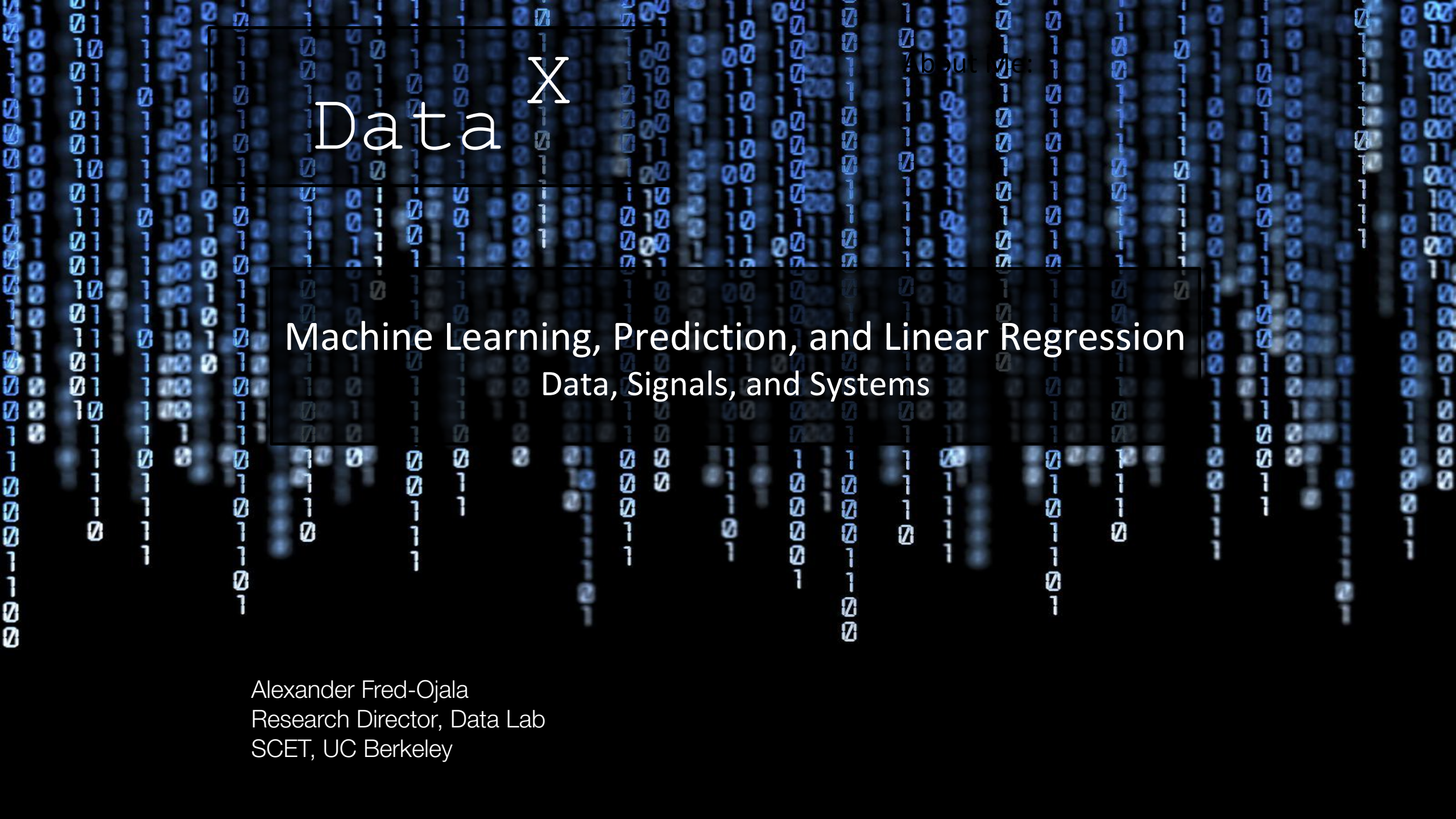


About me:

Machine Learning, Prediction, and Linear Regression

Data, Signals, and Systems

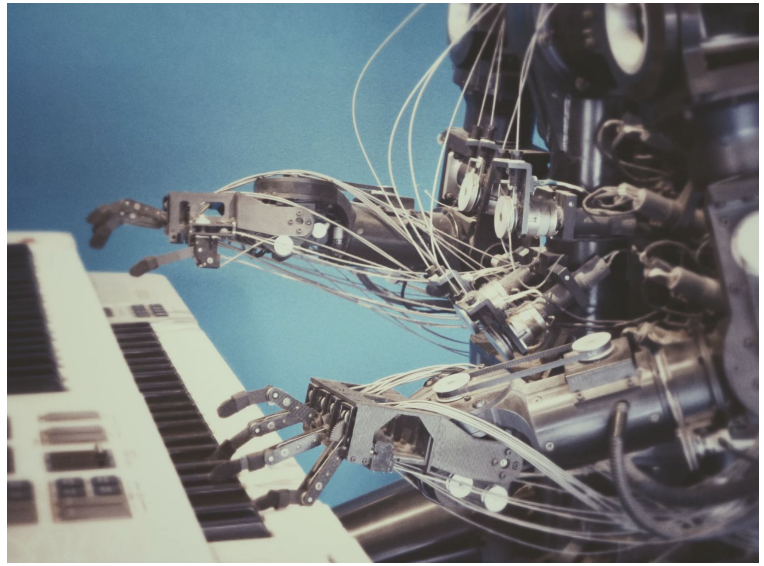
Alexander Fred-Ojala
Research Director, Data Lab
SCET, UC Berkeley

Machine Learning

Machine Learning: Most common Definitions

Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed.

- **Arthur Samuel, 1959**



*A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P** if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.*

- **Tom Mitchell, 1997**

Different types of Machine Learning

ML systems can be classified in accordance with how much *supervision* they receive. The four most common ML categories are:

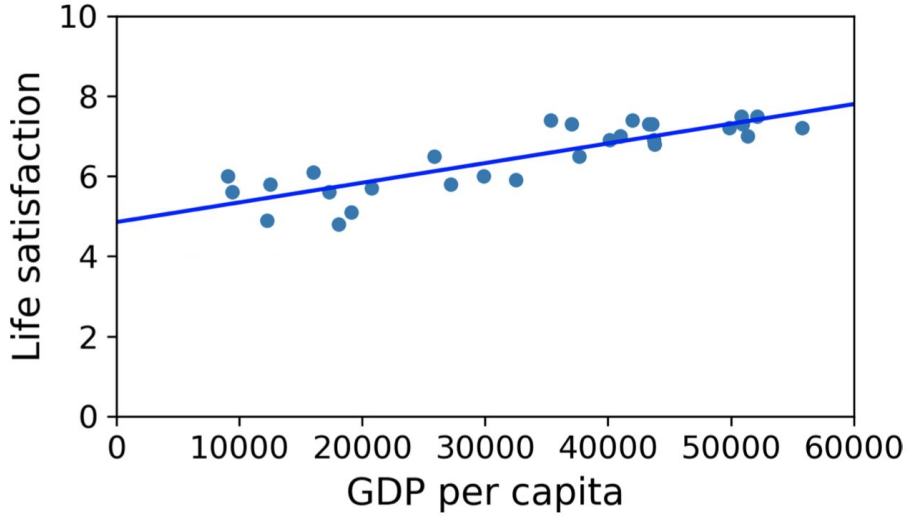
1. **Supervised Learning**
2. Semisupervised Learning
3. Unsupervised Learning
4. Reinforcement Learning

Supervised ML

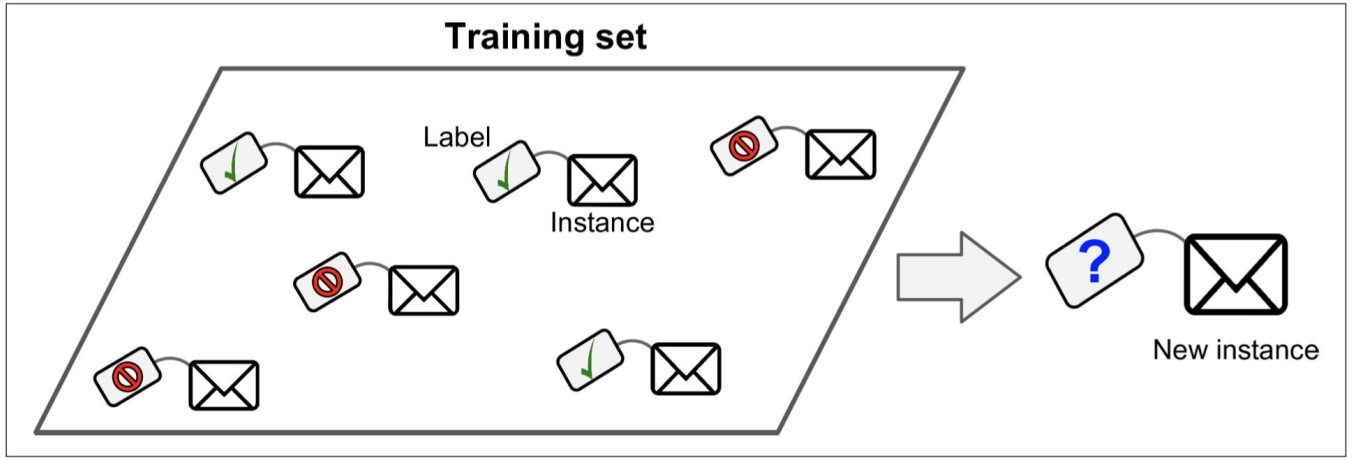
In supervised Machine Learning your training dataset *includes examples of the output / result / labels* you're trying to create a model to predict.

Two Supervised ML model categories:

Regression
(predict a continuous value)



Classification
(predict a category)

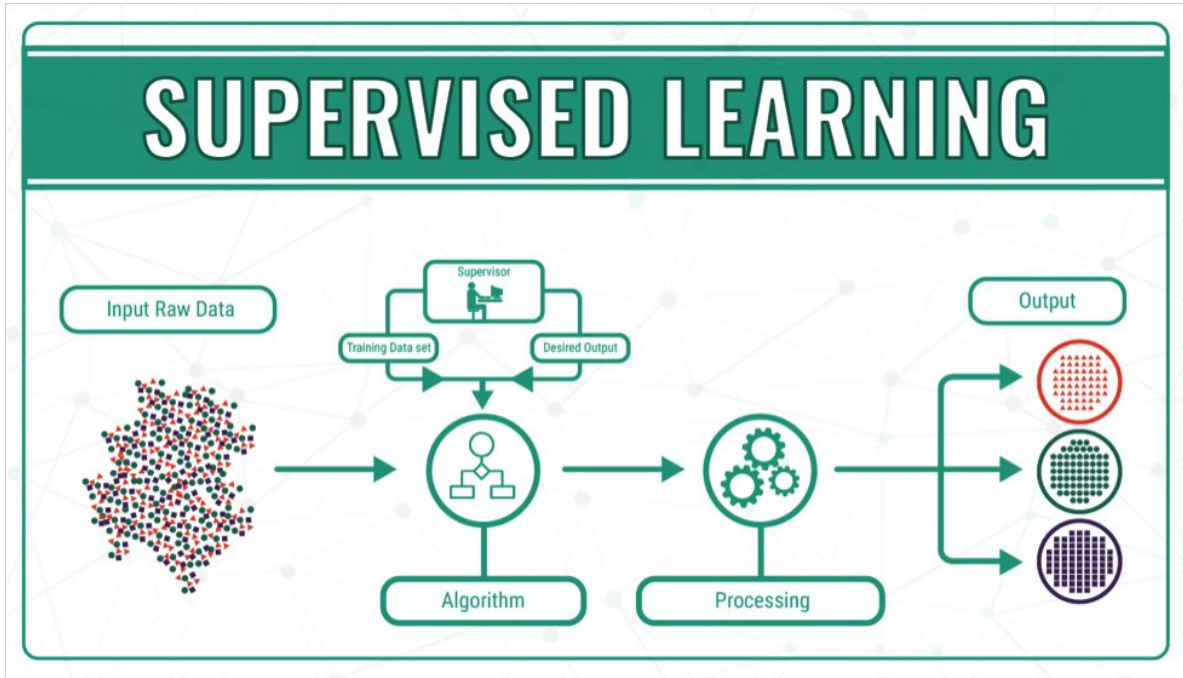


A labeled training set for supervised learning (e.g., spam classification)

Images: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, v2 2019, Aurélien Géron

Examples of Supervised Machine Learning Algorithms:

- **Linear Regression**
- **Logistic Regression**
- Decision Trees, Random Forests, Gradient Boosting
- Support Vector Machines (SVMs)
- k-Nearest Neighbors
- Naive Bayes
- Perceptrons
- Neural networks



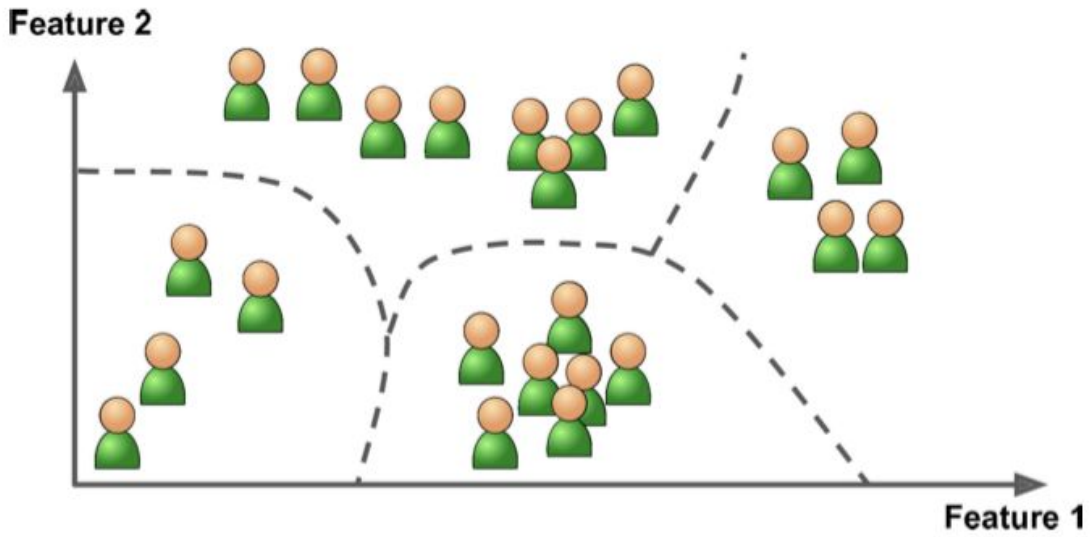
Unsupervised ML

In supervised Machine Learning your training dataset is unlabelled (ie you don't have any recorded output). These systems learn and uncovers patterns.

Examples:

Clustering

(segment the data into subgroups)



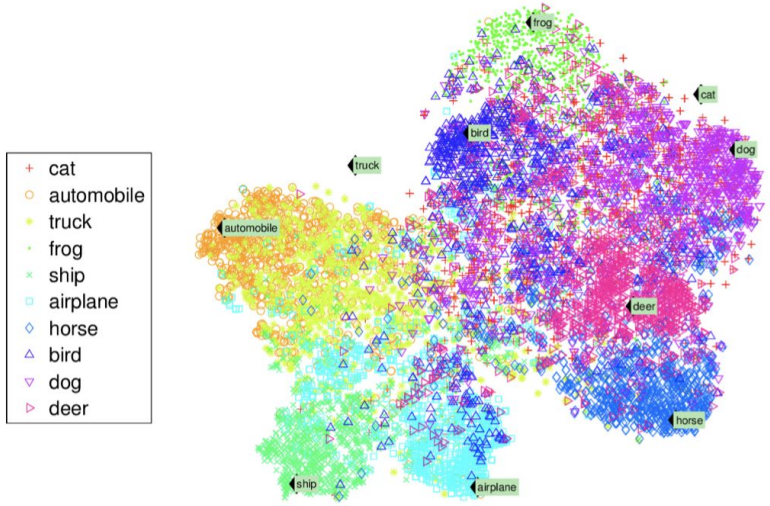
Anomaly Detection

(identify unusual patterns)

Images: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, v2 2019, Aurélien Géron

Examples of Unsupervised ML Algorithms:

- **Clustering**
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- **Association rule learning**
  - Apriori
  - Eclat



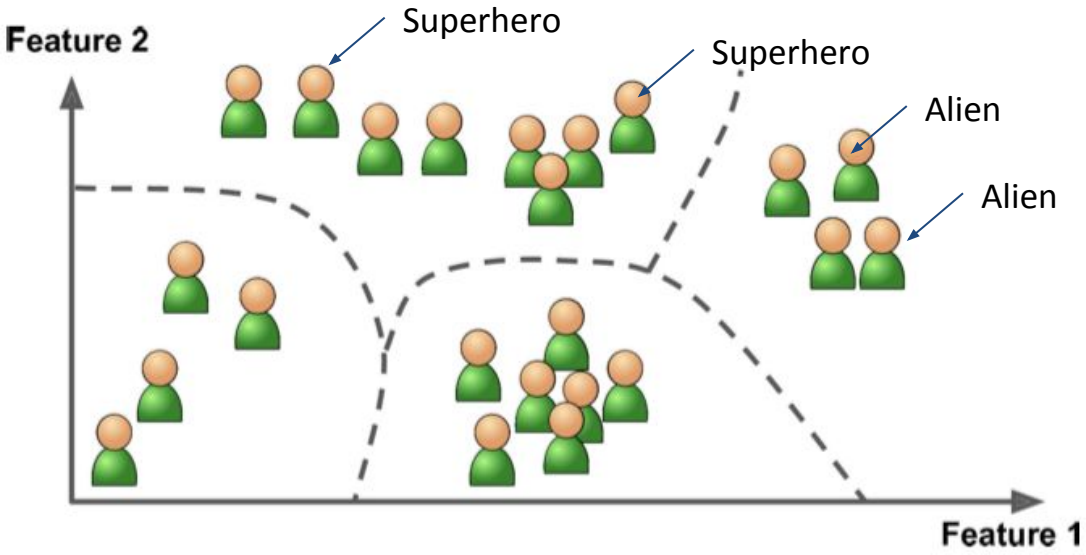
- **Visualization / dimensionality reduction**
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally-Linear Embedding (LLE)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)
- **Anomaly detection / novelty detection**
  - One-class SVM
  - Isolation Forest

List & Image from: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, v2 2019, Aurélien Géron

Semi-supervised Learning

Find characteristics of groups in unsupervised / unlabelled data. Label a couple of samples to form distributions of samples related to categories

Semi-supervised:



Cornell University

arXiv.org > cs > arXiv:1905.02249v1

Search... Help | Advanced

Computer Science > Machine Learning

MixMatch: A Holistic Approach to Semi-Supervised Learning

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, Colin Raffel

(Submitted on 6 May 2019)

Semi-supervised learning has proven to be a powerful paradigm for leveraging unlabeled data to mitigate the reliance on large labeled datasets. In this work, we unify the current dominant approaches for semi-supervised learning to produce a new algorithm, MixMatch, that works by guessing low-entropy labels for data-augmented unlabeled examples and mixing labeled and unlabeled data using MixUp. We show that MixMatch obtains state-of-the-art results by a large margin across many datasets and labeled data amounts. For example, on CIFAR-10 with 250 labels, we reduce error rate by a factor of 4 (from 38% to 11%) and by a factor of 2 on STL-10. We also demonstrate how MixMatch can help achieve a dramatically better accuracy-privacy trade-off for differential privacy. Finally, we perform an ablation study to tease apart which components of MixMatch are most important for its success.

youtu.be/t4kyRyKyOpo?t=777

Images: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, v2 2019, Aurélien Géron

Reinforcement Learning

Train a **learning system (called agent)** that can **observe the environment** (world that it interacts with). The agent selects and **performs actions, and get rewards** (or penalties) in return depending on if the action was good or not. The agent learn what is the **best strategy, called a policy, to get the most reward over time.** A policy defines what action the agent should choose when it is in a given situation.

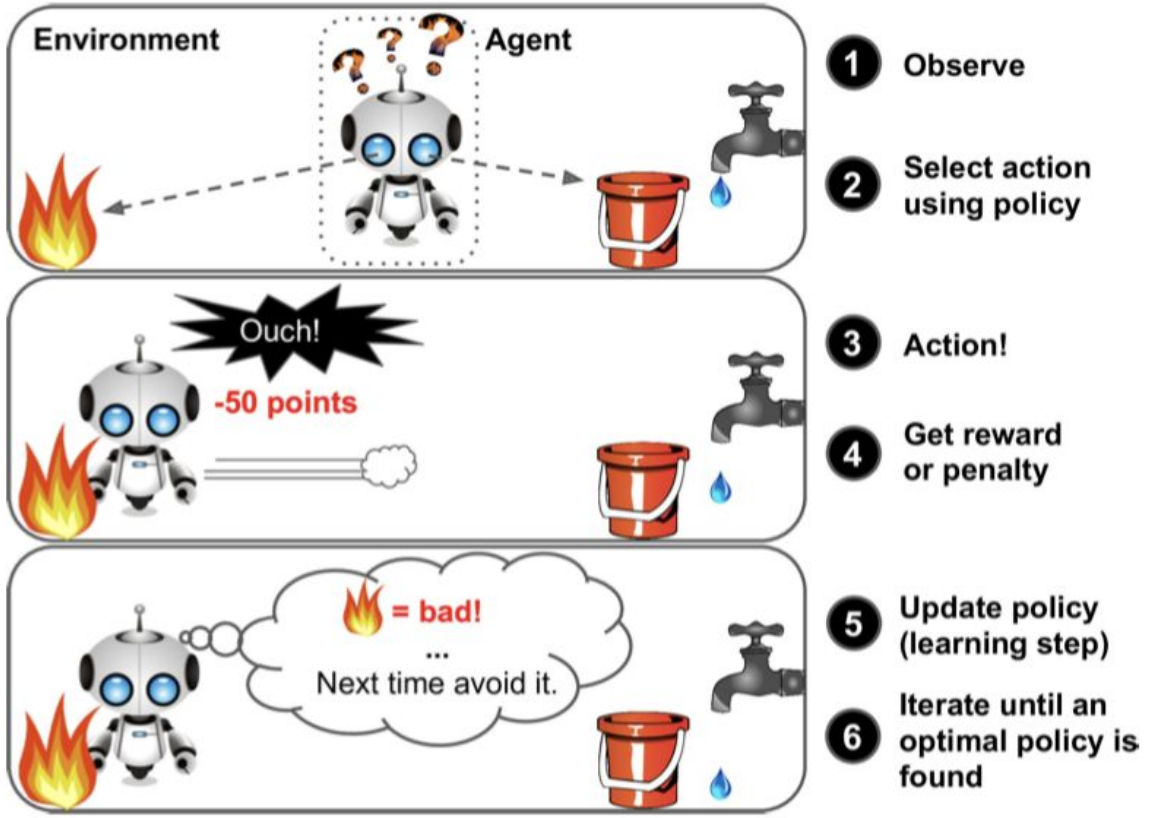


Image: Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, v2 2019, Aurélien Géron

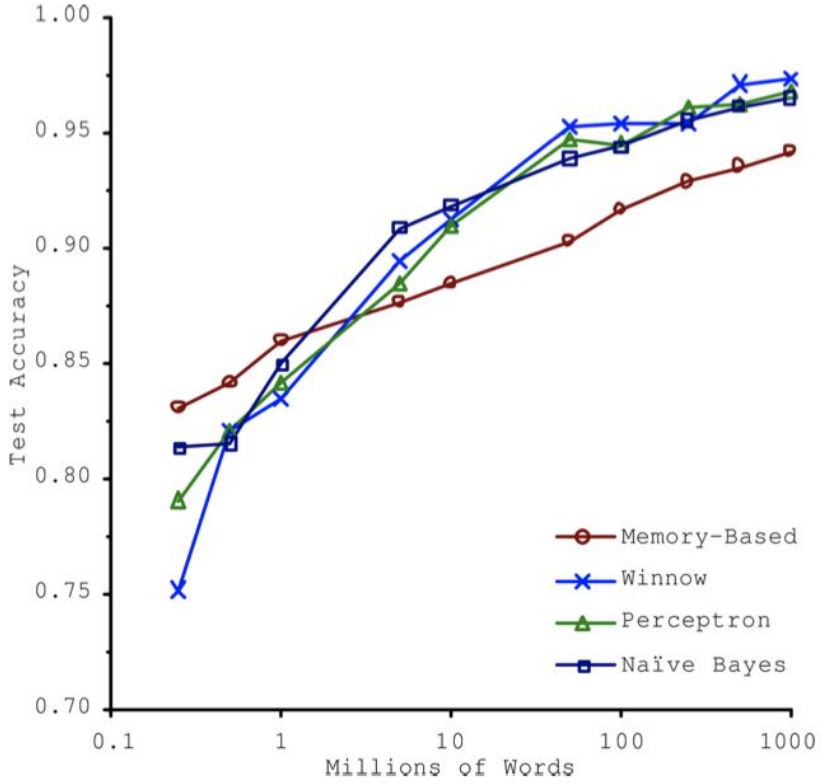
Other ML Model Characteristics

- **Batch Learning (Offline Learning)**
  - Train the system on all training data
  - Needs to retrain every time the model should be updated
- **Online Learning**
  - Train system incrementally
  - Learn about new data directly
  - Capable of performing out-of-core learning
- **Instance-based Learning**
  - Memorizes the structure of the data
  - Measures similarity of samples
  - Non-parametric models
  - E.g. K-nearest neighbors
- **Model-based learning**
  - Summarize the data with a fixed set of learnable parameters (independent of # training examples)
  - Training == finding the parameters
  - E.g. Linear Regression

Main Challenges of Machine Learning

1. Insufficient amount of data
2. Nonrepresentative Training Data
3. Poor-Quality Data
4. Irrelevant Features (Garbage in, Garbage out)
5. Overfitting the Training Data
6. Underfitting the Training Data

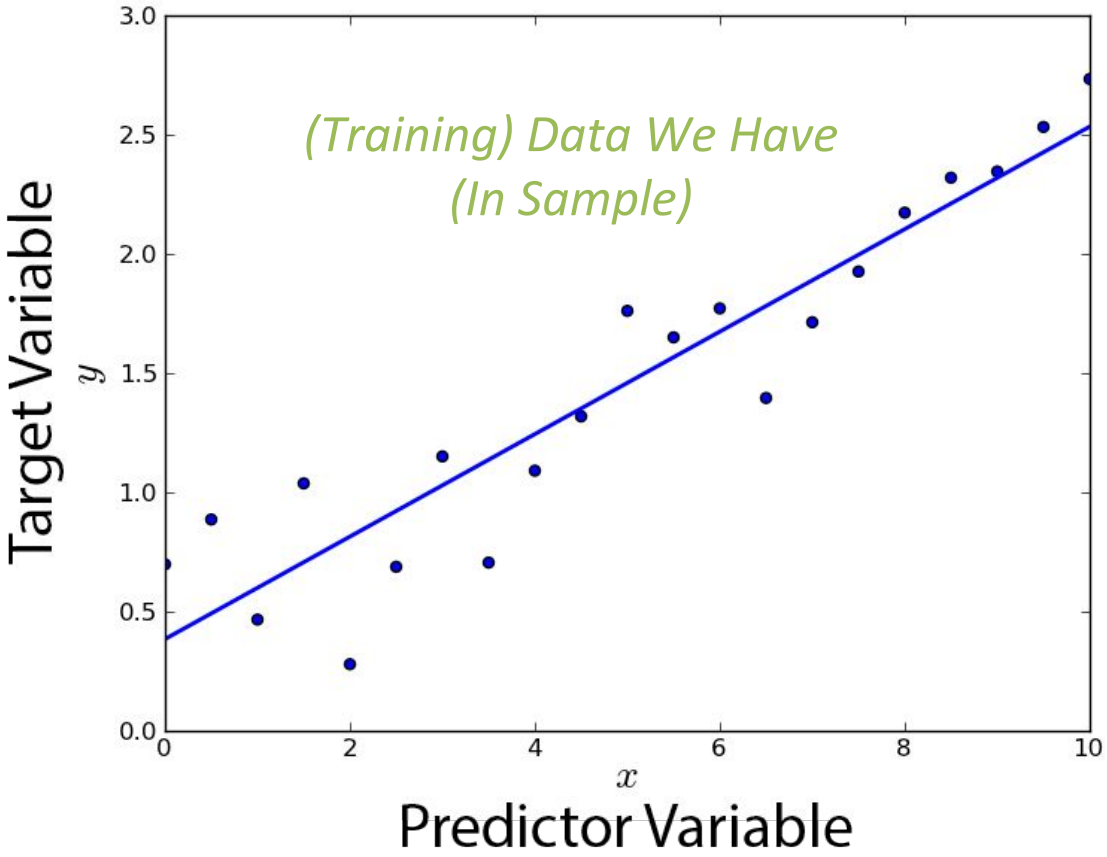
The Unreasonable Effectiveness of Data

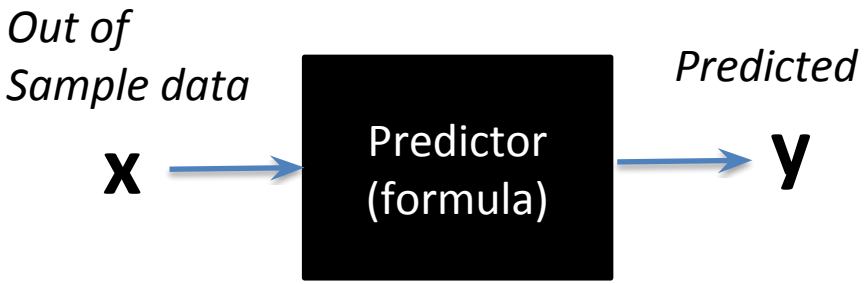


Michele Banko and Eric Brill, 2001, Microsoft Research

Introduction to Prediction

Prediction

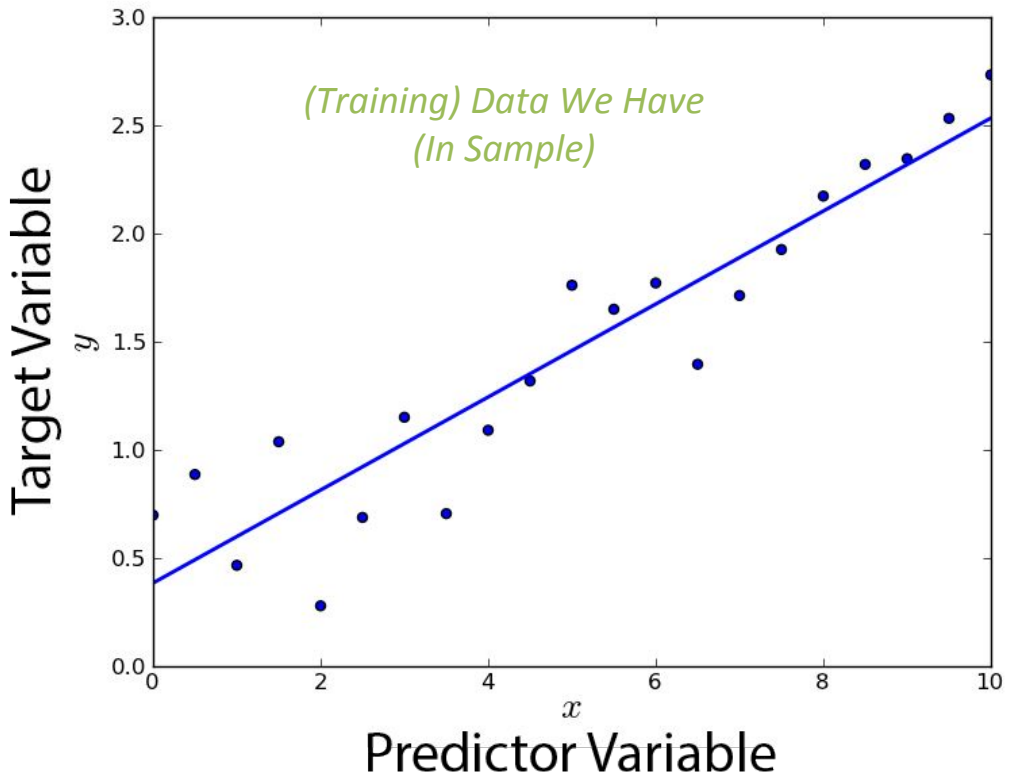




Our Goal: Being able to train a model that can make accurate predictions on *out-of-sample data*

Prediction

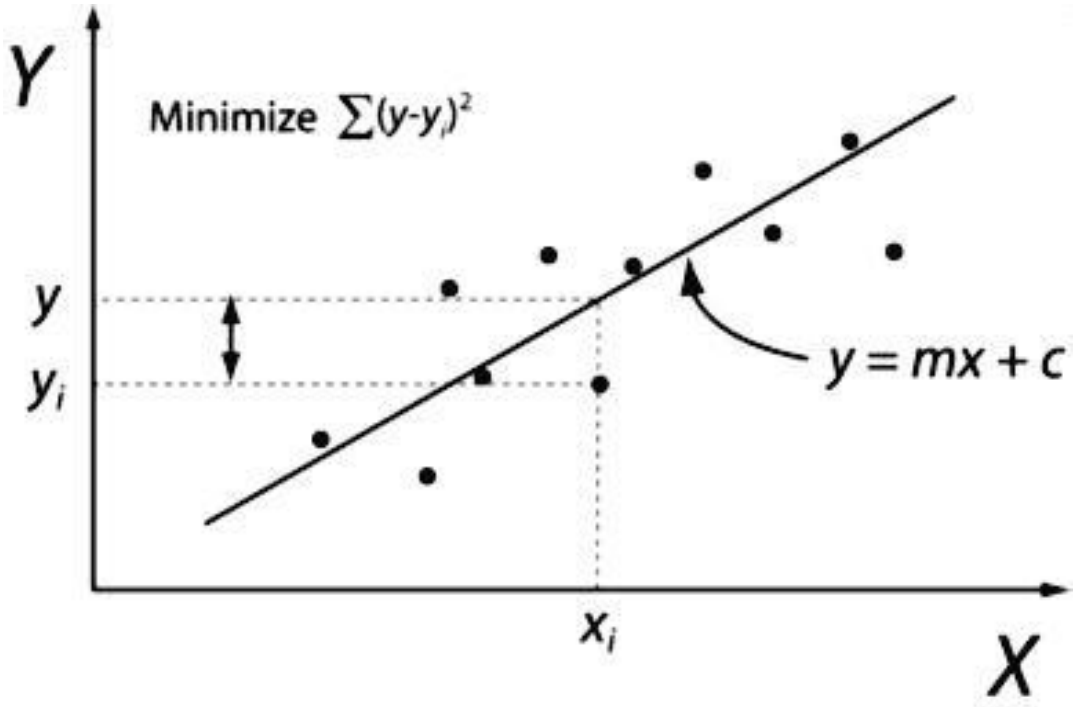
<u>Problem:</u> *How to construct a function F(x) that can predict the value for out-of-sample data?*



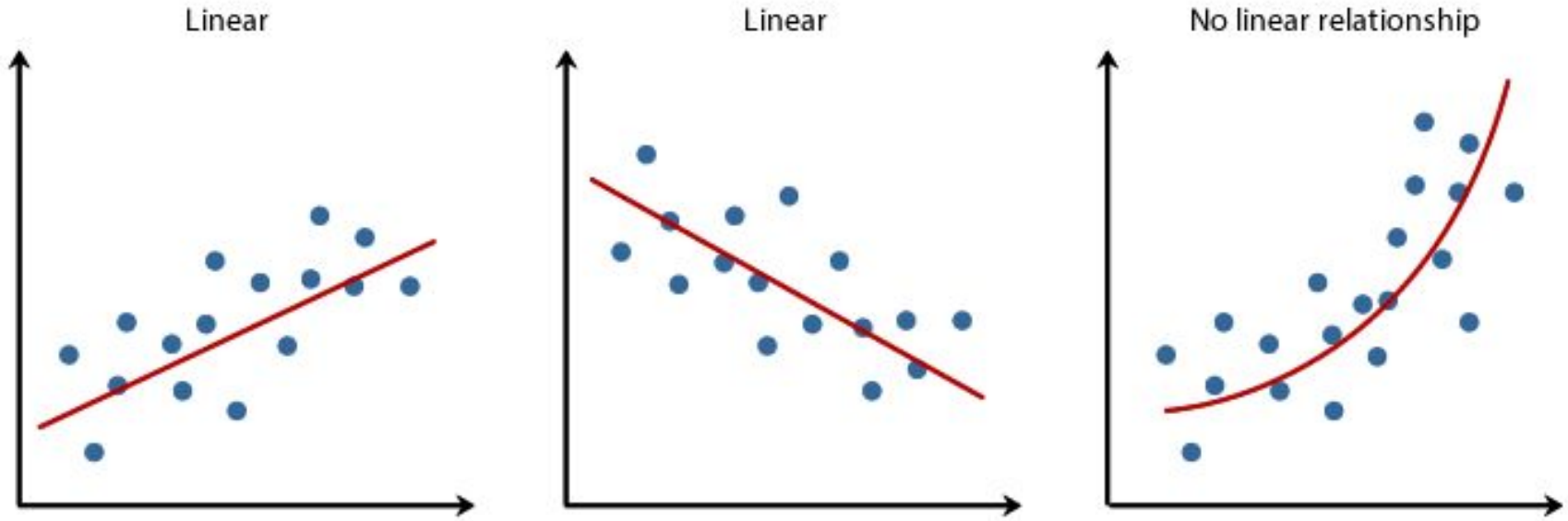
One way to make a prediction:

Choose a line that best fits the sample data

Then $\boldsymbol{y(x) = mx + c}$ is a predictor for a new out of sample $\boldsymbol{x}$



*Of Course, we can not assume that all data can be predicted by a **linear model***



- Model might be a poor fit (wrong model)
- Model might be too good of a fit or over-fit (only works well on the in sample data)

Image: Laerd Statistics, 2014

Best Linear Predictor

(if you just have 1 input X and 1 output Y)

This turns out to be the best linear predictor: $\hat{Y} = y(x) = mx + c$

$$L(Y \mid X) = \mathbb{E}(Y) + \frac{\operatorname{cov}(X, Y)}{\operatorname{var}(X)}[X - \mathbb{E}(X)]$$

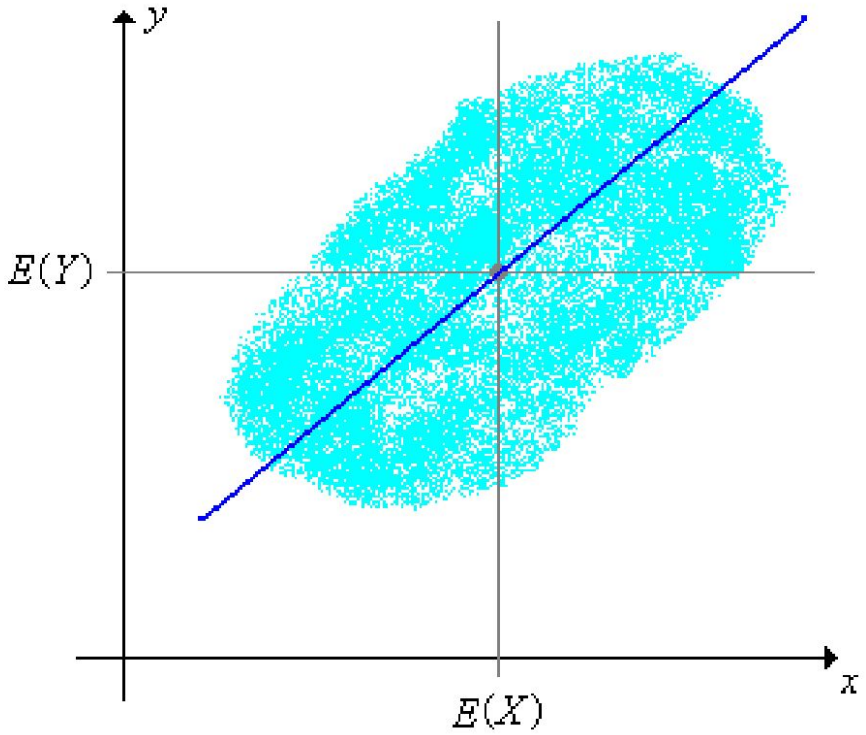
It's a line:

Runs though point: $(E[X],\ E[Y])$

slope: $m = \frac{COV(X,Y)}{VAR(X)}$

y-intercept $= E[Y] - mE[X]$

$= E[Y] - COV(X,Y) * \frac{E[X]}{VAR(X)}$



1. *What makes this the the best linear predictor?*
2. *How much error does this predictor have?*

Remember:

$COV(X,Y) = E[\ (X-\mu_x)\ (Y-\mu_y)\ ]$

$COV(X,Y) = E\ [X\ Y\ ] - E[X]\ E[Y]$

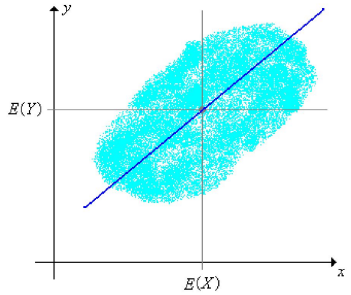
$COV(X,X) = VAR(X)$

1. What makes this the the best linear predictor?

(suppose X and Y are random variables)

This turns out to be the best linear predictor:

$$\hat{Y} = y(x) = mx + c$$



It's a line:

Runs though point: $(E[X],\ E[Y])$

slope: $m = \frac{COV(X,Y)}{VAR(X)}$

y-intercept $= E[Y] - mE[X]$

$= E[Y] - COV(X,Y) * \frac{E[X]}{VAR(X)}$

We want to minimize the mean squared prediction error

$$E[(\hat{Y} - Y_{actual})^2] = E[(mX + c - Y)^2]$$
$$= E[(m^2X^2 + 2cmX - 2mXY + c^2 - 2cY + Y^2)]$$
$$= m^2E[X^2] + 2cmE[X] - 2mE[XY] + c^2 - 2cE[Y] + E[Y^2]$$

Remember:

COV(X,Y) = E[$(X-\mu_x)$ $(Y-\mu_y)$]

COV(X,Y) = E [X Y] – E[X] E[Y]

COV(X,X) = VAR(X)

*Find solution by minimizing the error function w.r.t. the two variables, ie find the partial derivative of **m** and **c** and solve for 0.*

Simple Example: Calculate best linear predictor

Data Set in a Table, two variables

X	**Y**
2	10
4	5
3	9
5	4
6	3

Find the best equation of a line that fits the data

$$\hat{Y} = y(x) = mx + c$$

Runs though point: $(E[X],\ E[Y])$

slope: $m = \frac{COV(X,Y)}{VAR(X)}$

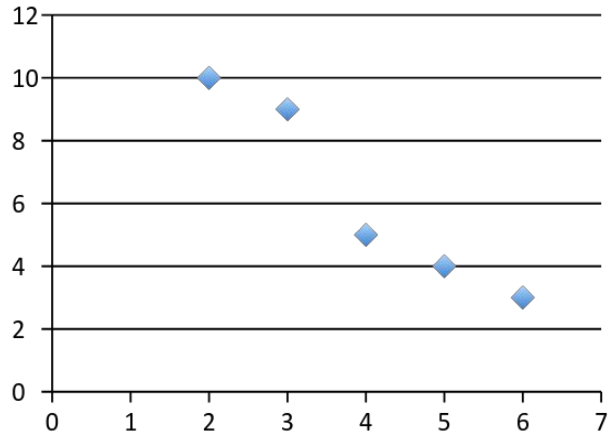
y-intercept = $E[Y] - mE[X]$

$= E[Y] - COV(X,Y) * \frac{E[X]}{VAR(X)}$

Example: Data Set in a Table 2 variables

X	Y	X*Y	X^2	y(x)
2	10	20	4	6.96
4	5	20	16	3.92
3	9	27	9	5.44
5	4	20	25	2.4
6	3	18	36	0.88

E[X]	E[Y]	E[XY]	E[X^2]
4	6.2	21	18



Solution:

$E[X] = 4, \qquad E[Y] = 6.2$

$Cov(X,Y) = E[XY] - E[X]E[Y]$
$= 21 - 4 * 6.2 = -3.8$

Var(X) = 18-16 = 2

$\text{y-int} = E[Y] - \frac{Cov(X,Y)}{Var(X)} * E[X]$
$= 6.2 - \left(\frac{-3.8}{2} * 4\right) = 13.8$

$m = \frac{Cov(X,Y)}{Var(X)} = \frac{-3.8}{2} = -1.9$

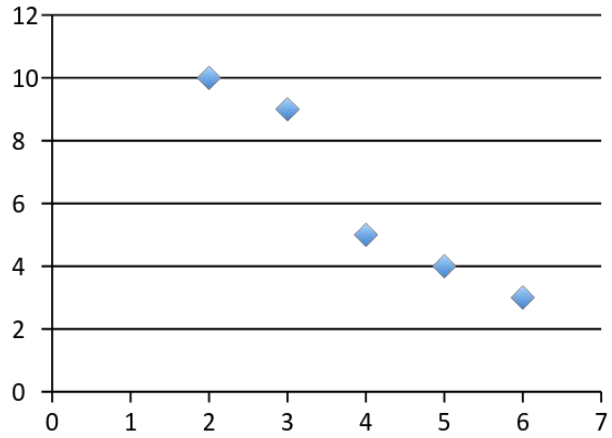
$y(x) = -1.9x + 13.8$

$E[y(x) - y_{actual})^2] = ?$

Example: Data Set in a Table
2 variables

X	Y	X*Y	X^2	y(x)
2	10	20	4	6.96
4	5	20	16	3.92
3	9	27	9	5.44
5	4	20	25	2.4
6	3	18	36	0.88

E[X]	E[Y]	E[XY]	E[X^2]
4	6.2	21	18

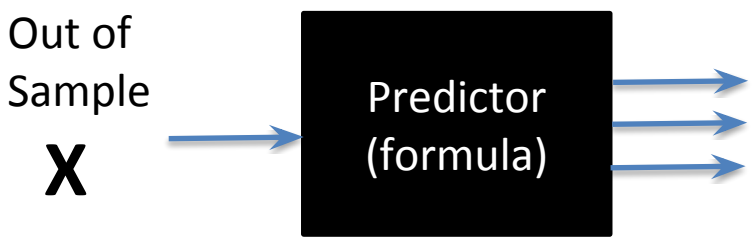


Code Sample

```
import numpy as np

x = np.array([2, 4, 3, 5, 6])
y = np.array([10, 5, 9, 4, 3])

E_x = np.mean(x)
E_y = np.mean(y)

cov_xy = np.mean(x*y)-E_x*E_y

y_0 = E_y - cov_xy/np.var(x)*E_x
m = cov_xy/np.var(x)

y_pred=m*x+y_0

print "E[(y_pred-y_actual)^2] =", np.mean(np.square(y_pred-y))
```

```
E[(y_pred-y_actual)^2] = 0.54
```

Prediction: Multiple Inputs / Multiple Outputs

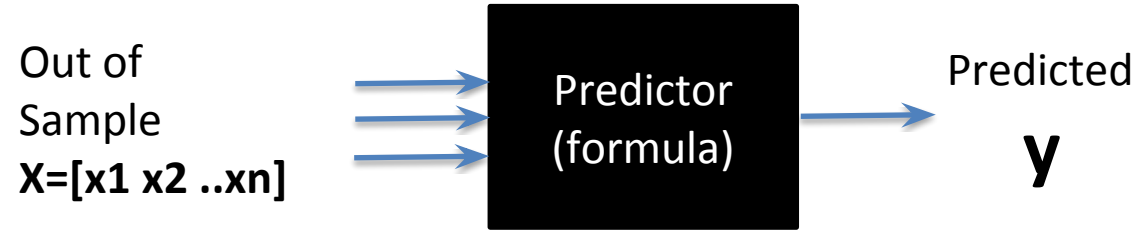


No problem,
we use multiple predictors.

y1 = g1(x)

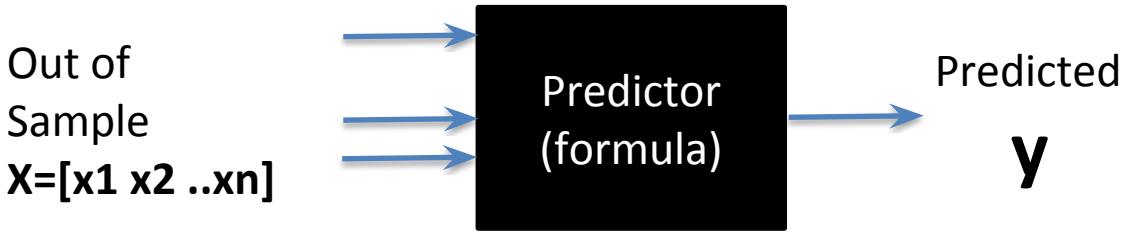
y2 = g2(x)

y3 = g3(x)



?

Prediction: Multiple Inputs



In this case, for linear prediction, we use a matrix format:

$$\begin{pmatrix} X \\ (Nxd) \end{pmatrix} \cdot \begin{pmatrix} W \\ (dx1) \end{pmatrix} = \begin{pmatrix} Y \\ (Nx1) \end{pmatrix}$$

$$x_{1,1}w_1 + x_{1,2}w_2 + x_{1,3}w_3 = y_1$$
$$x_{2,1}w_1 + x_{2,2}w_2 + x_{2,3}w_3 = y_2$$
$$\dots$$
$$\dots$$

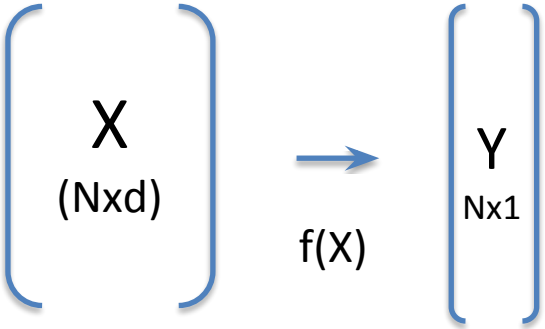
X is the in-sample data.
We need to figure out the model weights (coefficients) **w**.

With **w**, we can estimate any **y** for new **x**, i.e. out of sample data

A Supervised ML Framework

In Sample Data
which we have
- d features
- N samples

Use for training



Results in Y
(which we know for N samples)

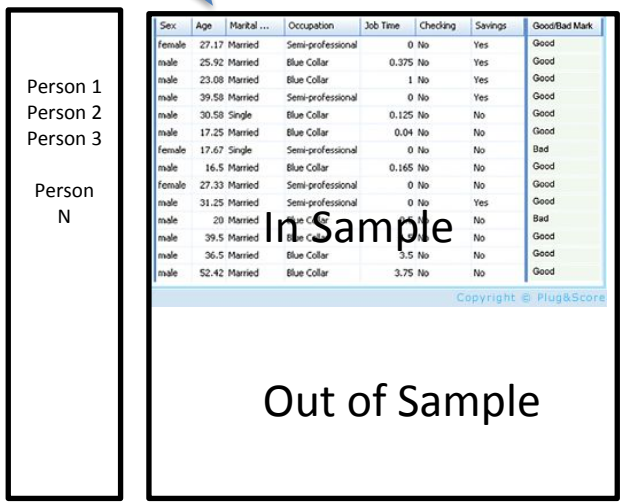
Sometimes measures in Y includes error or noise
Y = f(X) + e

We don't know:

P(X), the true distribution of X (we can only approximate)

We want to find function F
F: X->Y

Features: (d columns)
Age, income, zip code, ..



N rows:
- each row has customer information according to **d** features
- **Y has a value of interest**
  - *Credit score:* a number
  - *Classification:* "good customer" vs "poor customer"

An ML Framework

We use the in-sample data to train our model:
(x1,y1), (x2,y2)…xN,yN)

X → Algorithm: g(x) → **Y**

g: x->y,
g is a function, ie the predictor - classifier

H = Hypothesis Set
All Possible Algorithm Candidate **g**

Error measure: **Error(g, x)**

for example: E[y-g(x)]

Actual Estimated

- **g(x) is our estimate of f(x), that is the true function.**
  Note: The true y = f(x) + e
- **Many ways to measure error (squared error, absolute value, etc)**

Example: Predict Credit Score with Regression

In-sample data to train our model:
(x1,y1), (x2,y2)…xN,yN)

Data:

ID	Name	**x(i,1) Age**	**x(i,2) years w employer**	**x(i,3) Income**	**y(i,1) Credit Score**
1	John	25	3	50	660
2	Alice	23	2	60	580
3	Bill	28	1	80	425
4	Rahul	25	.5	59	320

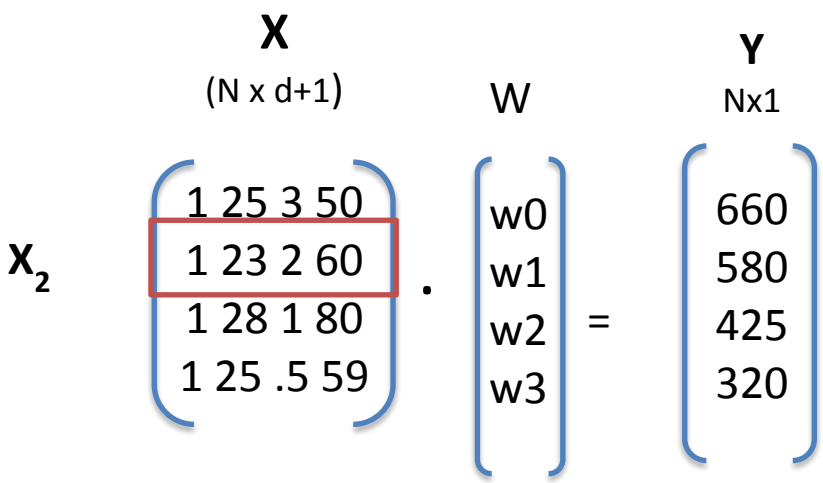
$$\underset{(N \times d+1)}{\mathbf{X}} \cdot \mathbf{w} = \underset{Nx1}{\mathbf{Y}}$$

$$\begin{pmatrix} 1 & 25 & 3 & 50 \\ 1 & 23 & 2 & 60 \\ 1 & 28 & 1 & 80 \\ 1 & 25 & .5 & 59 \end{pmatrix} \cdot \begin{pmatrix} w0 \\ w1 \\ w2 \\ w3 \end{pmatrix} = \begin{pmatrix} 660 \\ 580 \\ 425 \\ 320 \end{pmatrix}$$

We are now going to *choose a W that gives us the best predictor g(x) to estimate Y for out-of-sample data.*

This time, **Y** is the actual value we want to estimate
Notice: we added an extra column of 1s?

Example: Prediction with Regression



Y is the actual value we want to estimate.

Train with this to calculate the model parameters **W**

Then predict (or classify) with **W**

Out of sample
(a row of numbers)

X_i

→ **Predictor:** g(x) = Xw →

Estimated y

The Math: Linear Regression

Find the best model parameters by **minimizing the expected value of the squared error** of our predictions.

Find optimal Predictor:
g(x) = Xw

This is also called solving the **ordinary least squares** problem for linear regression.

$$SSE = E_{in}(w) = E[(Xw - Y)^2]$$

It is the same as finding the Normal equation:
mathworld.wolfram.com/NormalEquation.html

$$E_{in}(w) = \frac{1}{N} \sum_{i=1}^{N} \left(w^T x_i - y_i \right)^2$$

$$E_{in}(w) = \frac{1}{N}\|Xw - y\|^2$$

$$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y) = 0.$$

$$X^T X W = X^T Y$$

$$\boxed{W = (X^T X)^{-1} X^T Y}$$

$Y_{estimated} = X_{out\ of\ sample}\ W$

X^T = [dxN]

X = [Nxd]

$(X^T X)$ = [dxd] $(X^T X)^{-1}$ = [dxd]

W = $(X^T X)^{-1} X^T Y$ =

= [dxd] · [dxN] · Y= [Nxm]

= [dxN] x [Nxm] = dxm

Y estimated = X W =

Xout = Nout x d · W = dxm = N_out x m

N = # of X data rows
d= # of features
m= # of outputs in Y (usually 1)

Prediction with Regression (continued)

$$X = \underbrace{\begin{pmatrix} 1 & 25 & 3 & 50 \\ 1 & 23 & 2 & 60 \\ 1 & 28 & 1 & 80 \\ 1 & 25 & .5 & 59 \end{pmatrix}}_{X\ (N \times d+1)} \cdot \underbrace{\begin{pmatrix} w0 \\ w1 \\ w2 \\ w3 \end{pmatrix}}_{W} = \underbrace{\begin{pmatrix} 660 \\ 580 \\ 425 \\ 320 \end{pmatrix}}_{Y\ Nx1}$$

This time, **Y** is the actual value we want to estimate

Now we use the in-sample data to calculate the value of **W** (analytically).

$$W = (X^T X)^{-1} X^T Y$$

Code Sample

```
import numpy as np

x = np.array([
        [1,25,3,50],
        [1,23,2,60],
        [1,28,1,80],
        [1,25,0.5,59]
    ])
y = np.array([660,580,425,320])

print "W = ", np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y)
```

```
W =  [ 426.17283951  -16.04938272  149.44444444    3.7345679 ]
```

$$x_{0,}w_0 + x_{1,}w_1 + x_2w_2 + x_3w_3 = y_i$$

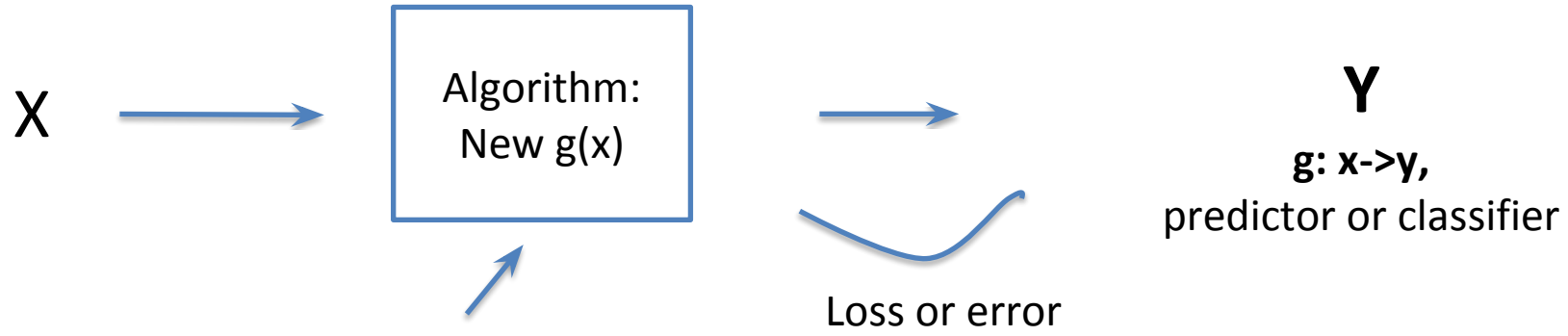
Data:		x(i,1)	x(i,2)	x(i,3)	y(i,1)
ID	Name	Age	years w employer	Income	Credit Score
1	John	25	3	50	660
2	Alice	23	2	60	580
3	Bill	28	1	80	425
4	Rahul	25	.5	59	320

How to solve this in an optimized way?

See fast.ai's comparison of different matrix factorization techniques + scikit-learn's implementation (primarily for finding the matrix inverse of the Normal Equation):

https://nbviewer.jupyter.org/github/fastai/numerical-linear-algebra/blob/master/nbs/6.%20How%20to%20Implement%20Linear%20Regression.ipynb

In the **ML framework**, there is no limit to the predictors or classifiers that can be used.



g can be chosen from,

Linear estimators:

- Any weighted sum
- The best fit line or plane

Non-linear functions:

- Neural Networks
- Polynomial Regression
- Decision Trees...

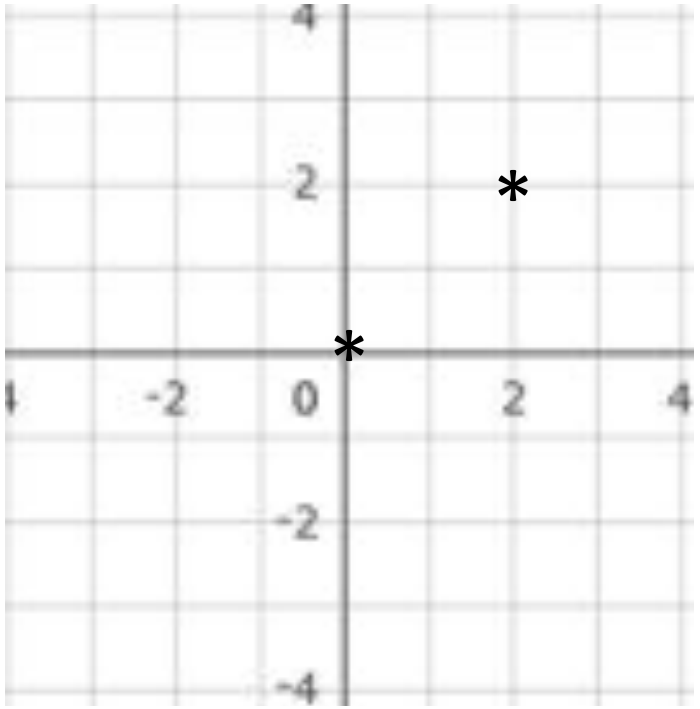
- We try different functions g until *g(x) is close to f(x)*
- For any out of sample **x**, we can predict **Y** or classify it

End of Section

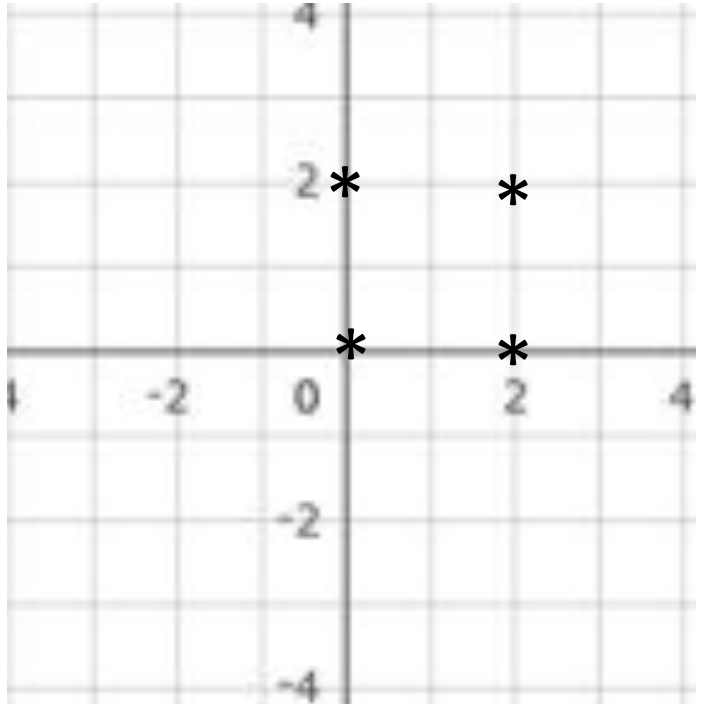
Test Your Understanding

Covariance

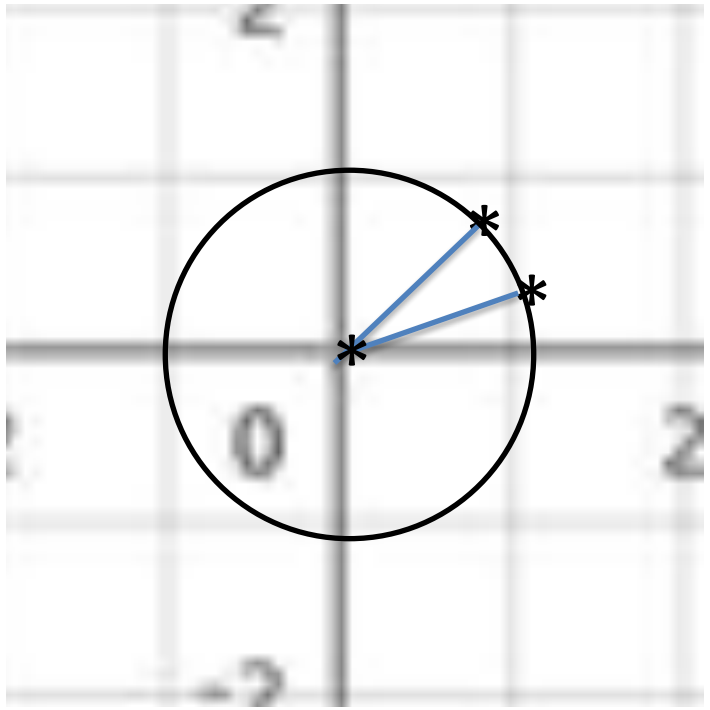
- What is the COV (X,Y) for these points:



(0,0) and (2,2)



(0,0),(2,2), (2,0), (0,2)



Points on a unit circle that are

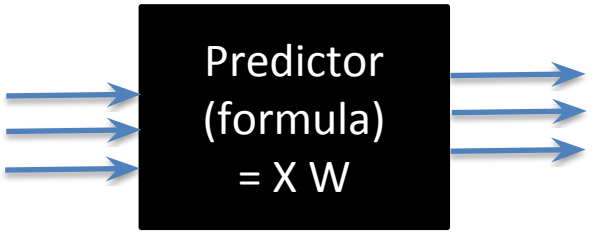
- 30 degrees and 45 degrees, and the origin

Multiple inputs and outputs

$$\text{If } Y_{predicted} = f(X,W) = X\,W$$

And W is a 4 x 3 matrix, then how many input features (d) are in X and how many outputs (m) are in Y?

X=[x1 x2 ..xd]



Predicted

y= =[y1 y2 ..ym]

About Me:

Part 2:

Gradient Descent, Classification & Logistic Regression

Alexander Fred-Ojala

Outline

1. Linear Regression recap
2. Gradient Descent
3. Feature scaling
4. Intro to Classification
5. Logistic Regression

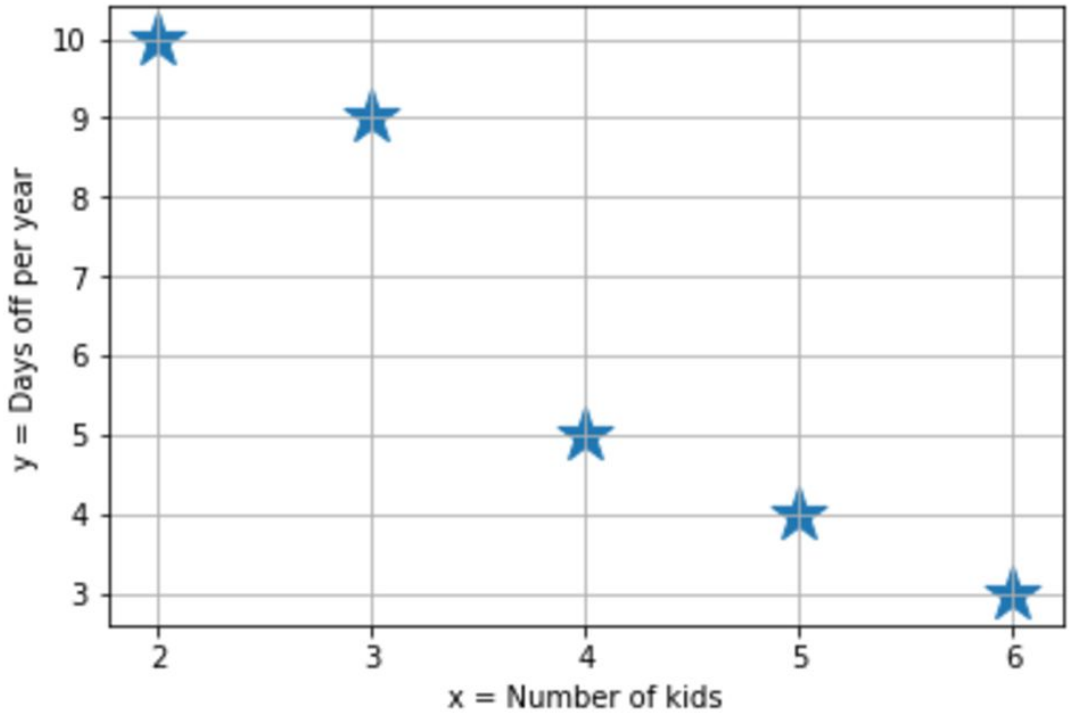
Recap: Linear Regression

Recap: Prediction

Given some data:

[(x_1,y_1) , (x_2 , y_2) … (x_m , y_m)]

x	y
2	10
4	5
3	9
5	4
6	3



Objective: Be able to predict y given new input x

Recap: Simple Linear Regression

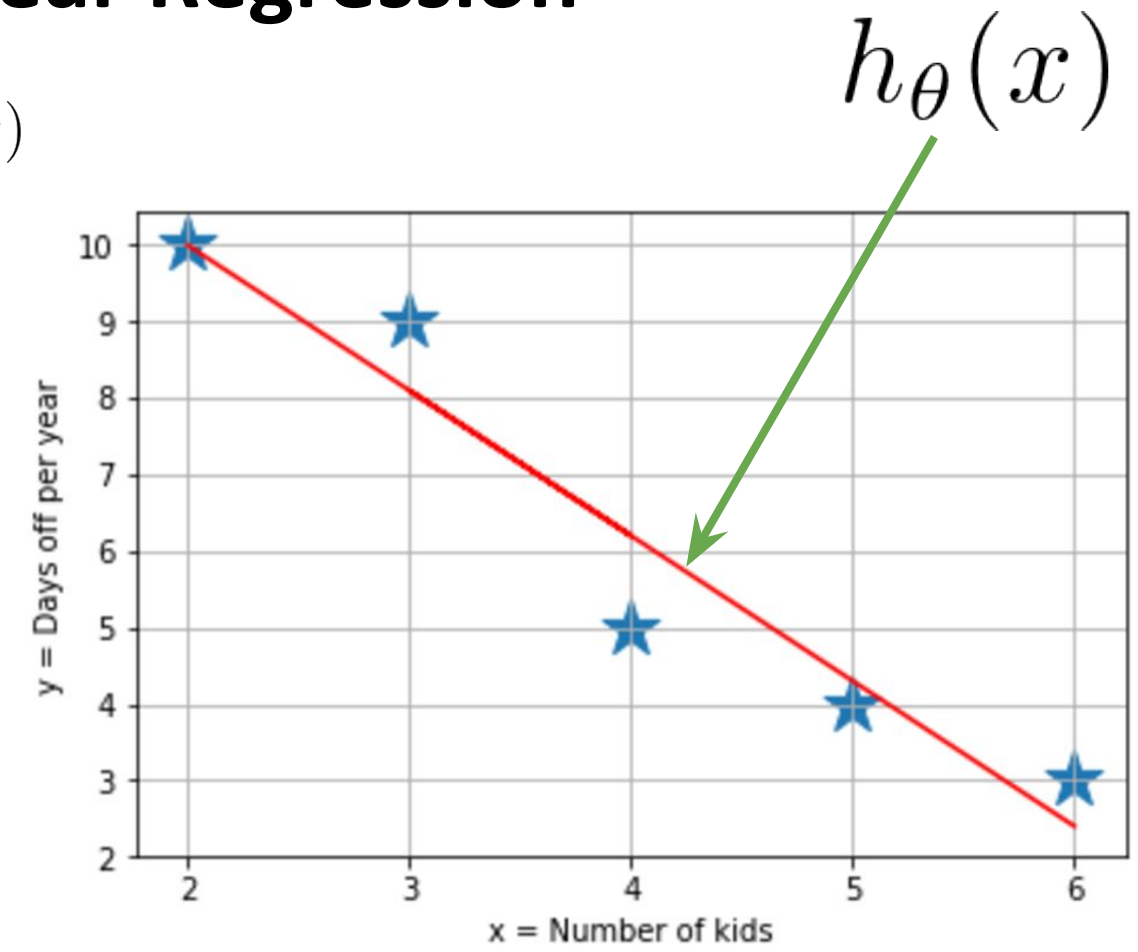
Simple Linear Regression: Hypothesis function $h_\theta(x)$

$$\hat{y} = f(x, \theta) = h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

$$x = \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$ **x is given input**

Objective: fit the best linear function to the training data, i.e. find optimal parameters θ

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



Recap: Multiple Linear Regression

Multiple Linear Regression: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta^T X$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ . \\ \theta_n \end{bmatrix}$$ is the parameter vector and

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & . & . & . & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & . & . & . & x_n^{(2)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_0^{(m)} & x_1^{(m)} & . & . & . & x_n^{(m)} \end{bmatrix}$$ is the feature vector and

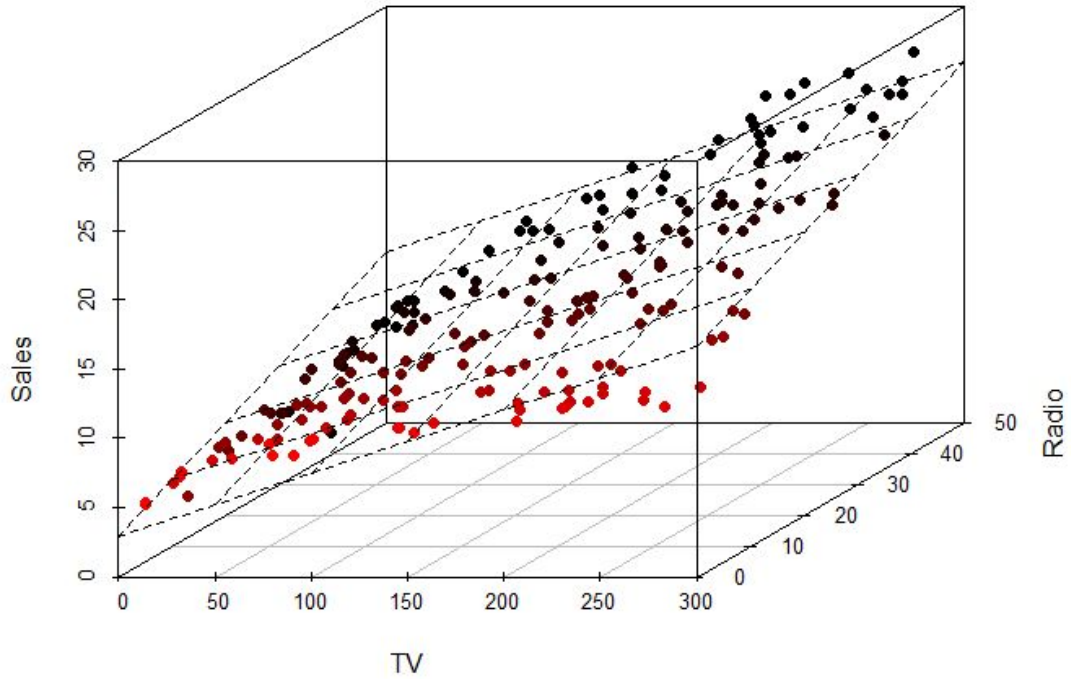
$$h_\theta(X) = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ . \\ . \\ h_\theta(x^{(m)}) \end{bmatrix}$$ is the hypotheses vector

Example of multiple linear regression (2 features)

x_1 = TV advertising

x_2 = Radio advertising

y = Sales

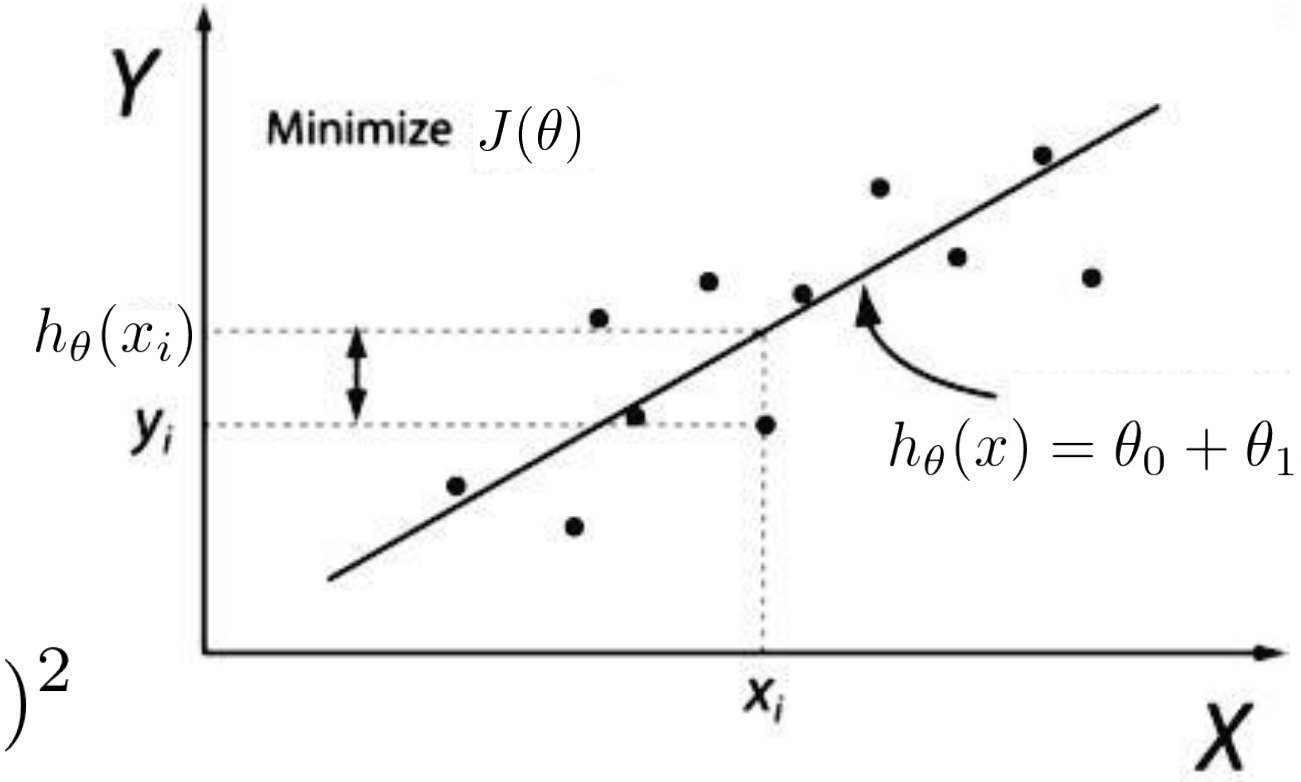


Source: 3.bp.blogspot.com/

Recap: Cost function, Mean Squared Error (MSE)

Simple Linear Regression

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1$$



Cost function:

Measures how good our predictions are (MSE)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Recap: Minimize cost function

Optimal model parameters minimizes J(θ)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

$$\min_\theta J(\theta) \implies \nabla_\theta J(\theta) = 0$$

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0$$

Minimize by taking the derivative w.r.t. θ = 0

Normal equation for Linear Regression

Closed form, analytical solution.

$$\theta = (X^T X)^{-1} X^T y$$

Pros:

- Finds optimum in one calculation
- Really quick for small data sets

Cons:

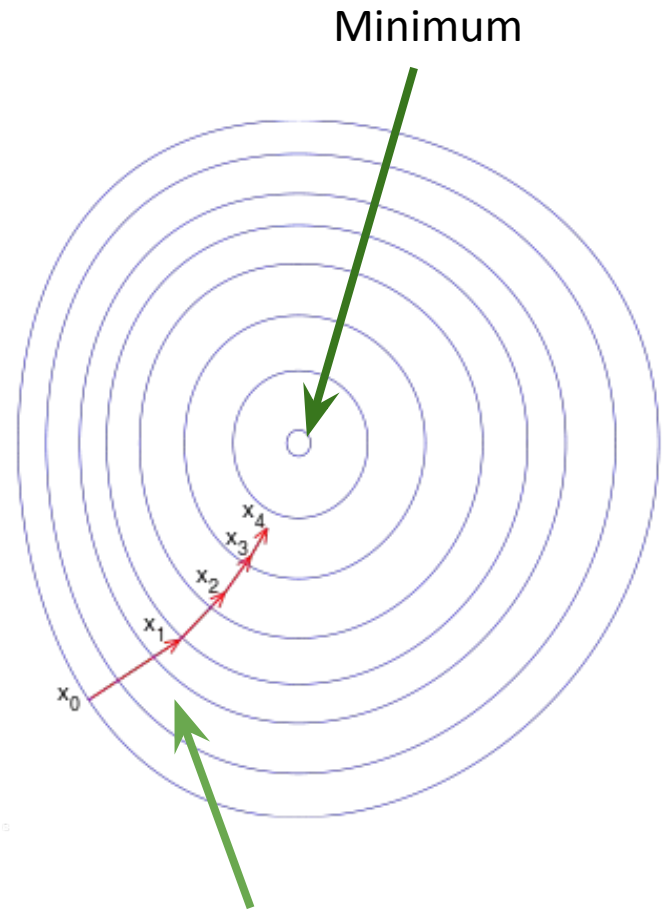
- $O(n^2 m)$ complexity, to calculate $(X^T X)$ and. O(n^3) to calculate $(X^T X)^{-1}$
- $(X^T X)^{-1}$ might not be invertible, ie singular (can be solved by using the pseduoinverse)

Gradient Descent

Introducing Gradient Descent

Gradient descent is a an iterative optimization algorithm for finding the minimum of a function.

To reach minima one takes steps proportional to the negative gradient (or approximate gradient) of the function at the current point.



Step sizes & neg. gradient directions

Introducing Gradient Descent

Alternative way of minimizing the cost function:

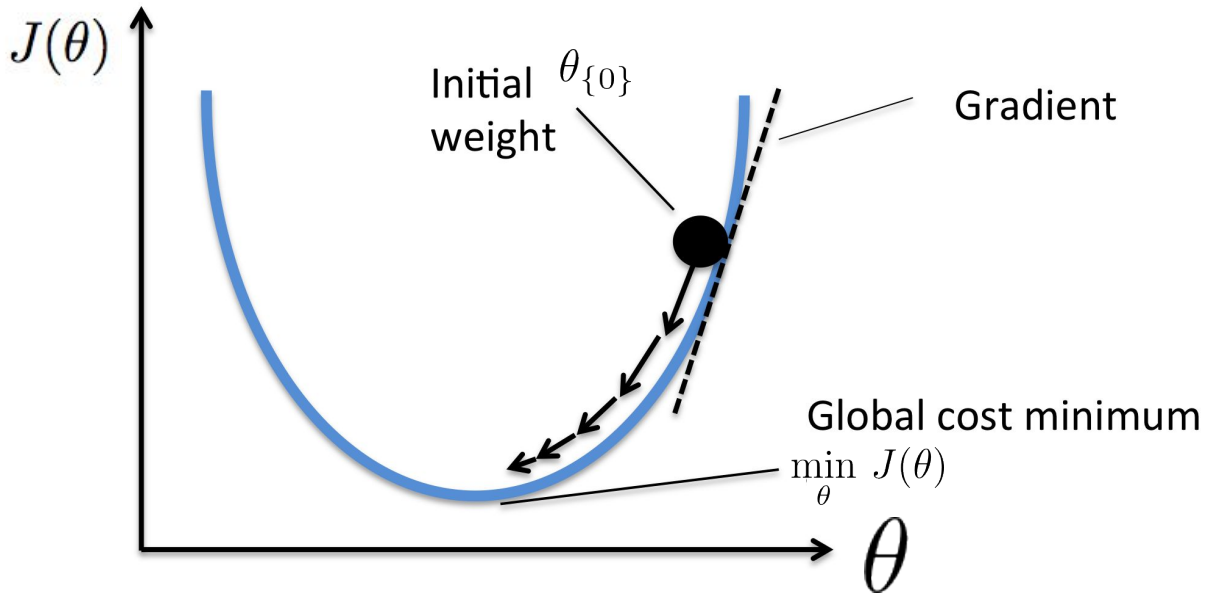
$$J(\theta) = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

- ***Will always converge because J(θ) is convex***

- Start with / initialize θ_0, θ_1 . E.g. $(\theta_0, \theta_1) = (0, 0)$
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$,

Illustration of Gradient Descent

for one parameter θ



Source: https://sebastianraschka.com

Gradient Descent Algorithm: Linear Regression

1. **Calculate the partial derivative** $\frac{\partial}{\partial \theta_j} J(\theta)$ for all j
2. Form the **update rule** for every parameter:

$$\theta_{j,iter+1} := \theta_{j,iter} - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_{j,iter} - \alpha/m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

3. **Choose a step size/ *learning rate*** α *(*often between 10^-6 and 10^2 -- not too big, then divergence).
4. **Update all the parameters** $\theta_0 \ldots \theta_n$ **by feeding in all training samples in X** ("batch" Gradient descent)
5. Stop when the error has converged.

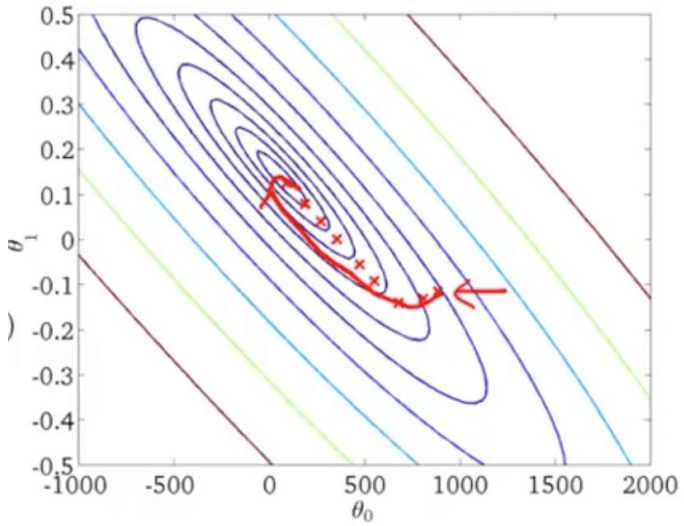
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

}



Source:Ritchie Ng

Gradient Descent Tips

Feature Scaling:

Gradient Descent will be quicker and more stable if the features are scaled.

Standardization

For all features:

- Subtract mean
- Divide by st.dev.

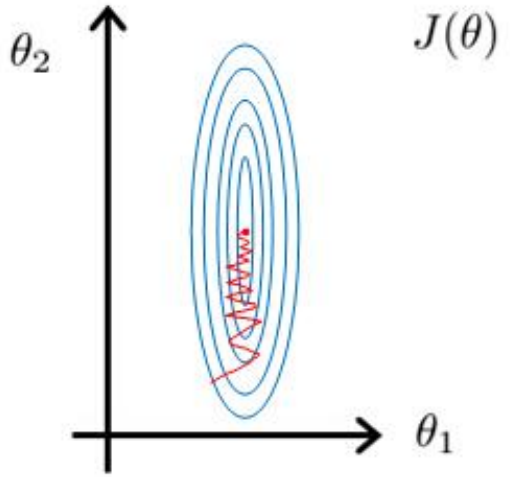
$$x_i \leftarrow \frac{x_i - \mu(x_i)}{\sigma(x_i)}$$

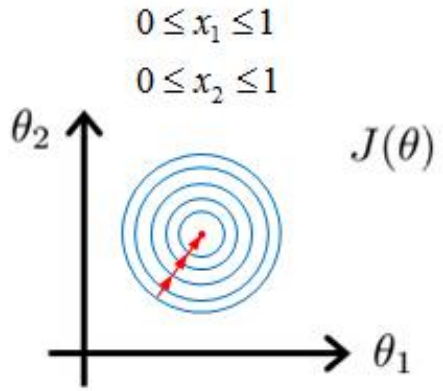
Min-max scaling

For all features:

- Subtract min(x_i)
- max(x_i)-min(x_i)

$$x_i \leftarrow \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}$$



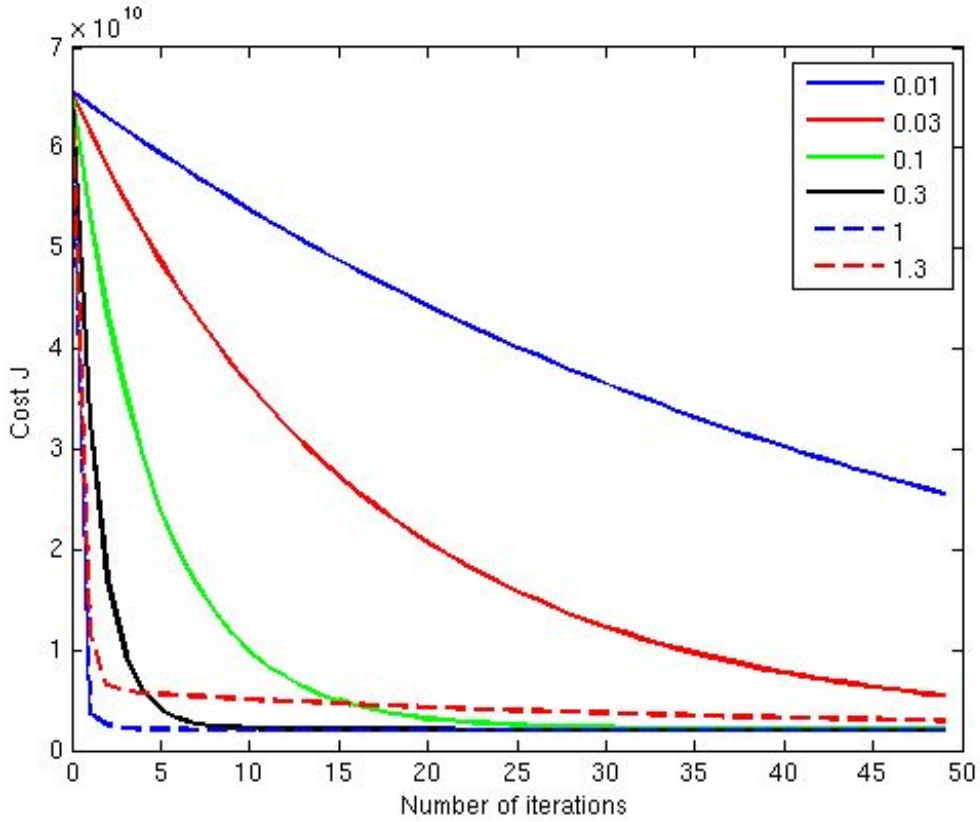


Gradient Descent Tips

Monitor convergence

Plot the value of the error function *J(θ)* at every iteration.

Check that the error becomes smaller. Plot for different learning rates to find the best one.



Gradient Descent

Pros

- **Will always converge** if learning rate α is chosen correctly
- **Fast** (time complexity is $O(m)$)
- Supports out of sample training (stochastic / mini batch G.D.)

Cons

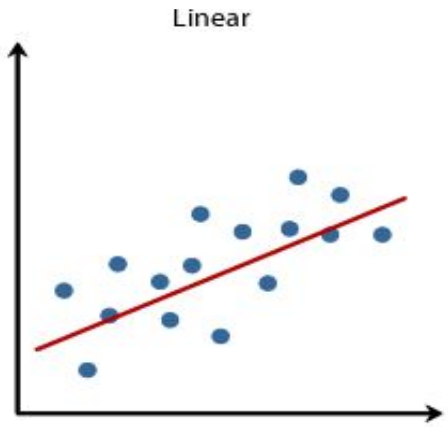
- **We have to choose learning rate** and initialize model parameters
- **Often takes many iterations**

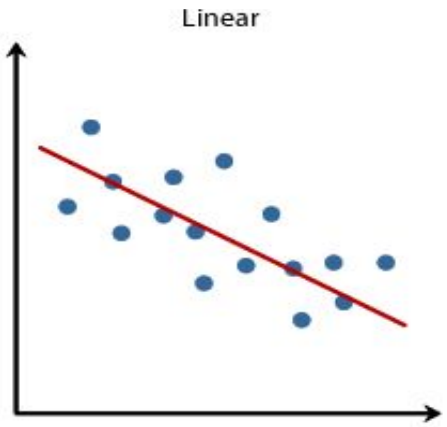
Classification

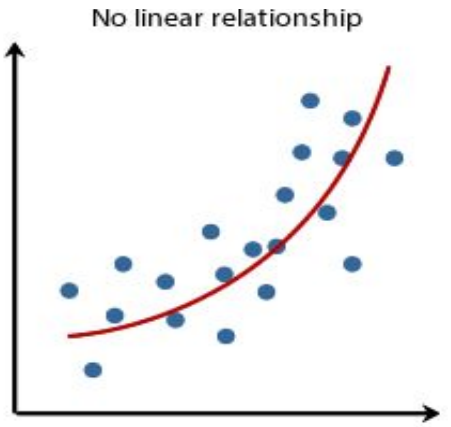
Regression vs. Classification

Regression:

- Continuous output y
- Quantitative approach
- Linear or Non-linear



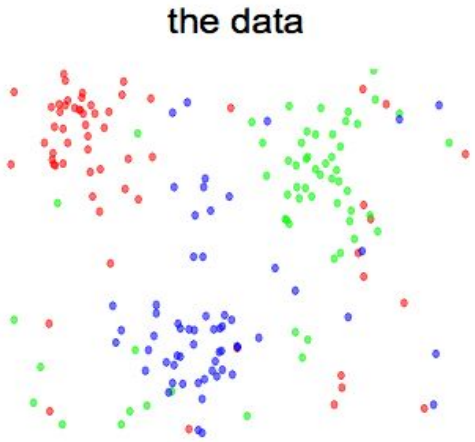




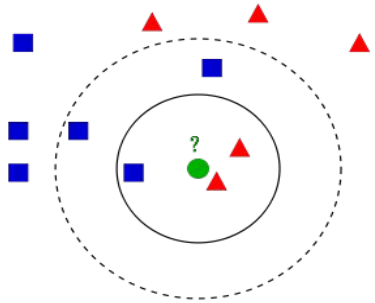
Classification:

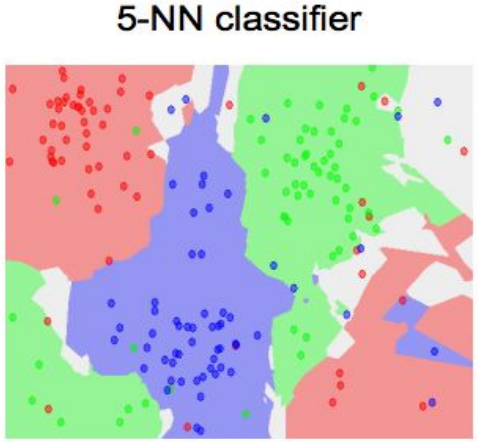
- Discrete output y
- Qualitative approach
- Linear or Non-linear

Ex. KNN,
Logistic, SVM, ..



KNN Method: Find the k nearest images and have them vote on the label (i.e. take the mode)

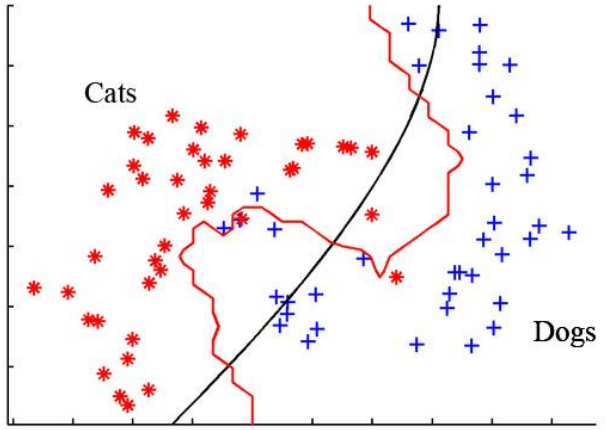


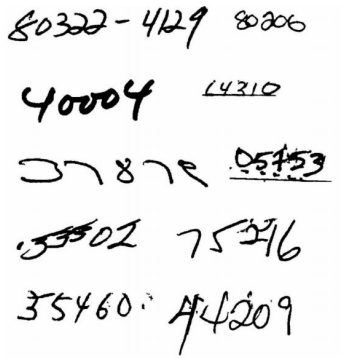


Examples of classification

Examples

- Weather: Sunny / Rainy
- Spam Detection
- Image Classification: Cats VS Dogs
- Image Classification: Recognizing Digits





Our Goal: Classify items

i.e. find the best hypothesis function $h_\theta(x)$ *that maps x to y*

X_i → **Model**

$$\hat{y} = f(x, \theta) = h_\theta(x)$$

Binary classification (cat vs dog):

y = 1 if picture is dog

Y = 0 if picture is cat

$$y \in \{0, 1\}$$

We have this data:
(X,Y): (x1,y1), (x2, y2) .. (xn,yn)

- xi and yi are arrays for each data element
- **Example:** xi = [12 15] = [height, weight], yi = male / female
- **For a picture:** xi = [32 x 32 x 3] multidim array, yi = cat / dog

Multi-class classification:

$y_i = [\, y_{i,0}, y_{i,1}, .. y_{i,k} \,]$

$y_i = [\, 1, 0, .. 0 \,]$

$$y \in \{0, 1, 2..k\}$$

Y(i,0) = 1 if picture is a dog
Y(i,1) = 1 if picture is a cat
Y(i,2) = 1 if picture is a elephant
etc.

Our Goal: To classify items.

We have this: (X,Y)

Xi → Model: $h_\theta(x)$

Actual Results:

$y_i = [\, y_{i,1}, y_{i,2}, .. y_{i,k} \,]$

$y_i = [\, 0, 1, .. 0 \,]$

Machine Learning Steps to train a classifier model

1. Choose model: $h_\theta(x)$ = estimate of Y
2. Define a loss function (J(θ)) = which is a function of **f**(Y_actual, Y_estimated)
3. Optimize across the parameter space (θ) to minimize the loss function

Linear Regression for Classification? (Not so good!)

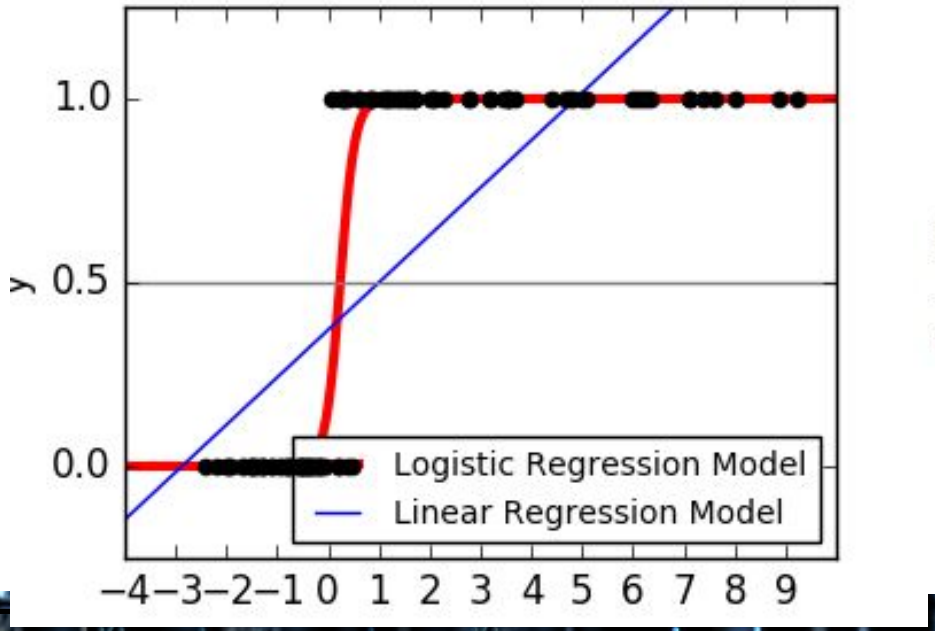
Why not choose a Linear model for classification?

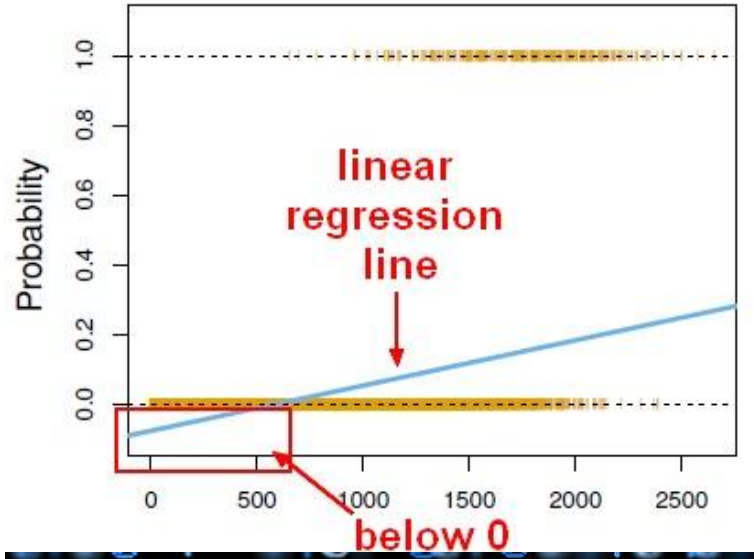
Because a line is not a good estimator for binary results (classification)

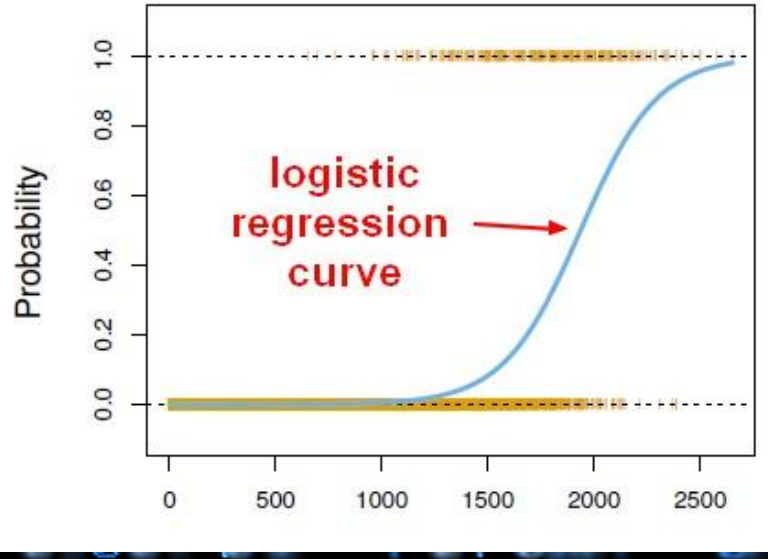
Linear model: $f(x,\theta) = h_\theta(x)$ = $\theta x_i = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \ldots$

Negative probabilities and biased towards majority class

Instead we use Logistic Regression!







Data X

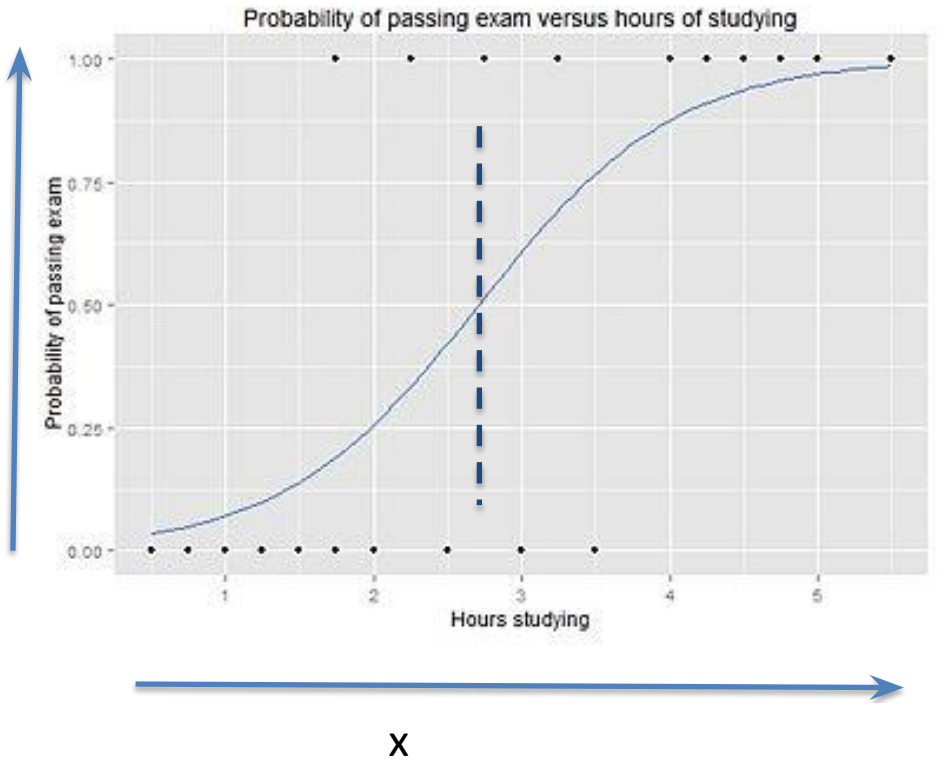
Logistic Regression

Classification Example

Data: students study for an exam
(x= hours studied, y = pass/not pass)

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

y, binary output
0 = fail
1 = pass



Problem:
We want to find a model that can predict the probability that the student passes given x hours of study

$$h_\theta(x) = P(y = 1|x;\theta)$$

If Prob >= 0.5, predict student will pass, y=1
If Prob < 0.5, predict student will fail, y=0

$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta)$$

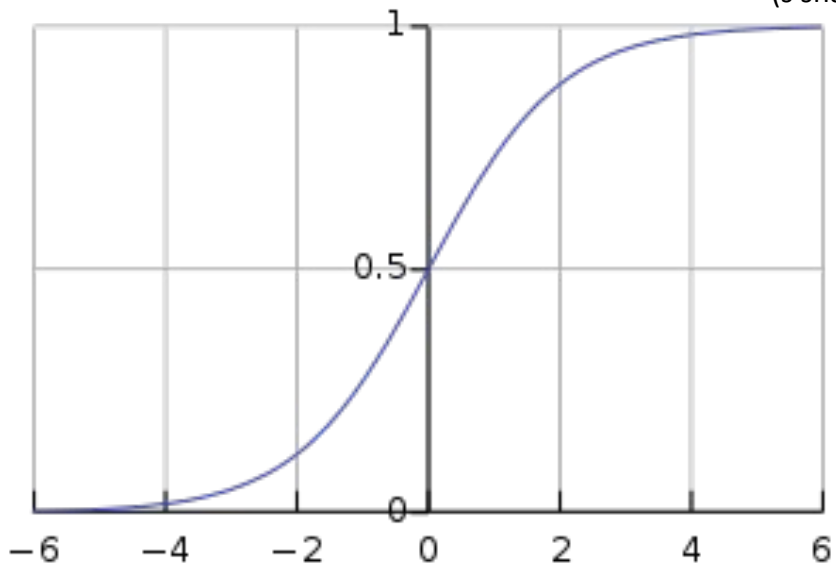
The logistic / sigmoid function

The sigmoid function:

$$z(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large positive t
z -> 1
Large negative t
z -> 0

(s shaped curve)



This function only evaluates to values between 0 and 1 for all real numbers (like a probability)

t, sum of weighted inputs + bias
$t = \theta_0 + x_1\theta_1$

$$z(t) = z(\theta^T x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}} = h_\theta(x)$$

If θ_1 is small→ slow rise
If θ_1 is large → fast rise

- **z(θx) is the probability that y = 1 given any x**
- The **decision boundary**, where the probability = 50%
- **z(θx) = ½** when $e^{-(\theta_0 + x_{i,1}\theta_1)} = 1$, ie $\theta_0 + x_1\theta_1 = 0$

Data X

Decision Boundary

The decision boundary separates our predicted categories in the feature space.

If we have two inputs, x_1 and x_2, the decision boundary is the line when the predicted probability for y=0 and y=1 is equal to 50%

$$h_\theta(x) = z(\theta^T X) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}} = 0.5$$

$$\Leftrightarrow$$

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

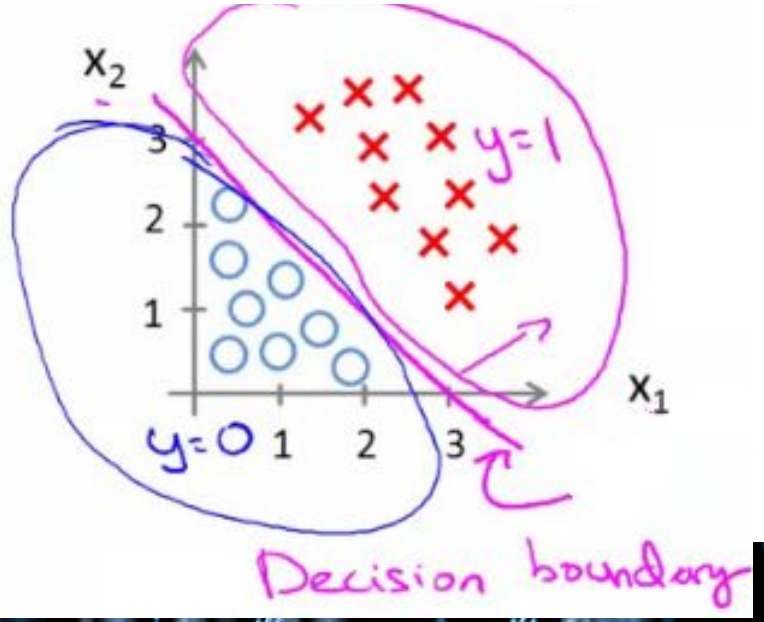
Example

$\theta_0 = -3 \quad \theta_1 = 1 \quad \theta_2 = 1$

Then $x_1 + x_2 - 3 \geq 0$

will predict y=1 and vice versa

(see example below)



Derivation of the logistic cost function

- How to choose parameters θ to find the best $h_\theta(x)$
- We need a **cost function J(θ)** that measures how well our classifier performs.

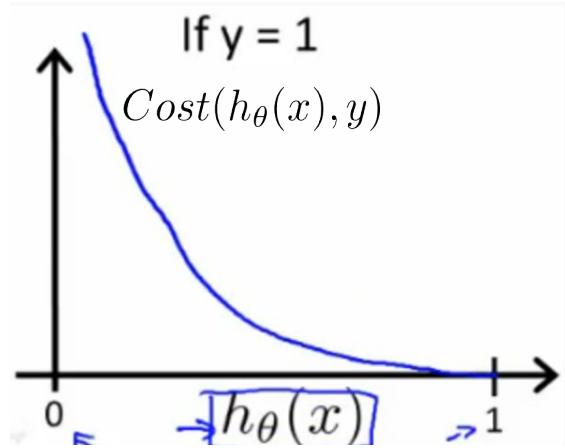
- Output y is binary and can only take on two values (0 or 1)
- Construct **J(θ)** that penalizes wrong predictions

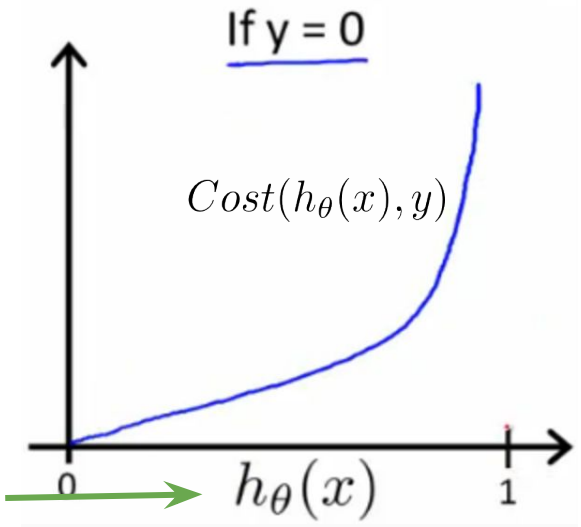
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \quad \text{if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \quad \text{if y} = 0$$

Cost plotted against predicted class probability when the true value is y=1 (top) or y=0 (bottom)



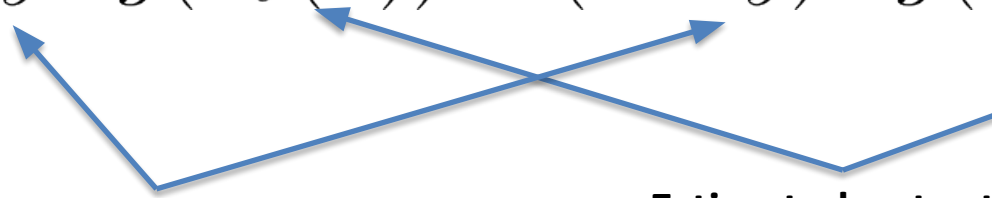


$$\frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1)}}$$

Logistic cost function

Cross Entropy for binary classification =

$$Cost(h_\theta(x), y) = -ylog(h_\theta(x)) - (1-y)log(1-h_\theta(x))$$



Actual output

y

Estimated output

$h_\theta(x)$

Note: Loss Function on the former slide can be added to form cross entropy for binary classification.

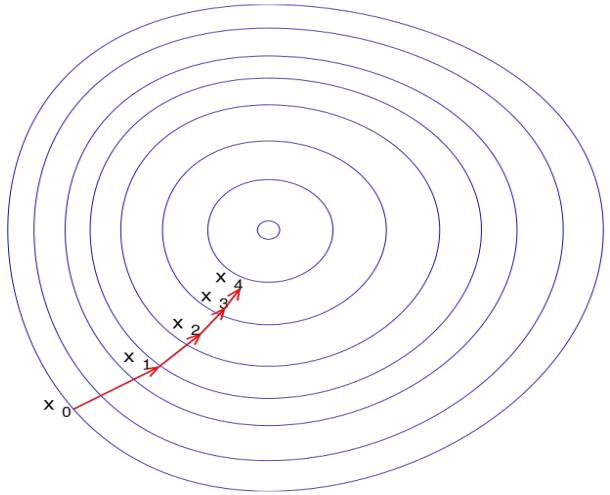
This cost function can be derived from the Maximum Likelihood estimation of the parameters.

Gradient Descent & Logistic Regression

J(θ) = is a cost function comparing our estimate $h_\theta(x)$ and the true y.

- Find optimal θ (first initialize θ with some random value)
- Take small steps in the direction where J(θ) is decreasing

<u>Update rule:</u> $\theta_{j+1} = \theta_j - [(\text{step size } \alpha) \text{ x gradient of } \mathbf{J(\theta)}]$



Formal update rule

looks exactly like Linear Regression, but note that $h_\theta(x)$ has changed

$$J(\theta)=\frac{-1}{m}[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Repeat {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

}

<u>Same as:</u>

Repeat {

$$\theta_j := \theta_j - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$$

}

Multi-class Logistic Regression: *One-vs-all*

$y \in \{0,1...k\}$

Sigmoid function:

$$z(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Large t -> 1, Small t -> 0

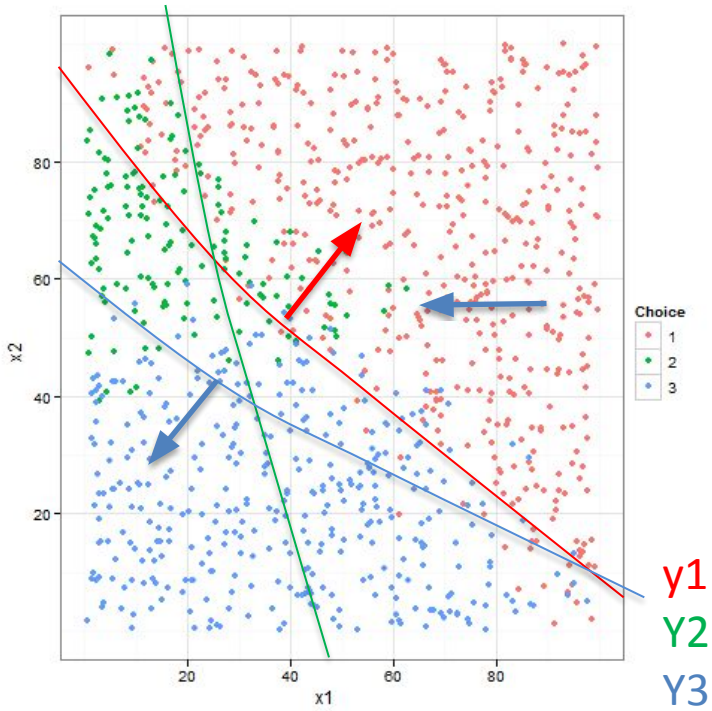
And if t has this form
$t = \theta x_i$ (in matrix form) $= \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} \cdots$

$$z(t) = \qquad z(\theta^T x) = h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots)}}$$

- Easily extends to multiple features (x1, x2, x3..)
- And multiple parameter weights

One-vs-all

- Take i:th class (against all other grouped into an alternative class), create decision boundary and calculate probability
- Final prediction will be the class that had the highest probability against all others.



$$h_\theta^{(0)}(x) = P(y = 0|x; \theta)$$

$$h_\theta^{(1)}(x) = P(y = 1|x; \theta)$$

...

$$h_\theta^{(k)}(x) = P(y = k|x; \theta)$$

$$prediction = max_i(h_\theta^{(i)}(x))$$

y1 boundary
Y2 boundary
Y3 boundary

Read about Softmax Regression for Multiclass Classification

p. 139 - 142 in the Textbook

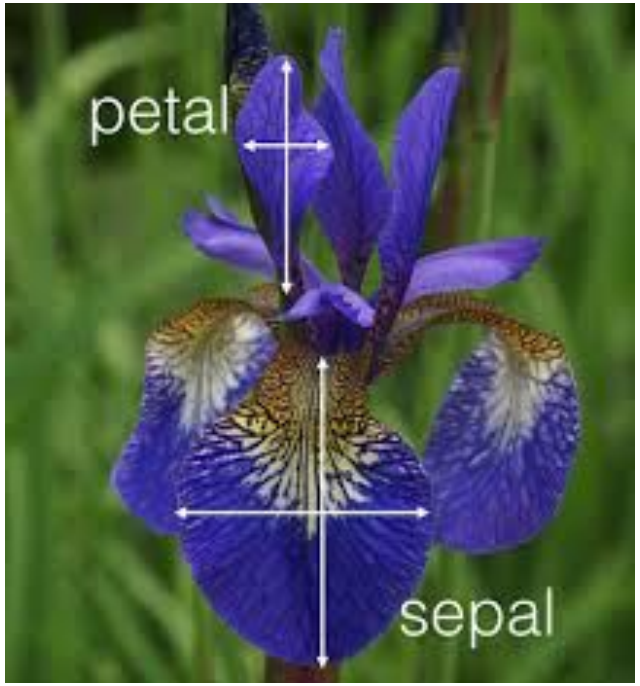
End of Section

References

- The material presented in this lecture references lecture material draws on the materials the following courses:
- UC Berkeley – CS 294-129 (Designing, Visualizing, and Understanding Deep Neural Networks): https://bcourses.berkeley.edu/courses/1453965/pages/cs294-129-designing-visualizing-and-understanding-deep-neural-networks
- Stanford – CS231n (Convolutional Neural Networks for Visual Recognition): http://cs231n.stanford.edu/
- Stanford – CS229 (Machine Learning) & Andrew Ng's Machine Learning at Coursera: http://cs229.stanford.edu/ & https://www.coursera.org/learn/machine-learning

Example Code: Logistic Regression in Scikit-learn

Example Code Sample with Logistic Regression Classifier



Input data

X: Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Y:
0 = 'setosa',
1 = 'versicolor',
2 = 'virginica’

```
print type (X)
print X[0:5]

<type 'numpy.ndarray'>
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]


print Y[0:5]
[0 0 0 0 0]
```

Data X

Example Code Sample with Logistic Regression Classifier

```
import numpy as np
from sklearn import linear_model, datasets

X = iris.data[:, 1:3]  # only the first two features.
Y = iris.target

# https://en.wikipedia.org/wiki/Logistic_regression
logreg = linear_model.LogisticRegression(C=1e5)

# we create an instance of Neighbours Classifier and fit the data.
logreg.fit(X, Y)

# predict a category for every row in X
Z = logreg.predict(X)
```

1 → 2 → 3 → 4 →

Class sklearn.linear_model.LogisticRegression

(penalty='l2',
dual=False,
tol=0.0001,
C=1.0,
fit_intercept=True,
intercept_scaling=1,
class_weight=None,
random_state=None,
solver='liblinear', max_iter=100,
multi_class='ovr', verbose=0,
warm_start=False,
n_jobs=1)

http://scikit-learn.org/stable/modules/generatedsklearn.linear_model.LogisticRegression.html

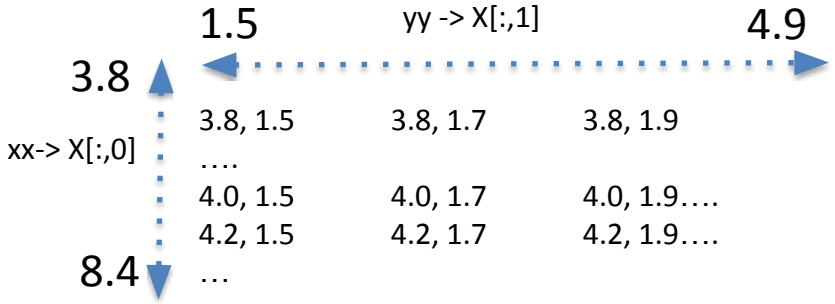
* Z[2] will be the predicted number for row X[2]

Code Samples with SciKit Learn

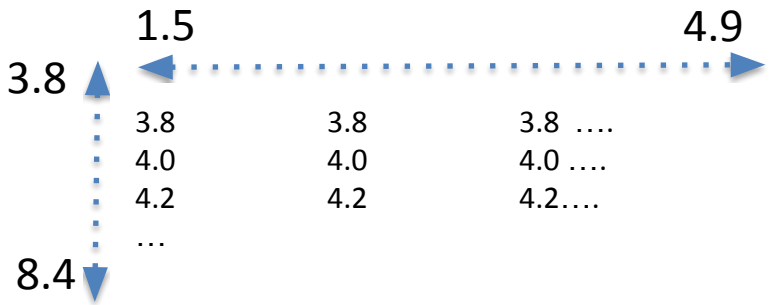
```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])
# numpy.ravel: Return a contiguous flattened array.
```

xx shape is (171, 231)
yy shape is (171, 231)

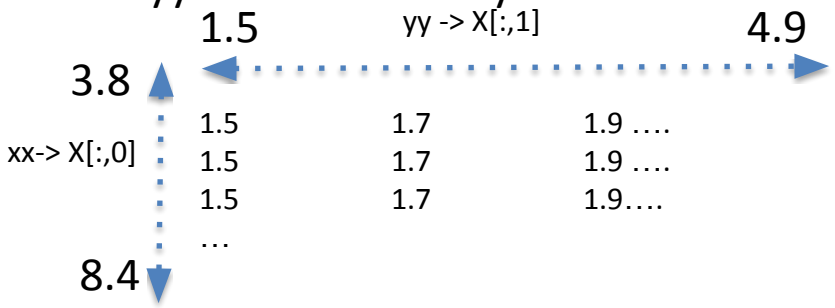
np.c returns shape (39501, 2)
[[3.8 1.5]
[3.82 1.5]
[3.84 1.5] …]
Z shape is (39501,)



xx is a matrix of all the first values

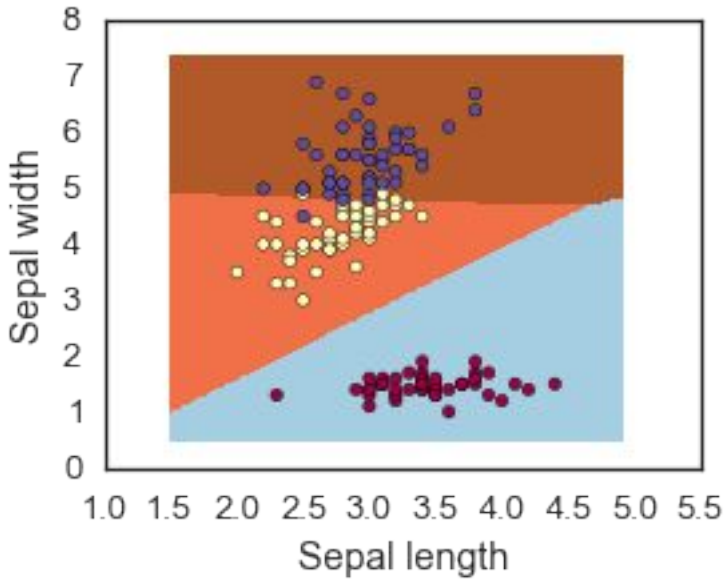


yy is matric of only the second values



Plotting the Results

```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k',
cmap=get_cmap("Spectral"))
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

#plt.xlim(xx.min(), xx.max())
#plt.ylim(yy.min(), yy.max())
#plt.xticks(())
#plt.yticks(())

plt.show()
```



Methods for LogisticRegression

Methods

`decision_function`(X)	Predict confidence scores for samples.
`densify`()	Convert coefficient matrix to dense array format.
`fit`(X, y[, sample_weight])	Fit the model according to the given training data.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict class labels for samples in X.
`predict_log_proba`(X)	Log of probability estimates.
`predict_proba`(X)	Probability estimates.
`score`(X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.
`sparsify`()	Convert coefficient matrix to sparse format.
`transform`(*args, **kwargs)	DEPRECATED: Support to use estimators as feature selectors will be removed in version 0.19.

`fit(X, y, sample_weight=None)` [source]

Fit the model according to the given training data.

Parameters: **X** : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vector, where n_samples is the number of samples and n_features is the number of features.

y : array-like, shape (n_samples,)

Target vector relative to X.

sample_weight : array-like, shape (n_samples,) optional

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

New in version 0.17: sample_weight support to LogisticRegression.

Returns: **self** : object

Returns self.

Fit and predict from ScikitLearn

`predict(X)` [source]

Predict class labels for samples in X.

Parameters: **X** : {array-like, sparse matrix}, shape = [n_samples, n_features]

Samples.

Returns: **C** : array, shape = [n_samples]

Predicted class label per sample.

Regularization

Why: To avoid over-fitting

How: You penalize your loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of your weights vector w

Your new loss function = $L(X,Y) + \lambda N(w)$

Tuning the regularization term λ: Cross-validation:

- divide your training data,
- train your model for a fixed value of λ and test it on the remaining subsets
- repeat this procedure while varying λ.
  Then you select the best λ that minimizes your loss function.

Shrinkage Methods II: An example