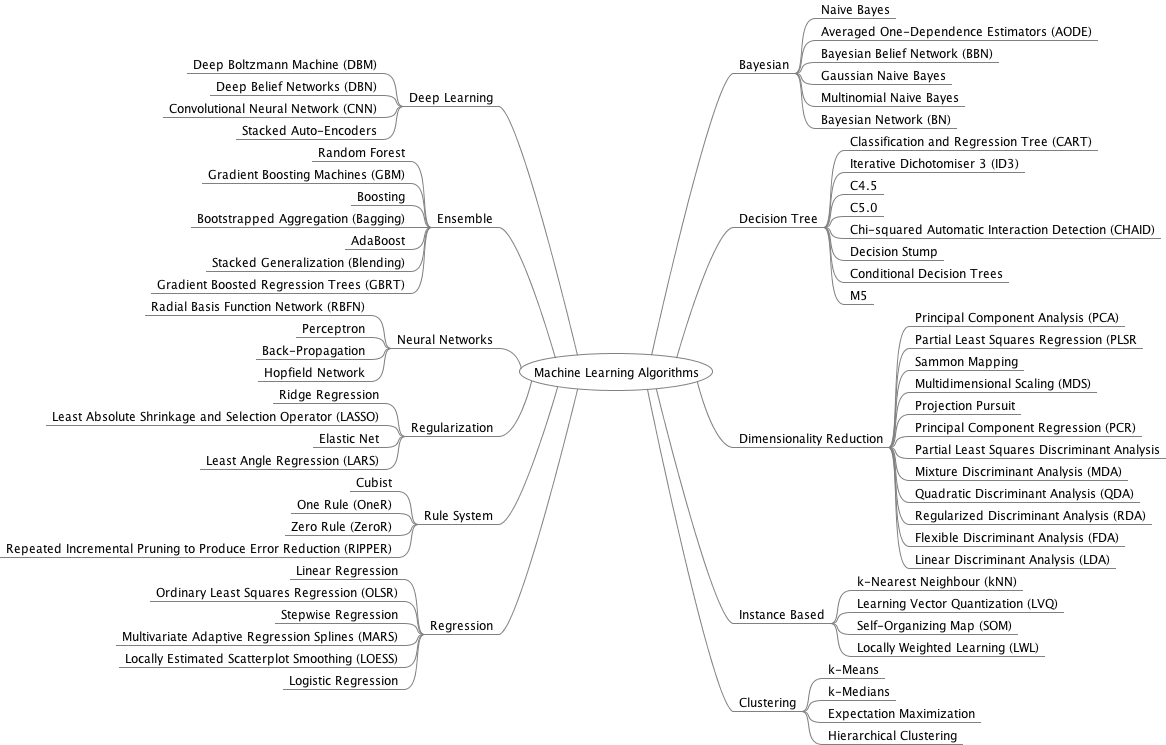
**What is machine learning?**

* Arthur Samuel (1959)
  + ***Machine learning:* "Field of study that gives computers the ability to learn without being explicitly programmed"**
    - Samuels wrote a checkers playing program
      * Had the program play 10000 games against itself
      * Work out which board positions were good and bad depending on wins/losses
* Tom Michel (1999)
  + ***Well posed learning problem:****"***A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."**
    - The checkers example,
      * E = 10000s games
      * T is playing checkers
      * P if you win or not

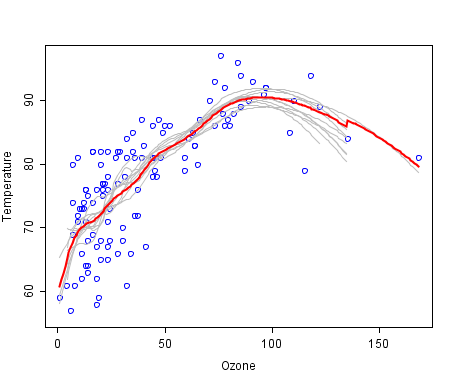


I want to give you two ways to think about and categorize the algorithms you may come across in the field.

* The first is a grouping of algorithms by the **learning style**.
* The second is a grouping of algorithms by **similarity** in form or function (like grouping similar animals together).

Both approaches are useful, but we will focus in on the grouping of algorithms by similarity and go on a tour of a variety of different algorithm types.

After reading this post, you will have a much better understanding of the most popular machine learning algorithms for supervised learning and how they are related.



A cool example of an ensemble of lines of best fit. Weak members are grey, the combined prediction is red.  
Plot from Wikipedia, licensed under public domain.

**Algorithms Grouped by Learning Style**

There are different ways an algorithm can model a problem based on its interaction with the experience or environment or whatever we want to call the input data.

It is popular in machine learning and artificial intelligence textbooks to first consider the learning styles that an algorithm can adopt.

**What is machine learning?**

* Here we...
  + Define what it is
  + When to use it
* Not a well defined definition
  + Couple of examples of how people have tried to define it
* Arthur Samuel (1959)
  + ***Machine learning:* "Field of study that gives computers the ability to learn without being explicitly programmed"**
    - Samuels wrote a checkers playing program
      * Had the program play 10000 games against itself
      * Work out which board positions were good and bad depending on wins/losses
* Tom Michel (1999)
  + ***Well posed learning problem:****"***A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."**
    - The checkers example,
      * E = 10000s games
      * T is playing checkers
      * P if you win or not
* Several types of learning algorithms
  + **Supervised learning**
    - Teach the computer how to do something, then let it use it;s new found knowledge to do it
  + **Unsupervised learning**
    - Let the computer learn how to do something, and use this to determine structure and patterns in data
  + Reinforcement learning
  + Recommender systems
* This course
  + Look at practical advice for applying learning algorithms

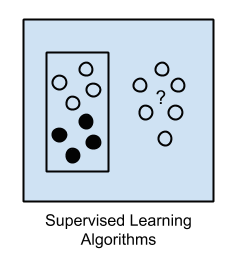
Learning a set of tools and **how** to apply

There are only a few main learning styles or learning models that an algorithm can have and we’ll go through them here with a few examples of algorithms and problem types that they suit.

This taxonomy or way of organizing machine learning algorithms is useful because it forces you to think about the roles of the input data and the model preparation process and select one that is the most appropriate for your problem in order to get the best result.

Let’s take a look at three different learning styles in machine learning algorithms:

**1. Supervised Learning**

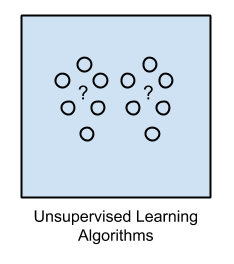
Input data is called training data and has a known label or result such as spam/not-spam or a stock price at a time.

A model is prepared through a training process in which it is required to make predictions and is corrected when those predictions are wrong. The training process continues until the model achieves a desired level of accuracy on the training data.

Example problems are classification and regression.

Example algorithms include Logistic Regression and the Back Propagation Neural Network.

**2. Unsupervised Learning**

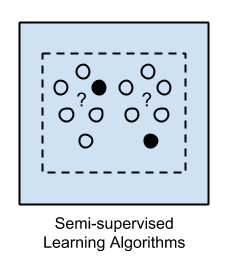
Input data is not labeled and does not have a known result.

A model is prepared by deducing structures present in the input data. This may be to extract general rules. It may be through a mathematical process to systematically reduce redundancy, or it may be to organize data by similarity.

Example problems are clustering, dimensionality reduction and association rule learning.

Example algorithms include: the Apriori algorithm and k-Means.

**3. Semi-Supervised Learning**

[](https://3qeqpr26caki16dnhd19sv6by6v-wpengine.netdna-ssl.com/wp-content/uploads/2013/11/Semi-supervised-Learning-Algorithms.png)Input data is a mixture of labeled and unlabelled examples.

There is a desired prediction problem but the model must learn the structures to organize the data as well as make predictions.

4.0 Reinforcement learning

The basic reinforcement is modeled as a [Markov decision process](https://en.wikipedia.org/wiki/Markov_decision_process):

1. a set of environment and agent states, *S*;
2. a set of actions, *A*, of the agent;
3. {\displaystyle P\_{a}(s,s')=Pr(s\_{t+1}=s'|s\_{t}=s,a\_{t}=a)} is the probability of transition from state s to state s' under action a.
4. {\displaystyle R\_{a}(s,s')} is the (expected) immediate reward after transition from *s* to {\displaystyle s'} with action *a*.
5. rules that describe what the agent observes

The rules are often [stochastic](https://en.wikipedia.org/wiki/Stochastic). The observation typically involves the scalar immediate reward associated with the last transition. In many works, the agent is assumed to observe the current environmental state (*full observability*). If not, the agent has *partial observability*. Sometimes the set of actions available to the agent is restricted (if a balance is zero, it cannot be reduced).

A reinforcement learning agent interacts with its environment in discrete time steps. At each time *t*, the agent receives an observation {\displaystyle o\_{t}}, which typically includes the reward {\displaystyle r\_{t}}. It then chooses an action {\displaystyle a\_{t}} from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state {\displaystyle s\_{t+1}} and the reward {\displaystyle r\_{t+1}} associated with the *transition* {\displaystyle (s\_{t},a\_{t},s\_{t+1})} is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The [agent](https://en.wikipedia.org/wiki/Software_agent) can choose any action as a function of the history and it can even randomize its [action selection](https://en.wikipedia.org/wiki/Action_selection).

When the agent's performance is compared to that of an agent that acts optimally, the difference in performance gives rise to the notion of *regret*. In order to act near optimally, the agent must reason about the long term consequences of its actions (i.e., maximize future income), although the immediate reward associated with this might be negative.

Thus, reinforcement learning is particularly well-suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including [robot control](https://en.wikipedia.org/wiki/Robot_control), elevator scheduling, [telecommunications](https://en.wikipedia.org/wiki/Telecommunications), [backgammon](https://en.wikipedia.org/wiki/Backgammon), [checkers](https://en.wikipedia.org/wiki/Checkers)[[3]](https://en.wikipedia.org/wiki/Reinforcement_learning#cite_note-FOOTNOTESuttonBartoChapter_11-3), and [go](https://en.wikipedia.org/wiki/Go_(game)) ([AlphaGo](https://en.wikipedia.org/wiki/AlphaGo" \o "AlphaGo)).

Example problems are classification and regression.4.0 ReExample algorithms are extensions to other flexible methods that make assumptions about how to model the unlabeled data.

**Overview**

When crunching data to model business decisions, you are most typically using supervised and unsupervised learning methods.

A hot topic at the moment is semi-supervised learning methods in areas such as image classification where there are large datasets with very few labeled examples.

**Algorithms Grouped By Similarity**

Algorithms are often grouped by similarity in terms of their function (how they work). For example, tree-based methods, and neural network inspired methods.

I think this is the most useful way to group algorithms and it is the approach we will use here.

This is a useful grouping method, but it is not perfect. There are still algorithms that could just as easily fit into multiple categories like Learning Vector Quantization that is both a neural network inspired method and an instance-based method. There are also categories that have the same name that describe the problem and the class of algorithm such as Regression and Clustering.

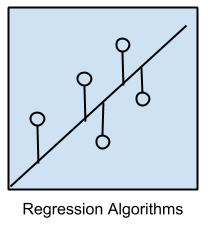
We could handle these cases by listing algorithms twice or by selecting the group that subjectively is the “best” fit. I like this latter approach of not duplicating algorithms to keep things simple.

In this section, I list many of the popular machine learning algorithms grouped the way I think is the most intuitive. The list is not exhaustive in either the groups or the algorithms, but I think it is representative and will be useful to you to get an idea of the lay of the land.

**Please Note**: There is a strong bias towards algorithms used for classification and regression, the two most prevalent supervised machine learning problems you will encounter.

Let’s dive in.

**Regression Algorithms**

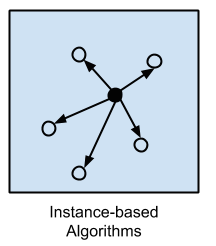
Regression is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model.

Regression methods are a workhorse of statistics and have been co-opted into statistical machine learning. This may be confusing because we can use regression to refer to the class of problem and the class of algorithm. Really, regression is a process.

The most popular regression algorithms are:

* Ordinary Least Squares Regression (OLSR)
* Linear Regression
* Logistic Regression
* Stepwise Regression
* Multivariate Adaptive Regression Splines (MARS)
* Locally Estimated Scatterplot Smoothing (LOESS)

**Instance-based Algorithms**

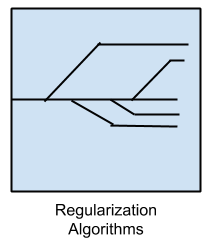
Instance-based learning model is a decision problem with instances or examples of training data that are deemed important or required to the model.

Such methods typically build up a database of example data and compare new data to the database using a similarity measure in order to find the best match and make a prediction. For this reason, instance-based methods are also called winner-take-all methods and memory-based learning. Focus is put on the representation of the stored instances and similarity measures used between instances.

The most popular instance-based algorithms are:

* k-Nearest Neighbor (kNN)
* Learning Vector Quantization (LVQ)
* Self-Organizing Map (SOM)
* Locally Weighted Learning (LWL)

**Regularization Algorithms**

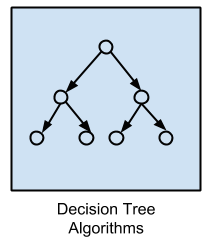
An extension made to another method (typically regression methods) that penalizes models based on their complexity, favoring simpler models that are also better at generalizing.

I have listed regularization algorithms separately here because they are popular, powerful and generally simple modifications made to other methods.

The most popular regularization algorithms are:

* Ridge Regression
* Least Absolute Shrinkage and Selection Operator (LASSO)
* Elastic Net
* Least-Angle Regression (LARS)

**Decision Tree Algorithms**

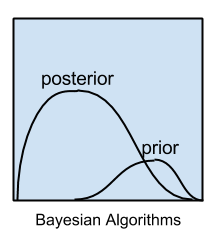
Decision tree methods construct a model of decisions made based on actual values of attributes in the data.

Decisions fork in tree structures until a prediction decision is made for a given record. Decision trees are trained on data for classification and regression problems. Decision trees are often fast and accurate and a big favorite in machine learning.

The most popular decision tree algorithms are:

* Classification and Regression Tree (CART)
* Iterative Dichotomiser 3 (ID3)
* C4.5 and C5.0 (different versions of a powerful approach)
* Chi-squared Automatic Interaction Detection (CHAID)
* Decision Stump
* M5
* Conditional Decision Trees

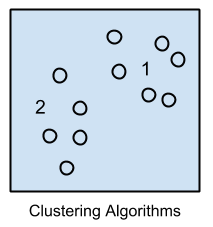
**Bayesian Algorithms**

Bayesian methods are those that explicitly apply Bayes’ Theorem for problems such as classification and regression.

The most popular Bayesian algorithms are:

* Naive Bayes
* Gaussian Naive Bayes
* Multinomial Naive Bayes
* Averaged One-Dependence Estimators (AODE)
* Bayesian Belief Network (BBN)
* Bayesian Network (BN)

**Clustering Algorithms**

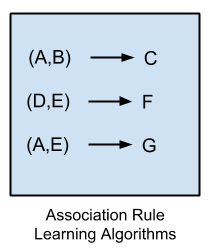
Clustering, like regression, describes the class of problem and the class of methods.

Clustering methods are typically organized by the modeling approaches such as centroid-based and hierarchal. All methods are concerned with using the inherent structures in the data to best organize the data into groups of maximum commonality.

The most popular clustering algorithms are:

* k-Means
* k-Medians
* Expectation Maximisation (EM)
* Hierarchical Clustering

**Association Rule Learning Algorithms**

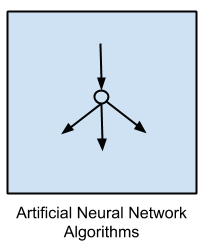
Association rule learning methods extract rules that best explain observed relationships between variables in data.

These rules can discover important and commercially useful associations in large multidimensional datasets that can be exploited by an organization.

The most popular association rule learning algorithms are:

* Apriori algorithm
* Eclat algorithm

**Artificial Neural Network Algorithms**

Artificial Neural Networks are models that are inspired by the structure and/or function of biological neural networks.

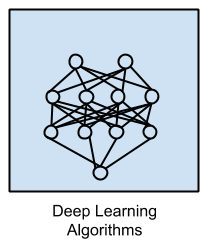
They are a class of pattern matching that are commonly used for regression and classification problems but are really an enormous subfield comprised of hundreds of algorithms and variations for all manner of problem types.

Note that I have separated out Deep Learning from neural networks because of the massive growth and popularity in the field. Here we are concerned with the more classical methods.

The most popular artificial neural network algorithms are:

* Perceptron
* Back-Propagation
* Hopfield Network
* Radial Basis Function Network (RBFN)

**Deep Learning Algorithms**

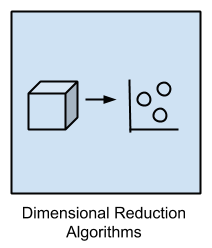
Deep Learning methods are a modern update to Artificial Neural Networks that exploit abundant cheap computation.

They are concerned with building much larger and more complex neural networks and, as commented on above, many methods are concerned with semi-supervised learning problems where large datasets contain very little labeled data.

The most popular deep learning algorithms are:

* Deep Boltzmann Machine (DBM)
* Deep Belief Networks (DBN)
* Convolutional Neural Network (CNN)
* Stacked Auto-Encoders

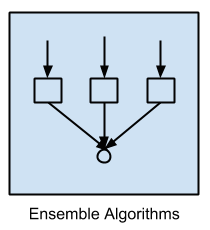
**Dimensionality Reduction Algorithms**

Like clustering methods, dimensionality reduction seek and exploit the inherent structure in the data, but in this case in an unsupervised manner or order to summarize or describe data using less information.

This can be useful to visualize dimensional data or to simplify data which can then be used in a supervised learning method. Many of these methods can be adapted for use in classification and regression.

* Principal Component Analysis (PCA)
* Principal Component Regression (PCR)
* Partial Least Squares Regression (PLSR)
* Sammon Mapping
* Multidimensional Scaling (MDS)
* Projection Pursuit
* Linear Discriminant Analysis (LDA)
* Mixture Discriminant Analysis (MDA)
* Quadratic Discriminant Analysis (QDA)
* Flexible Discriminant Analysis (FDA)

**Ensemble Algorithms**

Ensemble methods are models composed of multiple weaker models that are independently trained and whose predictions are combined in some way to make the overall prediction.

Much effort is put into what types of weak learners to combine and the ways in which to combine them. This is a very powerful class of techniques and as such is very popular.

* Boosting
* Bootstrapped Aggregation (Bagging)
* AdaBoost
* Stacked Generalization (blending)
* Gradient Boosting Machines (GBM)
* Gradient Boosted Regression Trees (GBRT)
* Random Forest

**Other Algorithms**

Many algorithms were not covered.

For example, what group would Support Vector Machines go into? Its own?

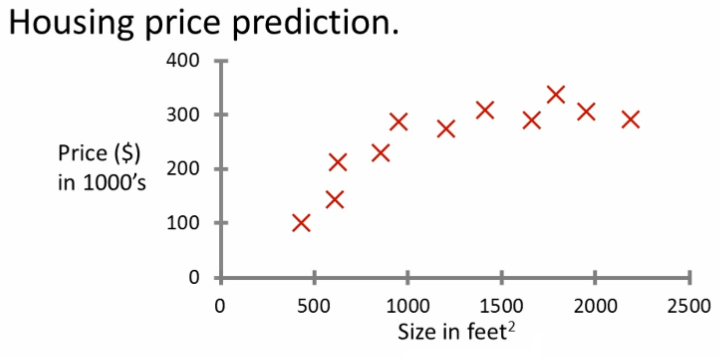
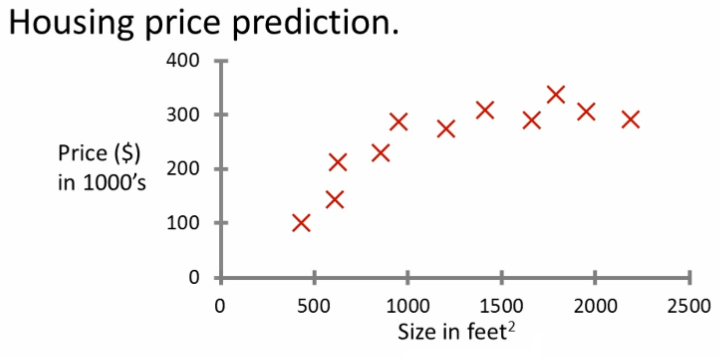
I did not cover algorithms from specialty tasks in the process of machine learning, such as:

* Feature selection algorithms
* Algorithm accuracy evaluation
* Performance measures

I also did not cover algorithms from specialty subfields of machine learning, such as:

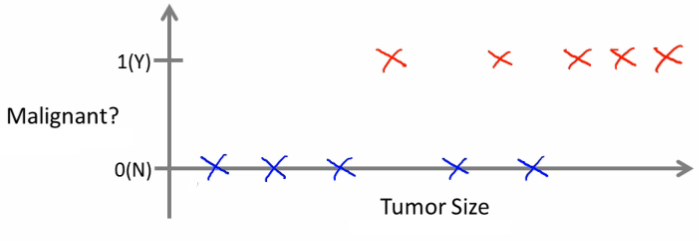
* Computational intelligence (evolutionary algorithms, etc.)
* Computer Vision (CV)
* Natural Language Processing (NLP)
* Recommender Systems
* Reinforcement Learning
* Graphical Models
* And more…

**Supervised learning - introduction**

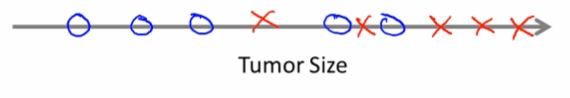
* Probably the most common problem type in machine learning
* Starting with an example
  + How do we predict housing prices
    - Collect data regarding housing prices and how they relate to size in feet
* **Example problem:** "Given this data, a friend has a house 750

square feet - how much can they be expected to get?"

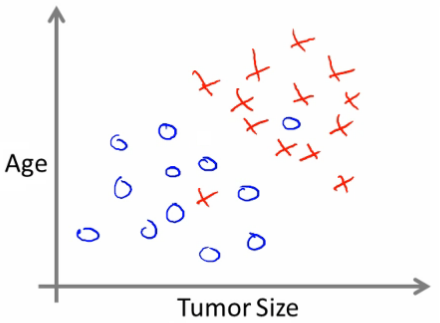
* What approaches can we use to solve this?
  + Straight line through data
    - Maybe $150 000
  + Second order polynomial
    - Maybe $200 000
  + One thing we discuss later - how to chose straight or curved line?
  + Each of these approaches represent a way of doing supervised learning
* *What does this mean?*
  + We gave the algorithm a data set where a "right answer" was provided
  + So we know actual prices for houses
    - The idea is we can learn what makes the price a certain value from the **training data**
    - The algorithm should then produce more right answers based on new training data where we don't know the price already
      * i.e. predict the price
* We also call this a **regression problem**
  + Predict continuous valued output (price)
  + No real discrete delineation
* Another example
  + Can we definer breast cancer as malignant or benign based on tumour size



* Looking at data
  + Five of each
  + Can you estimate prognosis based on tumor size?
  + This is an example of a **classification problem**
    - Classify data into one of two discrete classes - no in between, either malignant or not
    - In classification problems, can have a discrete number of possible values for the output
      * e.g. maybe have four values
        + 0 - benign
        + 1 - type 1
        + 2 - type 2
        + 3 - type 4
* In classification problems we can plot data in a different way



* Use only one attribute (size)
  + In other problems may have multiple attributes
  + We may also, for example, know age and tumor size

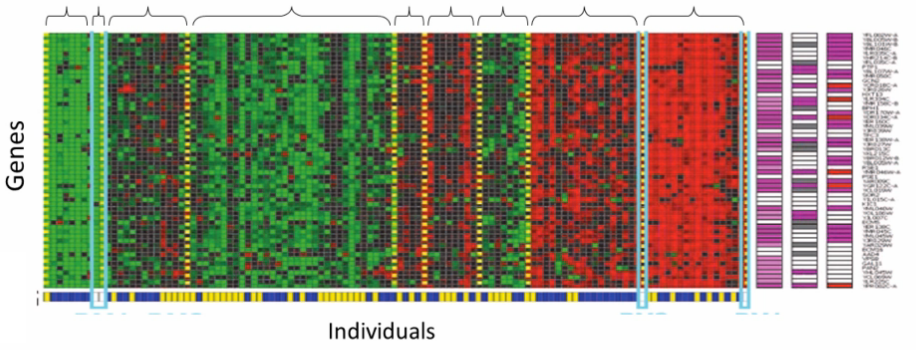
****

* Based on that data, you can try and define separate classes by
  + Drawing a straight line between the two groups
  + Using a more complex function to define the two groups (which we'll discuss later)
  + Then, when you have an individual with a specific tumor size and who is a specific age, you can hopefully use that information to place them into one of your classes
* You might have many features to consider
  + Clump thickness
  + Uniformity of cell size
  + Uniformity of cell shape
* The most exciting algorithms can deal with an infinite number of features
  + How do you deal with an infinite number of features?
  + Neat mathematical trick in support vector machine (which we discuss later)
    - If you have an infinitely long list - we can develop and algorithm to deal with that
* ***Summary***
  + Supervised learning lets you get the "right" answer
  + Regression problem
  + Classification problem

**Unsupervised learning - introduction**

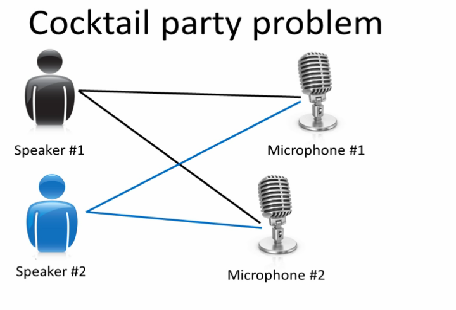
* Second major problem type
* In unsupervised learning, we get unlabeled data
  + Just told - here is a data set, can you structure it
* One way of doing this would be to cluster data into to groups
  + This is a **clustering algorithm**

**Clustering algorithm**

* Example of clustering algorithm
  + Google news
    - Groups news stories into cohesive groups
  + Used in any other problems as well
    - Genomics
    - Microarray data
      * Have a group of individuals
      * On each measure expression of a gene
      * Run algorithm to cluster individuals into types of people  
        
    - Organize computer clusters
      * Identify potential weak spots or distribute workload effectively
    - Social network analysis
      * Customer data
    - Astronomical data analysis
      * Algorithms give amazing results
* Basically
  + Can you automatically generate structure
  + Because we don't give it the answer, it's unsupervised learning

**Cocktail party algorithm**

* Cocktail party problem
  + Lots of overlapping voices - hard to hear what everyone is saying
    - Two people talking
    - Microphones at different distances from speakers



* Record slightly different versions of the conversation depending on where your microphone is
  + But overlapping none the less
* Have recordings of the conversation from each microphone
  + Give them to a cocktail party algorithm
  + Algorithm processes audio recordings
    - Determines there are two audio sources
    - Separates out the two sources
* Is this a very complicated problem
  + Algorithm can be done with one line of code!
  + **[W,s,v] = svd((repmat(sum(x.\*x,1), size(x,1),1).\*x)\*x');**
    - Not easy to identify
    - But, programs can be short!
    - Using octave (or MATLAB) for examples
      * Often prototype algorithms in octave/MATLAB to test as it's very fast
      * Only when you show it works migrate it to C++
      * Gives a much faster agile development
* Understanding this algorithm
  + **svd** - linear algebra routine which is built into octave
    - In C++ this would be very complicated!
  + Shown that using MATLAB to prototype is a really good way to do this

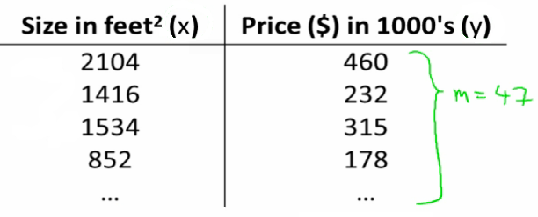
Summary

-Unsupervised learning has no “right” answer

-Clustering is an important tool

**Linear Regression**

* Housing price data example used earlier
  + Supervised learning regression problem
* What do we start with?
  + Training set (this is your data set)
  + Notation (*used throughout the course*)
    - m = number of **training examples**
    - x's = input variables / features
    - y's = output variable "target" variables
      * (x,y) - single training example
      * (xi, yj) - specific example (ith training example)
        + i is an index to training set



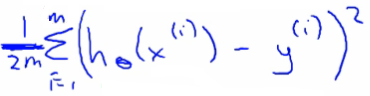
* With our training set defined - how do we used it?
  + Take training set
  + Pass into a learning algorithm
  + Algorithm outputs a function (denoted *h*) (h = **hypothesis**).Hypothesis is the term used for this function in ML
    - This function takes an input (e.g. size of new house)
    - Tries to output the estimated value of Y
* How do we represent hypothesis *h*?
  + Going to present h as;
    - hθ(x) = θ0 + θ1x
      * h(x) (shorthand)

http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr_files/Image%20%5b7%5d.png

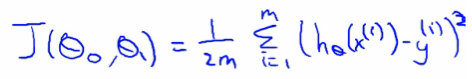
* What does this mean?
  + Means Y is a linear function of x!
  + θi are **parameters**
    - θ0 is zero condition
    - θ1 is gradient
* This kind of function is a linear regression with one variable
  + Also called **univariate linear regression**
* So in summary
  + A hypothesis takes in some variable
  + Uses parameters determined by a learning system
  + Outputs a prediction based on that input

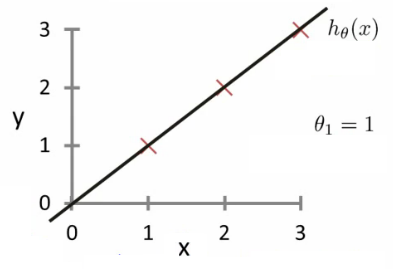
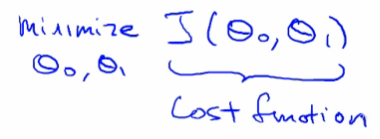
**Linear regression - implementation (cost function)**

* A cost function lets us figure out how to fit the best straight line to our data
* Choosing values for θi (parameters)
  + Different values give you different functions
  + If θ0 is 1.5 and θ1 is 0 then we get straight line parallel with X along 1.5 @ y
  + If θ1 is > 0 then we get a positive slope
* Based on our training set we want to generate parameters which make the straight line
  + Chosen these parameters so hθ(x) is close to y for our training examples
    - Basically, uses xs in training set with hθ(x) to give output which is as close to the actual y value as possible
    - Think of hθ(x) as a "y imitator" - it tries to convert the x into y, and considering we already have y we can evaluate how well hθ(x) does this
* To formalize this;
  + We want to want to solve a **minimization problem**
  + Minimize (hθ(x) - y)2
    - i.e. minimize the difference between h(x) and y for each/any/every example
  + Sum this over the training set



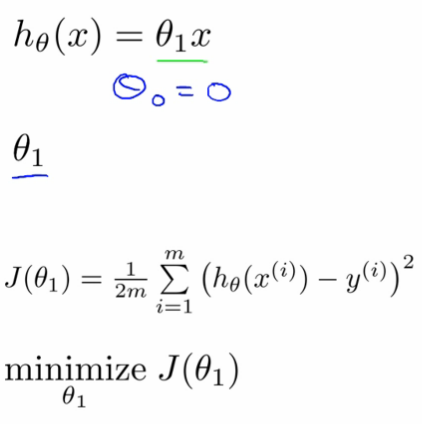
* Minimize squared different between predicted house price and actual house price
  + 1/2m
    - 1/m - means we determine the average
    - 1/2m the 2 makes the math a bit easier, and doesn't change the constants we determine at all (i.e. half the smallest value is still the smallest value!)
  + Minimizing θ0/θ1 means we get the values of θ0 and θ1 which find on average the minimal deviation of x from y when we use those parameters in our hypothesis function
* More cleanly, this is a cost function

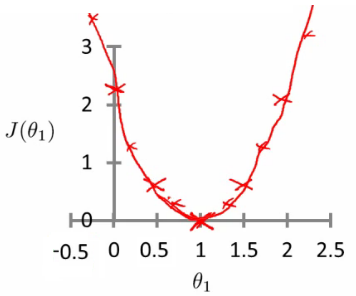


* And we want to minimize this cost function
  + Our cost function is (because of the summartion term) inherently looking at ALL the data in the training set at any time
* **So to recap**
  + **Hypothesis** - is like your prediction machine, throw in an *x* value, get a putative *y* value  
    
  + **Cost** - is a way to, using your training data, determine values for your θ values which make the hypothesis as accurate as possible  
    
    - This cost function is also called the squared error cost function
      * This cost function is reasonable choice for most regression functions
      * Probably most commonly used function
  + In case J(θ0,θ1) is a bit abstract, going into what it does, why it works and how we use it in the coming sections

**Cost function - a deeper look**

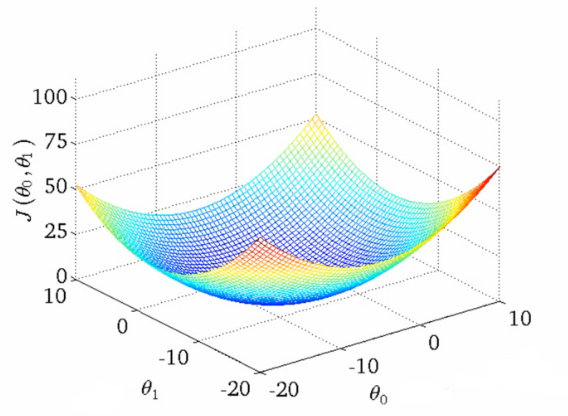
* Lets consider some intuition about the cost function and why we want to use it
  + The cost function determines parameters
  + The value associated with the parameters determines how your hypothesis behaves, with different values generate different
* Simplified hypothesis
  + Assumes θ0 = 0



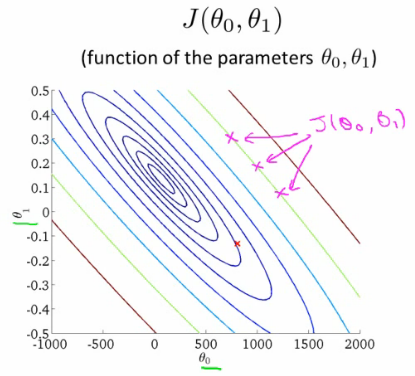
* Cost function and goal here are very similar to when we have θ0, but with a simpler parameter
  + Simplified hypothesis makes visualizing cost function J() a bit easier
* So hypothesis pass through 0,0
* Two key functins we want to understand
  + hθ(x)
    - Hypothesis is a function of x - function of what the size of the house is
  + J(θ1)
    - Is a function of the parameter of θ1
  + So for example
    - θ1 = 1
    - J(θ1) = 0
  + Plot
    - θ1 vs J(θ1)
    - Data
      * 1)
        + θ1 = 1
        + J(θ1) = 0
      * 2)
        + θ1 = 0.5
        + J(θ1) = ~0.58
      * 3)
        + θ1 = 0
        + J(θ1) = ~2.3
  + If we compute a range of values plot
    - J(θ1) vs θ1 we get a polynomial (looks like a quadratic)  
      
* The optimization objective for the learning algorithm is find the value of θ1 which minimizes J(θ1)
  + So, here θ1 = 1 is the best value for θ1

**A deeper insight into the cost function - simplified cost function**

* Assume you're familiar with contour plots or contour figures
  + Using same cost function, hypothesis and goal as previously
  + It's OK to skip parts of this section if you don't understand cotour plots
* Using our original complex hyothesis with two pariables,
  + So cost function is
    - J(θ0, θ1)
* Example,
  + Say
    - θ0 = 50
    - θ1 = 0.06
  + Previously we plotted our cost function by plotting
    - θ1 vs J(θ1)
  + Now we have two parameters
    - Plot becomes a bit more complicated
    - Generates a 3D surface plot where axis are
      * X = θ1
      * Z = θ0
      * Y = J(θ0,θ1)



* We can see that the height (y) indicates the value of the cost function, so find where y is at a minimum
* Instead of a surface plot we can use a **contour figures/plots**
  + Set of ellipses in different colors
  + Each colour is the same value of J(θ0, θ1), but obviously plot to different locations because θ1 and θ0 will vary
  + Imagine a bowl shape function coming out of the screen so the middle is the concentric circles

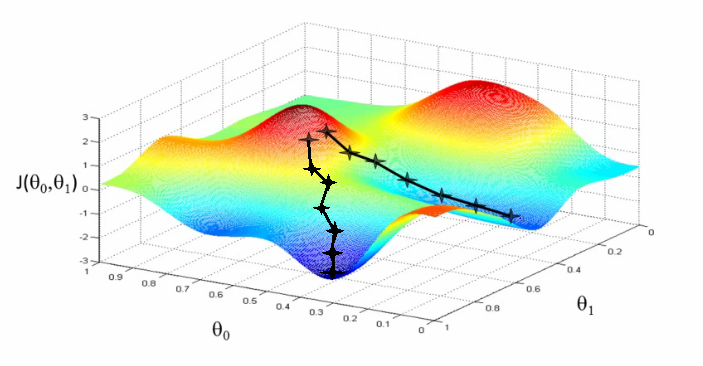


* Each point (like the red one above) represents a pair of parameter values for Ɵ0 and Ɵ1
  + Our example here put the values at
    - θ0 = ~800
    - θ1 = ~-0.15
  + Not a good fit
    - i.e. these parameters give a value on our contour plot far from the center
  + If we have
    - θ0 = ~360
    - θ1 = 0
    - This gives a better hypothesis, but still not great - not in the center of the countour plot
  + Finally we find the minimum, which gives the best hypothesis
* Doing this by eye/hand is a pain in the ass
  + What we really want is an efficient algorithm fro finding the minimum for θ0 and θ1

**Gradient descent algorithm**

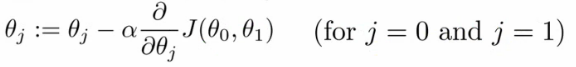
* Minimize cost function J
* Gradient descent
  + Used all over machine learning for minimization
* Start by looking at a general J() function
* Problem
  + We have J(θ0, θ1)
  + We want to get **min J(θ0, θ1)**
* Gradient descent applies to more general functions
  + J(θ0, θ1, θ2 .... θn)
  + min J(θ0, θ1, θ2 .... θn)

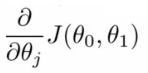
**How does it work?**

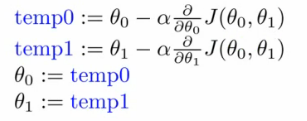
* Start with initial guesses
  + Start at 0,0 (or any other value)
  + Keeping changing θ0 and θ1 a little bit to try and reduce J(θ0,θ1)
* Each time you change the parameters, you select the gradient which reduces J(θ0,θ1) the most possible
* Repeat
* Do so until you converge to a local minimum
* Has an interesting property
  + Where you start can determine which minimum you end up  
    
  + Here we can see one initialization point led to one local minimum
  + The other led to a different one

**A more formal definition**

* Do the following until covergence

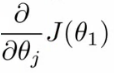


* What does this all mean?
  + Update θj by setting it to (θj - α) times the partial derivative of the cost function with respect to θj
* Notation
  + :=
    - Denotes assignment
    - NB a = b is a *truth assertion*
  + α (alpha)
    - Is a number called the **learning rate**
    - Controls how big a step you take
      * If α is big have an aggressive gradient descent
      * If α is small take tiny steps
* Derivative term  
  
  + Not going to talk about it now, derive it later
* There is a subtly about how this gradient descent algorithm is implemented
  + Do this for θ0 and θ1
  + For j = 0 and j = 1 means we **simultaneously**update both
  + How do we do this?
    - Compute the right hand side for both θ0and θ1
      * So we need a temp value
    - Then, update θ0and θ1 at the same time
    - We show this graphically below



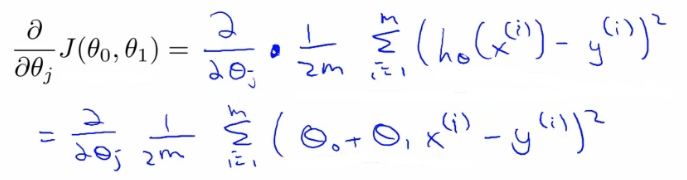
* If you implement the non-simultaneous update it's not gradient descent, and will behave weirdly
  + But it might look sort of right - so it's important to remember this!

**Understanding the algorithm**

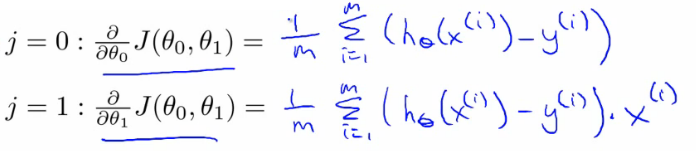
* To understand gradient descent, we'll return to a simpler function where we minimize one parameter to help explain the algorithm in more detail
  + min θ1 J(θ1) where θ1 is a real number
* Two key terms in the algorithm
  + Alpha
  + Derivative term
* Notation nuances
  + Partial derivative vs. derivative
    - Use partial derivative when we have multiple variables but only derive with respect to one
    - Use derivative when we are deriving with respect to all the variables
* Derivative term  
          
  + Derivative says
    - Lets take the tangent at the point and look at the slope of the line
    - So moving towards the mimum (down) will greate a negative derivative, alpha is always positive, so will update j(θ1) to a smaller value
    - Similarly, if we're moving up a slope we make j(θ1) a bigger numbers
* Alpha term (α)
  + What happens if alpha is too small or too large
  + Too small
    - Take baby steps
    - Takes too long
  + Too large
    - Can overshoot the minimum and fail to converge
* When you get to a local minimum
  + Gradient of tangent/derivative is 0
  + So derivative term = 0
  + alpha \* 0 = 0
  + So θ1 = θ1- 0
  + So θ1 remains the same
* As you approach the global minimum the derivative term gets smaller, so your update gets smaller, even with alpha is fixed
  + Means as the algorithm runs you take smaller steps as you approach the minimum
  + So no need to change alpha over time

**Linear regression with gradient descent**

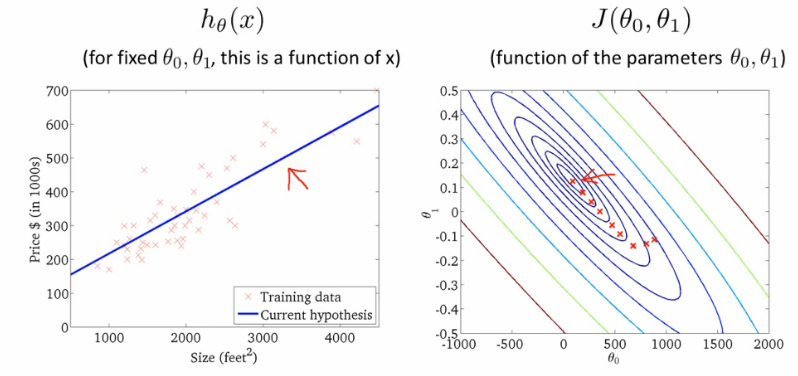
* Apply gradient descent to minimize the squared error cost function J(θ0, θ1)
* Now we have a partial derivative



* So here we're just expanding out the first expression
  + J(θ0, θ1) = 1/2m....
  + hθ(x) = θ0 + θ1\*x
* So we need to determine the derivative for each parameter - i.e.
  + When j = 0
  + When j = 1
* Figure out what this partial derivative is for the θ0 and θ1 case
  + When we derive this expression in terms of j = 0 and j = 1 we get the following



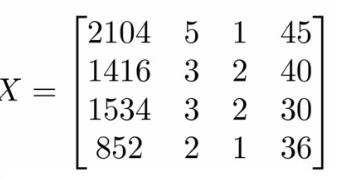
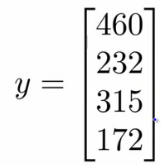
* To check this you need to know multivariate calculus
  + So we can plug these values back into the gradient descent algorithm
* How does it work
  + Risk of meeting different local optimum
  + The linear regression cost function is always a **convex function** - always has a single minimum
    - Bowl shaped
    - One global optima
      * So gradient descent will always converge to global optima
  + In action
    - Initialize values to
      * θ0 = 900
      * θ1 = -0.1



* End up at a global minimum
* This is actually **Batch Gradient Descent**
  + Refers to the fact that over each step you look at all the training data
    - Each step compute over m training examples
  + Sometimes non-batch versions exist, which look at small data subsets
    - We'll look at other forms of gradient descent (to use when m is too large) later in the course
* There exists a numerical solution for finding a solution for a minimum function
  + **Normal equations** method
  + Gradient descent scales better to large data sets though
  + Used in lots of contexts and machine learning

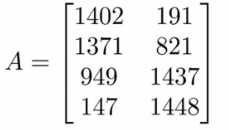
**What's next - important extensions**  
*Two extension to the algorithm*

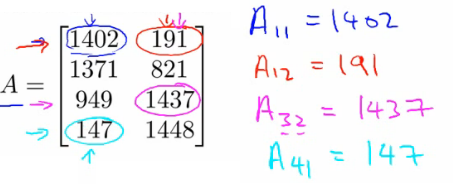
* **1) Normal equation for numeric solution**
  + To solve the minimization problem we can solve it [ min J(θ0, θ1) ] exactly using a numeric method which avoids the iterative approach used by gradient descent
  + Normal equations method
  + Has advantages and disadvantages
    - Advantage
      * No longer an alpha term
      * Can be much faster for some problems
    - Disadvantage
      * Much more complicated
  + We discuss the normal equation in the **linear regression with multiple features** section
* **2) We can learn with a larger number of features**
  + So may have other parameters which contribute towards a prize
    - e.g. with houses
      * Size
      * Age
      * Number bedrooms
      * Number floors
    - x1, x2, x3, x4
  + With multiple features becomes hard to plot
    - Can't really plot in more than 3 dimensions
    - Notation becomes more complicated too
      * Best way to get around with this is the notation of linear algebra
      * Gives notation and set of things you can do with matrices and vectors
      * e.g. Matrix

* We see here this matrix shows us
  + Size
  + Number of bedrooms
  + Number floors
  + Age of home
* All in one variable
  + Block of numbers, take all data organized into one big block
* Vector
  + Shown as *y*
  + Shows us the prices
* Need linear algebra for more complex linear regression modles
* Linear algebra is good for making computationally efficient models (as seen later too)
  + Provide a good way to work with large sets of data sets
  + Typically vectorization of a problem is a common optimization technique

**Matrices - overview**

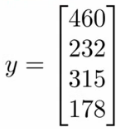
* Rectangular array of numbers written between square brackets
  + 2D array
  + Named as capital letters (A,B,X,Y)
* Dimension of a matrix are [Rows x Columns]
  + Start at top left
  + To bottom left
  + To bottom right
  + R[r x c] means a matrix which has r rows and c columns  
    
    - Is a [4 x 2] matrix
* Matrix elements
  + A(i,j) = entry in ith row and jth column



* Provides a way to organize, index and access a lot of data

**Vectors - overview**

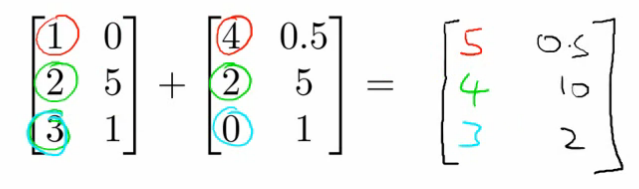
* Is an n by 1 matrix
  + Usually referred to as a lower case letter
  + n rows
  + 1 column
  + e.g.



* Is a 4 dimensional vector
  + Refer to this as a vector R4
* Vector elements
  + vi = ithelement of the vector
  + Vectors can be 0-indexed (C++) or 1-indexed (MATLAB)
  + In math 1-indexed is most common
    - But in machine learning 0-index is useful
  + Normally assume using 1-index vectors, but be aware sometimes these will (explicitly) be 0 index ones

**Matrix manipulation**

* ***Addition***
  + Add up elements one at a time
  + Can only add matrices of the *same dimensions*
    - Creates a new matrix of the same dimensions of the ones added



* ***Multiplication by scalar***
  + Scalar = real number
  + Multiply each element by the scalar
  + Generates a matrix of the same size as the original matrix