Table of Contents

[Supervised Learning 1](#_Toc29203192)

[1. Regression 2](#_Toc29203193)

[Linear Regression with One Variable 2](#_Toc29203194)

[**Evaluate Linear Regression: Cost function** 2](#_Toc29203195)

[**Estimate Parameters in Hypothesis: Gradient Descent** 5](#_Toc29203196)

[**Linear Algebra: Matrices and Vectors** 8](#_Toc29203197)

[Linear Regression with Multiple Variables 10](#_Toc29203198)

[**Speed Up Gradient Descent: Feature Scaling** 11](#_Toc29203199)

[**Speed Up Gradient Descent: choose learning rate**  11](#_Toc29203200)

[**Feature Selection and Polynomial Regression** 12](#_Toc29203201)

[Alternative to GD: Normal Equation 13](#_Toc29203202)

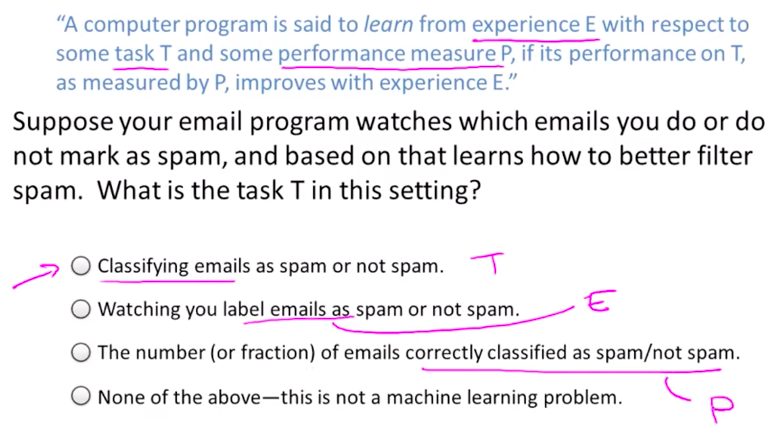
**Machine Learning Definition**

Arthur Samuel (1959): older, informal

A field of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell (1998): modern

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.



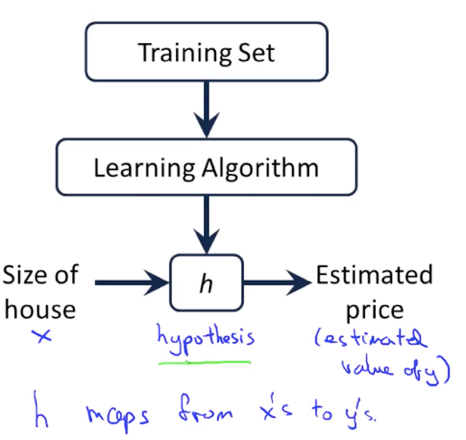
**Supervised Learning:** given the “right answer” for each example in the data

Regression: predict continuous outputs (e.g. housing prices)

Classification: predict discrete outputs (e.g. 0 benign, 1 malignant; 0 benign, 1 type I cancer, 2 type II cancer, 3 type III cancer)

**1. Regression**

**Linear Regression with One Variable**



Hypothesis function:

**Evaluate Linear Regression: Cost function**

Measure the accuracy of hypothesis function

i.e. squared error function/ mean squared error

* m = # of training examples
* The mean is halved (1/2) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the 1/2 term.

|  |
| --- |
| % Vectorization is converting code that relies on loops into matrix operations. It is more efficient and more concise.  % Using vectorization to implement linear regression is faster than using for loop.  % ============ Unvectorized implementation ============  prediction = 0;  for j = 1:n+1, % 1 to n + 1 instead of 0 to n  prediction += theta(j) \* x(j)  end;  % ============ Vectorized implementation ============  % X: design matrix containing training examples  % y: true output  % work for either one or multiple variables  function J = computeCost(X, y, theta)  m = length(y); % number of training examples  prediction = X \* theta % vectorization  sqrErrors = (prediction - y) .^ 2;  J = 1/(2 \* m) \* sum(sqrErrors);  end |

Assume

* Hypothesis function is a function of x, for fixed
* Cost function is a function of

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

Assume

* Hypothesis function is a function of x, for fixed
* Cost function is a function of

A 3D plot of cost function:

A close up of a map

Description automatically generated

A contour plot of cost function:

Taking any colour and going along the 'circle', one would expect to get the same value of the cost function J.

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Very close to min cost

**Estimate Parameters in Hypothesis: Gradient Descent**

An algorithm to find that minimize cost function J

Can be applied to more general function:

Standing at a point, which direction should I go to get to the lowest point?

🡪 Take partial derivative of the cost function (i.e. the slope of the tangent line)

🡪 Step down the cost function in the direction with the steepest descent

🡪 The size of each step is determined by learning rate : a larger , a larger step

🡪 Uses all training examples in each step, called “batch gradient descent”

🡪 Depending on where one starts on the graph, one can end up at different minimum points (susceptible to local minima).

A close up of a map

Description automatically generatedA close up of a piece of paper

Description automatically generated

A close up of a map

Description automatically generated

Repeat until convergence, update equation {

simultaneously update all , for j = 0 and j = 1

}

For j = 0:

For j = 1:

Gradient descent algorithm for linear regression:

Since linear regression only has one global minimum point, gradient descent always converges.

Repeat until convergence {

simultaneously update all

}

*update simultaneously at each iteration*

A screenshot of a cell phone

Description automatically generated

Assignment: a := b take the value of b and set a equal to it

a := a + 1

Truth assertion: a = b

~~a = a + 1~~

|  |
| --- |
| % work for either one or multiple variables  function [theta, J\_history] = gradientDescent(X, y, theta, alpha, num\_iters)  m = length(y);  J\_history = zeros(num\_iters, 1);    for iter = 1:num\_iters  theta = theta - (alpha / m) \* (X' \* (X \* theta - y)); % vectorization  % save cost J in every iteration  J\_history(iter) = computeCost(X, y, theta);  end  end |

Consider a simple cost function Repeat until convergence:

A close up of text on a white background

Description automatically generated

If is too small, gradient descent takes long time to converge.

A close up of a map

Description automatically generated

If is too large, gradient descent can overshoot the minimum. Since gradients have opposite signs on either side of the minimum, this could result in diverging oscillations. may oscillate inefficiently or diverge to .

A close up of a map

Description automatically generated

Choose such that gradient descent algorithm converges in a reasonable time.

How does gradient descent converge with a fixed ?

Derivative approaches 0 as we approach the bottom of convex cost function. So gradient descent will automatically take smaller steps towards a local minimum, we don’t need to decrease over time.

A close up of a map

Description automatically generated

### **Linear Algebra: Matrices and Vectors**

Matrix Vector Multiplication: Prediction = Data Matrix \* Parameters

+ More computationally efficient in most programming languages

e.g.

e.g. Three competing hypotheses:

Properties of Matrices multiplication:

* Not commutative: except for B = I
* Associative:

Inverse of 2x2 matrix

det(A) = |A|

A picture containing object

Description automatically generated

Inverse of 3x3 matrix:

Gauss-Jordan Elimination: reduced to row echelon form. If it returns a row of zero, not invertible.

A close up of a clock

Description automatically generated🡪 A close up of a clock

Description automatically generated🡪A close up of a clock

Description automatically generated🡪A close up of a clock

Description automatically generated

A non-square matrix doesn’t have an inverse matrix.

Matrices that don’t have inverse are non-invertible, called “singular” or “degenerate”.

Identity matrix:

For square matrix:

Transpose: 🡪

A close up of a sign

Description automatically generated A picture containing object

Description automatically generated

A picture containing object

Description automatically generated 

Use capital letters to denote matrices, lower-case letters to denote vectors.

## **Linear Regression with Multiple Variables**

n = # of features j = 0, ... n

m = # of training examples i = 1, ... m

= value of ith training example

= value of feature j in ith training example

e.g.

A screenshot of a cell phone

Description automatically generated

Hypothesis function (define :

Input value of ith training example:

Hypothesis parameters:

Cost function:

Gradient descent:

Repeat until convergence {

simultaneously update all for j = 0, ..., n

}

....

### **Speed Up Gradient Descent: Feature Scaling**

Feature Scaling: make features be in roughly the same range

If features have varying ranges of values, it is difficult to set a learning rate.

A feature with larger values has larger gradients. A learning rate which works well for the smaller features will cause overshooting of corrections for the larger features (due to larger gradients). may oscillate inefficiently or diverge to .

不一定非改成某个特定的range,只要features的ranges都差不多就可 e.g.

* Divided by max:
* Mean normalization: (do not apply to )
* Standardization:

*!!! When predicting, do the same feature scaling: use the same mu and sigma computed during training.*

e.g. 0 – 2000 square feet 🡪

1- 5 bedrooms 🡪

A close up of a logo

Description automatically generated A close up of a logo

Description automatically generated

|  |
| --- |
| % Standardization  function [X\_norm, mu, sigma] = featureNormalize(X)  mu = mean(X)  sigma = std(X)  X\_norm = (X-mean(X))./std(X)  end  *When predicting and normalizing test data, do not call this function.*  *Manually normalize using mu and sigma computed before.* |

### **Speed Up Gradient Descent: choose learning rate**

Check if GD is working properly: try a range of different , plot cost function J against # of iterations

For sufficiently small , J should decrease after each iteration.

If 𝛼 is too small, gradient descent converges slowly.

A close up of text on a white background

Description automatically generated A picture containing object

Description automatically generated

If J increases (not converge) or doesn’t decrease on every iteration, that means is too large.

oscillates and J increases.

Solution: use smaller

A picture containing object

Description automatically generatedA close up of a logo

Description automatically generated A close up of a map

Description automatically generated

### **Feature Selection and Polynomial Regression**

1). Combine multiple features into one: length \* width 🡪 area

2). Fit a polynomial hypothesis function:

e.g.

Let

*must do feature scaling in this case*

A close up of a map

Description automatically generatedA close up of a map

Description automatically generated

|  |
| --- |
| function [X\_poly] = polyFeatures(X, p)  % Takes a vector X (m, 1) and maps each example into its polynomial features (p-th power)  % Return a matrix X\_poly (m, p): i-th column is the i-th power of X  % Column 1: original X  % Column 2: X .^ 2  % Column 3: X .^ 3  % X\_poly(i, :) = [X(i) X(i).^2 X(i).^3 ... X(i).^p]  m = size(X, 1);  X\_poly = zeros(m, p);  for i = 1:p  X\_poly(:,i) = X .^ i;  end  end |

## **Alternative to GD: Normal Equation**

Another algorithm to find that minimizes cost function J

Solve for in one calculation. There is no “loop until convergence” like in gradient descent.

n = number of features j = 0, ... n

m = size of training set i = 1, ... m

Hypothesis function (with :

Hypothesis parameters:

Input value of ith training example: a column vector with n + 1 rows with =1

Design matrix: m \* n+1 matrix, each row is ith training example

m-dimensional vector

Cost function:

Replace sum and square by matric multiplication & Remove which is to ease derivation

Cost function:

A screenshot of a cell phone

Description automatically generatede.g. For m training examples and two features

Gradient descent:

- Need to choose

- Need many iterations

- Need feature scaling

+ Work well even when n is large,

Normal equation:

+ No need to choose

+ No need to iterate

+ No need to do feature scaling

- Needs to compute , higher complexity , slow if n is very large (>10,000)

**Non-invertibility**

Why is non-invertible:

* Redundant features: linearly independent (e.g. = size in , = size in )
* Too many features: size of training set (m) number of features (n)

(maybe non-invertible if m = n)

Solution: delete some features, or use regularization

|  |
| --- |
| function [theta] = normalEqn(X, y)  % pinv calculates pseudo inverse, works even when is non-invertible  theta = pinv(X'\*X) \* X' \* y;  end |