UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

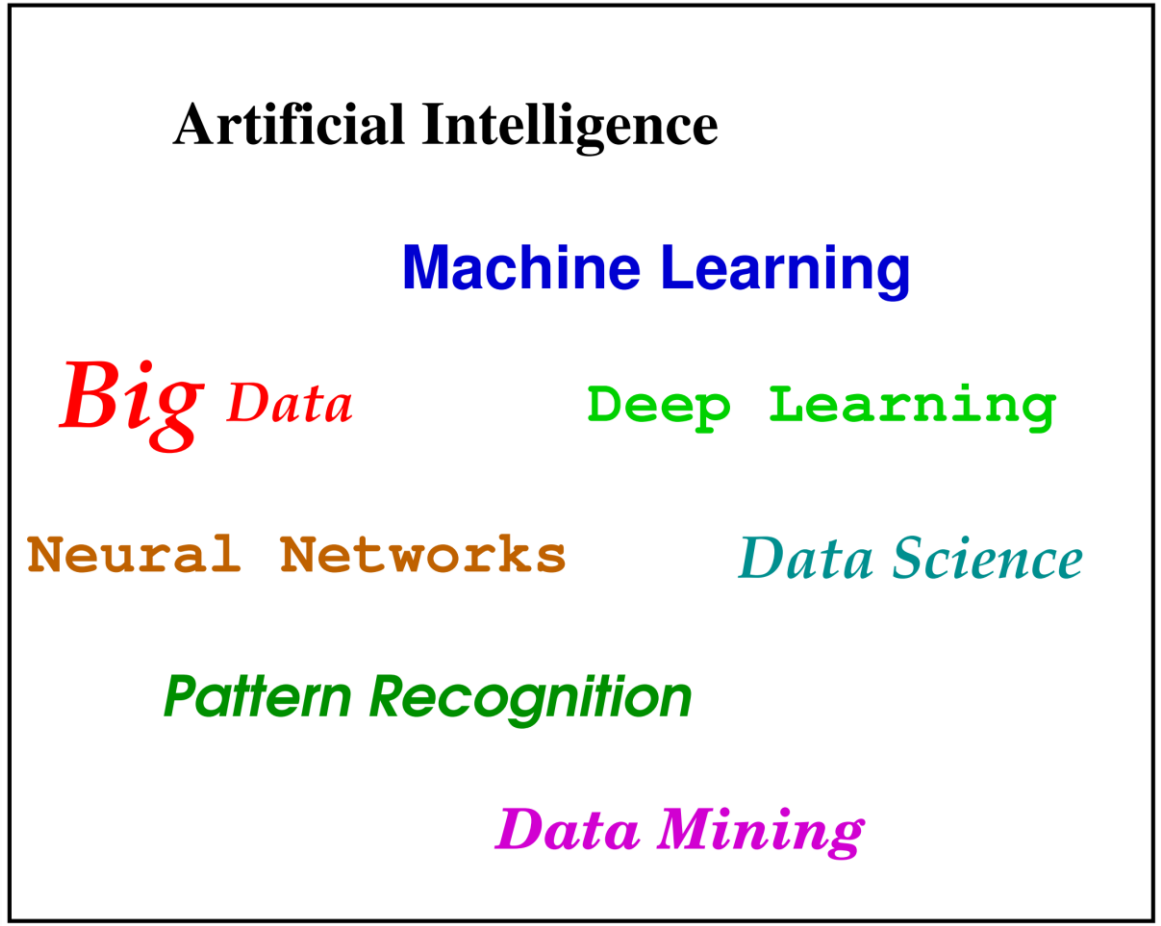
SPRING ‘23

LECTURE 1

INTRODUCTION

Instructor: Asst. Prof. Barış Başpınar

What are all these buzzwords?



Here is the dictionary!

Deep Learning = Neural Networks

Neural Networks $\subset$ Machine Learning

Machine Learning $\approx$ Artificial Intelligence

All other buzzwords $\approx$ Machine Learning

Timeline of Ups and Downs

AI: Artificial Intelligence ML: Machine Learning

- **1950's**: AI/ML is the great future.
- **1970's**: AI became a bad word.
- **1980's**: Neural Networks are the great future.
- **1990's**: Neural Networks are not that good.
- **2010's**: Neural Networks are great after all.
- **Now**: AI/ML is the great future.

Why Learning?

Jeopardy's Watson is a one-task machine. Big task, but one task. (2011)

What the buzzwords have in common

- The same core premise:
  - *Machine Learning*
  - *Artificial Intelligence*
  - *Data Mining*
  - *Pattern Recognition*

"Automated detection of a **pattern** based on the **data** "

Example: Credit Approval

Given the data of an applicant:

age	23 years
gender	male
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000

should we extend credit?

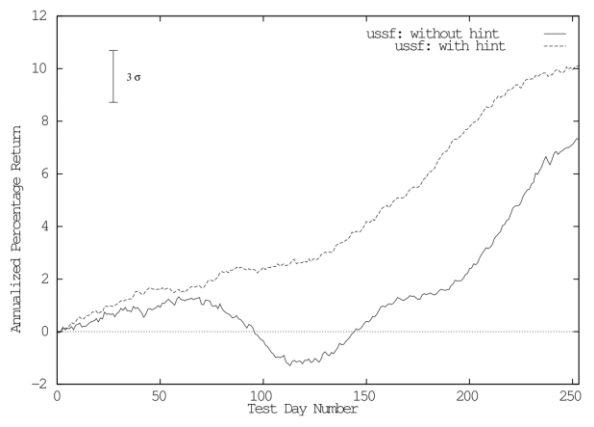
When should ML be used?

- ML is the technology of choice when:
  - A pattern exists.
  - We cannot pin it down mathematically.
  - We have a representative data set.

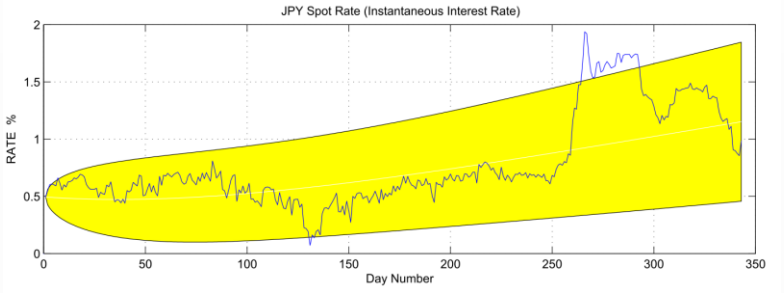
These criteria led to 3 waves of successful applications

1st Wave: Financial applications

- Market forecasting



- Financial model calibration



- Consumer and corporate credit assessment.

2nd Wave: E-commerce

- Recommender systems (Amazon, fashion, ...)

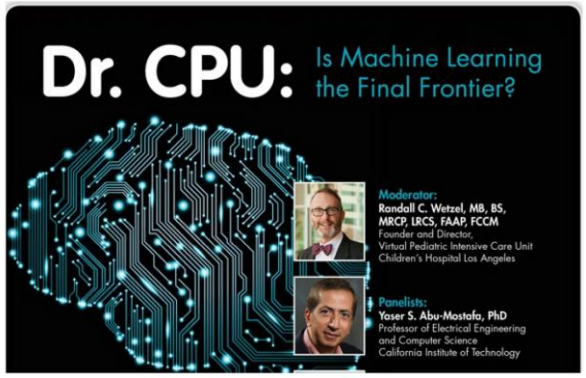
- Profiling

Famous ML e-commerce problem

US$1,000,000 Prize for the first 10% improvement (2006-2009)

3rd Wave: Medical Applications

- Medical diagnosis



- Data mining of medical records

The Growth of Machine Learning

- More Data:

  Enables us to pin down the pattern better.

  gradual → jump

- More Complex Models:

  Enables us to capture more complex patterns.

  gradual → jump

- More Computation:

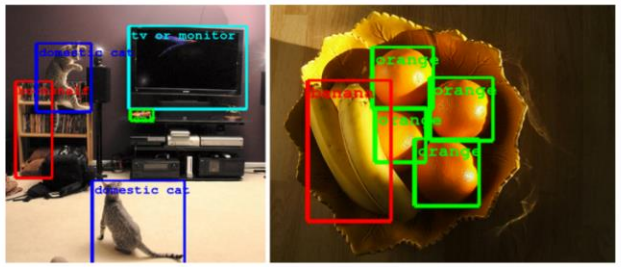
  Enables us to optimize very complex models.

  gradual → jump

ML Success Stories

The last years witnessed a huge surge in the practical impact of ML

speech recognition



object detection

machine translation

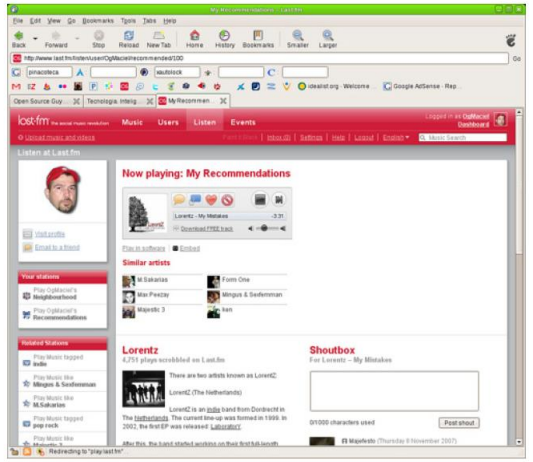
What happened?

A qualitative change in: data - models - computation

Different **data**: Total Profiling

Using multiple data sources:

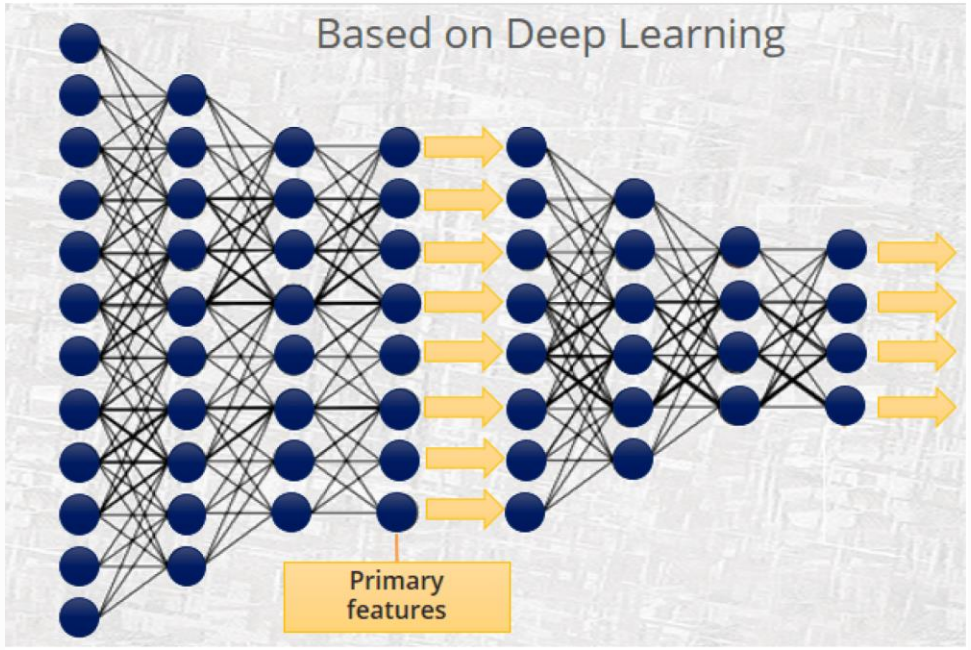
Movie preferences, Facebook posts, Amazon purchases, etc.



to profile a person.

What about the **models**?

A deep Neural Network with millions of parameters



Automated Feature Extraction

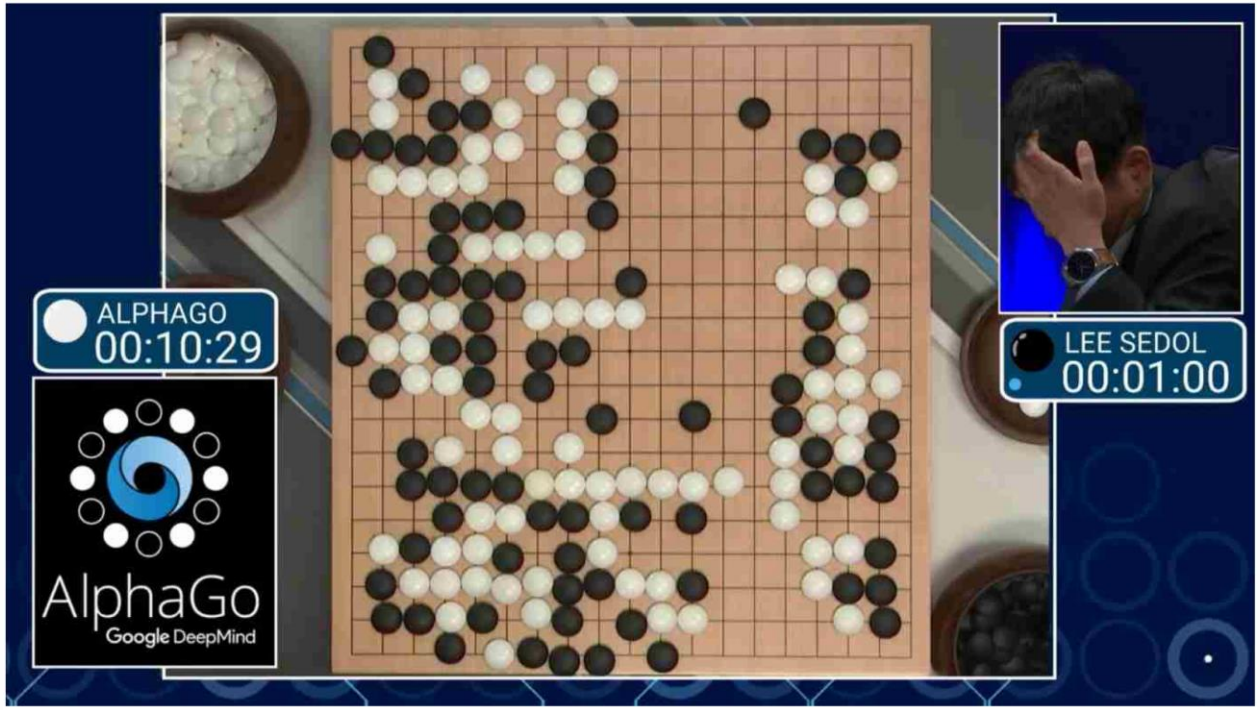
Jump in **computation** speed

Commercially available specialized hardware

Gain in ML speed is more than 2 orders of magnitude

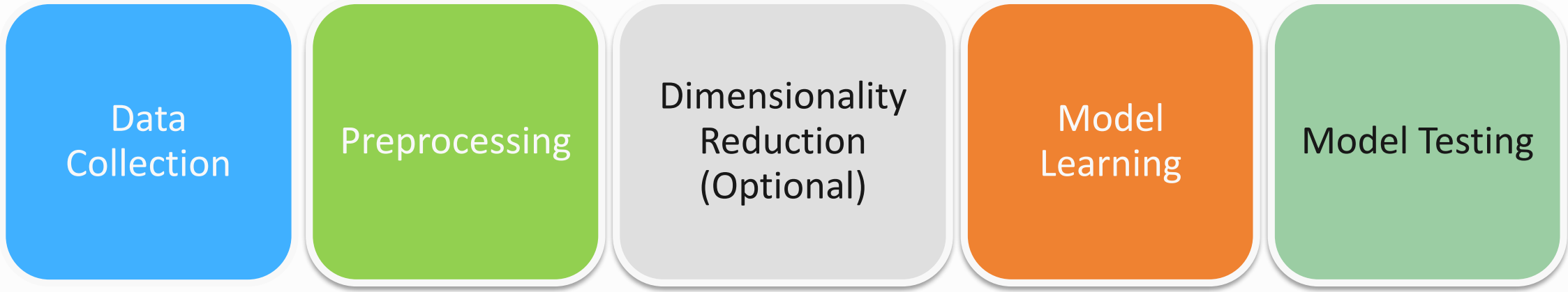
Real “Intelligence” achieved

From: Replicating human skills To: Beating human intelligence



ML system discovered novel strategic moves in the game of **Go** (2015)

Machine Learning Perspective



Learning Essentials

- ***Machine Learning:*** Data -> Train a Model on Data -> Make Predictions on New Data
- ***Feature Engineering:*** Extracting useful patterns from data that will be used for ML models for classification
- ***Feature Learning:*** Feature learning algorithms find the common patterns that are important to classify samples and extract them automatically to be used in a classification or regression process.
- ***Deep Learning:*** New methods and strategies designed to generate deep hierarchies of non-linear features so that Deep architectures with dozens of layers of non-linear hierarchical features can be trained

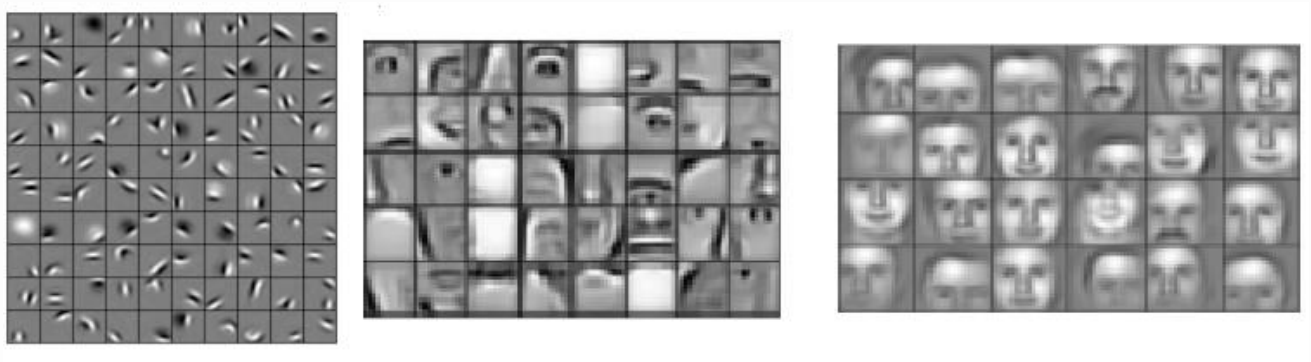
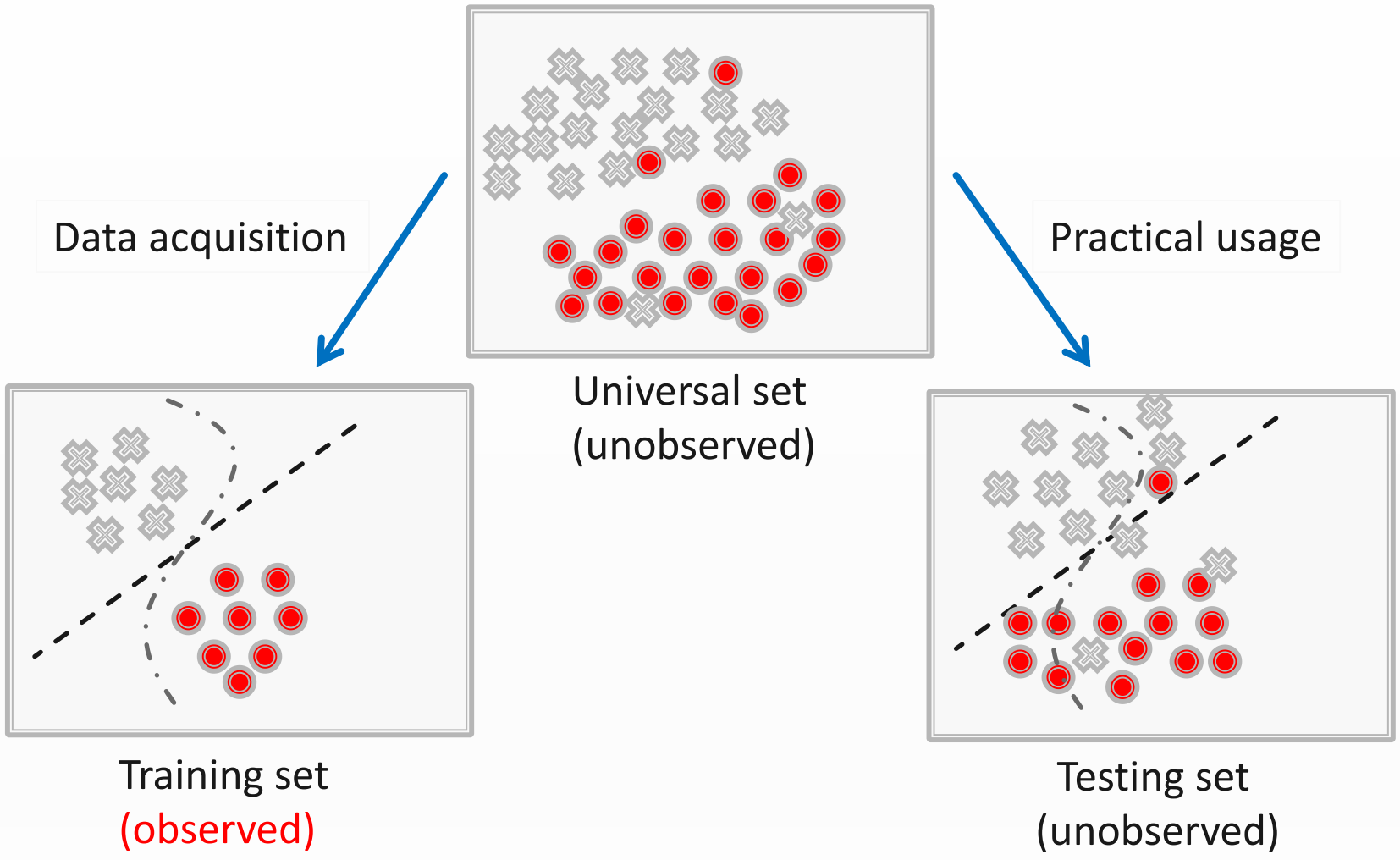


Image by Honglak Lee and colleagues (2011) , "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks"

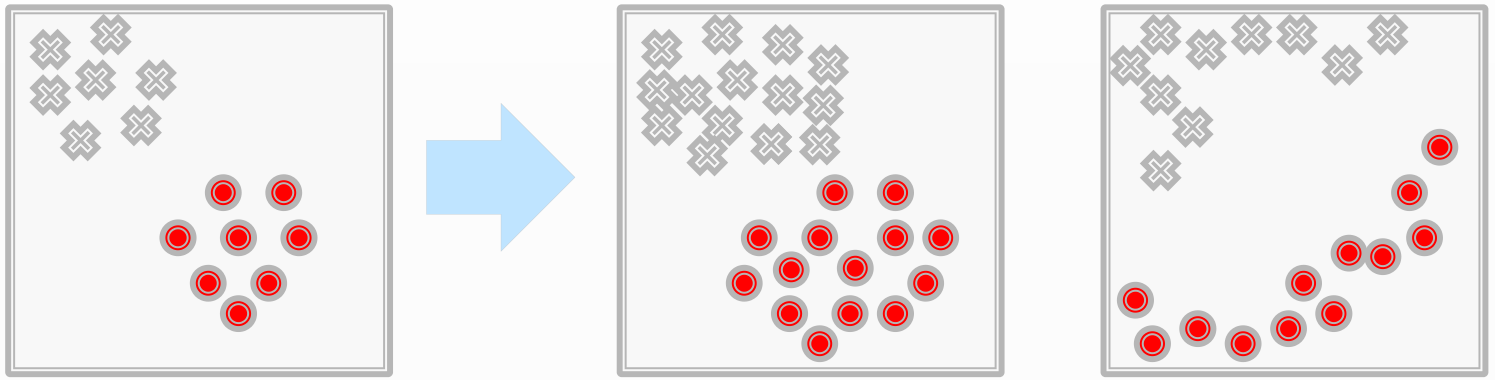
Training and testing



Training and testing

- Training is the process of making the system able to learn.

- No free lunch rule:
  - Training set and testing set come from the same distribution
  - Need to make some assumptions or bias



Performance

- There are several factors affecting the performance:
  - **Types of training** provided
  - The form and extent of any initial **background knowledge**
  - The **type of feedback** provided
  - The **learning algorithms** used

- Two important factors:
  - Modeling
  - Optimization

Algorithms

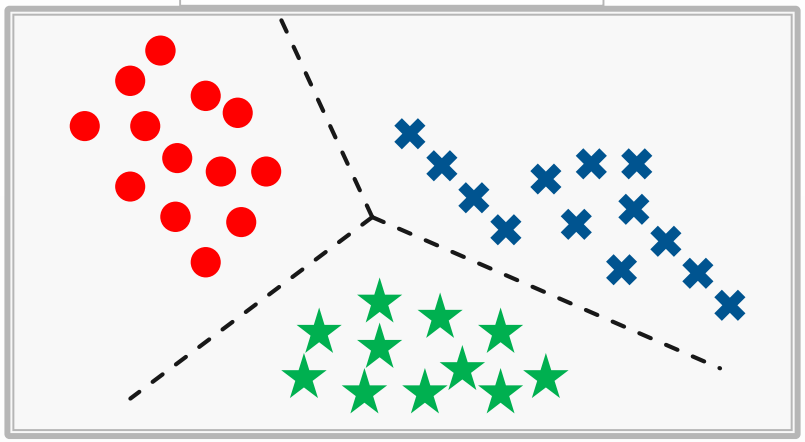
- The success of machine learning system also depends on the algorithms.
- The algorithms control the search to find and build the knowledge structures.
- The learning algorithms should extract useful information from training examples.

Algorithms

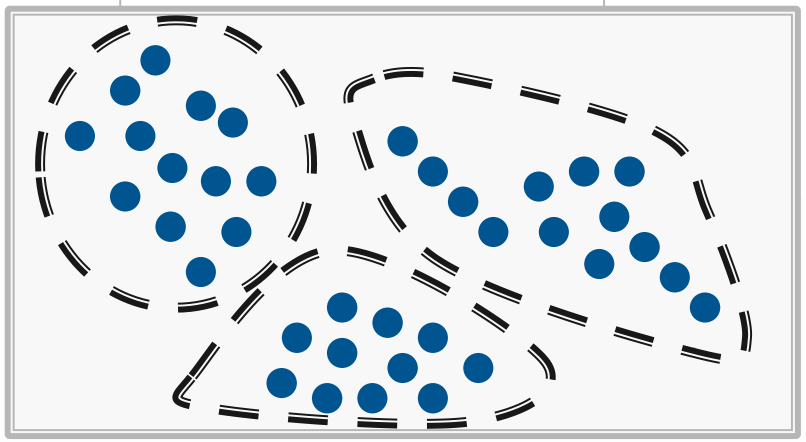
- **Supervised learning** ($\{x_n \in R^d, y_n \in R\}_{n=1}^N$)
  - Prediction
  - Classification (discrete labels), Regression (real values)
- **Unsupervised learning** ($\{x_n \in R^d\}_{n=1}^N$)
  - Clustering
  - Probability distribution estimation
  - Finding association (in features)
  - Dimension reduction
- **Semi-supervised learning**
- **Reinforcement learning**
  - Decision making (robot, chess machine)

Algorithms

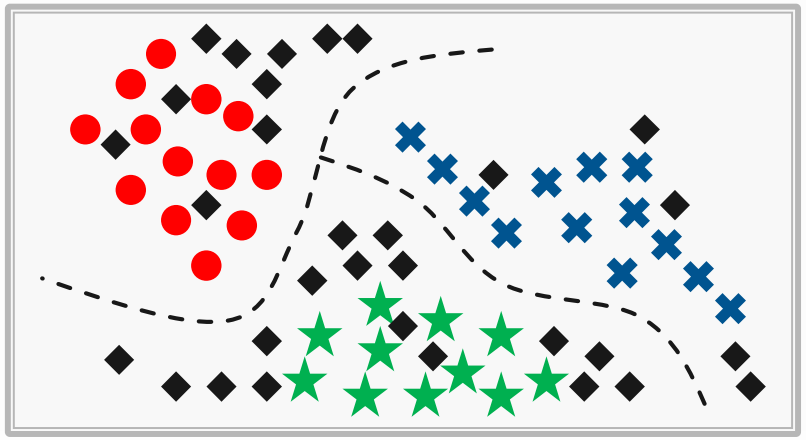
Supervised learning



Unsupervised learning

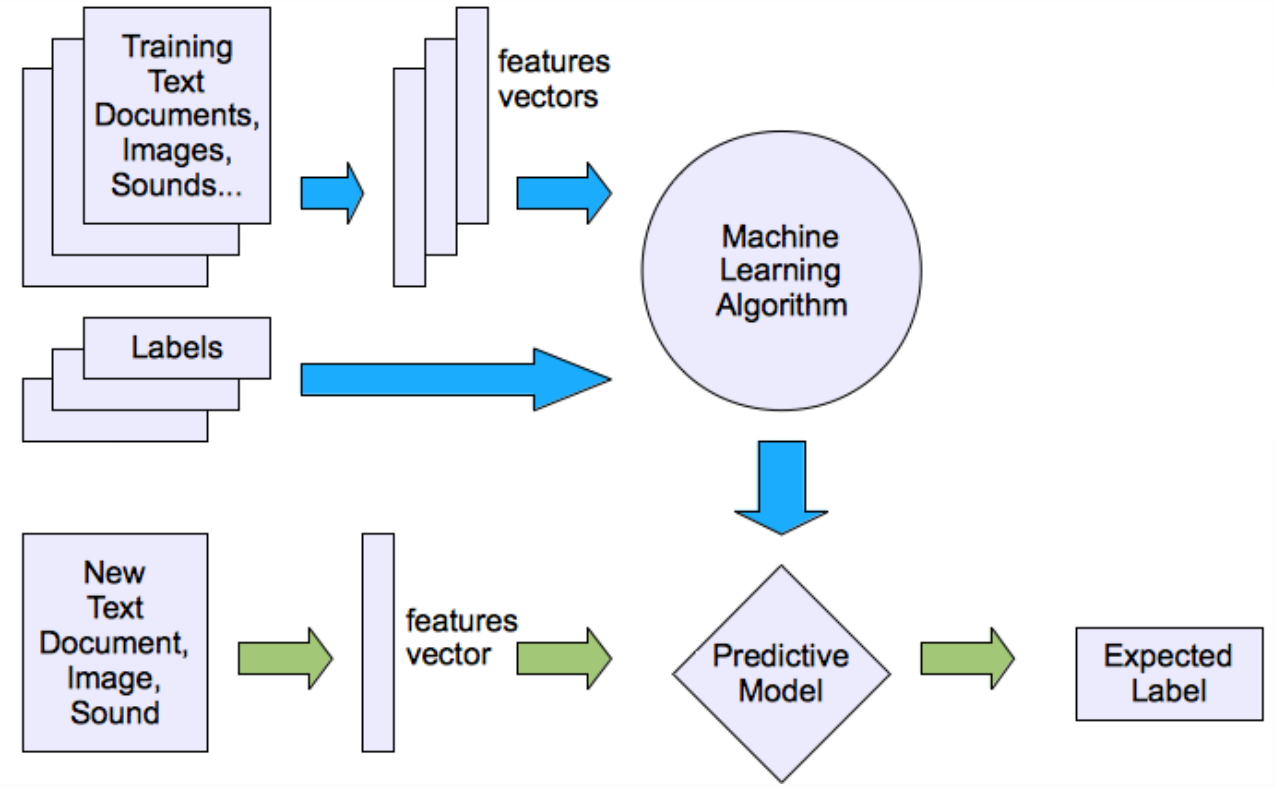


Semi-supervised learning



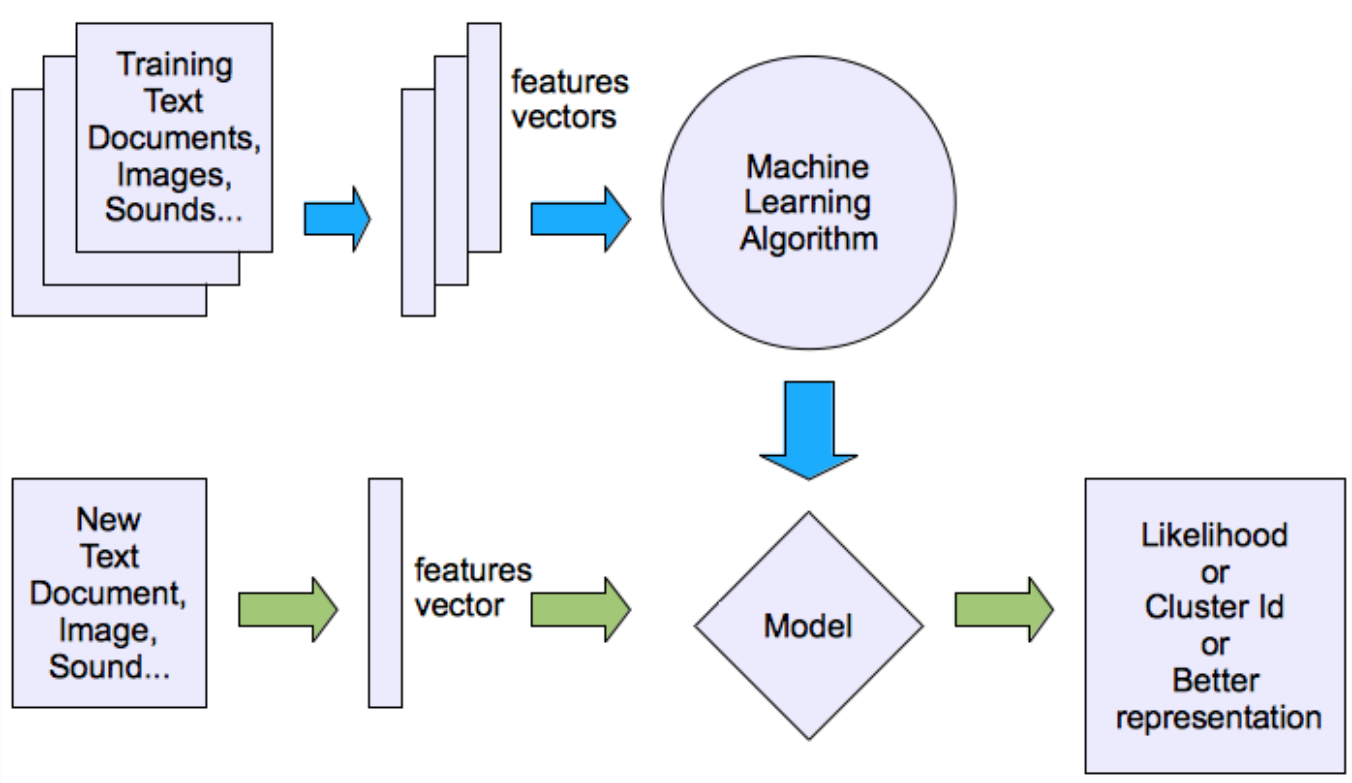
Machine learning structure

- Supervised learning



Machine learning structure

- Unsupervised learning



What are we seeking?

- Supervised: Low E-out or maximize probabilistic terms

$$error = \frac{1}{N}\sum_{n=1}^{N}[y_n \neq g(x_n)]$$

E-in: for training set

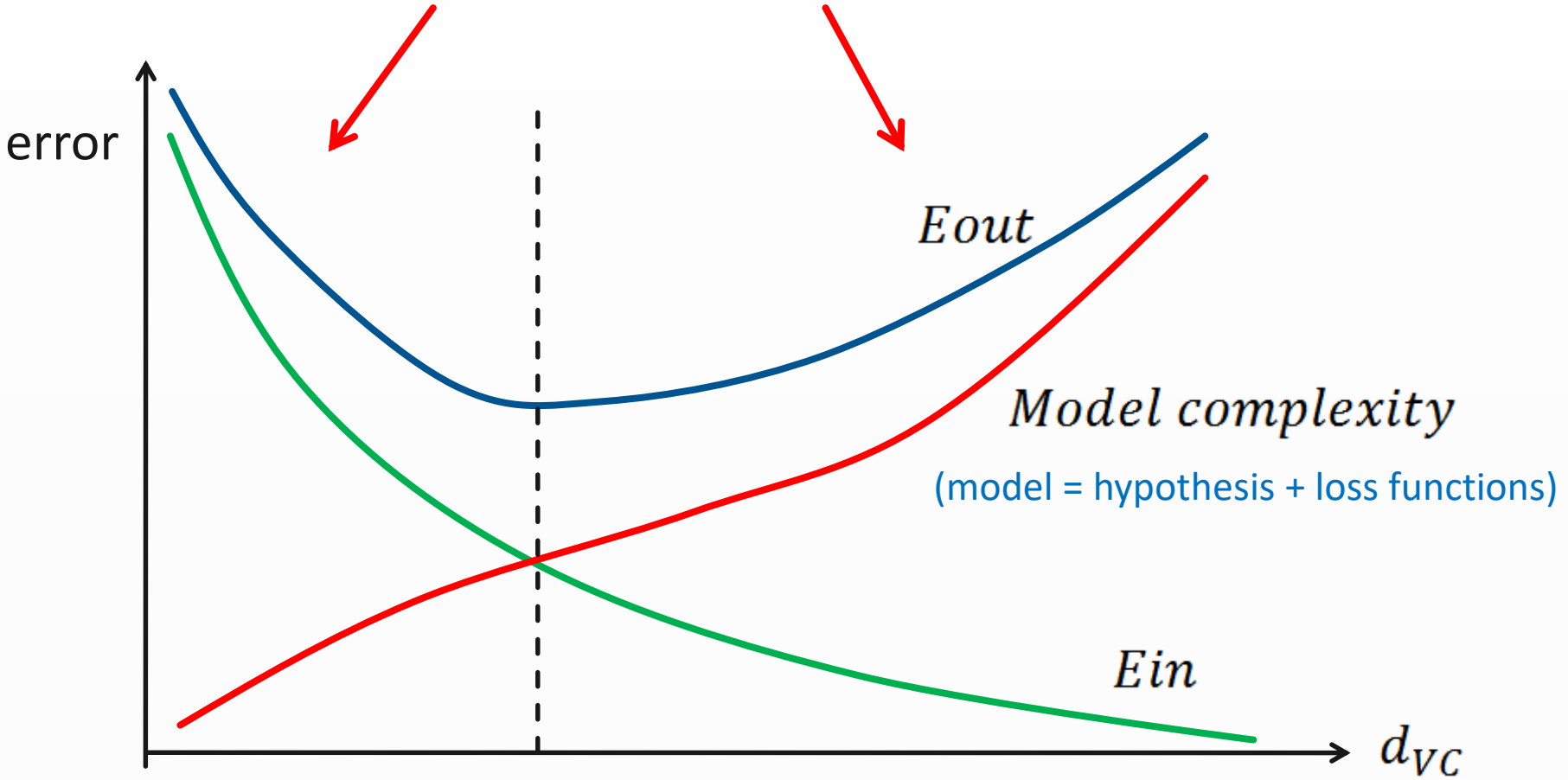
E-out: for testing set

$$Eout(g) \leq Ein(g) \pm O\left(\sqrt{\frac{d_{VC}}{N} ln\, N}\right)$$

- Unsupervised: Minimum quantization error, Minimum distance, MAP, MLE(maximum likelihood estimation)

What are we seeking?

Under-fitting VS. Over-fitting (fixed *N*)

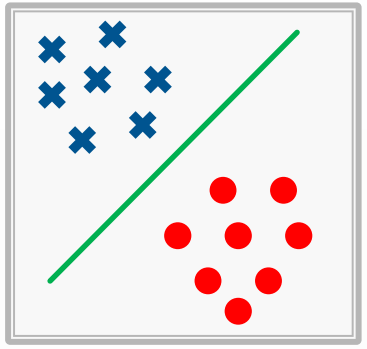


Learning techniques

- Supervised learning categories and techniques
  - **Linear classifier** (numerical functions)
  - **Parametric** (Probabilistic functions)
    - Naïve Bayes, Gaussian discriminant analysis (GDA), Hidden Markov models (HMM), Probabilistic graphical models
  - **Non-parametric** (Instance-based functions)
    - *K*-nearest neighbors, Kernel regression, Kernel density estimation, Local regression
  - **Non-metric** (Symbolic functions)
    - Classification and regression tree (CART), decision tree
  - **Aggregation**
    - Bagging (bootstrap + aggregation), Adaboost, Random forest

Learning techniques

- Linear classifier



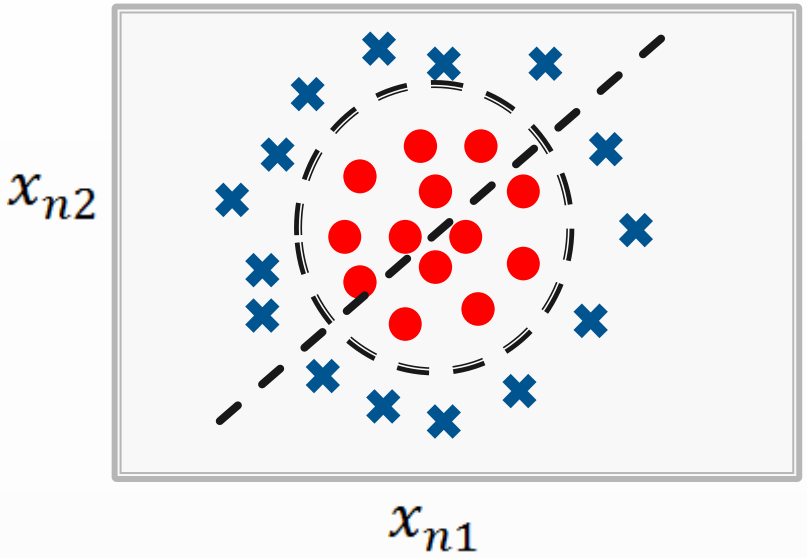
$$g(x_n) = sign(w^T x_n)$$

, where *w* is an *d*-dim vector (learned)

- Techniques:
  - Logistic regression
  - Support vector machine (SVM)
  - Multi-layer perceptron (MLP)

Learning techniques

- Support vector machine (SVM):



$$x_n = [x_{n1}, x_{n2}]$$

$$x_n = [x_{n1}, x_{n2}, x_{n1} * x_{n2}, {x_{n1}}^2, {x_{n2}}^2]$$

$$g(x_n) = sign(w^T x_n)$$

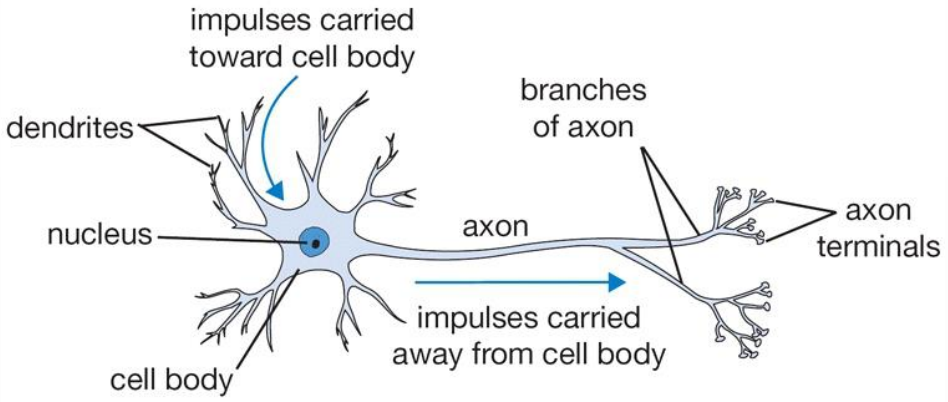
  - Non-linear case

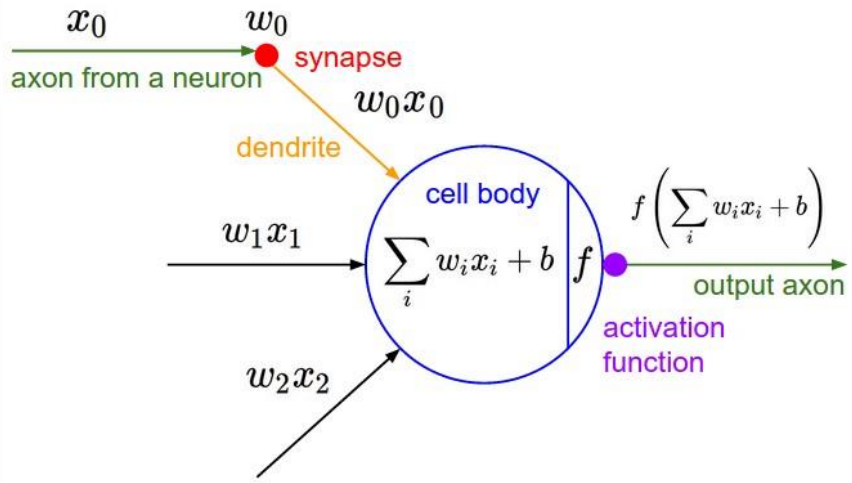
- Linear to nonlinear: **Feature transform** and **kernel function**

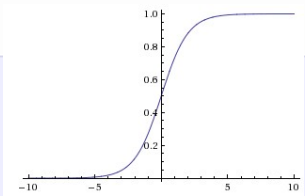
Learning techniques

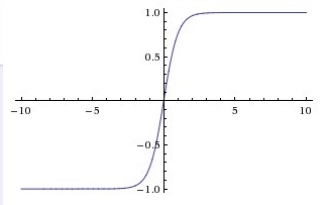
- Unsupervised learning categories and techniques
  - **Clustering**
    - K-means clustering
    - Spectral clustering
  - **Density Estimation**
    - Gaussian mixture model (GMM)
    - Graphical models
  - **Dimensionality reduction**
    - Principal component analysis (PCA)
    - Factor analysis

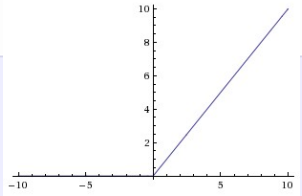
Deep Learning Essentials











```
class Neuron(object):
  # ...
  def forward(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

from: CS231n Convolutional Neural Networks for Visual Recognition, http://cs231n.github.io/neural-networks-1/

Neural Network Architectures

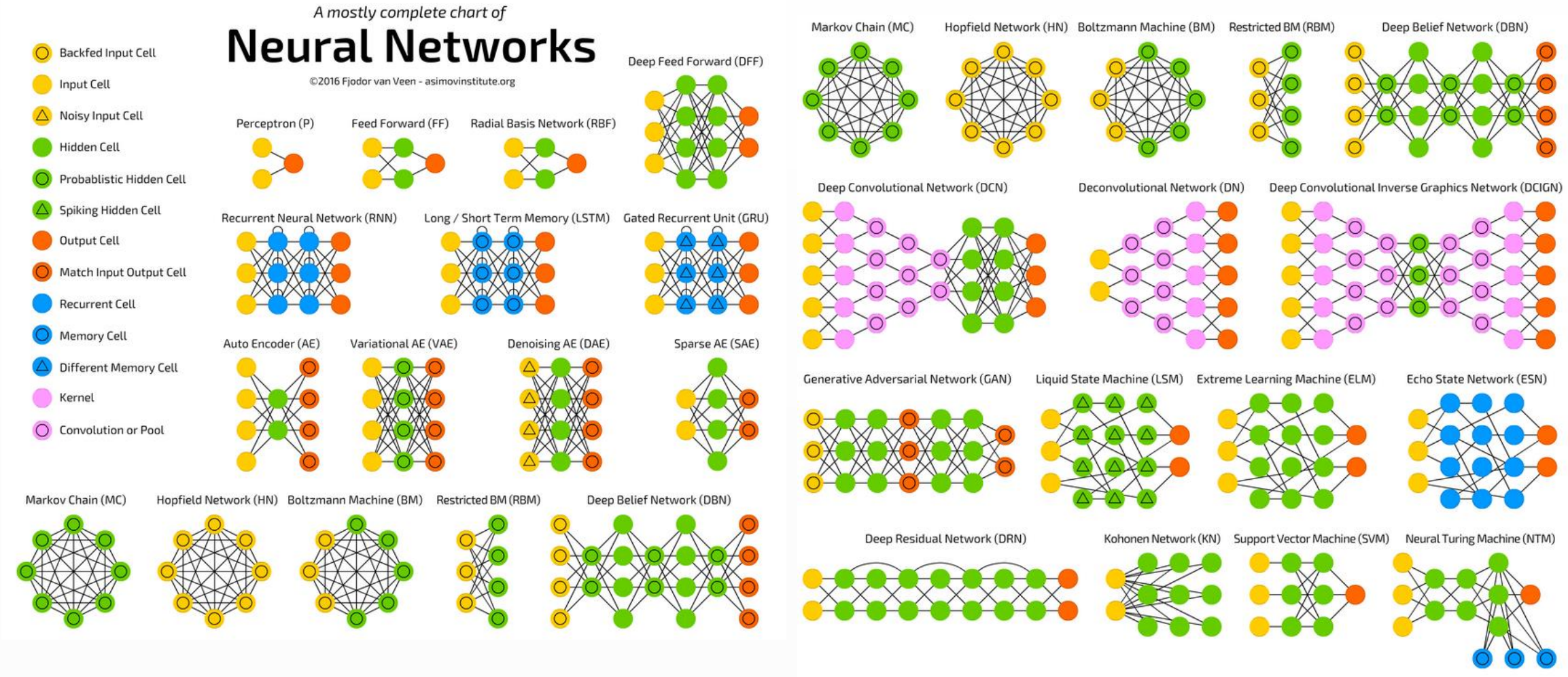


Image: The Asimov Institute – The Neural Network Zoo - http://www.asimovinstitute.org/neural-network-zoo/

Applications

- Face detection
- Object detection and recognition
- Image segmentation
- Multimedia event detection
- Economical and commercial usage

References

- W. L. Chao, J. J. Ding, “Integrated Machine Learning Algorithms for Human Age Estimation”, NTU, 2011.
- Y. S. Abu-Mostafa, “Artificial Intelligence: Evolution & Revolution (& Hype!)”, 2018.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

LECTURE 2

MACHINE LEARNING BASICS

Instructor: Asst. Prof. Barış Başpınar

Learning Algorithms: Tasks

- A learning algorithm is an algorithm that performs a task by using data.
- A learning problem usually has three components: Task, Performance and Experience
- Here are some common learning tasks
  - **Classification**: Find a function that maps input data to one of k categories
    - Example: Object recognition
  - **Classification with missing inputs**: Find a function or a collection of functions that maps input data to one of k categories, where the input vector might have missing values
    - Example: Medical diagnosis
  - **Regression**: Find a function that predicts a continuous output
    - Example: Predict expected amount an insured person will make
  - **Transcription**: Observe some unstructured data and transcribe it into discrete, textual form
    - Example: Image translation, speech recognition

Learning Algorithms: Tasks

- Here are more learning tasks
  - **Machine Translation**: Convert from one language to another
    - Example: Translation for natural languages
  - **Structured Output**: Any task that involves the output as a vector with important relationship among its elements
    - Example: Break down a sentence into nouns, verbs, etc, image captioning
  - **Anomaly Detection**: Flag unusual or atypical objects
    - Example: Spam filtering, credit card fraud detection
  - **Data Completion**: Estimate missing values in data
    - Example: Product recommendation
  - **Denoising**: Recover a clean example from a corrupt one
    - Example: Image and video denoising
  - **Density Estimation**: Compute the $p(\boldsymbol{x})$ that generated the data
    - Example: Estimate the structure of a PGM

Learning Algorithms: Performance

- We need a quantitative measure P to measure how well our algorithm is performing on task T
- For classification and translation tasks, we measure the performance by accuracy,
  - Just divide the number of correctly classified examples to number of all examples
- Performance is evaluated on *test data*, data that has never been seen by the algorithm before
  - This set is usually different from the training data
- Sometimes selecting the correct performance criteria can be a complicated task itself
  - For translation, should we measure the accuracy on word by word basis or a sentence by sentence basis?
  - For regression, should we penalize small medium frequency mistakes, or large rare mistakes

Learning Algorithms: Experience

- Many learning algorithms are categorized as supervised or unsupervised based on the data that experience
  - Roughly speaking, unsupervised algorithms try to determine $p(\boldsymbol{x})$ from $\boldsymbol{x}$
  - Whereas the supervised algorithm try to determine $p(\boldsymbol{y} \mid \boldsymbol{x})$ from $(\boldsymbol{x}, \boldsymbol{y})$
- However, sometimes the lines between the two categories get blurry
  - In semi-supervised algorithms, where the data is only partially labelled
  - Unsupervised algorithms can also be used for supervised learning
    - Simply estimate $p(\boldsymbol{x}, \boldsymbol{y})$ and then marginalize
  - And vice versa:
    - Use chain rule $p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$ to turn an unsupervised problem to a collection of n supervised problems

Learning Algorithms: Linear Regression Example

- Let's use a linear regression example to clarify these subjects
- In linear regression, our model is as follows:

$$\hat{y} = \boldsymbol{w}^{\top}\boldsymbol{x}$$

  - Where $\boldsymbol{w} \in R^n$ is a weight vector. Hence the task is to find $\boldsymbol{w}$ that does well according to some performance measure.

- A popular type of performance measure for these type of tasks is the mean squared error

$$\text{MSE}_{test} = \frac{1}{m}\sum_i \left(\hat{y}^{(\text{test})} - y^{(\text{test})}\right)^2$$

- The experience is gained by observing the dataset $\left(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})}\right)$
  - It can be shown that the optimal solution is found by $\nabla_{\boldsymbol{w}}\text{MSE}_{train} = 0$

$$\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}$$

Learning Algorithms: Linear Regression Example

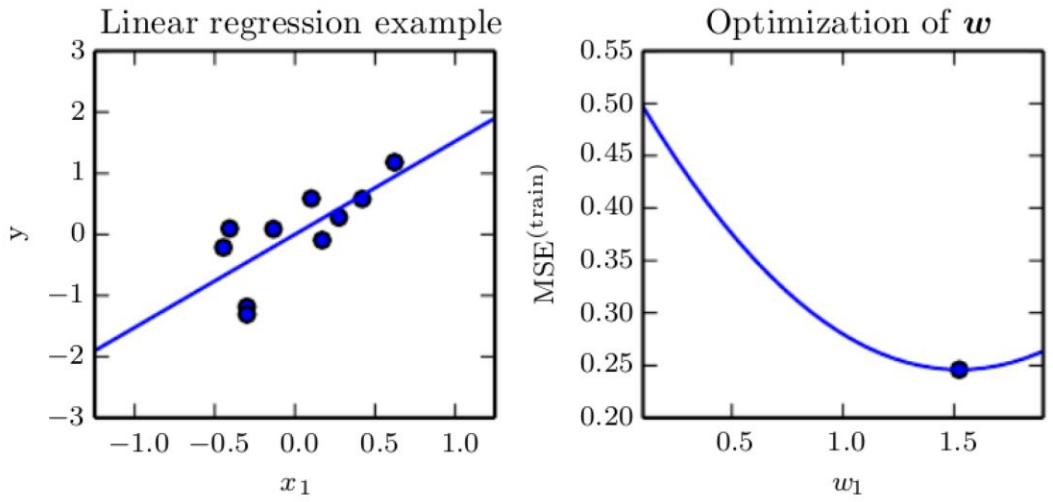


Figure 5.1: A linear regression problem, with a training set consisting of ten data points, each containing one feature. Because there is only one feature, the weight vector $\boldsymbol{w}$ contains only a single parameter to learn, w_1. *(Left)* Observe that linear regression learns to set w_1 such that the line $y = w_1 x$ comes as close as possible to passing through all the training points. *(Right)* The plotted point indicates the value of w_1 found by the normal equations, which we can see minimizes the mean squared error on the training set.

Capacity, Overfitting and Underfitting: Generalization

- A central challenge in machine learning is to perform well on new, **previously unseen** inputs.
  - This property is known as generalization. The error on the test data is referred to as *generalization error*
- But how can we say something about the test data, if all we see is training data?
  - If these datasets were generated arbitrarily, we can't...
  - However, if they come from the same distribution $p(\boldsymbol{x}, y)$, then we can say something! For instance, their mean should be equal
- In general, we want to find a $\boldsymbol{w}$ such that:
  I. Make the training error small
  II. Make the gap between training and test error small

  - Unfortunately, we rarely do both good at the same time
  - Failing on I, is known as underfitting, our model's capacity is not sufficient to capture $p(\boldsymbol{x}, y)$
  - Success on I but failing on II, is known as overfitting, our model's capacity is so high that it becomes overtuned to work only on training data

Capacity, Overfitting and Underfitting: Generalization

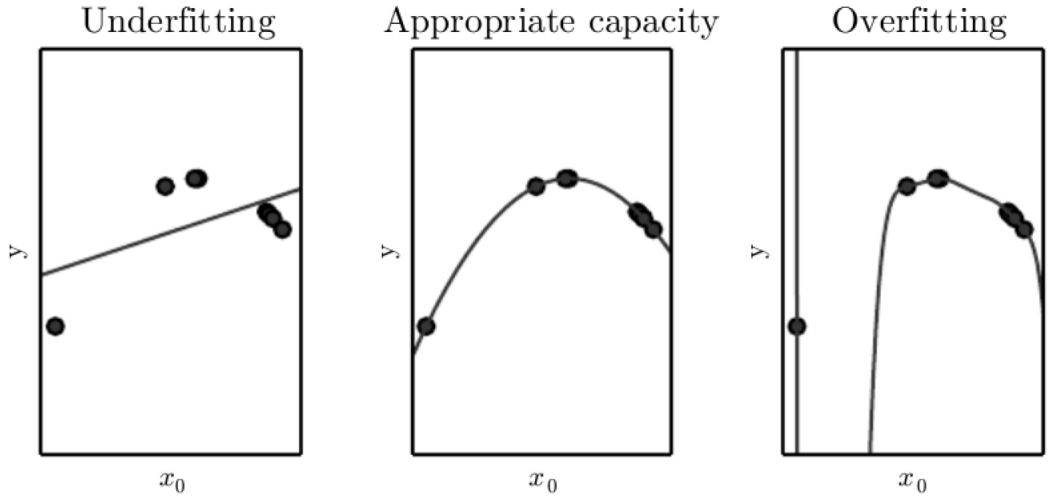


Figure 5.2: We fit three models to this example training set. The training data was generated synthetically, by randomly sampling x values and choosing y deterministically by evaluating a quadratic function. (*Left*) A linear function fit to the data suffers from underfitting—it cannot capture the curvature that is present in the data. (*Center*) A quadratic function fit to the data generalizes well to unseen points. It does not suffer from a significant amount of overfitting or underfitting. (*Right*) A polynomial of degree 9 fit to the data suffers from overfitting. Here we used the Moore-Penrose pseudoinverse to solve the underdetermined normal equations. The solution passes through all of the training points exactly, but we have not been lucky enough for it to extract the correct structure. It now has a deep valley in between two training points that does not appear in the true underlying function. It also increases sharply on the left side of the data, while the true function decreases in this area.

Capacity, Overfitting and Underfitting: Capacity

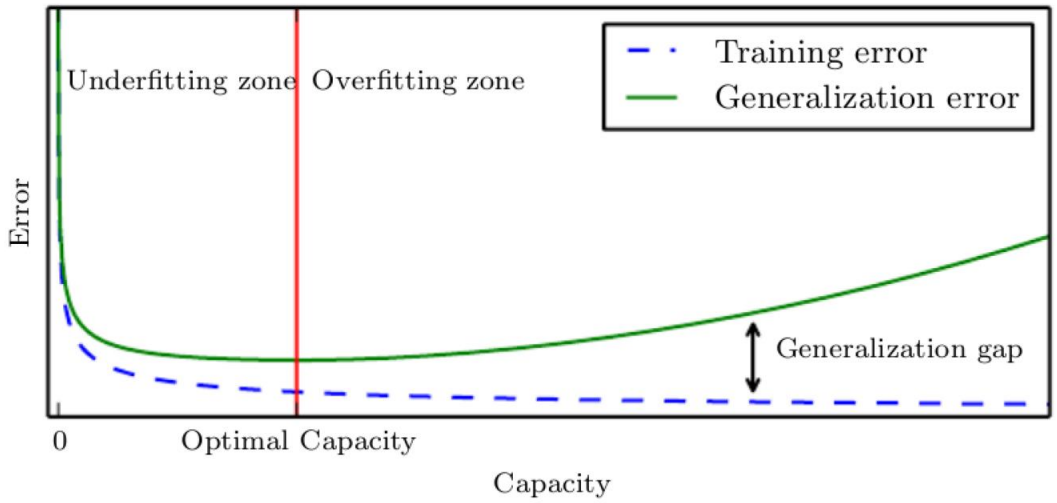
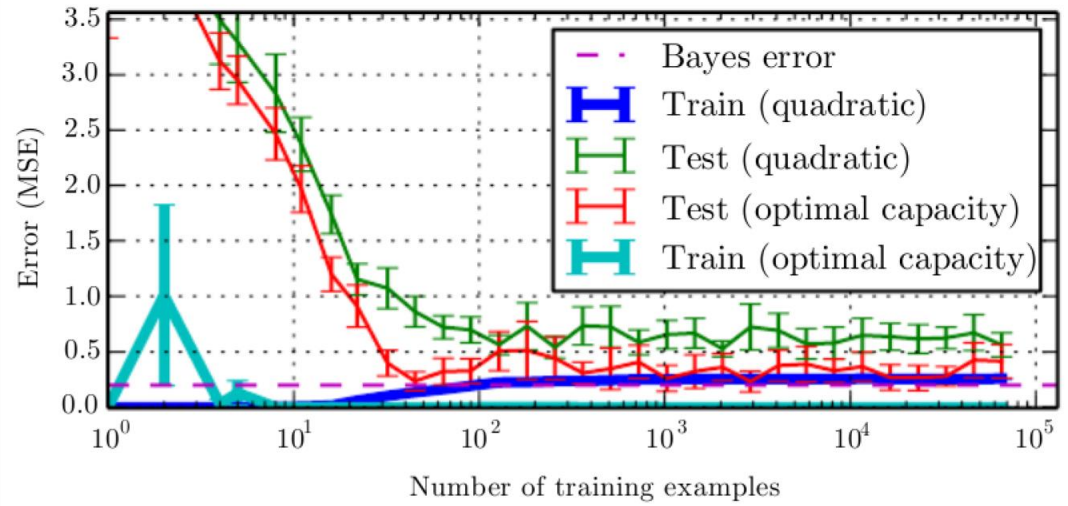


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the *underfitting regime*. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the *overfitting regime*, where capacity is too large, above the *optimal capacity*.

Capacity, Overfitting and Underfitting: Capacity



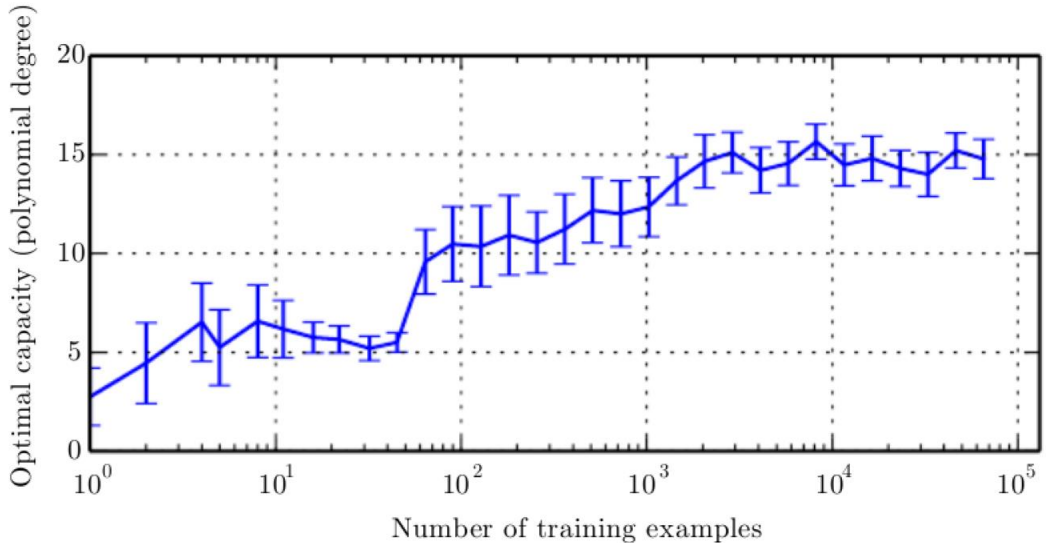
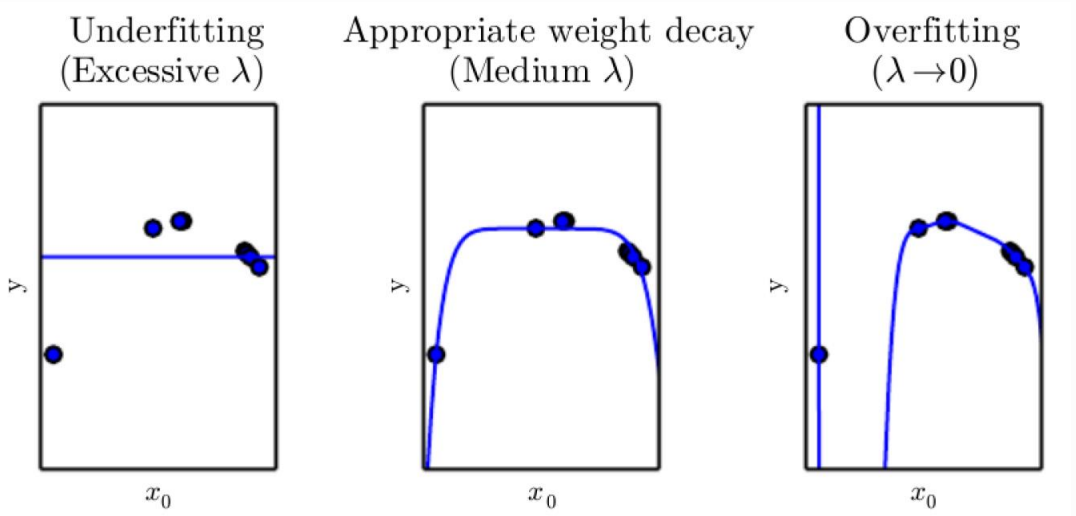


Figure 5.4: The effect of the training dataset size on the train and test error, as well as on the optimal model capacity.

Capacity, Overfitting and Underfitting: Regularization

- Regularization is any modification we make to the learning algorithm to reduce generalization error.
  - The most common form of regularization is to modify the loss function so that larger parameters are penalized

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}}(\boldsymbol{w}) + \lambda \boldsymbol{w}^{\top}\boldsymbol{w}$$



  - There are other forms of regularization, for instance ‘dropout‘ (randomly dropping some of the weights in your model) is very popular in deep learning

Hyperparameters and Validation Sets

- Hyperparameters are the parameters of our model that controls the capacity, as well as the behaviour of the algorithm
  - Examples: degree of a polynomial, number of layers in a neural net, structure of a PGM...
  - These are usually set by domain experts
- Can we 'learn' those parameters as well?
  - We can try optimizing them, this is called *hyperparameter optimization*
- However, using the whole training data does not make sense
  - The optimization procedure would simply select the parameters that result in highest capacity (i.e. overfitting)
- Similar to how we separated the whole data into test and training, we can further divide the training data into two disjoint subsets
  - The smaller part that is used for learning the hyperparameters is called the validation set

Estimators, Bias and Variance: Estimation

- How can we quantify the concept of overfitting/underfitting?
  - Classical statistics and estimation theory can help with that
  - We will take the frequentist view for the rest of this section
- Point estimation is an attempt to provide a single best estimate for a quantity of interest. We denote the point estimate of $\boldsymbol{\theta}$ as $\widehat{\boldsymbol{\theta}}$
- Let $\{x^{(1)}, \ldots, x^{(m)}\}$ be a set of m i.i.d. data points. A point estimator or statistic is any function of the data:

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$$

  - $\{x^{(1)}, \ldots, x^{(m)}\}$ is generated randomly, hence $\widehat{\boldsymbol{\theta}}_m$ is a random variable, even though $\boldsymbol{\theta}$ is not
  - We are interested in analyzing the relationship between $\widehat{\boldsymbol{\theta}}_m$ and $\boldsymbol{\theta}$, both for finite m and $m \rightarrow \infty$

Estimators, Bias and Variance: Bias

- The bias of an estimator is defined as:

$$\text{Bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta}$$

  - The expectation is taken over the data
  - An estimator is *unbiased* if Bias($\widehat{\boldsymbol{\theta}}_m$) = 0 and *asymptotically unbiased* if $\lim_{m\to\infty}$ Bias ($\widehat{\boldsymbol{\theta}}_m$) = 0

- Let $x^{(i)}$ be generated from a Bernoulli distribution with mean θ. Then the estimator $\hat{\theta}_m = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ is unbiased
  - The same estimator is also unbiased when $x^{(i)}$ are generated from a Gaussian distribution with mean μ
  - However, the estimator $\hat{\sigma}_m = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)$ is only asymptotically unbiased. We can make it unbiased by setting $\hat{\sigma}_m = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)$

- It might seem like unbiased estimators are preferable, but this is not necessarily true, we will soon see that biased estimators possess some interesting properties

Estimators, Bias and Variance: Variance and Standard Error

- Another interesting property of an estimator is how much it varies as a function of the data sample. This is called *variance*

$$\text{Var}(\hat{\theta}_m) = E\left[(\hat{\theta}_m - E[\hat{\theta}_m])^2\right]$$

  - Low variance estimators give similar outputs irregardless of the sampled data. Variance is an excellent measure for quantifying the uncertainty in our estimations.
  - For instance, for Gaussian distributions we can say that our estimate of the mean $\hat{\mu}$ falls into the interval of

$$\left(\hat{\mu}_m - 1.96\sqrt{\text{Var}(\hat{\theta}_m)}, \hat{\mu}_m + 1.96\sqrt{\text{Var}(\hat{\theta}_m)}\right)$$ with 95% probability.

  - Hence ideally, we want both low bias and low variance.

- For Bernoulli distribution, variance of the estimator is $\frac{1}{m}\theta(1-\theta)$
  - The variance decreases as m increases, this is a common property of popular estimators

Estimators, Bias and Variance: Bias-Variance Decomposition

- Bias and variance are two difference sources of error in an estimator
  - Bias measures the expected deviation from the true value.
  - Variance measures deviation from the expected estimator value among different subsets of data.

- Should we choose a model with low bias or low variance?
  - The answer is given by the key idea that mean squared error can be decomposed as follows

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2] = \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

  - Hence for a fixed MSE, there is a trade-off between variance and bias
  - In general, increasing capacity increases the variance and decreases the bias.

Estimators, Bias and Variance: Bias-Variance Trade-off

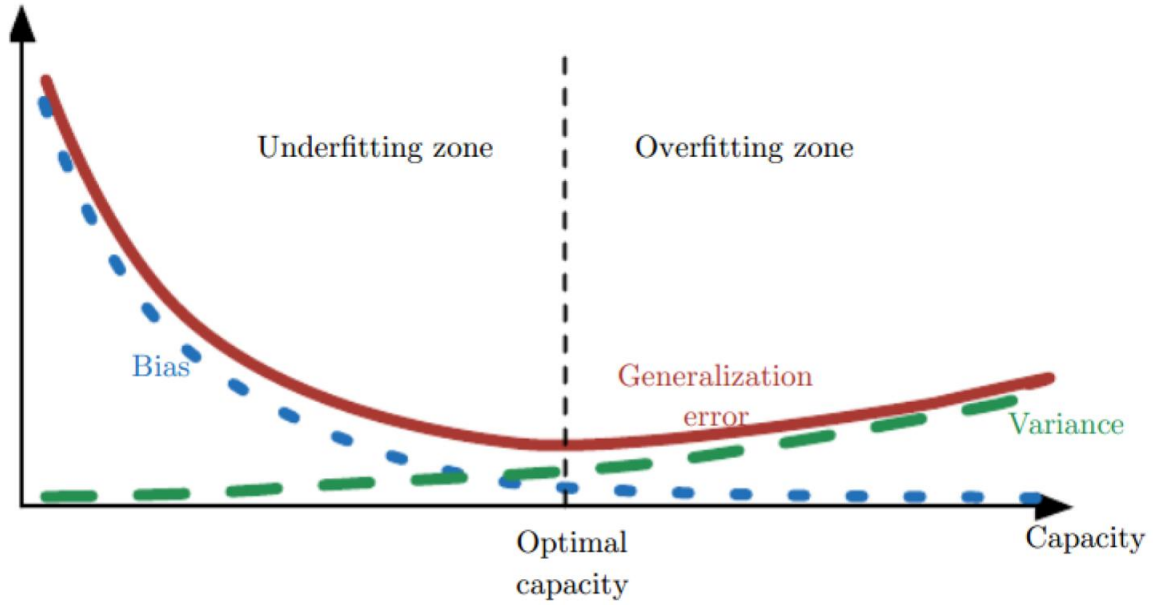


Figure 5.6: As capacity increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). If we vary capacity along one axis, there is an optimal capacity, with underfitting when the capacity is below this optimum and overfitting when it is above. This relationship is similar to the relationship between capacity, underfitting, and overfitting, discussed in Sec. 5.2 and Fig. 5.3.

Supervised Learning Algorithms: Probabilistic Supervised Learning

- The supervised learning is mainly about estimating the probability distribution $p(y \mid \boldsymbol{x})$
  - To use MLE for this job, we first define a family of probability distributions $p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$
- Linear regression problem can be expressed by the family:

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{\theta}^\top \boldsymbol{x}, \boldsymbol{I})$$

  - It can be shown that in this case θ_{ML} has a closed form expression, which is the same as least squares solution!

- For binary classification, we can use the logistic sigmoid function:

$$p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x})$$

  - No closed form solution in this case, we need to use numerical optimization methods.

Supervised Learning Algorithms: Parametric Supervised Learning

- Not all supervised learning algorithms are probabilistic. We can simply define a parametric family of deterministic models $y = f(\boldsymbol{x}; \boldsymbol{w})$ and use some loss function to find an optimal $\boldsymbol{w}^*$
- One of the most famous approaches to this problem is the Support Vector Machines (SVMs)
  - For binary classification, we let $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$
  - If $f(\boldsymbol{x}) > 0$ we predict that sample belongs to class 1 and vice versa
- SVM is useful for two reasons
  - Solving for optimal $\boldsymbol{w}$ is a convex optimization problem
  - It can be shown that:

$$\boldsymbol{w}^\top \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\top \boldsymbol{x}^{(i)}$$

- The second part is the key results. We can replace $\boldsymbol{x}^T \boldsymbol{x}^{(i)}$ by any inner product and we can still use the same algorithm!
  - This is how we generalize to nonlinear classifiers: $b + \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$
  - The k is called a $kernel$

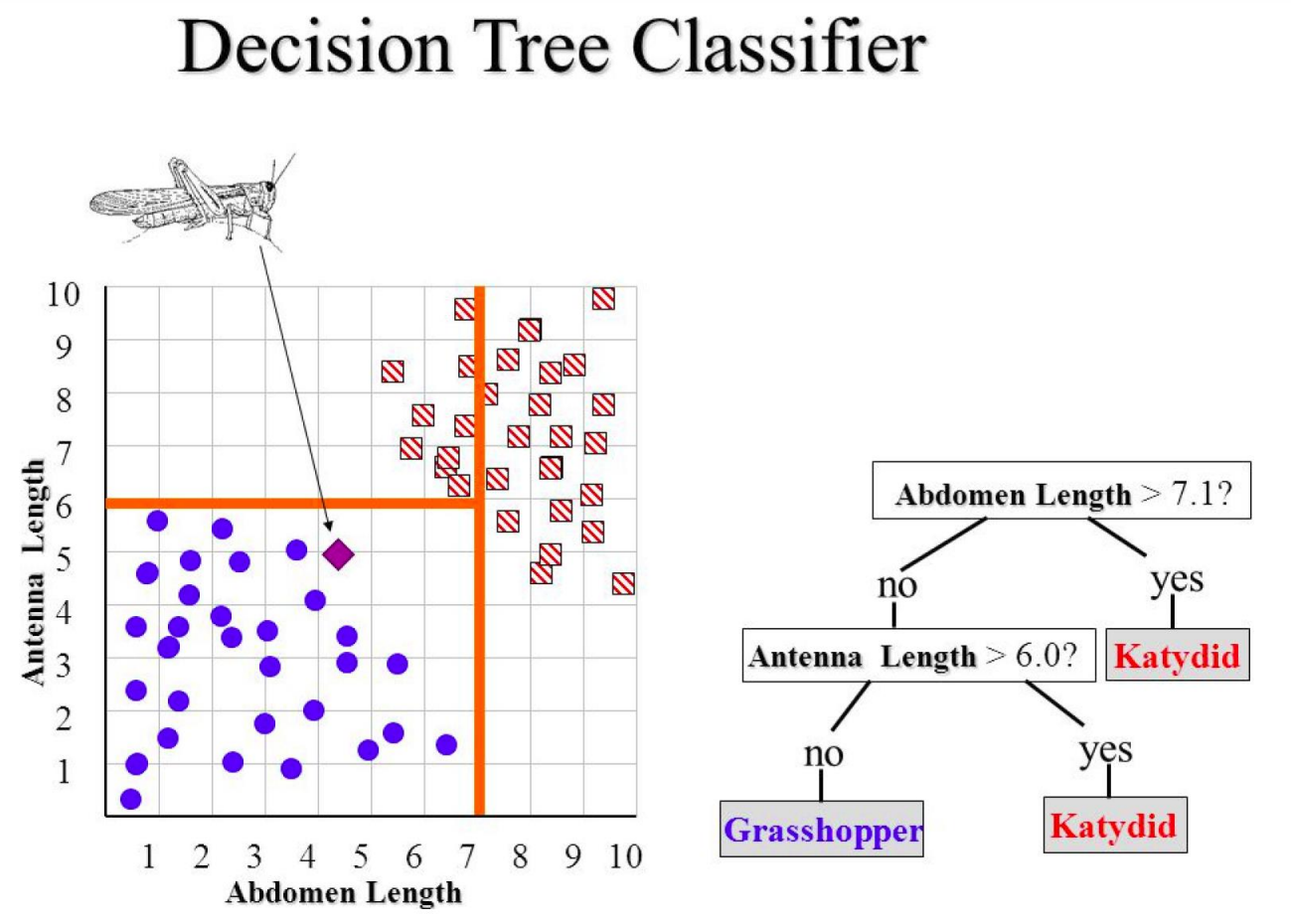
Supervised Learning Algorithms: Nonparametric Supervised Learning

- For most supervised learning algorithms we need to fix the form and number of parameters in the model beforehand.
  - Can we also learn those from the model?
  - Models that allow these are called *nonparametric*, in some sense, they let the data speak for itself

- One of the simplest nonparametric classifier is *k*-nearest neighborhood method
  - Fix a positive integer *k* (yes even nonparametric models have parameters)
  - For a new point, find the nearest *k* samples from the dataset, compute their class/output averages and simply predict this value for the new point

- There are other popular nonparametric classifiers as well, decision trees, random forests etc.

Supervised Learning Algorithms: Nonparametric Supervised Learning



Unsupervised Learning

- There are unsupervised learning algorithms for dimensionality reduction, e.g. PCA
  - PCA takes unlabeled and difficult to interpret data and transforms it into a much more manageable form

- Another popular form of unsupervised learning is clustering
  - How can we segment the data in a meaningful way?
  - k-means clustering is a very popular algorithm for this job.

- Generating association rules is another task that can be solved via unsupervised learning
  - An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database.

References

- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

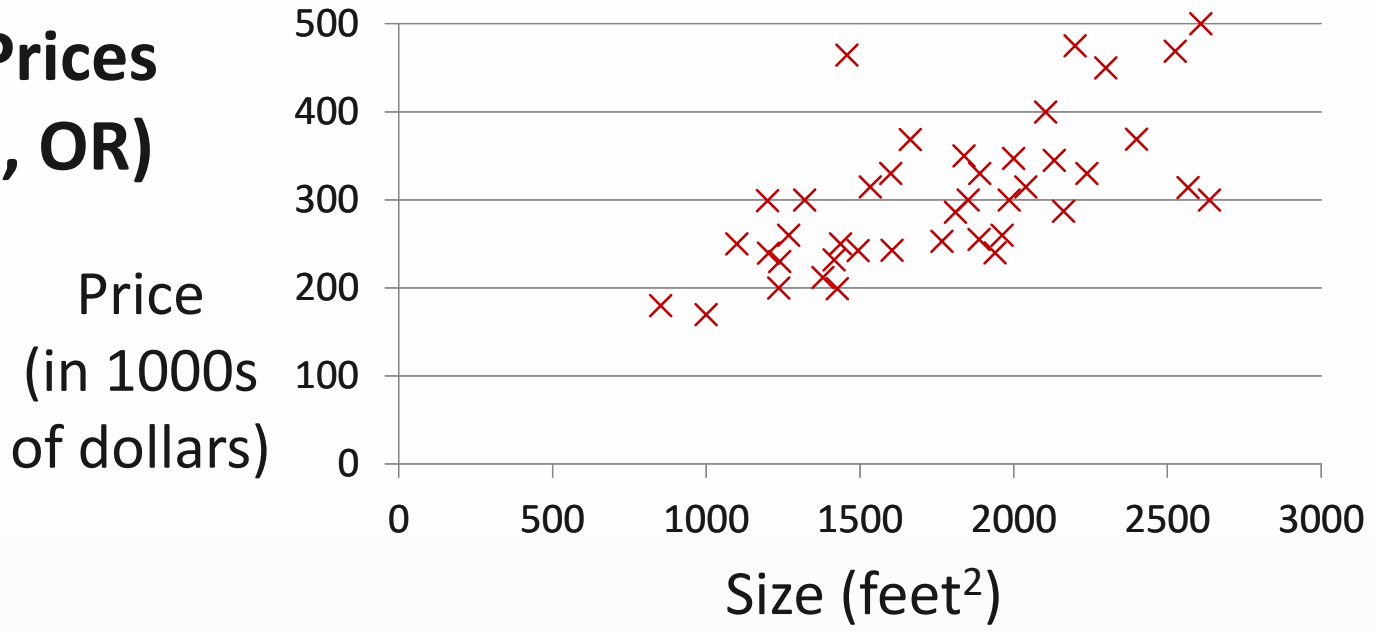
LECTURE 3

LINEAR AND POLYNOMIAL REGRESSION

Instructor: Asst. Prof. Barış Başpınar

Model Representation

Housing Prices (Portland, OR)



Supervised Learning

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

Regression Problem

Predict real-valued output

Model Representation

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

m = 50

Notation:

m = Number of training examples

x's = "input" variable / features

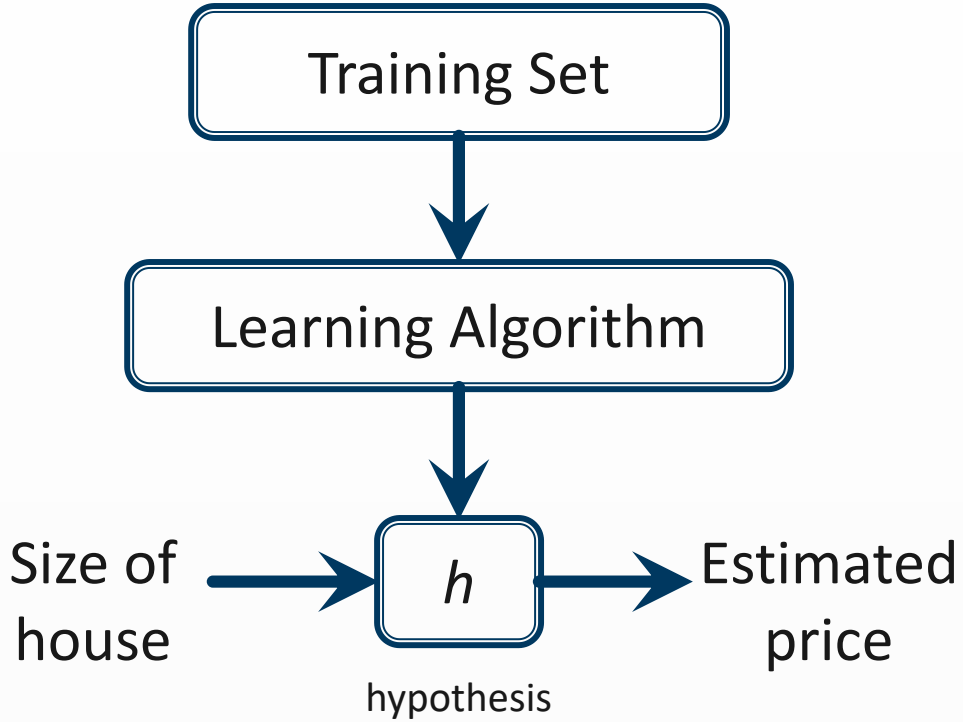
y's = "output" variable / "target" variable

$(x^{(1)}, y^{(1)}) = (2104, 460)$

$(x^{(2)}, y^{(2)}) = (1416, 232)$

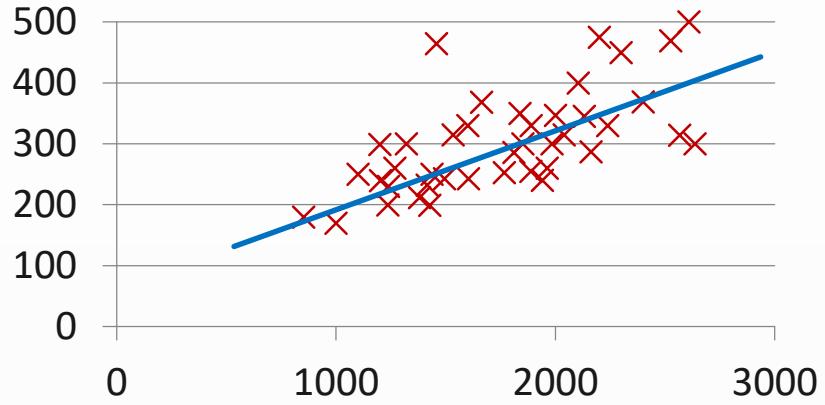
$(x^{(i)}, y^{(i)}) \rightarrow i^{th}$ training example

Model Representation: linear regression



How do we represent *h* ?

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



Linear regression with one variable.
Univariate linear regression.

Cost Function

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

m = 50

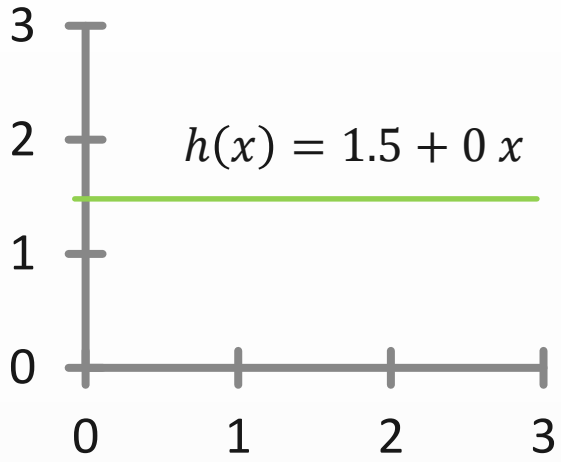
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

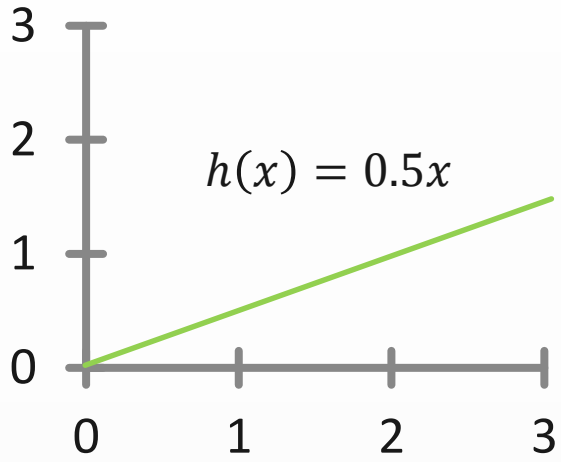
How to choose θ_i's ?

Cost Function

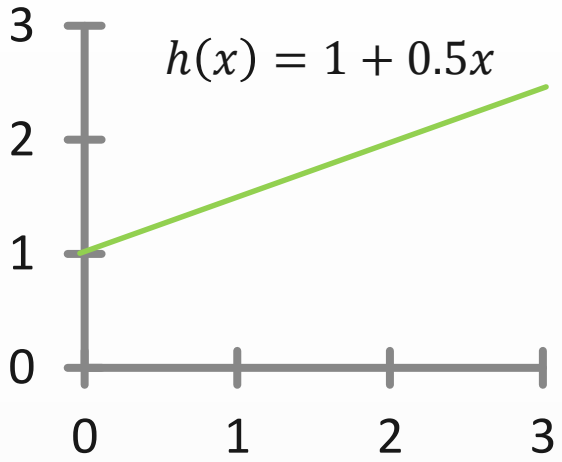
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$

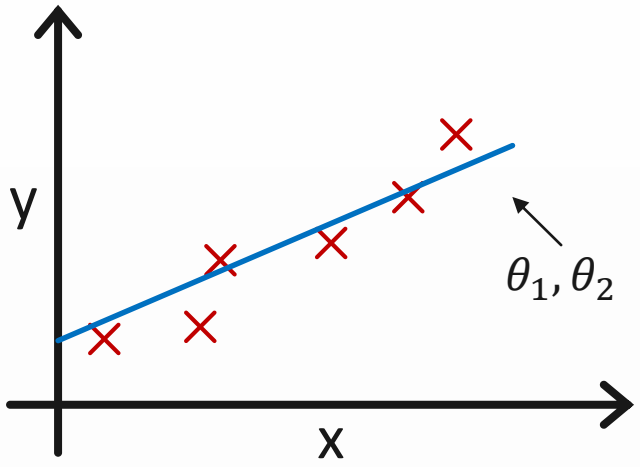


$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$

Cost Function



Number of training examples

Squared error function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Cost function

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

$$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Cost Function Intuition

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

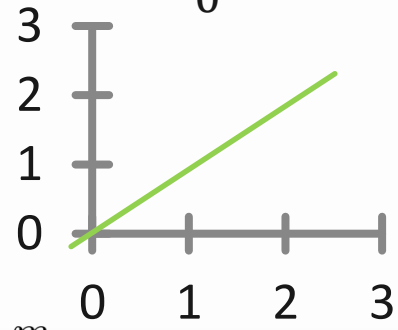
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Simplified

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

$\theta_0 = 0$

θ_1



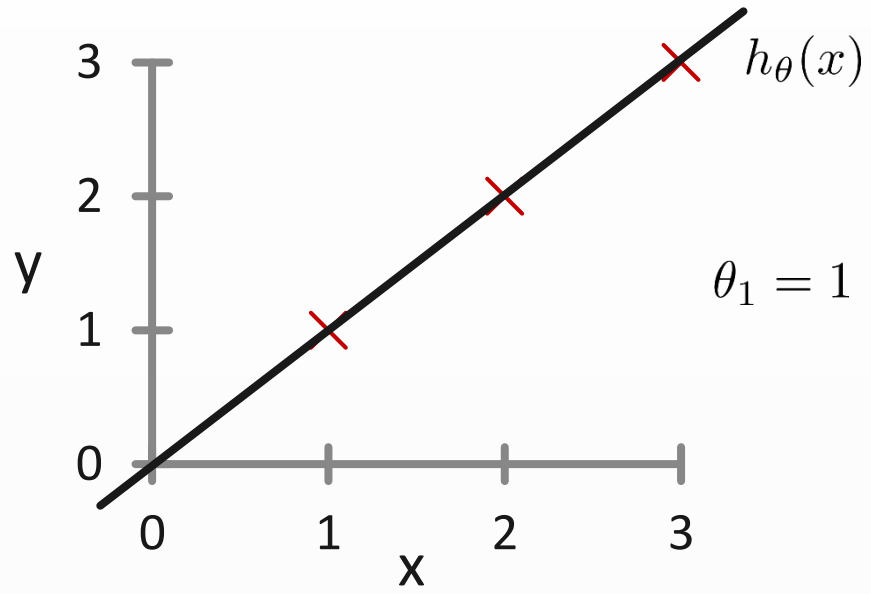
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

Cost Function Intuition

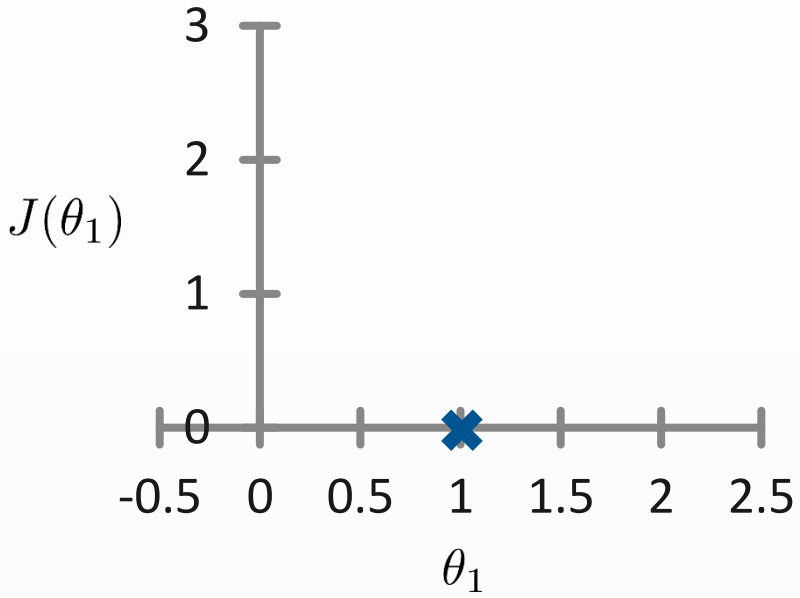
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



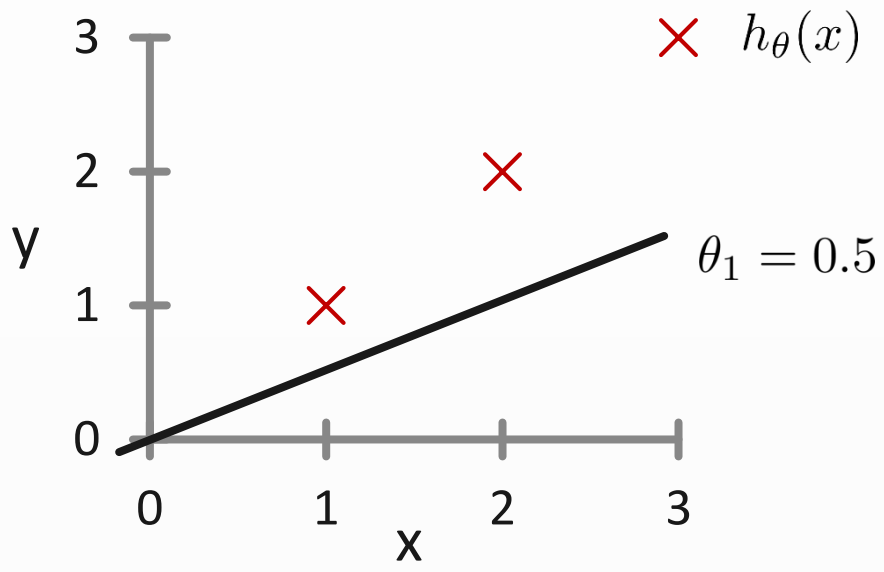
$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_1 x^{(i)} - y^{(i)}\right)^2 \longrightarrow J(1) = \frac{1}{2m}(0+0+0)^2 = 0$$

Cost Function Intuition

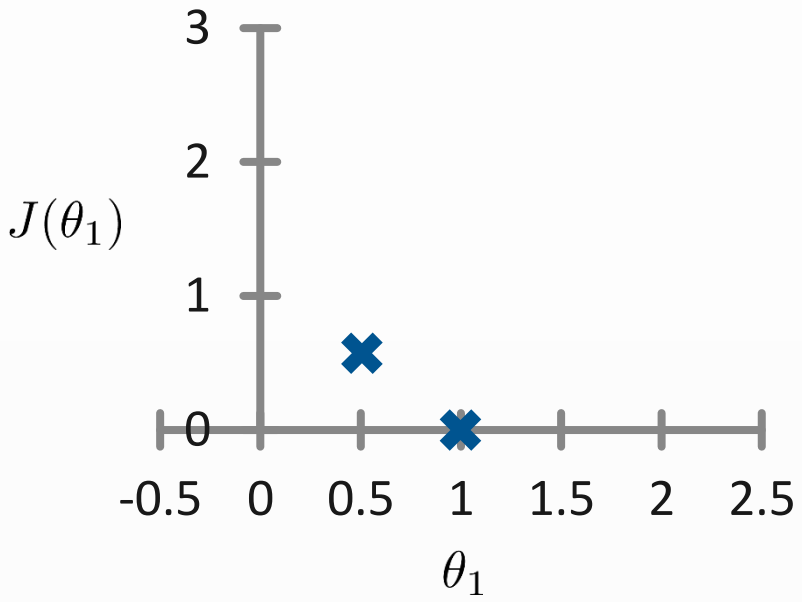
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)

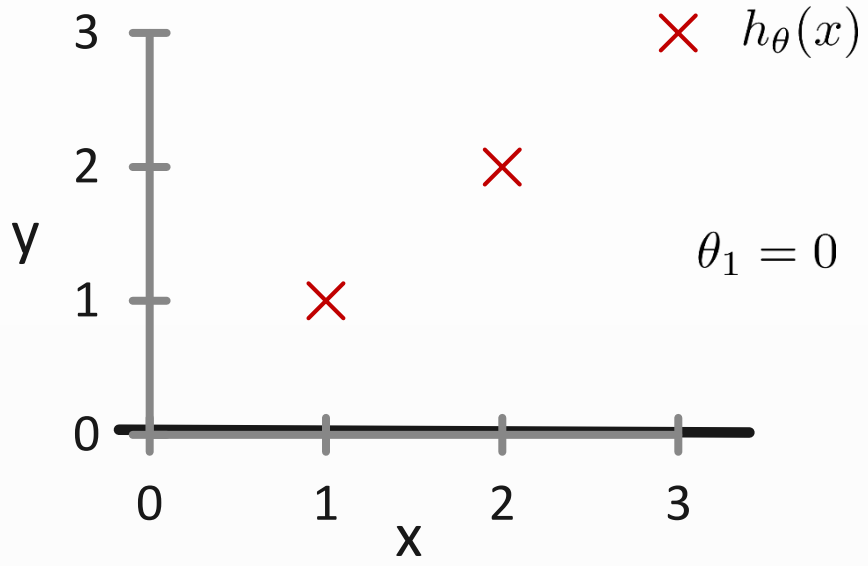


$$J(0.5) = \frac{1}{2 \times 3}[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2] = \frac{3.5}{6} = 0.58$$

Cost Function Intuition

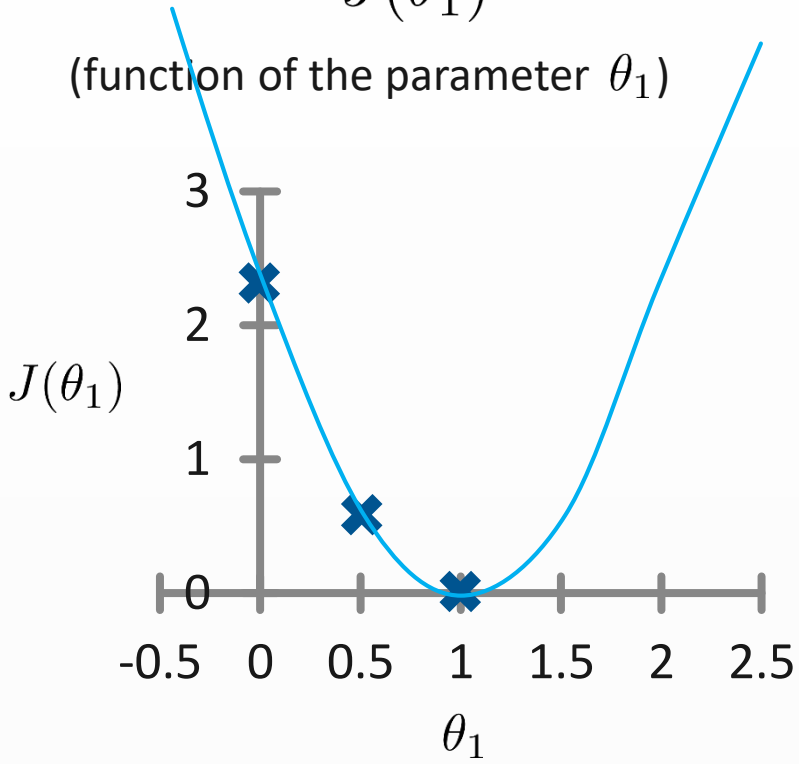
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



$\min_{\theta_1} J(\theta_1)$

$$J(0) = \frac{1}{2 \times 3}[(0-1)^2 + (0-2)^2 + (0-3)^2] = \frac{14}{6} = 2.33$$

Cost Function Intuition

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

Parameters: θ_0, θ_1

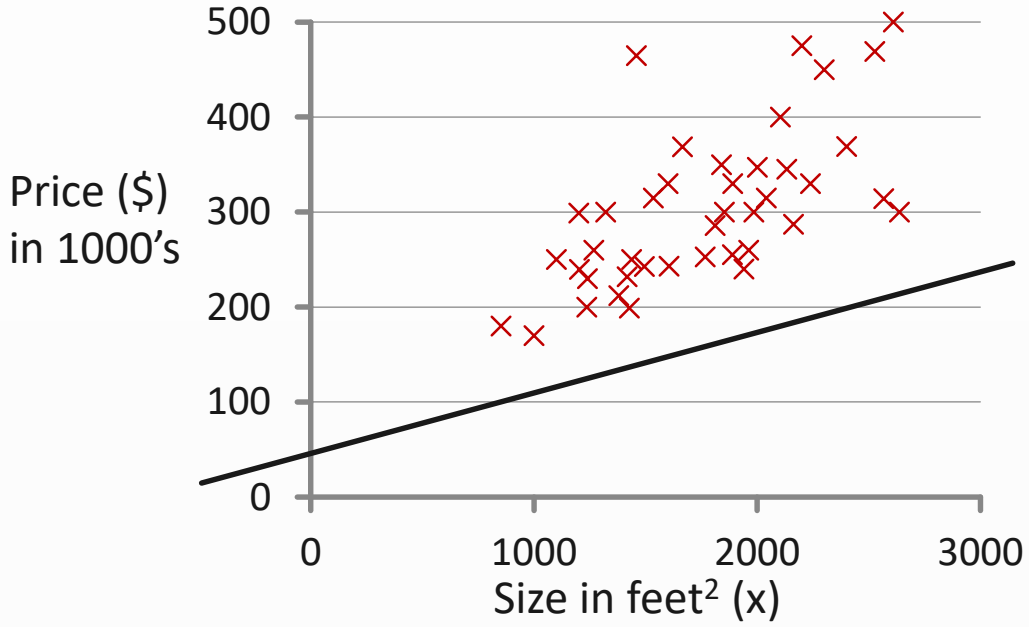
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost Function Intuition

$h_\theta(x)$

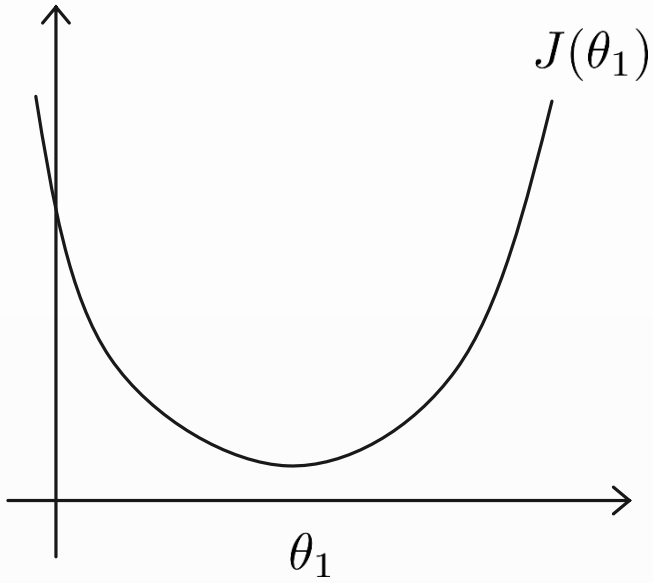
(for fixed θ_0, θ_1, this is a function of x)



$$h_\theta(x) = 50 + 0.06x$$

$J(\theta_0, \theta_1)$

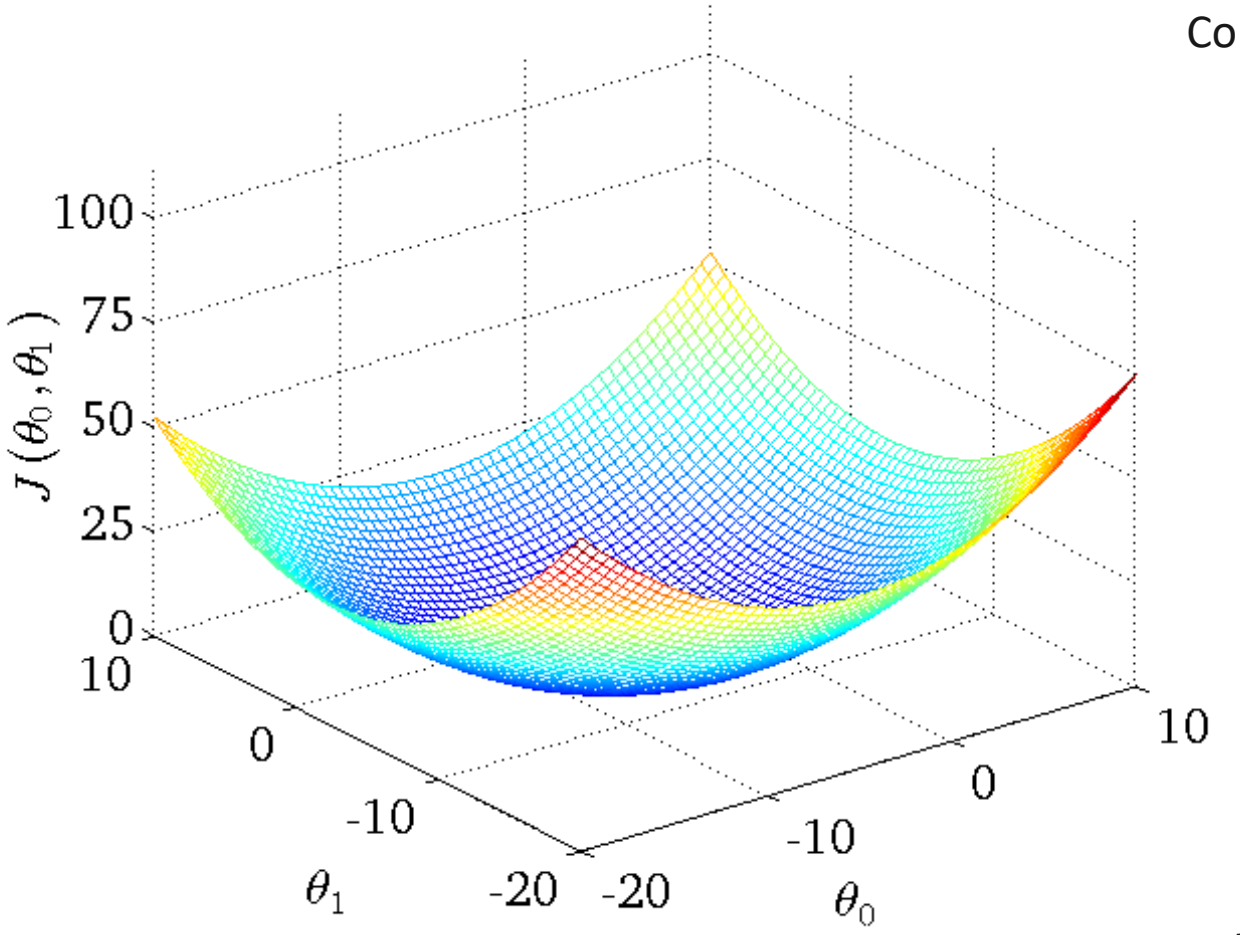
(function of the parameters θ_0, θ_1)



θ_0, θ_1

Cost Function Intuition

Contour plots

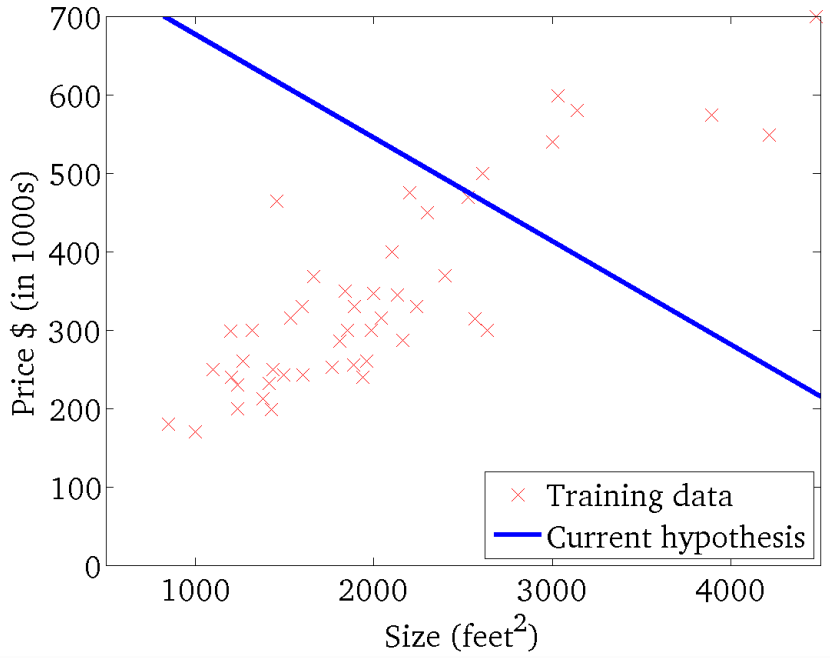


$J(\theta_0, \theta_1)$

Cost Function Intuition

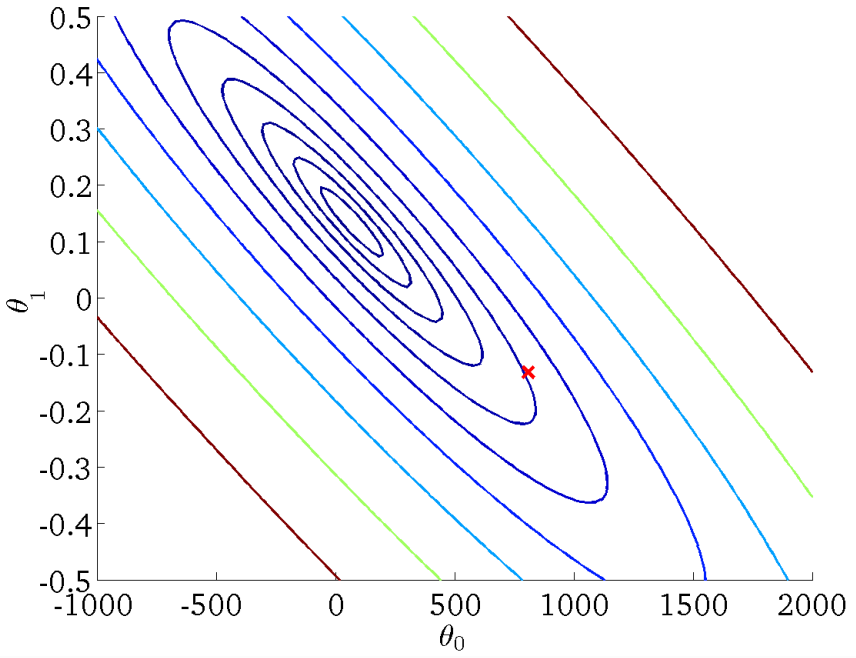
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

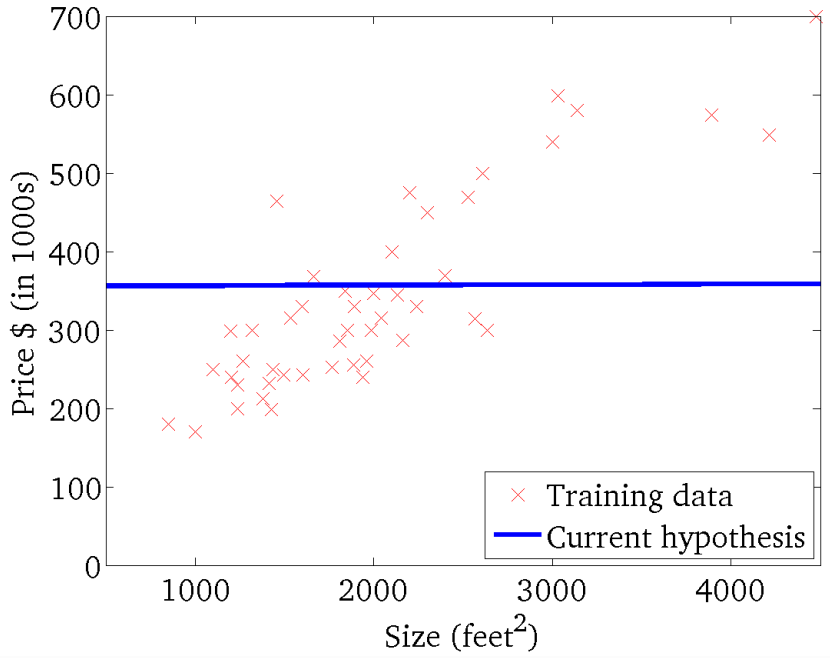
(function of the parameters θ_0, θ_1)



Cost Function Intuition

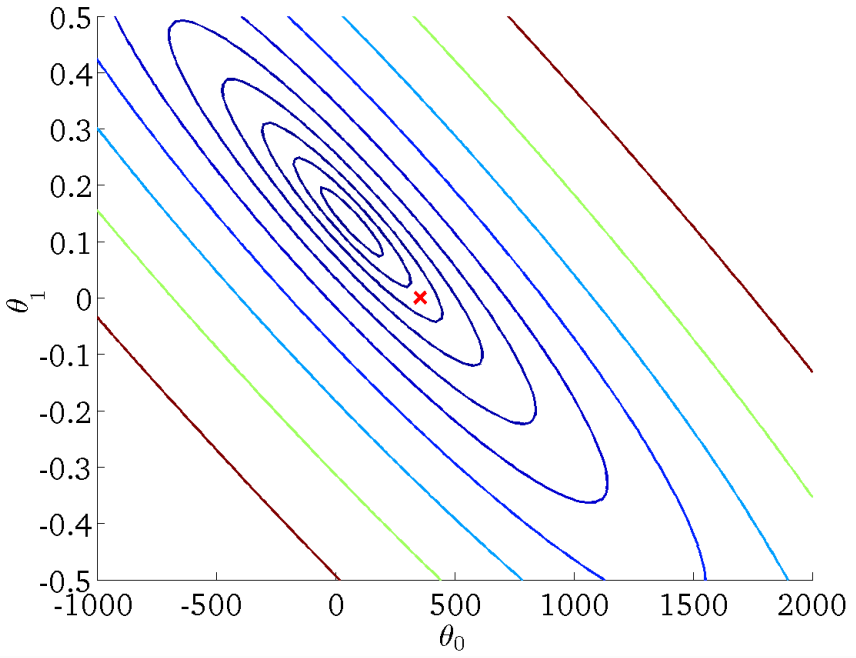
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent

Have some function $J(\theta_0, \theta_1)$

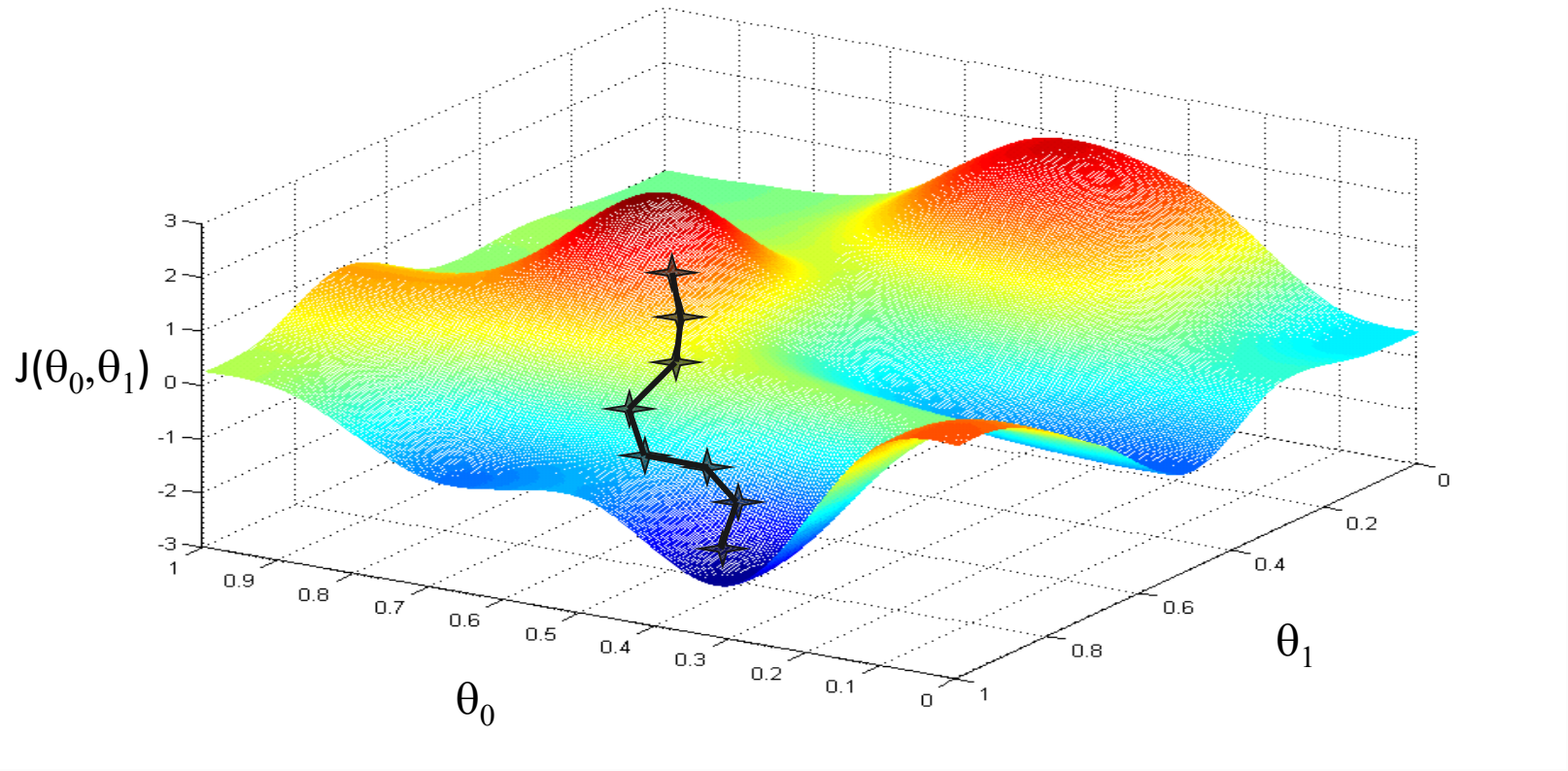
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

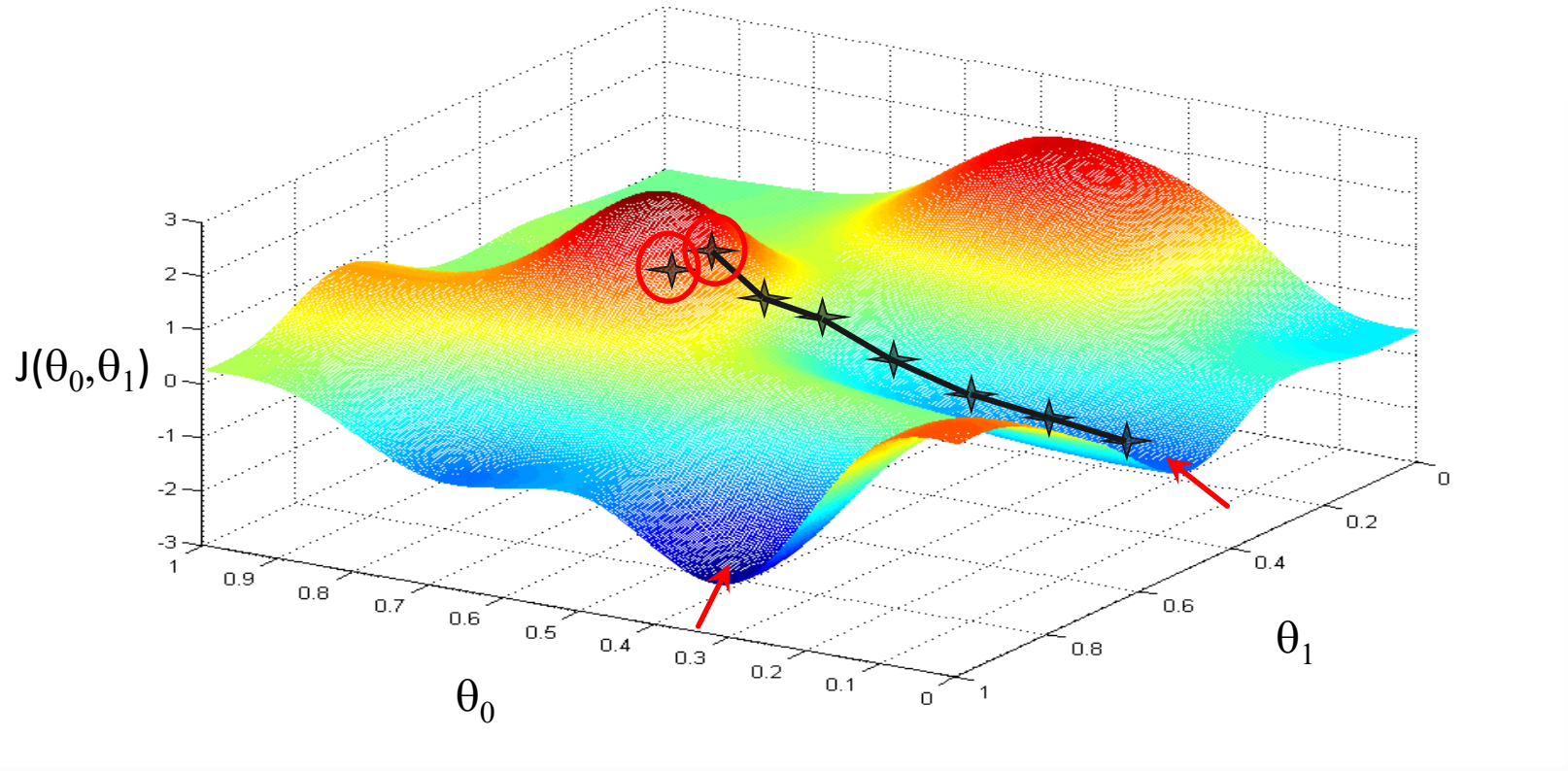
- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum

Gradient Descent



Gradient Descent



Gradient Descent Algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

learning rate

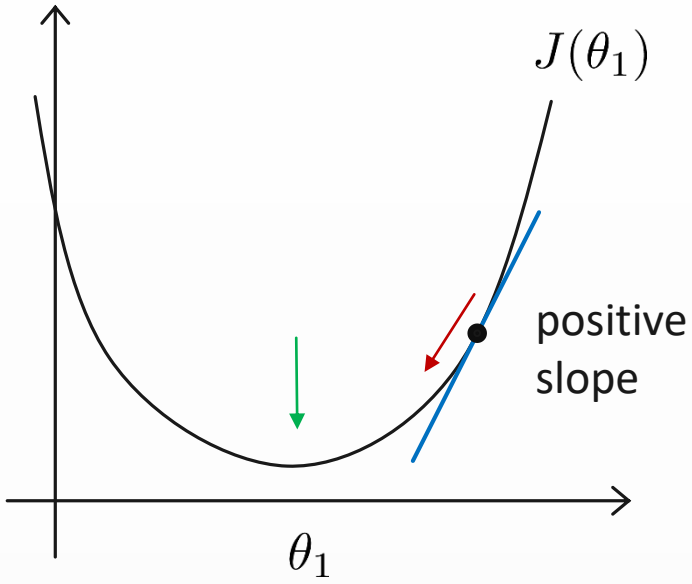
Correct: Simultaneous update

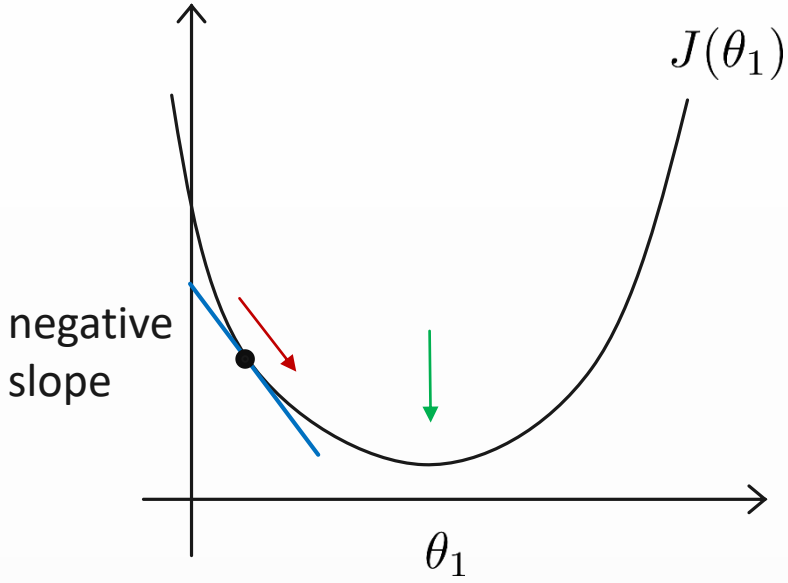
$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient Descent Intuition





$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)} \geq 0$$

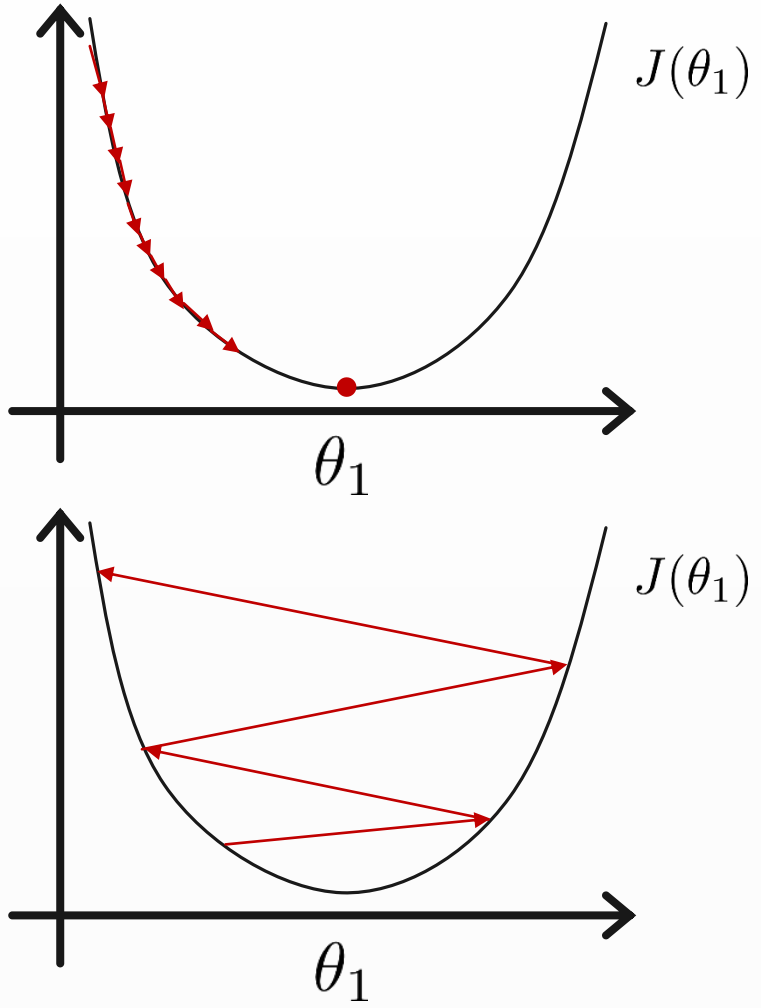
$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)} \leq 0$$

Gradient Descent Intuition

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

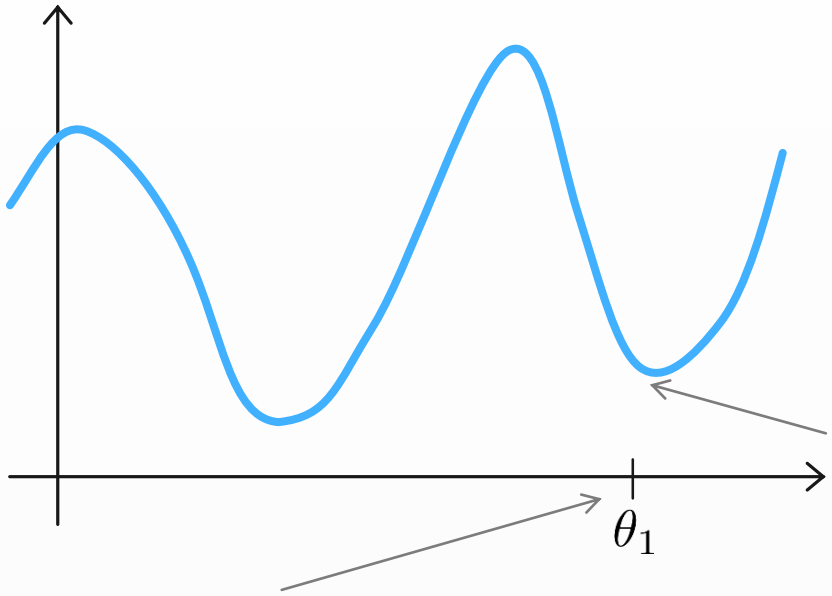
If α is too small, gradient descent can be slow.

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



Gradient Descent Intuition

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



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

- As we approach a local minimum, gradient descent will automatically take smaller steps

Gradient Descent for Linear Regression

Gradient descent algorithm

$$\text{repeat until convergence } \{$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

$$\}$$

Linear Regression Model

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

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Gradient Descent for Linear Regression

In order to implement this algorithm, we need to calculate the partial derivatives:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right) x^{(i)} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x^{(i)}$$

Gradient Descent for Linear Regression

$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

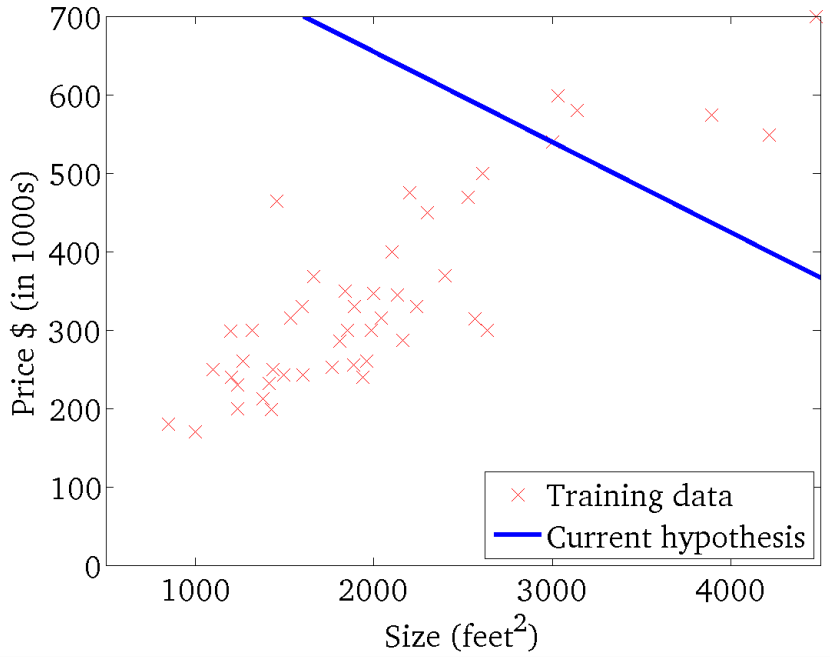
update θ_0 and θ_1 simultaneously

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

Gradient Descent for Linear Regression

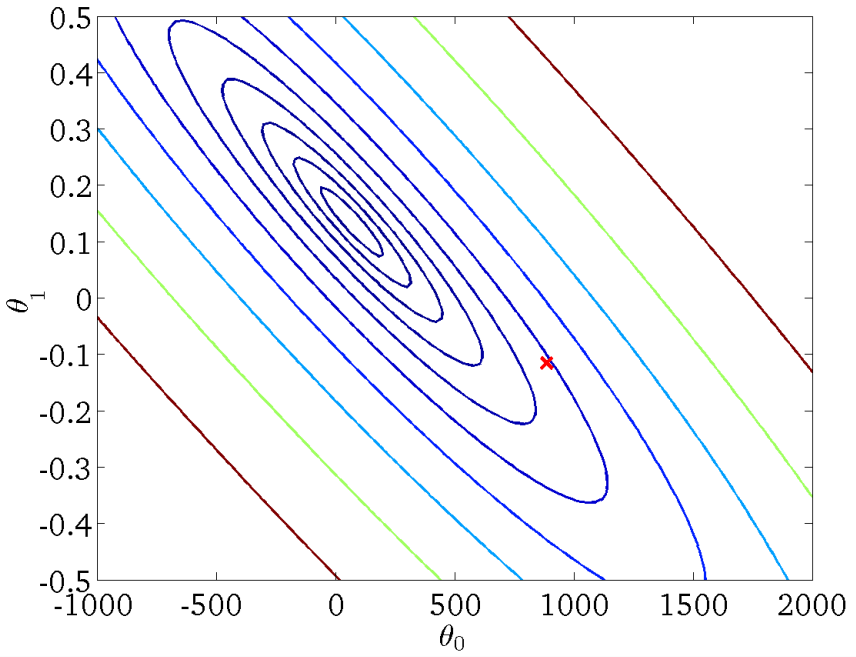
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

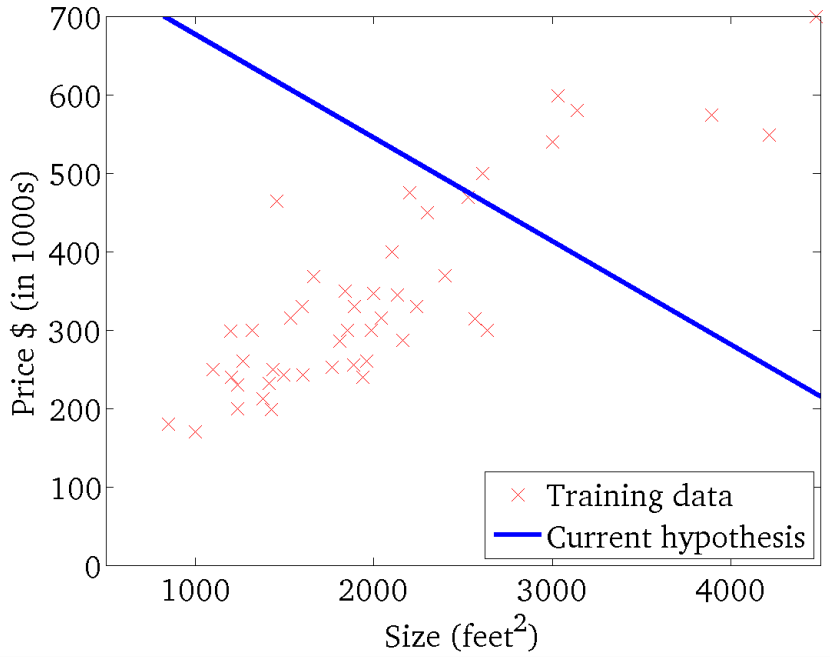
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

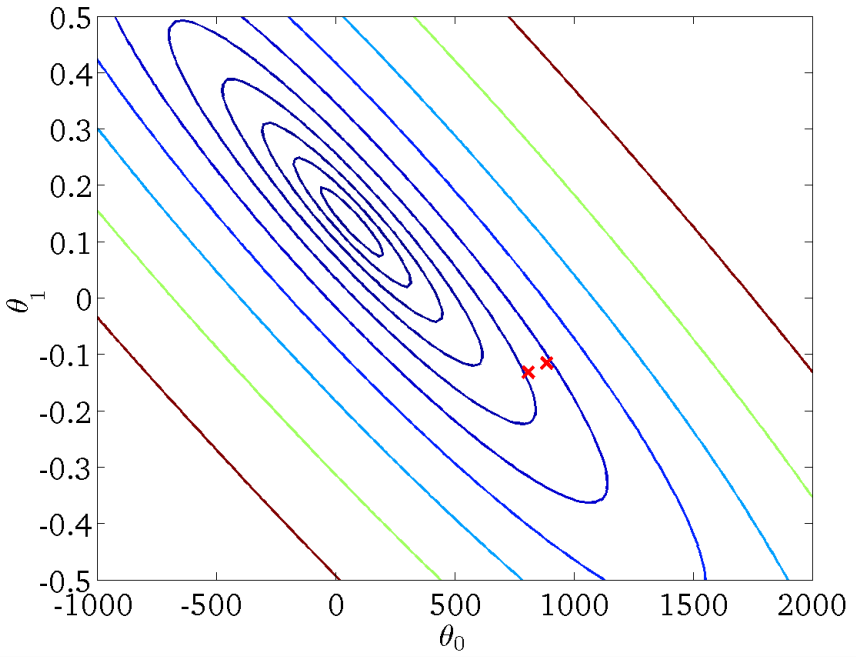
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

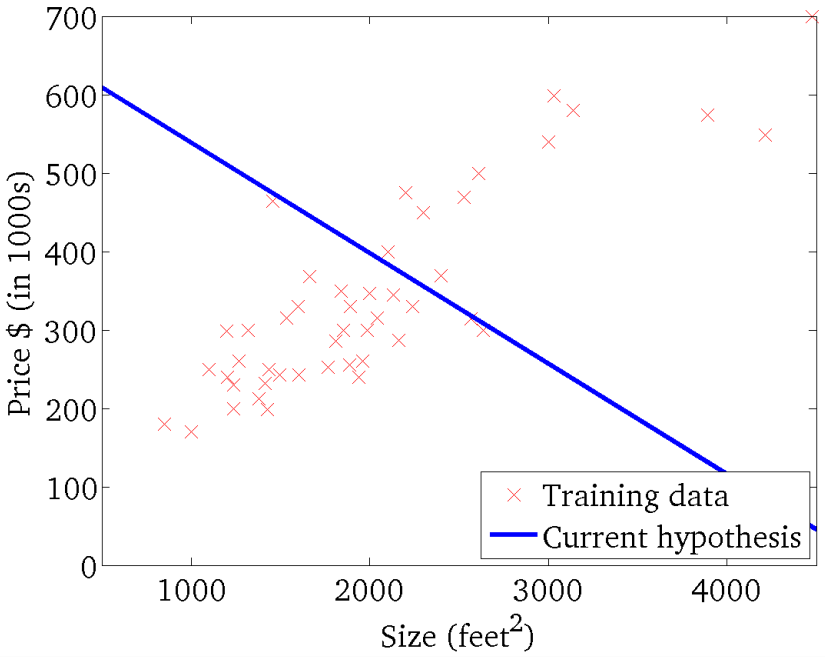
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

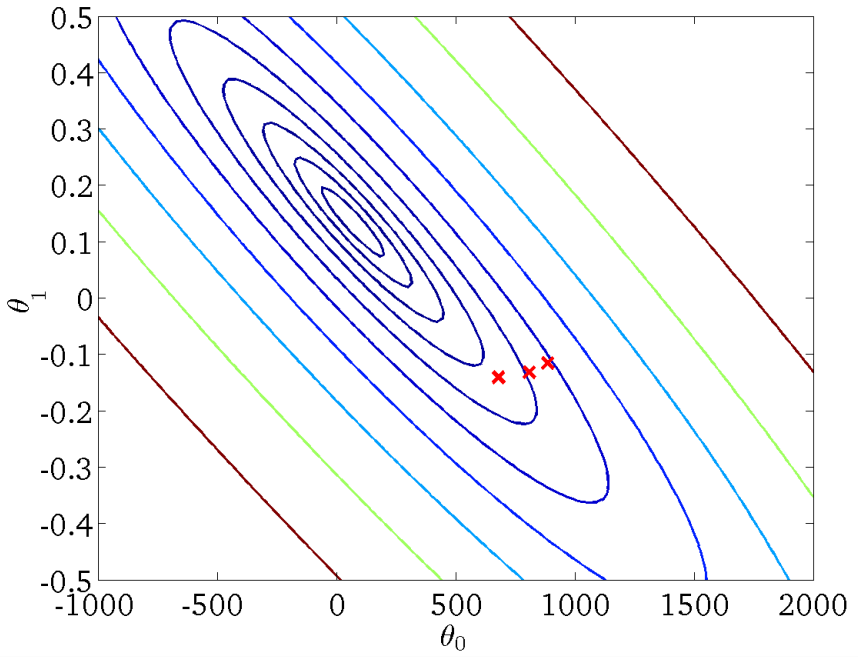
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

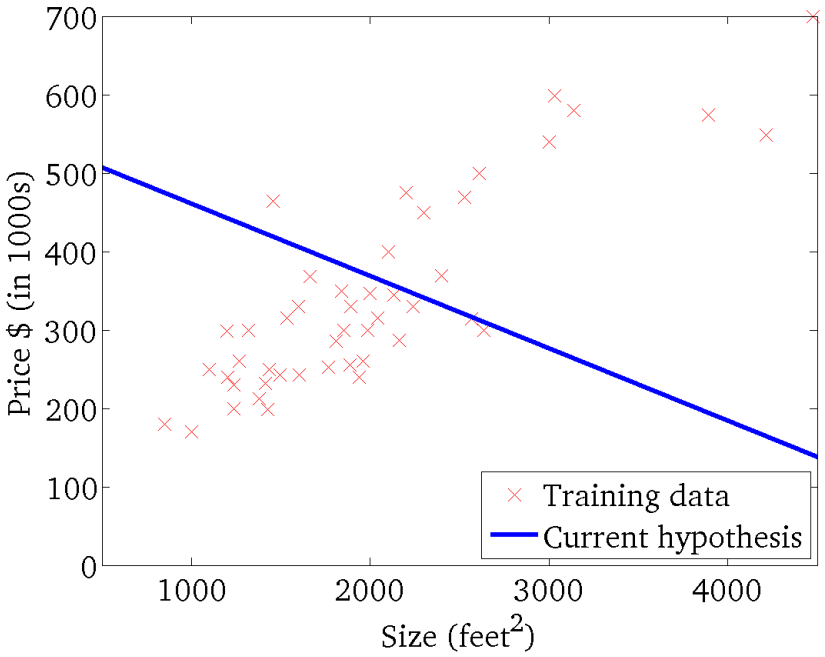
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

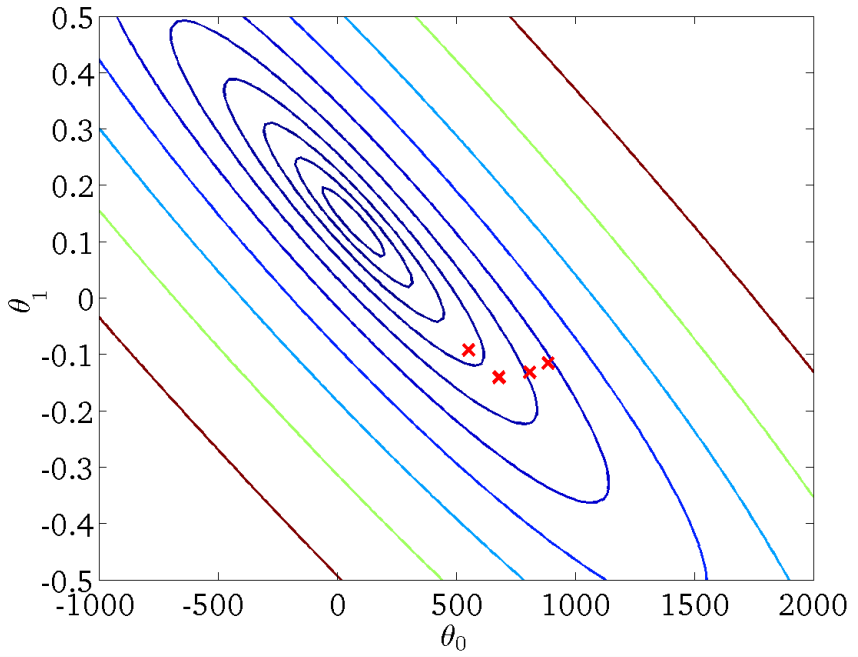
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

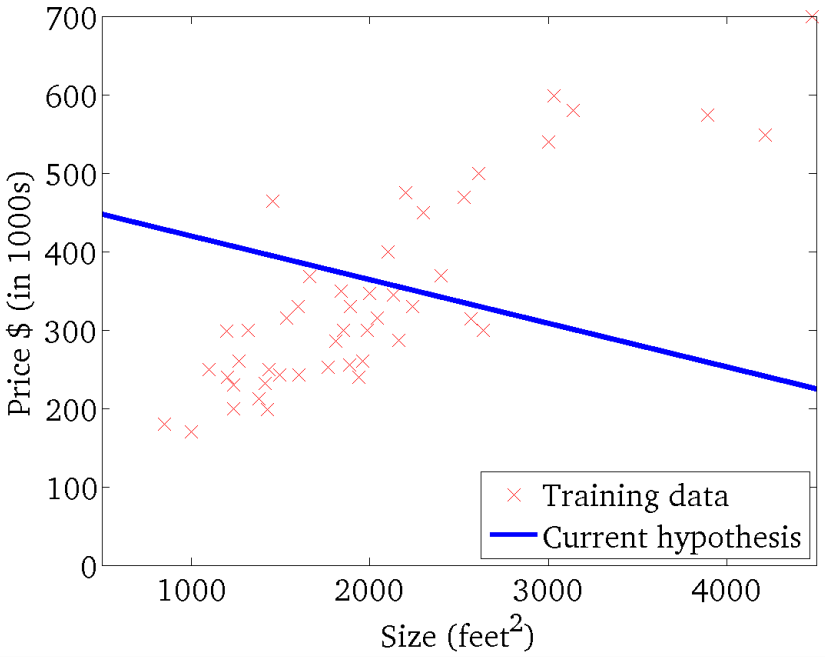
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

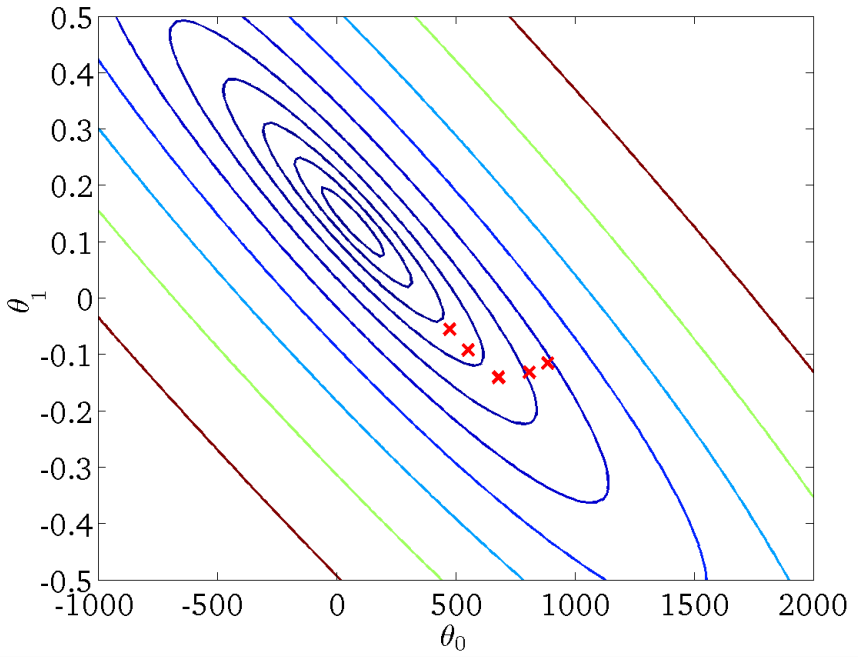
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

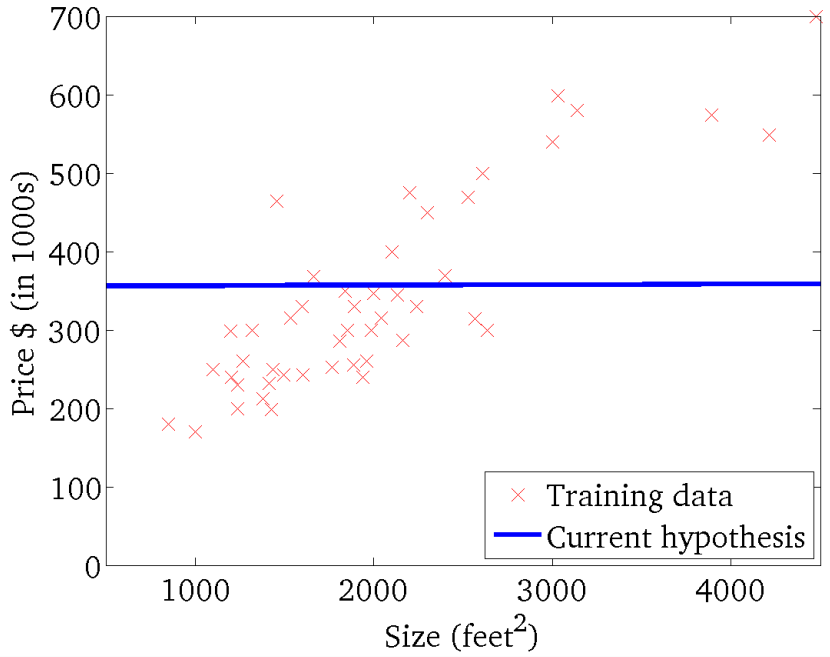
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

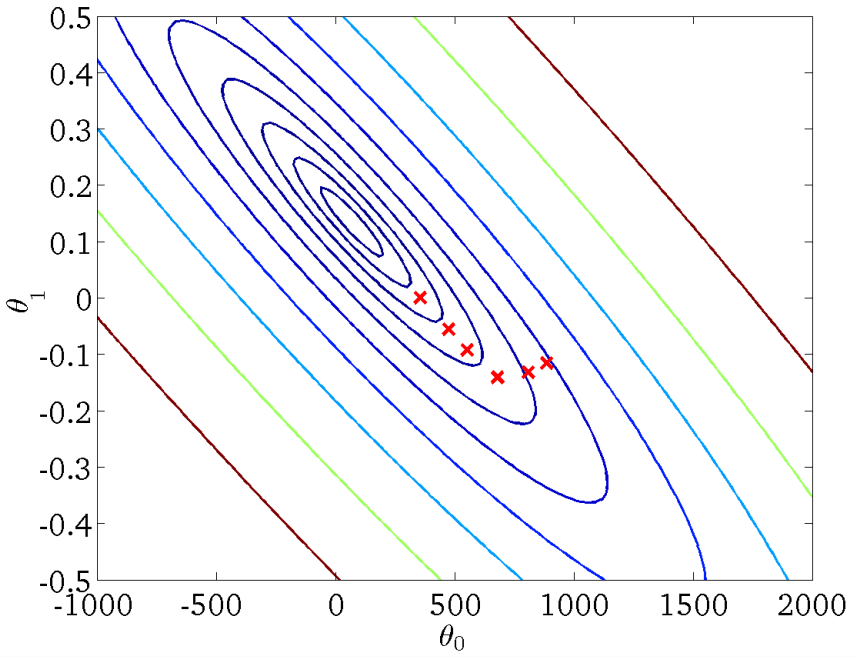
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

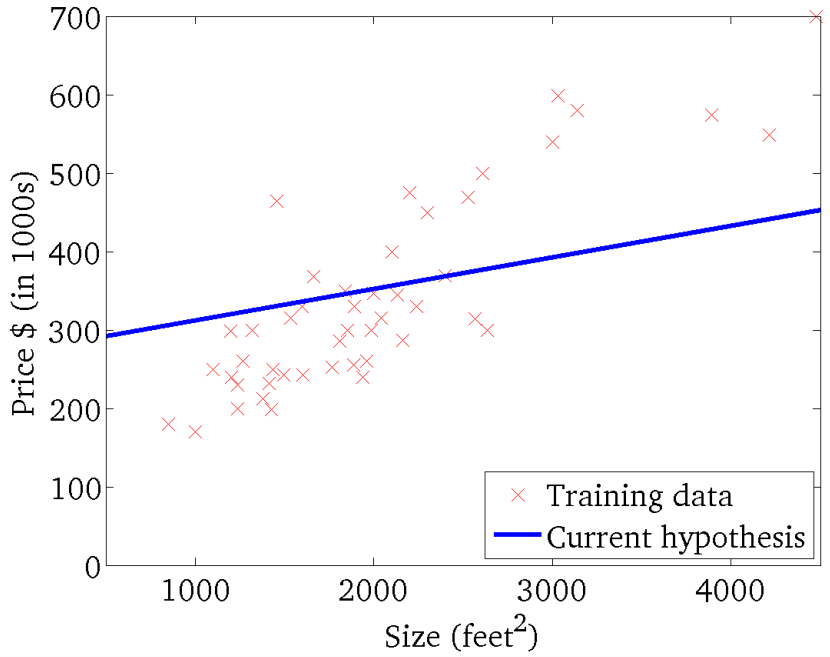
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

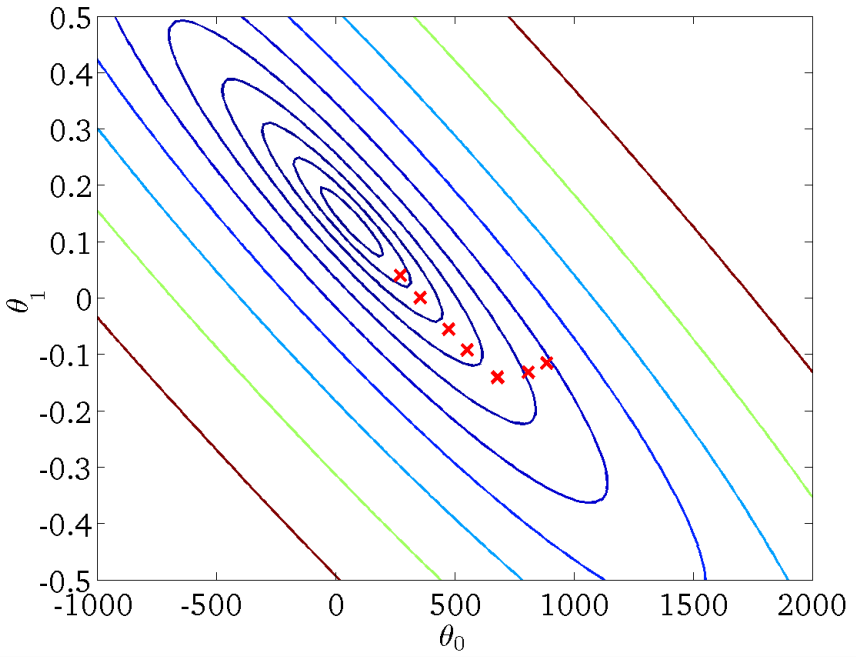
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

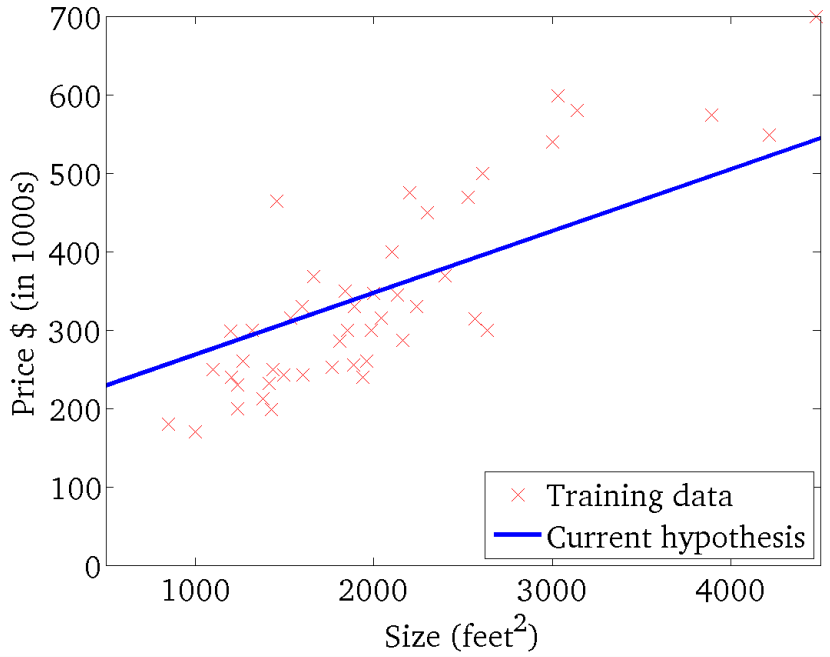
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

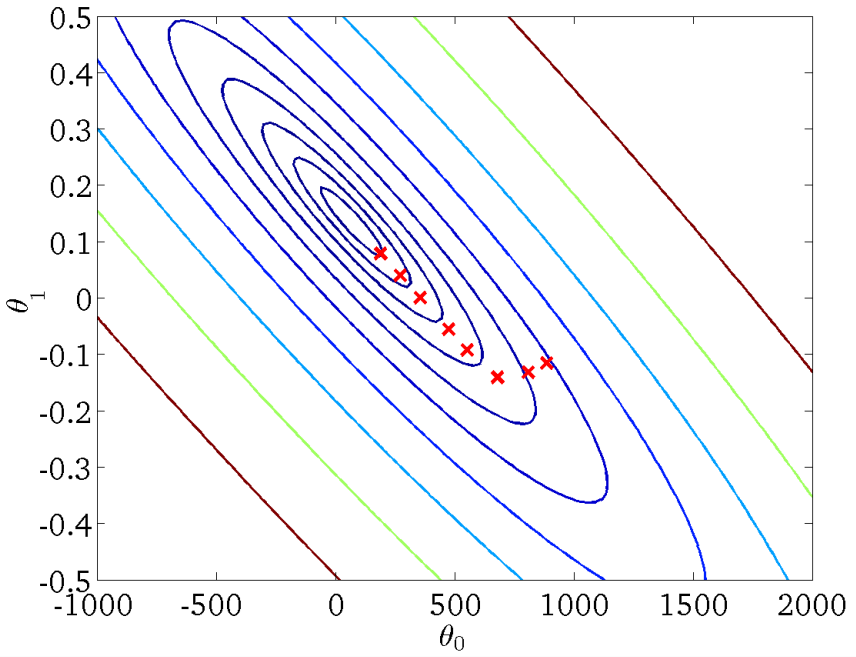
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

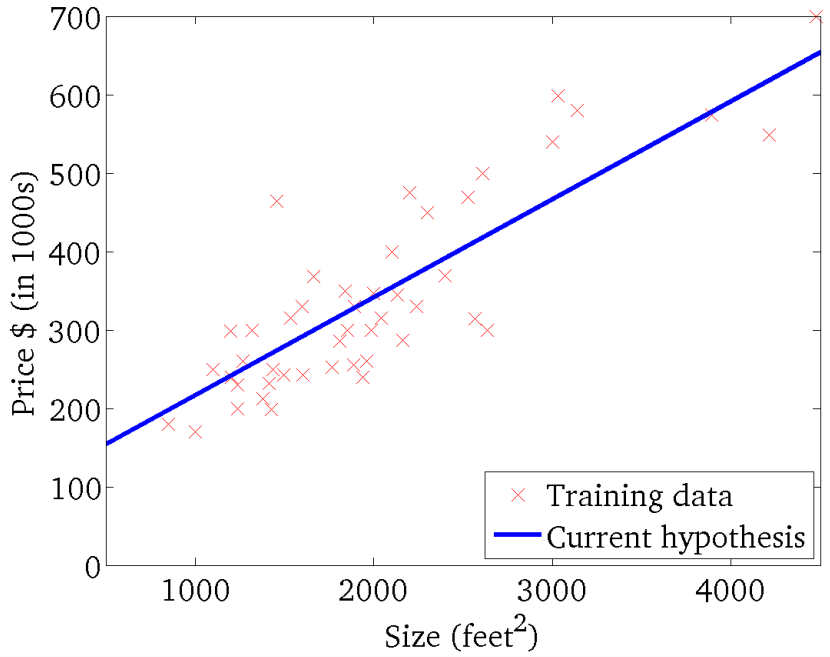
(function of the parameters θ_0, θ_1)



Gradient Descent for Linear Regression

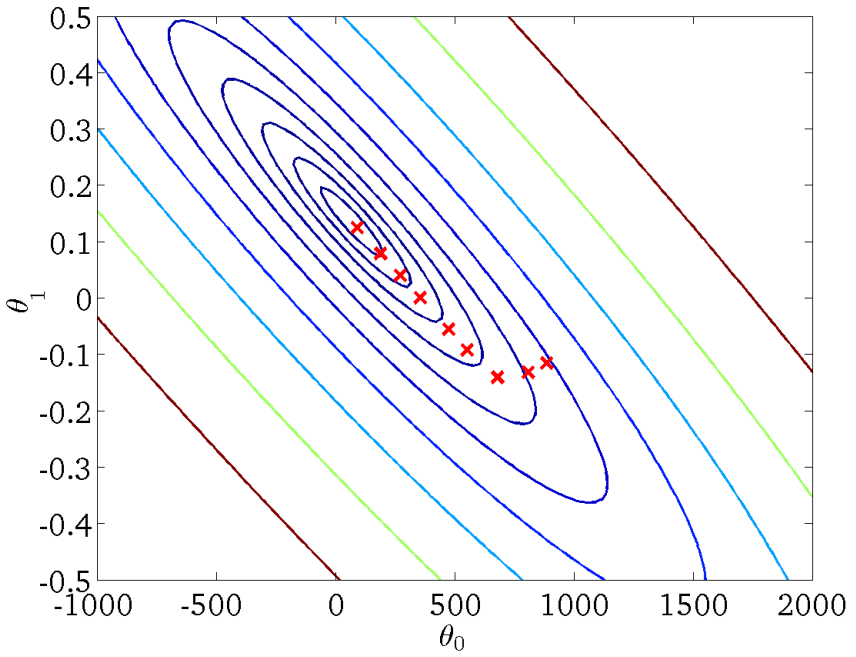
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Multiple Features (Variables)

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

$$x_3^{(2)} = 2$$

Multiple Features (Variables)

Hypothesis:

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

Now: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

For convenience of notation, define $x_0 = 1$

$$h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Multivariate linear regression

Gradient Descent with Multiple Variables

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$x_0^{(i)} = 1$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

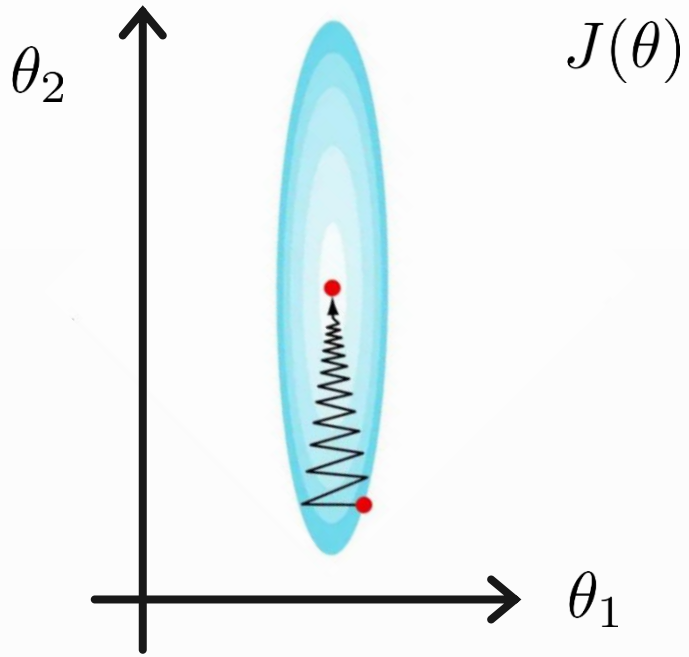
$\ldots$

Gradient Descent: Feature Scaling

Idea: Make sure features are on a similar scale

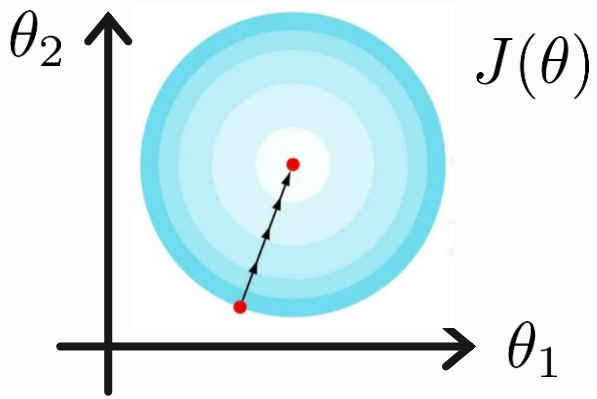
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Gradient Descent: Feature Scaling – Mean Normalization

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

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

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

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

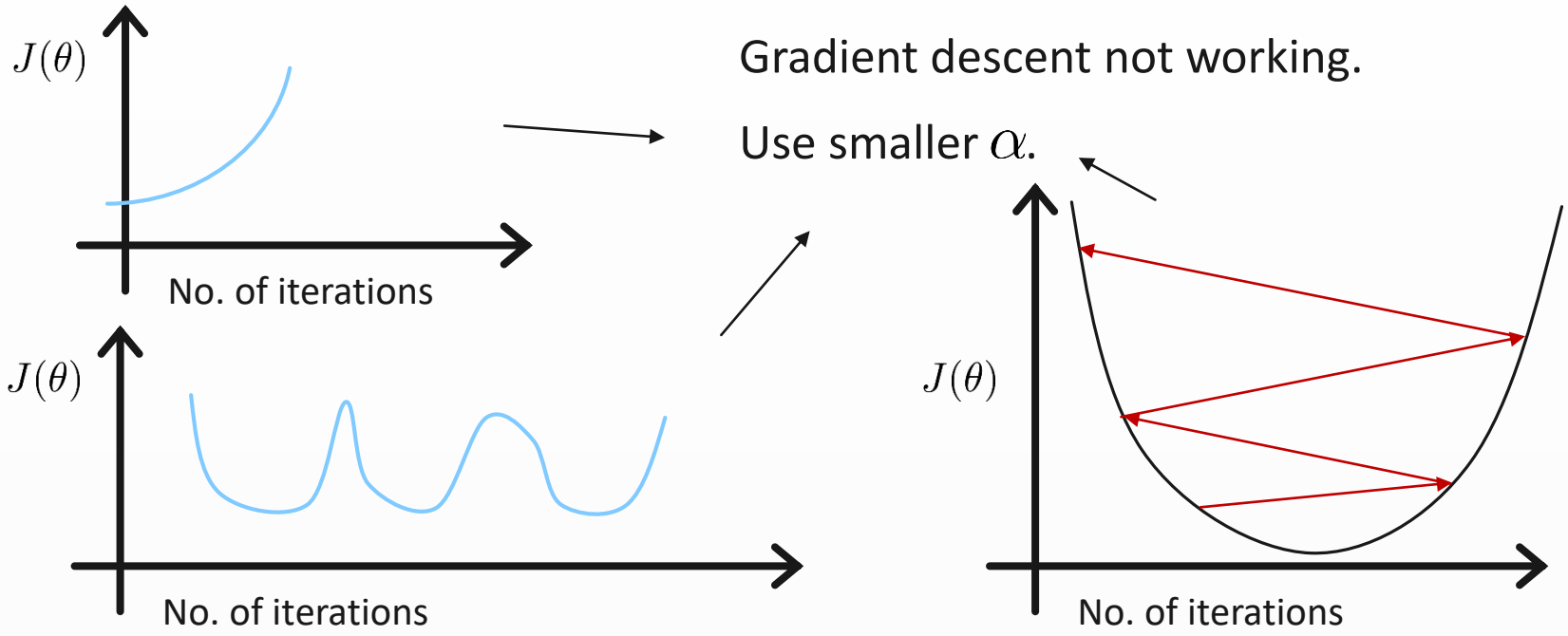
$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1} \qquad x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$$

μ_i: average value of x_i in training set

s_i: range (max-min) or standard deviation

Gradient Descent: Learning Rate

Making sure gradient descent is working correctly.



- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Gradient Descent: Learning Rate

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

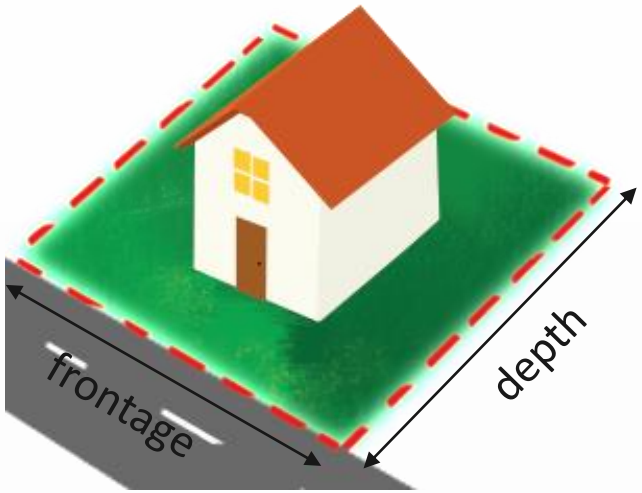
$$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, \ldots$$

$\xrightarrow{\quad}$ 3x

Features and Polynomial Regression

Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$



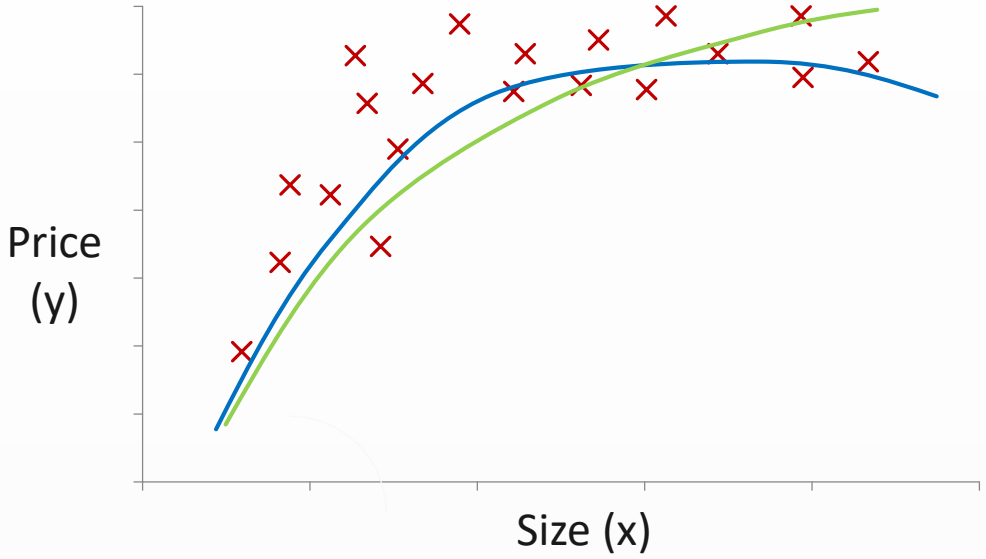
Area:

x = frontage * depth

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

Features and Polynomial Regression

Choice of features



$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

Normal Equation

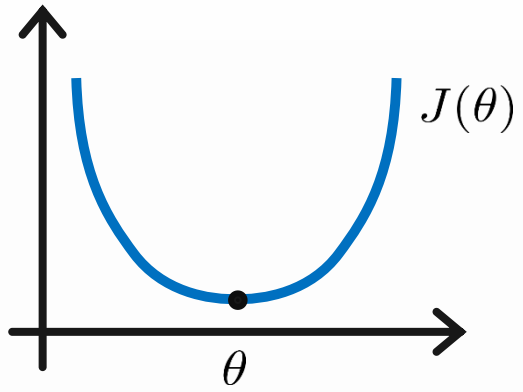
Normal equation: Method to solve for analytically.

Intuition: If 1D $(\theta \in \mathbb{R})$

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{\partial}{\partial\theta} J(\theta) = 0$$

Solve for θ



$$\theta \in \mathbb{R}^{n+1} \qquad J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial\theta_j} J(\theta) = \cdots = 0 \qquad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

Normal Equation

Examples: $m = 5$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1					

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 3000 & 4 & 1 & 38 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 540 \end{bmatrix}$$

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

Gradient Descent vs Normal Equation

m training examples, n features.

Gradient Descent

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α.
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.

References

- A. Ng. Machine Learning, Lecture Notes.
- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

LECTURE 4

LOGISTIC REGRESSION: CLASSIFICATION

Instructor: Asst. Prof. Barış Başpınar

Classification

Email: Spam / Not Spam?

Online Transactions: Fraudulent (Yes / No)?

Tumor: Malignant / Benign ?

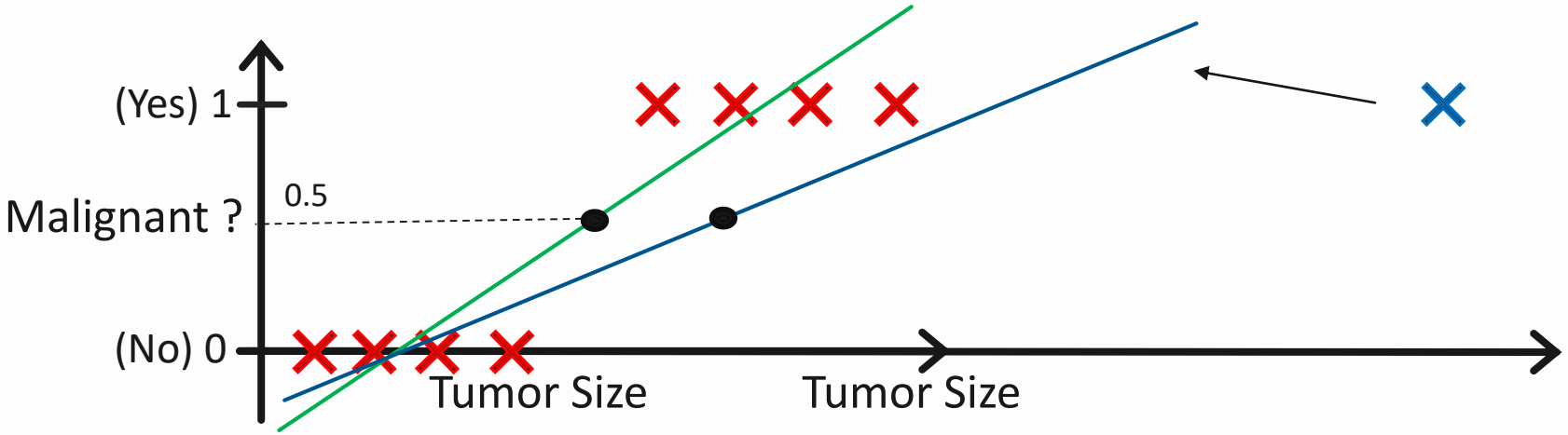
$y \in \{0, 1\}$

0: “Negative Class” (e.g., benign tumor)

1: “Positive Class” (e.g., malignant tumor)

Binary classification

Classification



Applying linear regression into classification problem often isn't a good idea

Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

Classification

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Hypothesis Representation

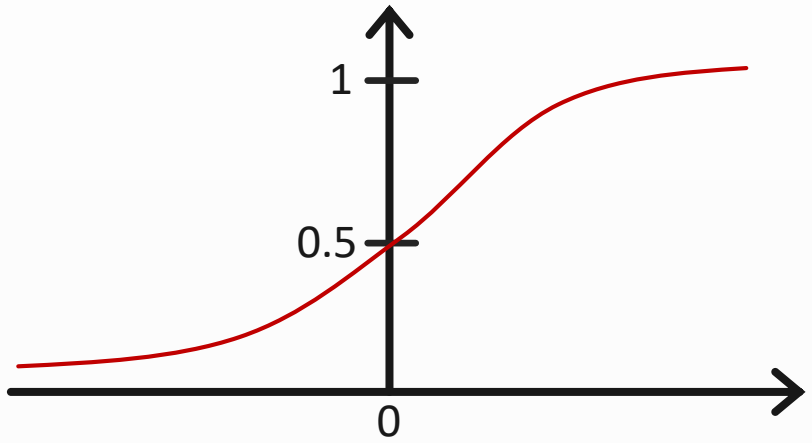
Logistic Regression Model

Want $0 \leq h_\theta(x) \leq 1$

$$h_\theta(x) = g(\theta^T x)$$

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

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Sigmoid function

Logistic function

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$$h_\theta(x) = 0.7$$

Tell patient that 70% chance of tumor being malignant

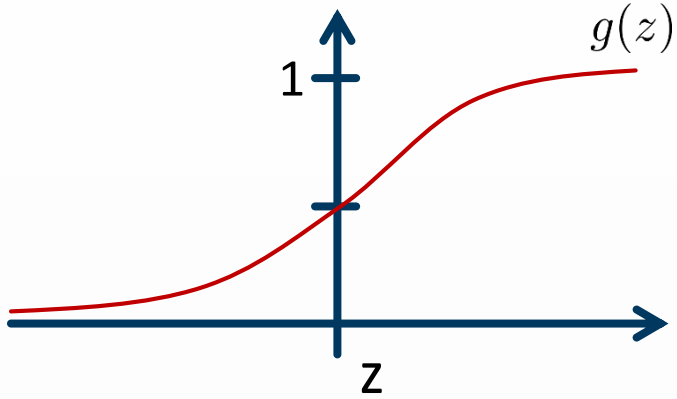
"probability that y = 1, given x, parameterized by θ"

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

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

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



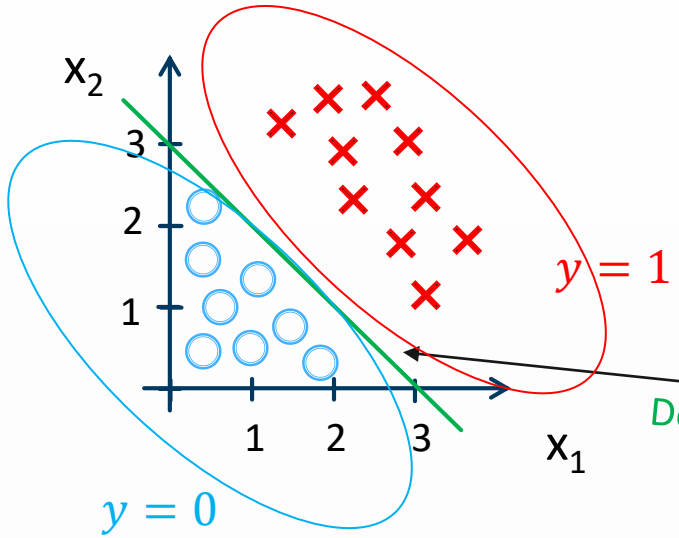
Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$$g(z) \geq 0.5 \quad \rightarrow \quad z \geq 0 \quad \rightarrow \quad \theta^T x \geq 0$$

predict "$y = 0$" if $h_\theta(x) < 0.5$

$$g(z) < 0.5 \quad \rightarrow \quad z < 0 \quad \rightarrow \quad \theta^T x < 0$$

Decision Boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

We have not talked how to choose θ yet. Assume that:

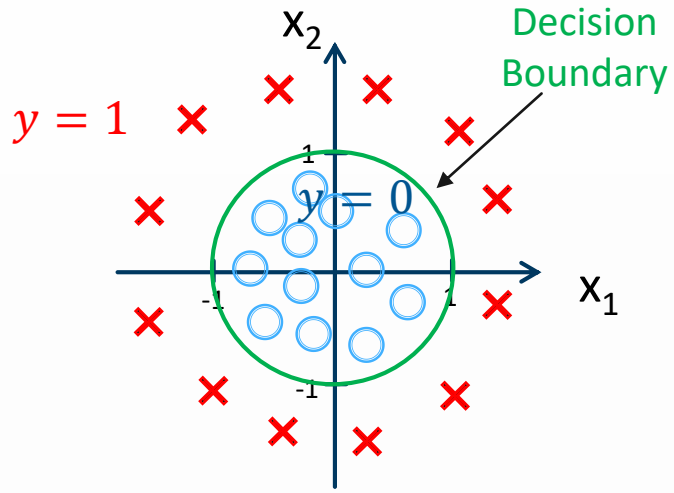
$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$h_\theta(x) = 0.5 \rightarrow y = 1$

$x_1 + x_2 = 3$

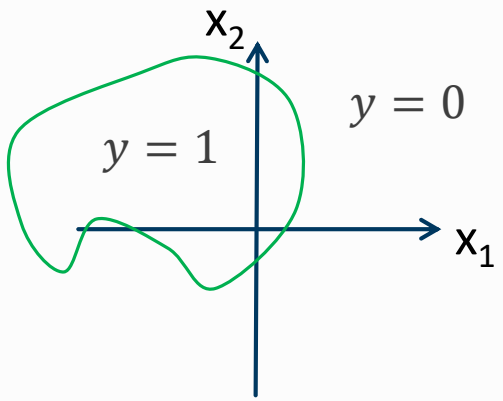
Non-linear Decision Boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \ldots)$$

Cost Function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

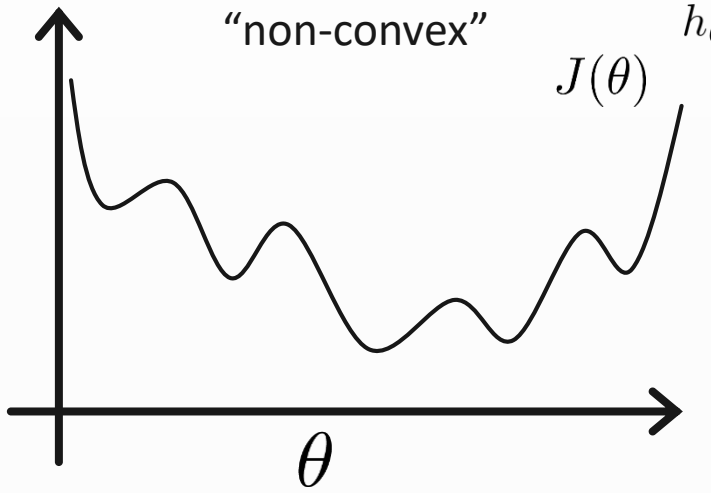
Cost Function

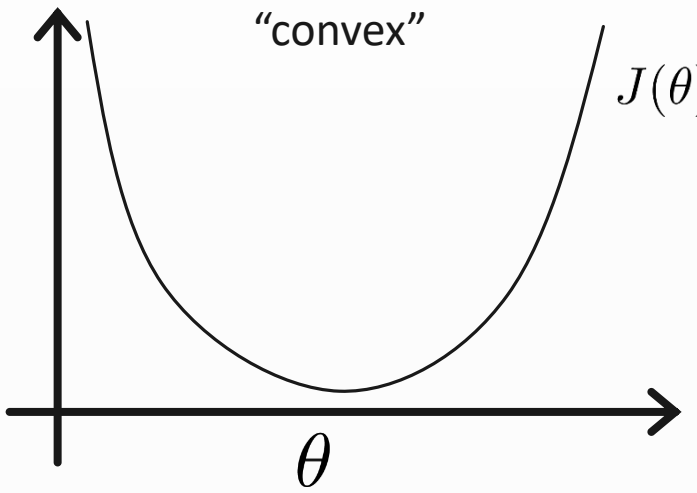
Linear regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Logistic regression's hypothesis

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



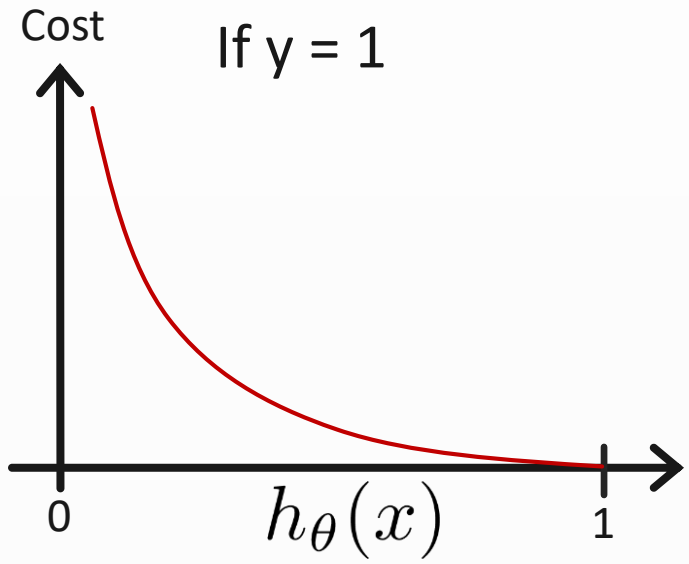


Define a different cost function that is convex in which the gradient descent can converge to the global minimum

Cost Function

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



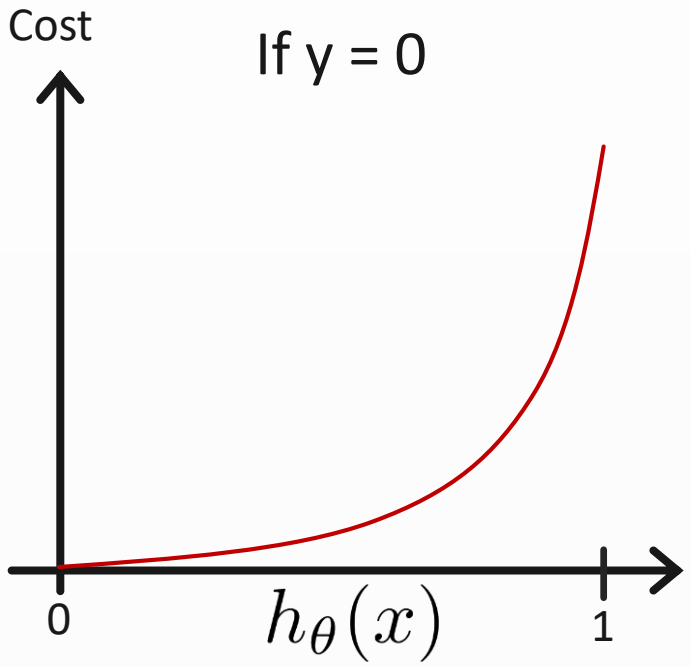
$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Cost Function

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Cost Function and Gradient Descent

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

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

Compact way to present this cost function:

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

"convex"

This cost function can also be derived from statistics using the principle of Maximum Likelihood Estimation

Cost Function and Gradient Descent

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

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Cost Function and Gradient Descent

Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

$\}$

(simultaneously update all θ_j)

Cost Function and Gradient Descent

Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Algorithm looks identical to linear regression!
But the definition of the hypothesis has changed.

Advanced Optimization

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent

Disadvantages:

- More complex

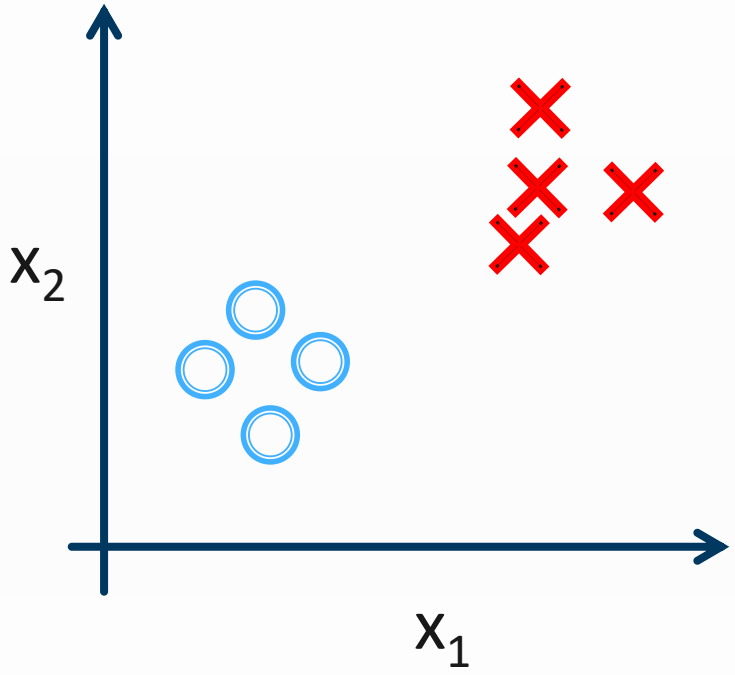
Multi-class Classification

Email foldering/tagging: Work, Friends, Family, Hobby

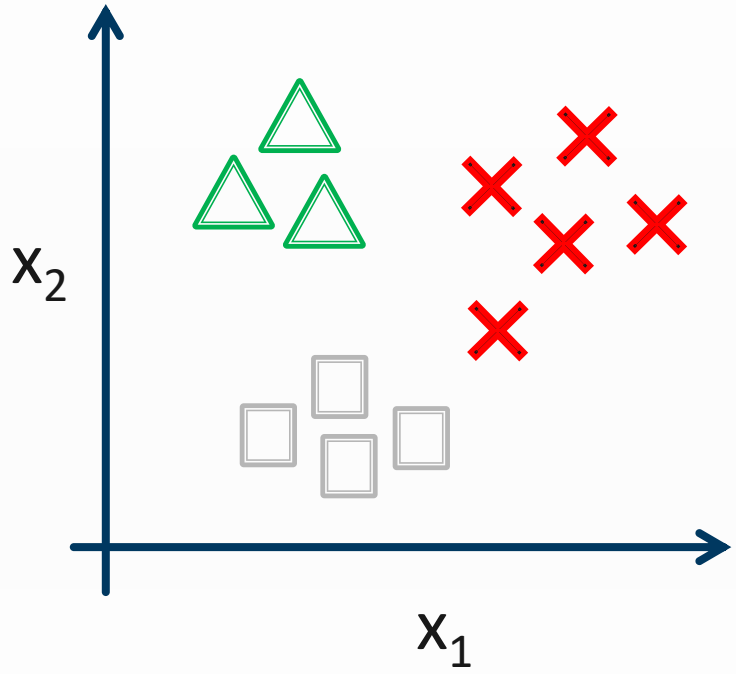
Medical diagrams: Not ill, Cold, Flu

Weather: Sunny, Cloudy, Rain, Snow

Binary classification:

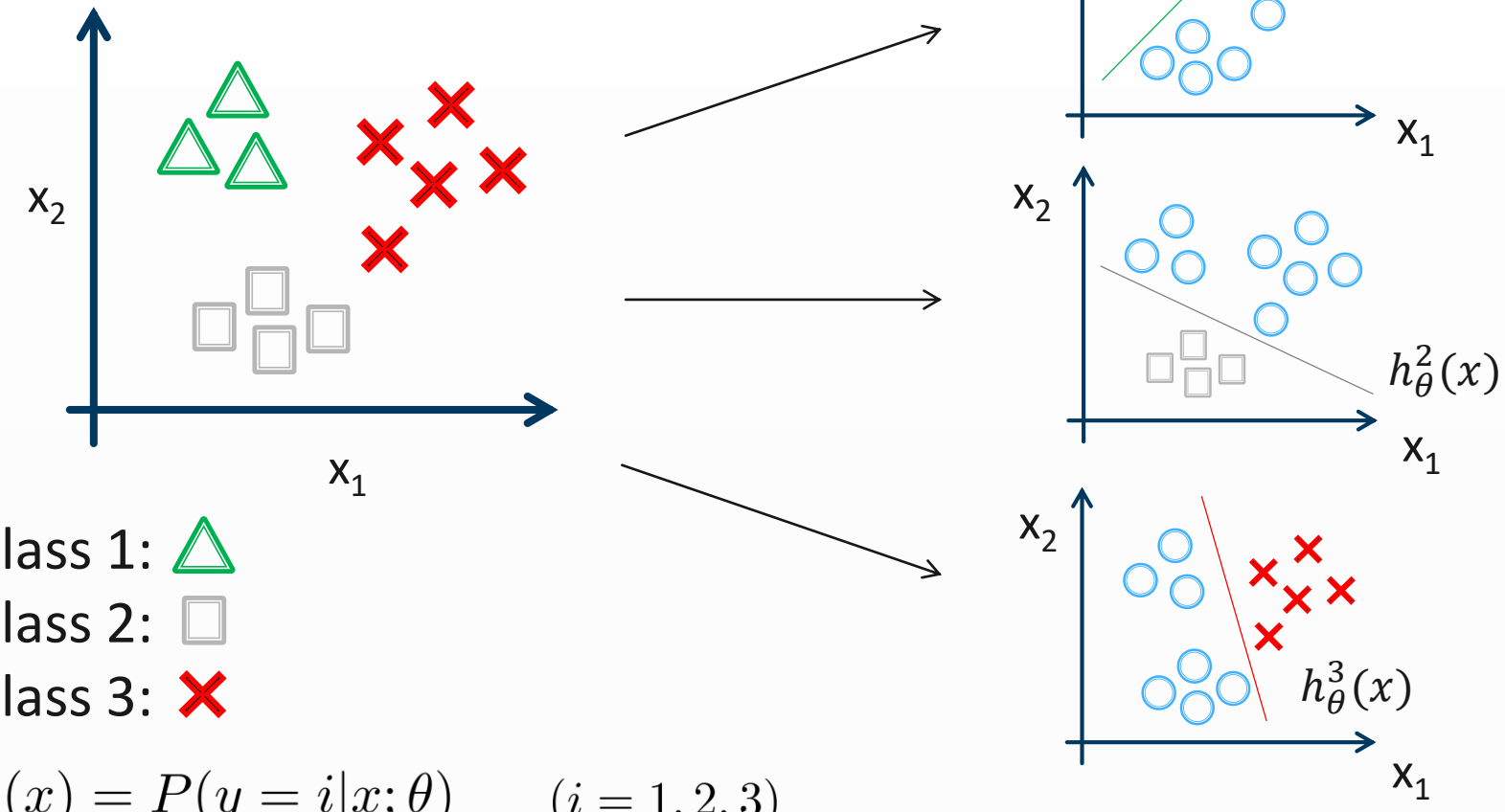


Multi-class classification:



Multi-class Classification

One-vs-all (one-vs-rest):



Class 1: △

Class 2: □

Class 3: ✕

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$

Multi-class Classification

One-vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

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

Logistic Regression Tips

- **Binary Output Variable**: This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge**: It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

References

- A. Ng. Machine Learning, Lecture Notes.
- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

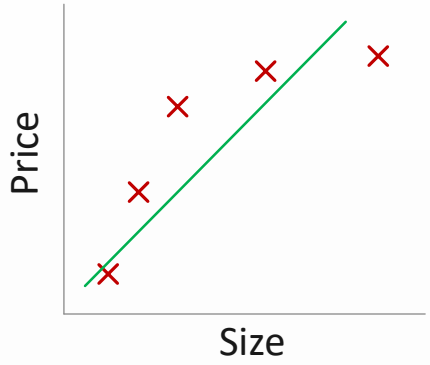
SPRING ‘23

LECTURE 5

REGULARIZATION

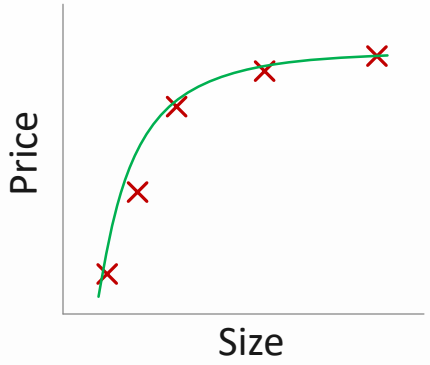
Instructor: Asst. Prof. Barış Başpınar

Example: Linear regression (housing prices)



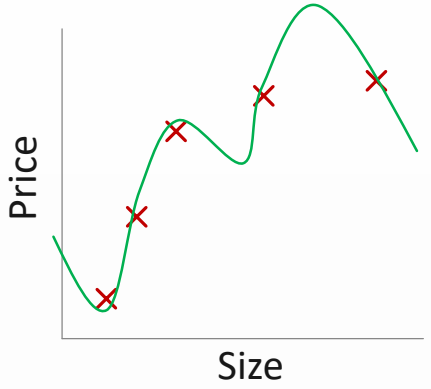
$\theta_0 + \theta_1 x$

Underfit, High Bias



$\theta_0 + \theta_1 x + \theta_2 x^2$

Just Right

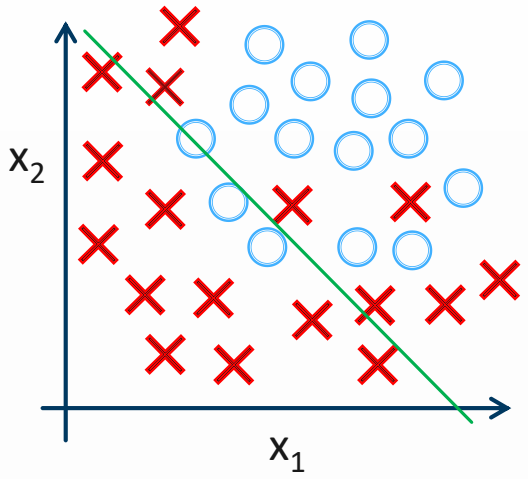


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfit, High Variance

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

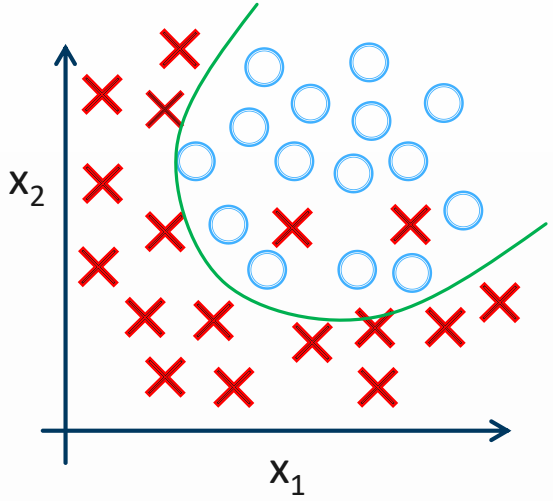
Example: Logistic regression



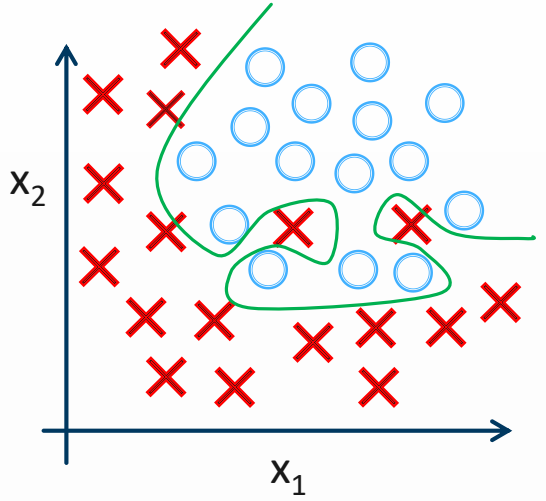
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)

"Underfit"



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



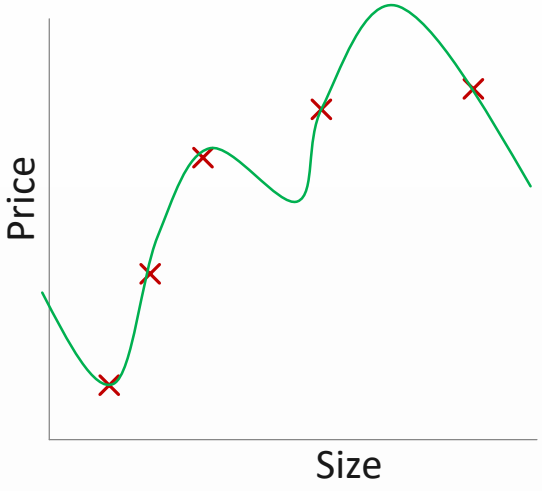
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

"Overfit"

Addressing overfitting

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

Many features due to available data



Many features due to high-degree polynomial

Overfitting can become a problem when:

- Using high-degree polynomials, or complex models
- Having a lot of features and very little training data

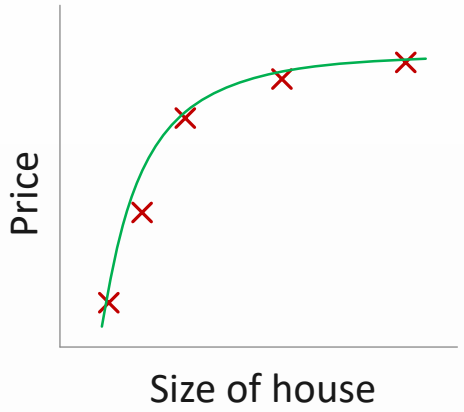
Addressing overfitting

Options:

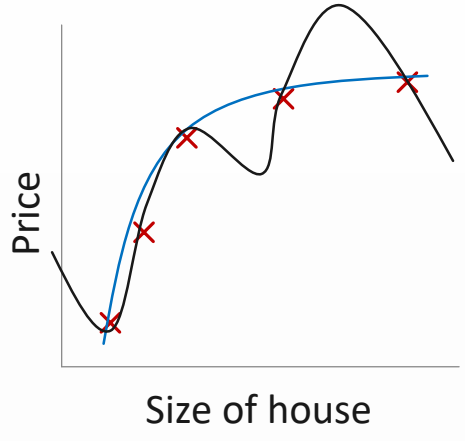
1. Reduce number of features
   - Manually select which features to keep.
   - Model selection algorithm (later in course)

2. Regularization
   - Keep all the features, but reduce magnitude/values of parameters
   - Works well when we have a lot of features, each of which contributes a bit to predicting

Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \quad + 1000\theta_3^2 \; + \; 1000\theta_4^2$$

$$\theta_3 \cong 0, \qquad \theta_4 \cong 0$$

Regularization

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

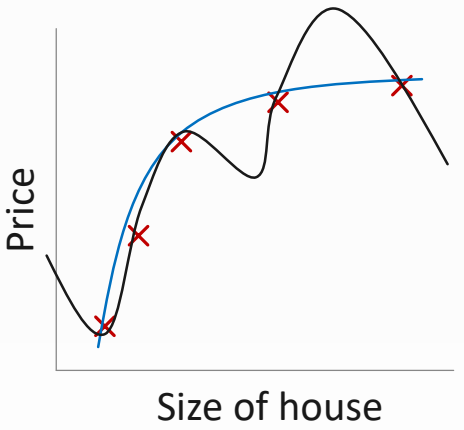
Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

Regularization parameter

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

$$\min_\theta J(\theta)$$



Regularization

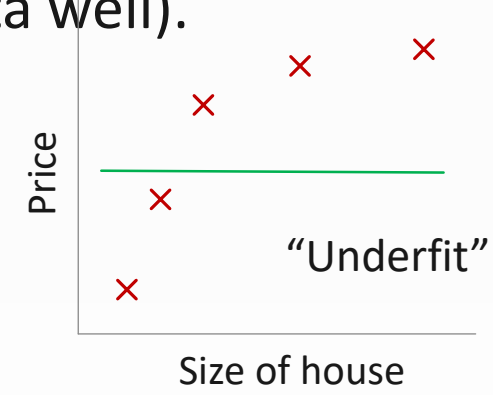
In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

$$\theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$



Regularized linear regression

$$\min_{\theta} J(\theta) \qquad J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

Gradient descent

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$(j = \cancel{0}, 1, 2, 3, \ldots, n)$$

$\}$

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

Regularized logistic regression

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradient descent

Repeat $\{$

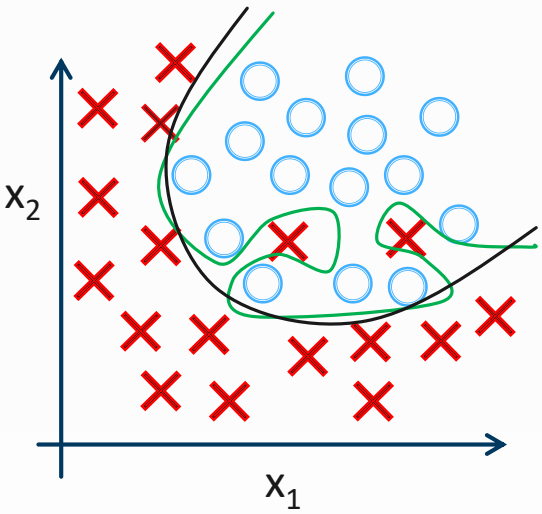
$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$(j = \cancel{0}, 1, 2, 3, \ldots, n)$$

$\}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



Alternative Regularization Terms

Ordinary Least Squares

$$\hat{y} = w^T \mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{p} ||w^T \mathbf{x}_i - y_i||^2$$

Ridge Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||^2$$

(We have already defined this one!)

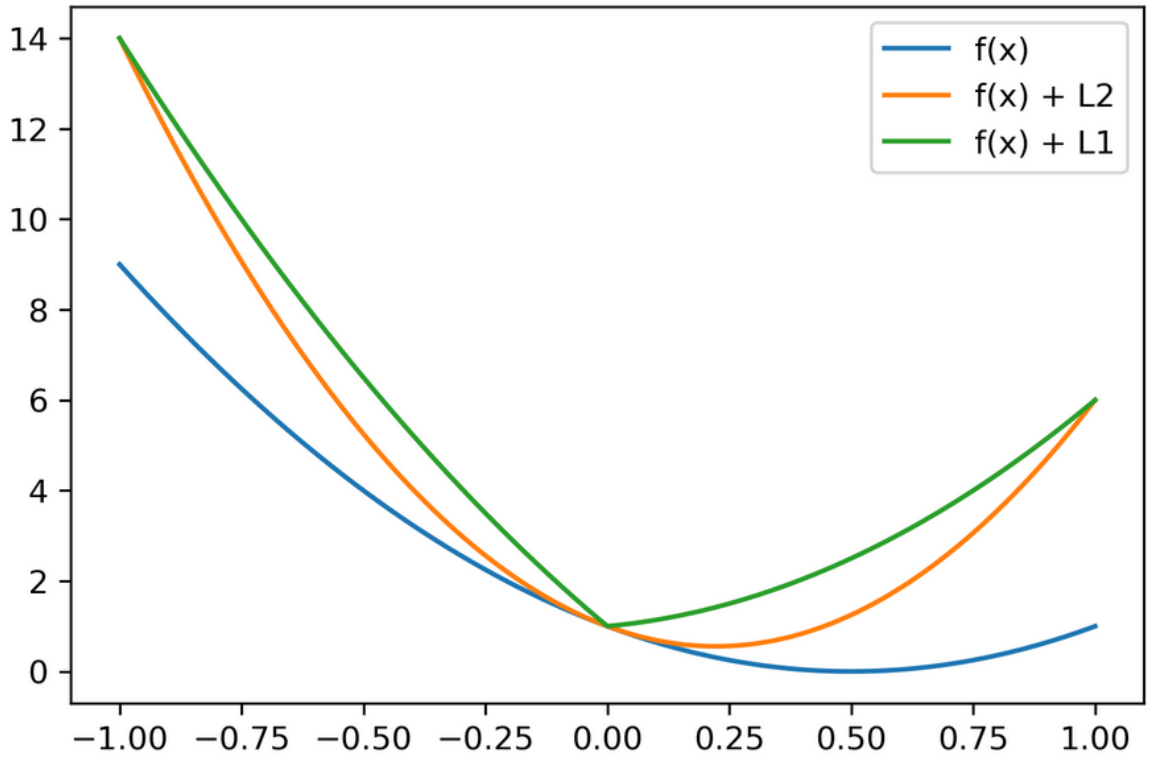
Lasso Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||_1$$

- Shrinks w towards zero like Ridge
- Sets some w exactly to zero - automatic feature selection!

Alternative Regularization Terms

Understanding L1 and L2 Penalties



$$f(x) = (2x-1)^2$$
$$f(x) + L2 = (2x-1)^2 + \alpha x^2$$
$$f(x) + L1 = (2x-1)^2 + \alpha|x|$$

Alternative Regularization Terms

Ridge regression

- In ridge regression, though, the coefficients (*w*) are chosen not only so that they predict well on the training data, but also to fit an additional constraint.
- We also want the magnitude of coefficients to be as small as possible; in other words, all entries of *w* should be close to zero.
- Intuitively, this means each feature should have as little effect on the outcome as possible (which translates to having a small slope), while still predicting well.
- The particular regularization term used by ridge regression is known as L2 regularization.
- Ridge is a more restricted model, so we are less likely to overfit.
- The optimum setting of alpha depends on the particular dataset we are using. Increasing alpha forces coefficients to move more toward zero, which decreases training set performance but might help generalization.

Alternative Regularization Terms

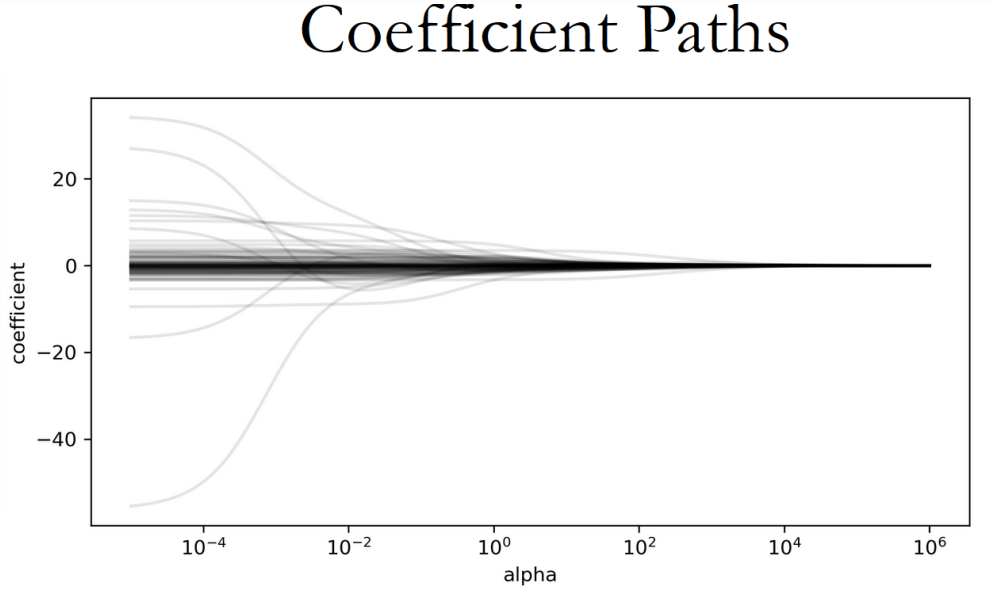
Lasso regression

- An alternative to Ridge for regularizing linear regression is Lasso. As with ridge regression, using the lasso also restricts coefficients to be close to zero, but in a slightly different way, called L1 regularization.
- The consequence of L1 regularization is that when using the lasso, some coefficients are *exactly zero*. This means some features are entirely ignored by the model.
- This can be seen as a form of automatic feature selection. Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of your model.
- Two important points:
  - Solution numeracy: Because L2 is Euclidean distance, there is always one right answer as to how to get between two points fastest. Because L1 is taxicab distance, there are as many solutions to getting between two points
  - Computational difficulty: L2 has a closed form solution because it's a square of a thing. L1 does not have a closed form solution because it is a non-differenciable piecewise function, as it involves an absolute value. For this reason, L1 is computationally more expensive

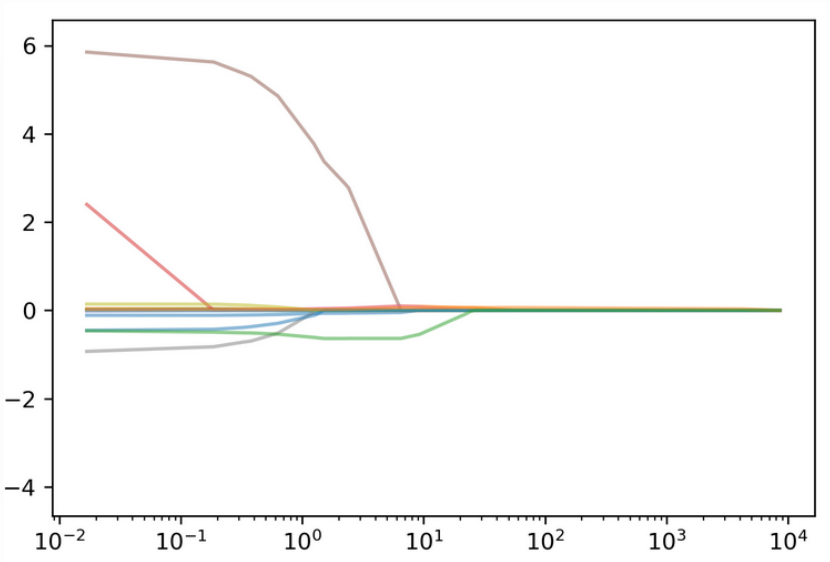
Alternative Regularization Terms

- In practice, ridge regression is usually the first choice between these two models.
- However, if you have a large amount of features and expect only a few of them to be important, Lasso might be a better choice.
- Similarly, if you would like to have a model that is easy to interpret, Lasso will provide a model that is easier to understand, as it will select only a subset of the input features.

Ridge



Lasso



References

- A. Ng. Machine Learning, Lecture Notes.
- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

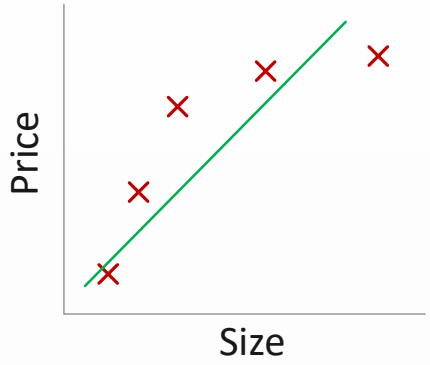
SPRING ‘23

LECTURE 5

REGULARIZATION

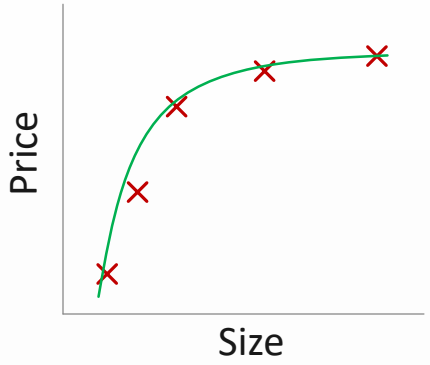
Instructor: Asst. Prof. Barış Başpınar

Example: Linear regression (housing prices)



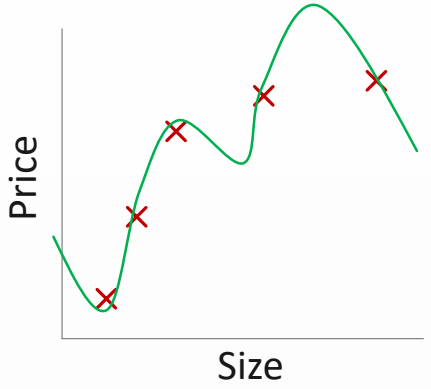
$\theta_0 + \theta_1 x$

Underfit, High Bias



$\theta_0 + \theta_1 x + \theta_2 x^2$

Just Right

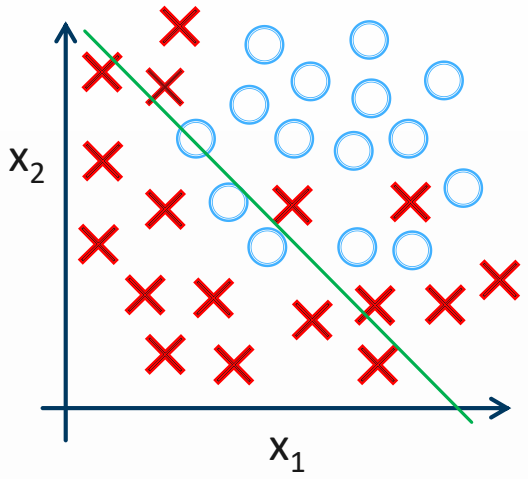


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfit, High Variance

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

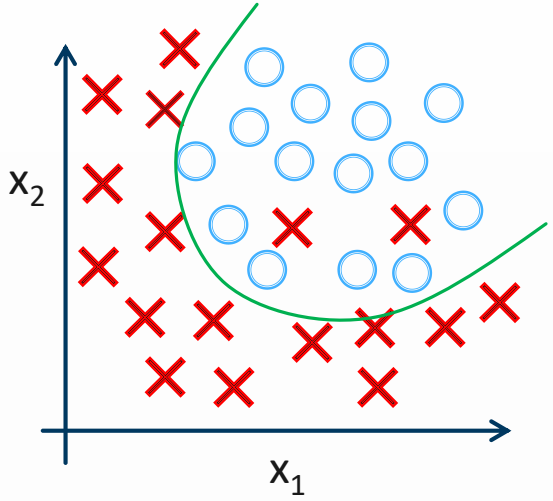
Example: Logistic regression



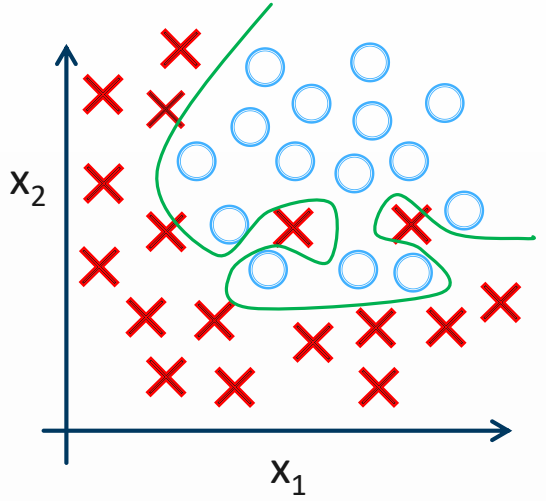
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)

"Underfit"



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



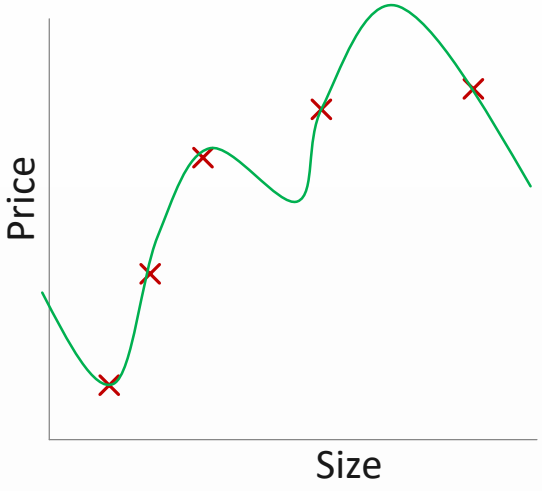
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

"Overfit"

Addressing overfitting

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

Many features due to available data



Many features due to high-degree polynomial

Overfitting can become a problem when:

- Using high-degree polynomials, or complex models
- Having a lot of features and very little training data

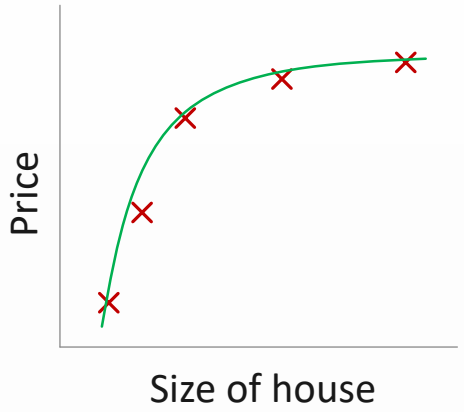
Addressing overfitting

Options:

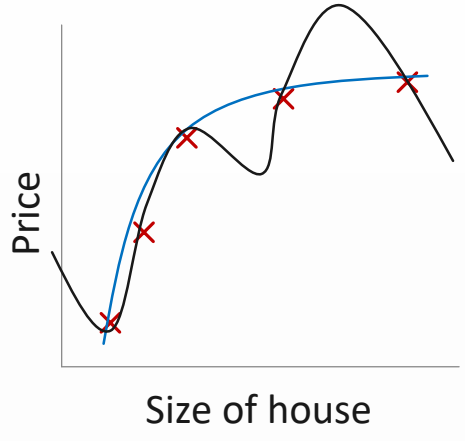
1. Reduce number of features
   - Manually select which features to keep.
   - Model selection algorithm (later in course)

2. Regularization
   - Keep all the features, but reduce magnitude/values of parameters
   - Works well when we have a lot of features, each of which contributes a bit to predicting

Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \quad + 1000\theta_3^2 \; + \; 1000\theta_4^2$$

$$\theta_3 \cong 0, \qquad \theta_4 \cong 0$$

Regularization

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

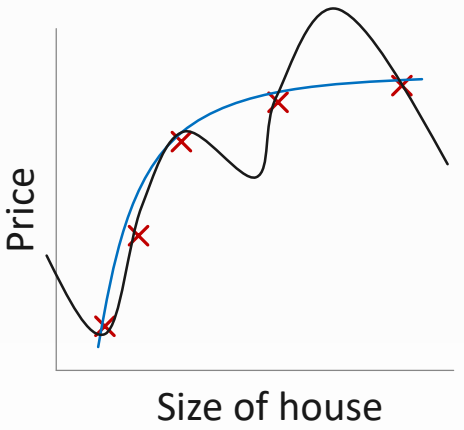
Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

Regularization parameter

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

$$\min_\theta J(\theta)$$



Regularization

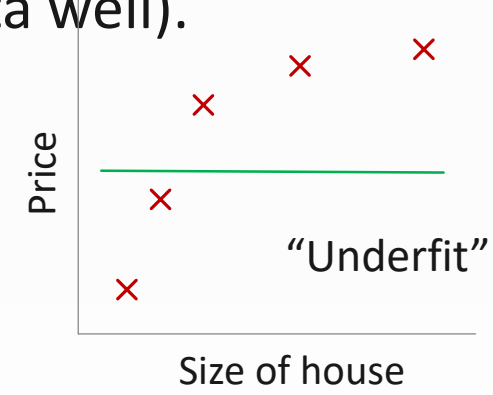
In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

$$\theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$



Regularized linear regression

$$\min_{\theta} J(\theta) \qquad J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

Gradient descent

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$(j = \cancel{0}, 1, 2, 3, \ldots, n)$$

$\}$

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

Regularized logistic regression

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradient descent

Repeat $\{$

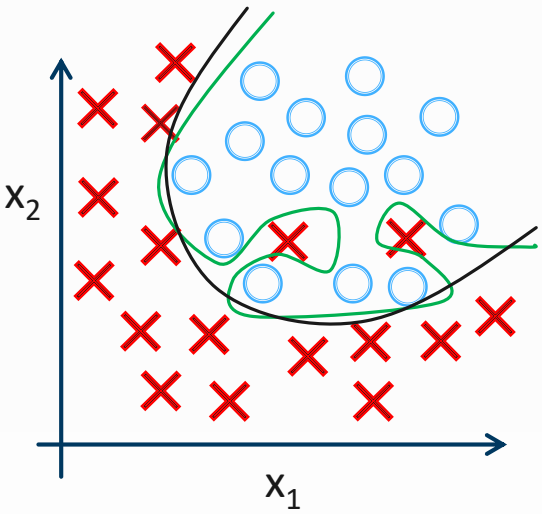
$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$$(j = \cancel{0}, 1, 2, 3, \ldots, n)$$

$\}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



Alternative Regularization Terms

Ordinary Least Squares

$$\hat{y} = w^T \mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{p} ||w^T \mathbf{x}_i - y_i||^2$$

Ridge Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||^2$$

(We have already defined this one!)

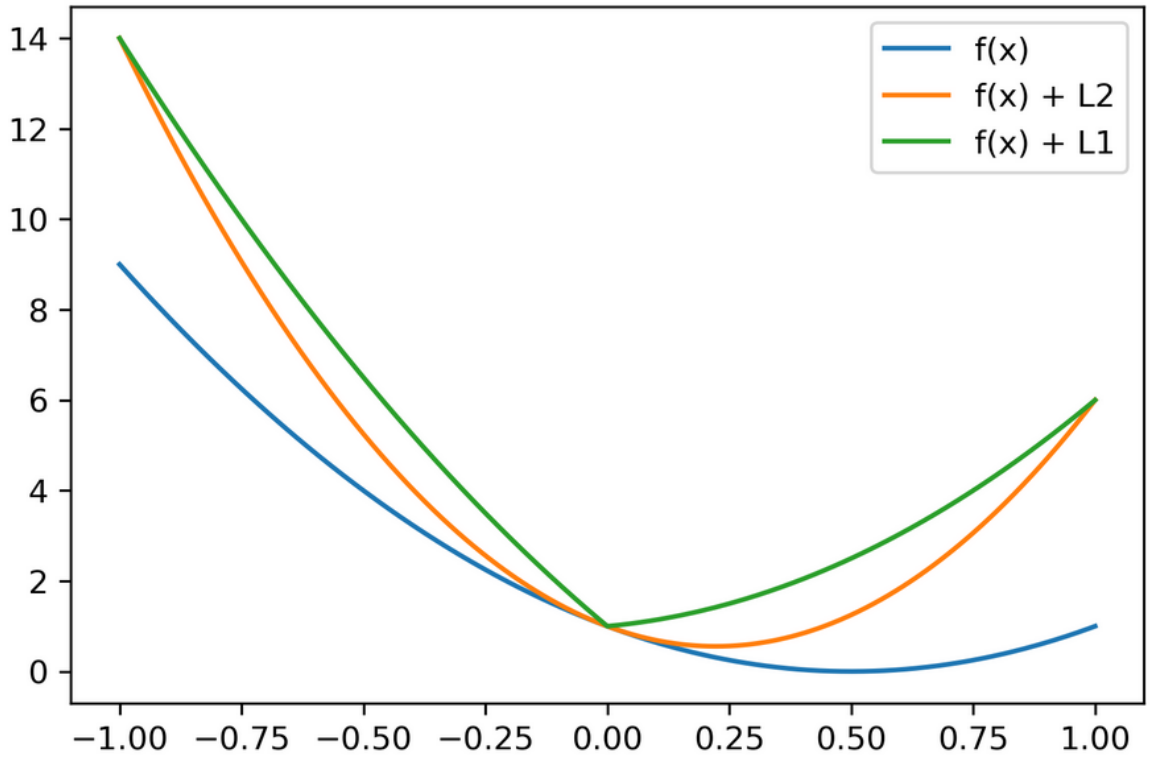
Lasso Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||_1$$

- Shrinks w towards zero like Ridge
- Sets some w exactly to zero - automatic feature selection!

Alternative Regularization Terms

Understanding L1 and L2 Penalties



$$f(x) = (2x-1)^2$$
$$f(x) + L2 = (2x-1)^2 + \alpha x^2$$
$$f(x) + L1 = (2x-1)^2 + \alpha|x|$$

Alternative Regularization Terms

Ridge regression

- In ridge regression, though, the coefficients (*w*) are chosen not only so that they predict well on the training data, but also to fit an additional constraint.
- We also want the magnitude of coefficients to be as small as possible; in other words, all entries of *w* should be close to zero.
- Intuitively, this means each feature should have as little effect on the outcome as possible (which translates to having a small slope), while still predicting well.
- The particular regularization term used by ridge regression is known as L2 regularization.
- Ridge is a more restricted model, so we are less likely to overfit.
- The optimum setting of alpha depends on the particular dataset we are using. Increasing alpha forces coefficients to move more toward zero, which decreases training set performance but might help generalization.

Alternative Regularization Terms

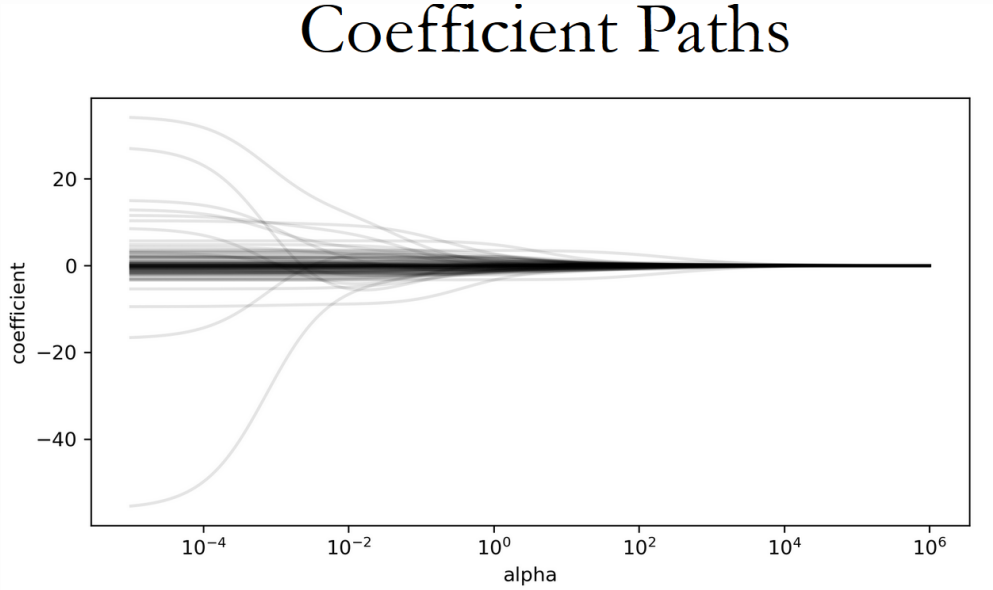
Lasso regression

- An alternative to Ridge for regularizing linear regression is Lasso. As with ridge regression, using the lasso also restricts coefficients to be close to zero, but in a slightly different way, called L1 regularization.
- The consequence of L1 regularization is that when using the lasso, some coefficients are *exactly zero*. This means some features are entirely ignored by the model.
- This can be seen as a form of automatic feature selection. Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of your model.
- Two important points:
  - Solution numeracy: Because L2 is Euclidean distance, there is always one right answer as to how to get between two points fastest. Because L1 is taxicab distance, there are as many solutions to getting between two points
  - Computational difficulty: L2 has a closed form solution because it's a square of a thing. L1 does not have a closed form solution because it is a non-differenciable piecewise function, as it involves an absolute value. For this reason, L1 is computationally more expensive

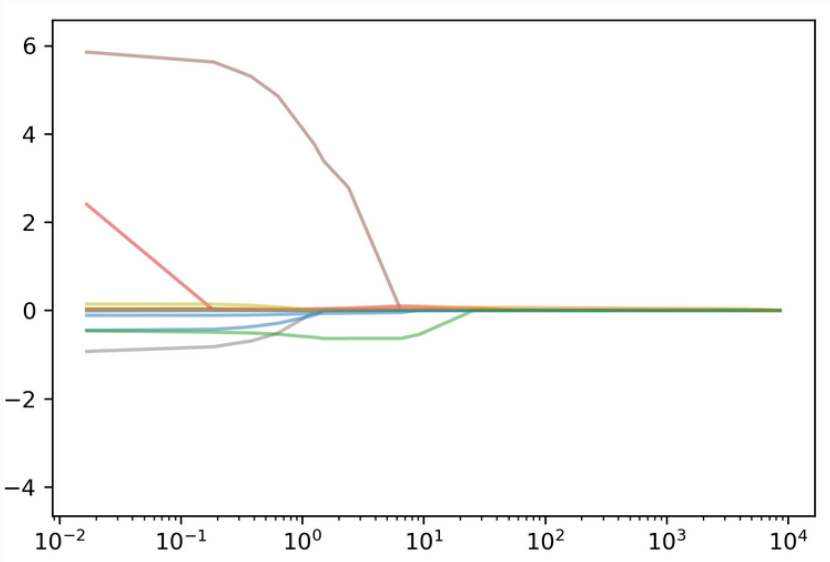
Alternative Regularization Terms

- In practice, ridge regression is usually the first choice between these two models.
- However, if you have a large amount of features and expect only a few of them to be important, Lasso might be a better choice.
- Similarly, if you would like to have a model that is easy to interpret, Lasso will provide a model that is easier to understand, as it will select only a subset of the input features.

Ridge



Lasso



References

- A. Ng. Machine Learning, Lecture Notes.
- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

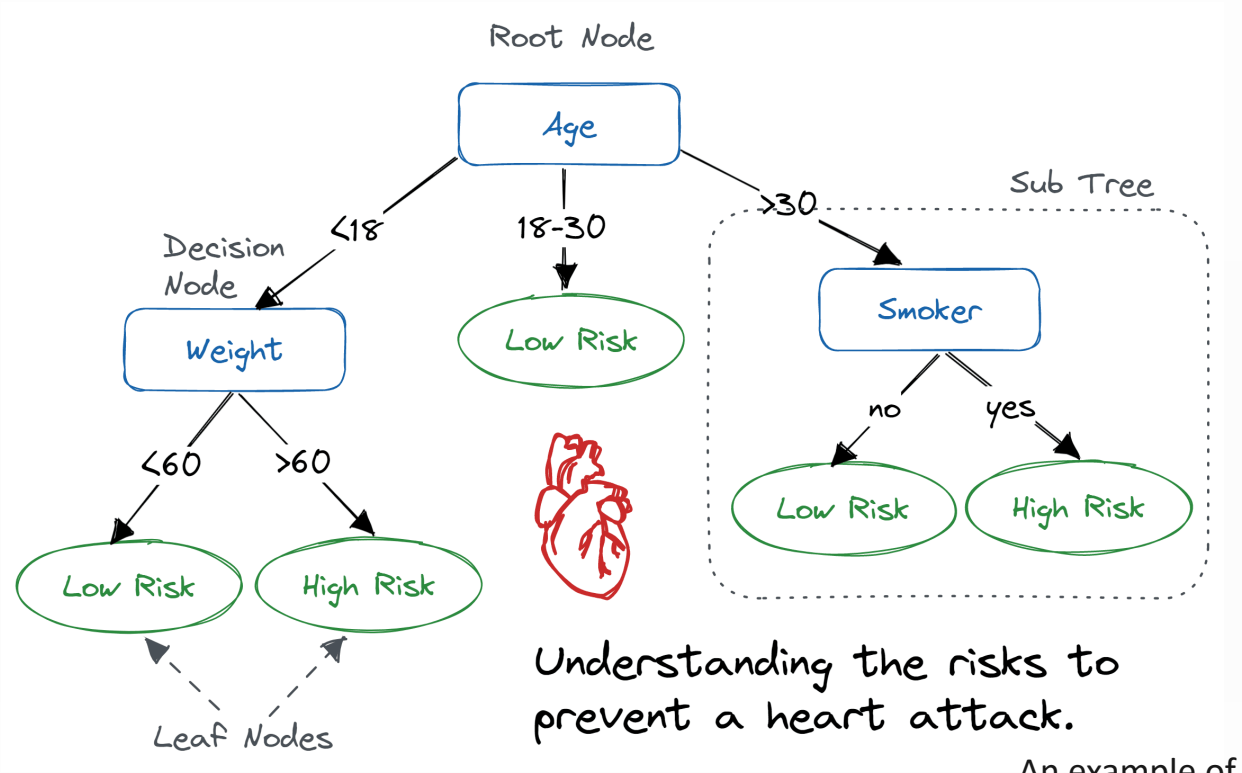
LECTURE 6

DECISION TREES

Instructor: Asst. Prof. Barış Başpınar

Decision Tree

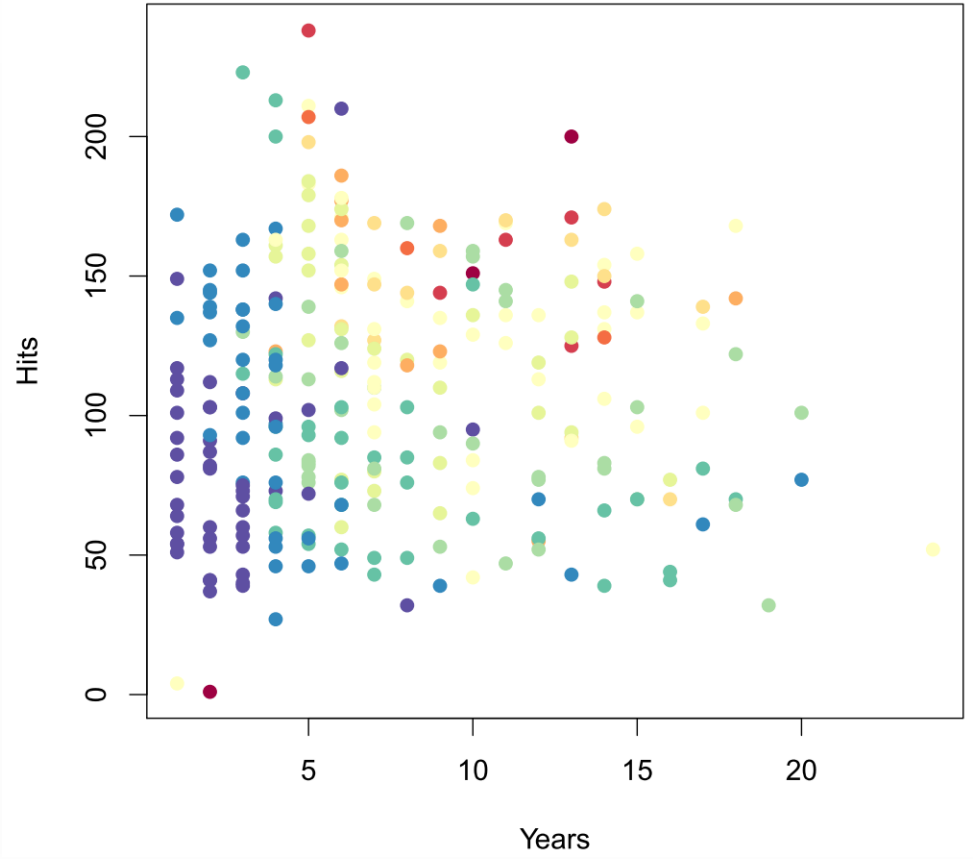
- Decision trees are non-parametric supervised learning method used for classification and regression
- These involve stratifying or segmenting the predictor space into a number of simple regions
- Essentially, they learn a hierarchy of if/else questions, leading to a decision

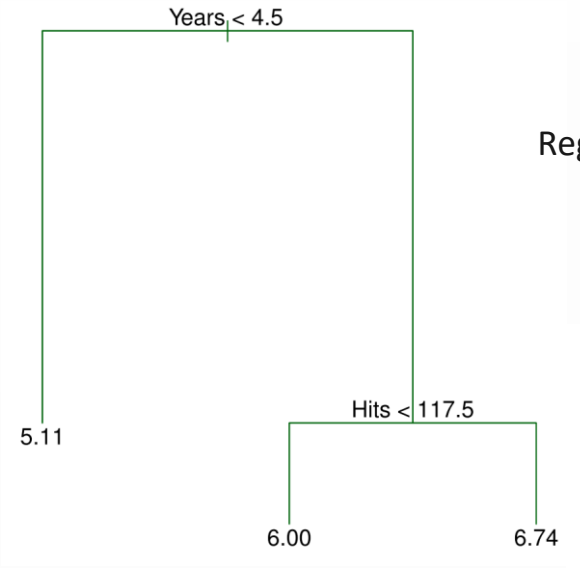


An example of classification tree

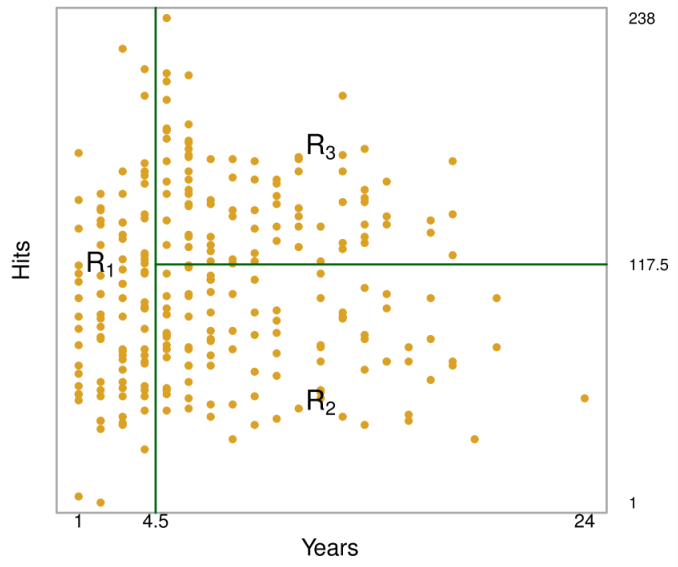
Baseball salary data - how would you stratify it?

- Salary is color-coded from low (blue, green) to high (yellow, red)



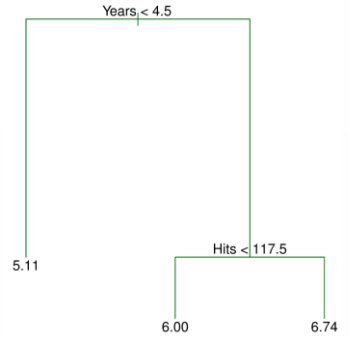


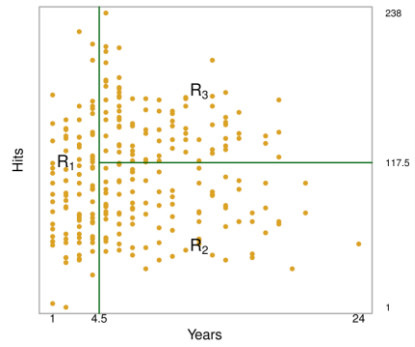
Regression tree for this data



Interpretation of Results

- Overall, the tree stratifies or segments the players into three regions of predictor space:
  - $R_1 = \{ X \mid Years < 4.5 \}$,
  - $R_2 = \{ X \mid Years \geq 4.5, Hits < 117.5 \}$, and
  - $R_3 = \{ X \mid Years \geq 4.5, Hits \geq 117.5 \}$

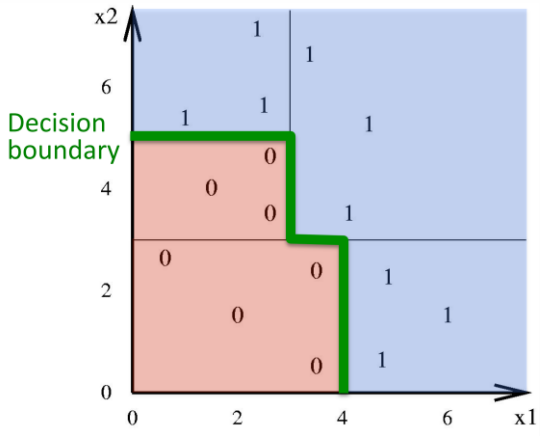


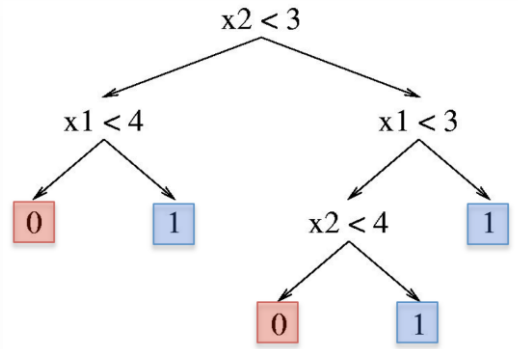


- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary
- But among players who have been in the major leagues for five or more years, players who made more Hits last year tend to have higher salaries
- It is easy to display, interpret and explain

Decision Trees – Decision Boundary

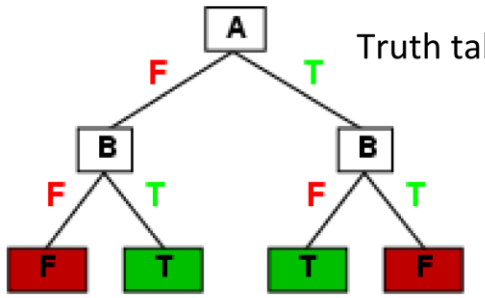
- Decision trees divide the feature space into axis-parallel (hyper-)rectangles
  - Non-linear decision boundaries can be generated





- Decision trees can represent any Boolean function of the input attributes

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Truth table row → path to leaf

Decision Trees

- Several decision tree algorithms:
  - ID3 (Iterative Dichotomiser 3)
    - Handling classification tasks
    - Using categorical attributes/features)
    - Using entropy ($H(s)$) or information gain (IG(s))

    for attribute selection
  - C4.5 (the successor of ID3)
    - Handling classification tasks
    - Using both continuous and categorical features
  - CART (classification and regression tree)
    - Handling both classification and regression tasks

- How to build a decision tree:
  - Start at the top of the tree
  - Grow it by "splitting" attributes one by one. To determine which attribute to split, look at "node impurity"
  - Assign leaf nodes the majority vote in the leaf
  - When we get to the bottom, prune the tree to prevent overfitting

- Key problems:
  - How to select best decision attribute?
  - How decide the threshold for the selected attribute? (for continuous features)
  - How to prune the tree?

Will the customer wait for a table?

- Let us firstly focus on a classification task with categorical variables (to illustrate the C4.5 algorithm)

Attributes/Features

1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
6.	Price: the restaurant's price range ($, $$, $$$).
7.	Raining: whether it is raining outside.
8.	Reservation: whether we made a reservation.
9.	Type: the kind of restaurant (French, Italian, Thai or Burger).
10.	WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

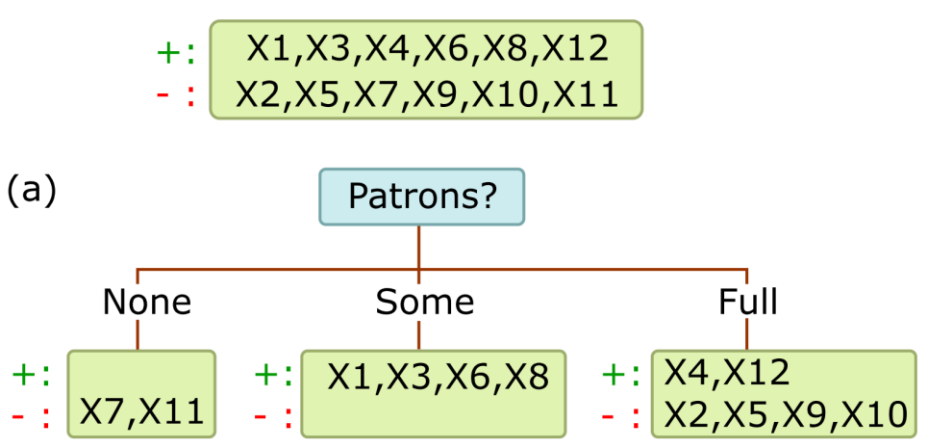
Continuous variables can be discretized to evaluate them as categorical variables

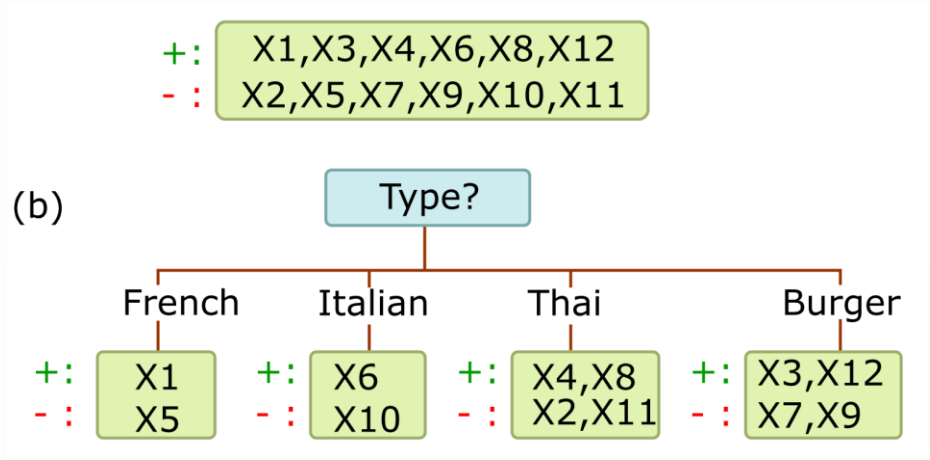
Examples/Samples

Example	Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Will the customer wait for a table?

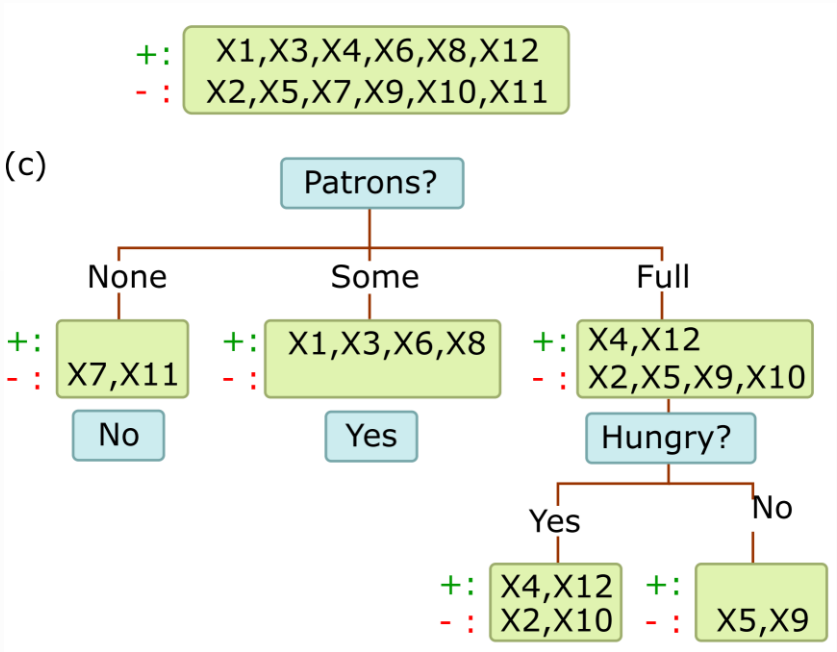
- Here are two options for the first feature to split at the top of the tree.
- Which one should we choose? Which one gives me the most information?





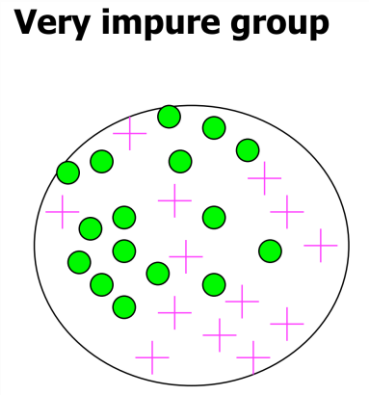
Will the customer wait for a table?

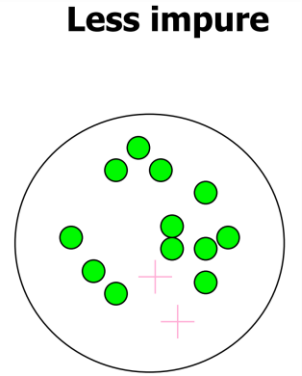
- What we need is a formula to compute “information” to chose the best attribute
- Before we do that, here’s another example. Let’s say we pick one of them (Patrons). Maybe then we’ll pick Hungry next, because it has a lot of “information”:

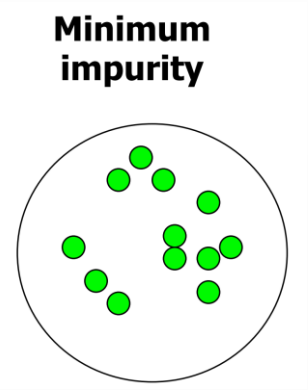


Choosing an Attribute

- **Idea**: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”
- **Metric**: Impurity/Entropy (informal)
  - Measures the level of impurity in a group of examples





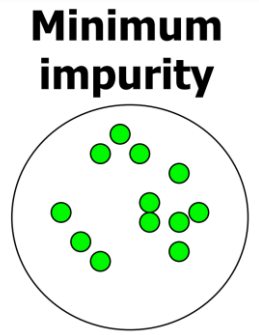


Entropy: a common way to measure impurity

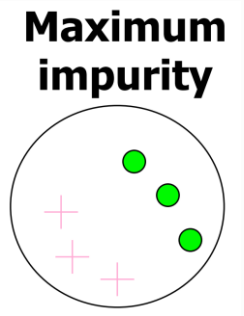
- Entropy $H(X)$ of a random variable X

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

- 2-Class Cases:
  - What is the entropy of a group in which all examples belong to the same class?
    - entropy = $-1\log_2 1 = 0$

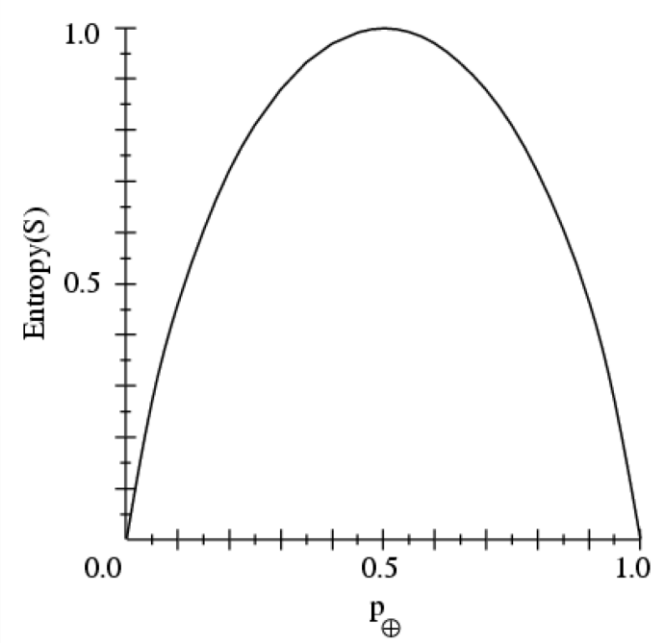


  - What is the entropy of a group with 50% in either class?
    - entropy = $-0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$



Entropy: a common way to measure impurity

- 2-Class Cases:



- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$H(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

Information Gain

- We want to determine which attribute in a given set of training feature vectors is most useful for discriminating between the classes to be learned
- Information gain tells us how important a given attribute of the feature vectors is
- We can use it to decide the ordering of attributes in the nodes of a decision tree (as in C4.5)

Entropy $H(X)$ of a random variable X:

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

Specific conditional entropy $H(X|Y=v)$ of X given $Y=v$:

$$H(X|Y=v) = -\sum_{i=1}^{n} P(X=i|Y=v)\log_2 P(X=i|Y=v)$$

Conditional entropy $H(X|Y)$ of X given Y:

$$H(X|Y) = \sum_{v \in values(Y)} P(Y=v)H(X|Y=v)$$

Mutual information (or Information Gain) of X and Y :

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

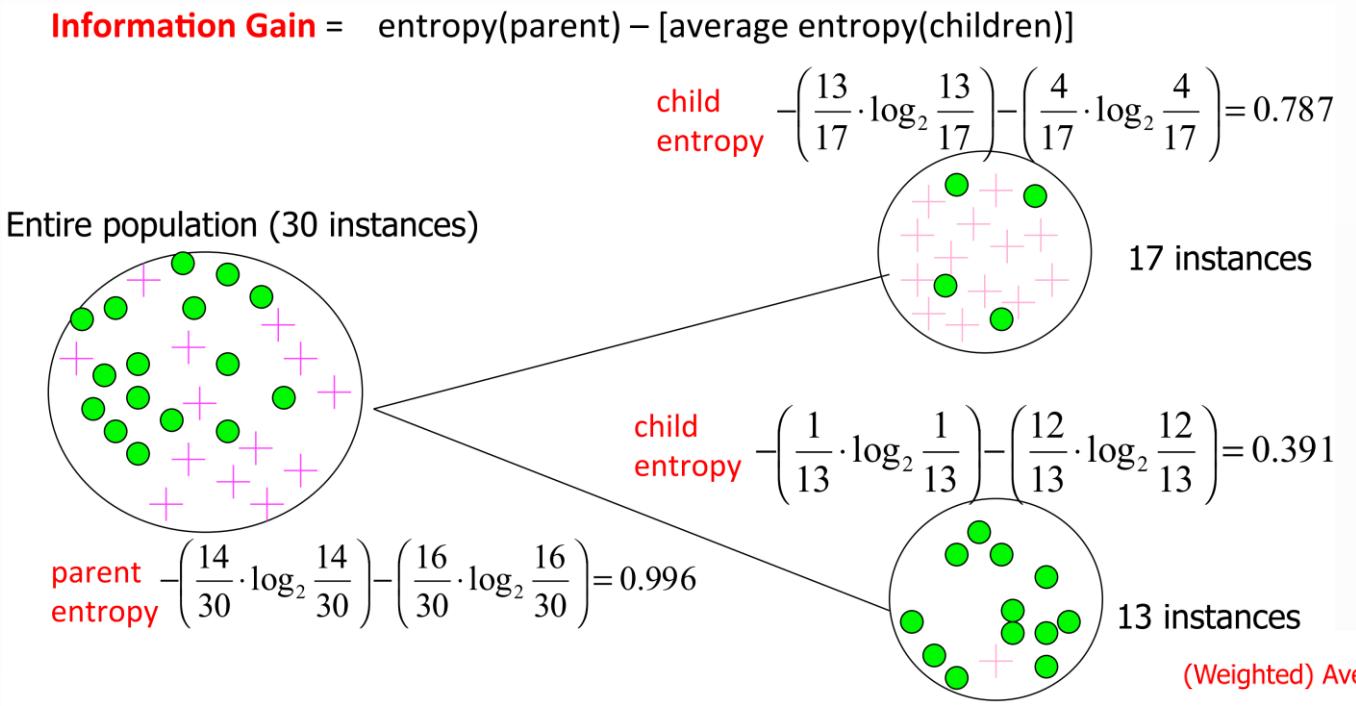
Information Gain

- Information Gain is the mutual information between input attribute A and target variable Y
  - Information Gain is the expected reduction in entropy of target variable Y for data sample S, due to sorting on variable A

$$Gain(S,A) = I_S(A,Y) = H_S(Y) - H_S(Y|A)$$

- An example for calculating information gain:

Information Gain = entropy(parent) – [average entropy(children)]



(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

Will the customer wait for a table?

- Back to the example with the restaurants:

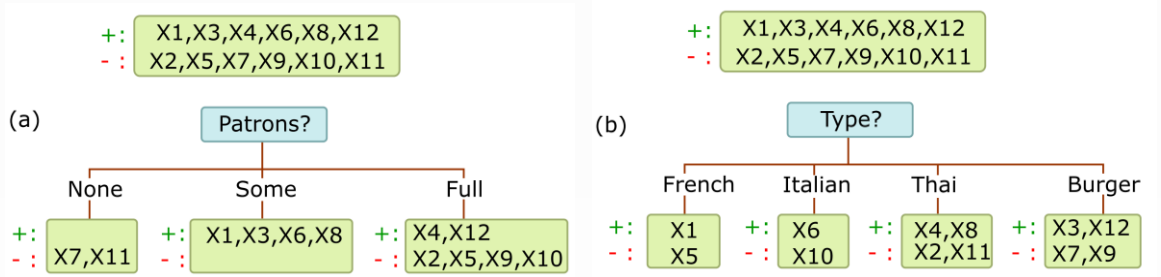
The Information Gain is calculated like this:

$$\begin{aligned}\text{Gain}(S, A) &= \text{expected reduction in entropy due to branching on attribute } A \\ &= \text{original entropy} - \text{entropy after branching}\end{aligned}$$

$$\text{Gain}(S, \text{Patrons}) = H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) - \left[\frac{2}{12}H([0,1]) + \frac{4}{12}H([1,0]) + \frac{6}{12}H\left(\left[\frac{2}{6}, \frac{4}{6}\right]\right)\right] \approx 0.541$$

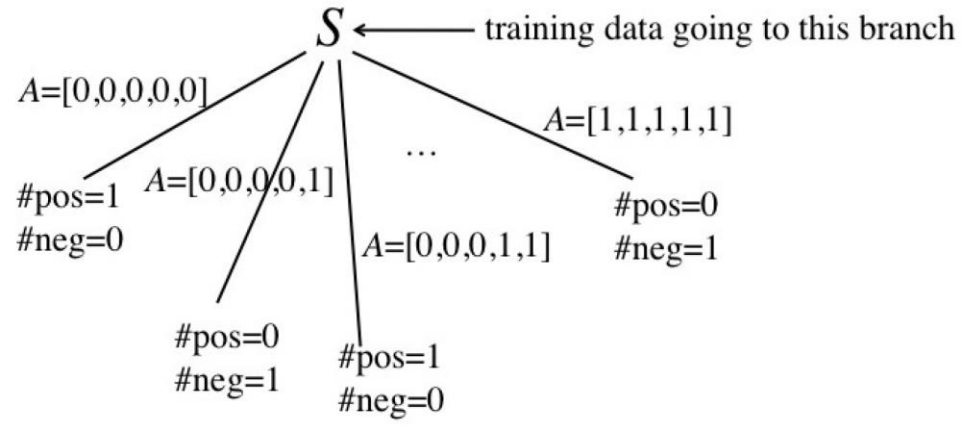
$$\text{Gain}(S, \text{Type}) = 1 - \left[\frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) + \frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right) + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right)\right] \approx 0$$

- Actually Patrons has the highest gain among the attributes, and is chosen to be the root of the tree
- In general, we want to choose the feature A that maximizes Gain(S, A)



Information Gain: A Comment

- One problem with Gain is that it likes to partition too much, and favors numerous splits: e.g., if each branch contains 1 example:



Then,

$$H\left[\frac{\#\text{pos}_j}{\#\text{pos}_j + \#\text{neg}_j}, \frac{\#\text{neg}_j}{\#\text{pos}_j + \#\text{neg}_j}\right] = 0 \text{ for all } j,$$

so all those negative terms would be zero and we'd choose that attribute over all the others.

- An alternative to Gain is the Gain Ratio

Gain Ratio

- We want to have a large Gain, but also we want small partitions. We'll choose our attribute according to:

$$\frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)} \begin{array}{l} \leftarrow \text{ want large} \\ \leftarrow \text{ want small} \end{array}$$

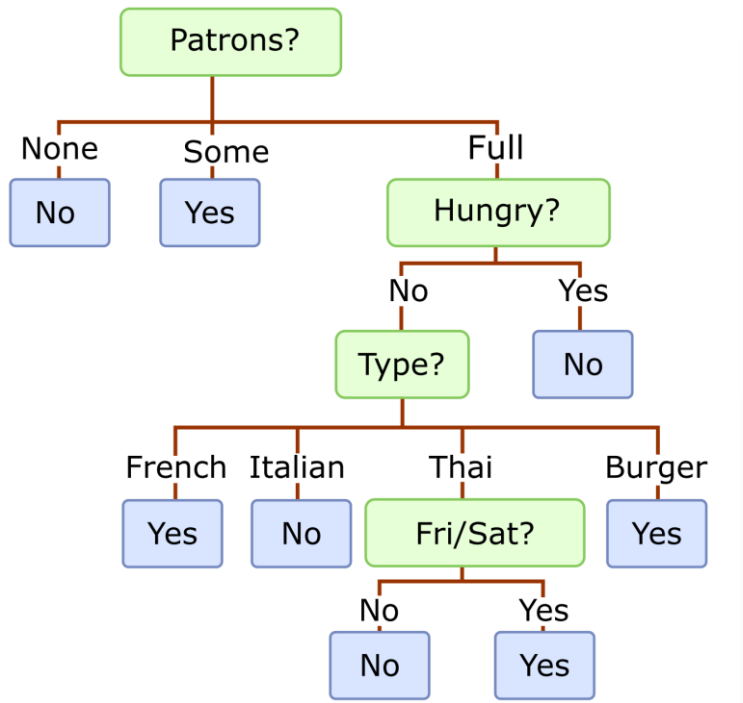
where SplitInfo(S, A) comes from the partition:

$$\text{SplitInfo}(S, A) = -\sum_{j=1}^{J} \frac{|S_j|}{|S|} \log\left(\frac{|S_j|}{|S|}\right)$$

where $|S_j|$ is the number of examples in branch j. We want each term in the S sum to be large. That means we want $\frac{|S_j|}{|S|}$ to be large, meaning that we want lots of examples in each branch.

Will the customer wait for a table?

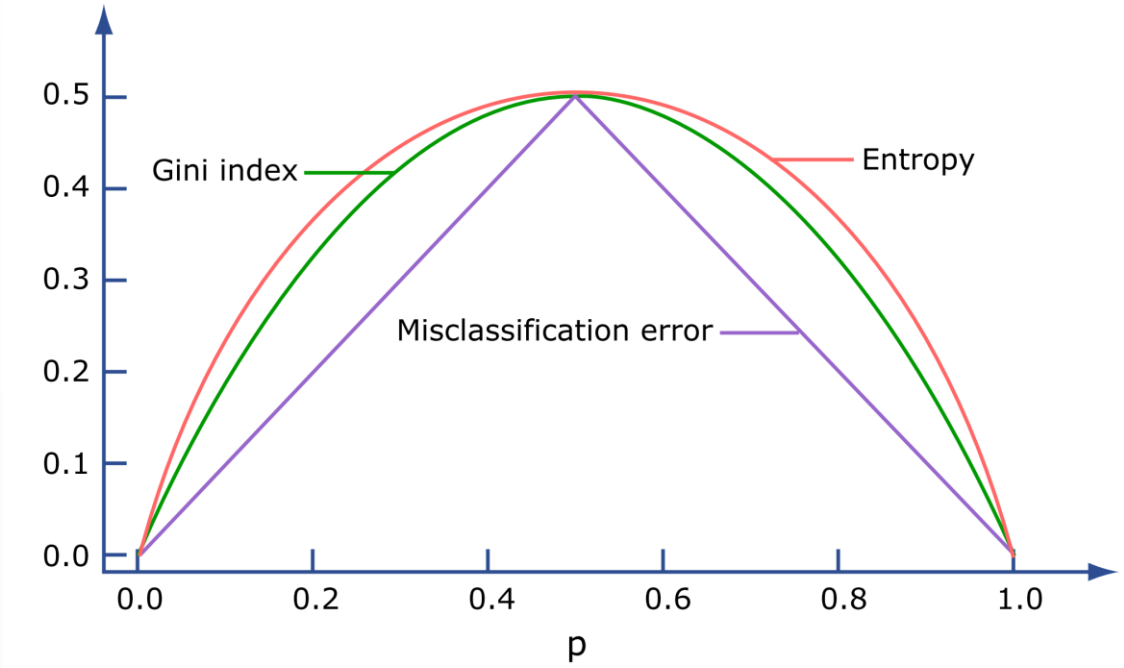
- Keep splitting until:
  - no more examples left (no point trying to split)
  - all examples have the same class
  - no more attributes to split
- For the restaurant example, we get tree in Figure
- There are possibilities to replace H([p, 1 − p]), it is not the only thing we can use!
  - One example is the Gini index 2p(1 − p) used by CART.
  - Another example is the value 1 − max(p, 1 − p)



The decision tree induced from the 12-example training set.

Alternative Metrics for Attribute Selection in Classification Trees

- C4.5 uses information gain for splitting, and CART uses the Gini index. (CART only has binary splits.)



Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through (0.5, 0.5).

- All three are similar, but cross-entropy and the Gini index are differentiable, and hence more amenable to numerical optimization.

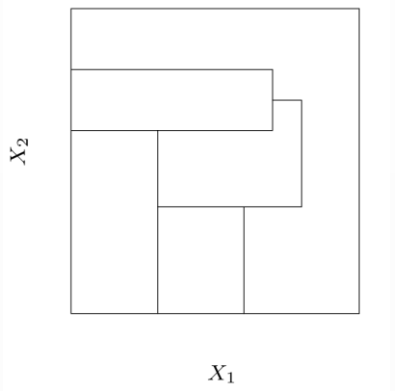
Misclassification error: $\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \max_k(\hat{p}_{mk})$

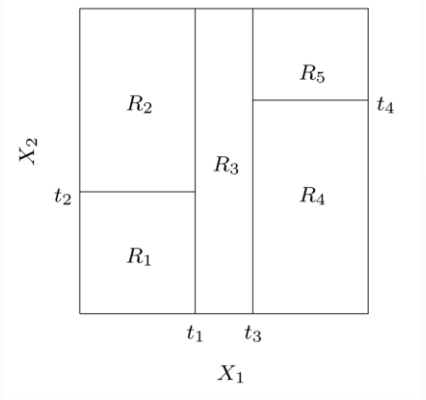
Gini index: $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).$

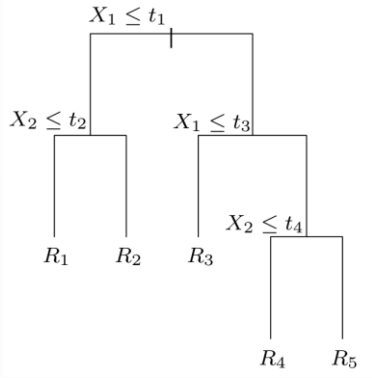
Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.$

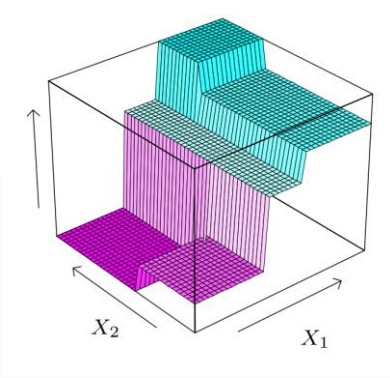
CART (Classification and Regression Tree)

- It uses recursive binary partitions like that in the second figure below
  - First split the space into two regions, and model the response by the mean of Y in each region,
  - Choose the variable and split-point to achieve the best fit,
  - Then one or both of these regions are split into two more regions,
  - And this process is continued, until some stopping rule is applied.









- Partitions and CART.
  - Second figure shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data
  - First figure shows a general partition that cannot be obtained from recursive binary splitting
  - Third figure shows the tree corresponding to the partition in the second figure, and
  - A perspective plot of the prediction surface appears in the last figure

- Note that multiway splits can be achieved by a series of binary splits

CART – Regression Task

- The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have
- Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

- If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m)$$

CART – Regression Task

- Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm.
- Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}$$

- Then we seek the splitting variable j and split point s that solve

$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right]$$

- For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i|x_i \in R_2(j,s))$$

  - The split point s can be determined very quickly. And hence by scanning through all of the inputs, determination of the best pair (j,s) is feasible

- Then this process is repeated on all of the resulting regions until completing the tree

CART – Regression Task: Pruning

- How large should we grow the tree?
  - Clearly a very large tree might overfit the data, while a small tree might not capture the important structure
  - Tree size is a tuning parameter governing the model's complexity
- One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold
  - This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it
- The preferred strategy is to grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using *cost-complexity pruning*, which we now describe

CART – Regression Task: Pruning

- We define a subtree $T \subseteq T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes
- We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$N_m = \#\{x_i \in R_m\},$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i,$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$

We define the cost complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

CART – Regression Task: Pruning

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

- The idea is to find, for each α, the subtree $T \subseteq T_0$ to minimize $C_\alpha(T)$
- The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data
  - Large values of α result in smaller trees T_α , and conversely for smaller values of α
  - As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0
- Estimation of α can be achieved by five- or tenfold cross-validation:
  - We select an optimal value $\hat{\alpha}$ to minimize the cross-validated sum of squares
  - We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$. Our final tree is $T_{\hat{\alpha}}$

CART – Classification Task

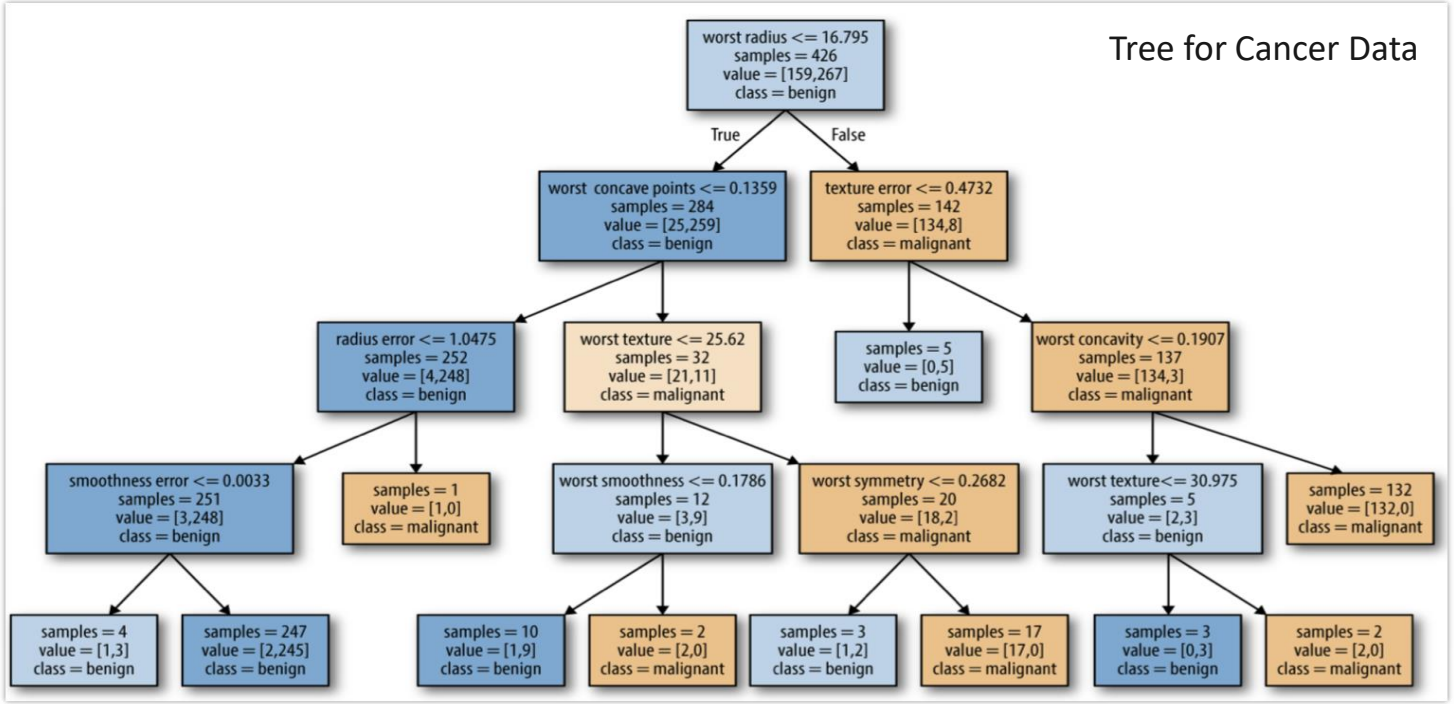
- If the target is a classification outcome taking values $1, 2, \ldots, K$, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes and pruning the tree.
- As presented before, an impurity measure should be used

$$
\begin{aligned}
&\text{Misclassification error:} && \frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.\\
&\text{Gini index:} && \sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).\\
&\text{Cross-entropy or deviance:} && -\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.
\end{aligned}
$$

- The cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. For this reason, either the Gini index or cross-entropy should be used when growing the tree.
- To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate.

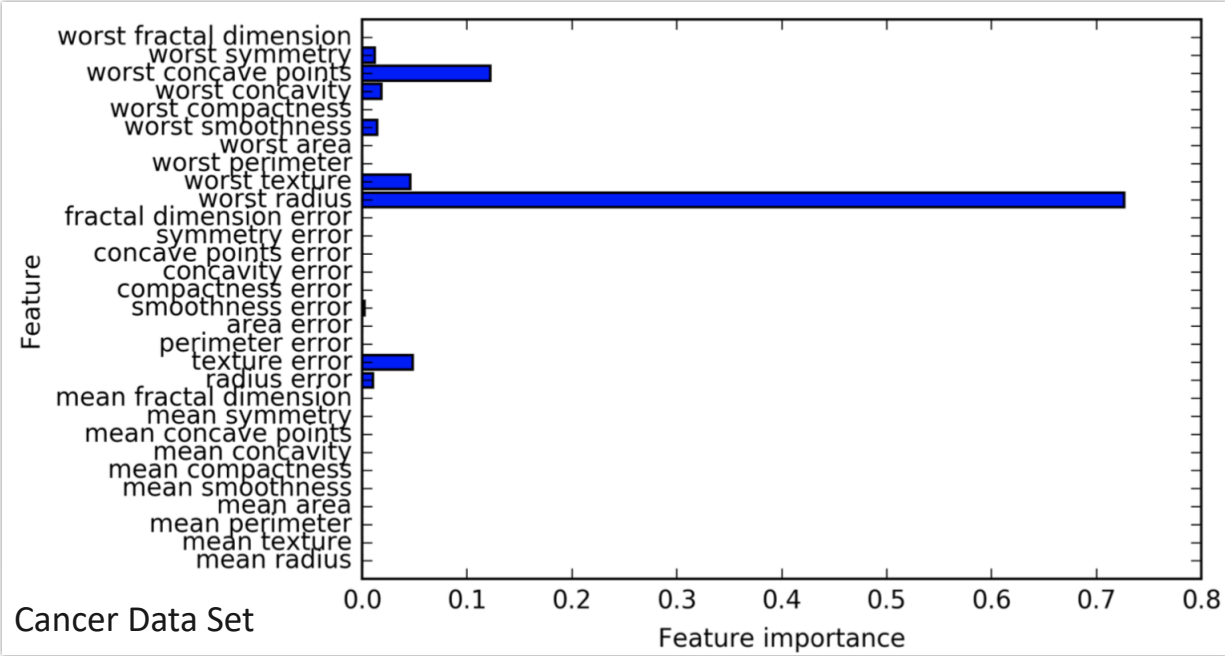
Visualizing the Decision Tree

- The visualization of the tree provides a great in-depth view of how the algorithm makes predictions, and is a good example of a machine learning algorithm that is easily explained to nonexperts.
- However, even with a tree of depth four, as seen here, the tree can become a bit overwhelming.



Feature Importance

- Instead of looking at the whole tree, which can be taxing, there are some useful properties that we can derive to summarize the workings of the tree.
- The most commonly used summary is *feature importance*, which rates how important each feature is for the decision a tree makes.
- It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.” The feature importances always sum to 1:



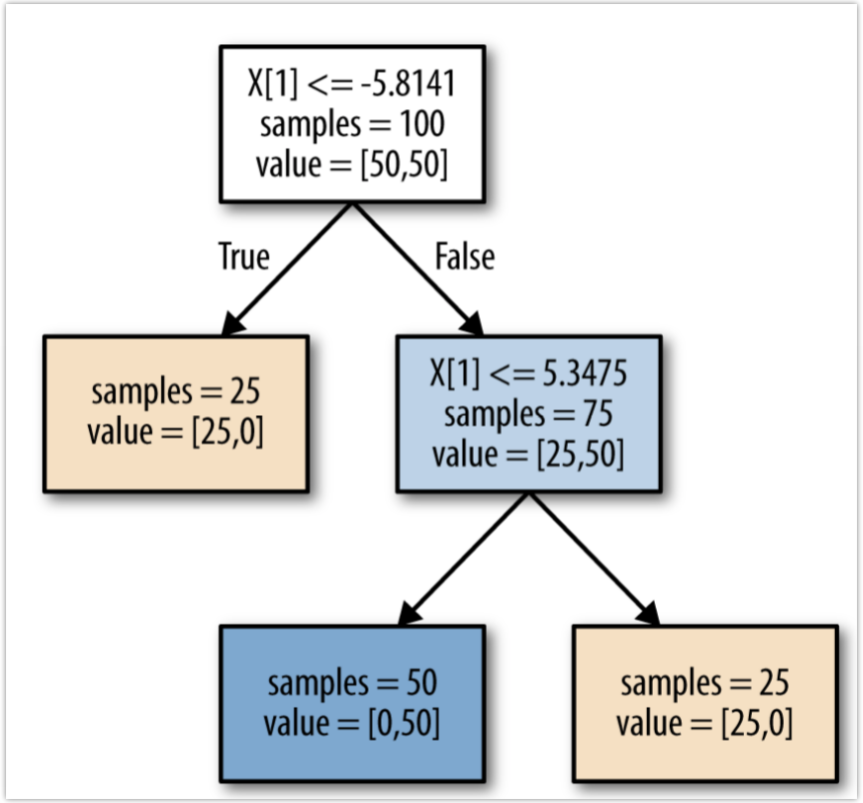
Cancer Data Set

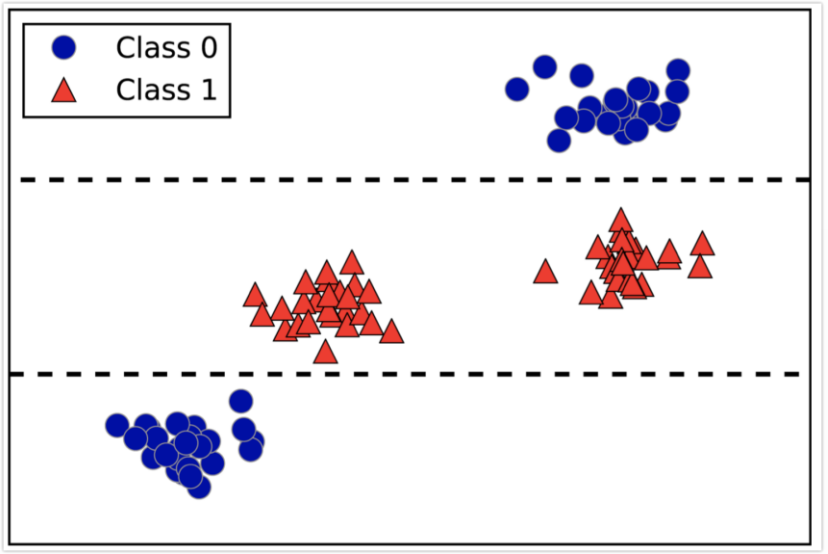
Feature Importance

- Here we see that the feature used in the top split (“worst radius”) is by far the most important feature. This confirms our observation in analyzing the tree that the first level already separates the two classes fairly well.
- However, if a feature has a low “feature_importance”, it doesn’t mean that this feature is uninformative.
  - It only means that the feature was not picked by the tree, likely because another feature encodes the same information.
- In contrast to the coefficients in linear models, feature importances are always positive, and don’t encode which class a feature is indicative of.
  - The feature importances tell us that “worst radius” is important, but not whether a high radius is indicative of a sample being benign or malignant.
  - In fact, there might not be such a simple relationship between features and class

Feature Importance

- The plot shows a dataset with two features and two classes. Here, all the information is contained in X[1], and X[0] is not used at all.





- X[0] is not used in the model, but we cannot conclude that it has no information or it is not important. A high accuracy model can also be obtained by including X[0].

- The relation between X[1] and the output class is not monotonous, meaning we cannot say “a high value of X[1] means class 0, and a low value means class 1” (or vice versa).

Advantages of Decision Trees

- The resulting model can easily be visualized and understood by nonexperts (at least for smaller trees).
- The algorithms are completely invariant to scaling of the data.
  - As each feature is processed separately, and the possible splits of the data don’t depend on scaling,
  - No preprocessing like normalization or standardization of features is needed for decision tree algorithms.
  - In particular, decision trees work well when you have features that are on completely different scales, or a mix of binary and continuous features.
  - Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Disadvantages of Decision Trees

- The main downside of decision trees is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.
- Therefore, in most applications, the ensemble methods are usually used in place of a single decision tree.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
  - Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node.
  - Such algorithms cannot guarantee to return the globally optimal decision tree.
  - This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Ensemble of Decision Trees

- Ensembles are methods that combine multiple machine learning models to create more powerful models
- There are many models in the machine learning literature that belong to this category,
- But there are two ensemble models that have proven to be effective on a wide range of datasets for classification and regression, both of which use decision trees as their building blocks:
  - Random forests
  - Gradient boosted decision trees

Random Forest

- As we just observed, a main drawback of decision trees is that they tend to overfit the training data.
- Random forests are one way to address this problem. **A random forest is essentially a collection of decision trees, where each tree is slightly different from the others.**
- The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.
- If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. This reduction in overfitting, while retaining the predictive power of the trees, can be shown using rigorous mathematics.
- To implement this strategy, we need to build many decision trees. Each tree should do an acceptable job of predicting the target, and should also be different from the other trees.
- Random forests get their name from injecting randomness into the tree building to ensure each tree is different. There are two ways in which the trees in a random forest are randomized:
  - by selecting the data points used to build a tree
  - by selecting the features in each split test

Building Random Forests

- To build a random forest model, you need to decide on the number of trees to build
  - Let's say we want to build 10 trees
  - These trees will be built completely independently from each other, and
  - The algorithm will make different random choices for each tree to make sure the trees are distinct
- To build a tree, we first take what is called a bootstrap sample of our data. That is, from our n data points, we repeatedly draw an example randomly with replacement (meaning the same sample can be picked multiple times), n times.
- This will create a data-set that is as big as the original dataset, but some data points will be missing from it (approximately one third), and some will be repeated.

To illustrate, let's say we want to create a bootstrap sample of the list ['a', 'b', 'c', 'd'].

- A possible bootstrap sample would be ['b', 'd', 'd', 'c'].
- Another possible sample would be ['d', 'a', 'd', 'a'].

Building Random Forests

- Next, a decision tree is built based on this newly created dataset. However, the algorithm is described for the decision tree is slightly modified.
- Instead of looking for the best test for each node, in each node the algorithm randomly selects a subset of the features, and it looks for the best possible test involving one of these features.
- The number of features that are selected is controlled by the "*max_features*" parameter.
- This selection of a subset of features is repeated separately in each node, so that each node in a tree can make a decision using a different subset of the features.

  - The bootstrap sampling leads to each decision tree in the random forest being built on a slightly different dataset.
  - Because of the selection of features in each node, each split in each tree operates on a different subset of features.
  - Together, these two mechanisms ensure that all the trees in the random forest are different.

Building Random Forests

- A critical parameter in this process is "*max_features*":
  - If we set "*max_features*" to "*n_features*", that means that each split can look at all features in the dataset, and no randomness will be injected in the feature selection (the randomness due to the bootstrapping remains, though).
  - If we set "*max_features*" to 1, that means that the splits have no choice at all on which feature to test, and can only search over different thresholds for the feature that was selected randomly.
    - Therefore, a high "*max_features*" means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features.
    - A low "*max_features*" means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.
- To make a prediction using the random forest, the algorithm first makes a prediction for every tree in the forest.
- For regression, we can average these results to get our final prediction.
- For classification, a "soft voting" strategy is used. This means each tree makes a "soft" prediction, providing a probability for each possible output label. The probabilities predicted by all the trees are averaged, and the class with the highest probability is predicted.

Random Forest Pseudocode

Random Forest for Regression or Classification.

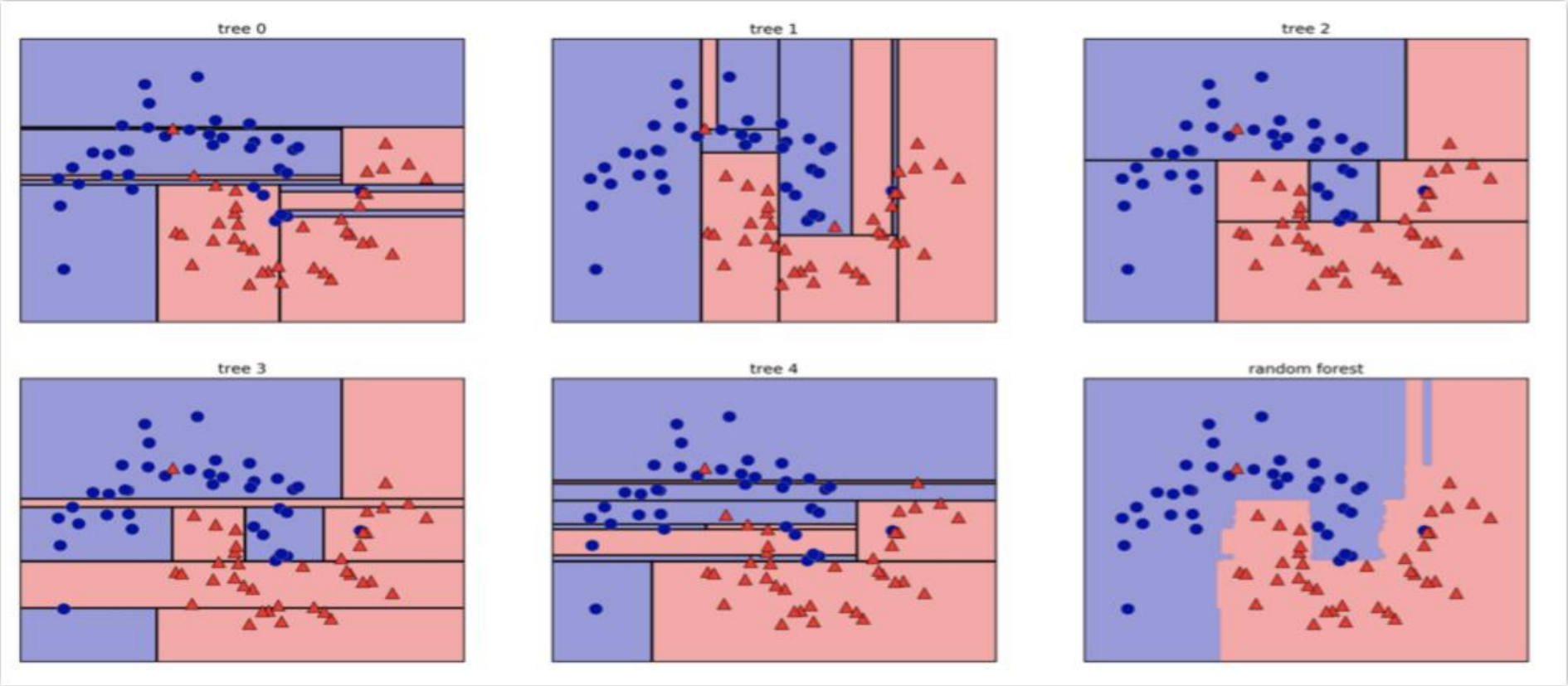
1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

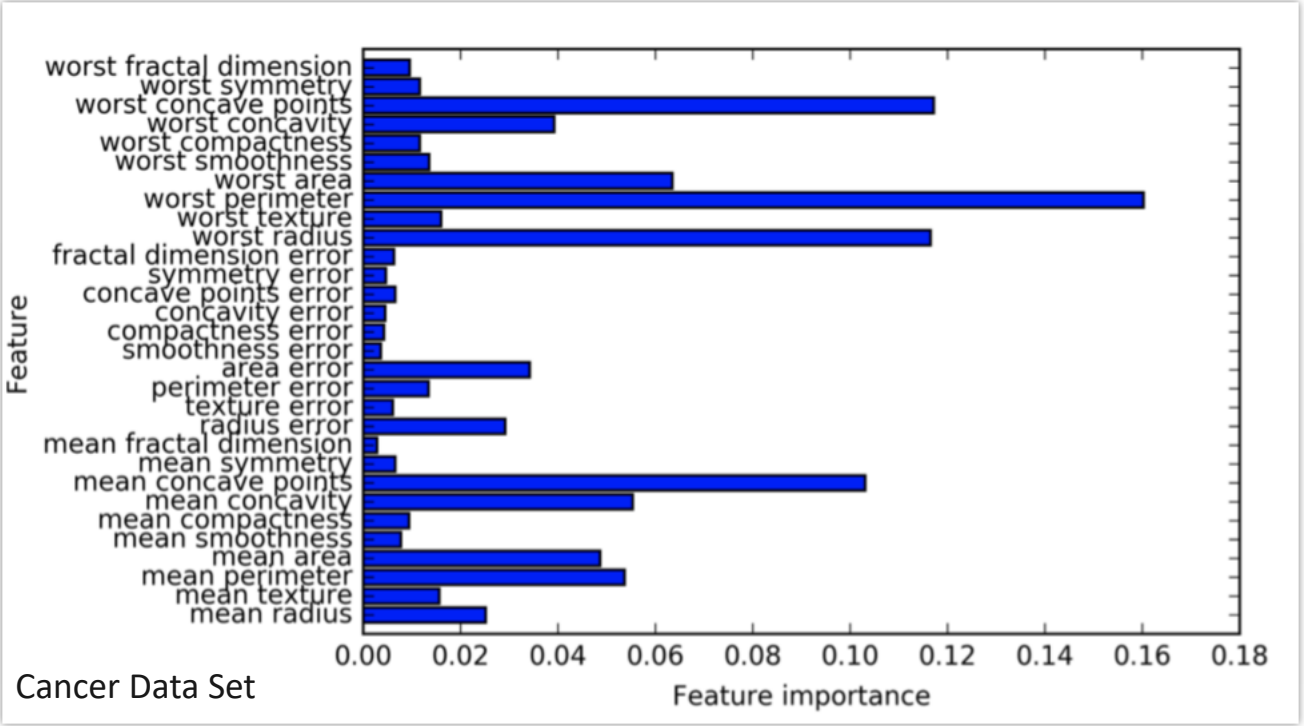
Two Moons Data



- The random forest overfits less than any of the trees individually, and provides a much more intuitive decision boundary.
- In any real application, it is used many more trees (often hundreds or thousands), leading to even smoother boundaries.

Feature Importance

- Similarly to the decision tree, the random forest provides feature importances, which are computed by aggregating the feature importances over the trees in the forest.
- Typically, the feature importances provided by the random forest are more reliable than the ones provided by a single tree.



Cancer Data Set

- As you can see, the random forest gives nonzero importance to many more features than the single tree.
- Similarly to the single decision tree, the random forest also gives a lot of importance to the “worst radius” feature, but it actually chooses “worst perimeter” to be the most informative feature overall.
- The randomness in building the random forest forces the algorithm to consider many possible explanations, the result being that the random forest captures a much broader picture of the data than a single tree.

Strengths, Weaknesses, and Parameters

- Random forests for regression and classification are currently among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data.
- Essentially, random forests share all of the benefits of decision trees, while making up for some of their deficiencies. One reason to still use decision trees is if you need a compact representation of the decision-making process.
- It is basically impossible to interpret tens or hundreds of trees in detail, and trees in random forests tend to be deeper than decision trees (because of the use of feature subsets). Therefore, if you need to summarize the prediction making in a visual way to nonexperts, a single decision tree might be a better choice.
- While building random forests on large data- sets might be somewhat time consuming, it can be parallelized across multiple CPU cores within a computer easily.
- You should keep in mind that random forests, by their nature, are random, and setting different random states (or not setting the random_state at all) can drastically change the model that is built.
- The more trees there are in the forest, the more robust it will be against the choice of random state. If you want to have reproducible results, it is important to fix the random_state.

Strengths, Weaknesses, and Parameters

- Random forests don't tend to perform well on very high dimensional, sparse data, such as text data. For this kind of data, linear models might be more appropriate.
- Random forests require more memory and are slower to train and to predict than linear models. If time and memory are important in an application, it might make sense to use a linear model instead
- The important parameters to adjust are "*n_estimators*", "*max_features*", and possibly pre-pruning options like "*max_depth*".
- For "*n_estimators*", larger is always better. Averaging more trees will yield a more robust ensemble by reducing overfitting.
  - However, there are diminishing returns, and more trees need more memory and more time to train.
- "*max_features*" determines how random each tree is, and a smaller "*max_features*" reduces overfitting.
- In general, it's a good rule of thumb to use the default values:
  - *max_features*=$\sqrt{n\_features}$ for classification and *max_features*=$\log_2(n\_features)$ for regression.
  - Adding "*max_features*" or "*max_leaf_nodes*" might sometimes improve performance. It can also drastically reduce space and time requirements for training and prediction.

References

- C. Rudin, Prediction: Machine Learning And Statistics, Lecture Notes.
- T. Hastie and R. Tibshirani, Statistical Learning, Lecture Note.
- E. Eaton, Decision Trees, Lecture Notes.
- J. Friedman, T. Hastie, and R. Tibshirani, “The Elements of Statistical Learning”, 2008.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

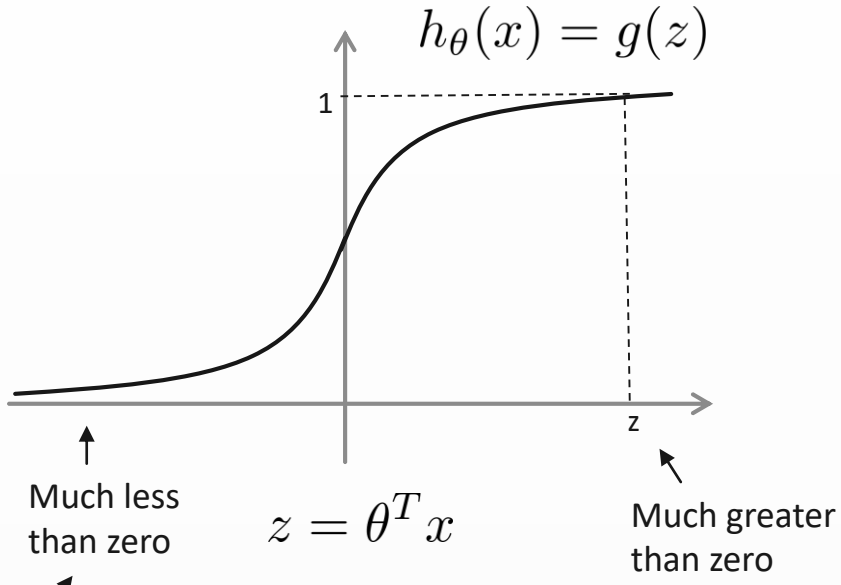
LECTURE 7

SUPPORT VECTOR MACHINES

Instructor: Asst. Prof. Barış Başpınar

Alternative View of Logistic Regression

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

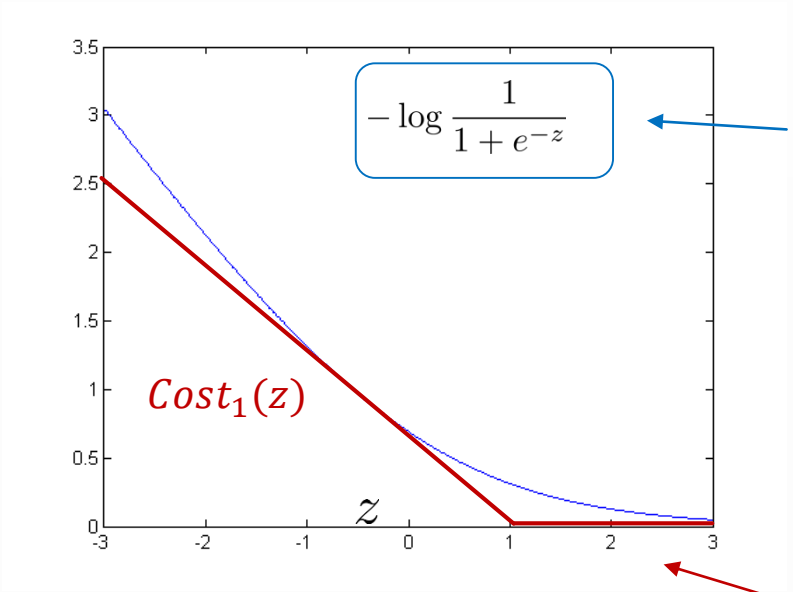
If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

Alternative View of Logistic Regression

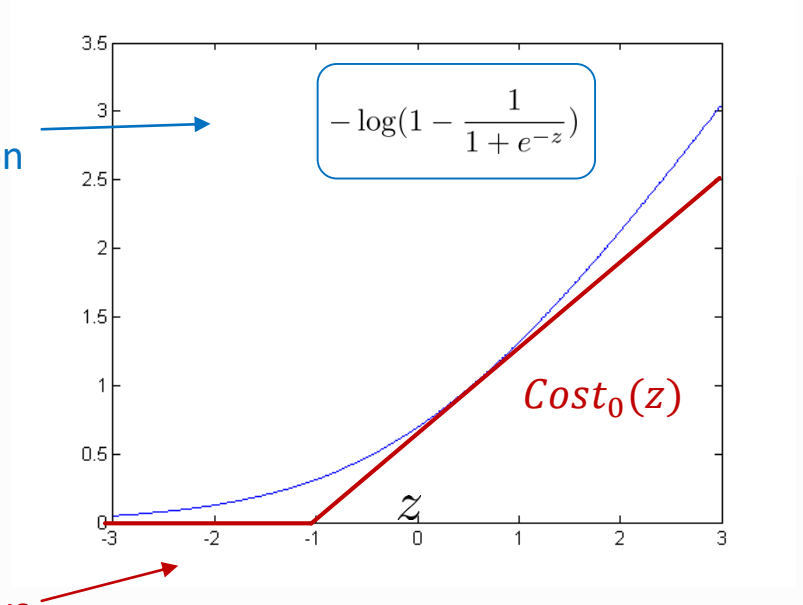
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log(1 - \frac{1}{1+e^{-\theta^T x}})$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



Logistic Regression

Alternative

Support Vector Machine

Logistic regression:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \left(-\log h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \left((-\log(1 - h_\theta(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

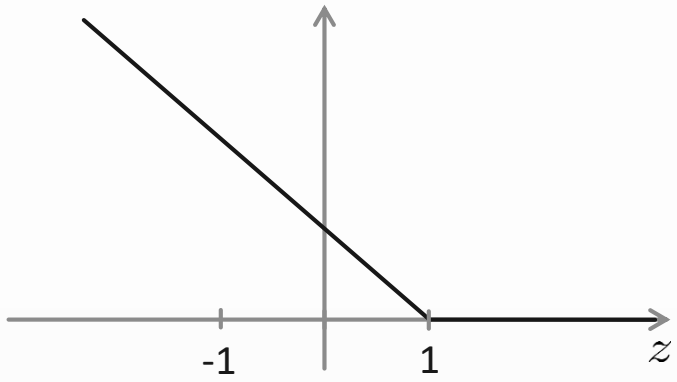
Support vector machine:

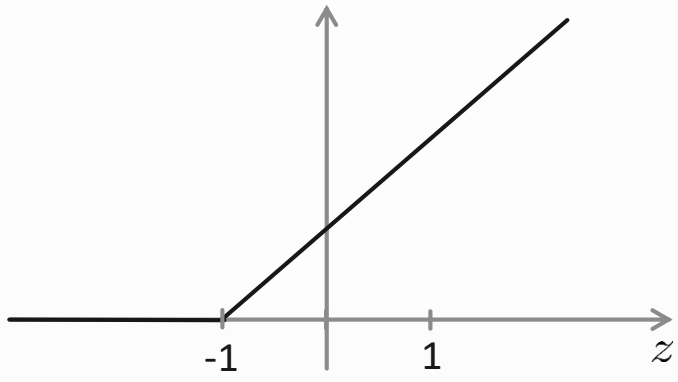
$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

C → controlling the cost trade-off

SVM Hypothesis: Large Margin Intuition

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$





If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

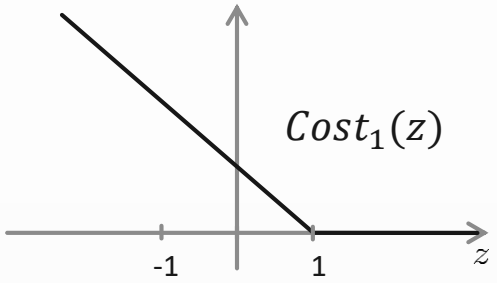
SVM Decision Boundary

$$\min_\theta C \sum_{i=1}^{m} \underbrace{\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right]}_{= 0} + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Assume
a very large value is chosen

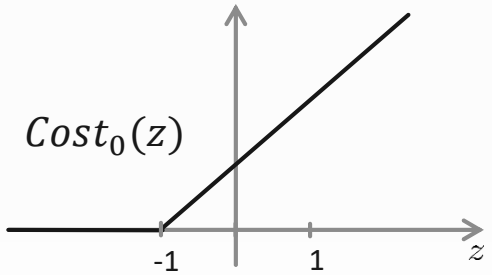
Whenever $y^{(i)} = 1$:

$$\theta^T x \geq 1$$



Whenever $y^{(i)} = 0$:

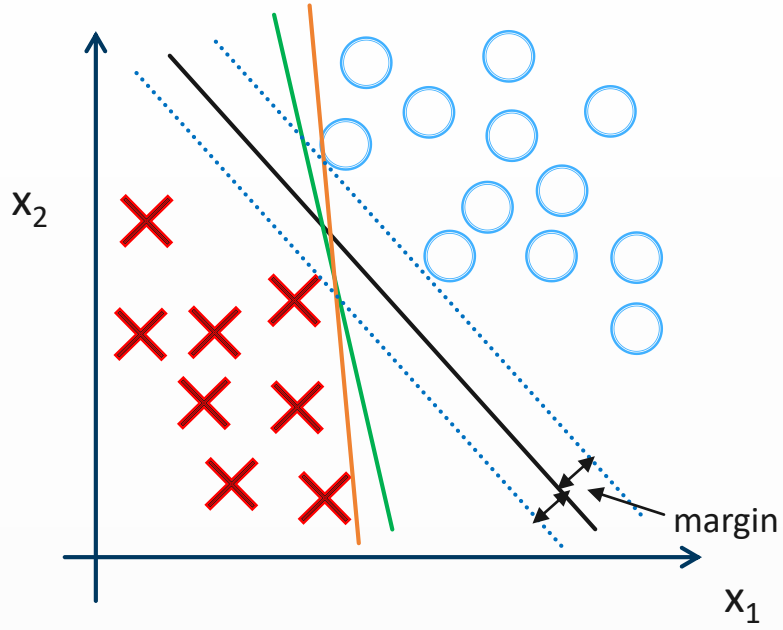
$$\theta^T x \leq -1$$



$$\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

$$\text{s.t.} \quad \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

SVM Decision Boundary: Linearly separable case

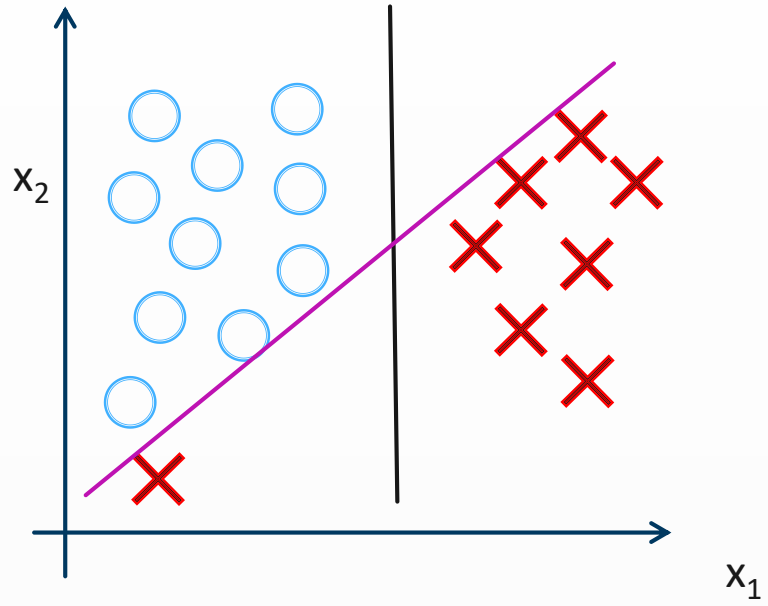


Large margin classifier

- There are many linear decision boundaries that separate the classes
  - But, many of them are not particularly good choices

- The black one (SVM decision boundary) is a more robust separator
  - Mathematically, it has larger margins (or larger minimum distance from any of my training examples)

Large margin classifier in presence of outliers



C is very large

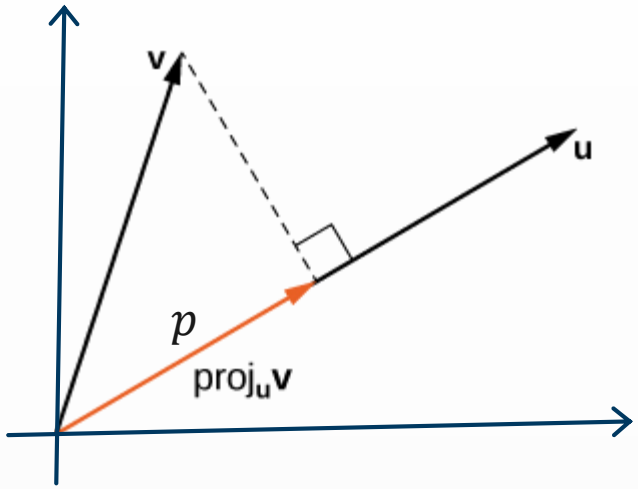
C is not too large

It is a regularization parameter

- If C is very large, SVM will be too sensitive to outliers
- Using not too large values, it will manage to generate a robust decision boundary such as black one

The mathematics behind large margin classification

Vector Inner Product



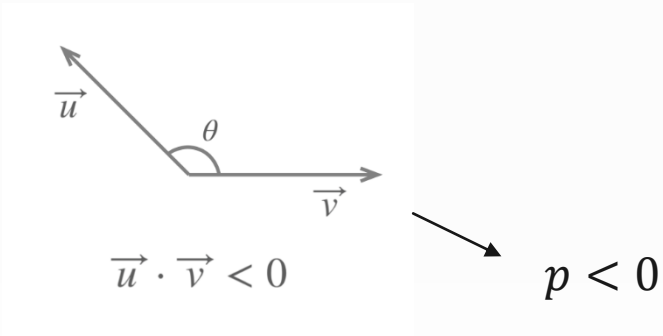
$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

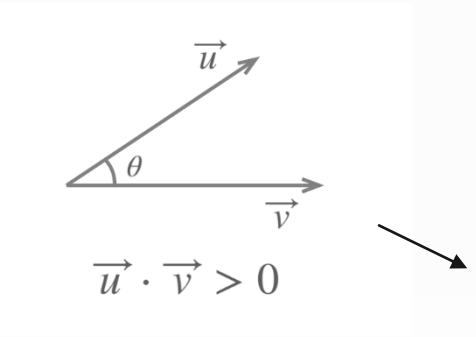
$$u^T v = ?$$

p = (signed) length of projection of vector v onto vector u

$\|u\|$ = length of vector u $\rightarrow$ $\|u\| = \sqrt{u_1^2 + u_2^2}$

$$u^T v = p\,\|u\| \qquad u_1 v_1 + u_2 v_2 = p\,\|u\|$$





The mathematics behind large margin classification

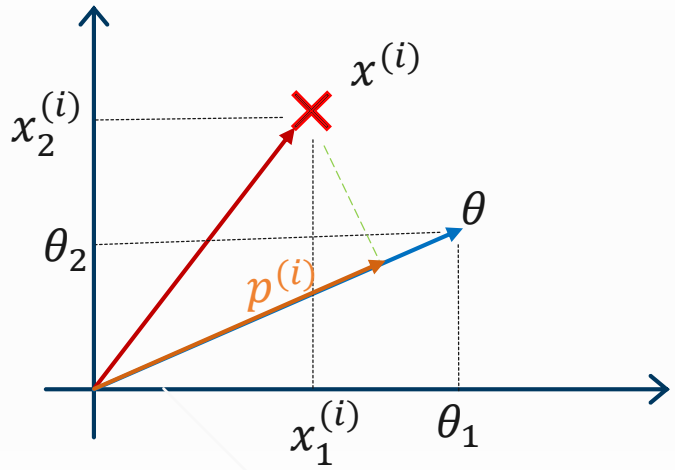
SVM Decision Boundary

$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2+\theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

$$\text{s.t.} \quad \theta^T x^{(i)} \geq 1 \qquad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \qquad \text{if } y^{(i)} = 0$$

Simplification $\theta_0 = 0,\ n = 2$



$$\theta^T x^{(i)} = p^{(i)}\|\theta\|$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

Constraints can be presented in terms of p

The mathematics behind large margin classification

SVM Decision Boundary

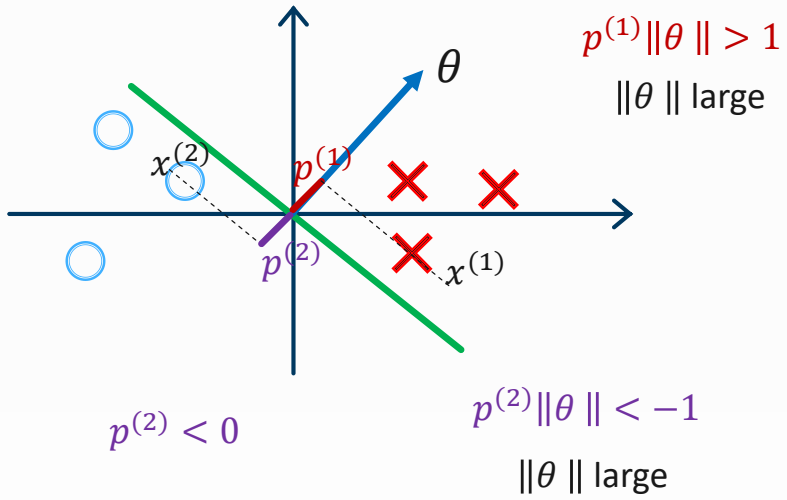
$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

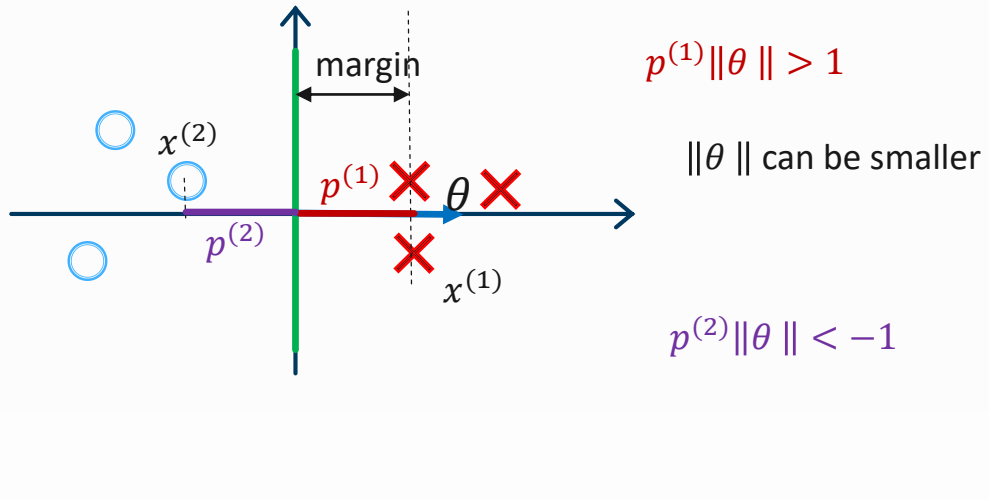
$$\text{s.t.} \quad p^{(i)} \cdot \|\theta\| \geq 1 \qquad \text{if } y^{(i)} = 1$$

$$p^{(i)} \cdot \|\theta\| \leq -1 \qquad \text{if } y^{(i)} = 0$$

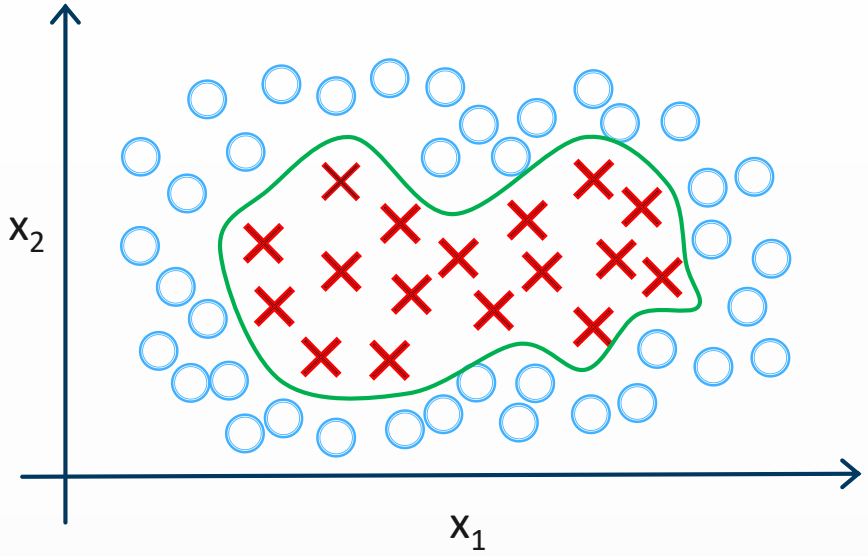
where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.

Simplification: $\theta_0 = 0$





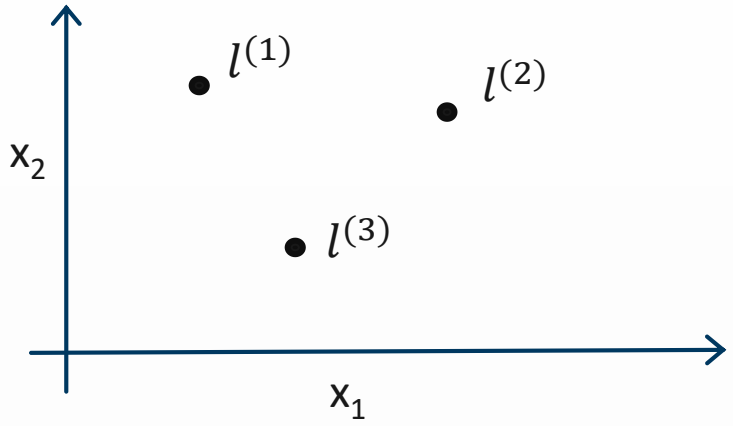
Non-linear Decision Boundary



$$\text{Predict } y = 1 \text{ if}$$
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Kernels



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

Given x:

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(2)})^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(3)})^2}{2\sigma^2}\right)$$

Kernel

Gaussian Kernel

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

If x if far from $l^{(1)}$:

$$f_1 \approx exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$$

For each landmark, a new feature:

$l^{(1)} \rightarrow f_1$

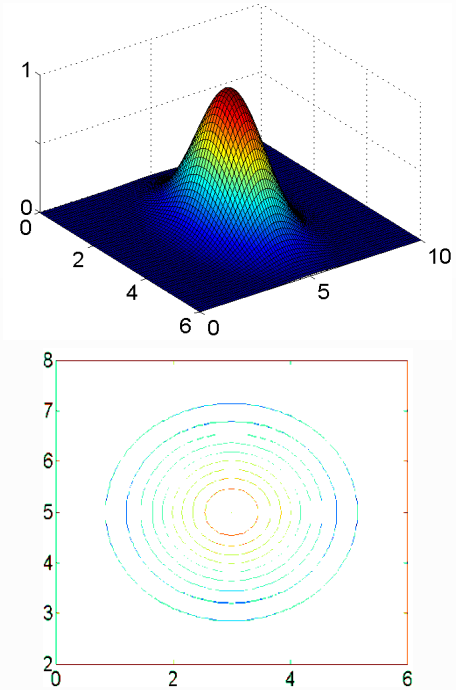
$l^{(2)} \rightarrow f_2$

$l^{(3)} \rightarrow f_3$

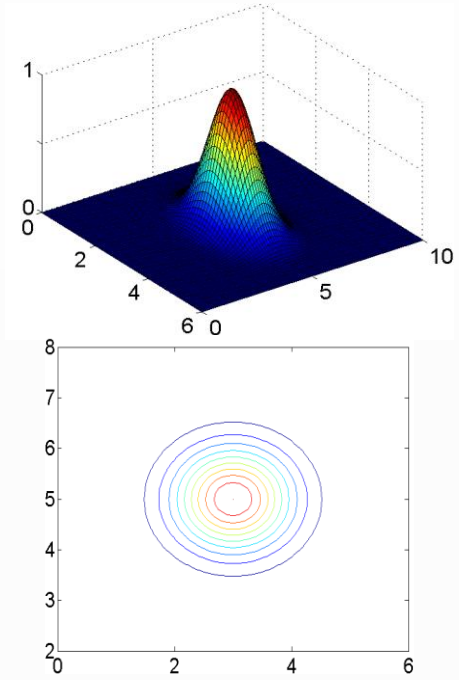
Kernels and Similarity

Example: $l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

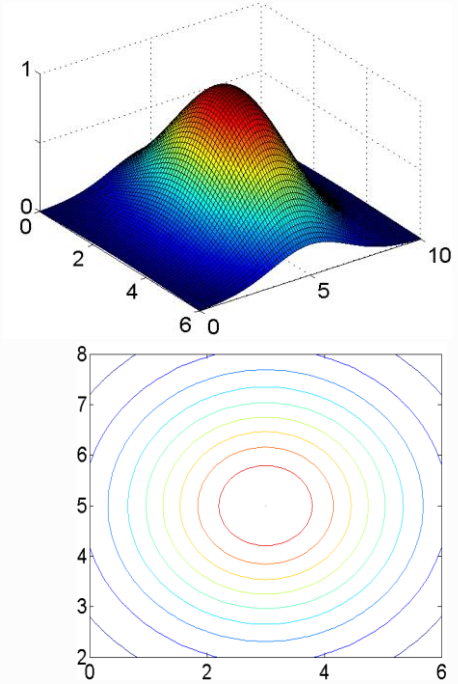
$\sigma^2 = 1$



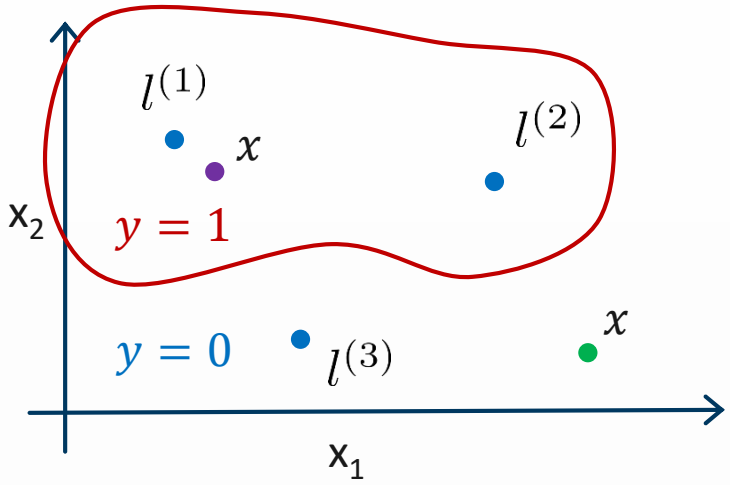
$\sigma^2 = 0.5$



$\sigma^2 = 3$



Kernels and Similarity



Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Assume:

$\theta_0 = -0.5, \quad \theta_1 = 1, \quad \theta_2 = 1, \quad \theta_3 = 0$

$f_1 \approx 1, \quad f_2 \approx 0, \quad f_3 \approx 0$

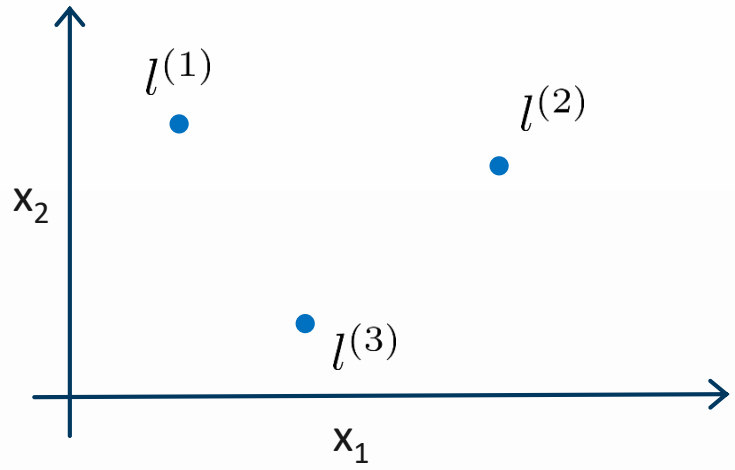
$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0 = 0.5 \geq 0 \quad \rightarrow y = 1$

$f_1 \approx 0, \quad f_2 \approx 0, \quad f_3 \approx 0$

$\theta_0 + \theta_1 \times 0 + \theta_2 \times 0 + \theta_3 \times 0 = -0.5 < 0 \quad \rightarrow y = 0$

We can learn pretty complex non-linear decision boundaries

Choosing the Landmarks

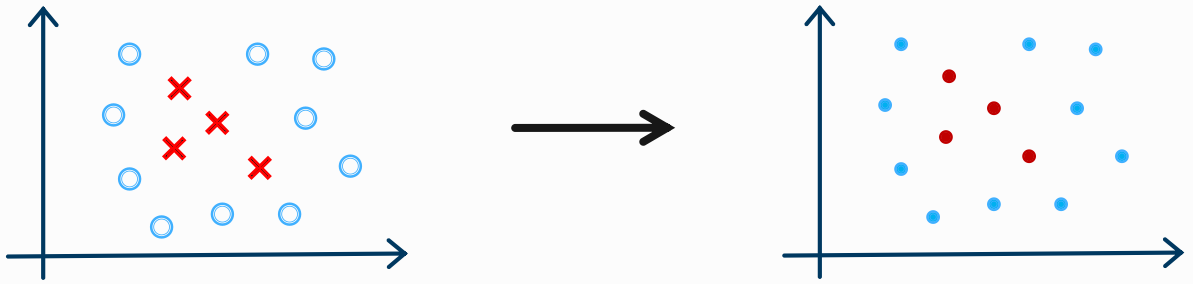


Given x:

$$f_i = \text{similarity}(x, l^{(i)})$$
$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,
choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$\ldots$$

For training example $(x^{(i)}, y^{(i)})$:

$$x^i \rightarrow \begin{array}{l} f_1^{(i)} = sim(x^i, l^1) \\ f_2^{(i)} = sim(x^i, l^2) \\ \vdots \\ f_m^{(i)} = sim(x^i, l^m) \end{array}$$

$$f^i = \begin{bmatrix} f_0^i \\ f_1^i \\ f_2^i \\ \vdots \\ f_m^i \end{bmatrix}$$

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

Predict "y=1" if $\theta^T f \geq 0$

Training:

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

$n = m$

~~$\theta^T x^i$~~

Used in this form to improve scalability

$\theta^T M \theta$

- The optimization problem that the SVM has is a convex opt. problem.
  - You don't need to worry about local optima.

- We can apply kernel idea trick for other algorithms like logistic regression,
  - But the computational tricks that apply for SVM don't generalize well to other algorithms
  - Using kernels with logistic regression will be very slow

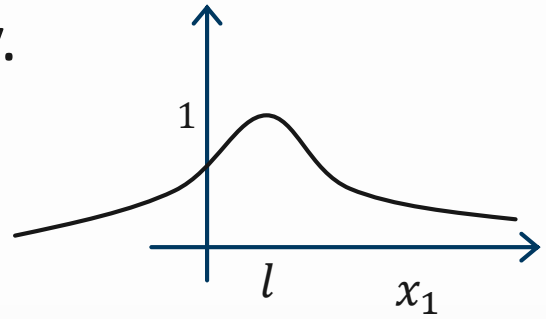
SVM Parameters

C ($= \frac{1}{\lambda}$). Large C: Lower bias, high variance. (Small λ)

Small C: Higher bias, low variance. (Large λ)

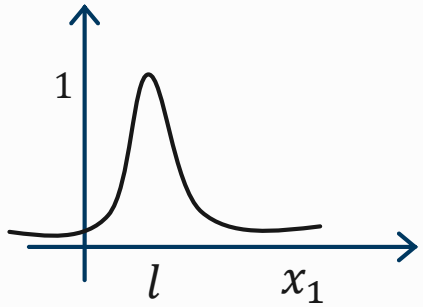
σ^2 Large σ^2: Features f_i vary more smoothly.

Higher bias, lower variance.



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



Using an SVM

Use SVM software package (e.g. liblinear, libsvm, …) to solve for parameters θ.

Need to specify:

Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel (“linear kernel”)

Predict “y = 1” if $\theta^T x \geq 0$

Gaussian kernel:

$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

Using an SVM

Kernel (similarity) functions:

$x^{(i)}$ $l^{(j)}$

```
function f = kernel(x1,x2)
```

$$f = \exp\left(-\frac{||\,\mathbf{x1} - \mathbf{x2}\,||^2}{2\sigma^2}\right)$$

```
return
```

Note: Do perform feature scaling before using the Gaussian kernel.

$$\|x - l\|^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$$

House pricing example, 1000 ft^2 1-5 bedrooms

This can dominate, perform feature scaling

Other Choices of Kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels. (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:

- Polynomial kernel: $(x^T l + constant)^{degree}$

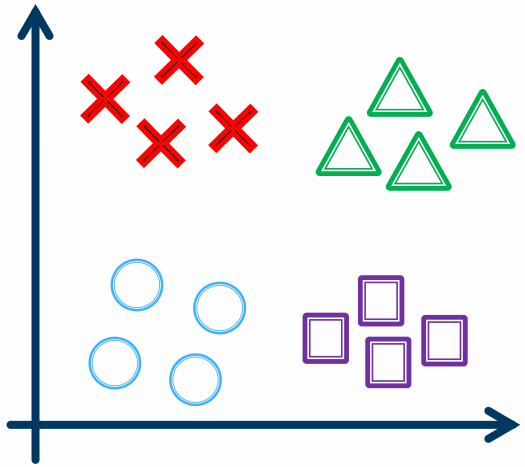
$$(x^T l + 1)^2, \quad (x^T l + 5)^3$$

  - Usually perform worse than Gaussian Kernel
  - It is not used that often

- Esoteric kernels: String kernel, chi-square kernel, histogram intersection kernel, …

Multi-class Classification

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$
Pick class i with largest $(\theta^{(i)})^T x$

Logistic Regression vs. SVMs

n =number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples
If n is large (relative to m):
Use logistic regression, or SVM without a kernel (“linear kernel”)

If n is small, m is intermediate:
Use SVM with Gaussian kernel

If n is small, m is large:
Create/add more features, then use logistic regression or SVM without a kernel

Neural network likely to work well for most of these settings, but may be slower to train.

Strengths and Weaknesses

- SVMs allow for complex decision boundaries, even if the data has only a few features.
- They work well on low-dimensional and high-dimensional data (i.e., few and many features), but don't scale very well with the number of samples.
  - Running an SVM on data with up to 10,000 samples might work well, but working with datasets of size 100,000 or more can become challenging in terms of runtime and memory usage.
- Another downside of SVMs is that they require careful preprocessing of the data and tuning of the parameters.
- It might be worth trying SVMs, particularly if all of your features represent measurements in similar units (e.g., all are pixel intensities) and they are on similar scales. Or, you should normalize the features.
- It can be difficult to understand why a particular prediction was made by an SVM model, and it might be tricky to explain the model to a non-expert.

References

- A. Ng. Machine Learning, Lecture Notes.
- I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, 2016.

UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

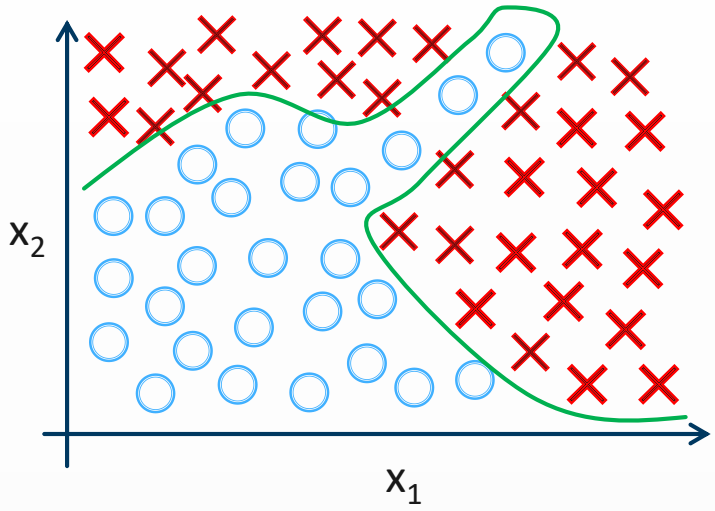
LECTURE 8

ARTIFICIAL NEURAL NETWORKS: REPRESENTATION

Instructor: Asst. Prof. Barış Başpınar

Non-linear Hypotheses

Non-linear Classification



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

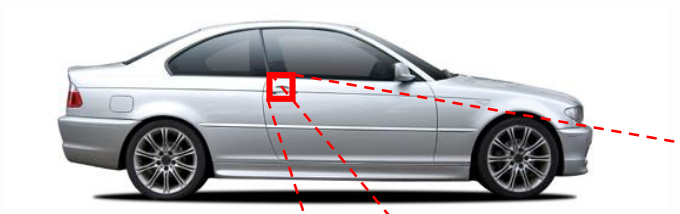
x_1 = size
x_2 = # bedrooms
x_3 = # floors
x_4 = age
$\cdots$
x_{100}

- The number of features will increase to create high-order polynomials
- This impact will be intensified when there are several features (e.g. 100 features in house pricing task -> 170000 features for 3rd degree polynomials)

Non-linear Hypotheses

What is this?

You see this:

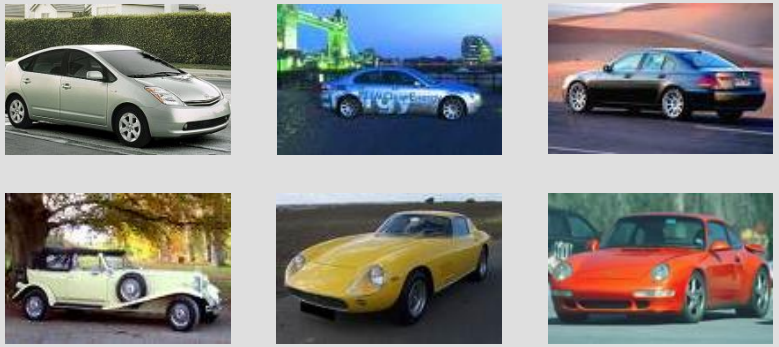


But the camera sees this:

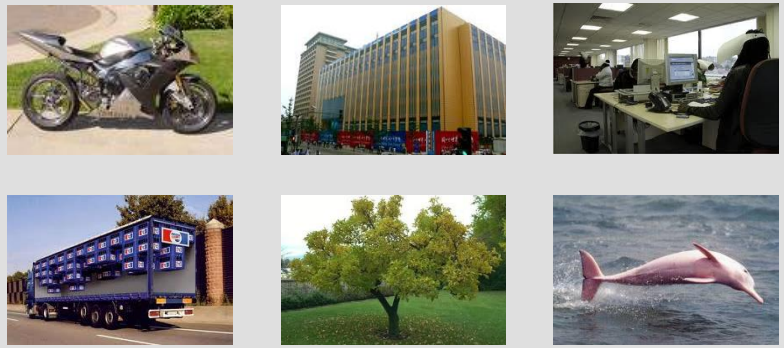
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Non-linear Hypotheses

Computer Vision: Car detection



Cars

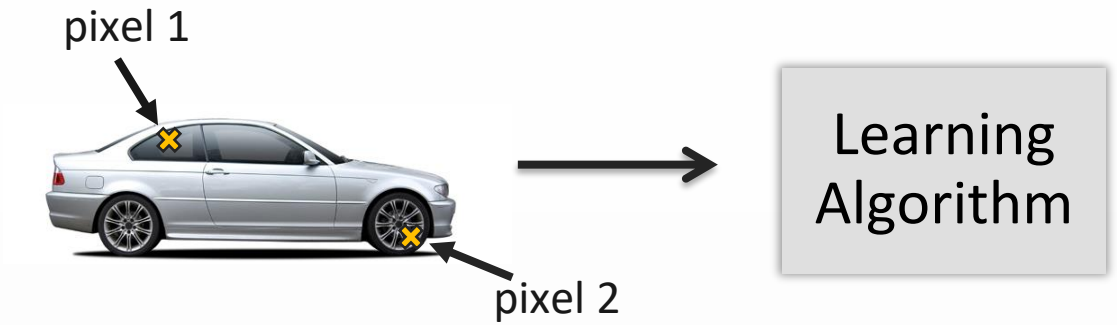


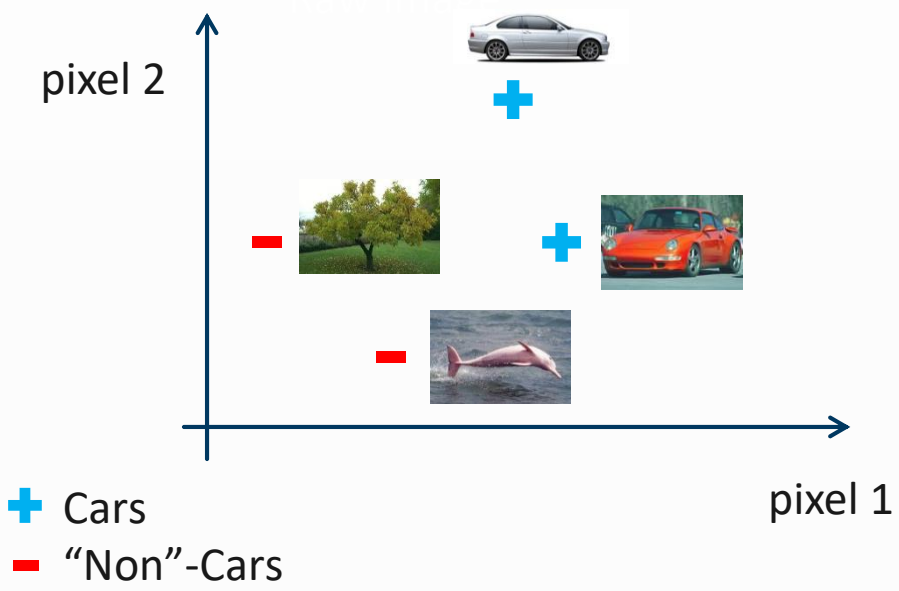
Not a car

Testing:

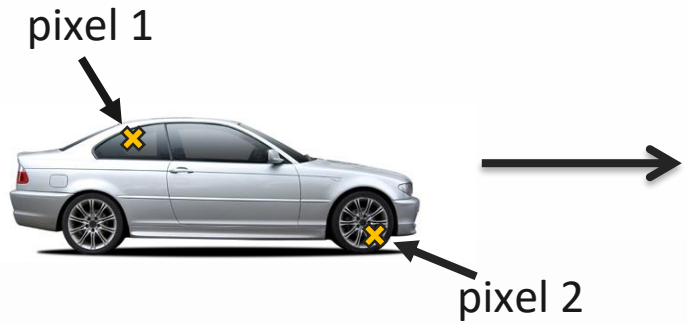
What is this?

Non-linear Hypotheses

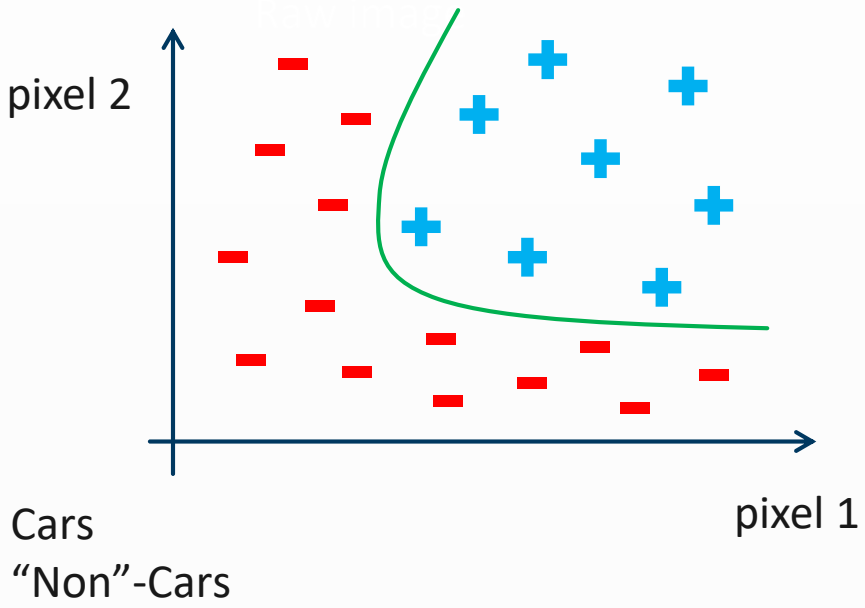




Non-linear Hypotheses



Learning Algorithm



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

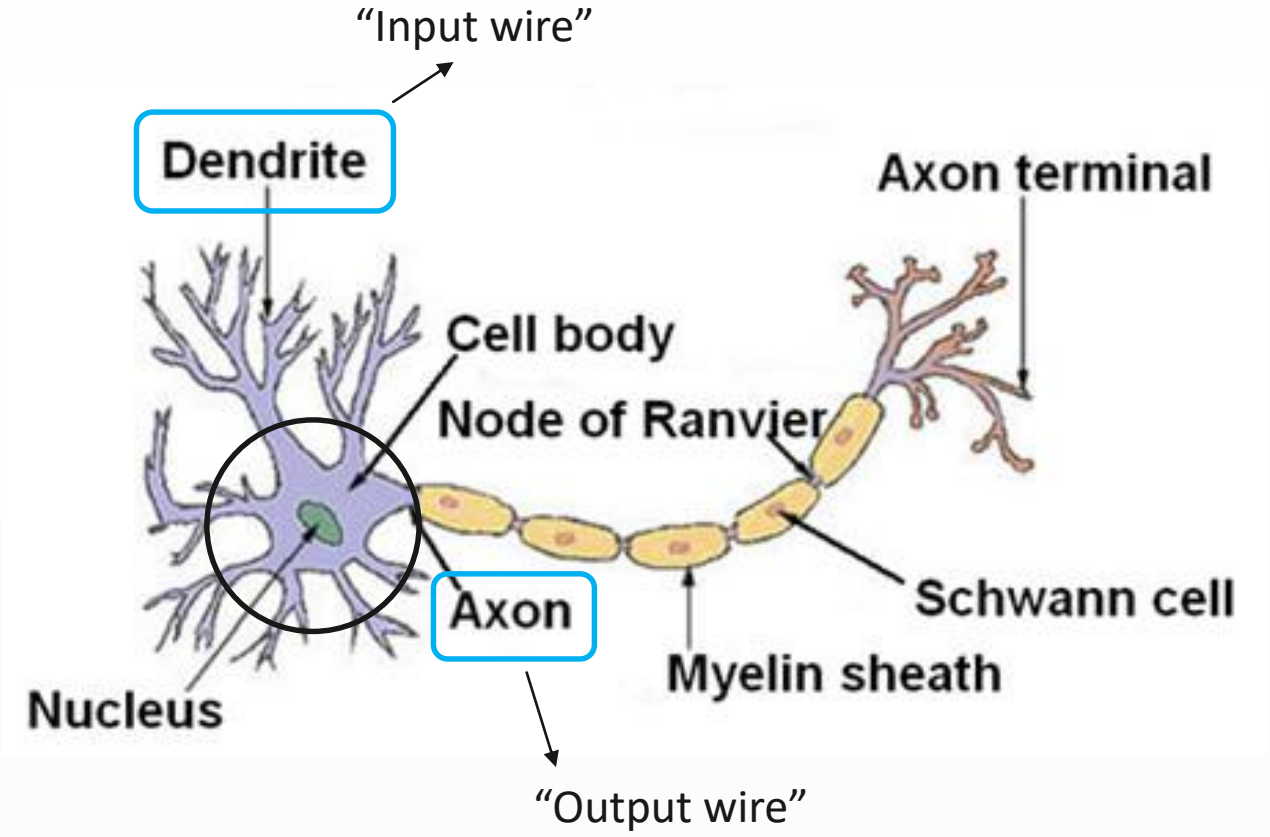
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

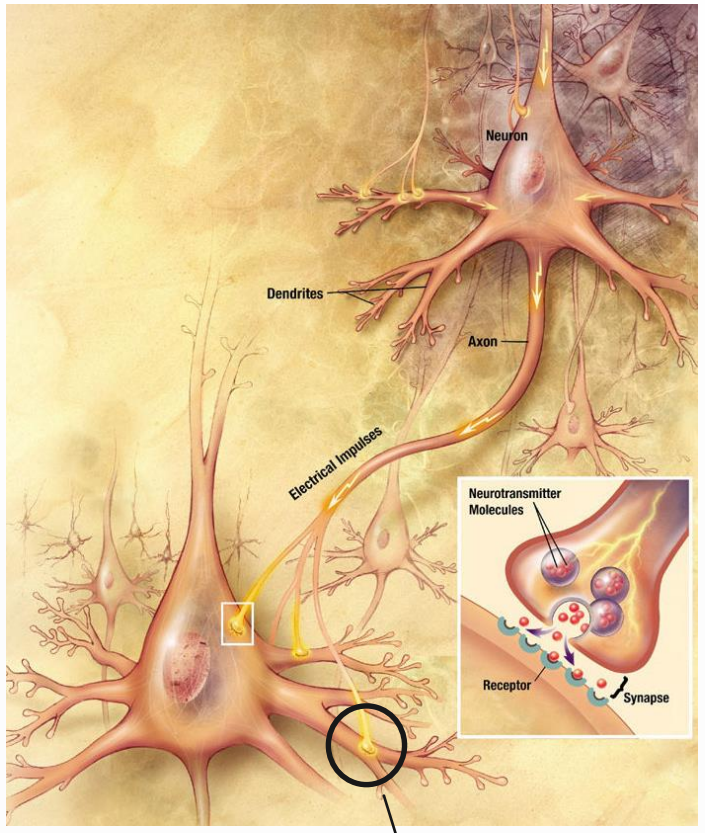
Quadratic features ($x_i \times x_j$): ≈3 million features

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

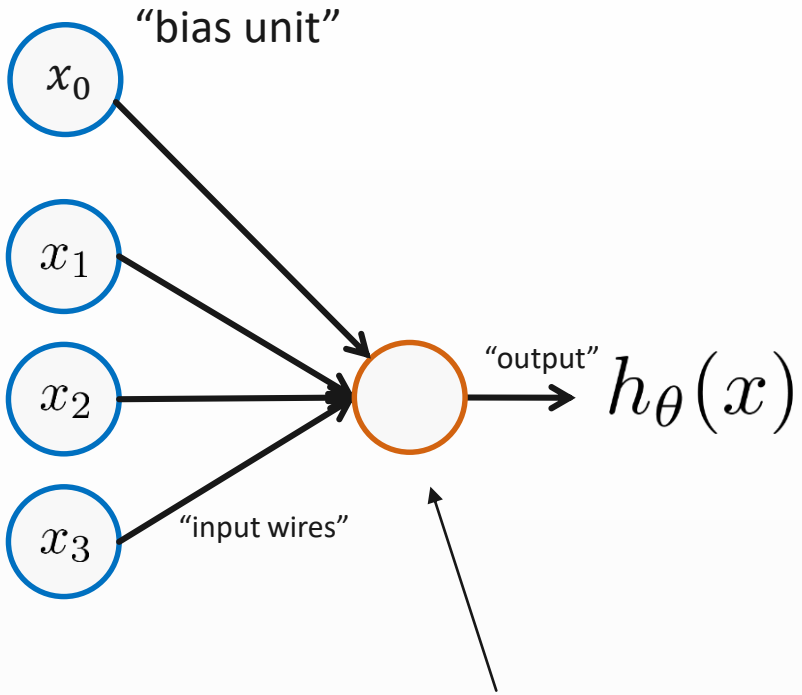
Neuron in the brain





"Output of neuron 1"
"Input of neuron 2"

Artificial Neuron Model: Logistic Unit

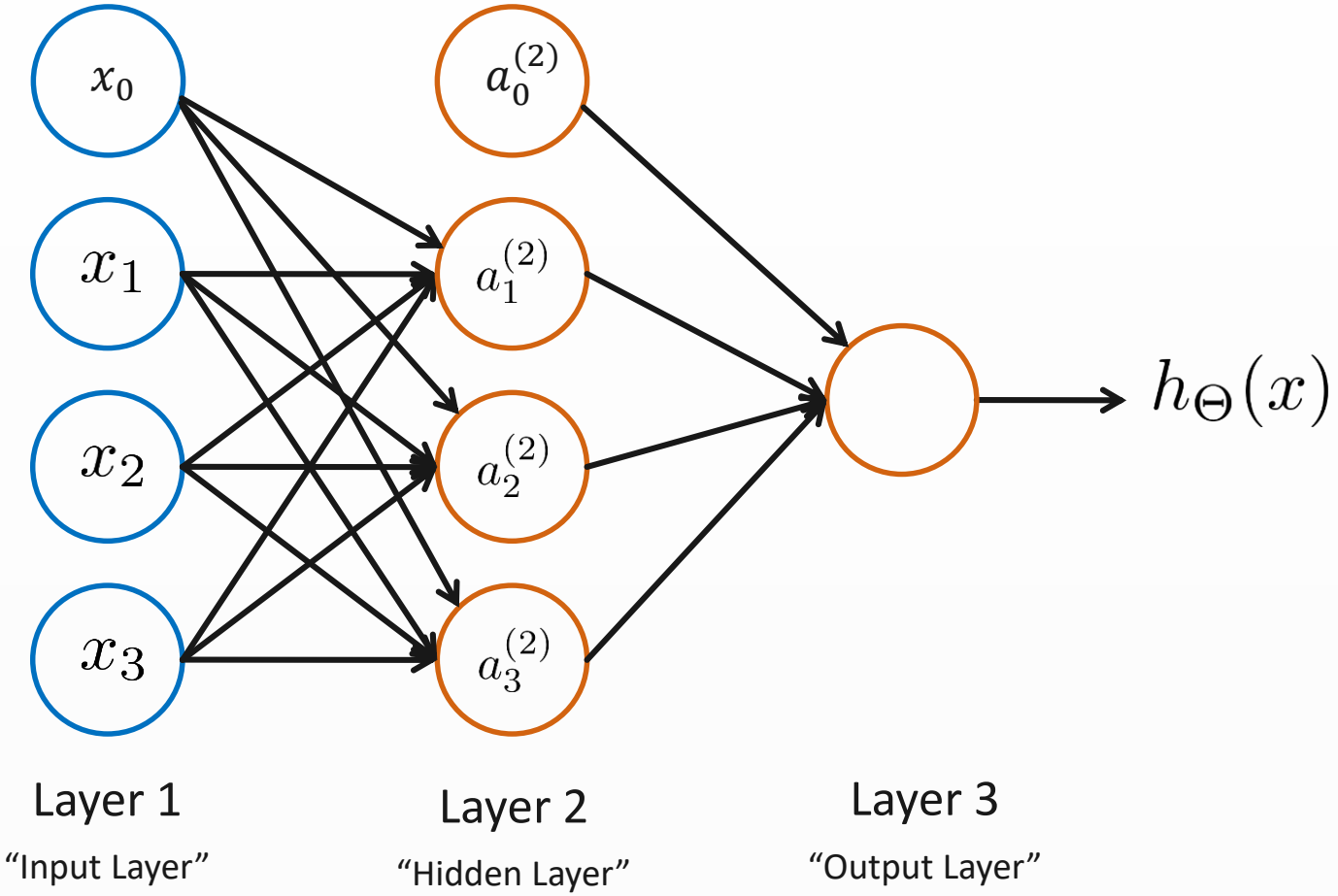


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

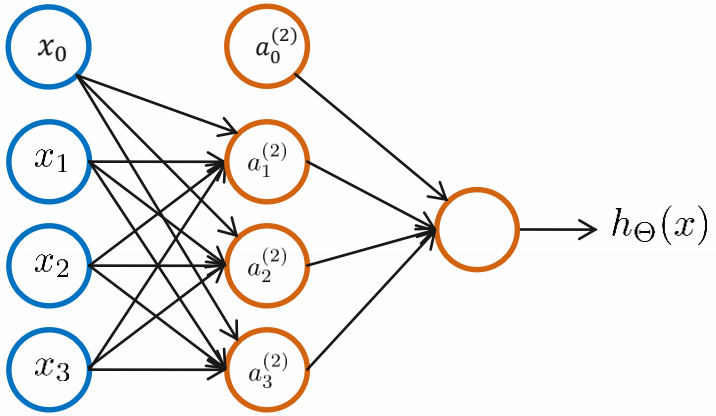
"weights"

An activation function (e.g. Sigmoid (logistic) activation function)

Artificial Neural Network (ANN)



Artificial Neural Network (ANN)



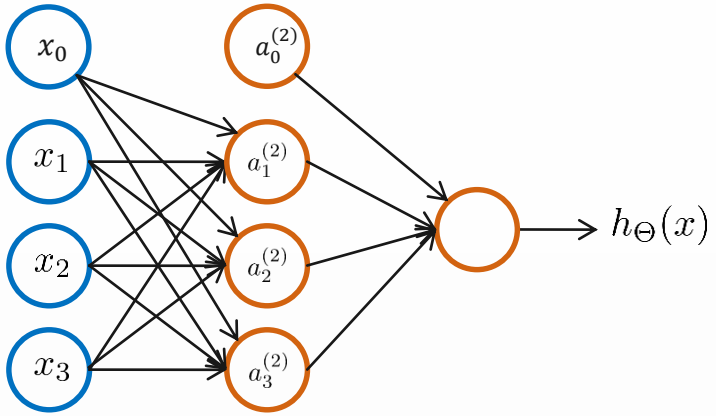
$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad z_1^{(2)}$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad z_2^{(2)}$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \quad z_3^{(2)}$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \quad z^{(3)}$$

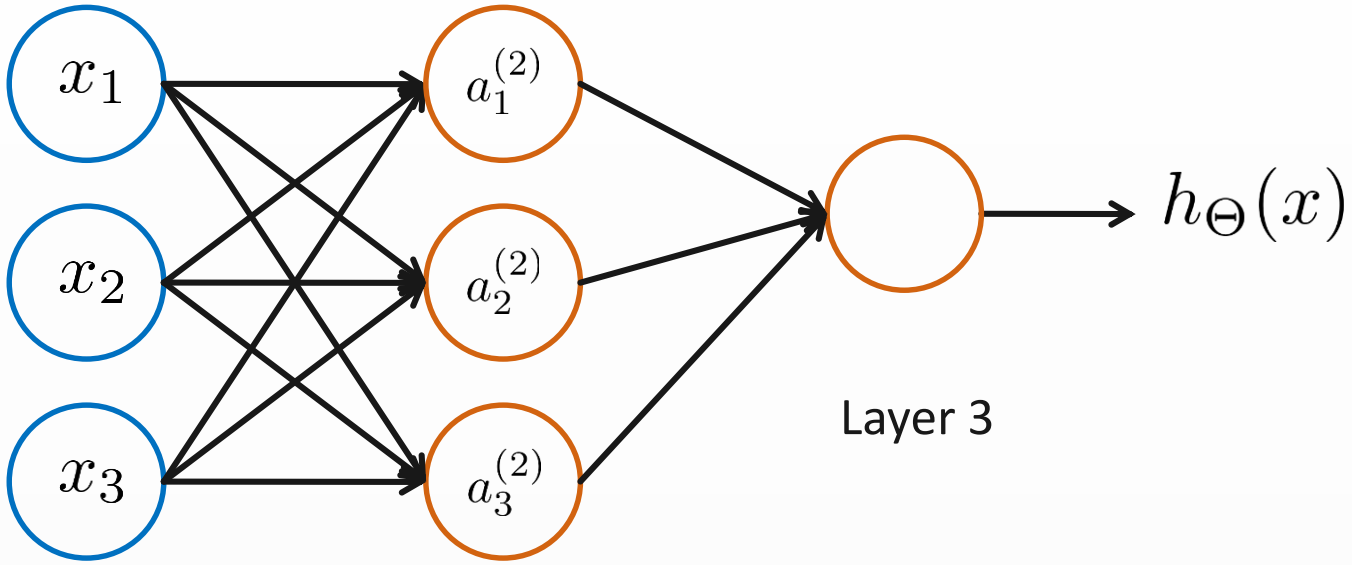
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

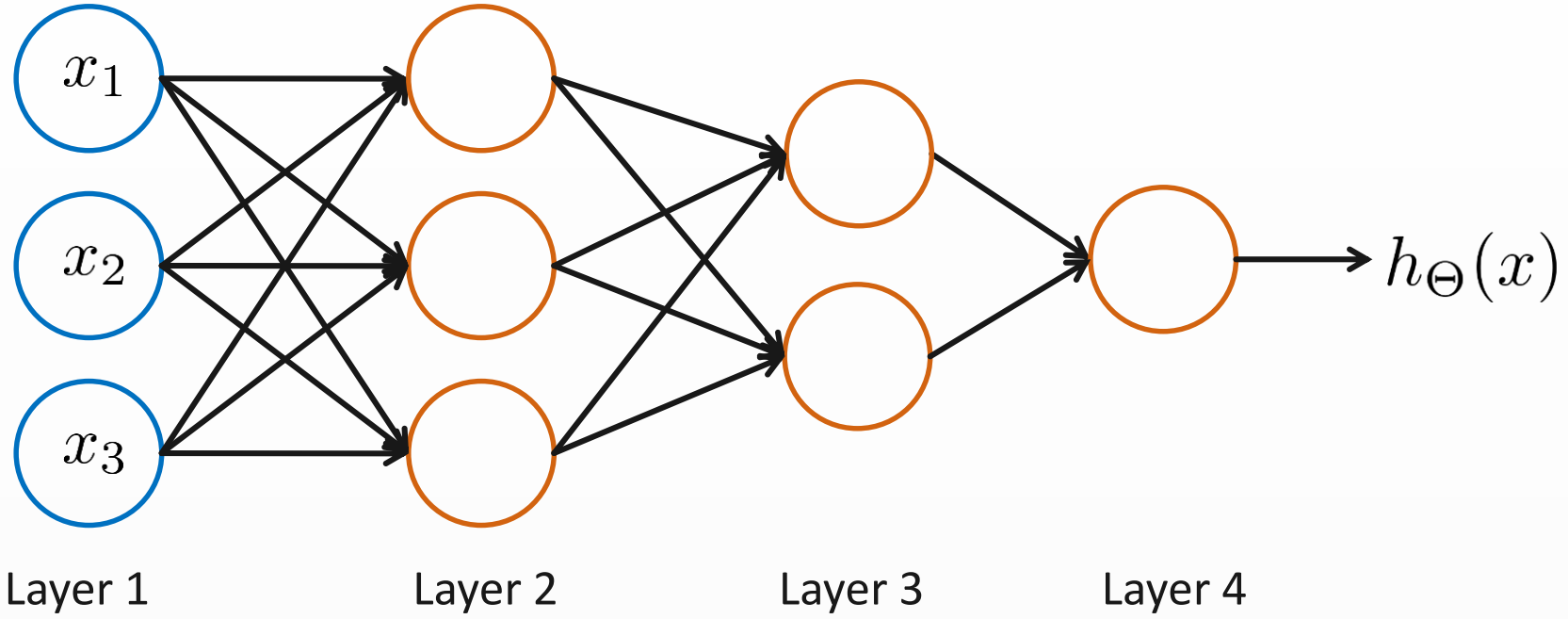
Neural Network learning its own features



$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

- $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ are new features that are derived by ANN

Other network architectures

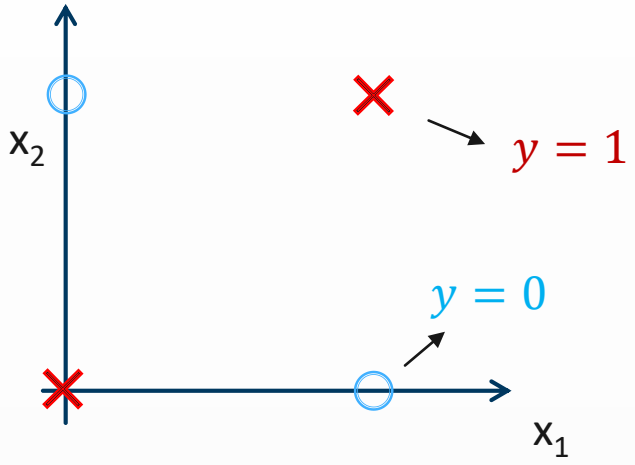


- Different architectures can be used
- You are not limited to single hidden layer
  - There could be several hidden layers with different number of neurons

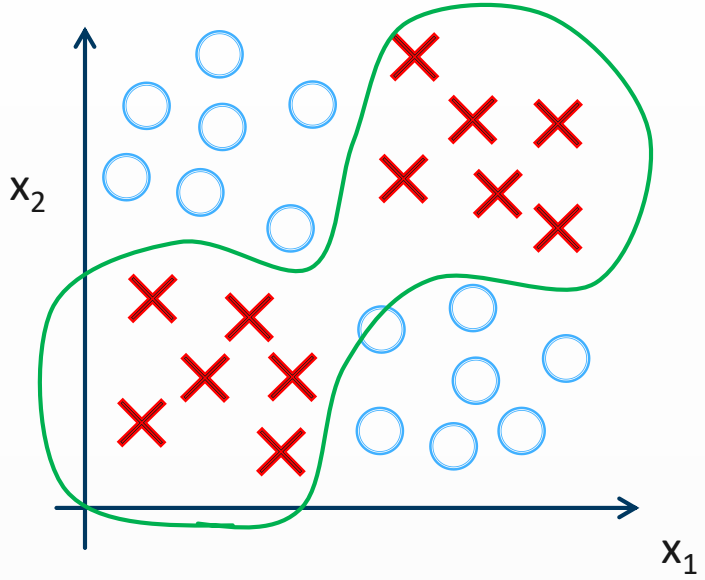
Examples and Intuitions

Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).



$$y = x_1 \text{ XOR } x_2$$
$$x_1 \text{ XNOR } x_2$$
$$\text{NOT } (x_1 \text{ XOR } x_2)$$



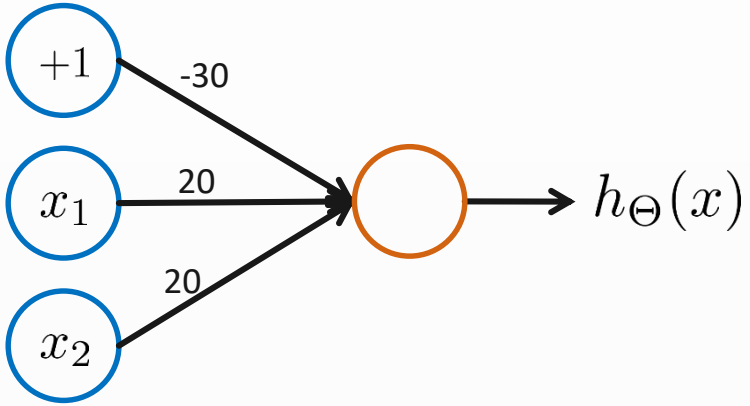
- It is not possible to present this operator using a linear decision boundary

Examples and Intuitions

Simple example: AND

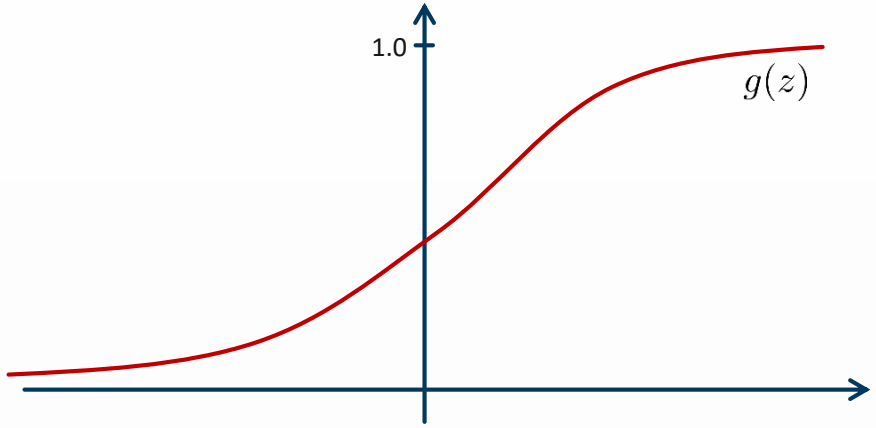
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

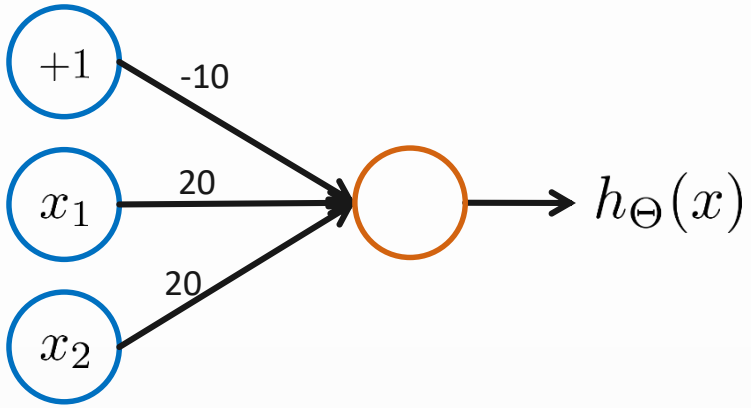
$g(z)$ → sigmoid activation function



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Examples and Intuitions

Example: OR function



x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

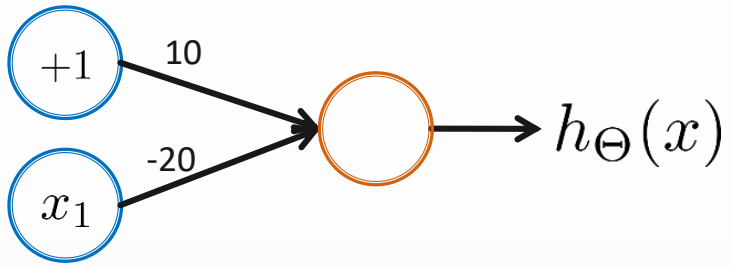
$$h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$$

$g(z) \rightarrow$ sigmoid activation function

Examples and Intuitions

x_1 AND x_2 | x_1 OR x_2

Negation:

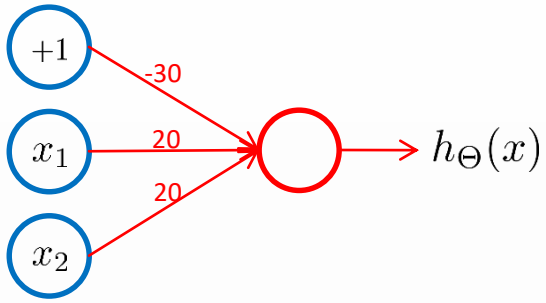


x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

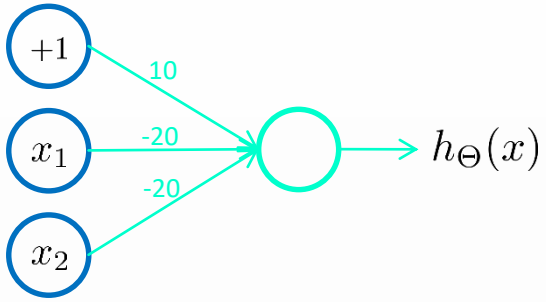
$h_\Theta(x) = g(10 - 20x_1)$ | (NOT x_1) AND (NOT x_2)

Examples and Intuitions

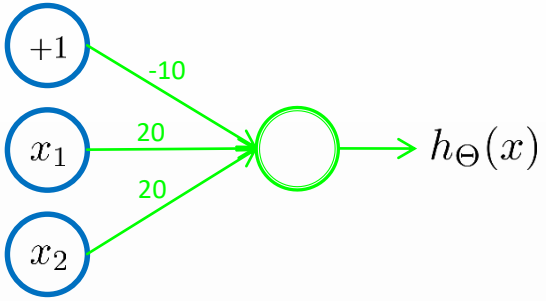
Putting it together: x_1 XNOR x_2



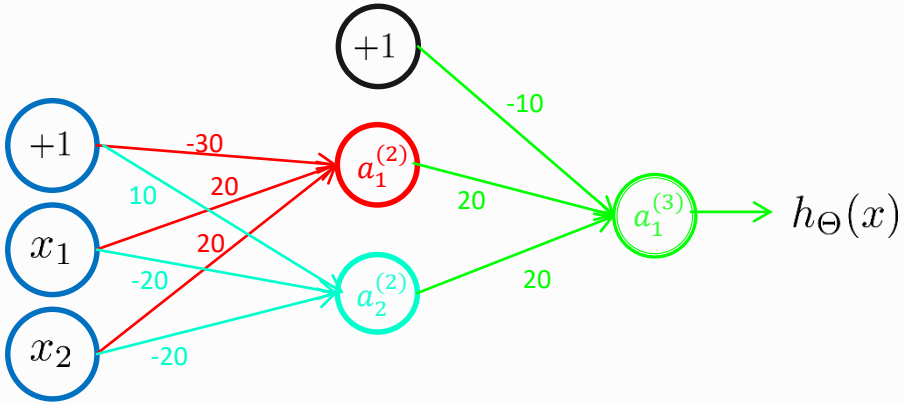
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

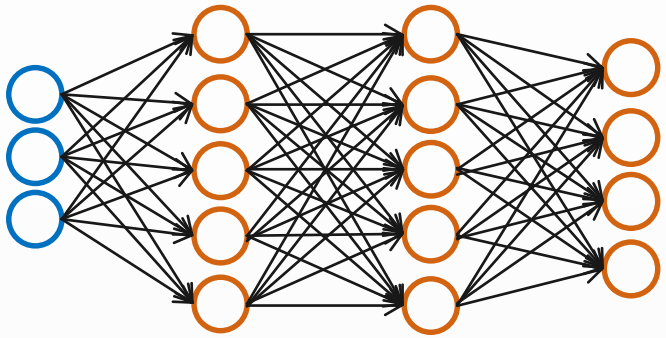
Multiple output units: One-vs-all

Pedestrian

Car

Motorcycle

Truck



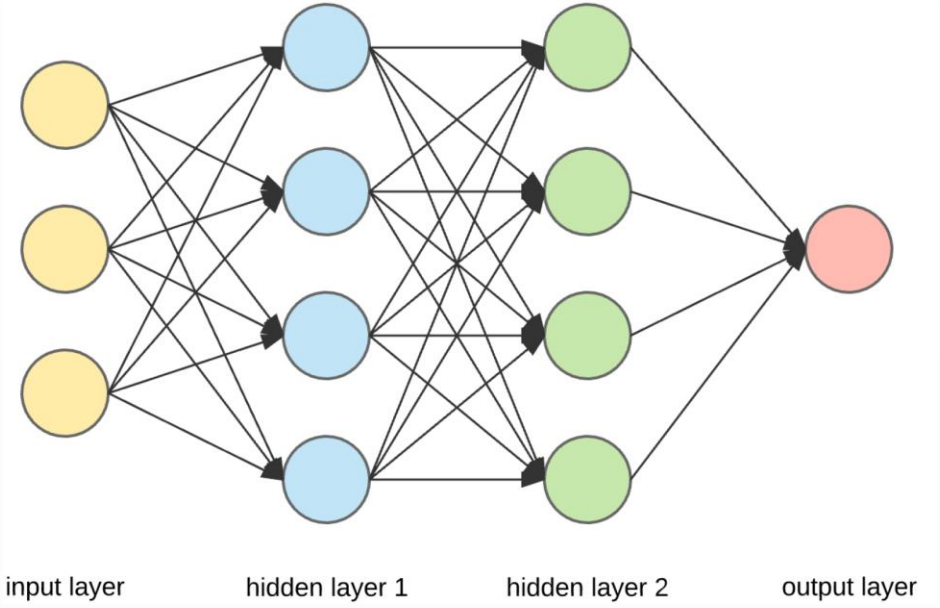
$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

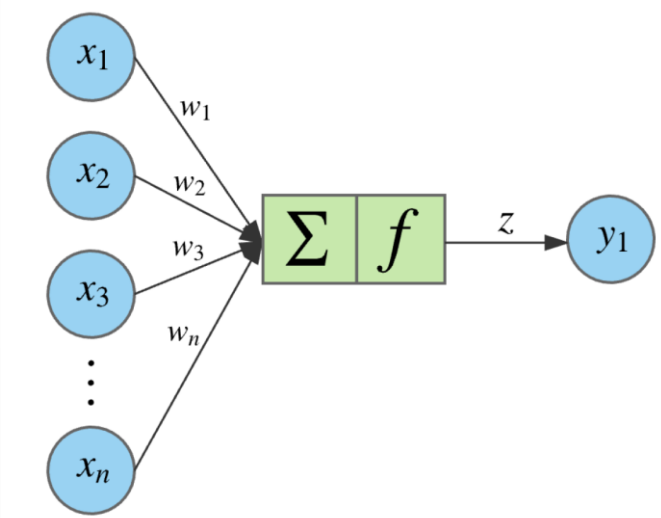
when pedestrian when car when motorcycle

ANN Summary

- An artificial neural network consists of an input and output layers and multiple hidden layers in between



- If we zoom in to one of hidden nodes, we see a linear combination followed by a nonlinear transformation



- Hence the network output is:

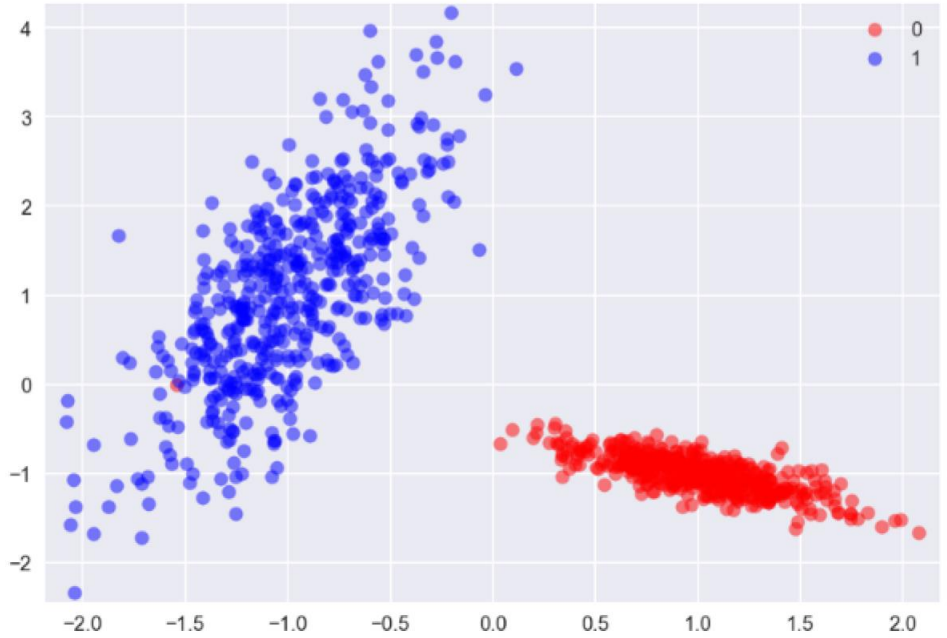
$$y = f(f(f(x.W_1).W_2).W_3)$$

Example: Linearly separable data

Let's implement logistic regression as a special case of ANN.
First, we generate some toy data

```
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                           n_informative=2, random_state=7, n_clusters_per_class=1)
plot_data(X, y)
```

linearly_separable_data_1.py hosted with ❤ by GitHub view raw

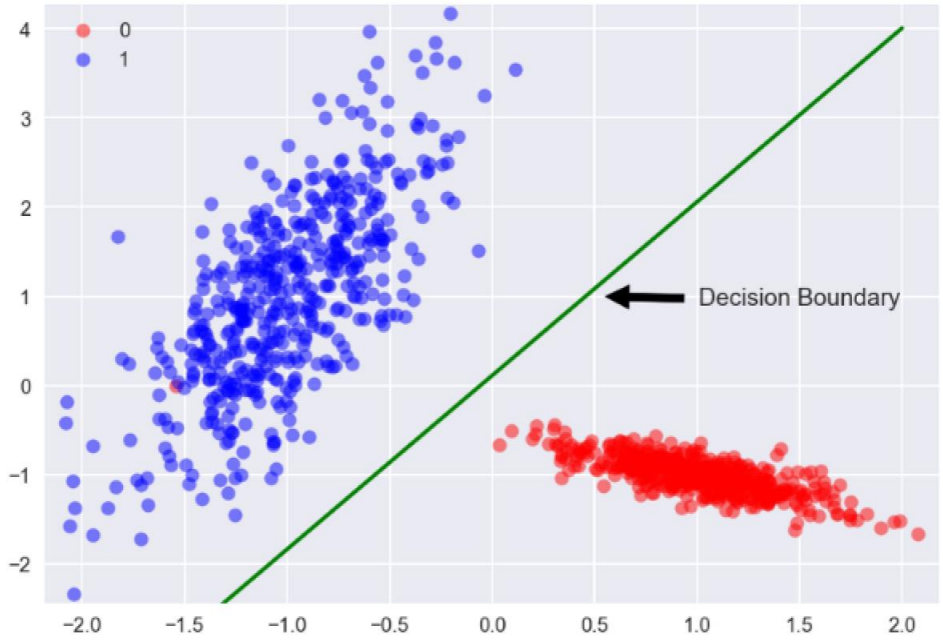


Example: Linearly separable data – Logistic Regression

Now call the logistic regression from scikit-learn

```
1  lr = LogisticRegression()
2  lr.fit(X, y)
```

linearly_separable_data_2.py hosted with ❤ by GitHub view raw



Example: Linearly separable data – Logistic Regression (with ANN)

We will use Keras with TensorFlow backend to solve the same example with ANNs

```
1  model = Sequential()
2  model.add(Dense(units=1, input_shape=(2,), activation='sigmoid'))
3
4  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
5
6  history = model.fit(x=X, y=y, verbose=0, epochs=50)
7  plot_loss_accuracy(history)
```

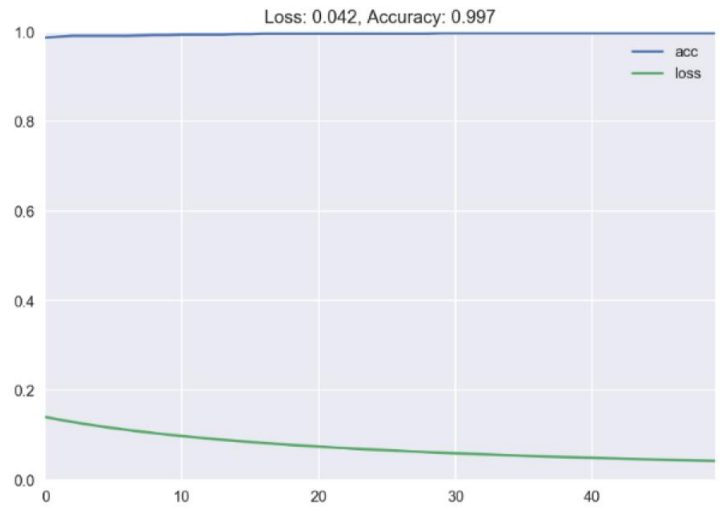
linearly_separable_data_3.py hosted with ❤ by GitHub view raw

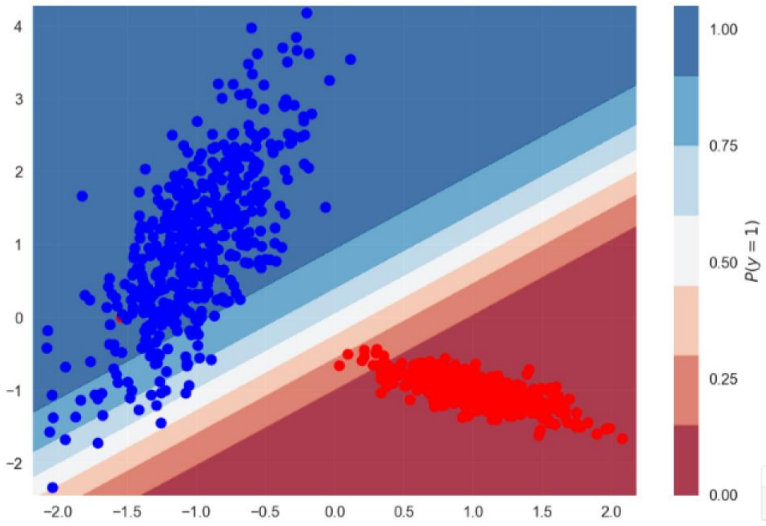
Defining ANN architectures with Keras is very straightforward, as you can see from the code snippet

- Here you can see how to define a simple feedforward hidden layer (referred to as *Dense* in Keras)
- Defining the solver parameters (loss function, optimizer, number of epochs etc) is also very straightforward

Example: Linearly separable data – Logistic Regression (with ANN)

Results look good, since this is a very easy problem:



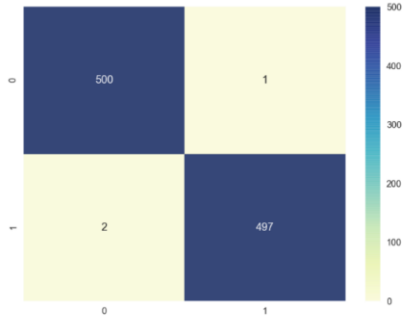



```
plot_confusion_matrix(model, X, y)
```

linearly_separable_data_6.py hosted with ♥ by GitHub view raw

```
plot_decision_boundary(lambda x: model.predict(x), X, y)
```

linearly_separable_data_4.py hosted with ♥ by GitHub view raw

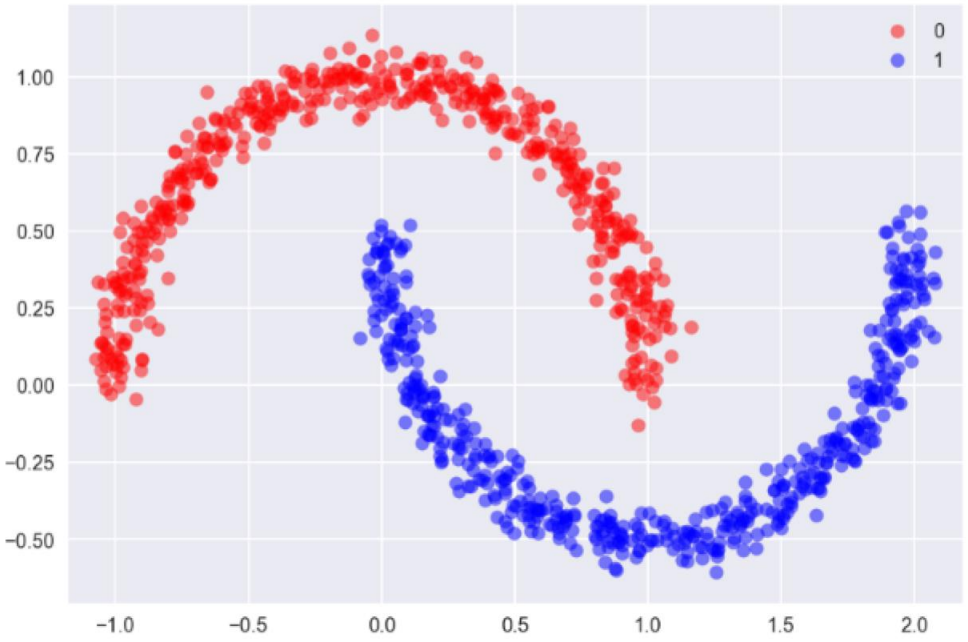


Example: Two Moons Data

So far so good, but what about data that are not linearly separable:

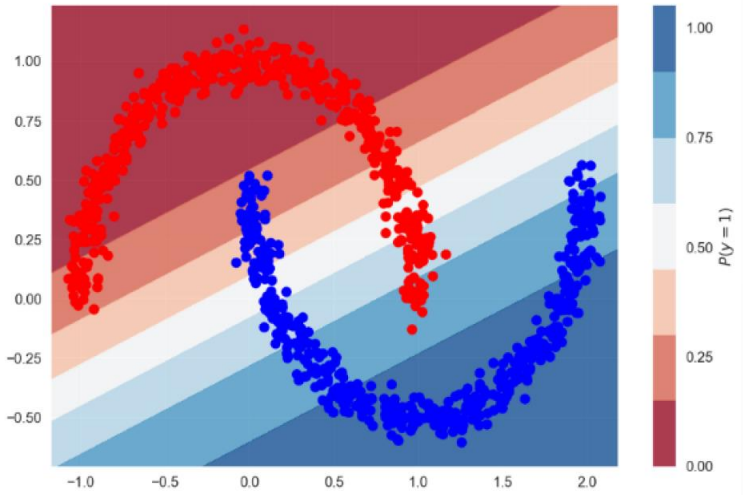
```python
X, y = make_moons(n_samples=1000, noise=0.05, random_state=0)
plot_data(X, y)
```

moons_1.py hosted with ❤ by GitHub view raw

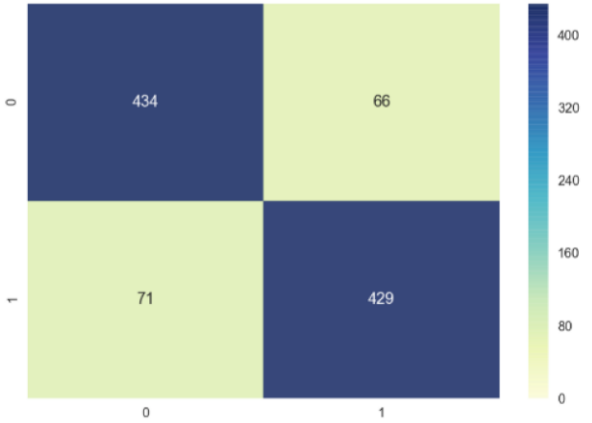


Example: Two Moons Data – Logistic Regression

Results are not that good anymore:



	precision	recall	f1-score	support
0	0.86	0.87	0.86	500
1	0.87	0.86	0.86	500
avg / total	0.86	0.86	0.86	1000

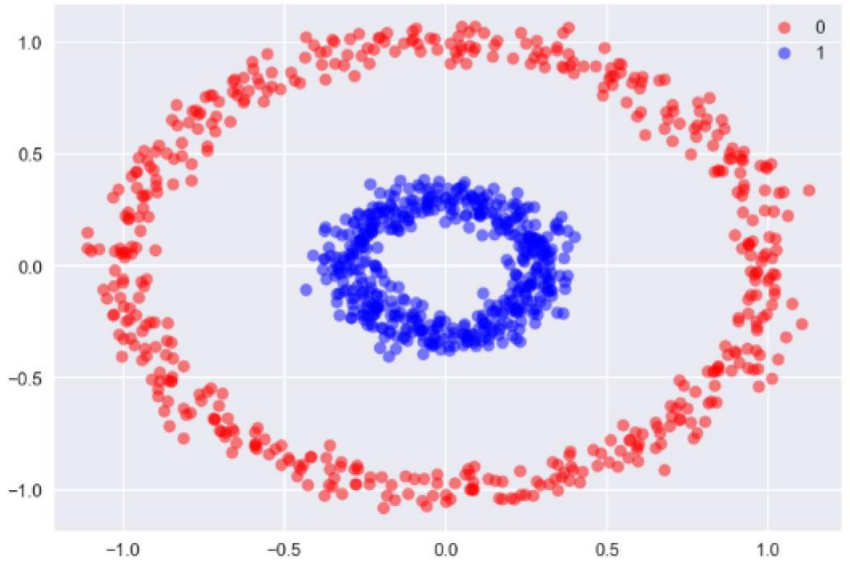


Example: Circles Data

We can also find examples where things go even worse:

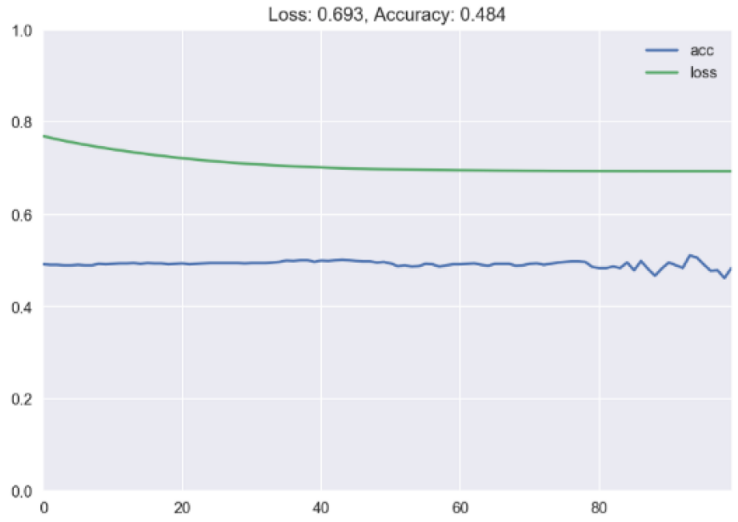
```
1  X, y = make_circles(n_samples=1000, noise=0.05, factor=0.3, random_state=0)
2  plot_data(X, y)
```

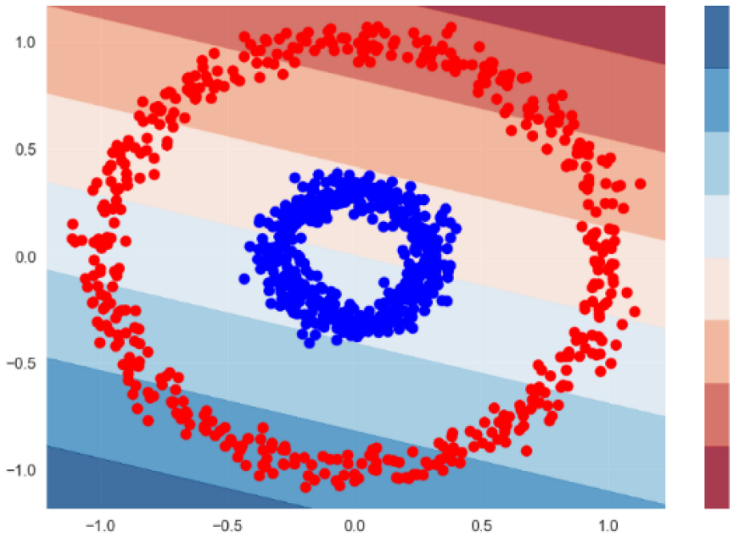
circles_1.py hosted with ❤ by GitHub view raw



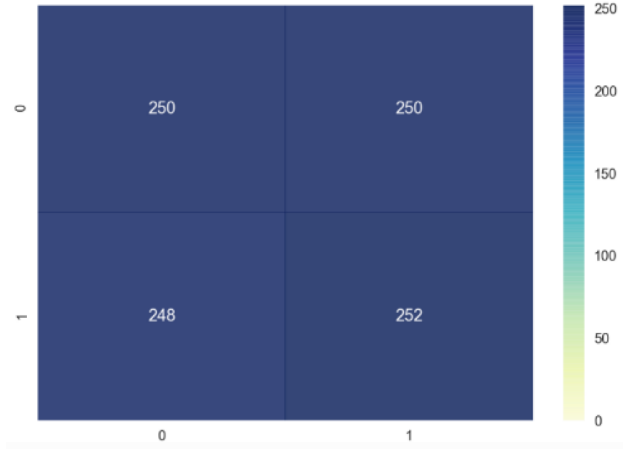
Example: Circles Data – Logistic Regression

We can also find examples where things go even worse:





	precision	recall	f1-score	support
0	0.50	0.50	0.50	500
1	0.50	0.50	0.50	500
avg / total	0.50	0.50	0.50	1000



Example: Two Moons Data – Deep ANN

Now let's redo this problematic examples using multiple layer ANNs:

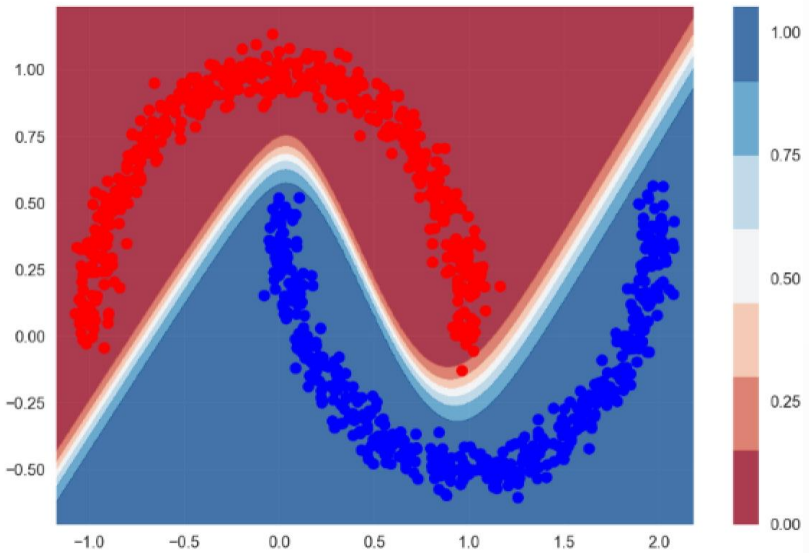
```python
X, y = make_moons(n_samples=1000, noise=0.05, random_state=0)

model = Sequential()
model.add(Dense(4, input_shape=(2,), activation='tanh'))
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))

model.compile(Adam(lr=0.01), 'binary_crossentropy', metrics=['accuracy'])

history = model.fit(X, y, verbose=0, epochs=100)

plot_loss_accuracy(history)
```

moons_5.py hosted with ❤ by GitHub view raw

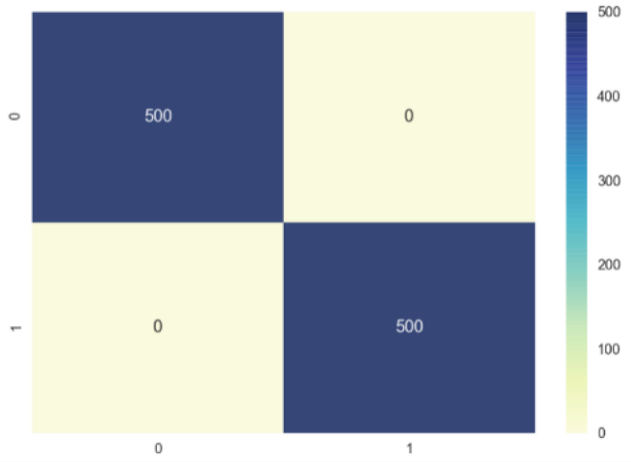
- Now we have 2 hidden layers with $\tanh$ activation function
- For the output layer we still use sigmoid since we are working on binary classification problems.

Example: Two Moons Data – Deep ANN

Although the model is not very deep, it still achieves full accuracy:

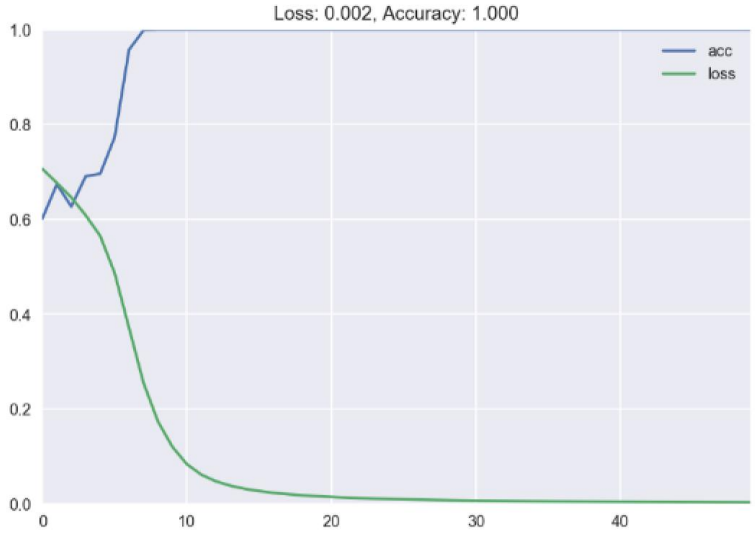


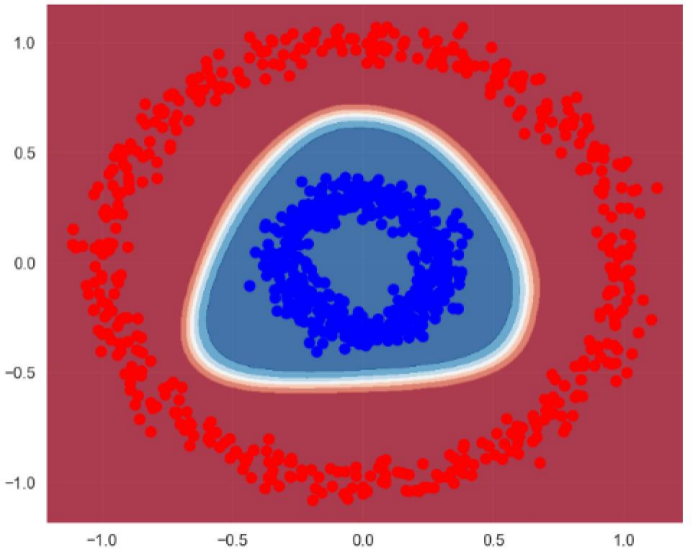
	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
avg / total	1.00	1.00	1.00	1000



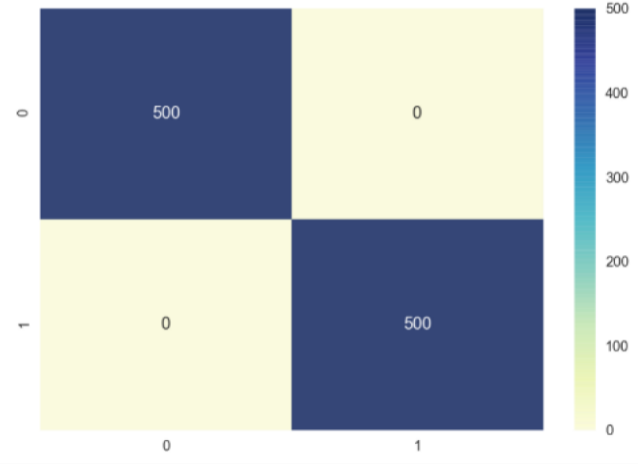
Example: Circles Data – Deep ANN

The same network also solves the circle problem:





	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
avg / total	1.00	1.00	1.00	1000

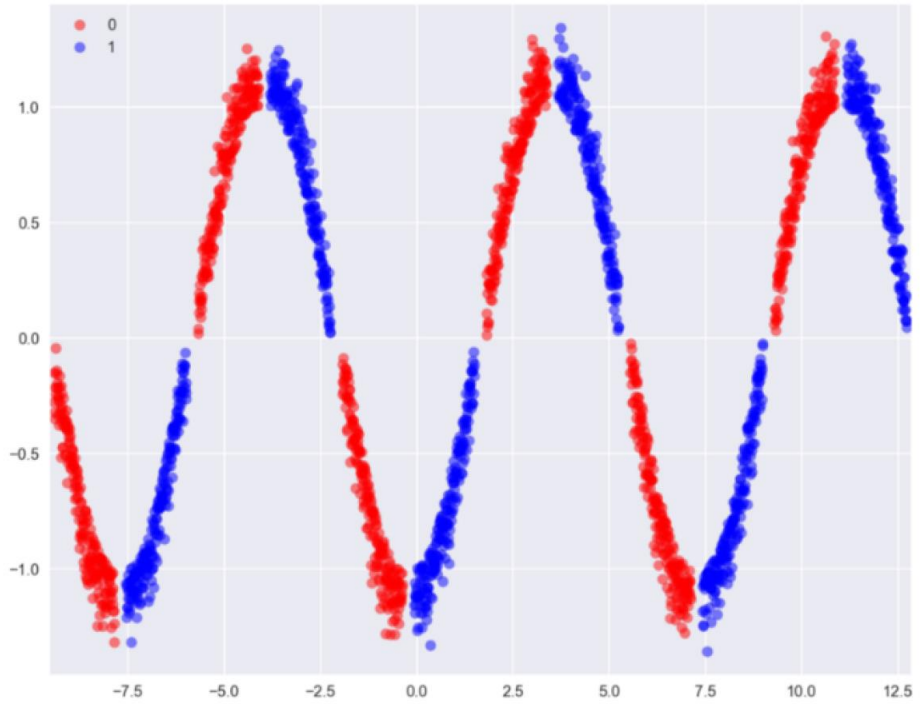


Example: Sin Wave Data – Deep ANN

Let's do something more interesting, a problem where a single connected decision boundary is not possible:

```
1  X, y = make_sine_wave()
2
3  plot_data(X, y, figsize=(10, 8))
```

sine_1.py hosted with ❤ by GitHub view raw



Example: Sin Wave Data – Deep ANN

We will need a deeper model for this problem

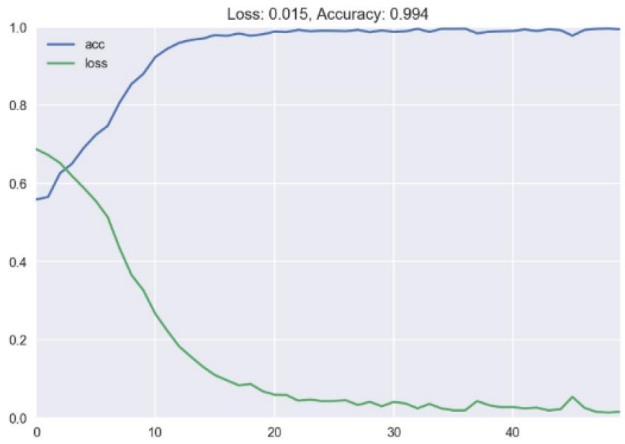
```
1  model = Sequential()
2  model.add(Dense(64, input_shape=(2,), activation='tanh'))
3  model.add(Dense(64, activation='tanh'))
4  model.add(Dense(64, activation='tanh'))
5  model.add(Dense(1, activation='sigmoid'))
6
7  model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
8
9  history = model.fit(X, y, verbose=0, epochs=50)
10
11 plot_loss_accuracy(history)
```

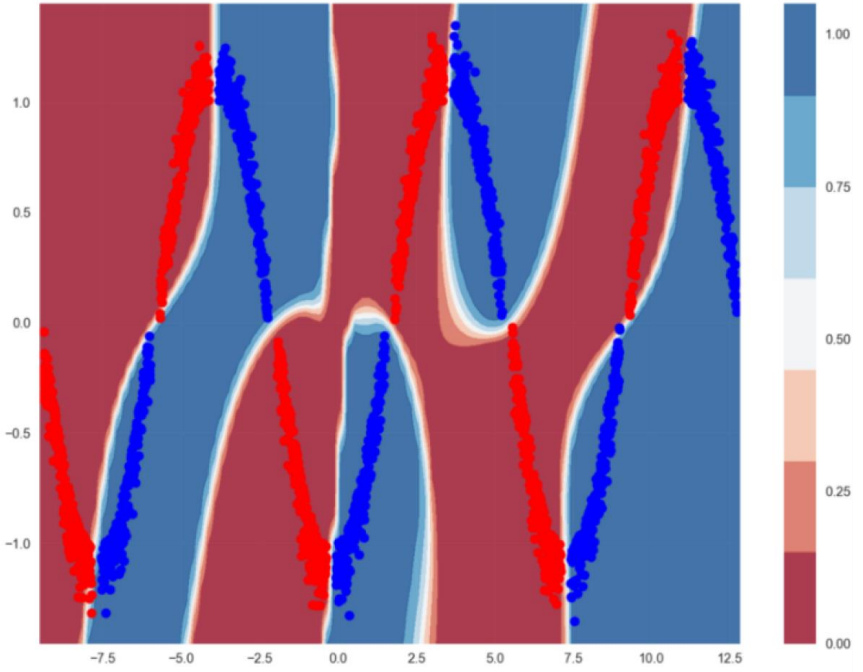
sine_2.py hosted with ❤ by GitHub view raw

- Setting number of layers and units for a problem is more of an art than technology, it is still an active area of research

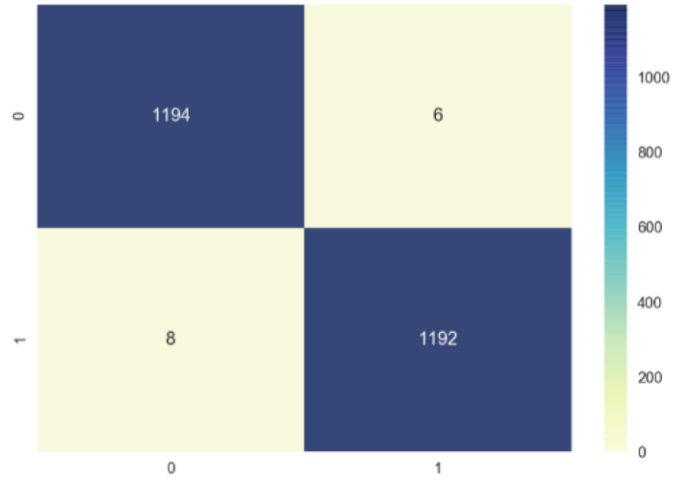
Example: Sin Wave Data – Deep ANN

Results for sine wave classification





	precision	recall	f1-score	support
0	0.99	0.99	0.99	1200
1	0.99	0.99	0.99	1200
avg / total	0.99	0.99	0.99	2400

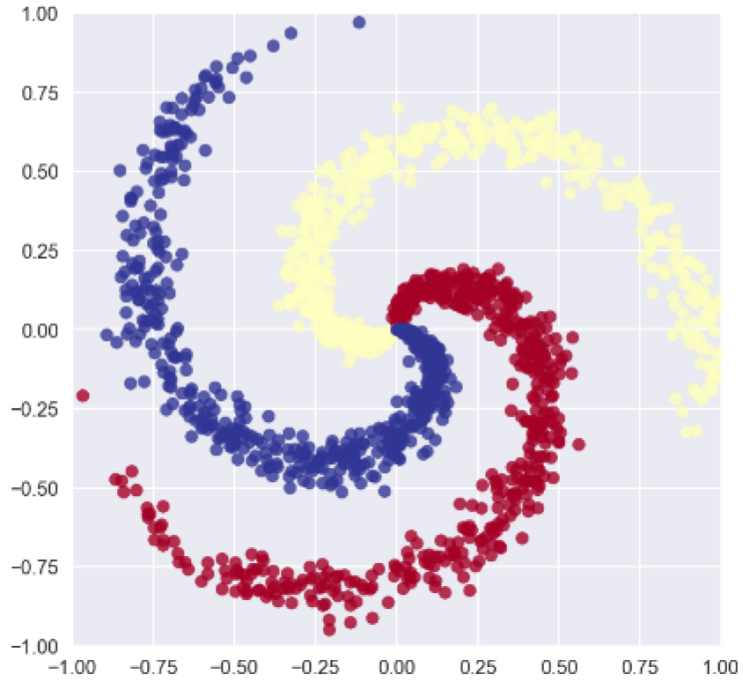


Example: Multi-class Classification

As a final toy example, let's attempt multi label classification

```
1  X, y = make_multiclass(K=3)
```

multiclass_1.py hosted with ❤ by GitHub view raw



Example: Multi-class Classification – Logistic Regression (Softmax)

First we generalize our linear classifier, the logistic regression algorithm, to handle multiple classes, this is known as the softmax classifier

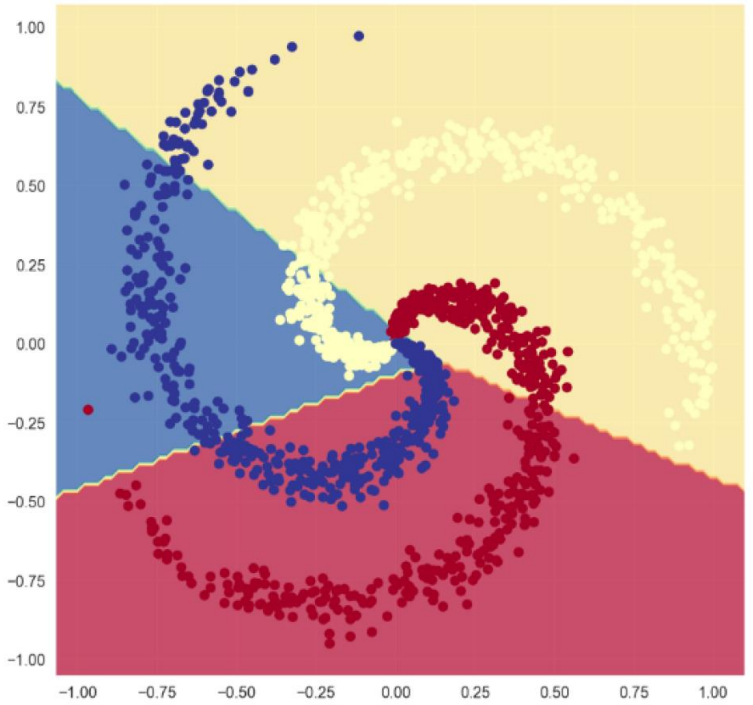
```
1  model = Sequential()
2  model.add(Dense(3, input_shape=(2,), activation='softmax'))
3
4  model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])
5
6  y_cat = to_categorical(y)
7  history = model.fit(X, y_cat, verbose=0, epochs=20)
8
9  plot_loss_accuracy(history)
```

multiclass_3.py hosted with ❤ by GitHub view raw

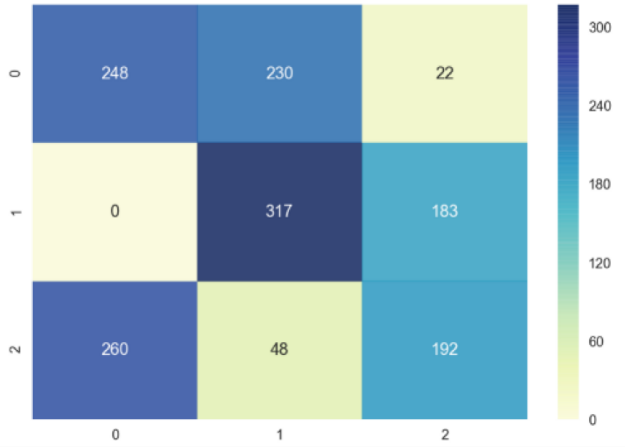
- Note that we need to convert our labels to categorical for this to work
- Output is a 3–vector, which can be interpreted as the probabilities of belonging to each class

Example: Multi-class Classification – Logistic Regression (Softmax)

Results for the linear classifier:



	precision	recall	f1-score	support
0.0	0.49	0.50	0.49	500
1.0	0.53	0.63	0.58	500
2.0	0.48	0.38	0.43	500
avg / total	0.50	0.50	0.50	1500



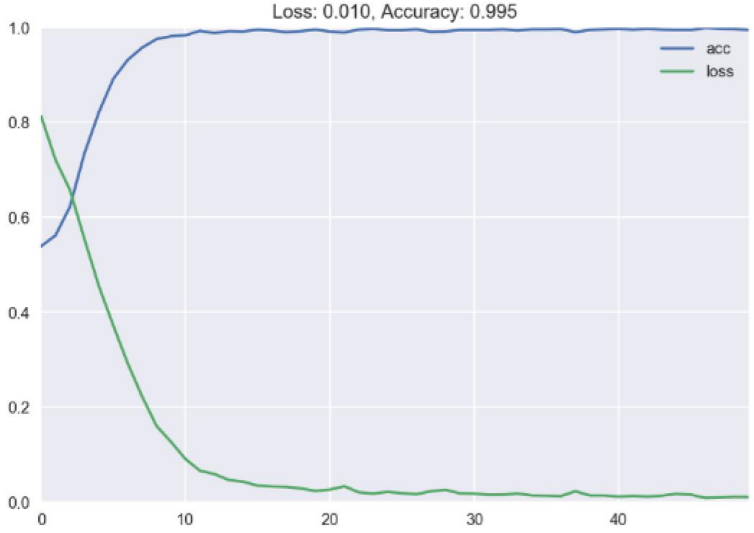
Example: Multi-class Classification – Deep ANN

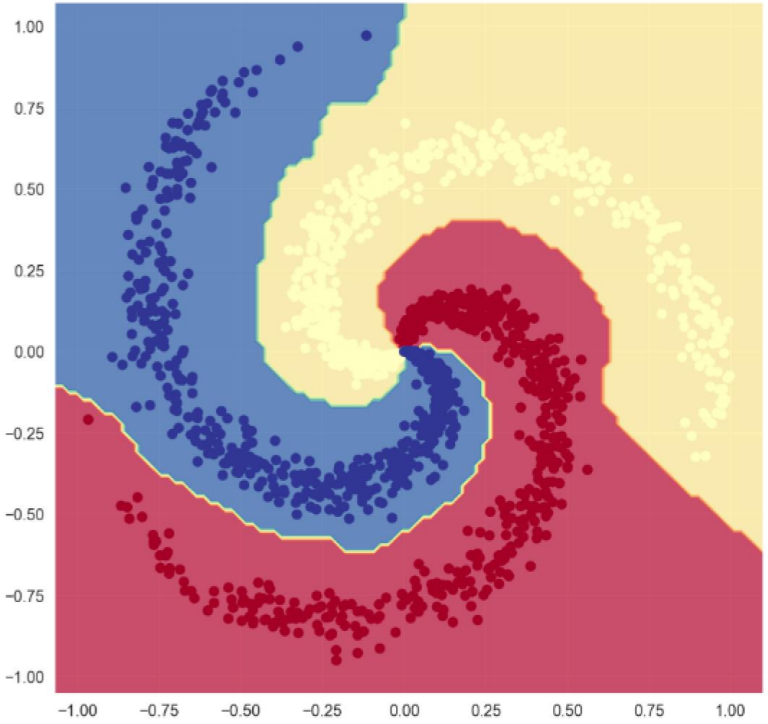
Now let's repeat the solution with a deep ANN:

```
model = Sequential()
model.add(Dense(64, input_shape=(2,), activation='tanh'))
model.add(Dense(32, activation='tanh'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(3, activation='softmax'))

model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])

y_cat = to_categorical(y)
history = model.fit(X, y_cat, verbose=0, epochs=50)

plot_loss_accuracy(history)
```

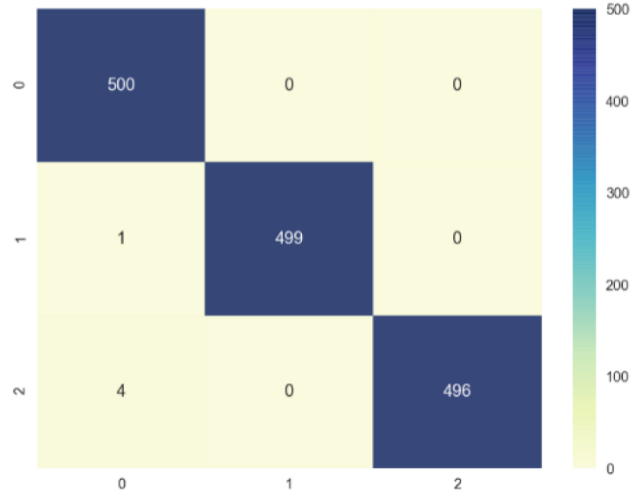
multiclass_4.py hosted with ❤ by GitHub view raw

Example: Multi-class Classification – Deep ANN





	precision	recall	f1-score	support
0.0	0.99	1.00	1.00	500
1.0	1.00	1.00	1.00	500
2.0	1.00	0.99	1.00	500
avg / total	1.00	1.00	1.00	1500



Example - Human Resources problem

Enough with toy datasets! Let's try attacking a real world problem. We will investigate the Human Resources problem (dataset taken form Kaggle).

- The objective is predicting if a given employee will leave the work based on his/her salary, number of years spent at the company etc.

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	promotion_last_5years	sales	salary	left
0	0.38	0.53	2	157	3	0	0	sales	low	1
1	0.80	0.86	5	262	6	0	0	sales	medium	1
2	0.11	0.88	7	272	4	0	0	sales	medium	1
3	0.72	0.87	5	223	5	0	0	sales	low	1
4	0.37	0.52	2	159	3	0	0	sales	low	1

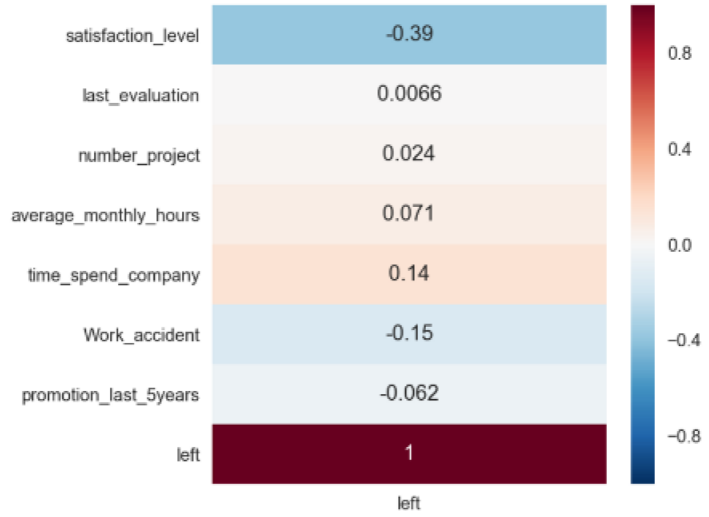
- This is a binary classification problem, since we only have two labels (left or stay)

Example - Human Resources problem

Let's do some data processing/cleaning. Here are the correlations between the label and other features

```
1  seaborn.heatmap(rawdf.corr()[['left']], annot=True, vmin=-1, vmax=1)
```

case_classification_1.py hosted with ❤ by GitHub view raw



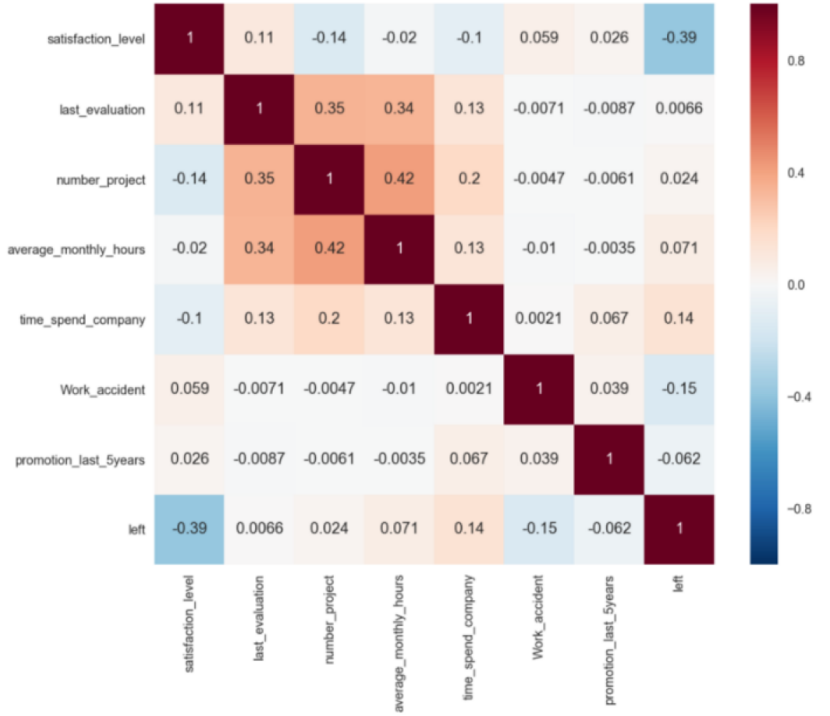
- As expected, there is a strong negative correlation between satisfaction level and leaving the company.

Example - Human Resources problem

What about all pairwise correlations?

```
1  seaborn.heatmap(rawdf.corr(), annot=True, square=True, vmin=-1, vmax=1)
```

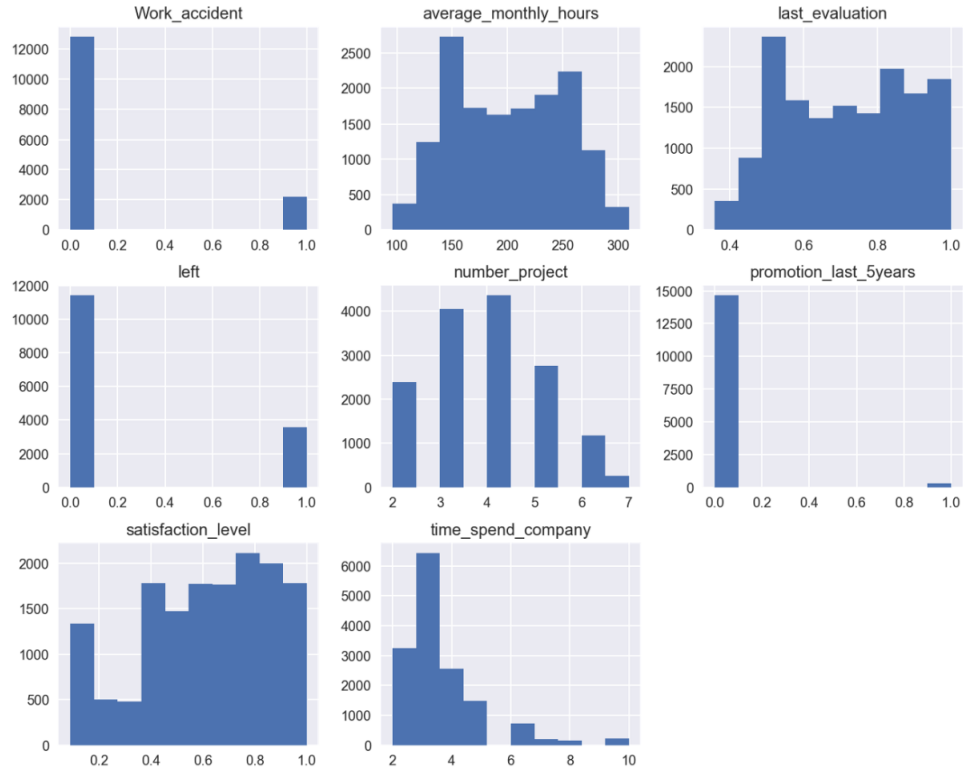
case_classification_2.py hosted with ♥ by GitHub view rav



- As expected, the monthly hours spent is positively correlated with number of projects

Example - Human Resources problem

Now lets look at the histograms to see which features might need normalization/scaling



- It seems like we need to normalize monthly hours, number of projects and time spend in company

Example - Human Resources problem

We can use standard scaler for normalization.

- We also need to convert our labels to categorical variables
- Then we separate our data as 30% for testing and 70% for training

```
ss = StandardScaler()
scale_features = ['average_monthly_hours', 'number_project', 'time_spend_company']
df[scale_features] = ss.fit_transform(df[scale_features])
```

case_classification_3.py hosted with ❤ by GitHub view raw

```
categorical_features = ['sales', 'salary']
df_cat = pd.get_dummies(df[categorical_features])
df = df.drop(categorical_features, axis=1)
df = pd.concat([df, df_cat], axis=1)
```

case_classification_4.py hosted with ❤ by GitHub view raw

```
X = df.drop('left', axis=1).values
y = df['left'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

case_classification_5.py hosted with ❤ by GitHub view raw

Example - Human Resources problem

Now we are ready for training! Let's compare the classical logistic regression with a deep ANN:

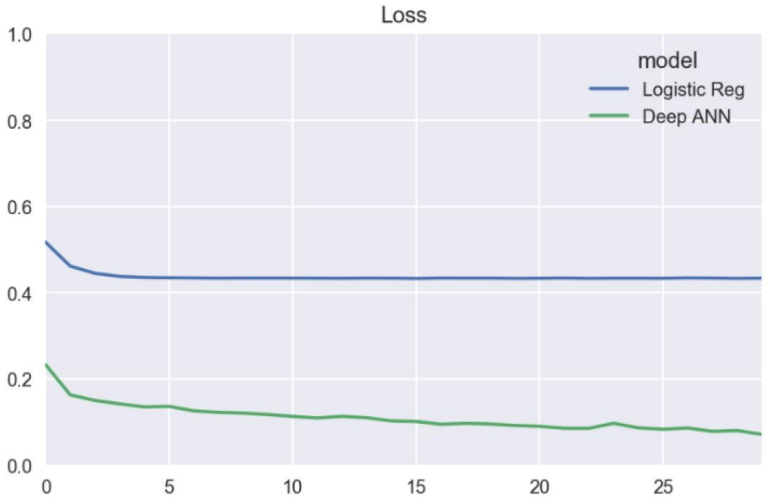
```python
deep_model = Sequential()
deep_model.add(Dense(64, input_shape=(X_train.shape[1],), activation='tanh'))
deep_model.add(Dense(16, activation='tanh'))
deep_model.add(Dense(1, activation='sigmoid'))

deep_model.compile(Adam(lr=0.01), 'binary_crossentropy', metrics=['accuracy'])

deep_history = deep_model.fit(X_train, y_train, verbose=0, epochs=30)
plot_loss_accuracy(deep_history)
```

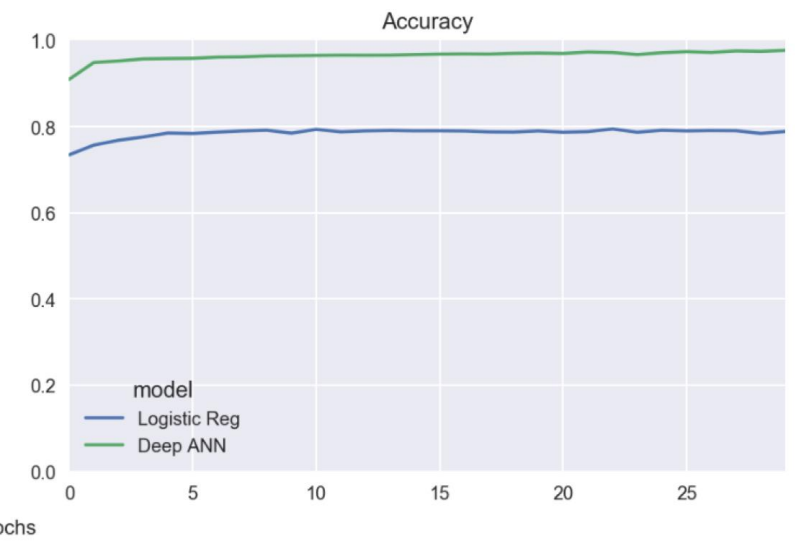
case_classification_7.py hosted with ❤ by GitHub view raw

- This model is not that deep, but technically any model with more than 1 hidden layer can be considered as *deep*

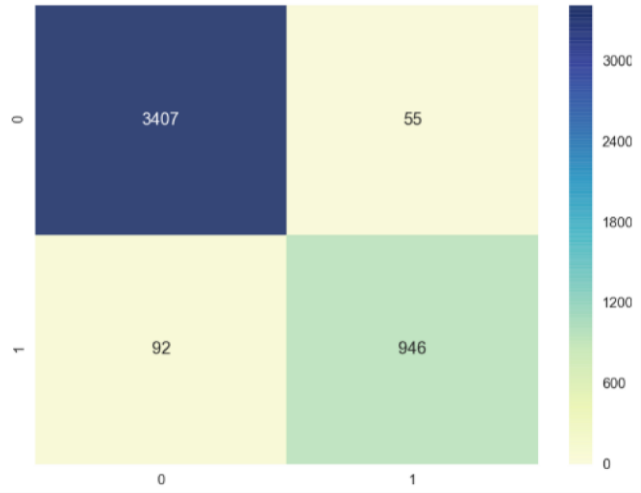
Example - Human Resources problem

Training results look good! ANN smashes logistic regression





	precision	recall	f1-score	support
0	0.97	0.98	0.98	3462
1	0.95	0.91	0.93	1038
avg / total	0.97	0.97	0.97	4500



Confusion matrix computed with **test data**

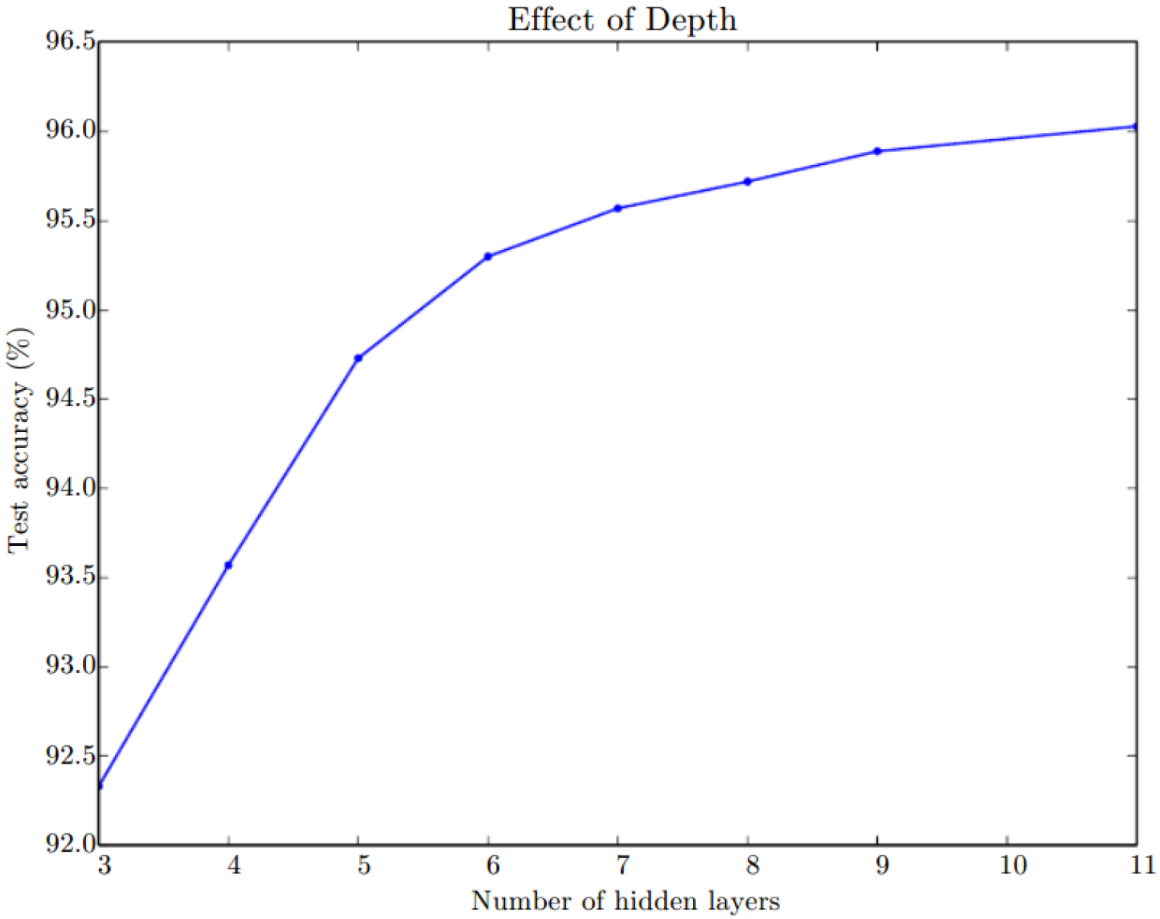
Improving the Performance

We can further improve the performance by doing the following:

- Train model for more epochs
- Use a different optimizer and/or loss function
- Increase the capacity by increasing the number of nodes per layer, such as instead of $64 - 16 - 1$, use $128 - 64 - 1$.
- Increase the capacity by increasing the number of layers, such as using $128 - 64 - 32 - 16 - 1$.

Architectural Considerations

Here are some plots that helps for gaining insight to model architecture design (taken from Goodfellow's book)



References

- A. Ng. Machine Learning, Lecture Notes.
- Arden Dertat's blog posts: https://medium.com/@ardendertat.
- I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning", 2016.