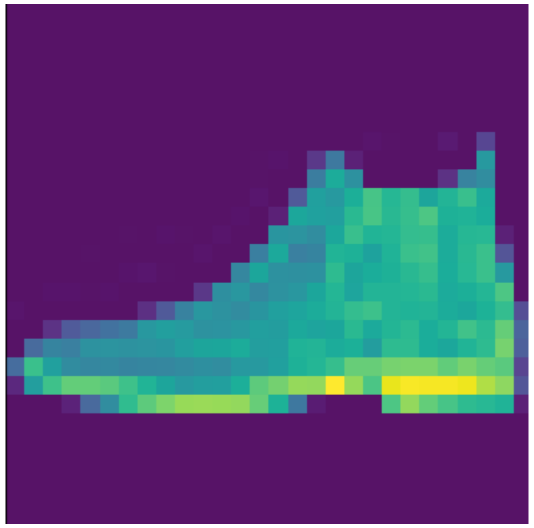
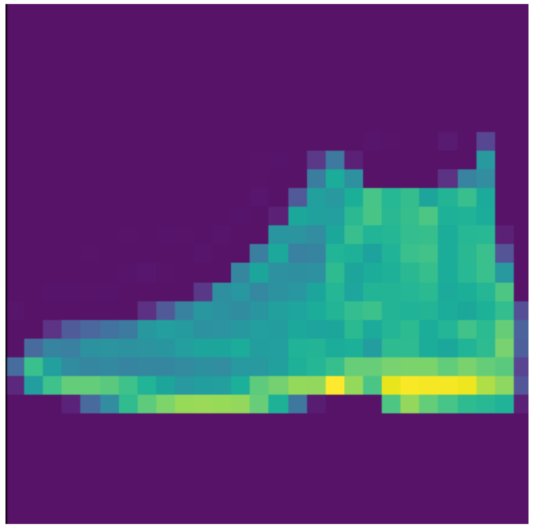
CS 4342: Class 1

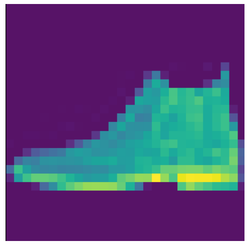
Jacob Whitehill





Fashion MNIST

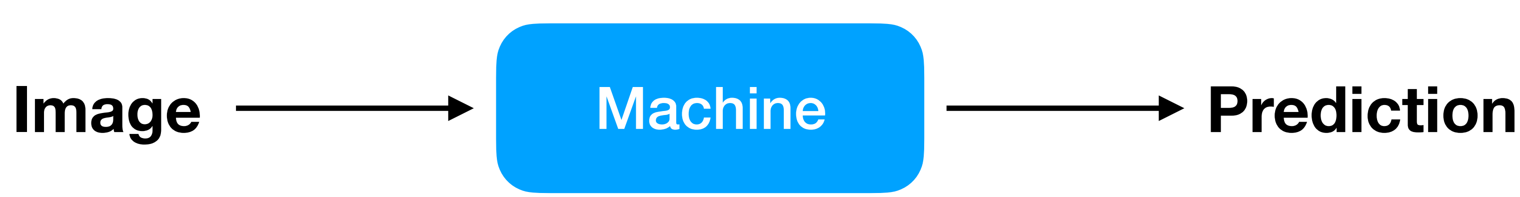
Fashion MNIST



- Fashion MNIST is a toy machine learning (ML) dataset to test the viability of new ML models & algorithms.
- It consists of 60K images, each of which is one of 10 different types of clothing: **t-shirt**, **boot**, **sandal**, etc.

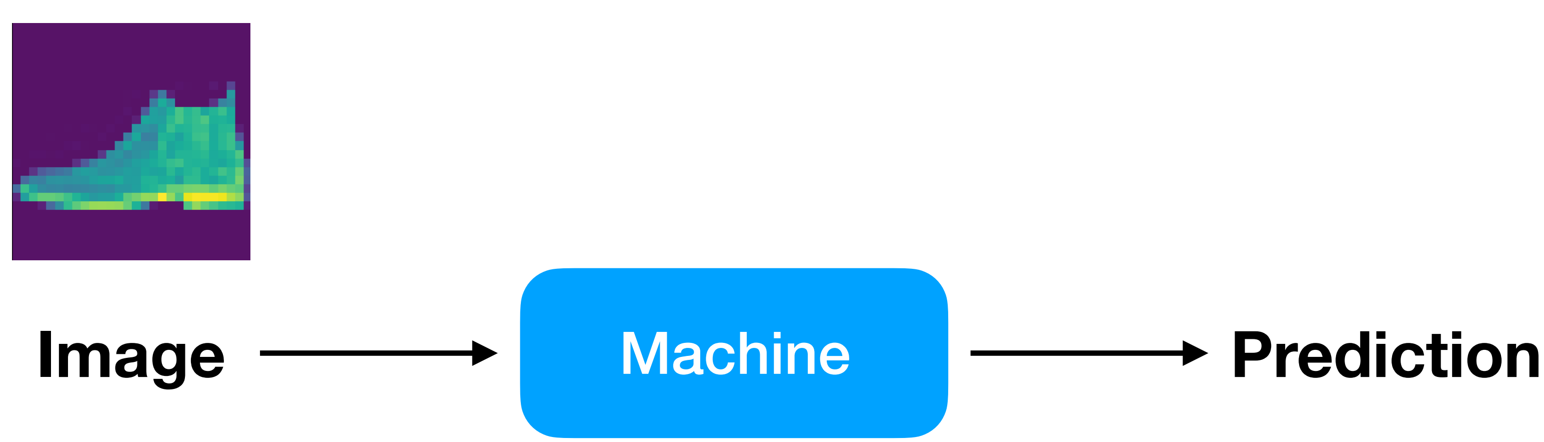
Fashion MNIST

- Using machine learning, we can create a “machine” to classify automatically which of 10 types of clothing is contained within an input image (28x28 pixels):



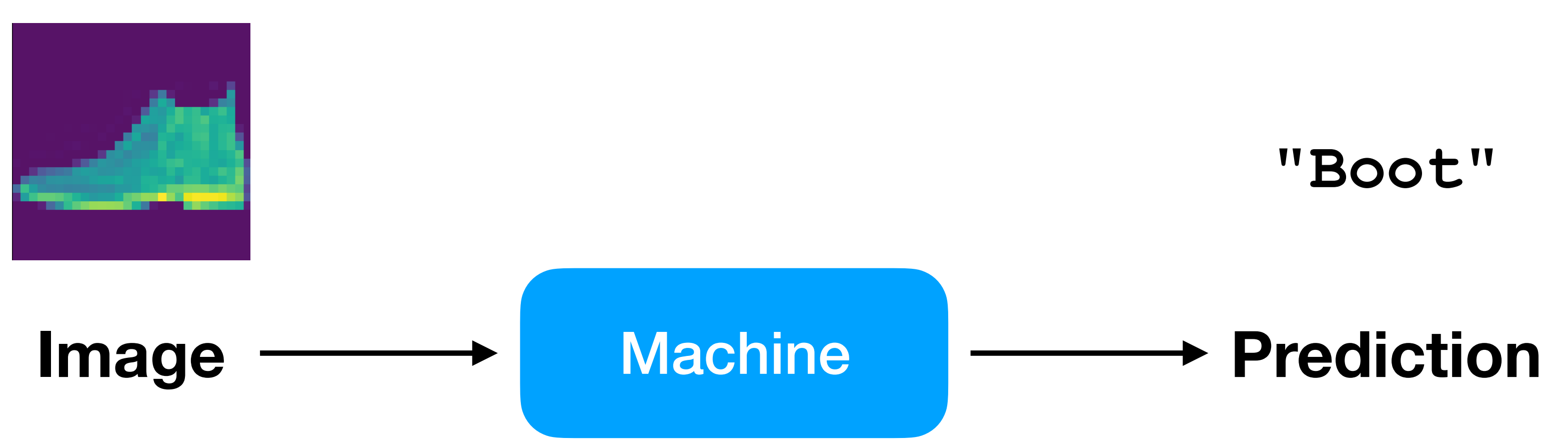
Fashion MNIST

- Using machine learning, we can create a “machine” to classify automatically which of 10 types of clothing is contained within an input image (28x28 pixels):



Fashion MNIST

- Using machine learning, we can create a “machine” to classify automatically which of 10 types of clothing is contained within an input image (28x28 pixels):



Fashion MNIST

- Using machine learning, we can create a “machine” to classify automatically which of 10 types of clothing is contained within an input image (28x28 pixels):

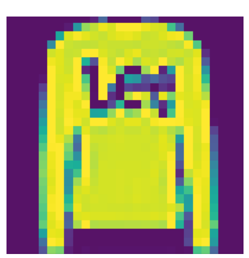
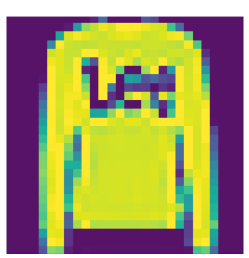


Image → Machine → **Prediction**

Fashion MNIST

- Using machine learning, we can create a “machine” to classify automatically which of 10 types of clothing is contained within an input image (28x28 pixels):



"Shirt"

Image → Machine → **Prediction**

Fashion MNIST

- It turns out that such machines can be constructed by surprisingly simple components, e.g.:
  - Matrix multiplication
  - Matrix addition
  - Maximum
  - Exponentiation
  - Normalization

Fashion MNIST

- In fact, the machine's **entire behavior** can be expressed as:

  ```
  normalize(exp(W3.dot(maximum(0, W2.dot(maximum(0, W1.dot(x) + b1)) + b2)) + b3))
  ```

  where **`x`** is the input image and **`W1`**, **`W2`**, **`W3`**, **`b1`**, **`b2`**, **`b3`** are matrices of real numbers.

- Demo.

Fashion MNIST

- The question that remains is…

Fashion MNIST

- The question that remains is…

Where do those matrices of numbers come from, and how do we know if they're "good"?

Fashion MNIST

- The question that remains is…

Where do those matrices of numbers come from, and how do we know if they're "good"?

- The next 7 weeks will be your journey to answer this question.

(-; Faces :-)

How old are these people?

Guess how old each person is based on their face image.

a

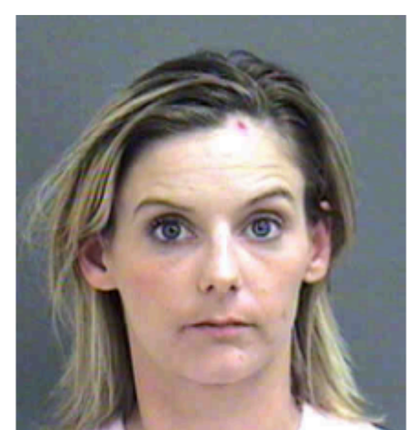
b

c

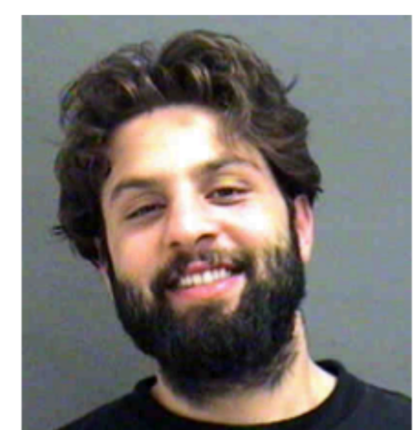
d

e

f



g



h

https://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_01299.pdf

Whose guesses were the best?

- Who of you was best at guessing the people's ages?
- How do we define "best"?
- We need some kind of **accuracy function.**

How to measure accuracy?

- Suppose there are *n* faces.
- Let $\mathbf{y} \in \mathbb{R}^n$ be an *n*-vector of the **ground-truth** age values.
- Let $\hat{\mathbf{y}} \in \mathbb{R}^n$ be an *n*-vector of **guesses** for the age values.
- We write y_i and $\hat{y}_i$ for the *i*th ground-truth and guess (i=1, …, n), respectively.
- What kinds of functions $f(\hat{\mathbf{y}}, \mathbf{y})$ might we define to express the accuracy of the guesses w.r.t. ground-truth?

How to measure accuracy?

- Percent exactly Correct: $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]$

  where $\mathbb{I}[\cdot]$ equals 1 if the condition is true, and 0 otherwise.

  - **Higher** scores are better.

How to measure accuracy?

- Percent exactly Correct: $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]$

  where $\mathbb{I}[\cdot]$ equals 1 if the condition is true, and 0 otherwise.

  - **Example**:
    $\mathbf{y}$ = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [65, 19, 38, 60]
    $\hat{\mathbf{y}}^{(2)}$ = [5, 6, 2, 61]

$f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) =$ **?**

$f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) =$ **?**

How to measure accuracy?

- Percent exactly Correct: $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]$

  where $\mathbb{I}[\cdot]$ equals 1 if the condition is true, and 0 otherwise.

  - **Example**:
    $\mathbf{y}$ = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [65, 19, 38, 60]
    $\hat{\mathbf{y}}^{(2)}$ = [5, 6, 2, 61]

    $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) = 0.25$

    $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) = 0.25$

    Problem: It gives no consideration to the *distance* from ground-truth.

How to measure accuracy?

- Percent exactly Correct: $f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]$

  where $\mathbb{I}[\cdot]$ equals 1 if the condition is true, and 0 otherwise.

  - **Example**:
    $\mathbf{y}$ = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [65, 19, 38, 60]
    $\hat{\mathbf{y}}^{(2)}$ = [5, 6, 2, 61]

  - For a **regression** problem such as ours — in which we the ground-truth can be any real number — f_{PC} is almost never used.

How to measure accuracy?

- Average distance: $$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$
  - **Lower** scores are better.

How to measure accuracy?

- Average distance:

$$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} y_i}_{\textbf{average ground-truth}} - \underbrace{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i}_{\textbf{average guess}}$$

  - **Lower** scores are better.
  - The average distance compares the average guess to the average ground-truth.

How to measure accuracy?

- Average distance: $$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$
  - **Lower** scores are better. $$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} y_i}_{\textbf{average ground-truth}} - \underbrace{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i}_{\textbf{average guess}}$$
  - The average distance compares the average guess to the average ground-truth.
  - But this is still not a good metric. **Exercise**:
    $\mathbf{y}$ = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [?]
    $\hat{\mathbf{y}}^{(2)}$ = [?]

Find values for $\mathbf{y}^{(1)}$, $\mathbf{y}^{(2)}$ such that $\mathbf{y}^{(1)}$ is intuitively much better but they have equal average distances to y.

How to measure accuracy?

- Average distance: $$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$
  - **Lower** scores are better. $$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} y_i}_{\textbf{average ground-truth}} - \underbrace{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i}_{\textbf{average guess}}$$
  - The average distance compares the average guess to the average ground-truth.
  - But this is still not a good metric. **Exercise**:
    y = [66, 18, 38, 61]
    ŷ(1) = [65, 17, 39, 64] $\quad f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) \quad = \qquad$ **?**
    ŷ(2) = [5, 79, 34, 67] $\quad f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) \quad = \qquad$ **?**

How to measure accuracy?

- Average distance: $$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$
  $$= \underbrace{\frac{1}{n}\sum_{i=1}^{n} y_i}_{\text{average ground-truth}} - \underbrace{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i}_{\text{average guess}}$$
  - **Lower** scores are better.
  - The average distance compares the average guess to the average ground-truth.
  - But this is still not a good metric. **Exercise:**
    y = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [65, 17, 39, 64]
    $\hat{\mathbf{y}}^{(2)}$ = [5, 79, 34, 67]

$$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) = \frac{1}{4}(1 + 1 - 1 - 3) = -0.5 \text{ yr}$$
$$f_{\text{avgdist}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) = \frac{1}{4}(61 - 61 + 4 - 6) = -0.5 \text{ yr}$$

The avgdist does not consider large *individual* deviations between a guess $\hat{y}_i$ and ground-truth y_i.

How to measure accuracy?

- Mean absolute error: $f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$
    - **Lower** scores are better.

    - **Example**:
      y = [66, 18, 38, 61]
      $\hat{\mathbf{y}}^{(1)}$ = [65, 17, 39, 64]
      $\hat{\mathbf{y}}^{(2)}$ = [5, 79, 34, 67]

$f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) \quad =$ **?**

$f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) \quad =$ **?**

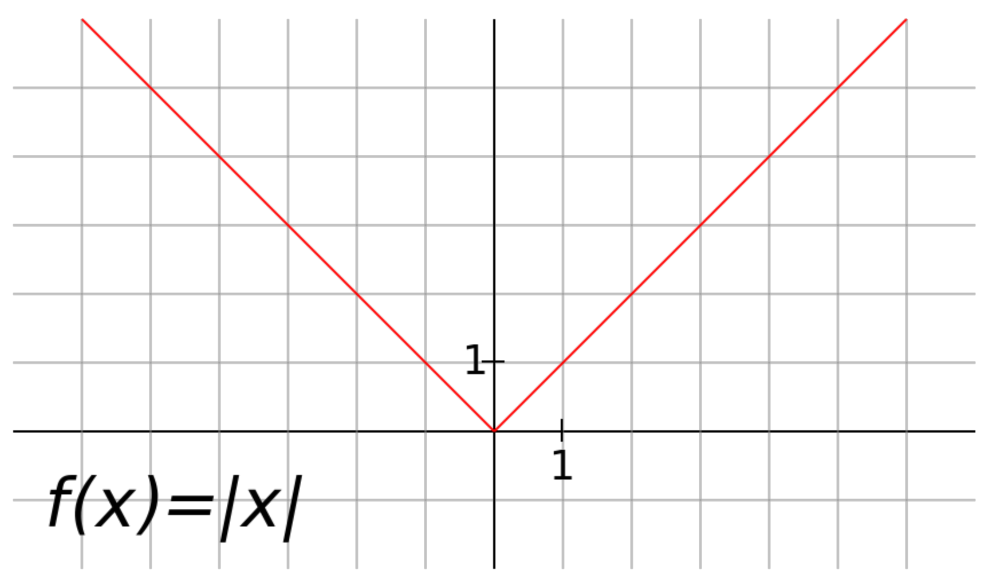
How to measure accuracy?

- Mean absolute error: $f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$
  - **Lower** scores are better.
  - **Example**:
    y = [66, 18, 38, 61]
    $\hat{\mathbf{y}}^{(1)}$ = [65, 17, 39, 64]
    $\hat{\mathbf{y}}^{(2)}$ = [5, 79, 34, 67]

$$
\begin{aligned}
f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) &= \frac{1}{4}(1+1+1+3) = 1.5 \text{ yr} \\
f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) &= \frac{1}{4}(61+61+4+6) = 33 \text{ yr}
\end{aligned}
$$

How to measure accuracy?

- Mean absolute error: $$f_{\mathrm{MAE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$
    - **Issue**: Absolute value is not differentiable at 0. This is not ideal as much of ML involves differential calculus.



https://commons.wikimedia.org/wiki/File:F(x)%3DAbs(x).svg

How to measure accuracy?

- Mean squared error: $f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - **Lower** scores are better.

$\mathbf{y}$ = [66, 18, 38, 61]
$\hat{\mathbf{y}}^{(1)}$ = [65, 17, 39, 64]
$\hat{\mathbf{y}}^{(2)}$ = [5, 79, 34, 67]

$$\begin{aligned} f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) &= \frac{1}{4}(1^2 + 1^2 + 1^2 + 3^2) = 3 \text{ yr}^2 \\ f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) &= \frac{1}{4}(61^2 + 61^2 + 4^2 + 6^2) = 1873.5 \text{ yr}^2 \end{aligned}$$

How to measure accuracy?

- Mean squared error: $$f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
    - f_{MSE} is also both differentiable and convex.
    - **Note**: f_{MSE} expresses the error in *squared* units.

How to measure accuracy?

- Root mean squared error: $f_{\text{RMSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \right)^{1/2}$
  - **Lower** scores are better.

y = [66, 18, 38, 61]
$\hat{\mathbf{y}}^{(1)}$ = [65, 17, 39, 64]
$\hat{\mathbf{y}}^{(2)}$ = [5, 79, 34, 69]

$$f_{\text{RMSE}}(\mathbf{y}, \hat{\mathbf{y}}^{(1)}) = \left(\frac{1}{4} (1^2 + 1^2 + 1^2 + 3^2) \right)^{1/2} \approx 1.73 \text{ yr}$$

$$f_{\text{RMSE}}(\mathbf{y}, \hat{\mathbf{y}}^{(2)}) = \left(\frac{1}{4} (61^2 + 61^2 + 4^2 + 6^2) \right)^{1/2} \approx 43.28 \text{ yr}$$

How to measure accuracy?

- Root mean squared error: $f_{\text{RMSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \right)^{1/2}$

  Note: f_{RMSE} function expresses the error in the same *scale* as the ground-truth: years (not years^2). However, in practice, f_{MSE} is more commonly used:

  - Square root is tedious.
  - Monotonic relationship:

    $\hat{\mathbf{y}}^{(1)}$ is better than $\hat{\mathbf{y}}^{(2)}$ in terms of RMSE $\Leftrightarrow$

    $\hat{\mathbf{y}}^{(1)}$ is better than $\hat{\mathbf{y}}^{(2)}$ in terms of MSE

How to measure accuracy?

- In general, there are many ways of measuring accuracy.
- The appropriate metric depends on the kind of task (e.g., regression, classification) and the application domain.
  - **Regression**:
    - Ground-truth values are usually elements of an infinite set (e.g., real numbers).
    - We care more about distance between guess and ground-truth than exact match.
    - Example: estimate age from a face image

How to measure accuracy?

- In general, there are many ways of measuring accuracy.
- The appropriate metric depends on the kind of task (e.g., regression, classification) and the application domain.
  - **Classification**:
    - Ground-truth values are usually elements of a finite set (e.g., {0,1}, {-1,+1}, {😀, 🧐, 😖, 😡}).
    - Example: estimate emotion from a face image.

How to measure accuracy?

- Some people define “accuracy” as only f_{PC}.
- Many other people (including me) treat “accuracy” as a broad family of functions:
  - When the accuracy measures the error (i.e., lower is better), it is often called a **cost** or a **loss**.
  - Loss, cost (e.g., f_{MSE}, f_{MAE}): want to *minimize*.
  - Accuracy (e.g., f_{PC}, f_{AUC}): want to *maximize*.

How old are these people?

Guess how old each person is based on their face image.

66

18

38

61

57

53

29

23

https://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_01299.pdf

Age estimation accuracy

- Ground-truth **y** = [66, 18, 38, 61, 57, 53, 29, 23]
- Compute your own MSE on these 8 images:

$$f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Who had the best (lowest) score?

Python and ML

- Python is one of the most popular languages for machine learning (ML).
- Advantages:
  - Interpreted — easy for debugging
  - High level of abstraction to accomplish large amounts of computation with concise syntax
  - Excellent library support for scientific computing & ML.
- Disadvantage:
  - Slower than C for many programming tasks.

Vectorization

- Computational bottleneck in Python: **iteration is slow**.

- To avoid large-scale iteration, we can often perform the same operation using matrix arithmetic.

  - With libraries such as **`numpy`**, Python can “offload” tedious computation to a high-performance low-level library for matrix manipulation (BLAS, CUDA, etc.).

  - This conversion is sometimes called **vectorization**.

- GPU-based fast linear algebra routines have resulted in big gains for machine learning applications.

Vectorization

- Example: dot product between two large vectors **x**, **y**

  - Iterative:
    ```
    def dot (x, y):  # x, y are Python lists
      total = 0
      for i in range(len(x)):
        total += x[i] * y[i]
      return total
    ```

  - Vectorized:
    ```
    def dot (x, y):  # x, y are numpy arrays
      return x.dot(y)  # Much faster!
    ```

Vector notation

- Vectors are special cases of matrices and thus obey all the rules of matrix arithmetic.

- In this course, all vectors are **column vectors** (unless stated otherwise).

- The **transpose** of a column vector is a **row vector** (and vice-versa).

- Example:

$$\mathbf{a} = \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right] \quad \mathbf{a}^\top = \left[\begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \right]$$

Vectorization

- We can thus write the inner (dot) product of two vectors **a** and **b** as:

$$\mathbf{a}^\top \mathbf{b} = \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right]^\top \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right]$$

Vectorization

- We can thus write the inner (dot) product of two vectors **a** and **b** as:

$$
\begin{aligned}
\mathbf{a}^\top \mathbf{b} &= \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right]^\top \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right] \\
&= \left[\begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \right] \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right]
\end{aligned}
$$

Vectorization

- We can thus write the inner (dot) product of two vectors **a** and **b** as:

$$\begin{aligned}
\mathbf{a}^\top \mathbf{b} &= \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array} \right]^\top \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right] \\
&= \left[\begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \right] \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right] \\
&= a_1 b_1 + a_2 b_2 + a_3 b_3
\end{aligned}$$

Vectorization

- We can thus write the inner (dot) product of two vectors **a** and **b** as:

$$
\begin{aligned}
\mathbf{a}^\top \mathbf{b} &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}^\top \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \\
&= \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \\
&= a_1 b_1 + a_2 b_2 + a_3 b_3 \\
&= \sum_{i=1}^{3} a_i b_i
\end{aligned}
$$

Vectorization

- Similarly, we can vectorize the MSE computation:

$$f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}(\mathbf{y} - \hat{\mathbf{y}})^\top(\mathbf{y} - \hat{\mathbf{y}})$$

Vectorization

- Similarly, we can vectorize the MSE computation:

$$f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}(\mathbf{y} - \hat{\mathbf{y}})^\top(\mathbf{y} - \hat{\mathbf{y}})$$

$$= \frac{1}{n}\left[\begin{array}{ccc} (y_1 - \hat{y}_1) & \ldots & (y_n - \hat{y}_n) \end{array}\right]\left[\begin{array}{c} (y_1 - \hat{y}_1) \\ \vdots \\ (y_n - \hat{y}_n) \end{array}\right]$$

Show numpy_speed.py