DSC 140B

Representation Learning

Lecture 16 | Part 1

Laplacian Eigenmaps

Problem: Graph Embedding

- **Given**: a similarity graph, target dimension k
- **Goal**: **embed** the nodes of the graph as points in $\mathbb{R}^k$ so that similar nodes are nearby
- **(One) Solution:** Embed using eigenvectors of the graph Laplacian

Problem: Non-linear Dimensionality Reduction

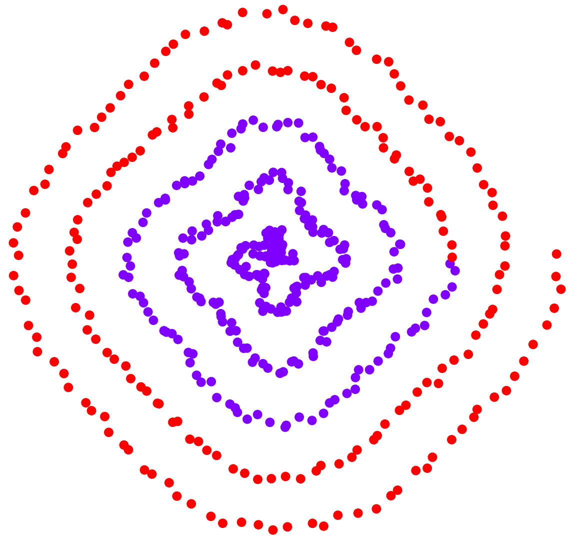
- **Given**: points in $\mathbb{R}^d$, target dimension k
- **Goal**: **embed** the points in $\mathbb{R}^k$ so that points that were close in $\mathbb{R}^d$ are close after

Idea

- Build a similarity graph from points in $\mathbb{R}^d$
  - epsilon neighbors, k-neighbors, or fully connected

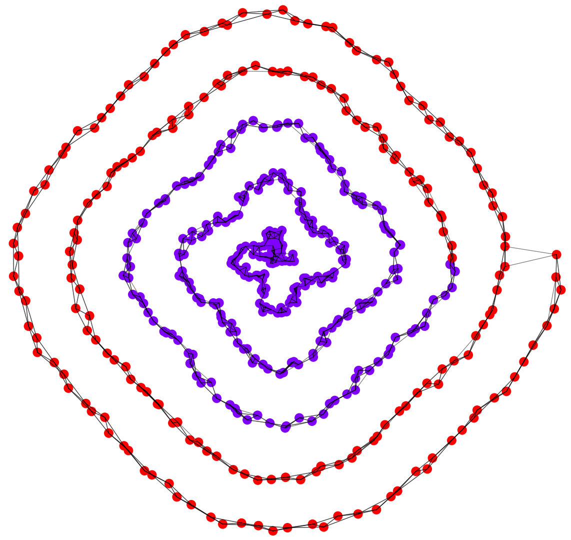
- Embed the similarity graph in $\mathbb{R}^k$ using eigenvectors of graph Laplacian

Example 1: Spiral



Example 1: Spiral

- Build a k-neighbors graph.
- Note: follows the 1-d shape of the data.

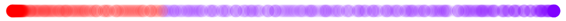


Example 1: Spectral Embedding

- Let W be the weight matrix (k-neighbor adjacency matrix)
- Compute $L = D - W$
- Compute bottom k non-constant eigenvectors of L, use as embedding

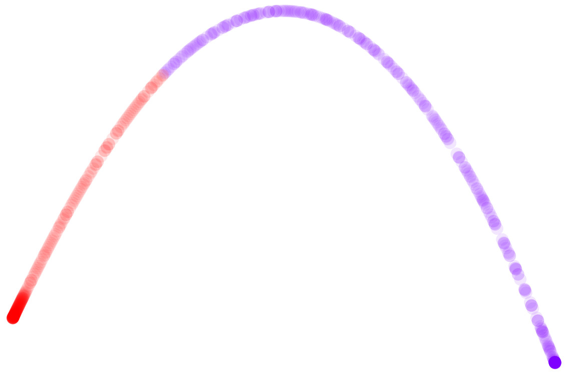
Example 1: Spiral

- Embedding into $\mathbb{R}^1$



Example 1: Spiral

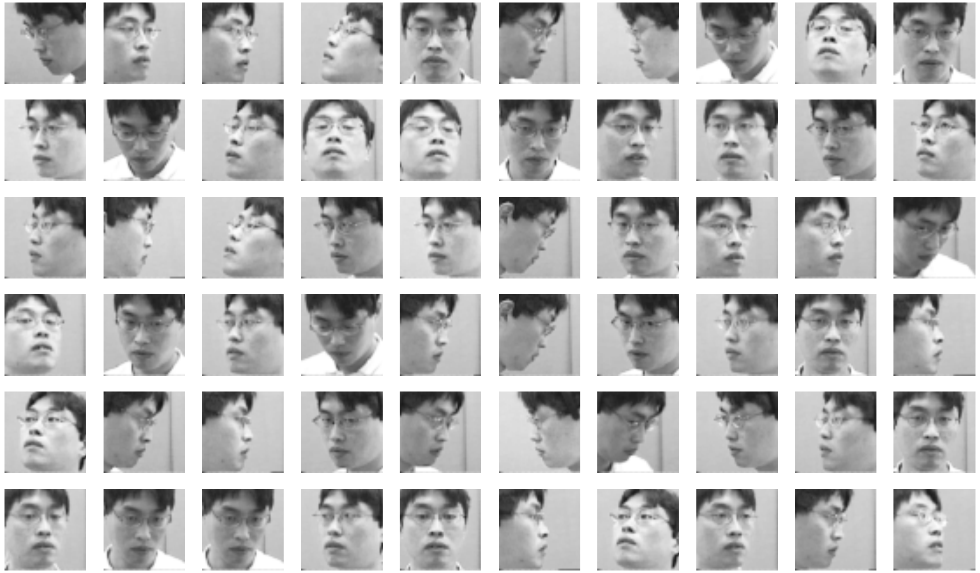
- Embedding into $\mathbb{R}^2$



Example 1: Spiral

```python
import sklearn.neighbors
import sklearn.manifold
W = sklearn.neighbors.kneighbors_graph(
        X, n_neighbors=4
)
embedding = sklearn.manifold.spectral_embedding(
        W, n_components=2
)
```

Example 2: Face Pose

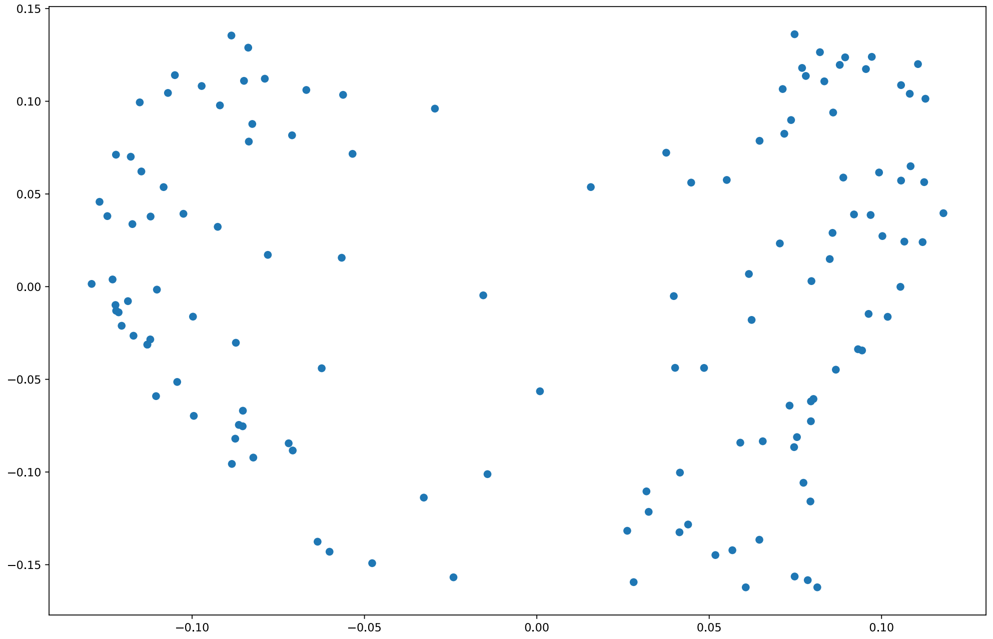


Example 2: Face Pose

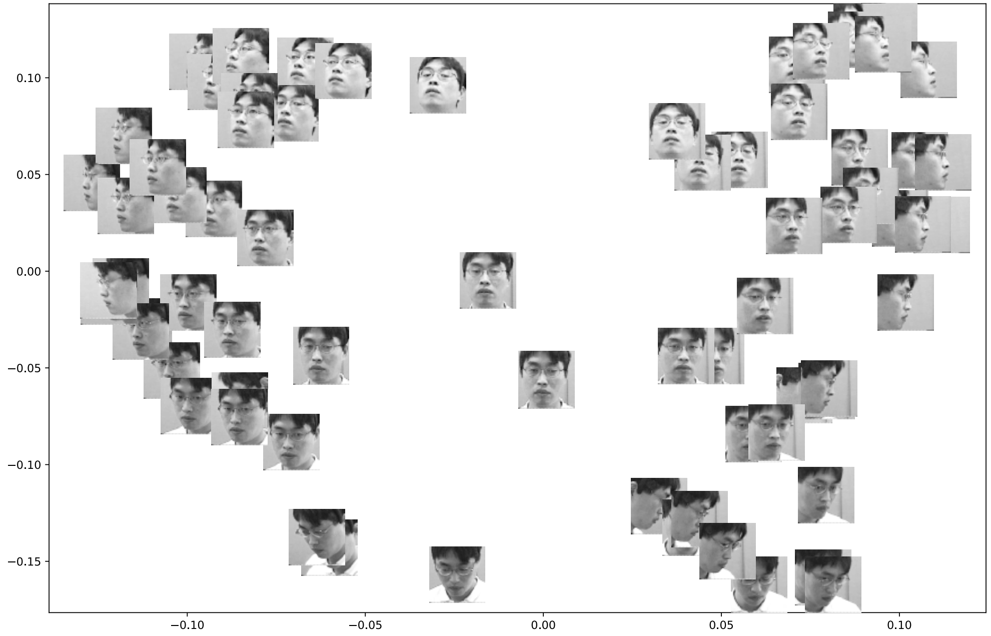
- Construct fully-connected similarity graph with Gaussian similarity

- Embed with Laplacian eigenmaps

Example 2: Face Pose



Example 2: Face Pose



DSC 140B
Representation Learning

Lecture 16 | Part 2

Prediction

So far...

- **Unsupervised** representation learning
  - Reveals the “structure” of the data
  - Dimensionality reduction and manifold learning
  - Useful for clustering, visualization, etc.

- What about **supervised** representation learning?
  - Suppose we want a representation that makes prediction easier.
  - PCA, etc., *can* help, but don’t consider labels.

What’s ahead...

1. We’ll remember a linear predictor from DSC 40A
    - Least squares regression / classification

2. We’ll see how using a different representation can make prediction easier

3. We’ll introduce **deep learning** as a way of finding representations optimized for prediction

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) $\rightarrow$ **binary**
  - Example: species (cat, dog, mongoose) $\rightarrow$ **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.

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

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

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

Prediction Functions

- **Problem**: There are **infinitely many** prediction functions.
  - $H_1(\vec{x}) = \underbrace{-2x_1 + 3x_5}$
  - $H_2(\vec{x}) = \sin(x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5)$
  - $H_3(\vec{x}) = \sqrt{x_1 + x_3}(x_1 - x_2x_5 + 100)$
  - ...[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	?

$\}$ data

$\mapsto$

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** $\{H\}$
- 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.

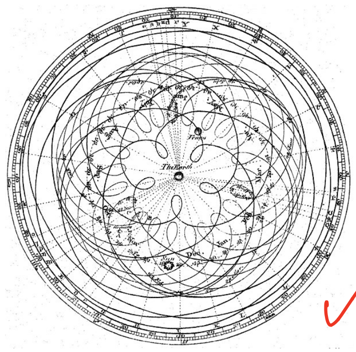
- By choosing a hypothesis class, we are saying something about what the prediction function should look like.

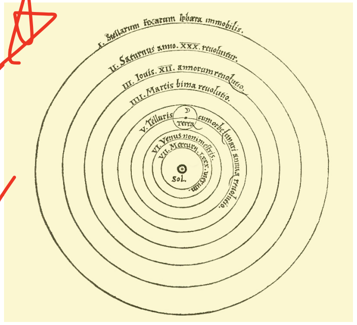
- Examples:
  - $\mathcal{H}$:= linear functions
  - $\mathcal{H}$:= functions of the form $\sin(w_1 x_1 + \ldots x_5 x_5)$
  - $\mathcal{H}$:= decision trees of depth 10
  - $\mathcal{H}$:= neural networks with one layer

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.




I. Stellarum Fixarum ſphæra immobilis.
II. Saturnus anno. XXX. revolvitur.
III. Iovis. XII. annorum revolutio.
IIII. Martis bima revolutio.
V. Telluris cum orbe lunari annua revolutio.
Terra
VI. Venus nonimestris.
VII. Mercury. LXXX. dierum.
Sol.

DSC 140B

Representation Learning

Lecture 16 | Part 3

Least Squares Regression

A Simple Prediction Function

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

A Simple Prediction Function

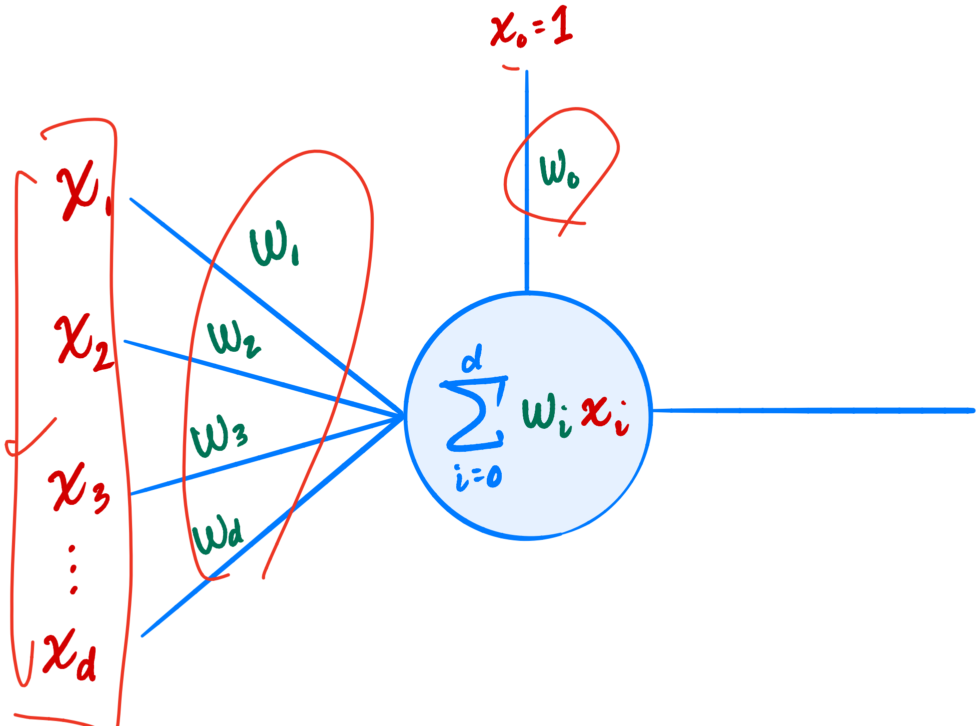
- Five of your friends rate a movie from 0-10:
  - x_1: 9
  - x_2: 3
  - x_3: 7
  - x_4: 2
  - x_5: 8
- Idea: predict a **weighted sum**.

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

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

function $\iff$ w

"Parameterization"

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

$w^* \in \mathbb{R}^6$

In General

weights

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

$$H(\vec{x}) = \underset{\text{bias}}{\boxed{w_0}} + \underline{w_1} x_1 + \underline{w_2} x_2 + \ldots + \underline{w_d} x_d$$

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

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 $\vec{x}$:

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

Simplification

- With augmentation, we can write as dot product:

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

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

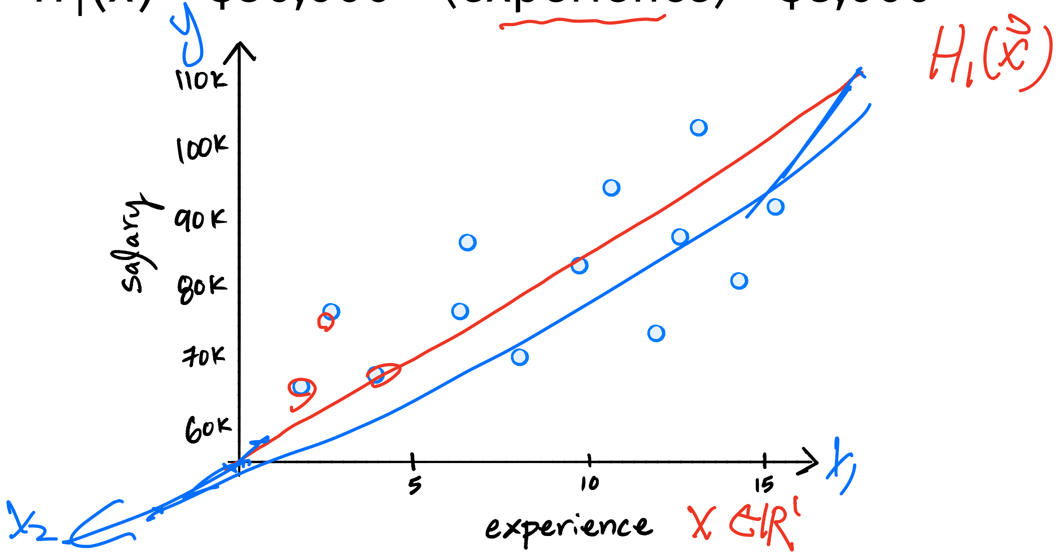
Geometric Meaning

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

Example

- A linear prediction function for salary.

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



Surface

- 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[4]
  - $\vec{x} \in R^1$, 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**.

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

Loss Function

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

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

$$L(\text{prediction for example } i, \text{correct answer for example } i)$$

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:

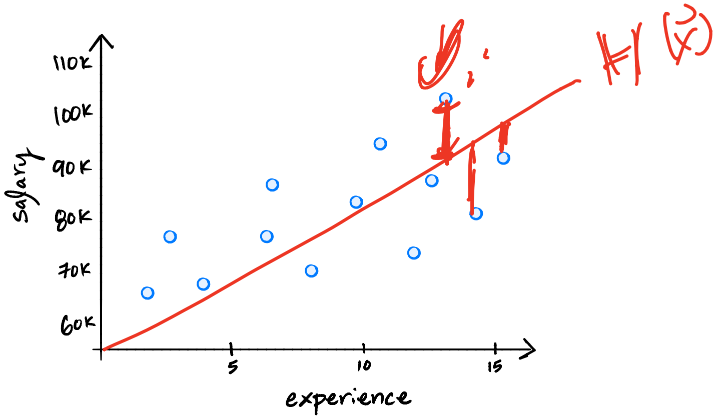
$$R(H) = \frac{1}{n} \underline{\sum_{i=1}^{n} L(H(\vec{x}^{(i)}), y_i)}$$

$$\mathbb{E}_{x \sim \tilde{P}_d(x)} \left[L(H(\vec{x}), y) \right]$$

- Note: depends on H and the data!

Loss Functions for Regression

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



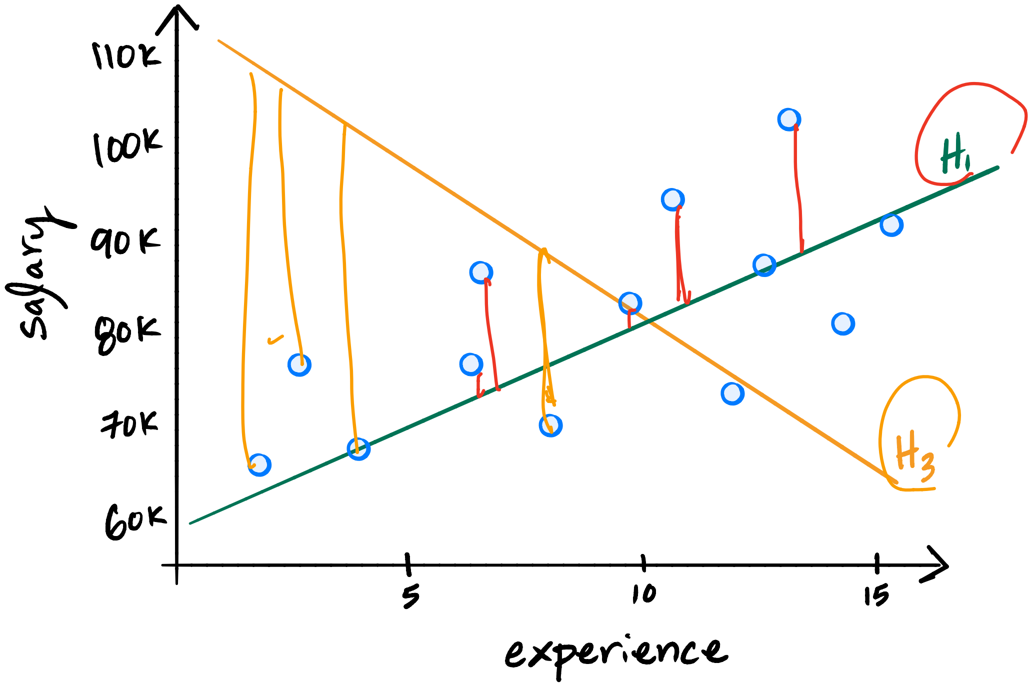
Mean Squared Error

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

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

- 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:

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

$\vec{w} \cdot \text{Aug}(\vec{x}^{(i)})$

- 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

$X \underset{n}{=} \underset{d+1}{\begin{bmatrix} \text{---} \\ \text{---} \end{bmatrix}}$

- We want to find $\vec{w}$ that minimizes the average square loss:

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

$\vec{w}$

$$\frac{\partial R_{sq}(\vec{w})}{\partial \vec{w}} = 0$$

$\vec{w}^*$

- $\underline{\text{Take the gradient,}}$ set to $\vec{0}$, solve.
- Solution: the **Normal Equations**, $\vec{w}$ $\underline{= (X^t X)^{-1} X^t \vec{y}}$ ✓

Design Matrix

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

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

n

$\cancel{d}+1$

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**.

[illegible]

Why linear?

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

DSC 140B
Representation Learning

Lecture 16 | Part 4

Least Squares Classifiers

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.

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 ...)

$z = H(\vec{x})$

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

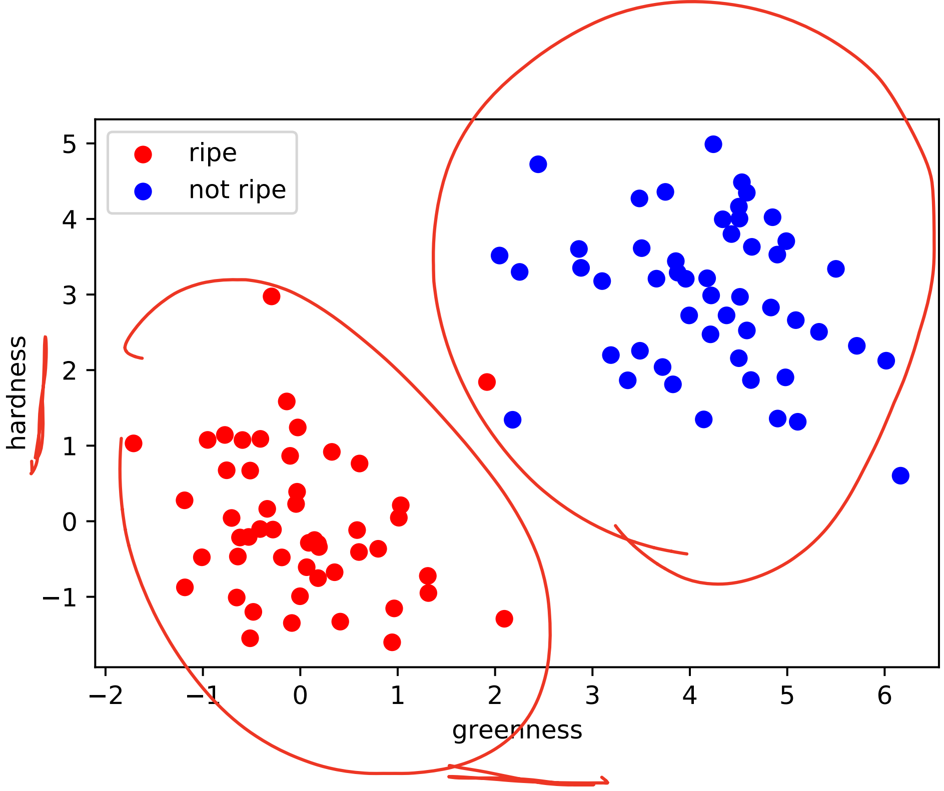
$$\text{sign}(z) = \begin{cases} 1 & \underline{z > 0} \\ -1 & \underline{z < 0} \\ 0 & \text{otherwise} \end{cases}$$

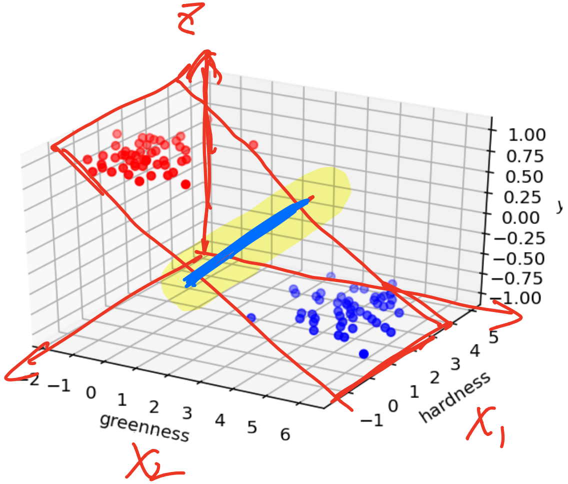
- ~~Final prediction:~~ $\boxed{\text{sign}(H(\vec{x}))}$

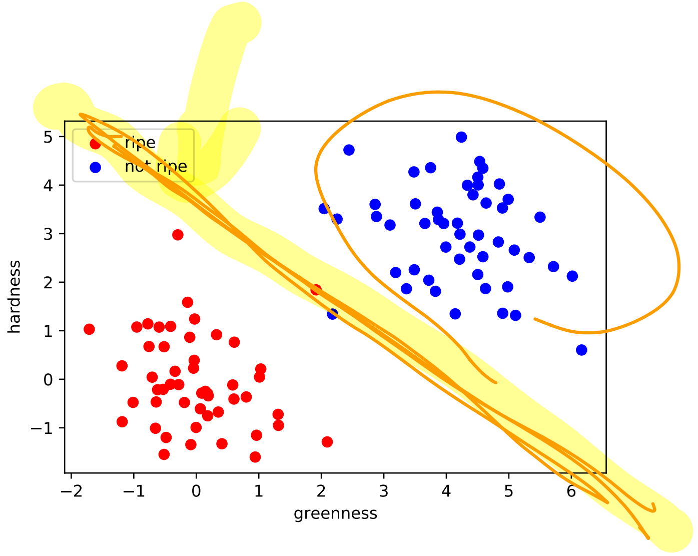
Example: Mango Ripeness

- 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

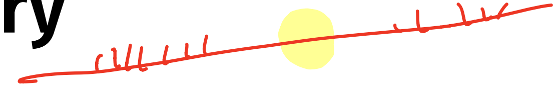




$$H(\underset{\sim}{x}) = w_1 x_1 + w_2 x_2 + w_0$$

$$\text{Sign}(H(\underset{\sim}{x}))$$

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[5]
  - $\vec{x} \in R^1$, 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.

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

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

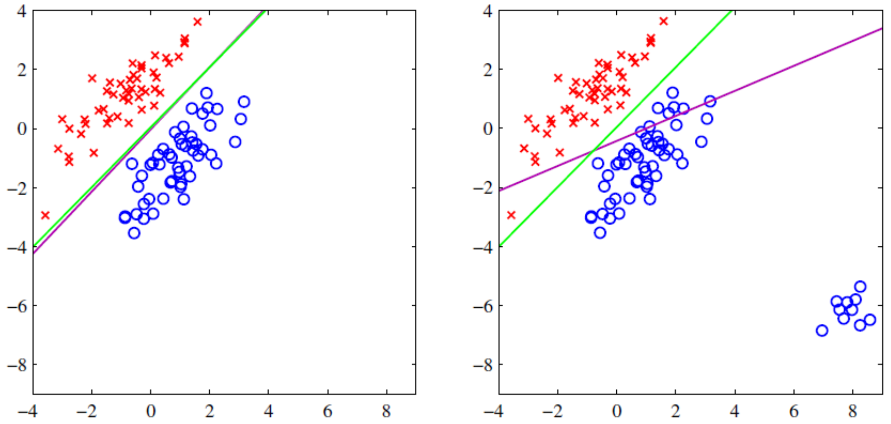


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding 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. [6]

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

Square Loss for Classification

- We **can** use the square loss for classification
  - The “least squares classifier”
- However, the square loss penalizes being “too correct”
- **Example**: suppose the correct label is 1. What is the square loss of predicting 10? -9?

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 140A)