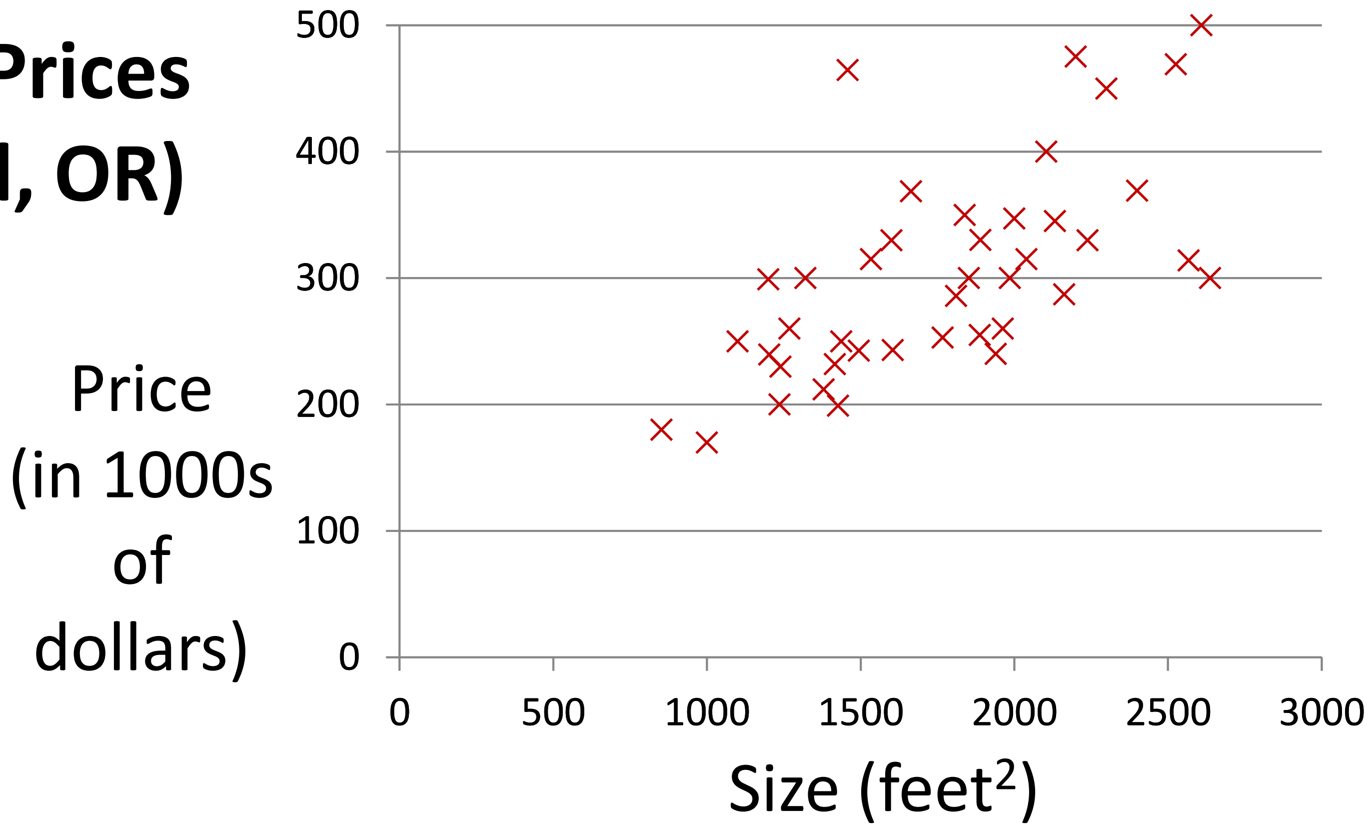
Lecture 2

Linear Regression

Housing Prices (Portland, OR)

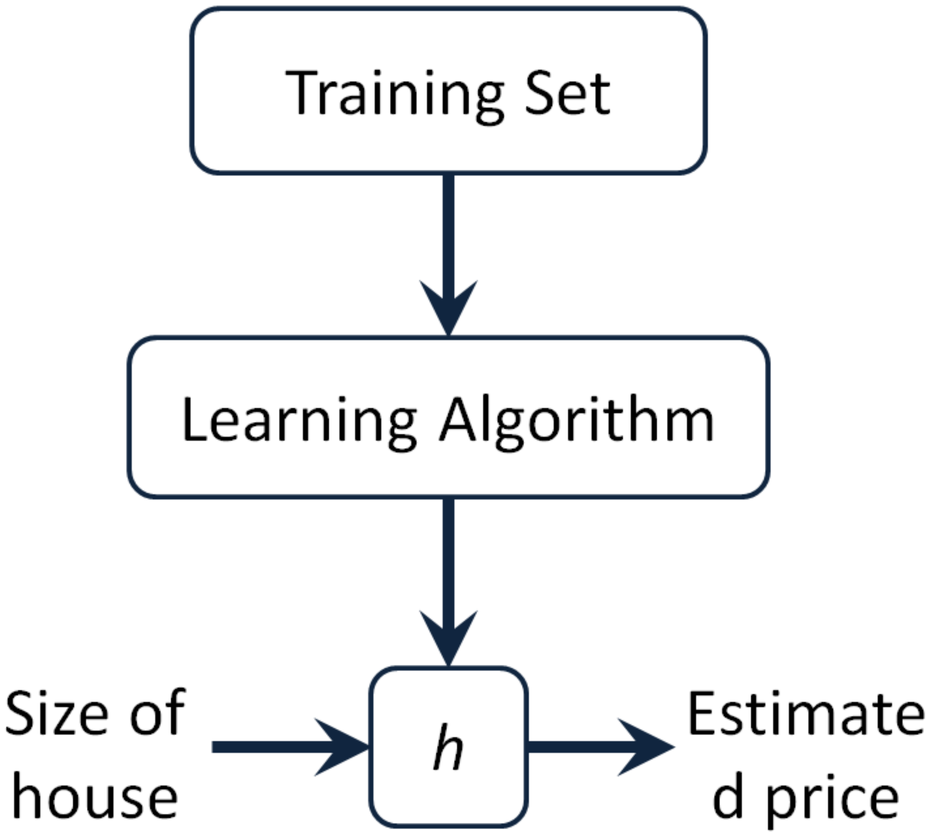


<u>Supervised Learning</u>

Given the "right answer" for each example in the data.

<u>Regression Problem</u>

Predict real-valued output



x

hypothesis

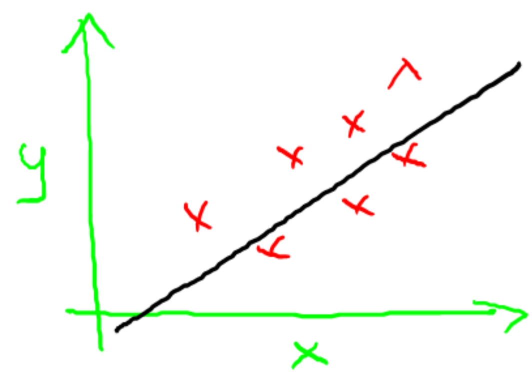
(estimated value of y)

h maps from x's to y's.

How do we represent *h* ?

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

Shorthand: $h(x)$



$h(x) = \theta_0 + \theta_1 x$

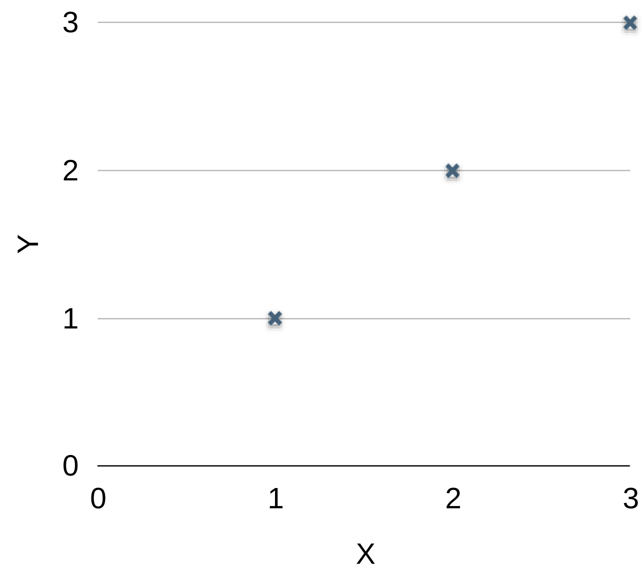
Linear regression with one variable.
Univariate linear regression.

Regression (data)

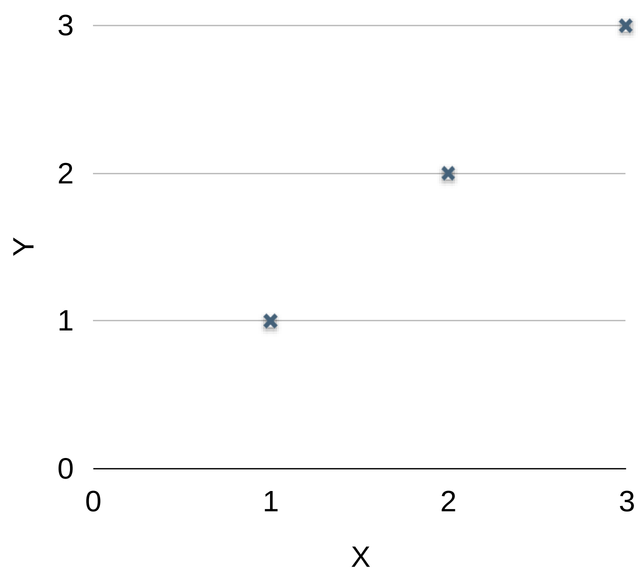
x	y
1	1
2	2
3	3

Regression (presentation)

x	Y
1	1
2	2
3	3

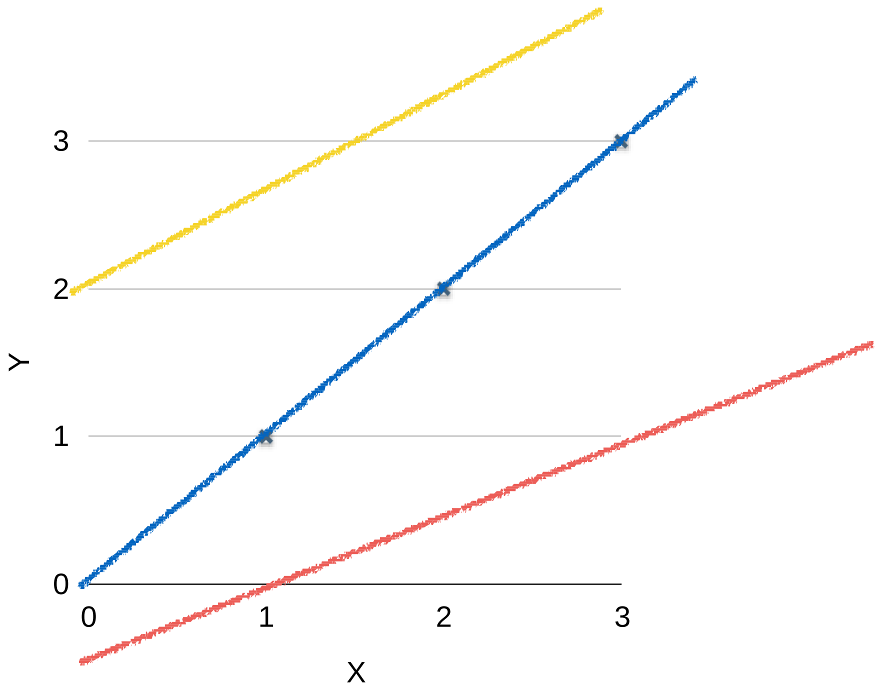


(Linear) Hypothesis

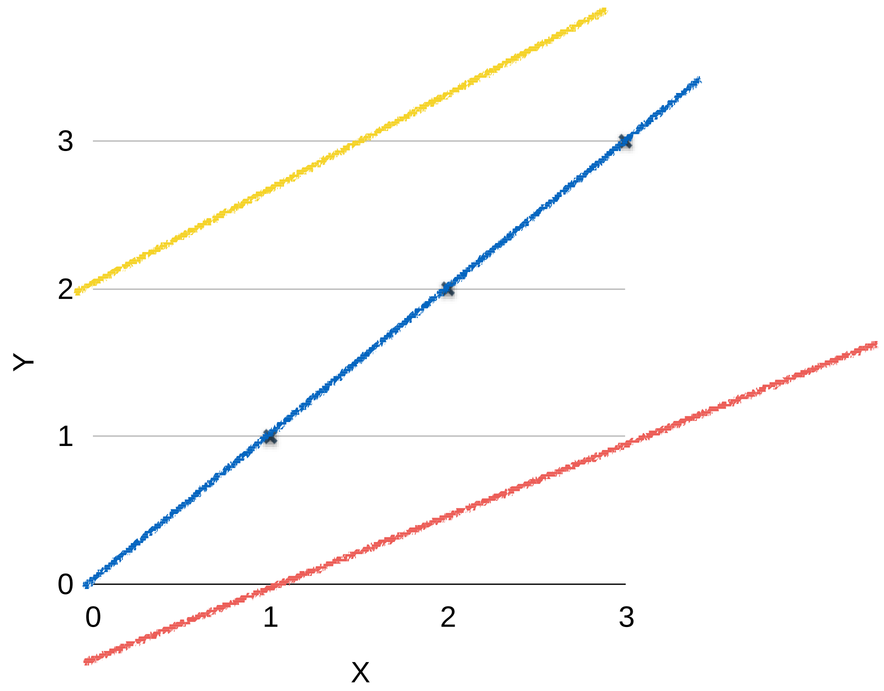


(Linear) Hypothesis

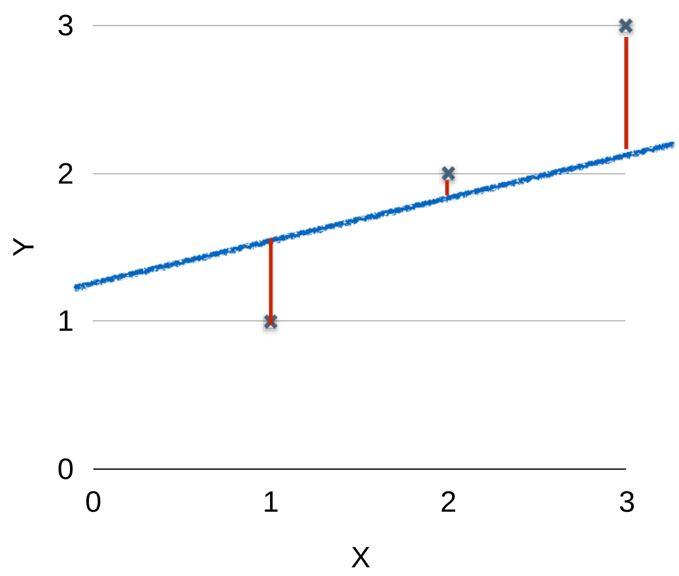
$$H(x) = Wx + b$$



Which hypothesis is better?



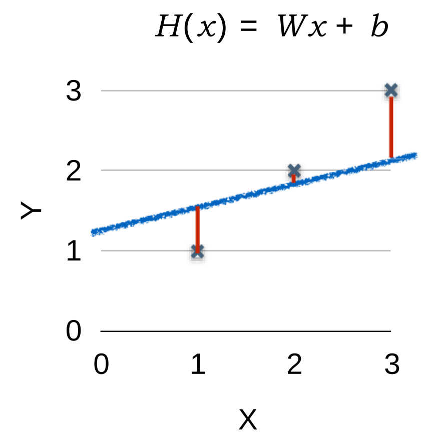
Which hypothesis is better?



Cost function

- How fit the line to our (training) data

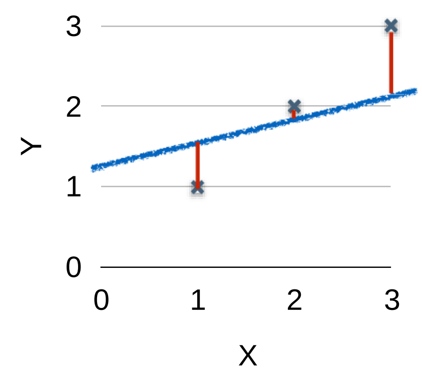
  $H(x) \quad y$



Cost function

- How fit the line to our (training) data

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



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

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

Hypothesis and Cost

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

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

What *cost(W)* looks like?

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

X	Y
1	1
2	2
3	3

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

What *cost(W)* looks like?

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

X	Y
1	1
2	2
3	3

- W=1, cost(W)=0

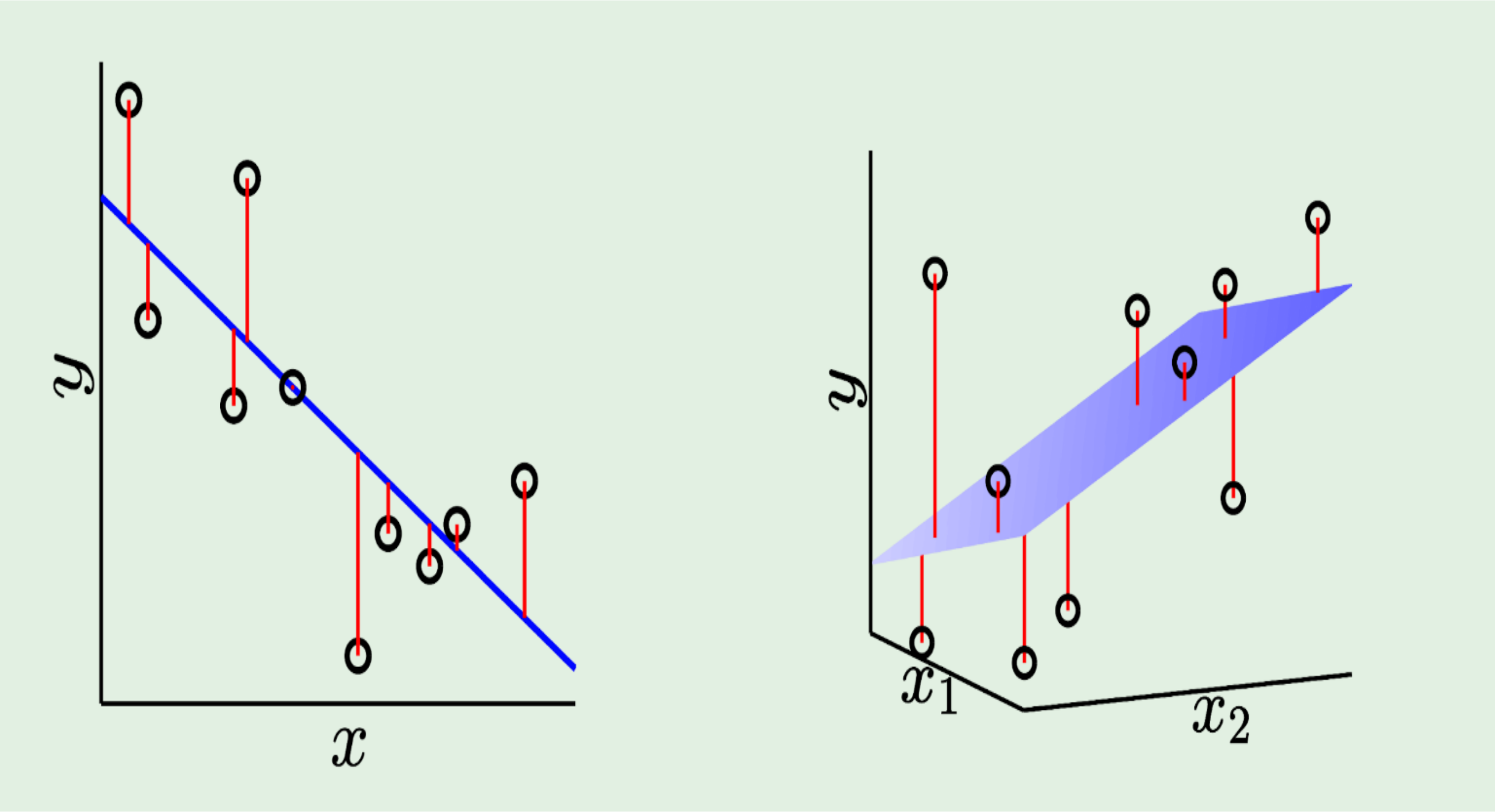
  $\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$

- W=0, cost(W)=4.67

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

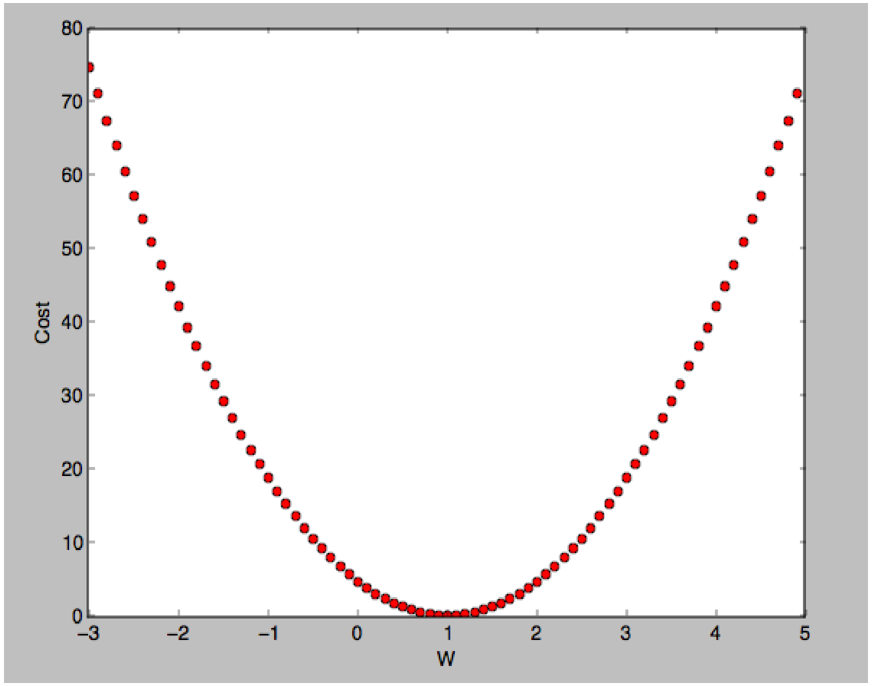
- W=2, cost(W)=?

Illustration of linear Regression



What *cost(W)* looks like?

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

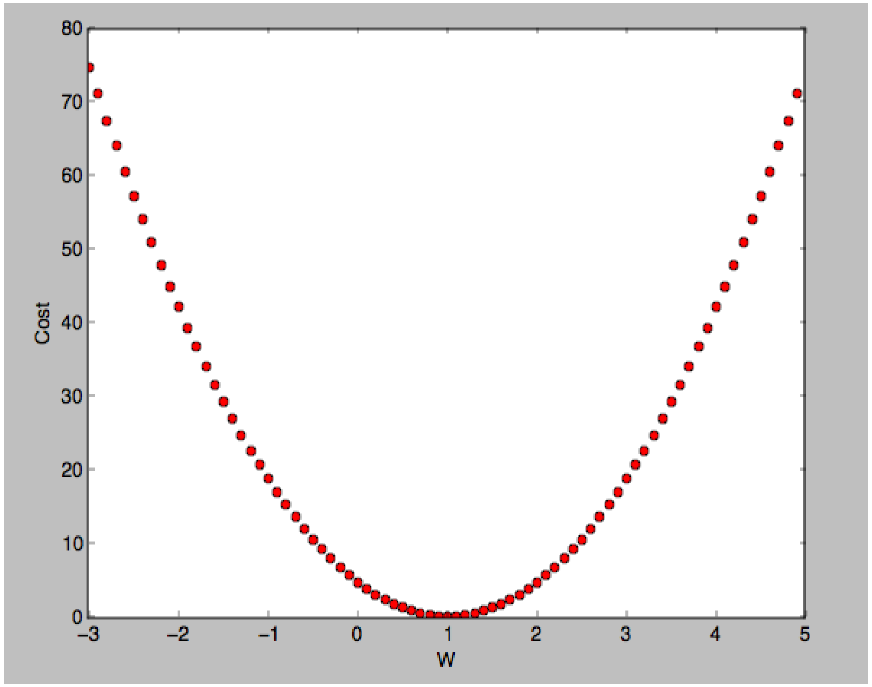


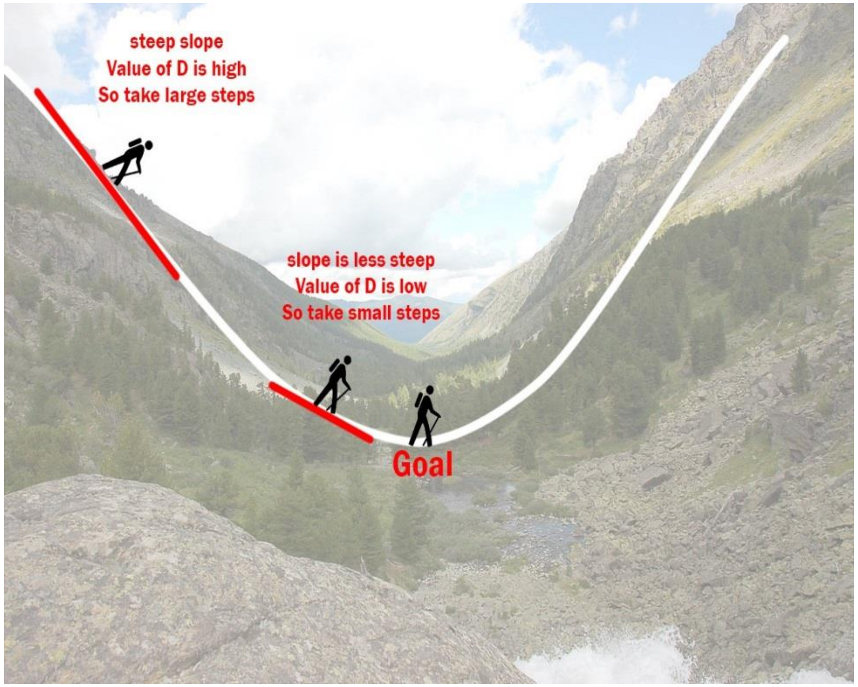
Gradient descent algorithm

- 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, ...)*

How it works?

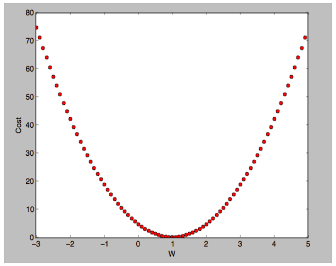
How would you find the lowest point?





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 cost(W, b)



- Each time you change the parameters, you select the gradient which reduces 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

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

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

$$\Downarrow$$

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

Formal definition

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

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

Formal definition

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

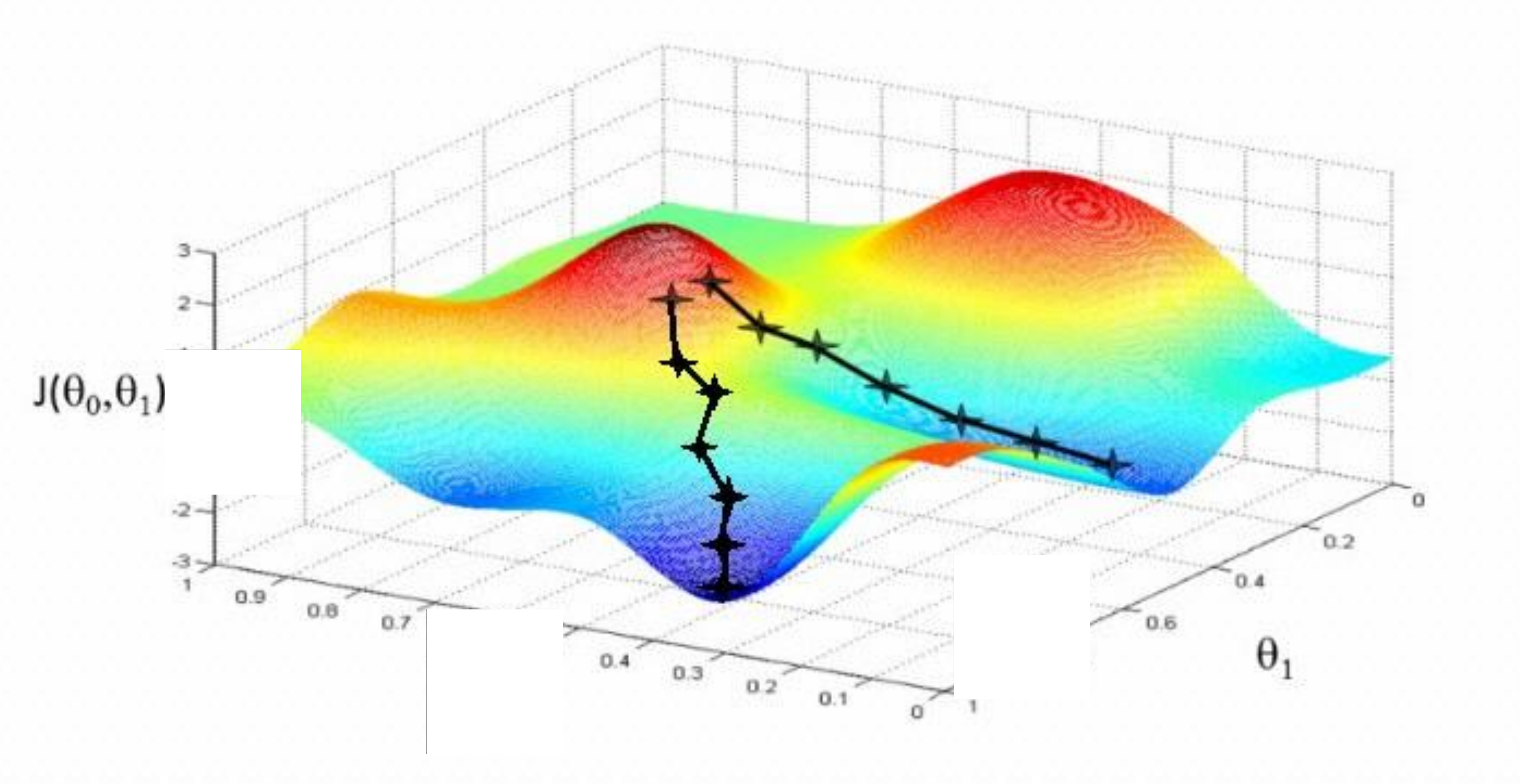
$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^{m} 2(W x^{(i)} - y^{(i)}) x^{(i)}$$

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

Gradient descent algorithm

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

Convex function

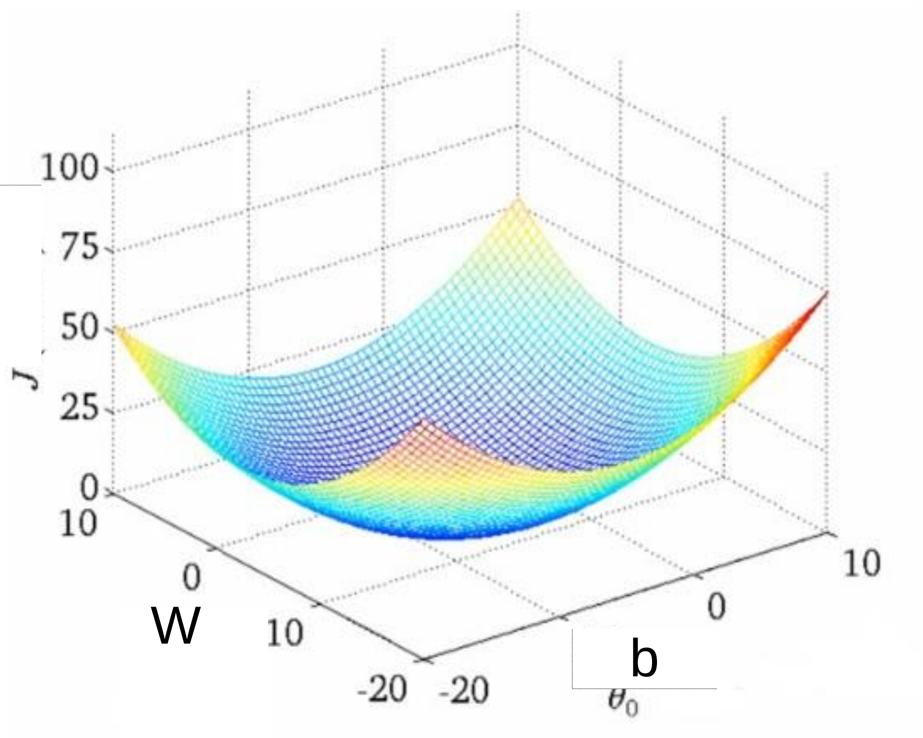


www.holehouse.org/mlclass/

Convex function

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

cost(W, b)

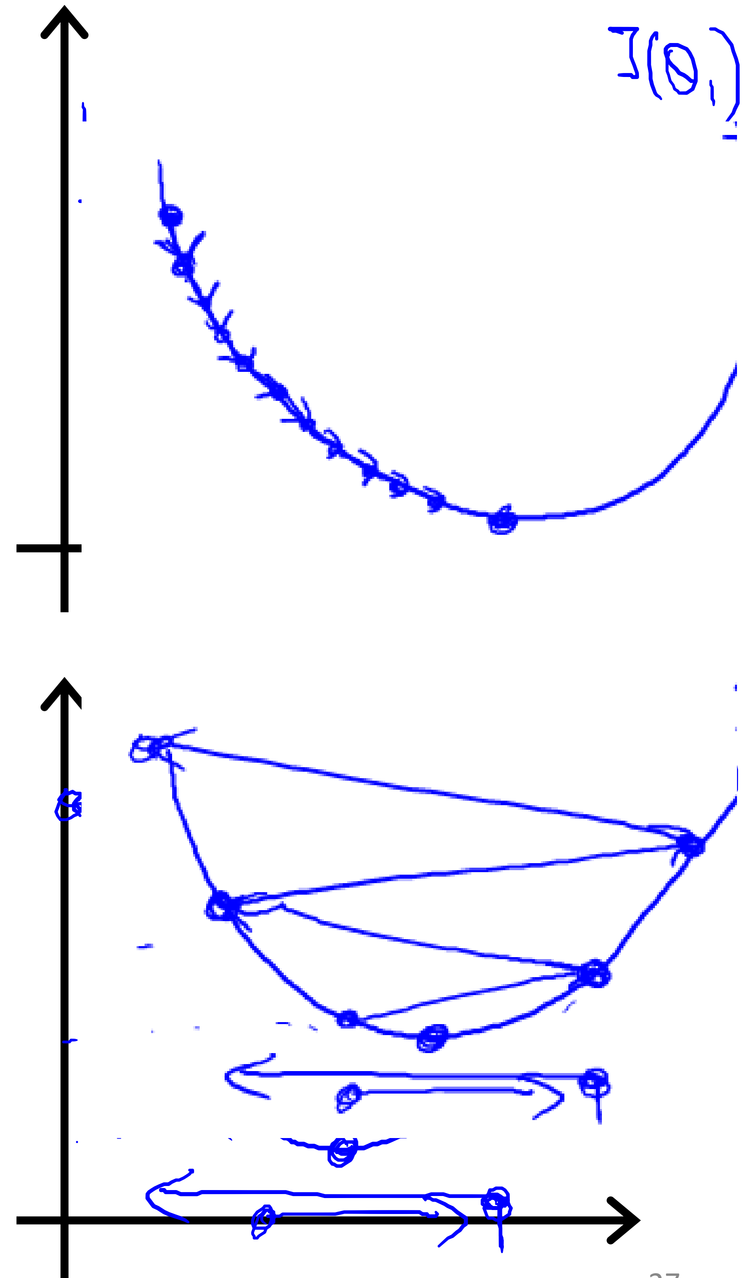


www.holehouse.org/mlclass/

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

If α is too small, gradient descent can be slow.

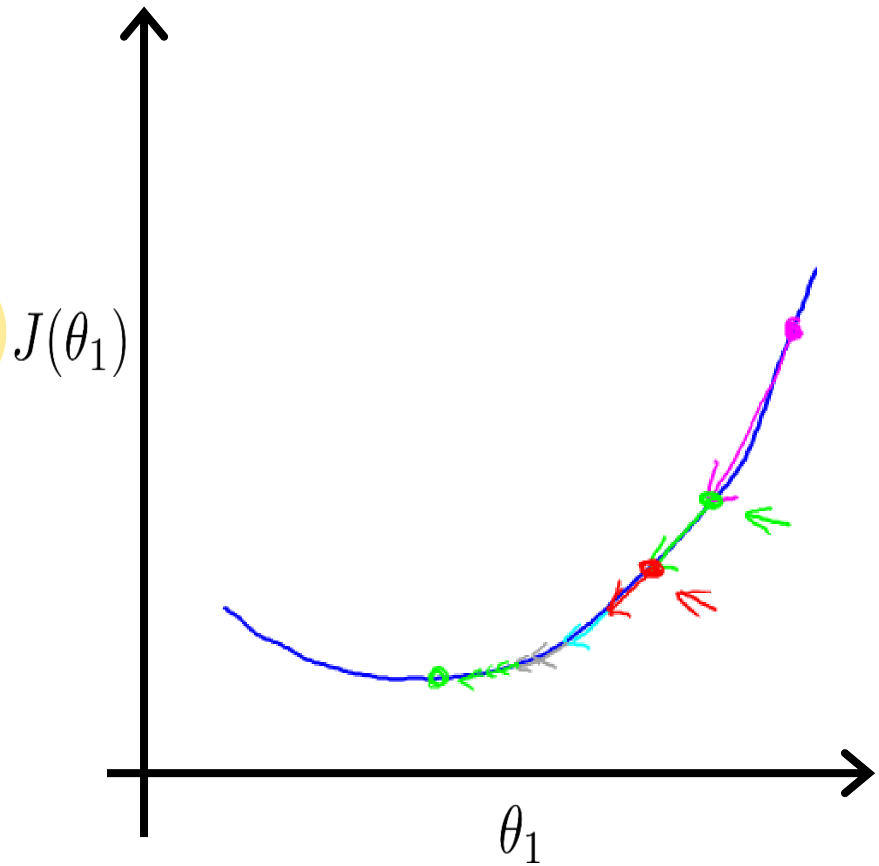
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



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

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

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Multi-variable linear regression

Predicting exam score:
regression using three inputs (x1, x2, x3)

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

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

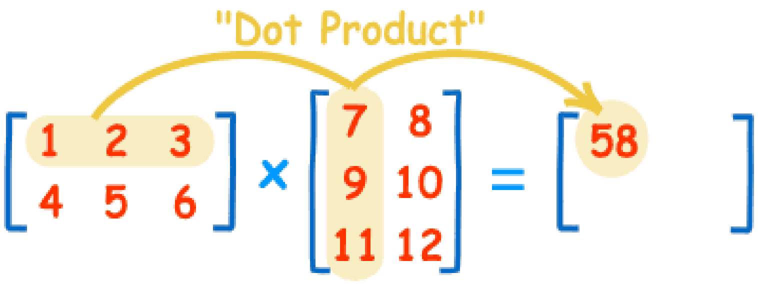
$$cost(W, b) = \frac{1}{m}\sum_{I=1}^{m}(H(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}) - y^{(i)})^2$$

Multi-variable

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

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

Matrix multiplication



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

Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n$$

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

$$H(X) = XW$$

Hypothesis using matrix

$$H(x_1, x_2, x_3) = x_1 w_1 + x_2 w_2 + x_3 w_3$$

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

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

Test Scores for General Psychology

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

$$H(X) = XW$$

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

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

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

$$w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n$$

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

$$H(X) = XW$$

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

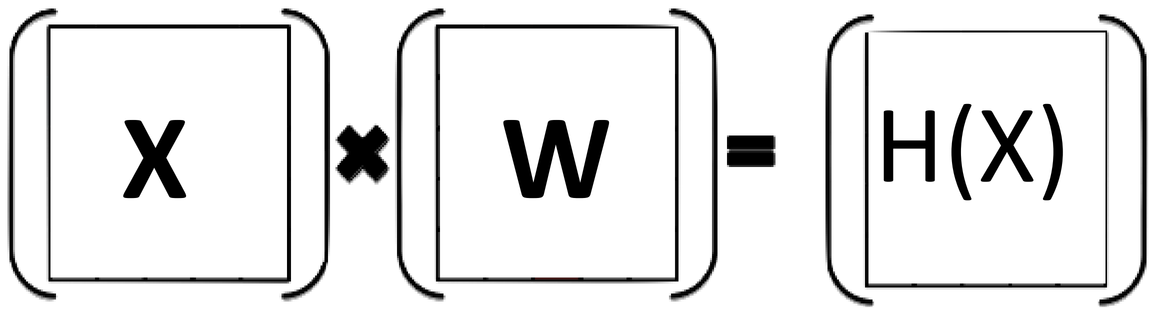
Hypothesis using matrix

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

[5, 3] **[3, 1]** [5, 1]

$$H(X) = XW$$

Hypothesis using matrix



[5, 3]   [?, ?]   [5, 1]

$$H(X) = XW$$

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)

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

[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] **[3, 2]** [n, 2]

$$H(X) = XW$$

Data preprocessing

Input processing

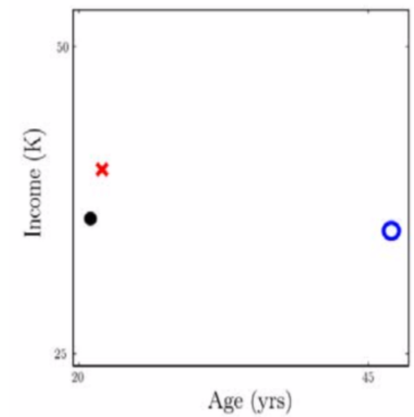
- Centering
- Normalizing
- Whitening

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)

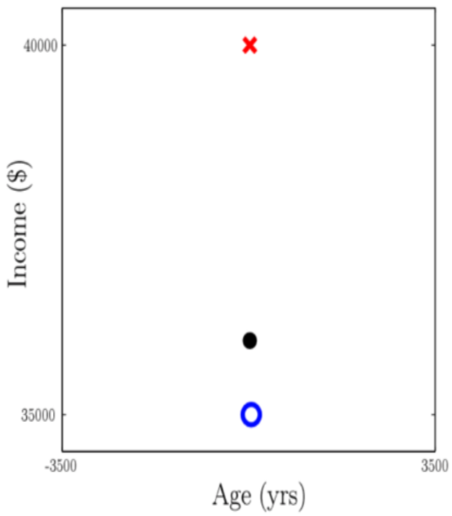
- Mr. Unknown who has “coordinates” (21 years, $36k) applies for credit.
- Should the BoL give him credit, according to some learning algorithms (e.g. nearest neighbor algorithm)?

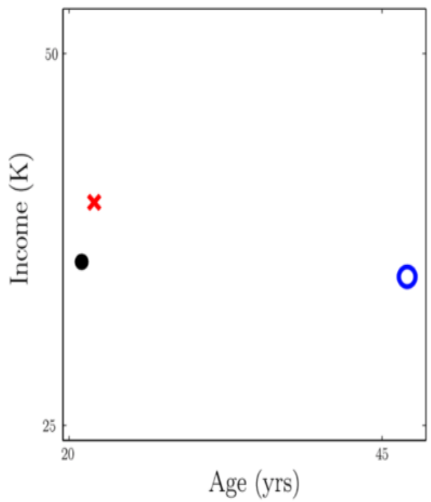


Example

- 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)





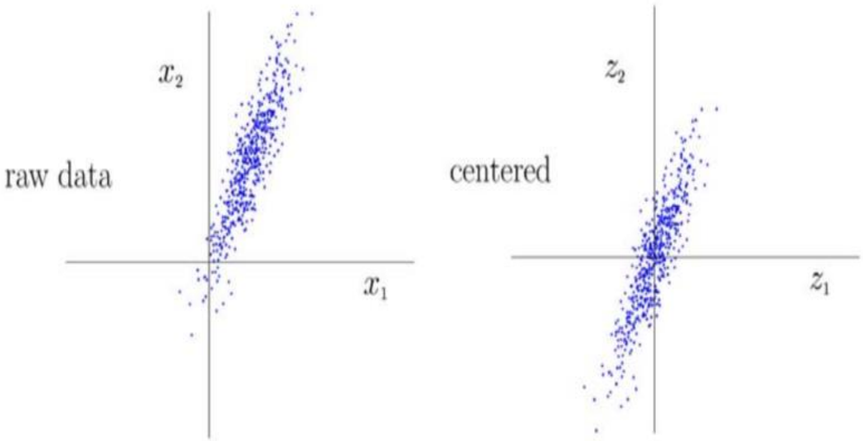
Uniform treatment of dimensions

- Most learning algorithms treat each dimension equally

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

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

Centering



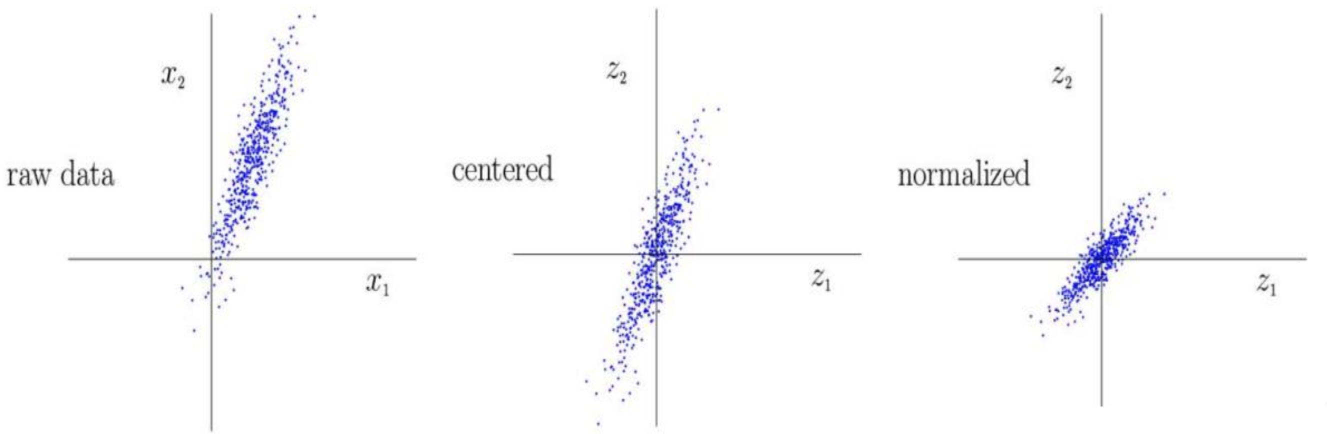
$\bar{x} \to$ mean of $\{\vec{x}\}$

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

$n: 1 \sim N$

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

Normalizing



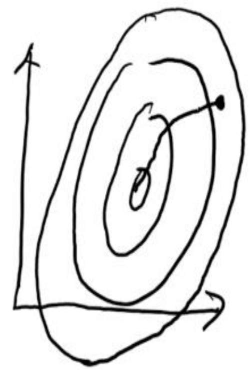
Centering and Normalization - Feature Scaling

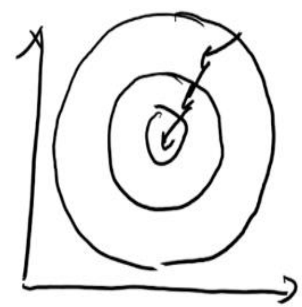
- Make sure features are on a similar scale

E_{in}, E_{out} (cost function)

x_1 : size → 2000 sqft

x_2 : # of bedrooms → 2~5





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

$x_0 = 1$

$0 \le x_1 \le 3$ ✓

$-100 \le x_2 \le 200$ ✗

$-\frac{1}{3} \le x_3 \le 2$ ✓

Standardization

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

- u 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

$$x_1 = \frac{size - 1000}{2000}$$

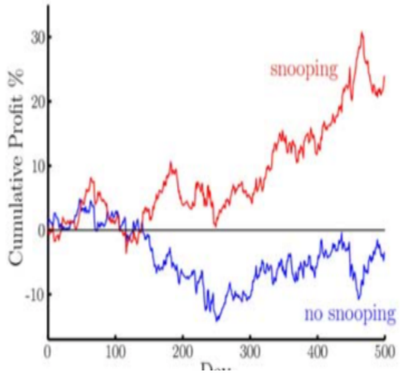
$$x_2 = \frac{\#bedroom - 2}{5}$$

$$-0.5 \le x_1 \le 0.5, \quad -0.5 \le x_2 \le 0.5$$

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.

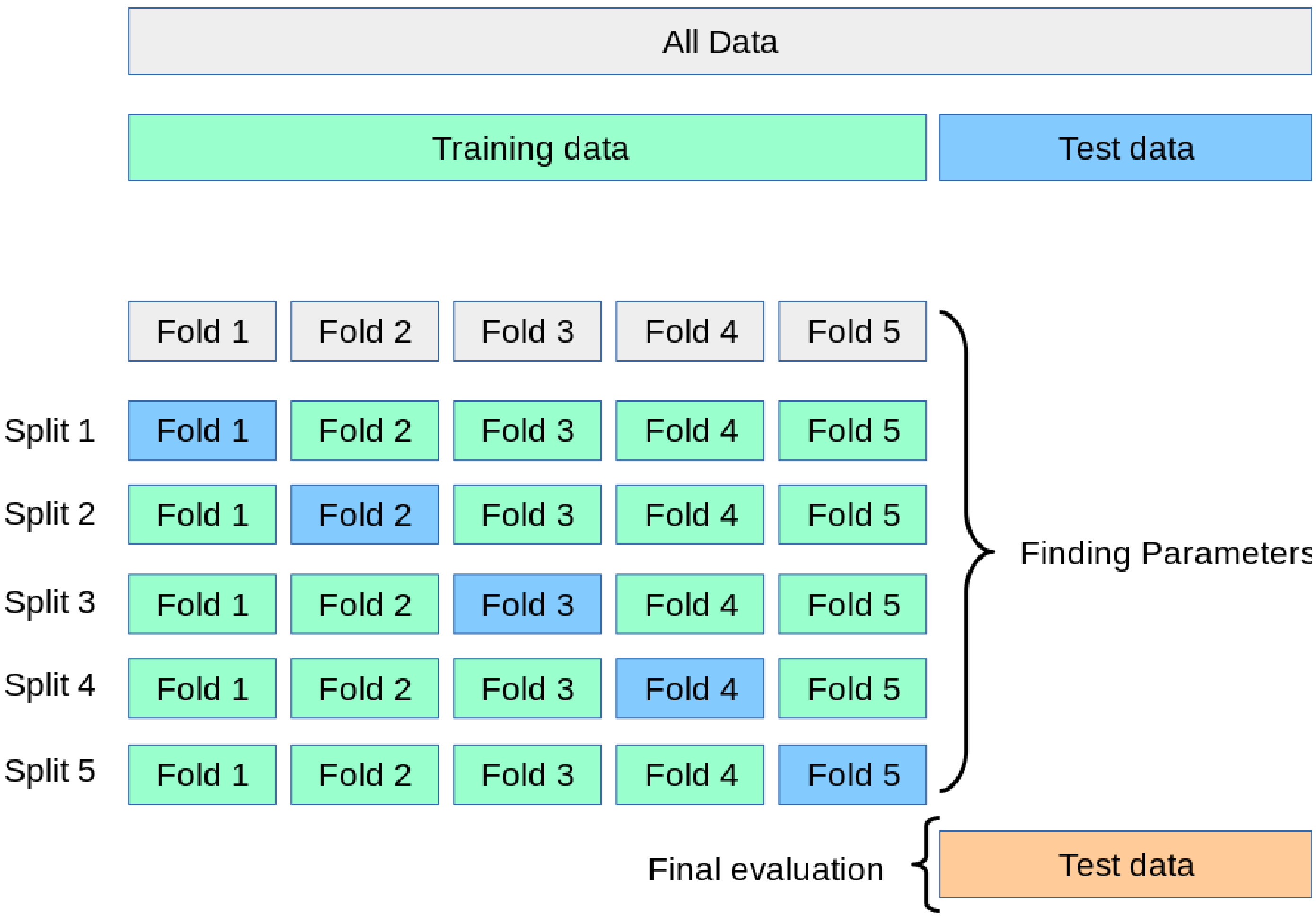
Why Validation set?

- 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 i=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 made
  - **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 categorial
  - 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

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

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

$$ErrorRate = 1 - accuracy$$

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

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

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

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

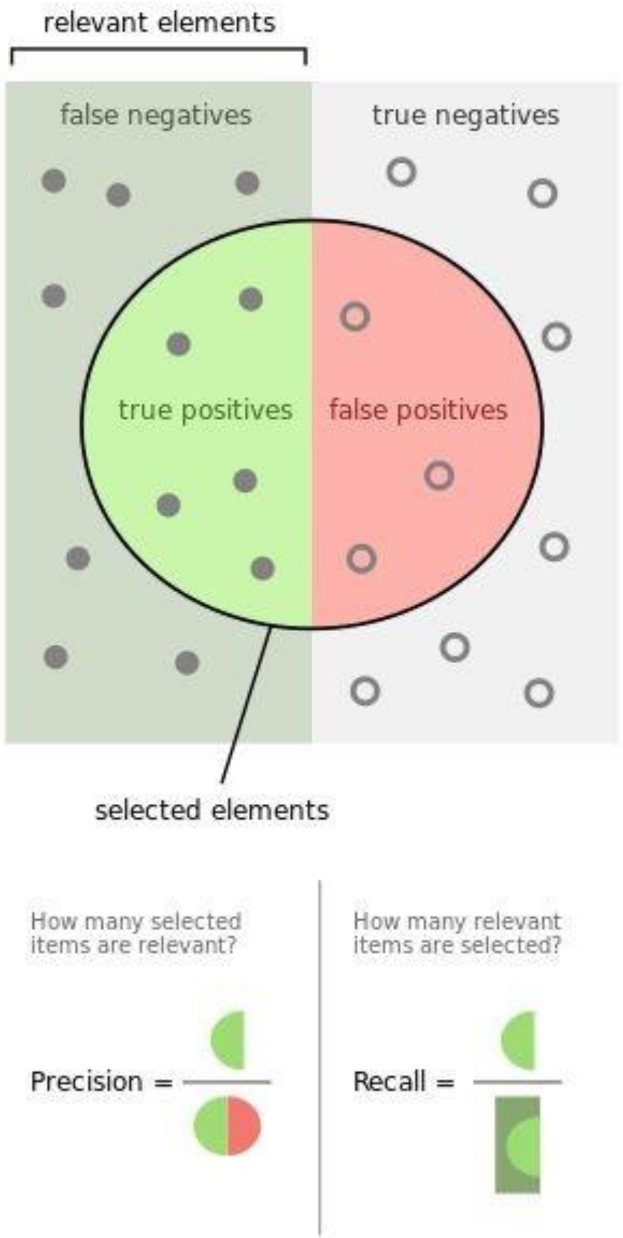
$$FN\ Rate = \beta = \frac{FN}{FN + TP} = 1 - sensitivity$$

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

F1 Score

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

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



accuracy =

true positives + true negatives / total

precision =

recall =

F1 score =

2 *

https://en.wikipedia.org/wiki/Precision_and_recall

Accuracy vs F-1 Score

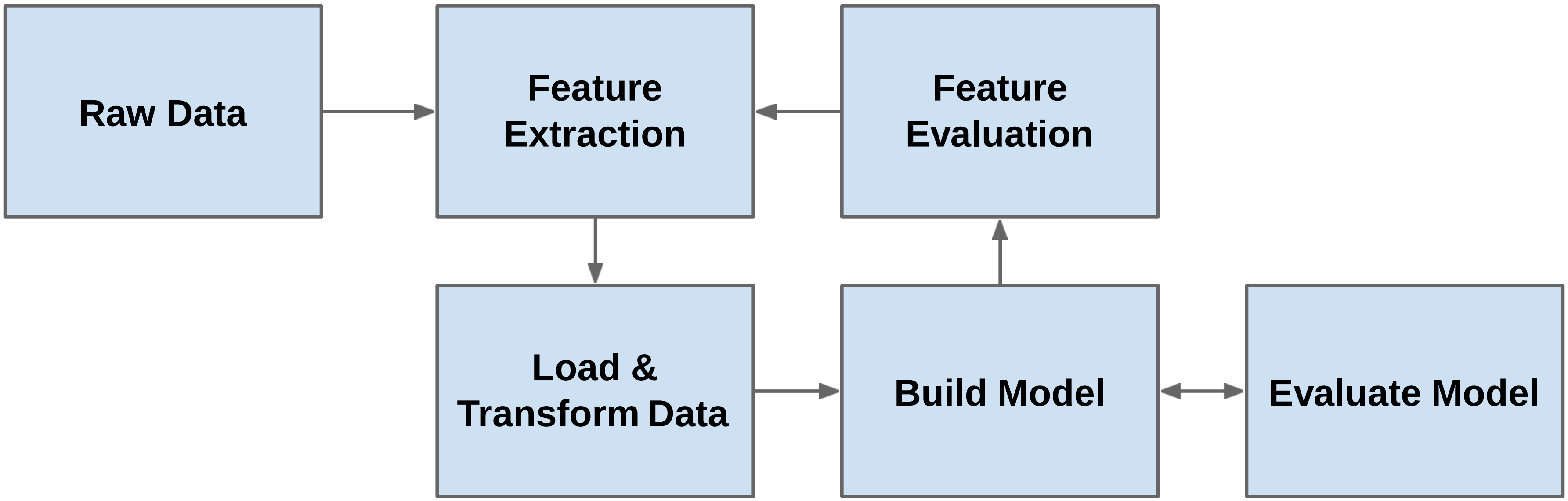
- **Accuracy** is used when the True Positives and True negatives are more important while **F1-score** is used when the False Negatives and False Positives are crucial
- A medical screening test?

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.

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

splits       = 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) Pipeline(steps=[('scaler', StandardScaler()), ('svc',
SVC())])
>>> 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