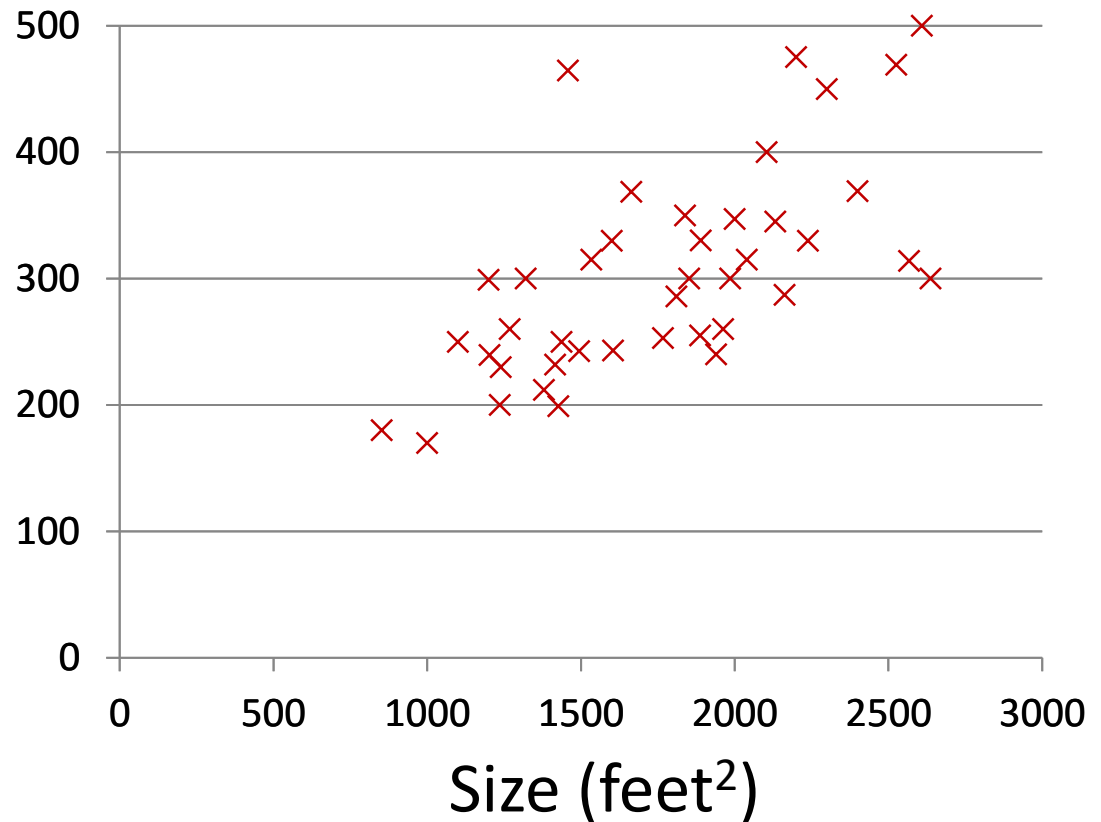


# Lecture 2

## Linear Regression

# Housing Prices (Portland, OR)

Price  
(in 1000s  
of  
dollars)

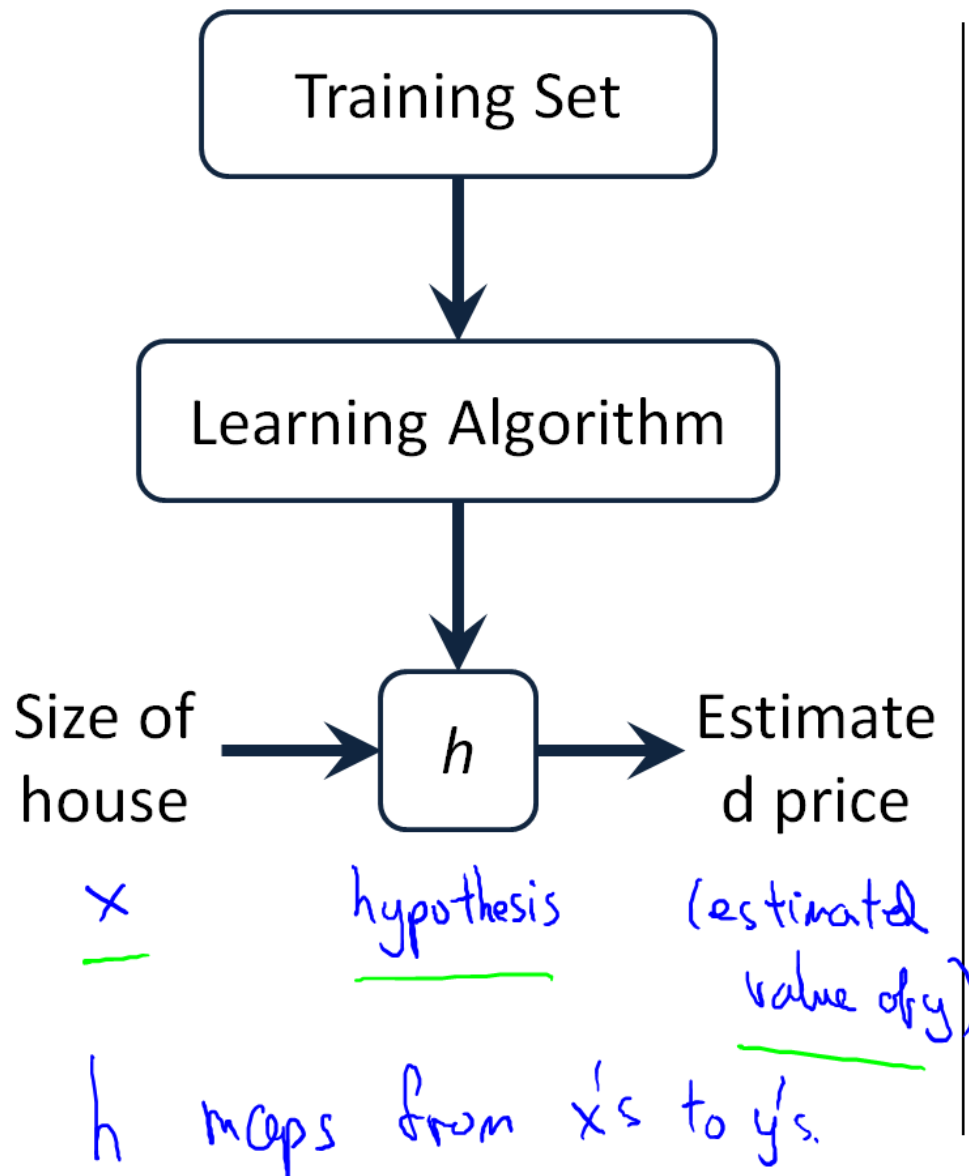


## Supervised Learning

Given the “right answer”  
for each example in the  
data.

## Regression Problem

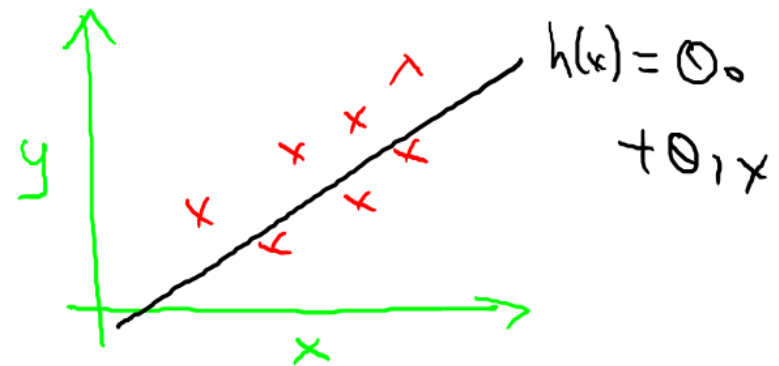
Predict real-valued output



How do we represent  $h$  ?

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Shorthand:  $h(x)$



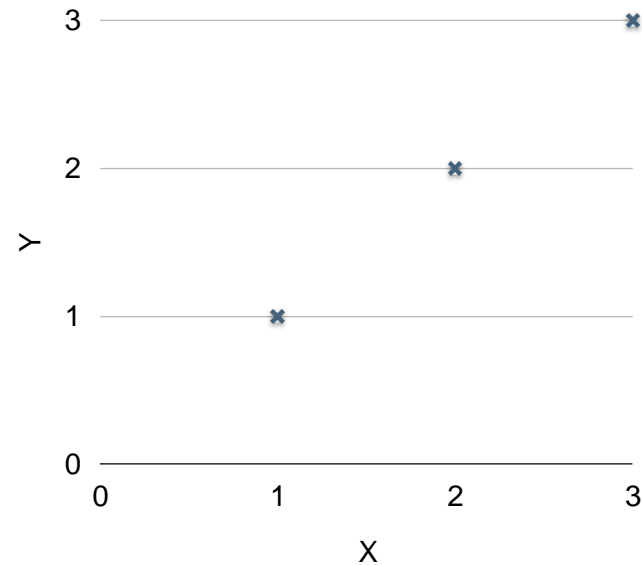
Linear regression with one variable.  
Univariate linear regression.

# Regression (data)

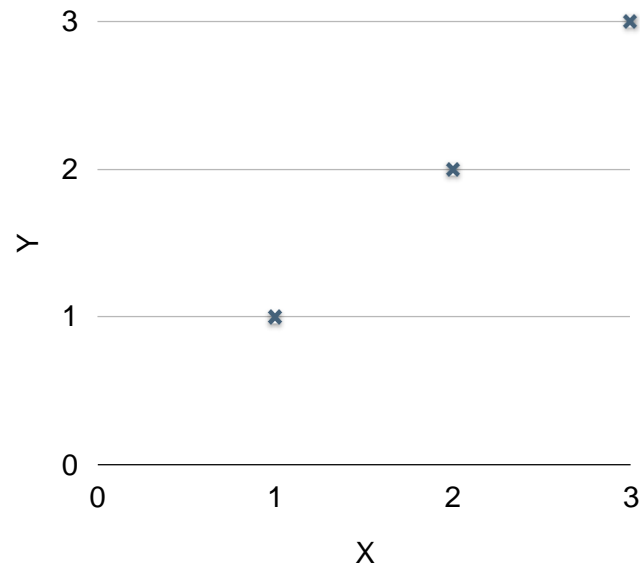
x	y
1	1
2	2
3	3

# Regression (presentation)

x	Y
1	1
2	2
3	3

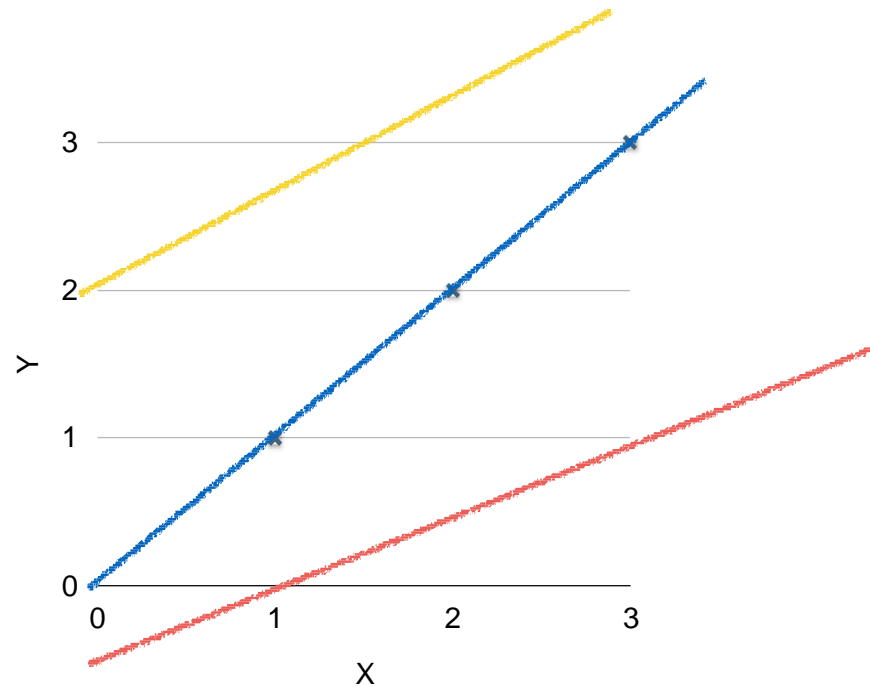


# (Linear) Hypothesis

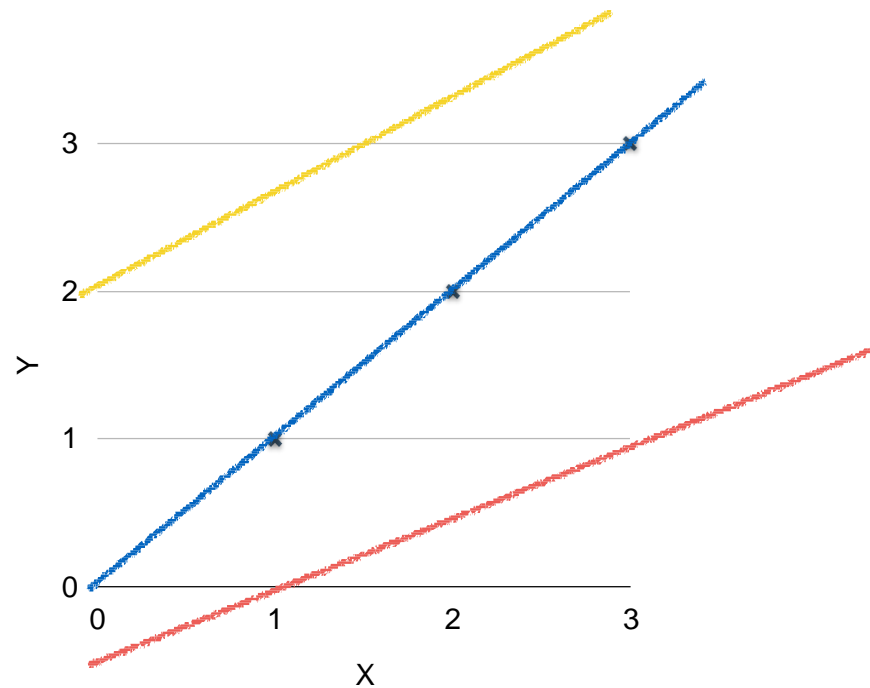


# (Linear) Hypothesis

$$H(x) = Wx + b$$

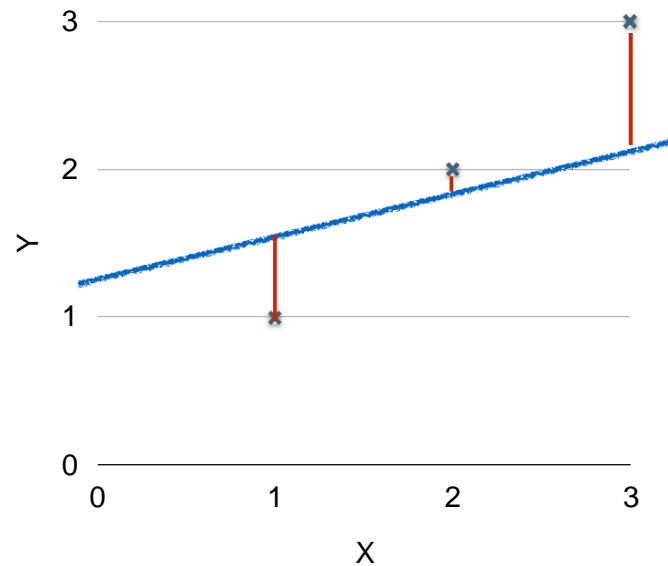


# Which hypothesis is better?





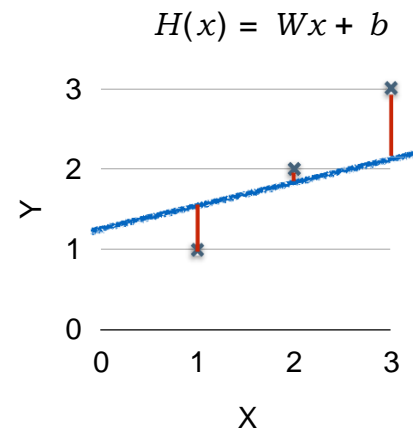
# Which hypothesis is better?



# Cost function

- How fit the line to our (training) data

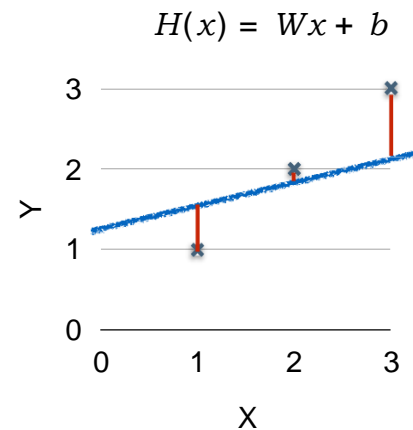
$$H(x) \quad y$$



# Cost function

- How fit the line to our (training) data

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$



$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

# Hypothesis and Cost

$$H(x) = Wx + b$$

$$\text{cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

# What $cost(W)$ looks like?

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- $W=1, cost(W)=?$

# What $cost(W)$ looks like?

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)})^2$$

X	Y
1	1
2	2
3	3

- $W=1, cost(W)=0$

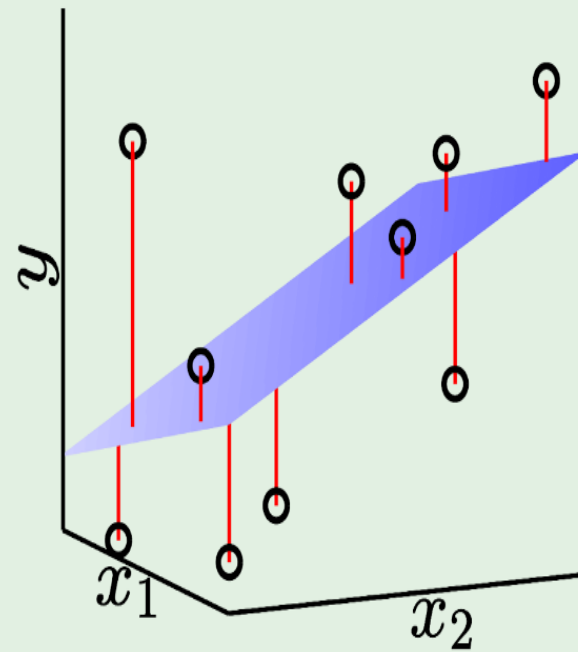
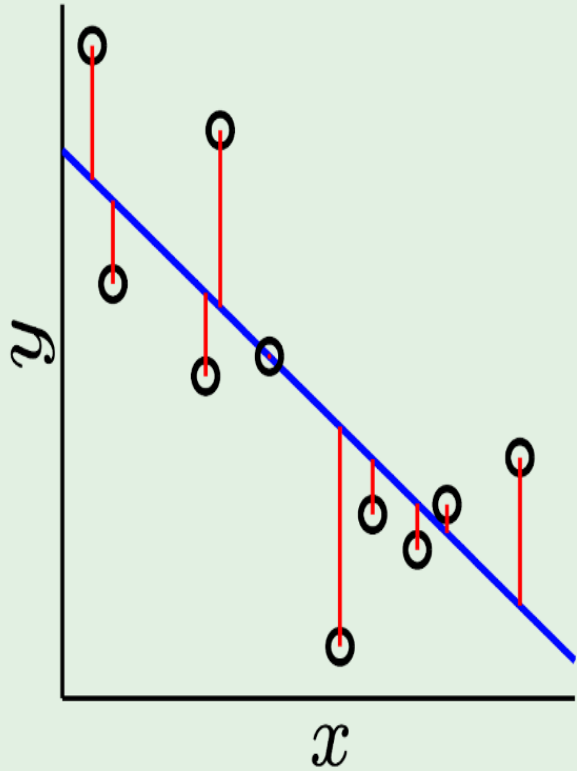
$$\frac{1}{3}((1 * 1 - 1)^2 + (1 * 2 - 2)^2 + (1 * 3 - 3)^2)$$

- $W=0, cost(W)=4.67$

$$\frac{1}{3}((0 * 1 - 1)^2 + (0 * 2 - 2)^2 + (0 * 3 - 3)^2)$$

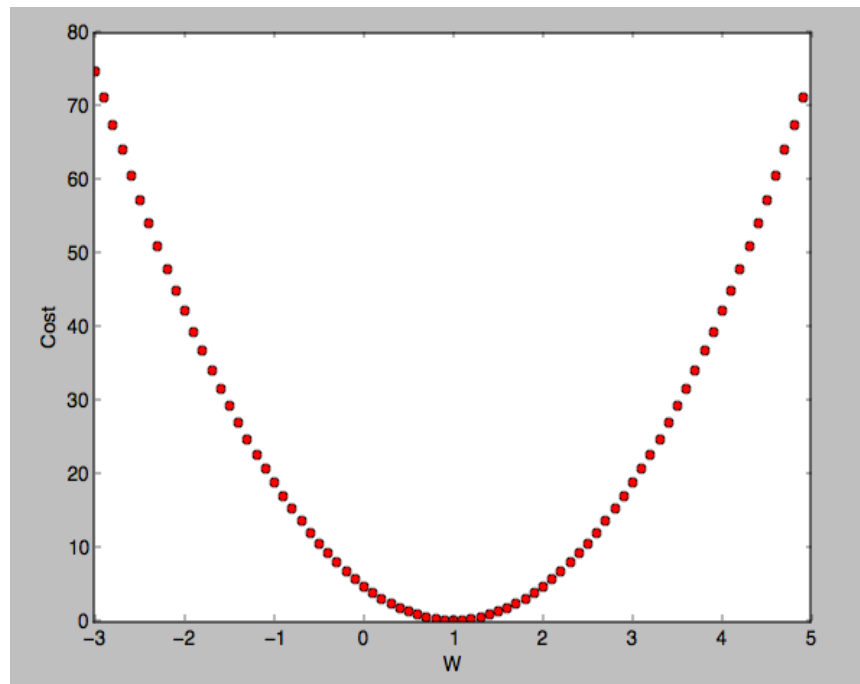
- $W=2, cost(W)=?$

## Illustration of linear Regression



# What $cost(W)$ looks like?

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)})^2$$



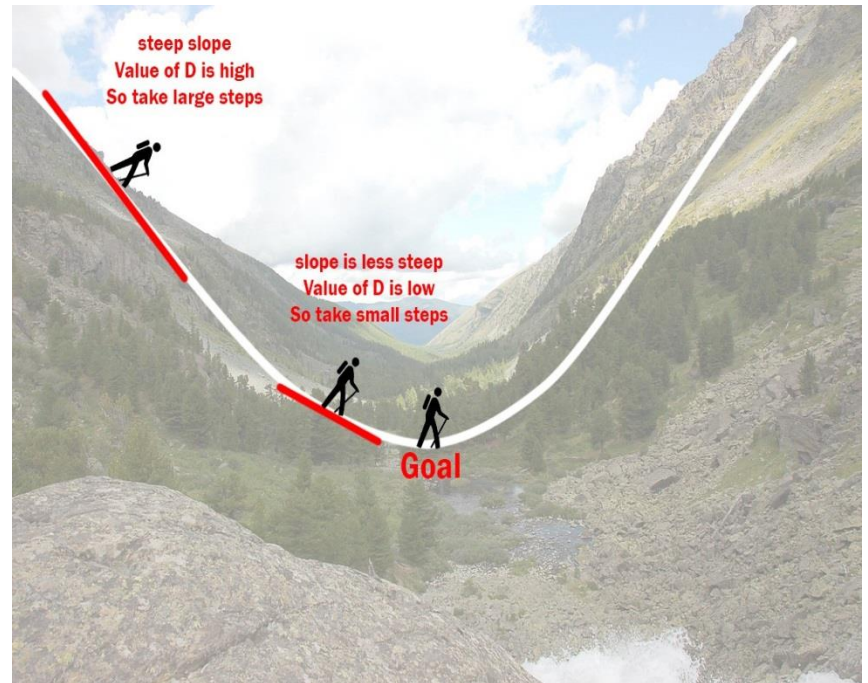
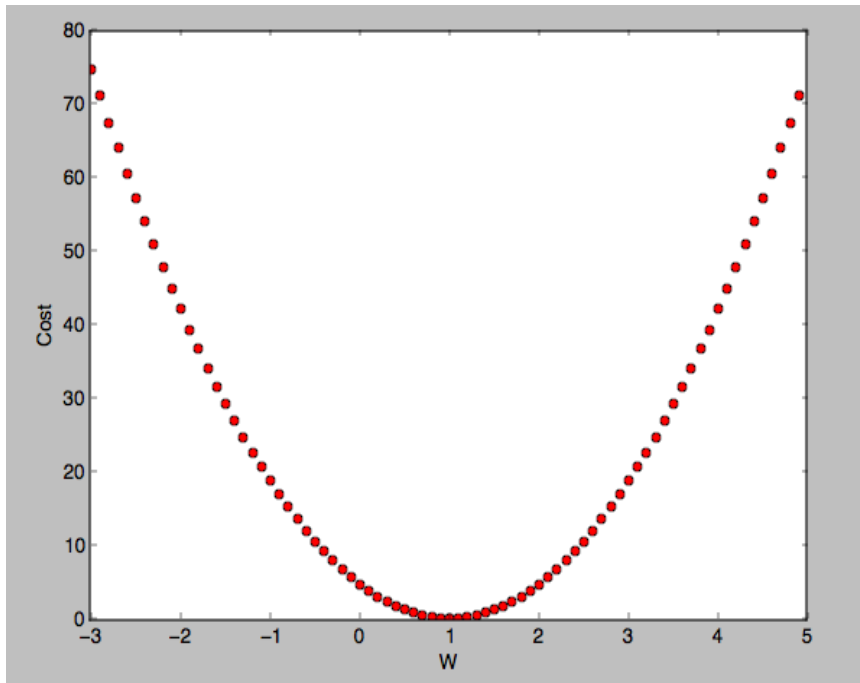


# Gradient descent algorithm

- Minimize cost function
- Gradient descent is used many minimization problems
- For a given cost function,  $cost(W, b)$ , it will find  $W, b$  to minimize cost
- It can be applied to more general function:  $cost(w1, w2, \dots)$

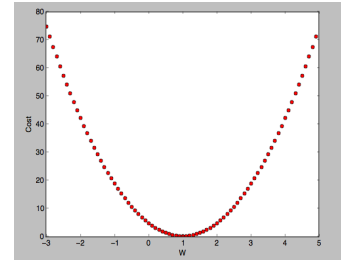
# How it works?

How would you find the lowest point?



# How it works?

- Start with initial guesses
  - Start at 0,0 (or any other value)
  - Keeping changing  $W$  and  $b$  a little bit to try and reduce  $\text{cost}(W, b)$
- Each time you change the parameters, you select the gradient which reduces  $\text{cost}(W, b)$  the most possible
- Repeat
- Do so until you converge to a local minimum



[http://www.holehouse.org/mlclass/01\\_02\\_Introduction\\_regression\\_analysis\\_and\\_gr.html](http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html)

# Gradient descent algorithm

- **Gradient descent** is an optimization algorithm used to minimize some function by iteratively moving in the direction of **steepest descent** as defined by the negative of the **gradient**. In **machine learning**, we use **gradient descent** to update the parameters of our model.

# Formal definition

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$



$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

# Formal definition

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$

# Formal definition

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^m (W x^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m 2(W x^{(i)} - y^{(i)}) x^{(i)}$$

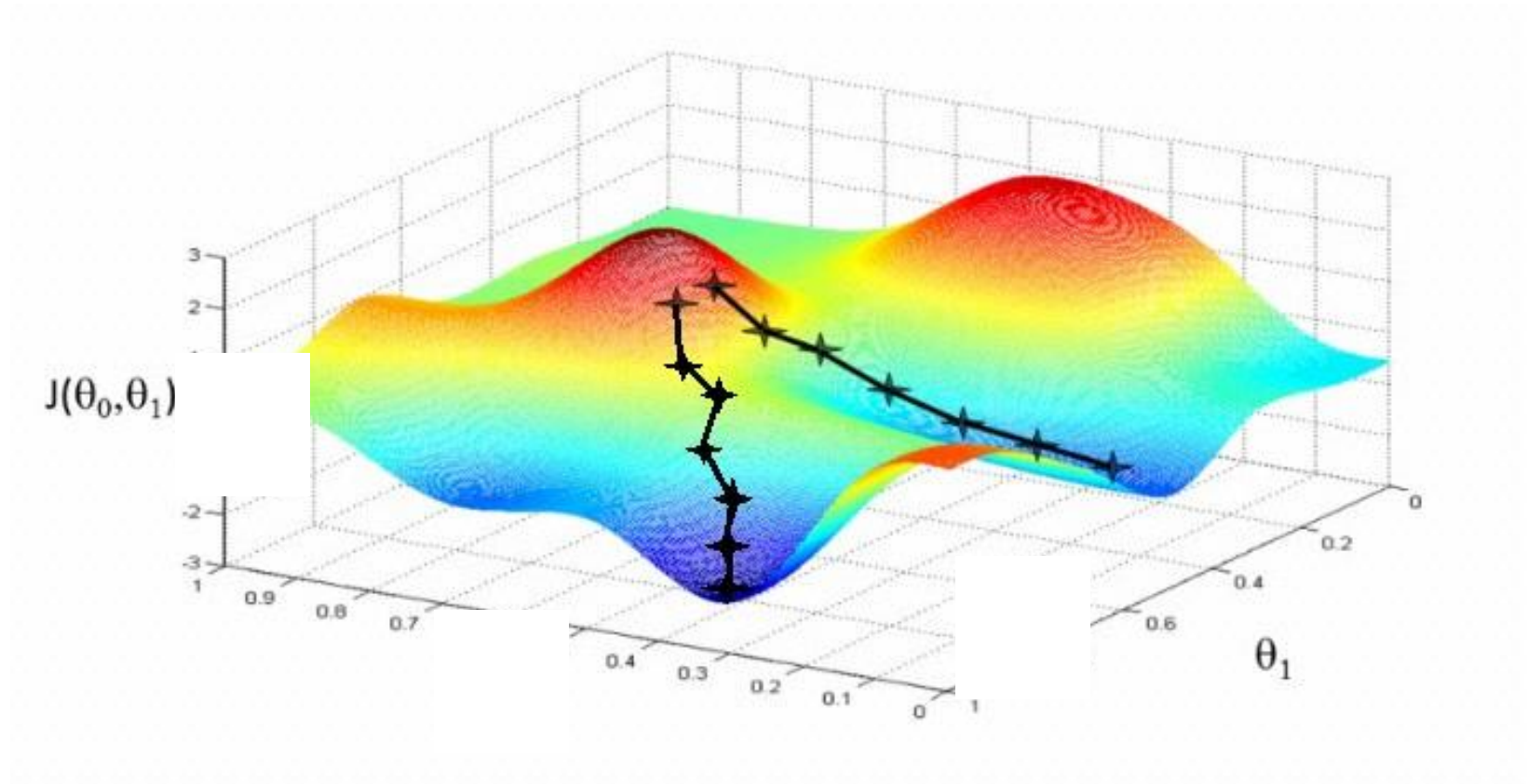
$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$

# Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$



# Convex function

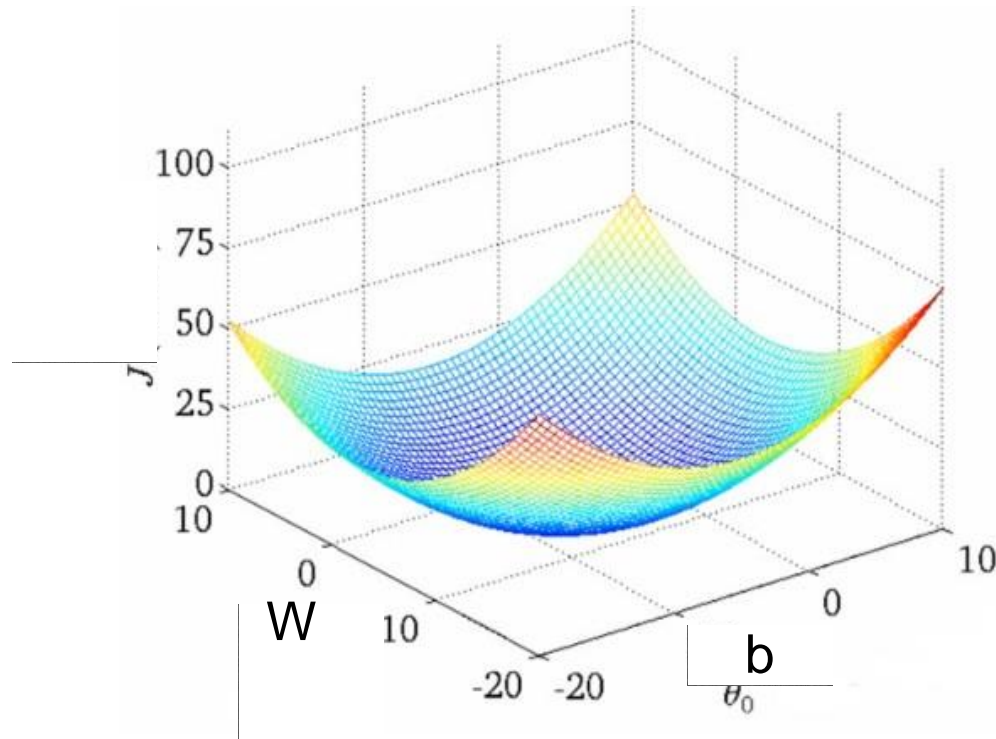


[www.holehouse.org/mlclass/](http://www.holehouse.org/mlclass/)

# Convex function

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$

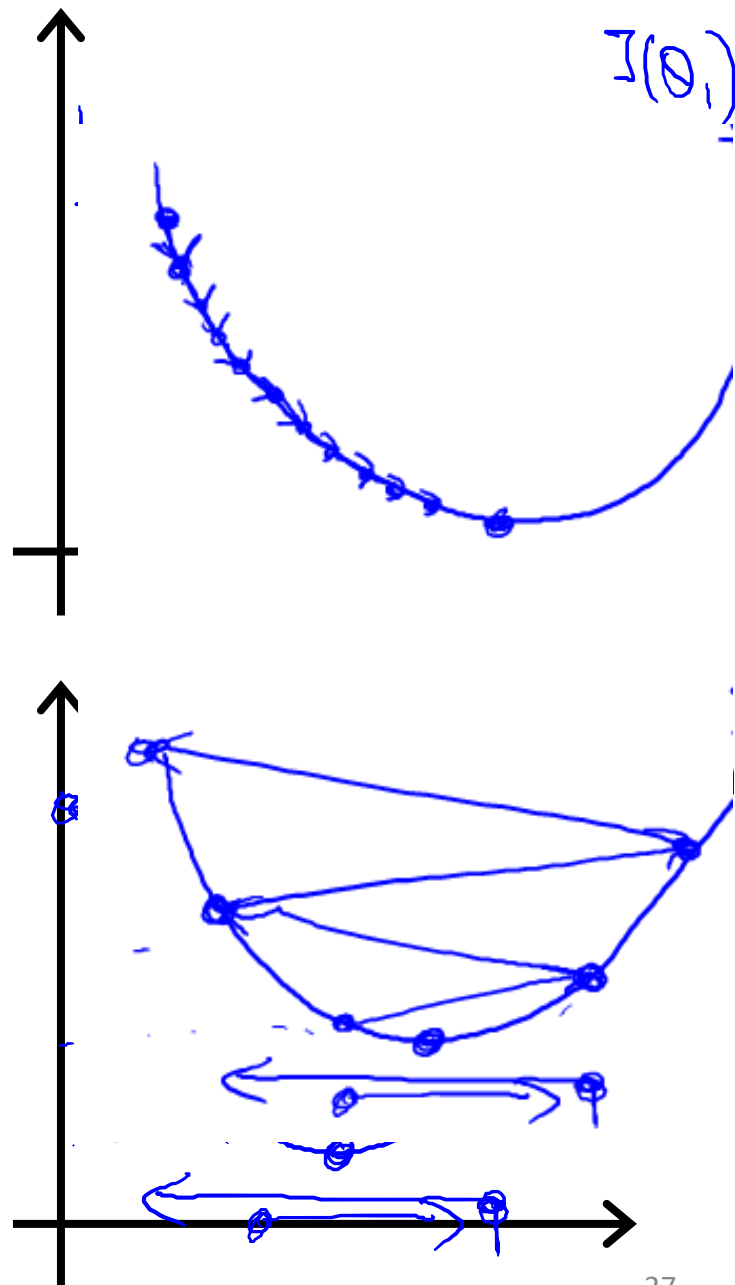
$cost(W, b)$



$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

If  $\alpha$  is too small, gradient descent can be slow.

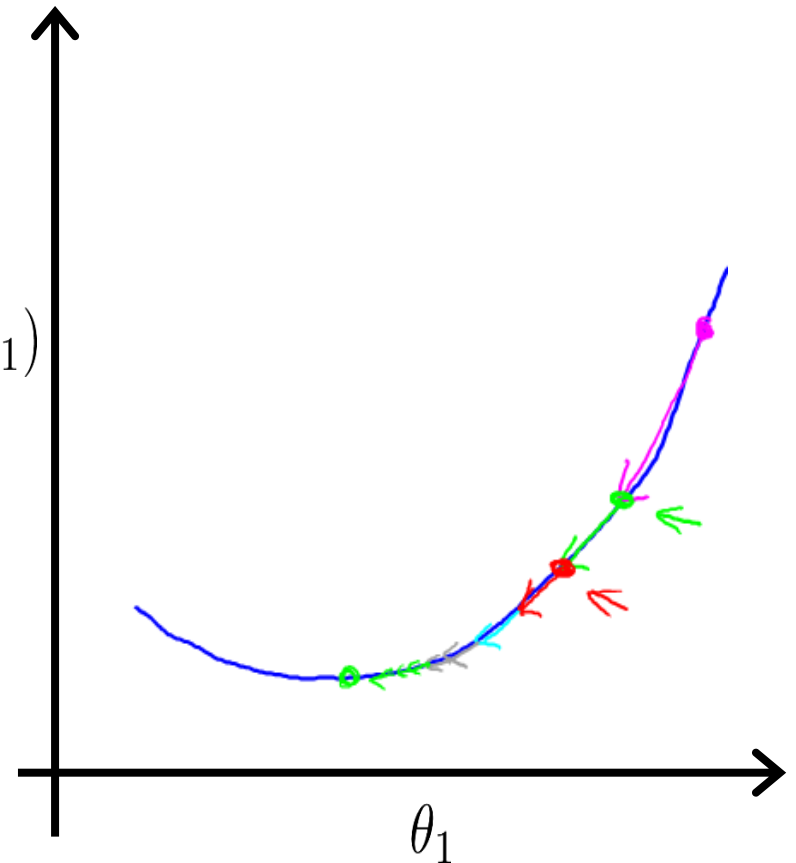
If  $\alpha$  is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Gradient descent can converge to a local minimum, even with the learning rate  $\alpha$  fixed.

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$

As we approach a local minimum  $J(\theta_1)$  of the cost function, gradient descent will automatically take smaller steps. So, no need to decrease  $\alpha$  over time.



# Multi-variable linear regression

Predicting exam score:  
regression using three inputs ( $x_1$ ,  $x_2$ ,  $x_3$ )

multi-variable/feature

$x_1$ (quiz 1)	$x_2$ (quiz 2)	$x_3$ (midterm 1)	Y (final)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

# Hypothesis

$$H(x) = Wx + b$$

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

# Cost function

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$cost(W, b) = \frac{1}{m} \sum_{I=1}^m (H(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}) - y^{(i)})^2$$

# Multi-variable

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$H(x_1, x_2, x_3, \dots, x_n) = w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b$$



# Matrix multiplication

The diagram shows the calculation of the dot product of the first row of the first matrix and the first column of the second matrix. A yellow curved arrow labeled "Dot Product" connects the first row of the first matrix to the first column of the second matrix. The first matrix is  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  and the second matrix is  $\begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$ . The result is shown as  $= \begin{bmatrix} 58 \end{bmatrix}$ . The numbers 1, 2, 3, 7, 8, 9, 10, and 58 are highlighted in yellow.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

<https://www.mathsisfun.com/algebra/matrix-multiplying.html>

# Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1w_1 + x_2w_2 + x_3w_3)$$

$$H(X) = XW$$

# Hypothesis using matrix

$$H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3$$

$x_1$	$x_2$	$x_3$	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

<https://www.symbolab.com/solver/matrix-calculator>

# Hypothesis using matrix

$$H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3$$

$x_1$	$x_2$	$x_3$	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1w_1 + x_2w_2 + x_3w_3)$$

$$H(X) = XW$$

Test Scores for General Psychology

<https://www.symbolab.com/solver/matrix-calculator>

# Many x instances

$x_1$	$x_2$	$x_3$	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1 w_1 + x_2 w_2 + x_3 w_3)$$

$x_1$	$x_2$	$x_3$	$Y$
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

## Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$$H(X) = XW$$

<https://www.symbolab.com/solver/matrix-calculator>

# Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[5, 3]

[3, 1]

[5, 1]

$$H(X) = XW$$

## Hypothesis using matrix

$$\begin{array}{ccc} \left( \begin{array}{|c|} \hline \mathbf{X} \\ \hline \end{array} \right) & \times & \left( \begin{array}{|c|} \hline \mathbf{W} \\ \hline \end{array} \right) = \left( \begin{array}{|c|} \hline \mathbf{H(X)} \\ \hline \end{array} \right) \\ [5, 3] & [?, ?] & [5, 1] \end{array}$$

$$H(X) = XW$$



# Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[n, 3]

[3, 1]

[n, 1]

$$H(X) = XW$$

# Hypothesis using matrix (n output)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \text{?} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3]    [?, ?]

[n, 2]

$$H(X) = XW$$

# Hypothesis using matrix (n output)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

$$[n, 3] \quad [3, 2]$$

$$[n, 2]$$

$$H(X) = XW$$

# Data preprocessing

# Input processing

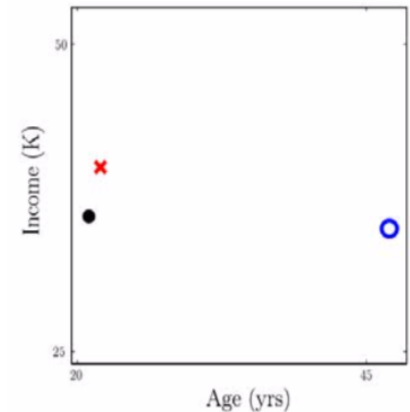
- Centering
- Normalizing
- Whitening

# Example

- Mr. Good and Mr. Bad were both given credit cards by the Bank of Learning (BoL)

	Mr. Good	Mr. Bad
(Age in years, Income in \$*1,000)	(47, 35)	(22,40)

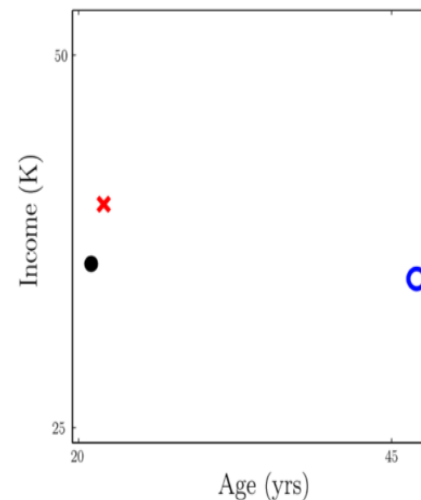
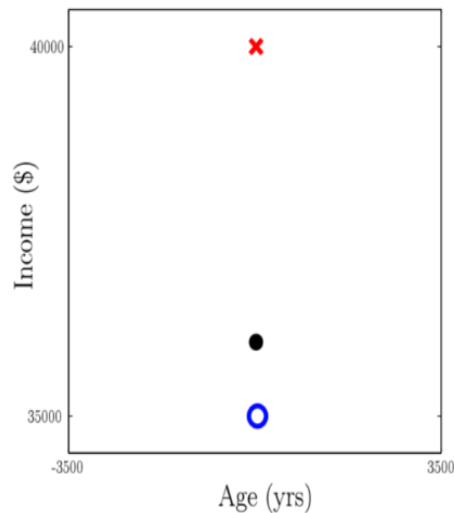
- Mr. Unknown who has “coordinates” (21 years, \$36k) applies for credit.
- Should the BoL give him credit, according to some learning algorithms (e.g. nearest neighbor algorithm)?



# Example

- What if, income is measured in dollars instead of “K” (thousands of dollars)?

	Mr. Good	Mr. Bad
(Age in years, Income in \$)	(47, 35000)	(22, 40000)



# Uniform treatment of dimensions

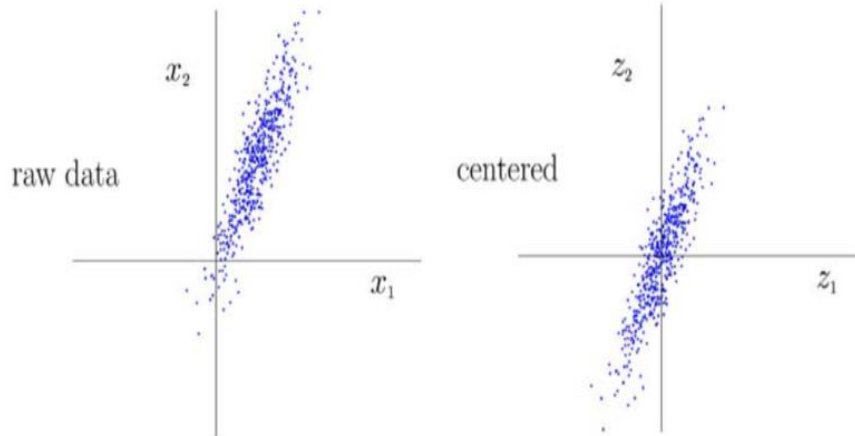
- Most learning algorithms treat each dimension equally

Nearear neighbor:  $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$

- **Input Preprocessing**
- Unless you want to emphasize certain dimensions, the data should be **preprocessed** to present each dimension on a similar scale



# Centering



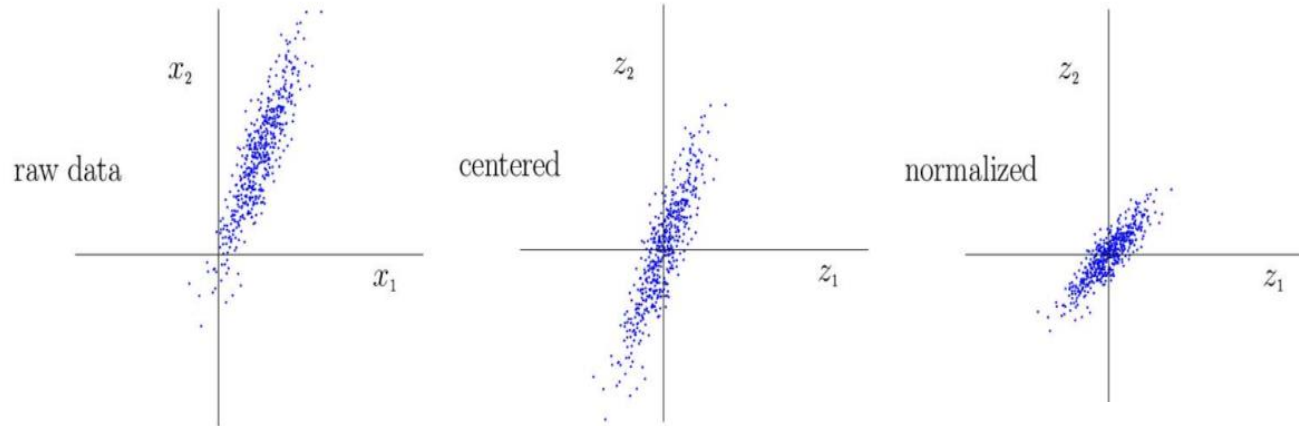
$\bar{\mathbf{x}} \rightarrow \text{mean of } \{ \vec{\mathbf{x}} \}$

$$\vec{\mathbf{z}}_n = \vec{\mathbf{x}}_n - \bar{\mathbf{x}}$$

$\hookrightarrow n: 1 \sim N$

$$\vec{\mathbf{z}} = \mathbf{0}$$

# Normalizing

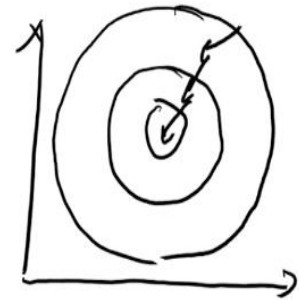


# Centering and Normalization - Feature Scaling

- Make sure features are on a similar scale  $\bar{E}_{in}, \bar{E}_{out}$  (cost function)

$x_1$ : size  $\rightarrow$  2000 sqft

$x_2$ : # of bedrooms  $\rightarrow$  2~5



- Get every feature into approximately  $-1 \leq x_i \leq 1$  range

$$x_0 = 1$$

$$0 \leq x_1 \leq 3 \quad \checkmark$$

$$-100 \leq x_2 \leq 200 \quad \times$$

$$-\frac{1}{3} \leq x_3 \leq 2 \quad \checkmark$$

# Standardization

$$X' = \frac{X - \mu}{\sigma}$$

- $\mu$  is the mean of the feature values and  $\sigma$  is the standard deviation of the feature values.
- Note that in this case, the values are not restricted to a particular range.

# Min-Max scaling

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- If the value of  $X$  is between the minimum and the maximum value, then the value of  $X'$  is between 0 and 1

# Centering & Normalization-Mean Normalization

- Replace  $x_i$  with  $x_i - \mu$  to make features have approximately zero mean  
(Do not apply to  $x_0 = 1$ )

E.g.  $x_1 = \frac{\text{size} - 1000}{2000}$

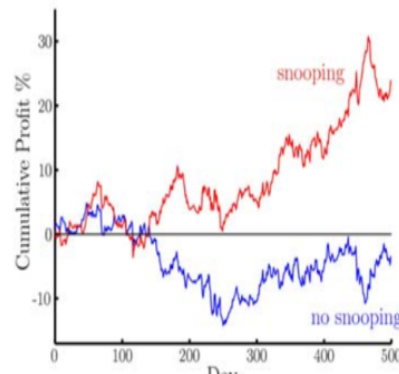
$$x_2 = \frac{\# \text{bedroom} - 2}{5}$$

(  $-0.5 \leq x_1 \leq 0.5$  ,  $-0.5 \leq x_2 \leq 0.5$

# Warning

## WARNING!

- Transforming data into a more convenient format has a hidden trap which leads to data snooping
- When using a test set, determine the input transformation from training data only
- Rule: lock away the test data until you have your final hypothesis



```
# Scale the numeric data by removing the mean and scaling to unit
# variance
sklearn.preprocessing.StandardScaler(copy=True, with_mean=True,
with_std=True)

scaler.fit(X[numeric_feature_labels])

X[numeric_feature_labels]=
scaler.transform(X[numeric_feature_labels])

# Split X into 70/30 partitions
X_train, X_remain, y_train, y_remain = train_test_split( X, y,
test_size = 0.3)

# Split X_remain into X_validate and X_test
X_validate, X_test, y_validate, y_test = train_test_split(X_remain,
y_remain, test_size=0.5)

# We now have a 70/15/15 split of X data

# Train a Logistic Regression Model
log_reg = LogisticRegression(solver='liblinear')
result = logreg.fit(X_train, y_train)

# Presume we have some custom function to optimize the model
optimized_model = custom_optimizer(log_reg, X_validate, y_validate)

# Finally, use the test data to evaluate the model
y_predict = optimized_model.predict(X_test)

# Compute the accuracy
prediction_accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
print("Accuracy: ", prediction_accuracy)
```



```
# Split X into 70/30 partitions
X_train, X_remain, y_train, y_remain = train_test_split( X, y,
test_size = 0.3)

# Split X_remain into X_validate and X_test
X_validate, X_test, y_validate, y_test = train_test_split(X_remain,
y_remain, test_size=0.5)

# We now have a 70/15/15 split of X data

# Scale only the training data
sklearn.preprocessing.StandardScaler(copy=True, with_mean=True,
with_std=True)
scaler.fit(X_train[numeric_feature_labels])
X_train[numeric_feature_labels]=
scaler.transform(X_train[numeric_feature_labels])

# Train a Logistic Regression Model
log_reg = LogisticRegression(solver='liblinear')
result = logreg.fit(X_train, y_train)

# Scale the validation data
scaler.transform(X_validate[numeric_feature_labels])
optimized_model = custom_optimizer(log_reg, X_validate, y_validate)

# Now scale the test data
scaler.transform(X_test[numeric_feature_labels])

# Finally, use the test data to evaluate the model
y_predict = optimized_model.predict(X_test)

# Compute the accuracy
prediction_accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
print("Accuracy: ", prediction_accuracy)
```

- **Training set**
  - biggest in terms of size set that is created out of the original dataset and is being used to fit the model.
- **Validation set**
  - Now the validation dataset is useful when it comes to hyper-parameter tuning and model selection.
  - A data scientist must try to determine the optimal hyperparameter values through trial and error. We call this process hyperparameter tuning.
  - For instance, in Deep Learning we use the validation set in order to find the optimal network layer size, the number of hidden units and the regularization term.
- **Test set**
  - Now that you have tuned the model by performing hyper-parameter optimisation, you should end up with the final model.
  - The testing set is used to evaluate the performance of this model and ensure that it can generalize well to new, unseen data points.
  - Compare the testing accuracy against the training accuracy in order to ensure that the model was not overfitted. This is the case when both accuracies are “close enough”.
  - When the training accuracy significantly outperforms testing accuracy, then there’s a good chance that overfitting has occurred.

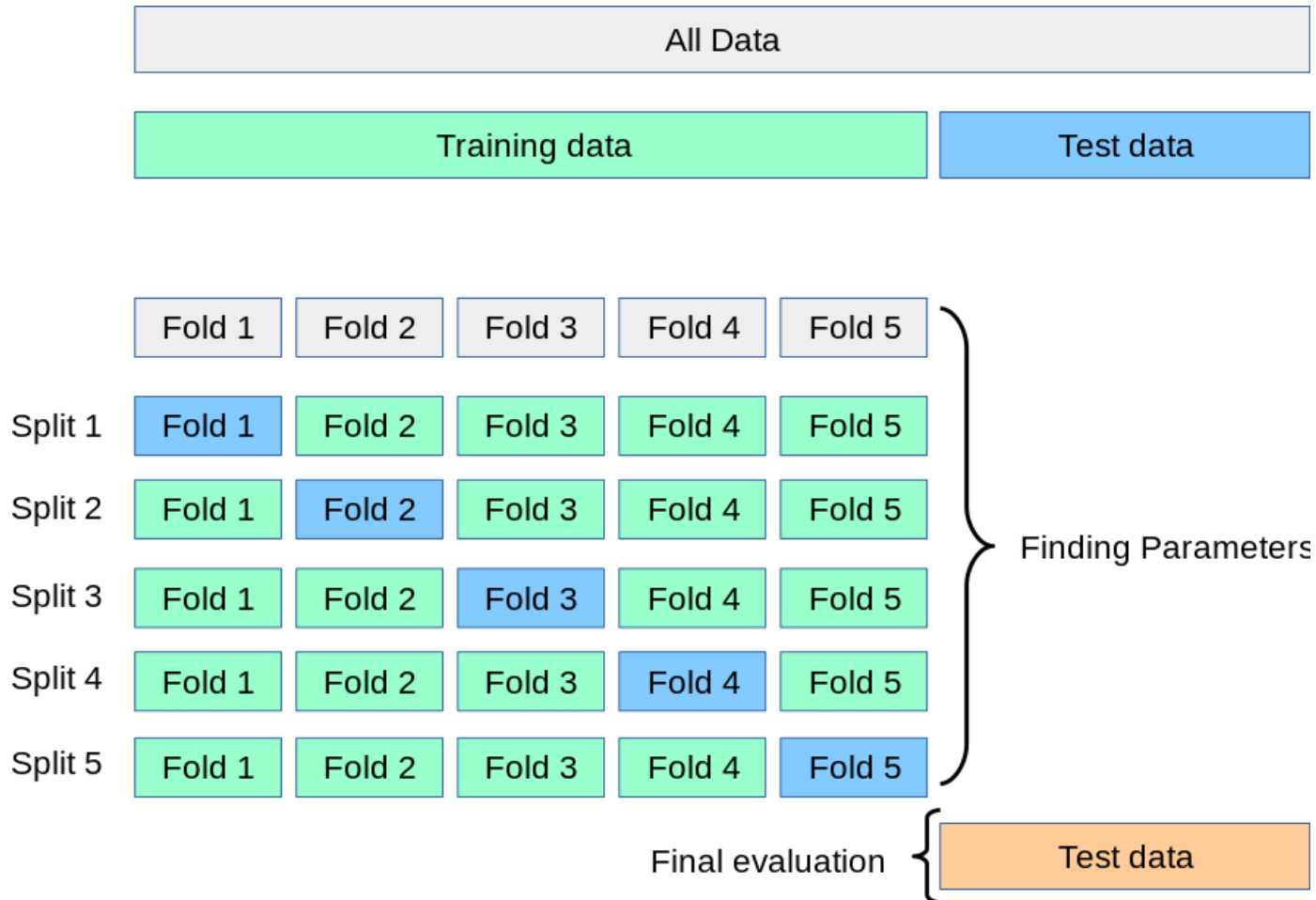
# Why Validation set?

- hyperparameter tuning.
- partitioning the data into three sets, it may drastically reduce the number of samples which can be used for training the model.
- In case that you don't have multiple models to select from, then validation set might be redundant.
- In this case, you just need a training and a testing set with a split ratio around 80:20 or 75:25.

# k-Fold cross-validation

- the training set is split into  $k$  smaller sets (or folds). The model is then trained using  $k-1$  of the folds and the last one is used as the validation set to compute a performance measure such as accuracy
- We typically choose either  $k=5$  or  $k=10$  as they find a nice balance between computational complexity and validation accuracy:

# k-Fold cross-validation



# Types of Data

- Numerical
  - Represents some sort of quantitative measurement
  - **Discrete data**: integer based, e.g., how many purchases did a customer make
  - **Continuous data**: has an infinite number of possible values, e.g., how much rain fell on a given day?
- Categorical
  - Qualitative data that has no inherent mathematical meaning
    - e.g., gender, State of Residence, Product category
  - You can assign numbers to categories to represent them
- Ordinal
  - A mixture of numerical and categorical
  - Categorical data that has mathematical meaning
    - E.g., movie ratings on a 1 – 5 scale (1 means it's a worse movie than a 2)

# Confusion Matrix

ACTUAL CLASS	PREDICTED CLASS		
		Yes	No
	Yes	TP	FN
	No	FP	TN

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$ErrorRate = 1 - accuracy$$

$$Precision = \text{Positive Predictive Value} = \frac{TP}{TP + FP}$$

$$Recall = \text{Sensitivity} = TP \text{ Rate} = \frac{TP}{TP + FN}$$

$$Specificity = TN \text{ Rate} = \frac{TN}{TN + FP}$$

$$FP \text{ Rate} = \alpha = \frac{FP}{TN + FP} = 1 - specificity$$

$$FN \text{ Rate} = \beta = \frac{FN}{FN + TP} = 1 - sensitivity$$

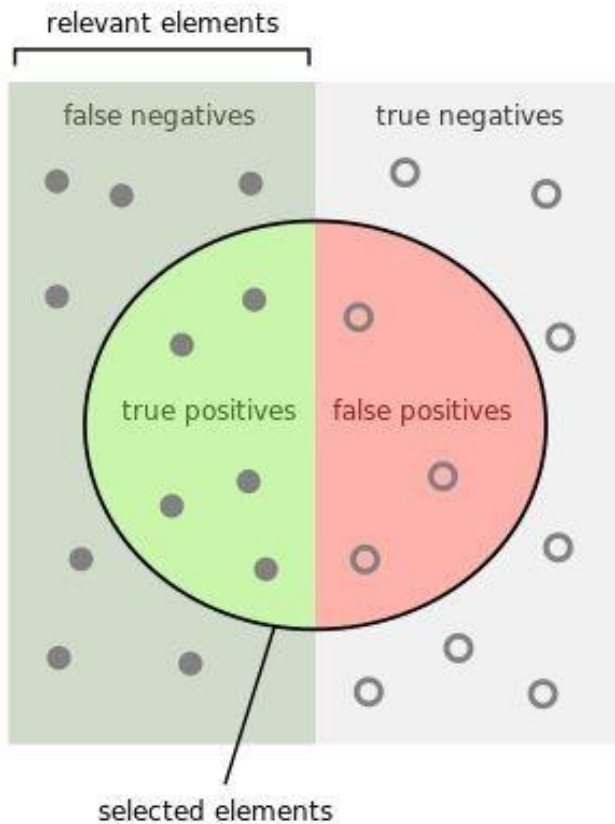
$$Power = sensitivity = 1 - \beta$$

# F1 Score

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

$$\textit{Accuracy} = \frac{TN + TP}{TN + FP + TP + FN}$$





$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{total}}$$

$$\text{precision} =$$

$$\text{recall} =$$

$$\text{F1 score} =$$

$$2 *$$

How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)

# Accuracy vs F-1 Score

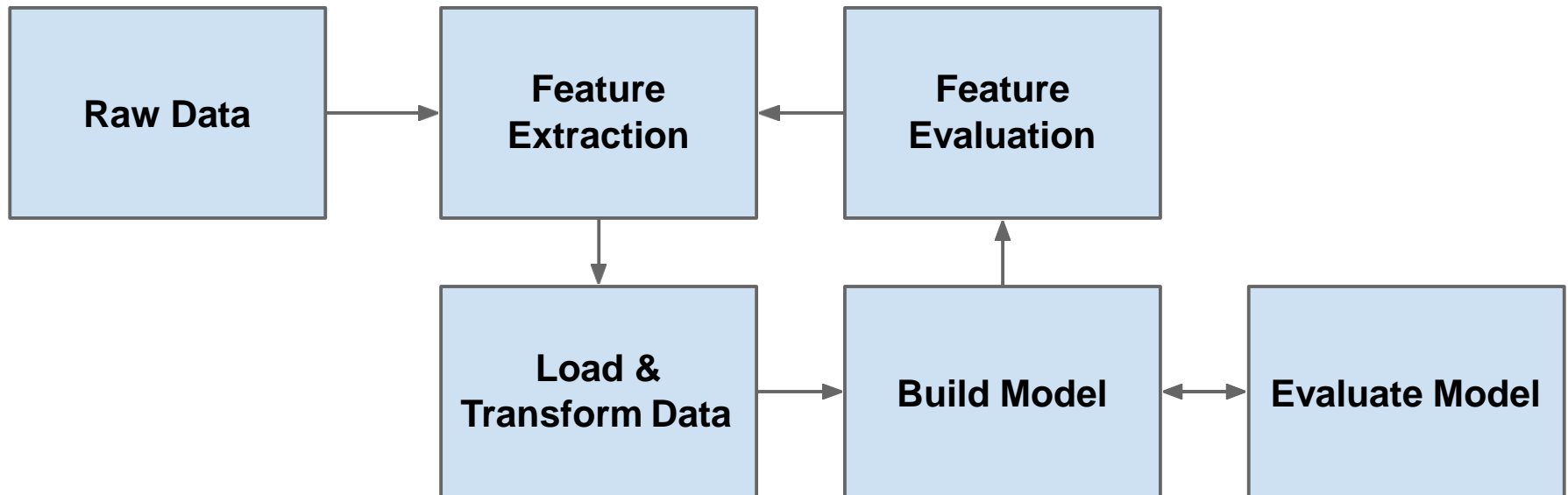
- **Accuracy** is used when the True Positives and True negatives are more important while **F1-score** is used when the False Negatives and False Positives are crucial
- A medical screening test?

# Missing Values

1. Do Nothing
2. Imputation Using (Mean/Median) Values
3. Imputation Using (Most Frequent) or (Zero/Constant) Values
4. Imputation Using k-NN

# Wrapping fit and predict

a development workflow:



```
class Transformer(Estimator):  
  
    def transform(self, X): """Transforms  
        the input data. """ # transform  
        ``X`` to ``X_prime`` return X_prime
```

```
from sklearn import preprocessing
```

```
Xt = preprocessing.normalize(X) # Normalizer  
Xt = preprocessing.scale(X)      # StandardScaler
```

```
imputer =Imputer(missing_values='Nan',  
                  strategy='mean') Xt  
= imputer.fit_transform(X)
```

Transformers

# MSE & Coefficient of Determination

In regressions we can determine how well the model fits by computing the mean square error and the coefficient of determination.

```
MSE = np.mean((predicted-expected)**2)
```

$R^2$  is a predictor of “goodness of fit” and is a value  $\in [0,1]$  where 1 is perfect fit.

```
from sklearn import metrics
from sklearn import cross_validation as cv

plits      = cv.train_test_split(X, y, test_size=0.2)
X_train, X_test, y_train, y_test = splits

model      = RegressionEstimator()
model.fit(X_train, y_train)

expected   = y_test
predicted  = model.predict(y_test)

print metrics.mean_squared_error(expected, predicted)
print metrics.r2_score(expected, predicted)
```

## K-Part Cross Validation

# Pipelines

`sklearn.pipeline.Pipeline(steps)`

- Sequentially apply *repeatable* transformations to final estimator that can be validated at every step.
- Each step (except for the last) must implement Transformer, e.g. `fit` and `transform` methods.
- Pipeline itself implements both methods of Transformer and Estimator interfaces.

<https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>



```
>>> from sklearn.svm import SVC
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.pipeline import Pipeline
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, ... random_state=0)
>>> pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])
>>> # The pipeline can be used as any other estimator
>>> # and avoids leaking the test set into the train set
>>> pipe.fit(X_train, y_train)
>>> pipe.score(X_test, y_test) 0.88
```

```
>>> from sklearn.naive_bayes import GaussianNB
>>> from sklearn.preprocessing import StandardScaler
>>> make_pipeline(StandardScaler(), GaussianNB(priors=None))
>>> Pipeline(steps=[('standardscaler', StandardScaler()), ('gaussiannb', GaussianNB())])
```

# Pipelined Feature Extraction

The most common use for the Pipeline is to combine multiple feature extraction methodologies into a single, repeatable processing step.

- `FeatureUnion`
- `SelectKBest`
- `TruncatedSVD`
- `DictVectorizer`

```
>>> from sklearn.datasets import load_digits
>>> from sklearn.feature_selection import SelectKBest, chi2
>>> X, y = load_digits(return_X_y=True)
>>> X.shape (1797, 64)
>>> X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
>>> X_new.shape (1797, 20)
```

[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.SelectKBest.html?highlight=selectkbest#sklearn.feature\\_selection.SelectKBest](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html?highlight=selectkbest#sklearn.feature_selection.SelectKBest)