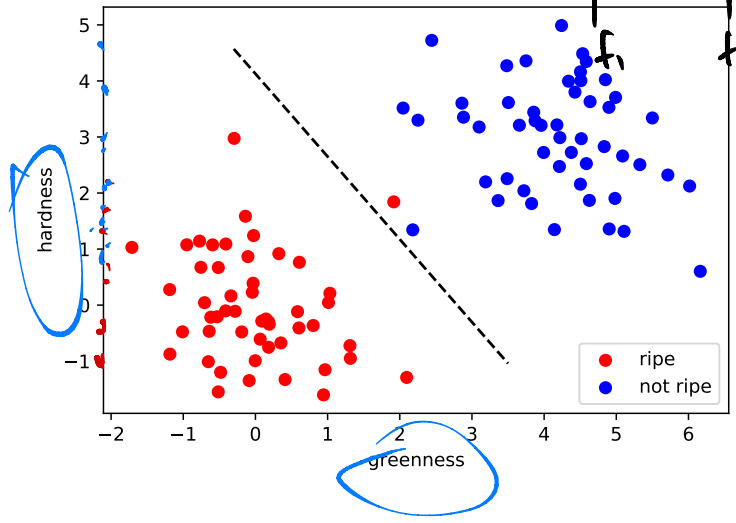
Classification as Regression



```
>>> # assume X is augmented design matrix
>>> # y is array of labels (1 = ripe, -1 = not ripe)
>>> w = np.linalg.lstsq(X, y)
>>> w
array([ 0.78442482, -0.28064357, -0.19005338])
```

That is,

$$H(\vec{x}) = 0.78 - 0.28 \times \text{greenness} - 0.19 \times \text{hardness}$$



$1 + w_1 x + w_2 x^2 + w_3 x^3$

f_1 f_2 f_3

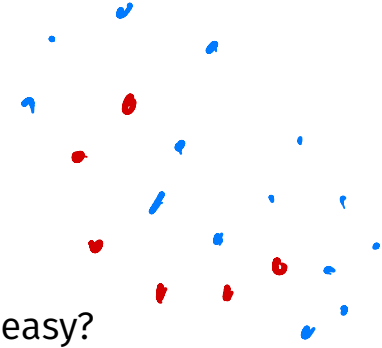
$H(x)$

x

Comments

- Binary classification in MNIST is **easy**.
  - Multiclass is a lot harder.
- If the problem is easy, linear prediction functions work well.
- And it doesn't matter so much which loss we use.

Next



- What if the problem isn’t so easy?