CS 4342: Class 3

Jacob Whitehill

Combining multiple predictors

- Determining smile/non-smile based on a single comparison is very weak.
- What if we combined *multiple* pairs and took the majority-vote (choose non-smile if tied) across all *m* comparisons?

$$
\begin{aligned}
g^{(j)}(\mathbf{x}) &= \mathbb{I}[\mathbf{x}_{r_1,c_1} > \mathbf{x}_{r_2,c_2}] \\
\hat{y} = g(\mathbf{x}) &= \mathbb{I}\left[\left(\frac{1}{m}\sum_{j=1}^{m} g^{(j)}(\mathbf{x})\right) > 0.5\right]
\end{aligned}
$$

Step-wise classification

- Pseudocode:

```
predictors = []  # Empty list
For j = 1, ..., m:
  1. Find next best predictor given what's already in predictors
  2. Add it to predictors
```

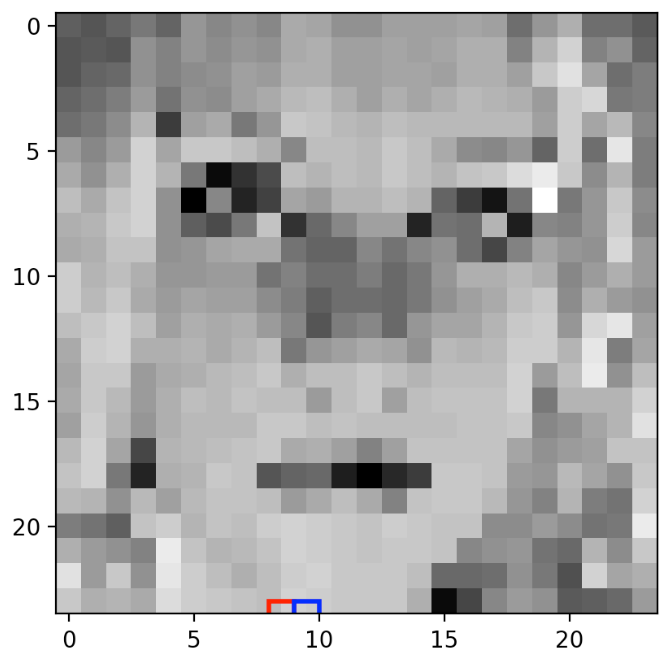
- Run smile_demo.py and optimize on 10 images.

Step-wise classification

- Accuracy (on 10 images): 100%.
- Learned feature:

Step-wise classification

- Accuracy (on 10 images): 100%.
- Learned feature (somewhat counterintuitive):



- What happened?

Jacob Whitehill, WPI

Overfitting

- When we optimized the m=1 features on a set of just 10 images, we discovered a **spurious relationship** between the image $\mathbf{x}$ and the target label y.

    - Spurious: the relationship would not **generalize** to a much larger set of images.

Overfitting

- When we optimized the m=1 features on a set of just 10 images, we discovered a **spurious relationship** between the image $\mathbf{x}$ and the target label y.
  - Spurious: the relationship would not **generalize** to a much larger set of images.
- **Problem**: we have many features (331200) but very few images (10) we need to classify.
  - Out of 331200, it's not hard to find a few features that happen to discriminate smiles/non-smiles just by chance.
- This is called **overfitting** to the dataset.

Training versus testing data

Training versus testing

- In machine learning, we always optimize the parameters/features of our classifier/regressor on a **training set** $\mathcal{D}^{\mathrm{train}}$.
- We then measure accuracy on a **testing set** $\mathcal{D}^{\mathrm{test}}$ that is disjoint from (contains no common elements with) the training set.
- The accuracy on the testing set characterizes how well our machine will perform on *new* data.
- The training and testing sets should be collected in the *same manner*.

Training versus testing

- What if the test accuracy was bad?
  - Then we should make some changes to the architecture or training procedures of our machine and re-train.
- To estimate the accuracy of the new machine, we should evaluate it on a *new* test set!
  - Why?

Training versus testing

- When we re-train a ML model, there are many things we can change, e.g.:
  - L1/L2 regularization strength
  - SVM kernel type + parameters
  - Number of neural network layers, #units/layer
  - Learning rate, momentum, etc.

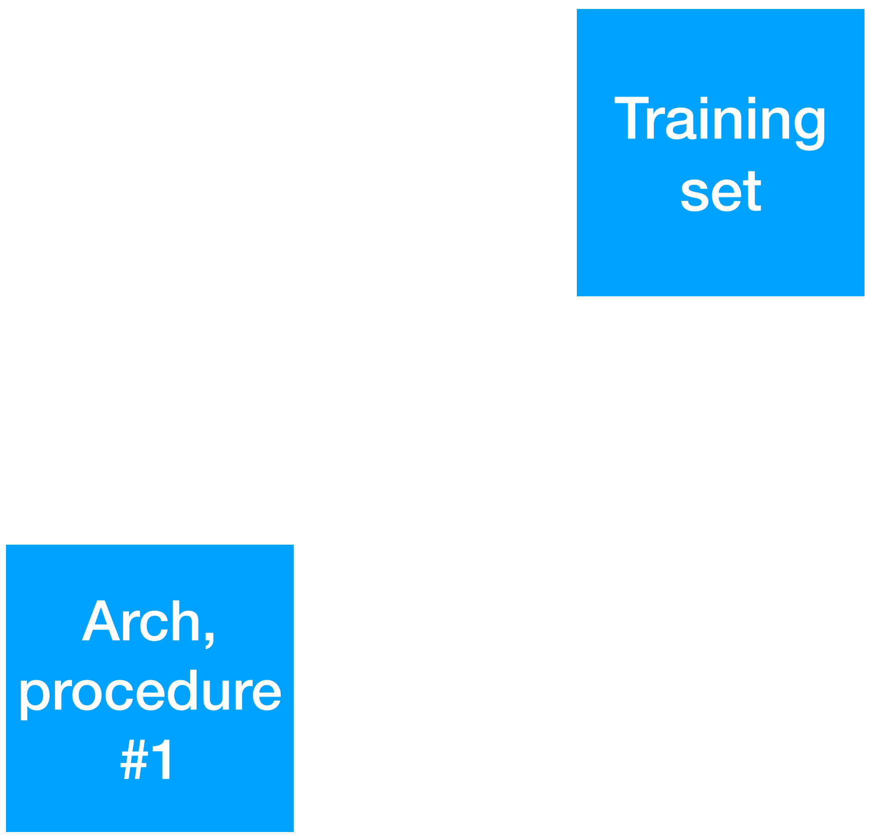
Jacob Whitehill, WPI

Training versus testing

- When we *iterative*ly re-train many ML models by optimizing on the *test set*, we might find good values for these choices just *by chance*.

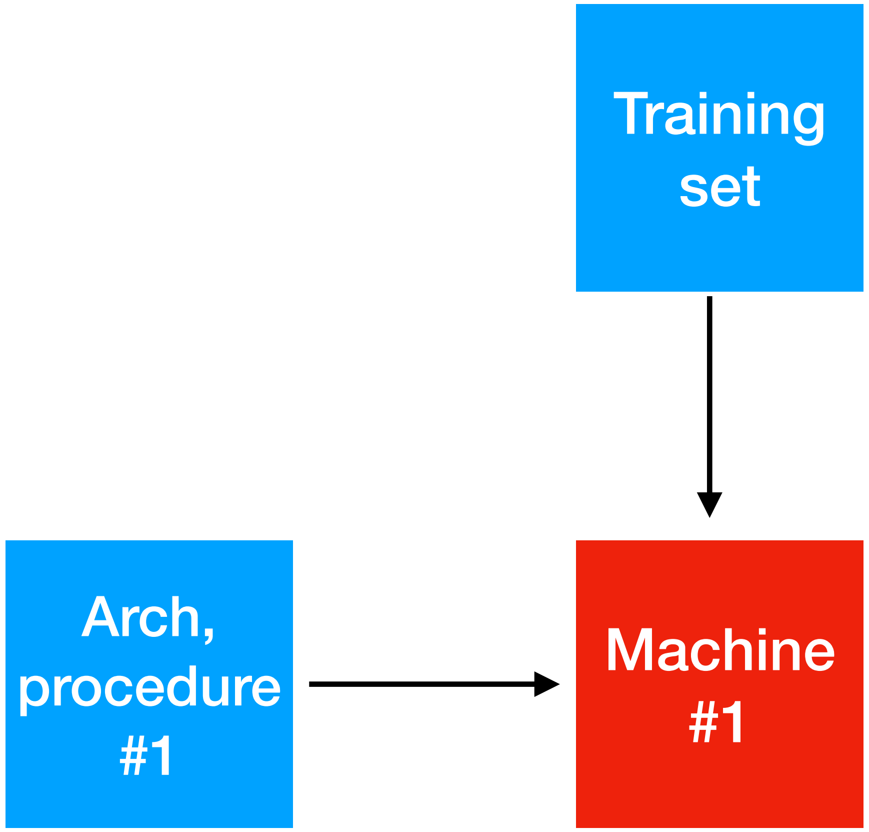
Implicit cheating

- Train a model on the training set using a particular architecture and training procedure.



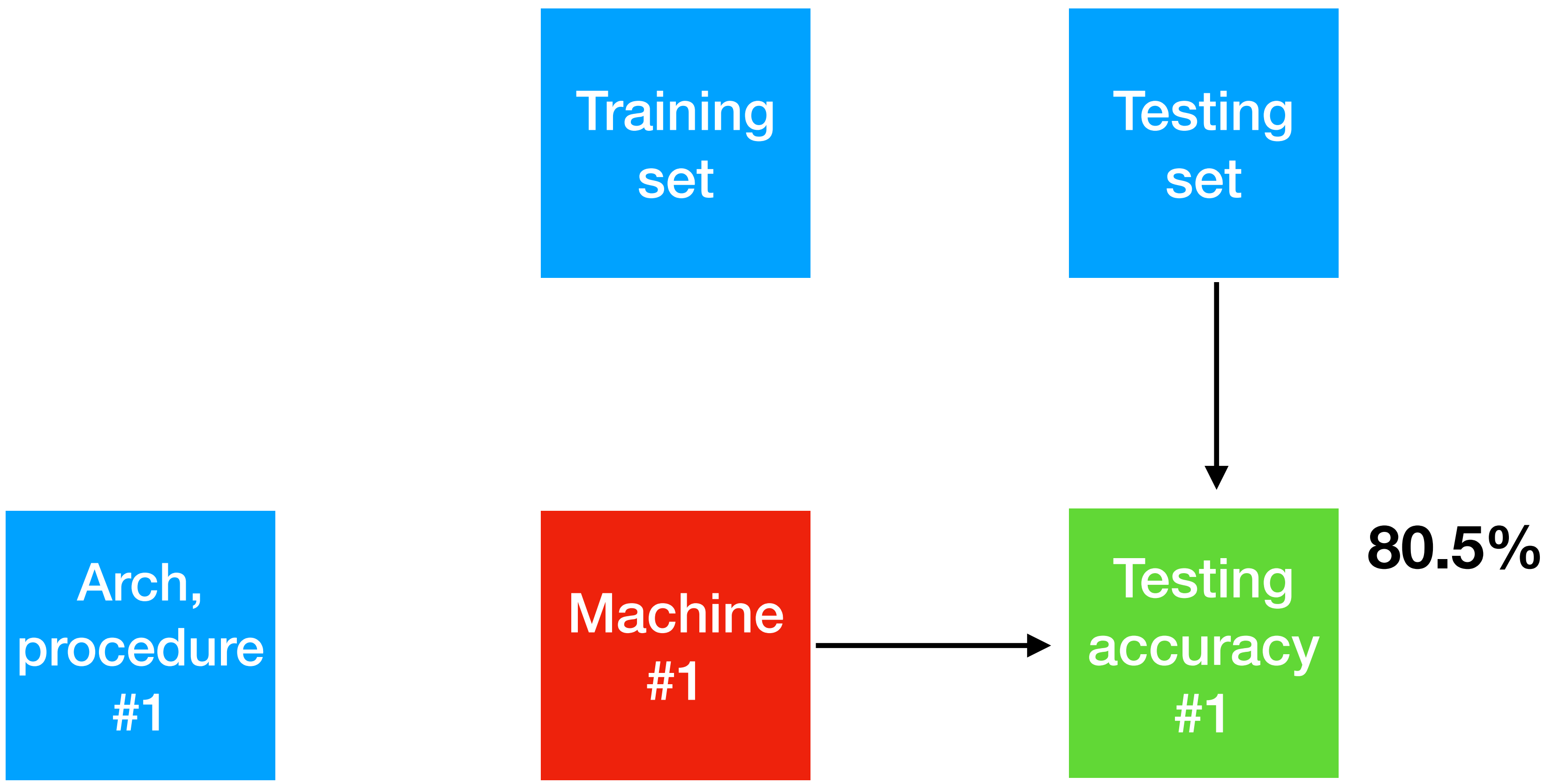
Implicit cheating

- Train a model on the training set using a particular architecture and training procedure.



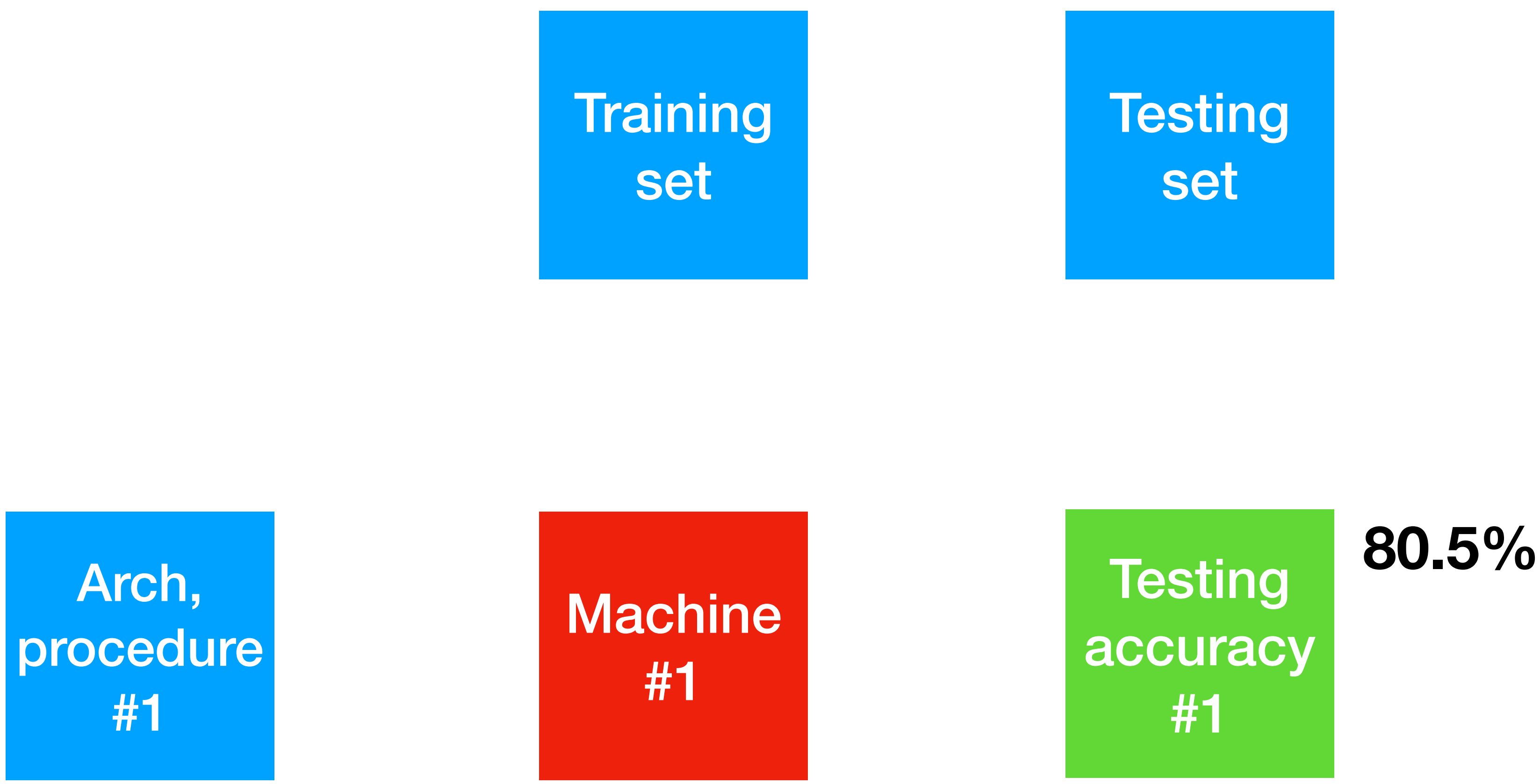
Implicit cheating

- Evaluate the trained machine on the testing set.



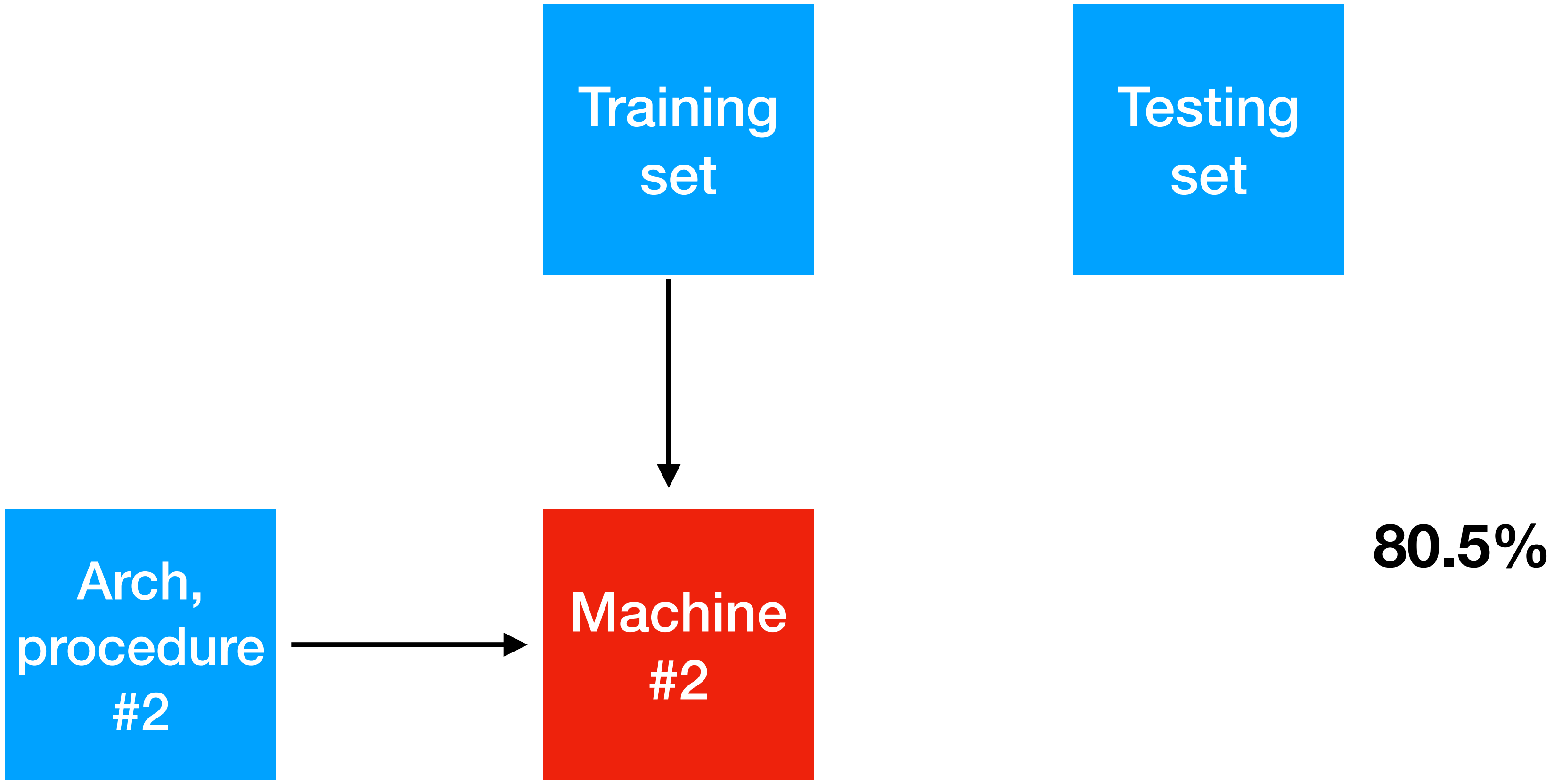
Implicit cheating

- Accuracy not good enough?



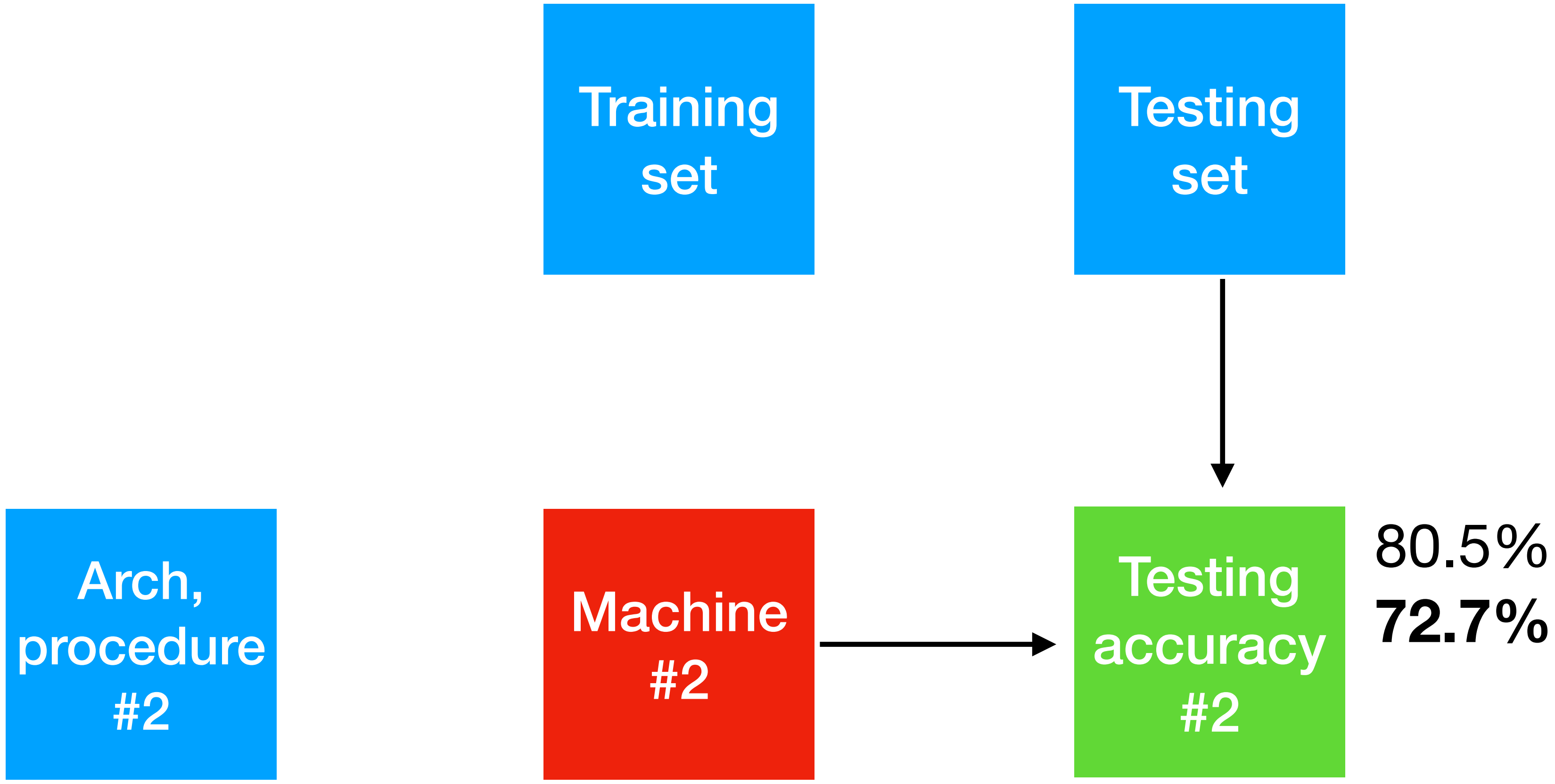
Implicit cheating

- Choose a different design and try again!



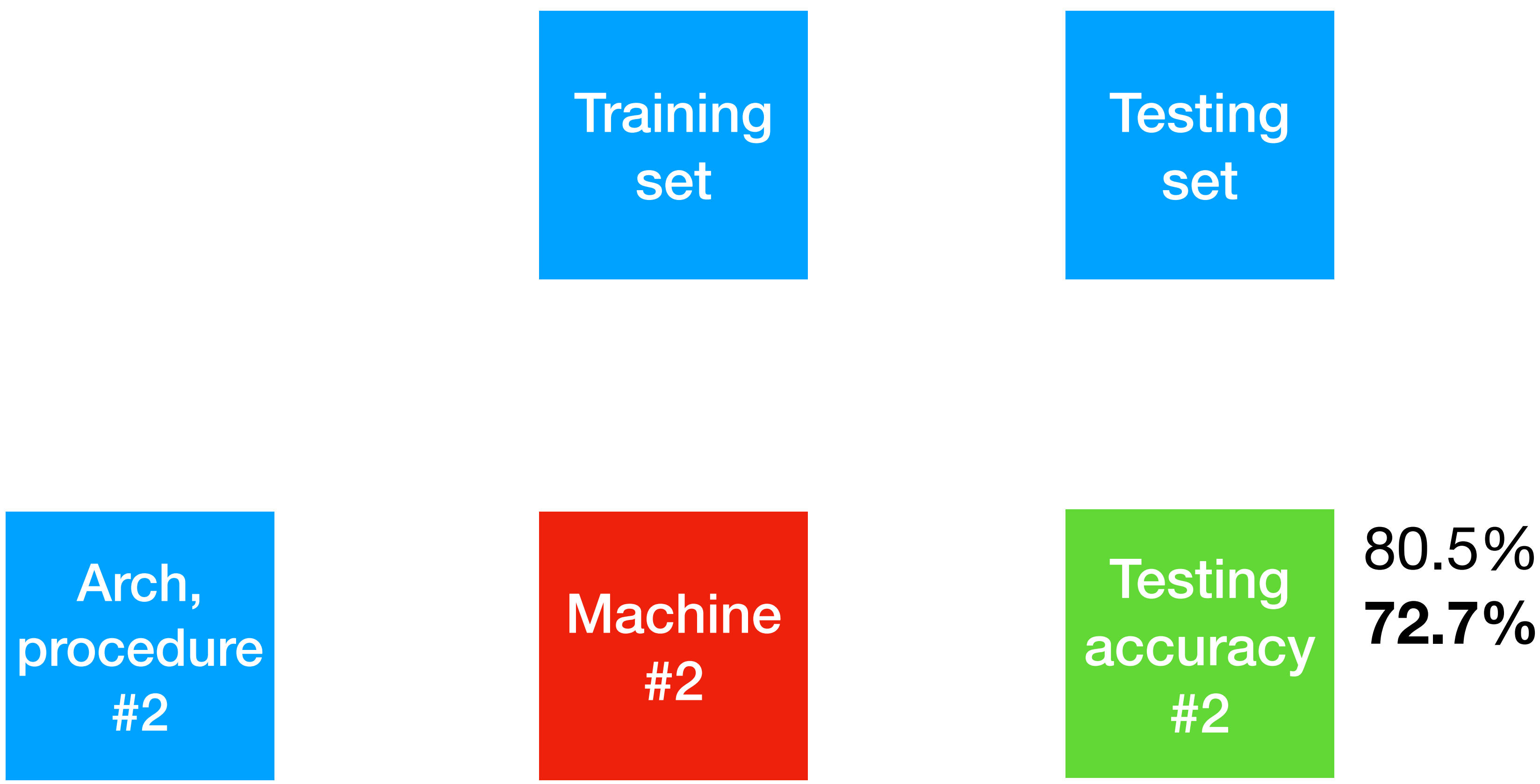
Implicit cheating

- Evaluate the new machine on the testing set.



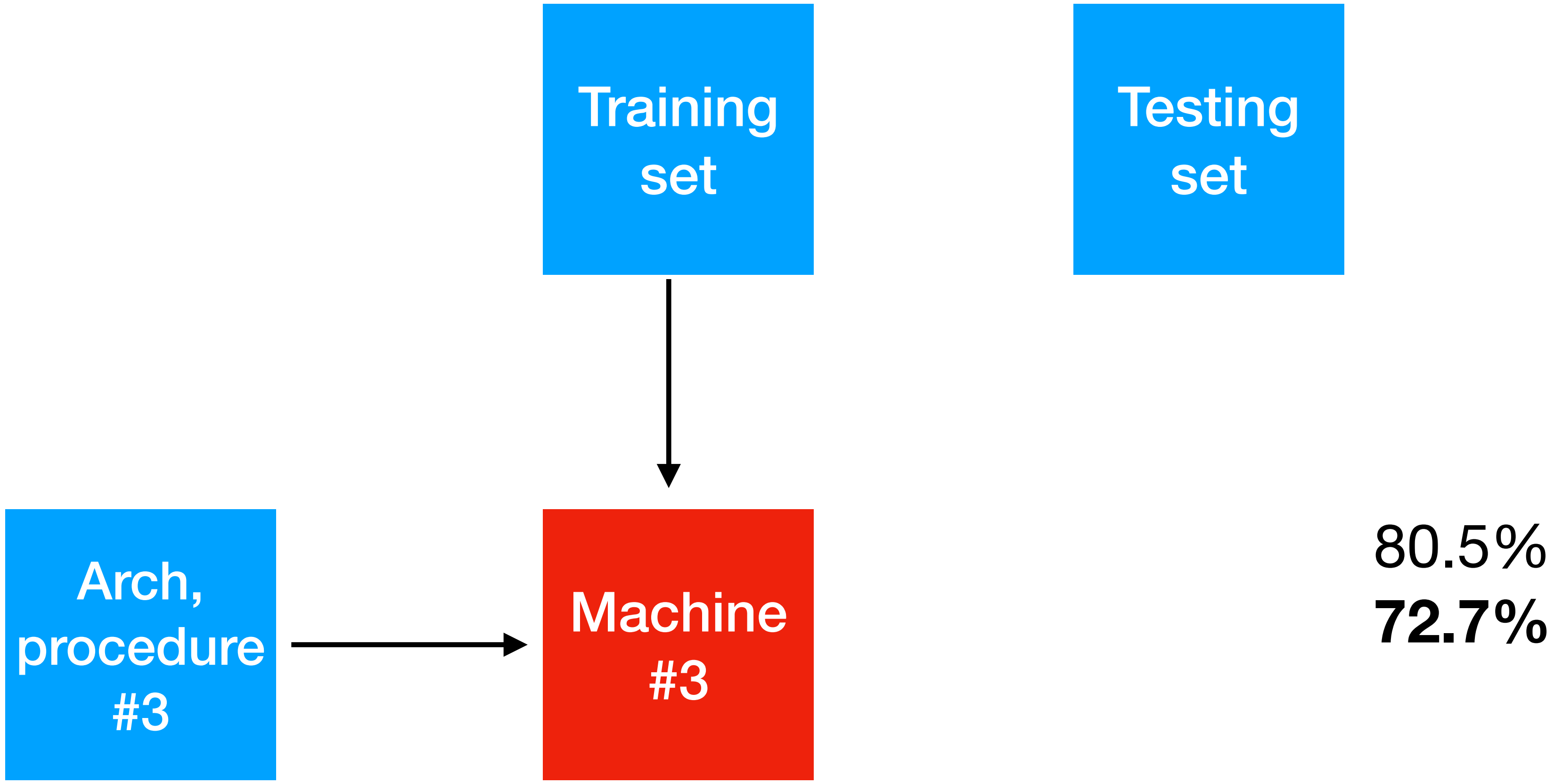
Implicit cheating

- Accuracy still not good enough?



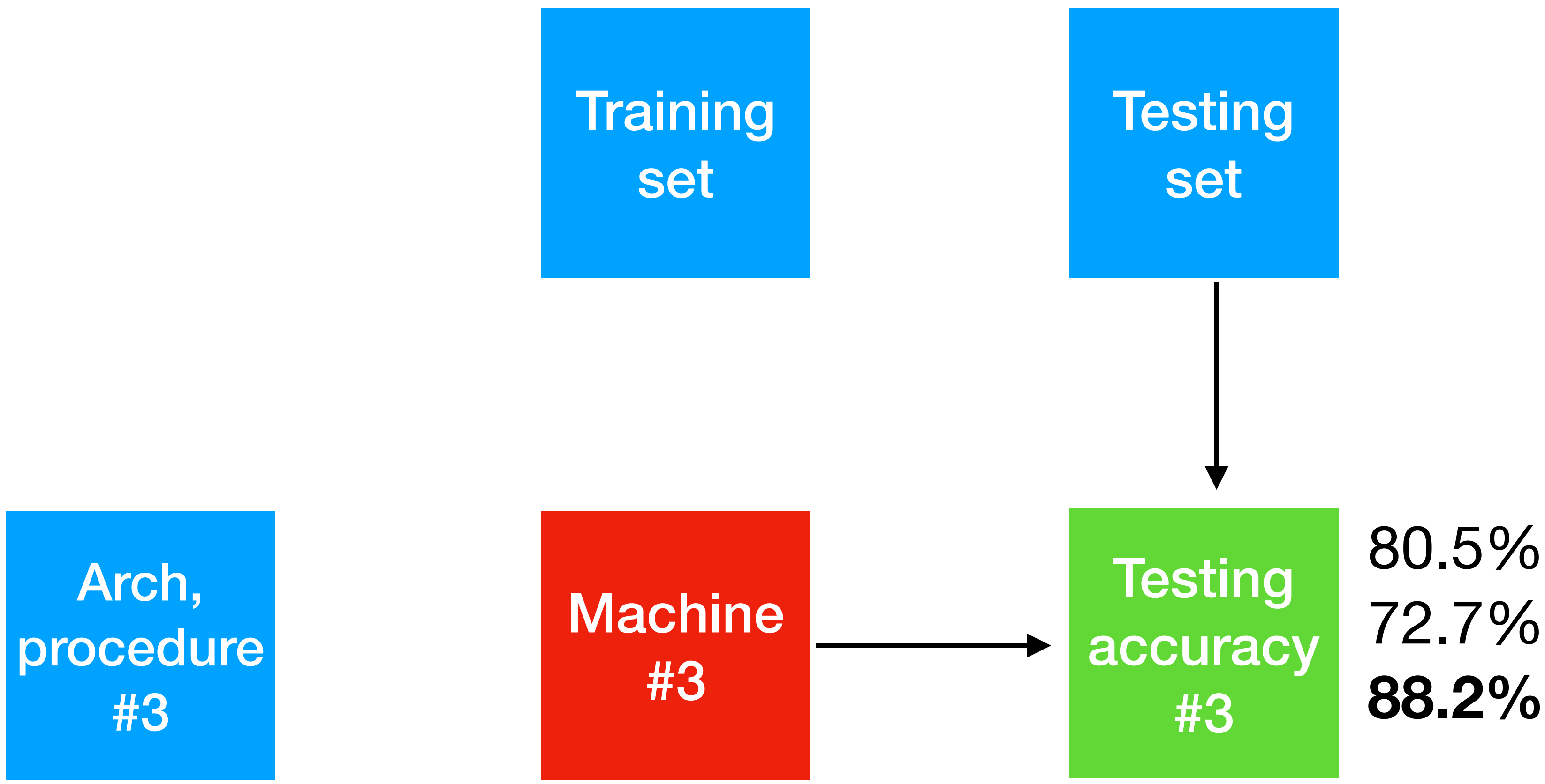
Implicit cheating

- Choose yet another design and try again!



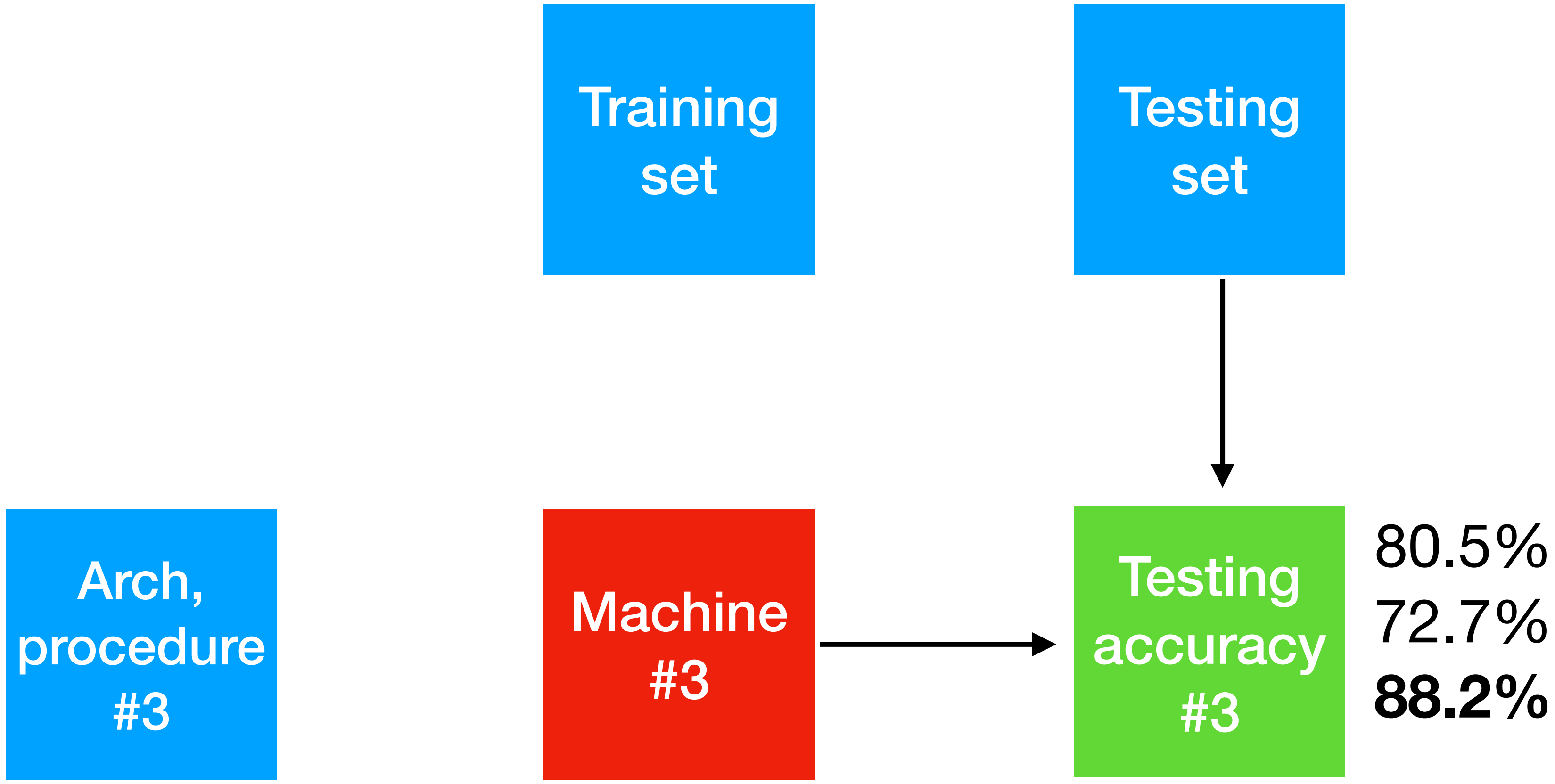
Implicit cheating

- Evaluate the new machine on the testing set.



Implicit cheating

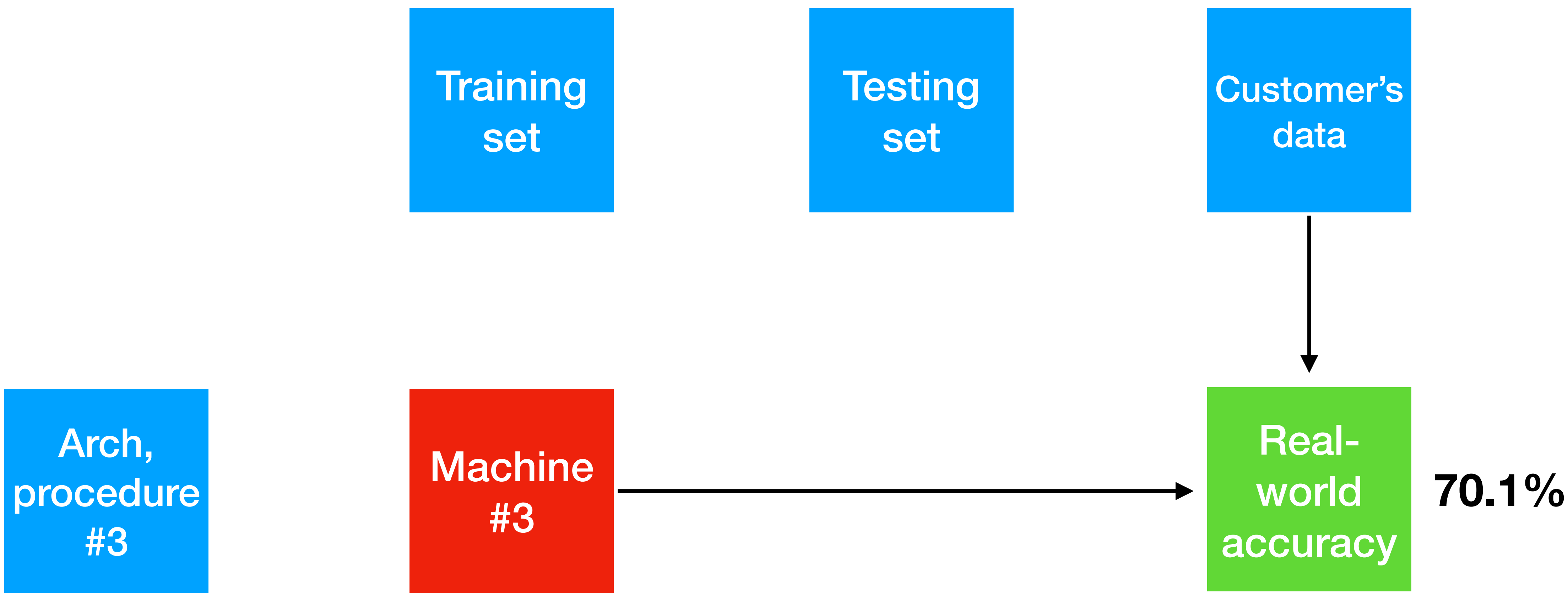
- Much better! Let’s keep machine #3 and sell it on Amazon!



Jacob Whitehill, WPI

Implicit cheating

- Oops — the real-world accuracy was much less than what we estimated on the test set!



Linear regression

Limitations of our feature set

- So far, the predictors we have considered are very simple:
  - Is pixel (r_1,c_1) brighter than pixel (r_2,c_2)?
- We can't even express simple relationships such as:
  - "Pixel (r_1,c_1) is at least 5 bigger than pixel (r_2,c_2)"
  - "2 times pixel (r_1,c_1) is bigger than pixel (r_2,c_2)"
  - "2 times pixel (r_1,c_1) plus 4 times pixel (r_2,c_2) is larger than pixel (r_3,c_3)".

Linear regression

- We can harness these kinds of relationships using **linear regression**.
- Let’s switch back to the age estimation problem…

Linear algebra

- A **column vector** is a (n x 1) matrix.
- A **row vector** is a (1 x n) matrix.
- The **transpose** of (n x k) matrix **A**, denoted $\mathbf{A}^T$, is (k x n).
- Multiplication of matrices **A** and **B**:
  - Only possible when: **A** is (n x k) and **B** is (k x m)
  - Result: (n x m)

Linear algebra

- The **inner product** between two column vectors (same length) **x**, **y** can be written as: $\mathbf{x}^\mathsf{T}\mathbf{y}$

$$\begin{bmatrix} x_1 & \dots & x_m \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \sum_{i=1}^{m} x_i y_i$$

- An inner product produces a **scalar**.

Linear algebra

- The **outer product** between two column vectors (same length) **x**, **y** can be written as: $\mathbf{xy}^\mathsf{T}$:

$$\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & \dots & y_m \end{bmatrix} = \begin{bmatrix} x_1 y_1 & \dots & x_1 y_m \\ \vdots & \ddots & \vdots \\ x_m y_1 & \dots & x_m y_m \end{bmatrix}$$

- An outer product produces a **matrix.**

Example

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 4 & -1 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ 8 & -2 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 0 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 6 & 0 \end{bmatrix}$$

Linear algebra

- The **sum** of **multiple outer products** can be expressed as the multiplication of two matrices:

$$
\begin{aligned}
\mathbf{x}^{(1)}\mathbf{y}^{(1)}{}^\top + \ldots \mathbf{x}^{(n)}\mathbf{y}^{(n)}{}^\top &= \sum_{i=1}^{n} \mathbf{x}^{(i)}\mathbf{y}^{(i)}{}^\top \\
&= \begin{bmatrix} | & & | \\ \mathbf{x}^{(1)} & \ldots & \mathbf{x}^{(n)} \\ | & & | \end{bmatrix} \begin{bmatrix} — & \mathbf{y}^{(1)} & — \\ & \vdots & \\ — & \mathbf{y}^{(n)} & — \end{bmatrix} \\
&\doteq \mathbf{X}\mathbf{Y}^\top
\end{aligned}
$$

Example

$$\mathbf{x} \quad \mathbf{y}^\mathsf{T} \quad \mathbf{x}\mathbf{y}^\mathsf{T}$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 4 & -1 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ 8 & -2 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 0 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 6 & 0 \end{bmatrix}$$

$$\mathbf{X} \quad \mathbf{Y}^\mathsf{T} \quad \mathbf{X}\mathbf{Y}^\mathsf{T}$$

$$\begin{bmatrix} 1 & -1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 4 & -1 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 14 & -2 \end{bmatrix}$$

Linear algebra

- Here's a special case:

$$
\begin{aligned}
\mathbf{x}^{(1)}\mathbf{x}^{(1)}{}^\top + \ldots \mathbf{x}^{(n)}\mathbf{x}^{(n)}{}^\top &= \sum_{i=1}^{n} \mathbf{x}^{(i)}\mathbf{x}^{(i)}{}^\top \\
&= \left[\begin{array}{ccc} | & & | \\ \mathbf{x}^{(1)} & \ldots & \mathbf{x}^{(n)} \\ | & & | \end{array} \right] \left[\begin{array}{ccc} — & \mathbf{x}^{(1)} & — \\ & \vdots & \\ — & \mathbf{x}^{(n)} & — \end{array} \right] \\
&\doteq \mathbf{X}\mathbf{X}^\top
\end{aligned}
$$

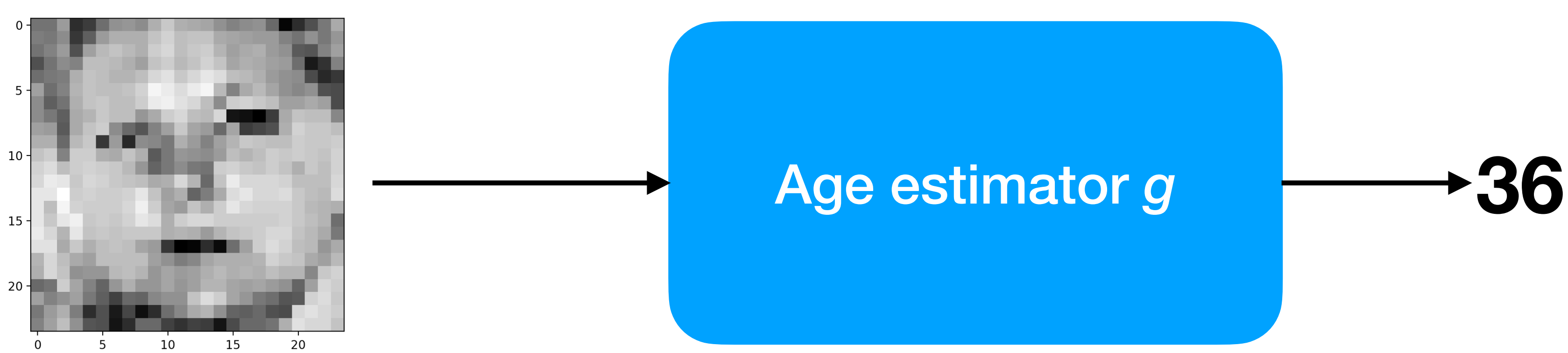
Linear regression

- Linear regression is built as a linear combination of all the inputs **x**:

image pixels

$$\hat{y} = g(\mathbf{x}; \mathbf{w}) = \sum_{j=1}^{m} \mathbf{x}_j \mathbf{w}_j = \mathbf{x}^\top \mathbf{w}$$

- Here, we treat the image **x** as a *vector* (even though it represents a 2-d image).



x $\hat{y}$

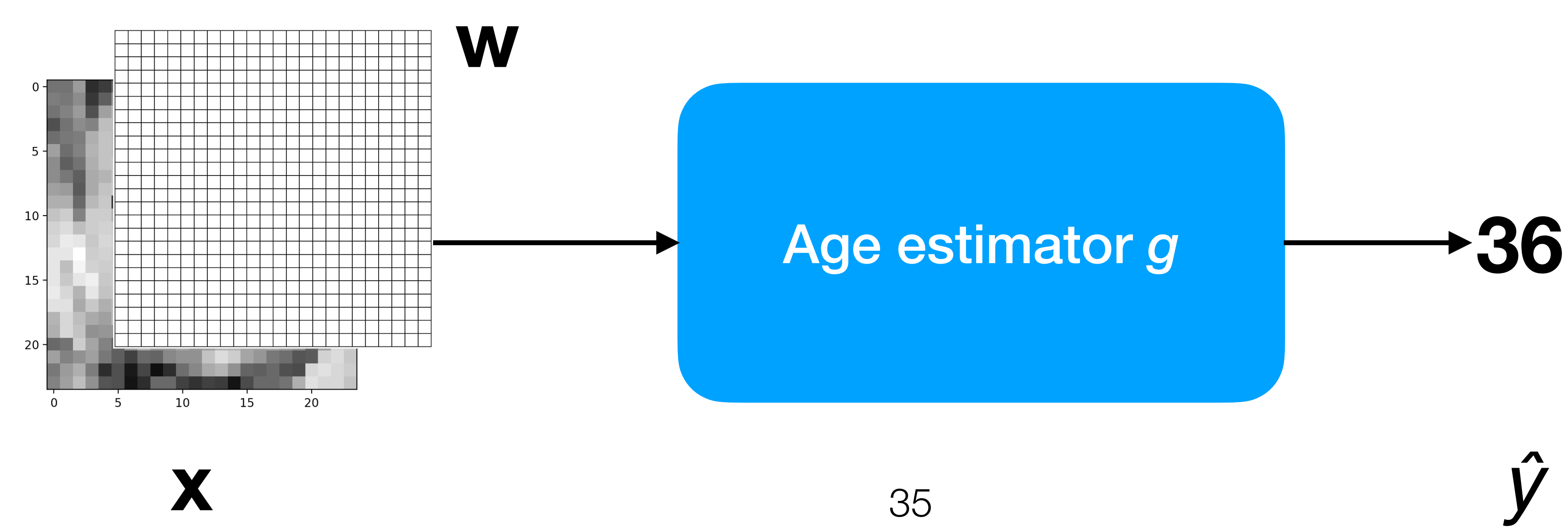
Linear regression

- Linear regression is built as a linear combination of all the inputs **x**:

image pixels

$$\hat{y} = g(\mathbf{x}; \mathbf{w}) = \sum_{j=1}^{m} \mathbf{x}_j \mathbf{w}_j = \mathbf{x}^\top \mathbf{w}$$

- Vector **w** represent an "overlay image" that weights the different pixel intensities of **x**.



Linear regression

- Imagine a 2x2 pixel "image" **x** and a weight matrix **w**:

2	5
0	3

x

1	3
2	4

w

- Then $\hat{y} =$?

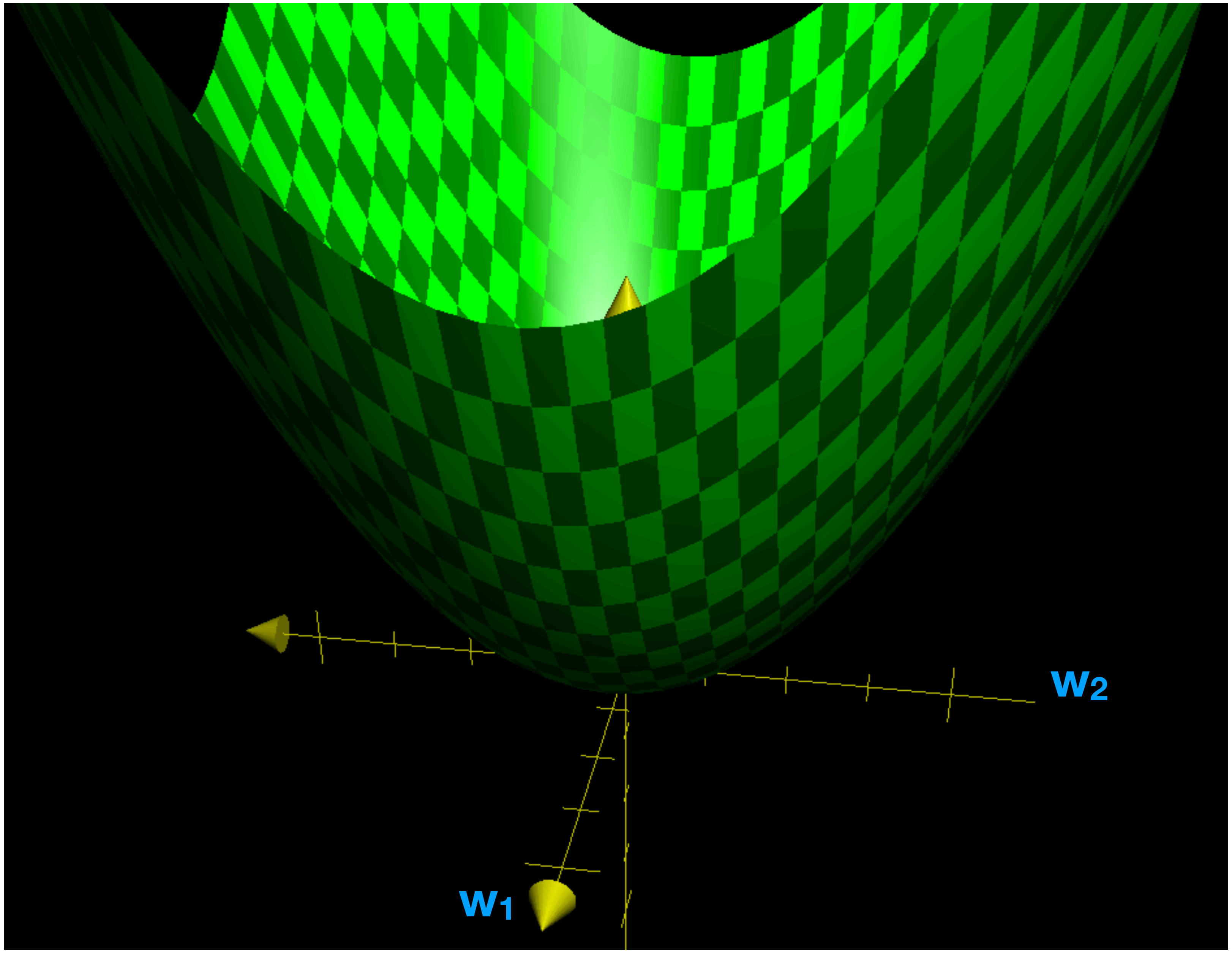
Linear regression

- How should we choose each "weight" $\mathbf{w}_j$?
- Let's define the **loss** function that we seek to minimize:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \frac{1}{2n} \sum_{i=1}^{n} \left(g(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)} \right)^2 \\
&= \frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2
\end{aligned}
$$

The 2 in the denominator will slightly simplify the algebra later…

What does f(w) look like?



Linear regression

- **w** can be any real-valued vector; hence, we can use differential calculus to find the minimum of f_{MSE}.

- Just derive the **gradient** of f_{MSE} w.r.t. **w**, set to 0, and solve.

- Since f_{MSE} is a convex function, we are guaranteed that this critical point is a global minimum.

Gradient vector

- For a real-valued function $f : \mathbb{R}^m \rightarrow \mathbb{R}$, we define the gradient w.r.t. **w** as:

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{w}_m} \end{bmatrix}$$

- In other words, the gradient is a column vector containing all first partial derivatives w.r.t. **w**.

Gradient vector: exercise 1

$$f(\mathbf{w}) = f\left(\left[\begin{array}{c} w_1 \\ w_2 \end{array}\right]\right) = 3w_1^2 - \sin(2w_2)$$

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \left[\begin{array}{c} \\ \\ \end{array}\right]$$

Gradient vector: exercise 2

$$f\left(\mathbf{x}, \mathbf{w}, y\right) = \frac{1}{2}(\mathbf{x}^\top \mathbf{w} - y)^2$$

$$\nabla_{\mathbf{w}} f\left(\mathbf{x}, \mathbf{w}, y\right) =$$

Gradient vector: exercise 2

$$f(\mathbf{x}, \mathbf{w}, y) = \frac{1}{2}(\mathbf{x}^\top \mathbf{w} - y)^2$$
$$= \frac{1}{2}(x_1 w_1 + x_2 w_2 - y)^2$$

$$\nabla_{\mathbf{w}} f(\mathbf{x}, \mathbf{w}, y) =$$

Solving for w

- The gradient of f_{MSE} is thus:

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)
\end{aligned}$$

Solving for $\mathbf{w}$

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left(\mathbf{x}^{(i)}{}^\top\mathbf{w} - y^{(i)}\right)$$

$$0 \quad = \quad \sum_i \mathbf{x}^{(i)}\mathbf{x}^{(i)}{}^\top\mathbf{w} - \sum_i \mathbf{x}^{(i)}y^{(i)}$$

Solving for $\mathbf{w}$

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left({\mathbf{x}^{(i)}}^\top\mathbf{w} - y^{(i)}\right)$$

$$\begin{aligned} 0 &= \sum_i \mathbf{x}^{(i)}{\mathbf{x}^{(i)}}^\top\mathbf{w} - \sum_i \mathbf{x}^{(i)}y^{(i)} \\ \sum_i \mathbf{x}^{(i)}{\mathbf{x}^{(i)}}^\top\mathbf{w} &= \sum_i \mathbf{x}^{(i)}y^{(i)} \end{aligned}$$

Solving for $\mathbf{w}$

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left(\mathbf{x}^{(i)\top}\mathbf{w} - y^{(i)}\right)$$

$$\begin{aligned}
0 &= \sum_i \mathbf{x}^{(i)}\mathbf{x}^{(i)\top}\mathbf{w} - \sum_i \mathbf{x}^{(i)}y^{(i)} \\
\sum_i \mathbf{x}^{(i)}\mathbf{x}^{(i)\top}\mathbf{w} &= \sum_i \mathbf{x}^{(i)}y^{(i)} \\
\mathbf{w} &= \left(\sum_i \mathbf{x}^{(i)}\mathbf{x}^{(i)\top}\right)^{-1}\sum_i \mathbf{x}^{(i)}y^{(i)}
\end{aligned}$$

Linear regression: matrix notation

- To compute **w**, do *not* use np.linalg.inv.

- Instead, use np.linalg.solve, which avoids explicitly computing the matrix inverse.

Linear regression: matrix notation

- Let's define a matrix **X** to contain all the training images:

$$\mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{bmatrix}$$

  - In statistics, **X** is called the **design matrix.**

- Let's define vector **y** to contain all the training labels:

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Linear regression: matrix notation

- Using summation notation, we derived:

$$\mathbf{w} = \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right)$$

- Using matrix notation, we can write the solution as:

$$\mathbf{w} = \boxed{?}$$

Linear regression: matrix notation

- Once we've "trained" the weights **w**, we can estimate the *y*-value (label) for any **x**.

- We can compute the { $\hat{y}^{(i)}$ } for a set of images { $\mathbf{x}^{(i)}$ } in one-fell-swoop using matrix operations.

- Let's define our design matrix **X** as before:

$$\mathbf{X} = \left[\begin{array}{ccc} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{array} \right]$$

- Then our estimates of the labels is given by:

$$\hat{\mathbf{y}} = \mathbf{X}^\top \mathbf{w}$$

Jacob Whitehill, WPI

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$\hat{\mathbf{y}} = \mathbf{X}^\top \mathbf{w}$$

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\ &= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\end{aligned}$$

This is the index of the *image*.

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}
\end{aligned}
$$

This is the index of the *pixel*.

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} \\ \vdots & \vdots \\ \mathbf{x}_1^{(n)} & \mathbf{x}_2^{(n)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}
\end{aligned}
$$

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} \\ \vdots & \vdots \\ \mathbf{x}_1^{(n)} & \mathbf{x}_2^{(n)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} w_1 + \mathbf{x}_2^{(1)} w_2 \\ \vdots \\ \mathbf{x}_1^{(n)} w_1 + \mathbf{x}_2^{(n)} w_2 \end{bmatrix}
\end{aligned}
$$

Jacob Whitehill, WPI