WEEK\_1:

Machine Learning

Supervised Learning

Unsupervised Learning

Supervised Learning:

ML:Linear Regression with One Variable

Model Representation

The Hypothesis Function

Cost Function

ML:Gradient Descent

Gradient Descent for Linear Regression

ML:Linear Algebra Review

Matrices and Vectors

Addition and Scalar Multiplication

Matrix-Vector Multiplication

Matrix-Matrix Multiplication

Inverse and Transpose

WEEK\_2:

ML:Linear Regression with Multiple Variables

Cost function

Gradient Descent for Multiple Variables

Matrix Notation

Feature Normalization

Gradient Descent Tips

Features and Polynomial Regression

Polynomial Regression

Normal Equation

Normal Equation Noninvertibility

WEEk\_3:

ML:Logistic Regression

Binary Classification

Decision Boundary

Cost Function

Simplified Cost Function and Gradient Descent

Advanced Optimization

Multiclass Classification: One-vs-all

ML:Regularization

Cost Function

Regularized Linear Regression

Normal Equation

Regularized Logistic Regression

WEEK\_4:

ML:Neural Networks: Representation

Non-linear Hypotheses

Neurons and the Brain

Model Representation I

Model Representation II

Examples and Intuitions I

Examples and Intuitions II

Multiclass Classification

WEEK\_5:

ML:Neural Networks: Learning

Cost Function

Backpropagation Algorithm

Backpropagation Intuition

Implementation Note: Unrolling Parameters

Gradient Checking

Random Initialization

Putting it Together

Explanation of Derivatives Used in Backpropagation

WEEK\_6:

ML:Advice for Applying Machine Learning

Deciding What to Try Next

Evaluating a Hypothesis

Model Selection and Train/Validation/Test Sets

Without the Validation Set (note: this is a bad method - do not use it)

Use of the CV set

With the Validation Set (note: this method presumes we do not also use the CV set for regularization)

Diagnosing Bias vs. Variance

Regularization and Bias/Variance

Learning Curves

Deciding What to Do Next Revisited

Diagnosing Neural Networks

Model Selection:

ML:Machine Learning System Design

Prioritizing What to Work On

Error Analysis

Error Metrics for Skewed Classes

Trading Off Precision and Recall

Data for Machine Learning

WEEK\_7:

Support Vector Machine (SVM)

Large Margin Intuition

Mathematics Behind Large Margin Classification (Optional)

Kernels I

Kernels II

Choosing SVM Parameters

Using An SVM

Multi-class Classification

Logistic Regression vs. SVMs

WEEK\_8:

ML:Clustering

Unsupervised Learning: Introduction

K-Means Algorithm

Optimization Objective

Random Initialization

Choosing the Number of Clusters

Bonus: Discussion of the drawbacks of K-Means

ML:Dimensionality Reduction

Motivation I: Data Compression

Motivation II: Visualization

Principal Component Analysis Problem Formulation

Principal Component Analysis Algorithm

Reconstruction from Compressed Representation

Choosing the Number of Principal Components

Advice for Applying PCA

WEEK\_9:

ML:Anomaly Detection

Problem Motivation

Gaussian Distribution

Developing and Evaluating an Anomaly Detection System

Anomaly Detection vs. Supervised Learning

Choosing What Features to Use

ML:Recommender Systems

Problem Formulation

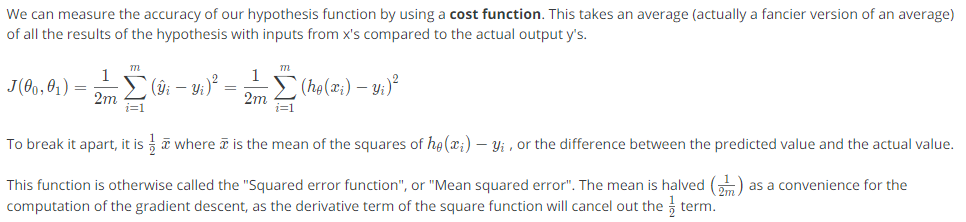
Content Based Recommendations

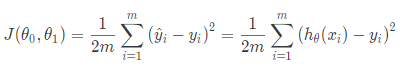
Collaborative Filtering

Collaborative Filtering Algorithm

Vectorization: Low Rank Matrix Factorization

Implementation Detail: Mean Normalization



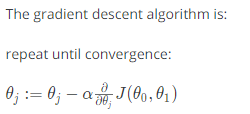


"Squared error function", or "Mean squared error"

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line (defined by hθ(x)) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of J(θ,θ1) will be 0.

**ML:Gradient Descent**

**So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in hypothesis function. That's where gradient descent comes in.**

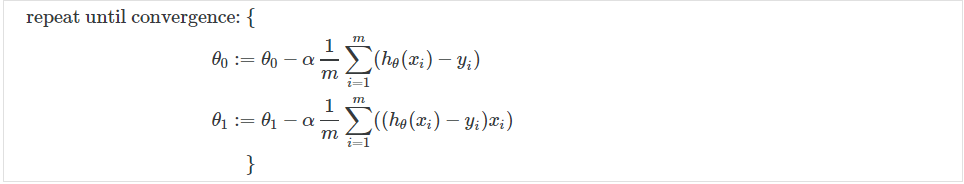


where

j=0,1 represents the feature index number.

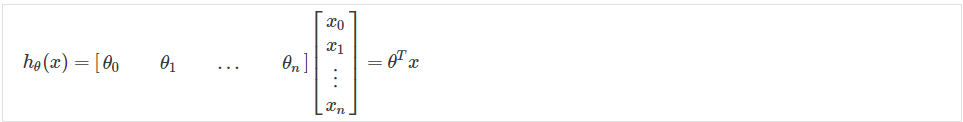
### ****Gradient Descent for Linear Regression****

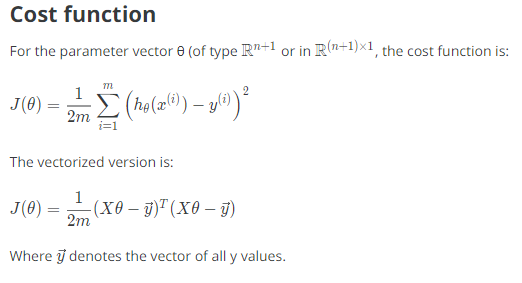
When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas are out of the scope of this course, but a really great one can be found here):

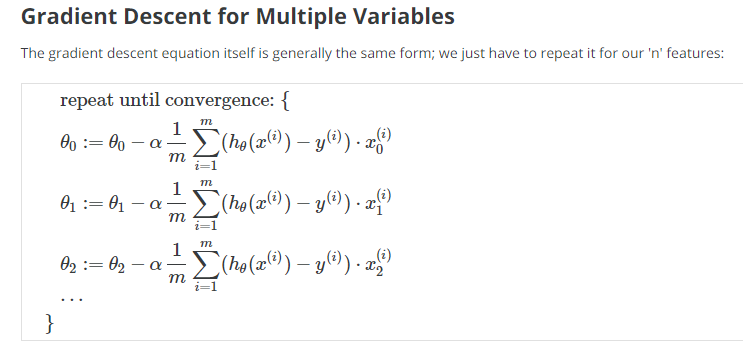


# ML:Linear Regression with Multiple Variables









# Feature Normalization

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

−1 ≤ x\_{(i)}*x*(*i*)​ ≤ 1

or

−0.5 ≤ x\_{(i)}*x*(*i*)​ ≤ 0.5

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

Two techniques to help with this are **feature scaling** and **mean normalization**.



Where μ\_i*μi*​ is the **average** of all the values for feature (i) and s\_i*si*​ is the range of values (max - min), or s\_i*si*​ is the standard deviation.

# Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways.

We can **combine** multiple features into one. For example, we can combine x\_1*x*1​ and x\_2*x*2​ into a new feature x\_3*x*3​ by taking x\_1*x*1​⋅x\_2*x*2​.

### ****Polynomial Regression****

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

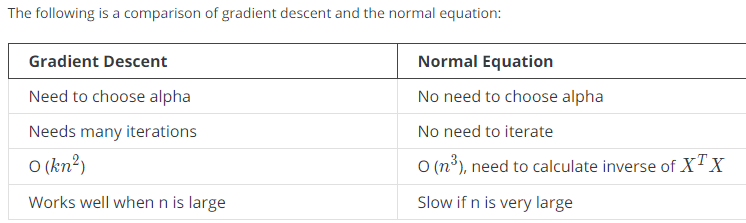
eg. if x\_1*x*1​ has range 1 - 1000 then range of x\_1^2*x*12​ becomes 1 - 1000000 and that of x\_1^3*x*13​ becomes 1 - 1000000000.

# Normal Equation

The "Normal Equation" is a method of finding the optimum theta **without iteration.**



There is **no need** to do feature scaling with the normal equation.



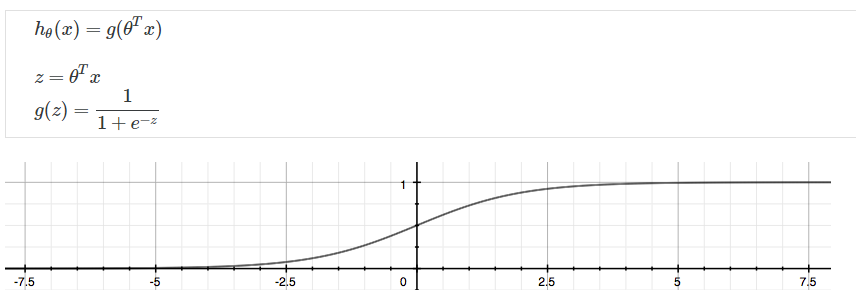
# ML:Logistic Regression

Now we are switching from regression problems to **classification problems**. Don't be confused by the name "Logistic Regression"; it is named that way for historical reasons and is actually an approach to classification problems, not regression problems.

# ****Binary Classification****

Instead of our output vector y being a continuous range of values, it will only be 0 or 1.

One method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. This method doesn't work well because classification is not actually a linear function.



# ML:Regularization

**The Problem of Overfitting**

Regularization is designed to address the problem of overfitting.

High bias or underfitting is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. eg. if we take h\_\theta(x) = \theta\_0 + \theta\_1x\_1 + \theta\_2x\_2*hθ*​(*x*)=*θ*0​+*θ*1​*x*1​+*θ*2​*x*2​ then we are making an initial assumption that a linear model will fit the training data well and will be able to generalize but that may not be the case.

At the other extreme, overfitting or high variance is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

This terminology is applied to both linear and logistic regression. There are two main options to address the issue of overfitting:

1) Reduce the number of features:

a) Manually select which features to keep.

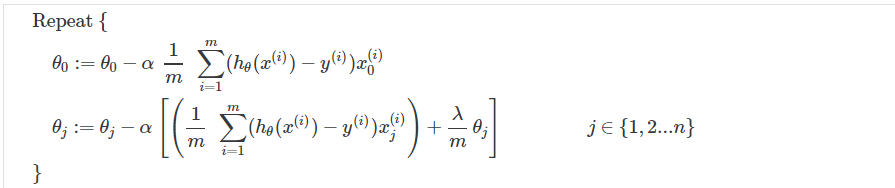
b) Use a model selection algorithm (studied later in the course).

2) Regularization

Keep all the features, but reduce the parameters \theta\_j*θj*​.

Regularization works well when we have a lot of slightly useful features.

# Regularized Linear Regression



# ML:Neural Networks: Representation

# Non-linear Hypotheses

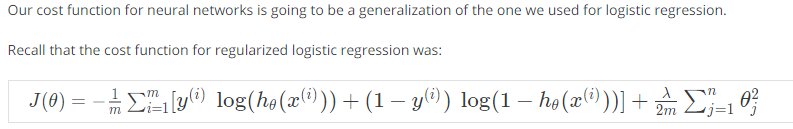
Neural networks offers an alternate way to perform machine learning when we have complex hypotheses with many features.

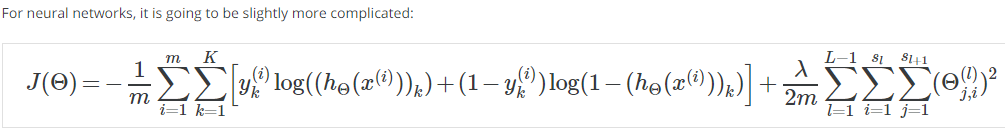
# Neurons and the Brain

Neural networks are limited imitations of how our own brains work. They've had a big recent resurgence because of advances in computer hardware.

There is evidence that the brain uses only one "learning algorithm" for all its different functions. Scientists have tried cutting (in an animal brain) the connection between the ears and the auditory cortex and rewiring the optical nerve with the auditory cortex to find that the auditory cortex literally learns to see.

At a very simple level, neurons are basically computational units that take input (**dendrites**) as electrical input (called "spikes") that are channeled to outputs (**axons**).





We have added a few nested summations to account for our multiple output nodes. In the first part of the equation, between the square brackets, we have an additional nested summation that loops through the number of output nodes.

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

# Backpropagation Algorithm

"Backpropagation" is neural-network terminology for minimizing our cost function, just like what we were doing with gradient descent in logistic and linear regression.

# ML:Advice for Applying Machine Learning

Errors in your predictions can be troubleshooted by:

* Getting more training examples
* Trying smaller sets of features
* Trying additional features
* Trying polynomial features
* Increasing or decreasing λ

Don't just pick one of these avenues at random. We'll explore diagnostic techniques for choosing one of the above solutions in the following sections.

Model Selection and Train/Validation/Test Sets

* Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis.
* The error of your hypothesis as measured on the data set with which you trained the parameters will be lower than any other data set.

In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result.

**Use of the CV set**

To solve this, we can introduce a third set, the **Cross Validation Set**, to serve as an intermediate set that we can train d with. Then our test set will give us an accurate, non-optimistic error.

One example way to break down our dataset into the three sets is:

* Training set: 60%
* Cross validation set: 20%
* Test set: 20%

We can now calculate three separate error values for the three different sets.

**With the Validation Set (note: this method presumes we do not also use the CV set for regularization)**

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with J\_{test}(\Theta^{(d)})*Jtest*​(Θ(*d*)), (d = theta from polynomial with lower error);

This way, the degree of the polynomial d has not been trained using the test set.

(Mentor note: be aware that using the CV set to select 'd' means that we cannot also use it for the validation curve process of setting the lambda value).

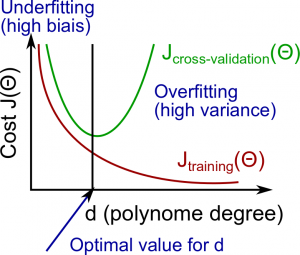
Diagnosing Bias vs. Variance

In this section we examine the relationship between the degree of the polynomial d and the underfitting or overfitting of our hypothesis.

* We need to distinguish whether **bias** or **variance** is the problem contributing to bad predictions.
* High bias is underfitting and high variance is overfitting. We need to find a golden mean between these two.

The training error will tend to **decrease** as we increase the degree d of the polynomial.

At the same time, the cross validation error will tend to **decrease** as we increase d up to a point, and then it will **increase** as d is increased, forming a convex curve.

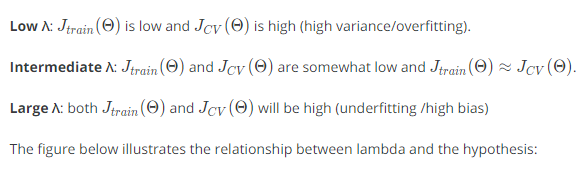


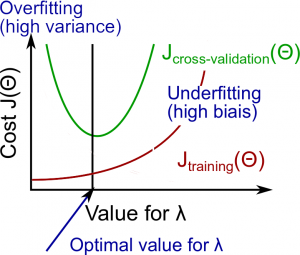
Regularization and Bias/Variance

Instead of looking at the degree d contributing to bias/variance, now we will look at the regularization parameter λ.

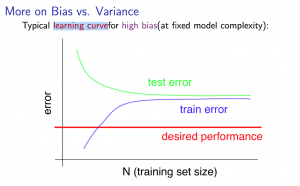
* Large λ: High bias (underfitting)
* Intermediate λ: just right
* Small λ: High variance (overfitting)

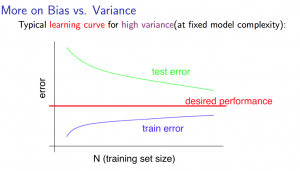
A large lambda heavily penalizes all the Θ parameters, which greatly simplifies the line of our resulting function, so causes underfitting.

The relationship of λ to the training set and the variance set is as follows: 



If a learning algorithm is suffering from **high variance**, getting more training data is **likely to help.**





Deciding What to Do Next Revisited

Our decision process can be broken down as follows:

* Getting more training examples

Fixes high variance

* Trying smaller sets of features

Fixes high variance

* Adding features

Fixes high bias

* Adding polynomial features

Fixes high bias

* Decreasing λ

Fixes high bias

* Increasing λ

Fixes high variance

## Model Selection:

Choosing M the order of polynomials.

How can we tell which parameters Θ to leave in the model (known as "model selection")?

There are several ways to solve this problem:

* Get more data (very difficult).
* Choose the model which best fits the data without overfitting (very difficult).
* Reduce the opportunity for overfitting through regularization.

**Intuition for the bias-variance trade-off:**

* Complex model => sensitive to data => much affected by changes in X => high variance, low bias.
* Simple model => more rigid => does not change as much with changes in X => low variance, high bias.

One of the most important goals in learning: finding a model that is just right in the bias-variance trade-off.

**Regularization Effects:**

* Small values of λ allow model to become finely tuned to noise leading to large variance => overfitting.
* Large values of λ pull weight parameters to zero leading to large bias => underfitting.

**Model Complexity Effects:**

* Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
* Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.
* In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

**A typical rule of thumb when running diagnostics is:**

* More training examples fixes high variance but not high bias.
* Fewer features fixes high variance but not high bias.
* Additional features fixes high bias but not high variance.
* The addition of polynomial and interaction features fixes high bias but not high variance.
* When using gradient descent, decreasing lambda can fix high bias and increasing lambda can fix high variance (lambda is the regularization parameter).
* When using neural networks, small neural networks are more prone to under-fitting and big neural networks are prone to over-fitting. Cross-validation of network size is a way to choose alternatives.

Support Vector Machine

The **Support Vector Machine** (SVM) is yet another type of *supervised* machine learning algorithm. It is sometimes cleaner and more powerful.

### ****Logistic Regression vs. SVMs****

If n is large (relative to m), then use logistic regression, or SVM without a kernel (the "linear kernel")

If n is small and m is intermediate, then use SVM with a Gaussian Kernel

If n is small and m is large, then manually create/add more features, then use logistic regression or SVM without a kernel.

In the first case, we don't have enough examples to need a complicated polynomial hypothesis. In the second example, we have enough examples that we may need a complex non-linear hypothesis. In the last case, we want to increase our features so that logistic regression becomes applicable.

**Note**: a neural network is likely to work well for any of these situations, but may be slower to train.

Finally, note that the hypothesis of the Support Vector Machine is not interpreted as the probability of y being 1 or 0 (as it is for the hypothesis of logistic regression). Instead, it outputs either 1 or 0. (In technical terms, it is a discriminant function.)

# Large Margin Intuition

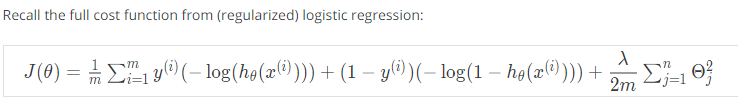
A useful way to think about Support Vector Machines is to think of them as Large Margin Classifiers.

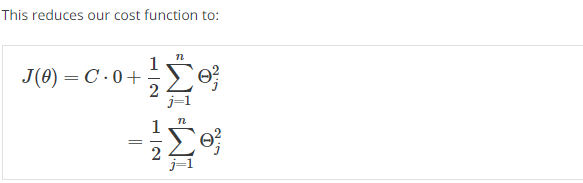
Recall the decision boundary from logistic regression (the line separating the positive and negative examples). In SVMs, the decision boundary has the special property that it is **as far away as possible** from both the positive and the negative examples.

The distance of the decision boundary to the nearest example is called the **margin**. Since SVMs maximize this margin, it is often called a *Large Margin Classifier*.

The SVM will separate the negative and positive examples by a **large margin**.

Recall the cost function for (unregularized) logistic regression:





Recall the decision boundary from logistic regression (the line separating the positive and negative examples). In SVMs, the decision boundary has the special property that it is **as far away as possible** from both the positive and the negative examples.

The distance of the decision boundary to the nearest example is called the **margin**. Since SVMs maximize this margin, it is often called a *Large Margin Classifier*.

The SVM will separate the negative and positive examples by a **large margin**.

This large margin is only achieved when **C is very large**.

Data is **linearly separable** when a **straight line** can separate the positive and negative examples.

If we have **outlier** examples that we don't want to affect the decision boundary, then we can **reduce** C.

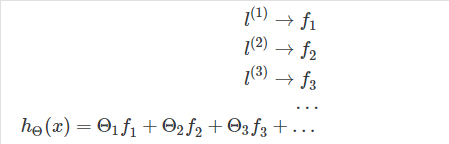
Increasing and decreasing C is similar to respectively decreasing and increasing λ, and can simplify our decision boundary.

# Kernels I

**Kernels** allow us to make complex, non-linear classifiers using Support Vector Machines.

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

To do this, we find the "similarity" of x and some landmark l^{(i)}*l*(*i*):



ML:Clustering

Unsupervised Learning: Introduction

Unsupervised learning is contrasted from supervised learning because it uses an **unlabeled** training set rather than a labeled one.

In other words, we don't have the vector y of expected results, we only have a dataset of features where we can find structure.

Clustering is good for:

* Market segmentation
* Social network analysis
* Organizing computer clusters
* Astronomical data analysis

K-Means Algorithm

The K-Means Algorithm is the most popular and widely used algorithm for automatically grouping data into coherent subsets.

1. Randomly initialize two points in the dataset called the *cluster centroids*.
2. Cluster assignment: assign all examples into one of two groups based on which cluster centroid the example is closest to.
3. Move centroid: compute the averages for all the points inside each of the two cluster centroid groups, then move the cluster centroid points to those averages.
4. Re-run (2) and (3) until we have found our clusters.

# Choosing the Number of Clusters

Choosing K can be quite arbitrary and ambiguous.

**The elbow method**: plot the cost J and the number of clusters K. The cost function should reduce as we increase the number of clusters, and then flatten out. Choose K at the point where the cost function starts to flatten out.

However, fairly often, the curve is **very gradual**, so there's no clear elbow.

**Note:** J will **always** decrease as K is increased. The one exception is if k-means gets stuck at a bad local optimum.

Another way to choose K is to observe how well k-means performs on a **downstream purpose**. In other words, you choose K that proves to be most useful for some goal you're trying to achieve from using these clusters.

# ML:Dimensionality Reduction

**Data Compression**

* We may want to reduce the dimension of our features if we have a lot of redundant data.
* To do this, we find two highly correlated features, plot them, and make a new line that seems to describe both features accurately. We place all the new features on this single line.

Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

Note: in dimensionality reduction, we are reducing our features rather than our number of examples. Our variable m will stay the same size; n, the number of features each example from x^{(1)}*x*(1) to x^{(m)}*x*(*m*) carries, will be reduced.

### ****Data Visualization****

It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to 3 or less in order to plot it.

We need to find new features, z\_1, z\_2*z*1​,*z*2​(and perhaps z\_3*z*3​) that can effectively **summarize** all the other features.

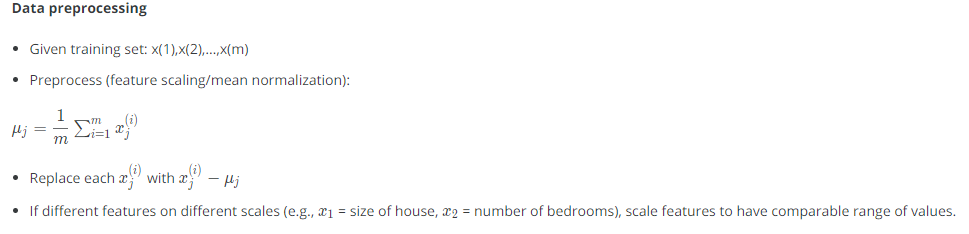
Example: hundreds of features related to a country's economic system may all be combined into one feature that you call "Economic Activity."

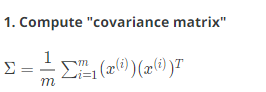
Given two features, x\_1*x*1​ and x\_2*x*2​, we want to find a single line that effectively describes both features at once. We then map our old features onto this new line to get a new single feature.

The same can be done with three features, where we map them to a plane.

# Principal Component Analysis Algorithm

Before we can apply PCA, there is a data pre-processing step we must perform:







[U,S,V] = svd(Sigma);

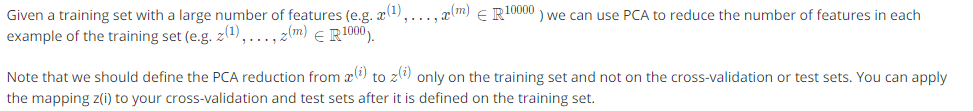
**3. Take the first k columns of the U matrix and compute z**

We'll assign the first k columns of U to a variable called 'Ureduce'. This will be an n×k matrix. We compute z with:



## Advice for Applying PCA

The most common use of PCA is to speed up supervised learning.





Applications

* Compressions

Reduce space of data

Speed up algorithm

* Visualization of data

Choose k = 2 or k = 3

Don't assume you need to do PCA. **Try your full machine learning algorithm without PCA first.** Then use PCA if you find that you need it.

# ML:Anomaly Detection

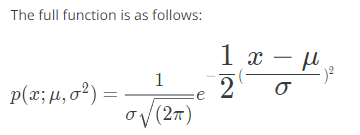
Just like in other learning problems, we are given a dataset

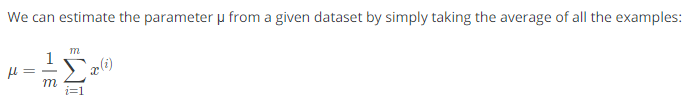
We are then given a new example, x\_{test}*xtest*​, and we want to know whether this new example is abnormal/anomalous.

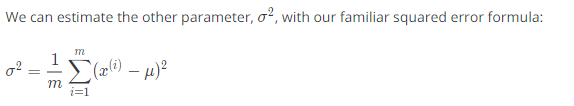
We define a "model" p(x) that tells us the probability the example is not anomalous. We also use a threshold ϵ (epsilon) as a dividing line so we can say which examples are anomalous and which are not.

# Gaussian Distribution

The Gaussian Distribution is a familiar bell-shaped curve that can be described by a function \mathcal{N}(\mu,\sigma^2)N(*μ*,*σ*2)







Anomaly Detection vs. Supervised Learning

When do we use anomaly detection and when do we use supervised learning?

Use anomaly detection when...

* We have a very small number of positive examples (y=1 ... 0-20 examples is common) and a large number of negative (y=0) examples.
* We have many different "types" of anomalies and it is hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

Use supervised learning when...

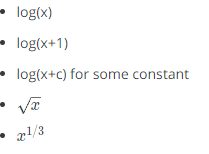
* We have a large number of both positive and negative examples. In other words, the training set is more evenly divided into classes.
* We have enough positive examples for the algorithm to get a sense of what new positives examples look like. The future positive examples are likely similar to the ones in the training set.

# Choosing What Features to Use

The features will greatly affect how well your anomaly detection algorithm works.

We can check that our features are **gaussian** by plotting a histogram of our data and checking for the bell-shaped curve.

Some **transforms** we can try on an example feature x that does not have the bell-shaped curve are:



# ML:Recommender Systems

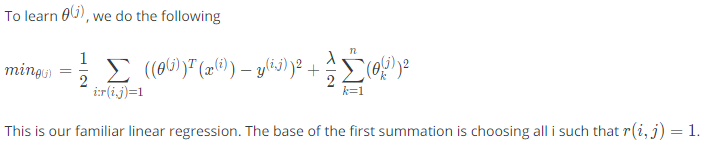
Recommendation is currently a very popular application of machine learning.

Say we are trying to recommend movies to customers. We can use the following definitions

# Content Based Recommendations

We can introduce two features, x\_1*x*1​ and x\_2*x*2​ which represents how much romance or how much action a movie may have (on a scale of 0−1).

One approach is that we could do linear regression for every single user. For each user j, learn a parameter \theta^{(j)} \in \mathbb{R}^3*θ*(*j*)∈R3. Predict user j as rating 

# Collaborative Filtering

# Vectorization: Low Rank Matrix Factorization

# Implementation Detail: Mean Normalization