1.Introduction

What's Machine Learning?

- Arthur Samuel's definition

> Field of study that gives computers the ability to learn without being expplicityly programmed.

- Tom Mitchell's definition

> 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.

- Classfications:
    - Supervised Learning
    - Unsupervised Learning

Supervised Learning

- definition

    > we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output

- problems
    - regression

        > we are trying to predict results within a continuous output,meaning that we are trying to map input variables to some continuous fuction.

    - classification

        > we are instead trying to predict results in a discrete output.
        >
        > we are trying to map input varibles into discrete categories.

Unsupervised Learning

- definition

    > unsupervised learning allows us to approach problems with little or no idea what our results should look lke. We can derive structure form data where we don't neccessarily know of the variables.
    >
    > We can derive this structure by clustering the data based on relationships among the variables in the data.

2.Linear Regression with One Variable

Model Repr esentation

- notation

    $(x^{(i)}, y^{(i)})$: training example . input-output or input_features-target

    m : a training set

    X / Y : the space of input/output values . X = Y = R

$h(x)$: a predicor for the corresponding value of y . called a hypothesis

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

- problem
  - regression problem

    when the target variable that we're trying to predict is coninuous.
  - classification problem

    when y can take on only a small number of discrete values.

Cost Function

- function
  - hypothesis function

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

    θ_0, θ_1
  - cost function / mean squared error function

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

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

Gradient Descent

- Outline:
  - start with some parameters

    common choise : $\theta_0 = 0, \theta_1 = 0$
  - keep changing θ_0, θ_1 to reduce $j(\theta_0 = \theta_1)$

    until we hopefully end up at a local minimum
- algorithm

  > repeat simultaneously until convergence:
  >
  > { $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)\ (for\ j = 0\ and\ j = 1)$
  >
  > }

α : learning rate. controls how big a step
  - α is too small: gradient descent can be slow
  - α is too large: gradient descent can overshoot the minimum.

    It may fail to converge,or even diverge.

- batch

  batch gradient descent: refers to the fact that in every step of gradient descent, we're looking at all of the training examples.

3.Linear Algebra Review

Matrices and Vectors

Matrix：Rectangular array of numbers

Vector：An n*1 matrix

Matrix Multiplication Pr operties

$$A * B \neq B * A$$

$$(A * B) * C = A * (B * C)$$

Identity Matrix ：denoted I_{n*n}

$$AI = IA = A$$

Inverse and Transpose

`inverseofA = inv(A)`

`transposeofA = A'`

4.Linear Regression with Multiple Varialbe

Multiple Featur es

n : number of features

$x^{(i)}$: input features of i^{th} training example (R^n)

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

Gradient Descent for Multiple V ariables

Hypothesis : $h_\theta(x) = \theta^T x$

Parameters : $\theta = [\theta_0\ \theta_1 \ldots \theta_n]$

Cost function: $J(\theta) = \frac{1}{2m}\Sigma_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

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

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

Gradient Descent in Pr actice I - Featrue Scaling

Idea: Make sure features are on a similar scale.

Get every feature into approximaltely a $-1 \le x_i \le 1$ range.

- Mean normalization

  repalce x_i with $x_i - \mu_i$ to make features have approximately 0 mean

$$x_1 := \frac{x_1 - \mu_1}{s_1}$$

  μ_1:average value of x1 in training set

  s_1:range(max - min)

Gradient Descent in Pr actice II - 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.

Features and Polynomial Regr ession

We can change the behavior or curve of our hypothesis function by making it a quadratic,cubic or square root function(or any other form).

Normal Equation

To compute θ ,and then minimize J function:

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

Gradient Descent	Normal Equation
1.Need to choose α	1.No need to choose α
2.Need many iterations	2.Do not need to iterate
3. $O(kn^2)$Work well enven when n is large	3.Need to compute $(X^T X)^{-1}$
	4.$O(n^3)$Slow if n is very large

Normal Equation and Non-invertibility

`inv()`:$(X^T X)$ can't be non-invertible

`pinv()`:$(X^T X)$ even be non-invertible (pseudo-inverse)

Noninvertible's causes:

1. Redundant features, where tow features are very closely related(i.e. linearly dependent)
2. Too many fetures(i.e. m$\leq$n). We should delete some features or use regularization.

5.Octave/Matlab Tutorial

Basic Oper ations

```
>> a = pi
a =  3.1416
>> disp(sprintf('show 3 decimals: %0.3f',a))
show 3 decimals: 3.142
>> format long  % change the format of value
>> a
a =  3.14159265358979
```

```
>> a = 1:0.3:2
a =
    1.0000    1.3000    1.6000    1.9000
```

```
>> a = zeros(1,3)
a =
   0   0   0
```

```
>> a = 3*ones(2,4)
a =
   3   3   3   3
   3   3   3   3
```

```
>> a=rand(2,3)
a =
   0.25599   0.54999   0.34204
   0.73561   0.31876   0.76891
```

```
>> a = randn(2,3) % Gaussian distribution with mean=0
a =
   0.32532  -0.26668   0.15293
   0.29101  -0.25914  -0.36278
```

```
>> a = 5 + 4*randn(1000,1);
>> hist(a) % get the pic
```

```
>> a = eye(3)
a =
Diagonal Matrix
   1   0   0
   0   1   0
   0   0   1
```

Moving Data Around

```
>> a = [1 2 ;3 4;5 6]
a =
   1   2
   3   4
   5   6
>> a(2,2)
ans =  4
>> a(3,:)
ans =
   5   6
>> a=[a,[11;12;13]]
a =
    1    2   11
    3    4   12
    5    6   13
>> size(a,1)
ans =  3
>> size(a,2)
ans =  2
>> length(a)
ans =  3
```

```
>> pwd
ans = /home/yule
>> who
Variables in the current scope:
a    ans  b    sz
>> whos
Variables in the current scope:
   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        a           3x2                         48  double
        ans         1x10                        10  char
        b           1x2                          2  char
        sz          1x2                         16  double
Total is 20 elements using 76 bytes
>> clear
>> load featuresX.dat
>> load('featuresX.dat)
>> featuresX %show the data
>> size(featuresX)
ans =
        47      2
>> v = featuresX(1:10)
>> save save_v.mat v
>> save save_v.txt v
>> load save_v.mat
```

```
>> a=[1 2;3 4]
a =
   1   2
   3   4
>> a(:)
ans =
   1
   3
   2
   4
```

Computing on Data

```
>> a
a =

   1   2
   3   4
   5   6
>> a >3
ans =
   0   0
   0   1
   1   1
>> find(a>3)
ans =
   3
   5
   6
>> max(a) % <=> max(a,[],1)
ans =
   5   6
>> max(a,[],2)
ans =
   2
   4
   6
>> max(max(a))
ans =  6
```

```
>> a=magic(3)
a =
   8   1   6
   3   5   7
   4   9   2
>> sum(a,1)
ans =
   15   15   15
>> sum(a,2)
ans =
   15
   15
   15
>> b=a.*eye(3)
b =
   8   0   0
   0   5   0
   0   0   2
>> b'
ans =
   8   0   0
   0   5   0
   0   0   2
>> flipud(b)
ans =
   0   0   2
   0   5   0
   8   0   0
```

Plotting Data

```
>> help plot
>> subplot(1,2,1)
>> axis([0 1 -1 1])
>> imagesc(a)
>> imagesc(a),colormap gray;
```

Control Statements

```
>> for i=1:10,
v(i)=2^i;
end;
```

```
function [y1,y2] = returntwo(x)
y1=x^2;
y2=x^3;
>> [a,b] = returntwo(9)
a =  81
b =  729
```

Vectorization

$\sum_{j=0}^{n} \theta_j x_j = \theta^T x$

6.Logistic Regression

Classification

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

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

Hypothesis Repr esentation

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

Decision Boundary

Suppose predict 'y=0' if $h_\theta(x) \le 0.5$: $\theta^T x \le 0$

Cost Function

cost function : $Cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$ (non-convex)

$$Cost(h_\theta(x), y) = -log(h_\theta(x)) \ (if \ y = 1)$$
$$Cost(h_\theta(x), y) = -log(1 - h_\theta(x)) \ (if \ y = 0)$$
$$Cost(h_\theta(x), y) = -ylog(H_\theta(x)) - (1 - y)log(1 - h_\theta(x))$$

Simplified Cost Function

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

To fit parametersθ :

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

Advanced Optimization

Introduction of other optimization method.

Multi-class Classfication:One-vs-All

Easy to understand.

7.Regularization

The problem of overfitting

overfitting: high variance

Options:

1. Reduce number of features
    1. Manually select which features to keep
    2. Model selection algorithm
2. Regularization
    1. Keep all the features,but reduce magnitude/values of parameters θ_j
    2. Works well when we have a lot of features, each of which contributes a bit to predicting y.

Cost Function

Aim - Small values for parameters :

1. simpler hypothesis
2. less prone to overfitting

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

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

Regularized linear r egression

1. Gradient descent

$$\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha\ \frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha\ \left[\left(\frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right], \quad j \in \{1, 2...n\} \\
&\}
\end{aligned}$$

也可以移项得到更新表达式的另一种表示形式

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

2. Normal equation

$$\begin{aligned}
&\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y \\
&\text{where }\ L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}
\end{aligned}$$

Regularized logistic r egression

Cost function:

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

Gradient descent:

$$
\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \, \frac{1}{m} \, \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha \, \left[\left(\frac{1}{m} \, \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right], \quad j \in \{1, 2...n\} \\
&\}
\end{aligned}
$$

8.Neural Networks:Representation

Non-linear hypotheses

Neurons and the brain

Model Representation

Nothing worths recording.

9.Neural Networks:Learning

Cost Function

- preview:Cost function of Logistic Regression:

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

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

- **Cost Function of Neural Network:**

  $h_\Theta(x) \in \Re^K \ (h_\Theta(x))_i = i^{th} output$

  $L =$ total no. of layers in network

  $s_l =$ no. of units in layer

$$J(\Theta) = -\frac{1}{m} [\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - (h_\Theta(x^{(i)}))_k)] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Backpropagation algorithm

- **Forward propagation** : a_j^l : the activation of node j in layer l

$a^{(1)} = x$

$z^{(i)} = \Theta^{(i-1)} a^{(i-1)}$

$a^{(i)} = g(z^{(i)})\ (add\ a_0^{(i)})$

- **Back Propagation** : δ_j^l : the 'error' of node j in layer l

  $\delta^{(L)} = a^{(L)} - y$

  $\delta(i) = (\Theta^{(i)})^T \delta^{i+1} . * g'(z^{(i)})$

  $g'(z^{(i)}) = a^{(i)} . * (1 - a^{(i)})$

  Formally.

  $cost(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$

  $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$

- **Backpropagation algorithm**

  Aim: to $\min_\Theta J(\Theta)$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\lambda = 0)$$

> Traing set {$(x^{(1)}, y^{(1)})$,$(x^{(2)}, y^{(2)})$...$(x^{(m)}, y^{(m)})$}
>
> Set $\Delta_{ij}^{(l)} = 0$
>
> For i=1 to m:
>
> Forward_propagation(x, Θ) #compute $a^{(l)}$
>
> Back_Propagation(a, y, Θ) #compute $\delta^{(l)}$
>
> $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
>
> if j == 0:
>
> $\Delta_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$
>
> if j != 0:
>
> $\Delta_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$
>
> $$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \Delta_{ij}^{(l)}$$

Backpropagation Intuition

$\delta(i) = (\Theta^{(i)})^T \delta^{i+1}$ #I don't know which is correct...

Implementation Note:Unr olling Parameters

Gradient Checking

Random Initialzation

Symmetry breaking:

> Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
>
> (i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

Putting it together

- Pick a network architecture

  1. reasonable default: 1 hidden layer
  2. if hidden layer>1, have smae no. of hidden units in every layer
  3. usually the more the better

- Training a neual network

  1. Randomly initialize weights
  2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
  3. Implement code to compute cost function $J(\Theta)$
  4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$
  5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$. Then disable gradient checking code.
  6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

10.Advice for Applying Machine Learning

Deciding What to T ry Next

- Next:

1. Get more training examples

2. Try smaller sets of eatures
3. Try getting additional features
4. Try adding polynomial features
5. Try decreasing λ
6. Try increasing λ

- Machine learning diagnostic

  A test that you can run to gain insight what is or not working with a learning algorithm, and gain guidance as to how best to imporove its performance.

Evaluating a Hypothesis

```
if((h(x)>=0.5)&&(y==0) || (h(x)<=0.5)&&(y==1))
        err(h(x),y) = 1
else
        err(h(x),y) = 0
```

$$Test\ error = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h(x_{test}^{(i)}), y^{(i)})$$

Model Selection and T raining/validation/test sets

60% : 20% : 20%

Diagnosing Bias vs. V ariance

High bias is underfitting and high variance is overfitting. Ideally, we need to find a golden mean between these two.

High bias (underfitting): both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. Also,$J_{CV}(\Theta) \approx J_{train}(\Theta)$.

High variance (overfitting):$J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

Regularization and Bias/V ariance

choose the model and the regularization term λ :

1. Create a list of lambdas (i.e. λ ∊ {0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24});
2. Create a set of models with different degrees or any other variants.

3. Iterate through the λs and for each λ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the$J_{CV}(\Theta)$ **without** regularization or λ = 0.
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

Learning Curves

- **Experiencing high bias:**

  Low training set size: causes $J_{train}(\Theta)$ to be low and $J_{CV}(\Theta)$ to be high.

  Large training set size: causes both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ to be high with $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

  If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.

- **Experiencing high variance:**

  Low training set size: $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be high.

  Large training set size: $J_{train}(\Theta)$ increases with training set size and $J_{CV}(\Theta)$ continues to decrease without leveling off. Also, $J_{train}(\Theta) < J_{CV}(\Theta)$ but the difference between them remains significant.

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

Deciding What to Do Ne xt

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 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.

11.Machine Learning System Design

Prioritizing What to W ork On

how to improve the accuracy of this classifier:

- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

Error Analysis

The recommended approach to solving machine learning problems is to:

- Start with a simple algorithm, implement it quickly, and test it early on your cross validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.

Error Metrics for Sk ewed Classes

- Precision/Recall
  - $Precision = \frac{True\ positive}{True\ positive + False\ positive}$
  - $Recall = \frac{True\ positive}{True\ positive + False\ negative}$

Trading Off Pr ecision and Recall

1. if we want to predict y=1 only if very confident:

   Higher precision, lower recall

2. if we want to avoid missing too many cases of y=1

Lower precision, Higher recall

$F_1 Score = 2\frac{Precision * Recall}{Precision + Recall}$

Data For Machine Learning

12.Support Vector Machines

Optimization Objective

- Logistics Regression

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

  $cost_1(\theta^T x^{(i)}) = -\log h_\theta(x^{(i)})$

  $cost_0(\theta^T x^{(i)}) = -\log(1 - h_\theta(x^{(i)}))$

- Support Vector Machine

  1. **Cost Function**

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

  $cost_1(\theta^T x^{(i)}) = -\log h_\theta(x^{(i)})$

  $cost_0(\theta^T x^{(i)}) = -\log(1 - h_\theta(x^{(i)}))$

  $C = \frac{1}{\lambda}$

  Large C: Lower bias, high variance

  Small C: Higher bias, low variance

  2. **Hypothesis**

  $if\ \theta^T x \geq 0 : h_\theta(x) = 1,\ otherwise : h_\theta(x) = 0$

Large Margin Intution

if $y = 1$, we want $\theta^T x \geq 1$

if $y = 0$, we want $\theta^T x \leq 1$

Mathematics Behind Lar ge Margin Classification

- Vector Inner Product

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

  $||u|| = \sqrt{u_1^2 + u_2^2}$

  projection: $p = \frac{u^T v}{||u||}$

- SVM Decision Boundary

  When C is very large, the cost function is $\min_\theta [C * 0 + \frac{1}{2} \sum_{j=1}^n \theta_j^2]$

  simplication: C is very large, $\theta_0 = 0,\ n = 2$

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

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

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

  $\Longleftrightarrow p^{(i)} \geq 1\ \ if\ y^{(i)} = 1$

  $p^{(i)} \leq 1\ \ if\ y^{(i)} = 0$

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

Kernels

- Gaussian kernel function

  $k(x, l^{(i)}) = similarity(x, l^{(i)}) = exp(-\frac{||x - l^{(i)}||^2}{2\sigma^2})$

  1. if $x \approx l^{(i)}$:

     $k(x, l^{(i)}) \approx 1$

  2. if x is far from l:

     $k(x, l^{(i)}) \approx 0$

  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.

- SVM with Kernels

  Given $(x^{(2)}, y^2), (x^{(2)}, y^2) \ldots, (x^{(m)}, y^m)$

  choose landmarks : $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)} \ldots, l^{(m)} = x^{(m)}$

  Given example x,compute features f:

$$f = \begin{bmatrix} f_0 = 1 \\ f_1 = similarity(x, l^{(1)}) \\ f_2 = similarity(x, l^{(2)}) \\ \dots \\ f_m = similarity(x, l^{(m)}) \end{bmatrix}$$

- Hypothesis:

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

- Training:

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

- Using an SVM

  package:`liblinear` `libsvm`

  Step:

  1. choose parameter C
  2. choose kernel
  3. choose σ^2

13.Unsupervied Learning

K-Means Algorithm

Step1. cluster assignment

Step2. move centroid

```
Randomly initialize K cluster centroids u1,u2,...uK
Repeat{
    #step 1
    for i = 1 to m:
        c := index(from 1 to K) of cluster centroid closest to x
        # c = min(x-uk)^2
    #step 2
    for k = 1 to K:
        uk := average(mean) of points assigned to cluster k
}
```

Optimization Objective

$c^{(i)}$: index of cluster(1,2,...,K) to which example $x^{(i)}$ is currently assigned

μ_k : cluster centroid k($\mu_k \in R^n$)

$\mu_{c^{(i)}}$: cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Optimization objective:

distortion cost function(失真代价函数):

$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)=\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}-\mu_{c^{(i)}}||^2$$

$$\min_{c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K} J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)$$

Random Initialization

Randomly pick K training examples: Set $\mu_1,\dots,\mu_K$ equal to these K examples

```
for i =1 to 100{ # to avoid local optima
    Randomly initialize K-means
    Run K-means.
    Compute distortion cost funciton J()
}
```

Choosing th Number of CLusters

- Choosing the value of K:

  Anyway...

14.Dimensionality Reduction

Motivation I: Data Compr ession

Motivation II: Data Visualization

Principal Component Analysis Pr oblem Formulation

Reduce from n-demension to k-dimension:

 Find k vectors onto which to project the data, so as to minimize the projection error.

PS.PCA is not linear regression

Principal Component Analysis Algorithm

- Data preprocessing
  1. mean normalization:

     $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

     Replace each $s_j^{(i)}$ with $x_j - \mu_j$
  2. scale features:

     $x_j = \frac{x_j = \mu_j}{s_j}$
- PCA algorithm
  1. Compute covariance matrix:

     $\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$
  2. Compute eigenvectors of matrix Σ:

     $[U, S, V] = svd(\Sigma)$ or use $eig(\Sigma)$ # Singular value decomposition

     where $U = [u^{(1)}, u^{(2)}, \ldots, u^{(n)}] \in R^{n*n}$

     so $U_{reduce} = [u^{(1)}, u^{(2)}, \ldots, u^{(k)}] \in R^{n*k}$

     $Z = U_{reduce}^T X \in R^k$

Choosing the Number of Principa Components

Average squared projection error:

$\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2$

Total variation in the data:

$\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2$

choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2} \le 1\%$$

(It means 99% of variance is retained)

For given k:

$[U, S, V] = svd(\Sigma)$ where $S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & \dots & \\ & & & S_{nn} \end{bmatrix}$

$$\frac{\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - x_{approx}^{(i)}||^2} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$$

So pick smallest value of k for which $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 99\%$

Reconstruction from Compressed Representation

$Z \in R^k$ $U_{reduce} \in R^{n*k}$

$X_{approx} = U_{reduce} Z \in R^n$

Advice for Applying PCA

Before implementing PCA, first try running whatever you want to do with the original data $x^{(i)}$.

Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

15.Anomaly Detection

Problem Motivation

Example:

1. Fraud detection
2. Manufacturing
3. Monitoring computers in a data center

Gaussian Distribution

$$p(x; \mu; \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

- Parameter estimation

  mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$

variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$

standard deviation: σ

Algorithm

Given new example x,compute p(x):

$p(x) = \Pi_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

Anomaly if $p(x) < \epsilon$

Developing and Evaluating an anomaly detection system

1. Fit model p(x) on training set
2. On a cross validation/test example x,predict
$$f(x) = \begin{cases} 1 & if\ p(x) < \epsilon\ (anomaly) \\ 0 & if\ p(x) \geq \epsilon\ (normal) \end{cases}$$
3. Possible exvaluation metrics:
    - True positive,false positive,false negative,true negative
    - Precision/Recall
    - F1-score
4. use creoss validation set to choose parameter ϵ

Anomaly Detection vs. Supervised Learning

Anomaly Detecton	**Supervised Learning**
Very small number of positive examples.	Large number of positive and negative examples.
Hard for any algorithm to learn from positive examples what the anomalies look like.	Enough positive examples for algorithm to get a sense of what positive examples are like.
future anomalies may look nothing like any of the anomalous examples we've seen so far.	Future positive examples likely to be similar to ones in training set.

Choosing What Featur es to Use

Choose features that might take on unususlly large or small values in the event of an anomaly.

Multivariat Gaussian Distribution

Don't model $p(x_1), p(x_2)..., tec.$ separately.

Model p(x) all in one go.

Parameters:$\mu \in R^n, \Sigma \in R^{n*n}$

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$$

Anomaly Detection using the Multivariate Gaussian Distribution

1. Fit model p(x) by setting:

   mean: $\mu = \frac{1}{m}\sum_{i=1}^m x^{(i)}$

   variance: $\Sigma = \frac{1}{m}\sum_{i=1}^m (x^{(i)}-\mu)(x^{(i)}-\mu)^T$

2. Given a new example x, compute:

   $p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$

   Flag an anomaly if $p(x) < \epsilon$

Original model	**Multivariate Gaussian**
$p(x_1;\mu_1,\sigma_1^2)\times...\times p(x_n;\mu_n,\sigma_n^2)$	$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$
Mannually create features to capture anomalies where x1,x2 take unusual combinations of values.	Automatically captures correlations between features.
Computationally cheaper	computationally more expensiv
OK even if m is small.(m:training set size)	Must have m>n, or esle Σ is non-invertible

16.Recommender Systems

Problem formulaiton

n_u: no.users

n_m: no.movies

$r(i,j)$: 1 if user j has rated movie i

$y^{(i,j)}$: rating given by user j to movie i

Content Based Recommendations

$\theta^{(j)}$: parameter vector for user j

$x^{(i)}$: feature vector for movie i

$m^{(j)}$: no. of movies rated by user j

For user j, movie i, predicted rating : $(\theta^{(j)})^T(x^{(i)})$

To learn $\theta^{(j)}$(parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

Optimization algorithm:

$$J(\theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (x_k^{(j)})^2$$

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$$

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}) x_k^{(i)} \ (for\ k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha (\sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)}) \ (for\ k \neq 0)$$

Collaborative filtering

Optimization algorithm:

Given $\theta^{(1)},\dots,\theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(x_k^{(j)})^2$$

Given $\theta^{(1)},\dots,\theta^{(n_u)}$, to learn $x^{(1)},\dots,x^{(n_m)}$:

$$J(\theta^{(1)},\dots,\theta^{(n_u)}) = \frac{1}{2}\sum_{j=1}^{n_m}\sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(x_k^{(j)})^2$$

Collaborative filtering algorithm

Given $x^{(1)},\dots,x^{(n_m)}$ and movie ratings, can estimate $\theta^{(1)},\dots,\theta^{(n_u)}$:

$$min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Given $\theta^{(1)},\dots,\theta^{(n_u)}$, can estimate $x^{(1)},\dots,x^{(n_m)}$:

$$\min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_m}\sum_{k=1}^{n}(x_k^{(j)})^2$$

Minimizing $x^{(1)},\dots,x^{(n_m)}$ and $\theta^{(1)},\dots,\theta^{(n_u)}$ simultaneously:

$$J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)}) =$$
$$\frac{1}{2}\sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_m}\sum_{k=1}^{n}(x_k^{(j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$$\min_{x,\theta} J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)})$$

Step:

1. Initialize $x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)}$ to small random values
2. Minimize $J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)})$ using guadient descent
3. For a user with parameters θ and a movie with features x, predict a star rating of $\theta^T x$

Vectorization: Low Rank Matrix Factorization

How to find movies j related to movie i ?

$small\ ||x^{(i)} - x^{(j)}||$

Implementational Detial: Mean Normalization

recommend a new user the popular(high x) movies.

17.Large Scale Machine Learning

Learning with Large Datasets

Stochastic Gradient Descent

- Batch gradient descent

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

  Repeat{

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

  }

- Stochastic gradient descent

  $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

  $J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$

  1. Randomly shaffle dataset
  2. Repeat{

     for i in range(1,m):

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

     }

Mini-Batch Gradient Descent

Batch gradient descent: Use m example in each iteration

Stochastic Gradient Descent: Use 1 example in each iteration

Mini-Batch Gradient Descent: Use b example in each iteration

b = 2~100

Stochastic Gradient Descent Convergence

Online Learning

Map-reduce and data par alelism

Combine:

$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{n} (temp_j^{(i)})$

Many learning algorithms can be expressed as computing sums of functions over the training set.

18.Application Example: Photo OCR

Problem Description and Pipeline

1. Text detection
2. Character segmentation
3. Character classification

Sliding Windo ws

Getting Lots of Data and Artificial Data Synthesis

Ceiling Analysis: What Part of the Pipeline to W ork on Next