Coursera, Stanford

Machine Learning

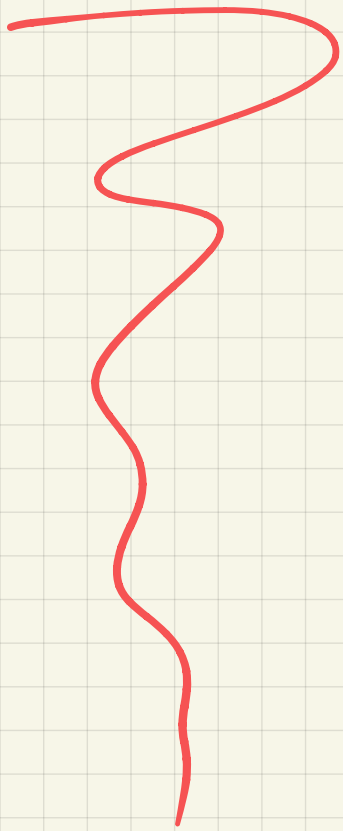
Andrew N.G.

Course Summary: Main Topics

- Supervised Learning
  - Linear Regression
  - Logistic Regression
  - Neural Networks
  - Support Vector Machines (SVM)
- Unsupervised Learning
  - K-means
  - PCA
  - Anomaly Detection
- Special Applications / special topics
  - Recommender Systems
  - large Scale machine learning
- Advice on building a ML System
  - Bias / Variance
  - Regularization
  - Deciding what to work on next:
    - Evaluation of Learning Algorithms
    - Learning curves
    - Error Analysis
    - Ceiling Analysis

Disclaimer: ALL the notes written bellow were Strongly based on the course material.

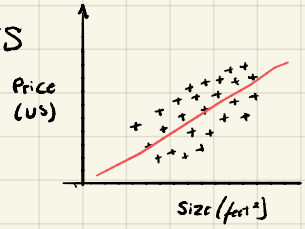
Airton Tiago — 22-04-2020



Week 1 - Linear Regression with one Variable

Model Representation

- Housing Prices



Supervised Learning → Given the "right answer" for each example in the data

Regression Problem → Predict real-valued output

Classification Problem → Predict discrete-valued output

↳ Training Set of housing prices

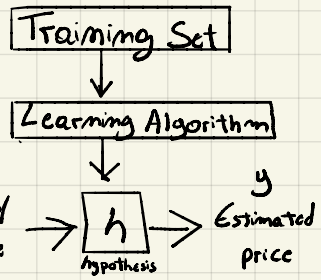
Notation

m = Number of training examples

x's = "Input" variable / features

y's = "Output" variable / "Target" variable

$x^{(1)}$ = 1st line of data set (training)



h maps from x's to y's

- How do we Represent h? $h_\theta(x) = \theta_0 + \theta_1 x$ →

→ Linear Regression with one variable

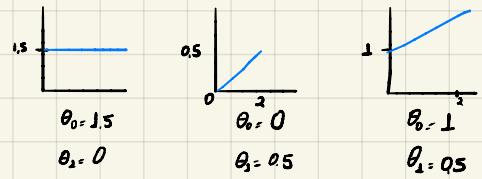
↳ Univariate Linear Regression

↳ fancy way to say "one variable"

Cost function

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

θ_i's = Parameters → How to choose them?



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

So, we need to solve a minimization problem: $\underset{\theta_0, \theta_1}{\text{minimize}} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ (m = size of training)

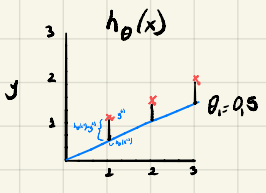
$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$

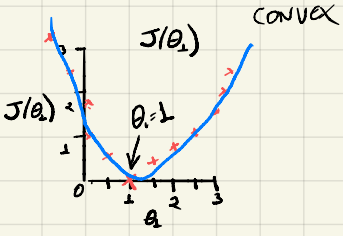
$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ → $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

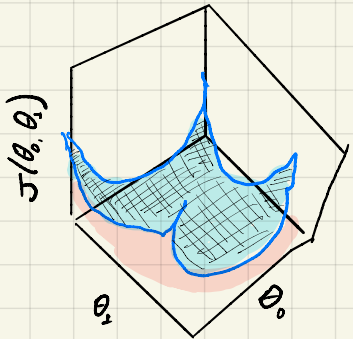
$J(\theta_0, \theta_1)$ ↳ cost function

↳ Squared error function

Cost function Intuition







for fixed θ_1, this is a function of x

$J(0.5) = \frac{1}{2m}\left[(0.5-1.0)^2 + (1-2)^2 + (1.5-3)^2\right]$

$= \frac{1}{2\cdot 3}\left[3.5\right] \cong 0.58$

function of the parameter θ_1

Gradient Descent

We will use this Gradient to descend to lowest point in the cost function
When you have two or more derivatives of the same function, they are called Gradient.

Repeat until convergence $\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j=0 \text{ and } j=1) \}$ → Partial derivative (tangential line to a function) → determine direction

The slope of the tangent is the derivative at that point and it will give us a direction to move towards.

↳ Learning Rate = controls how big is the step

↳ Correct: Simultaneous update

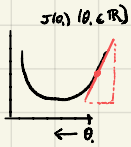
$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

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

$\rightarrow \quad \theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

$:=$ Assignment



$\theta_1 = \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$ → Positive ≥ 0

$\theta_1 = \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$ → Negative ≤ 0

Learning Rate α

- If α is Too small, gradient descent can be SLOW 
- If α is Too Large, gradient descent can overshoot the minimum. It may fail to converge or even diverge  zig-zag
- If θ_1 is initialized in the local minimum the slope is equal to zero → $\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)} = 0 \rightarrow \theta_1 := \theta_1$

↳ Gradient descent can converge To a local minimum, even with the learning rate α fixed

↳ As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over Time.

Gradient descent Algorithm

repeat until convergence $\left\{ \begin{array}{l} \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \\ \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \end{array} \right\}$ Update θ_0 and θ_1 SIMULTANEOUSLY

↳ Convex function = Bowl-shaped

↳ There is no Local minima, just global optimum → for Linear Regression, in specific choice of cost function $J(\theta_0, \theta_1)$.

Sometimes is called by: "Batch" Gradient Descent

↳ "Batch" = each step of gradient descent uses all the training examples

↳ All the time we're looking to this: $\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$

↳ sum all over for each batch

- Gradient descent performs better than normal equation when using LARGE datasets.

While gradient descent can be susceptible to local minima in general, the optimization problem we've proposed above for linear regression has only one global, and no other local, optima: thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function.

The ellipses shown in the Left are the countours of a quadratic function. Also shown is the trajectory taken by gradient descent, which was initialized at (18,30). The x's in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through as it converged to its minimum.

Linear Algebra Review

- Matrices and Vectors

Definitions:

- Matrix: Rectangular array of numbers
- Dimension: Number of rows x Number of columns
- Matrix Elements: Entries of matrix.
  - A_{ij} = "i, y entry" in the i^{th} row, j^{th} column
- Vector: An $n \times 1$ matrix $\rightarrow y = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

- Matrix Multiplication

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \rightarrow A_{2\times 2}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow B_{2\times 3}$$

$$A_{2\times 2} \times B_{2\times 3} = C_{2\times 3}$$

(has to be the same; dimension of result)

$$\rightarrow \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} (2\times 1 + 1\times 4) & (2\times 2 + 1\times 5) & (2\times 3 + 1\times 6) \\ (1\times 1 + 3\times 4) & (1\times 2 + 3\times 5) & (1\times 3 + 3\times 6) \end{bmatrix} = \begin{bmatrix} 6 & 9 & 12 \\ 13 & 17 & 21 \end{bmatrix}$$

- Properties

→ $A \times B \neq B \times A$ (Not commutative)

→ Identity Matrix: denoted I (or $I_{n\times n}$)

↳ Examples: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ → for any matrix A, $A \cdot I = I \cdot A = A$

- Inverse and Transpose

- Not all numbers have an inverse

Matrix Inverse

↳ If A is an $m \times m$ matrix, and if it has an inverse, $A(A^{-1}) = A^{-1}A = I$

Matrices that don't have an inverse are "singular" or "degenerate"

- Matrix Transpose

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix} \qquad B = A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$$

Let A be an $m \times n$ matrix, and let $B = A^T$

then B is an $n \times m$ matrix, and $B_{ij} = A_{ji}$

Week 2 - Linear Regression with Multiples Variables

- Multiple features

previously: ~~$h_\theta(x) = \theta_0 + \theta_1 x$~~ → now → $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ → for convenience of notation, define $x_0 = 1$ ($x_0^{(i)} = 1$)

Gradient Descent for multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 \overset{+}{x_0} + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \dots, \theta_n$

Cost function: ~~$J(\theta_0, \theta_1, \dots, \theta_n)$~~ $= \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

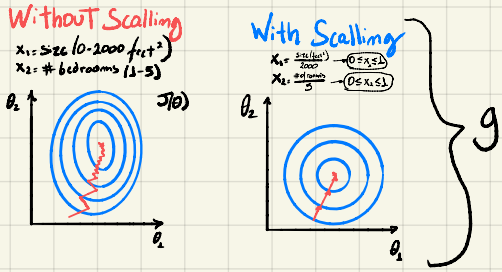
↳ $J(\theta)$

Gradient Descent: Repeat: $\{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\}$ (Simultaneously update for every $j = 0, \dots, n$)

Now we have $n \geq 1$, before it was $n = 1$

Gradient Descent in Practice

Feature Scaling: Make sure features are on similar scale



} Get every feature into approximately a $-1 \leq x_i \leq 1$ range

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

↳ $x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ (μ_1: Average value of x in training set; s_1: range (max-min) or Standard deviation)

$x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$

How to make sure gradient descent is working correctly?

↳ "Debugging":

↳ Example automatic convergence test:

↳ Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration

- $J(\theta)$ should decrease after every iteration

↳ for sufficiently small α, $J(\theta)$ should decrease on every iteration

↳ But if α is too small, gradient descent can be slow to converge.

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

features and polynomial regression

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$, in the house price prediction example x_1, x_2, x_3 would be $\begin{cases} x_1 = (\text{size}) \\ x_2 = (\text{size})^2 \\ x_3 = (\text{size})^3 \end{cases}$

Normal Equation

Method to solve for θ analytically

Singular = Non-invertible matrix

Examples: $m = 4$.

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

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

m x (n+1)

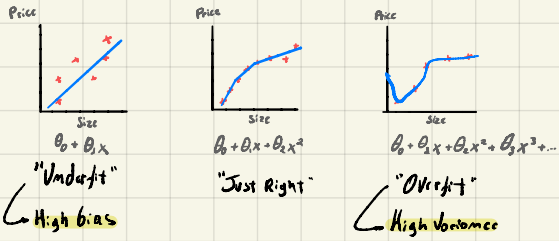
m-dimensional vector

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

Andrew N

The problem of Overfitting

Example: Linear Regression (housing prices)



"Underfit" — High bias

"Just Right"

"Overfit" — High variance

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

↳ How well an hypothesis apply to a new example

Addressing Overfitting

1. Reduce number of features
   - Manually select which features to keep
   - Model selection algorithm
2. Regularization
   - Keep all the features, but reduce magnitude/values of parameters θ_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.
   - $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^i) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$

Regularization term

Regularization parameter

↳ If set to large number, for example 10^{10}. It can lead to "underfit" where $h_\theta(x) = \theta_0$

If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their cost

So we want to make the following function more quadratic:

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

We'll want to eliminate the influence of $\theta_3 x^3$ and $\theta_4 x^4$. Without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our cost function:

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

↳ We have added two extra terms at the end to INFLATE the cost of θ_3 and θ_4.

Week 3 - Logistic Regression

Logistic Regression

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

  where g is the sigmoid function $g(z) = \frac{1}{1+e^{-z}}$

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

- Cost function

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

- Gradient

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

- Regularized Logistic function

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j \quad \text{for } j \geq 1$$

Week 4 - Neural Networks: Representation

- Non-Linear hypothesis
- Neurons and the Brain

  Neural Networks =

  Origins: Algorithms that try to mimic the brain

  Was very widely used in 80s and early 90s; popularity diminished in late 90s.

  Recent resurgence: State-of-the-art technique for many applications

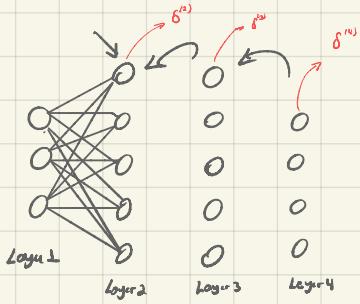
- Model Representation

Week 5 - Neural Networks: Learning

- Cost function
- Backpropagation Algorithm = Algorithm that try to minimize Cost function

  $\delta_j^{(l)}$ = "error" of node j in layer l.

  ↳ we do not compute it for input layer



for each output unit (Layer $L=4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j$

$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$ (element-wise)

$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

"Backpropagation" is neural-network terminology for minimizing our cost function

↳ the goal is to compute:

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

That is we want to minimize our cost function J using an optimal set of parameters in theta.

So, in this part of the course we looked at the equations we are to compute the partial derivative of $J(\Theta)$:

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$$

- **Backpropagation Algorithm**

Training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

for $i = 1$ to m:

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Gradient checking → computed before compute back propagation.

↳ will assure that our back propagation works as intended.

We can approximate the derivative of our cost function with:

$$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta + \varepsilon) - J(\Theta - \varepsilon)}{2\varepsilon}$$

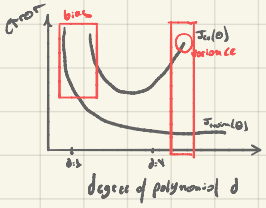
ε should be $\approx 10^{-4}$

Week 6 - Advice for Applying ML

- Machine Learning diagnostic
- Evaluate a hypothesis
- Model selection and train/validation/test sets
- Diagnosing bias vs variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high).

Is it a bias problem or a variance problem?



Bias (underfit):

J_{train} high

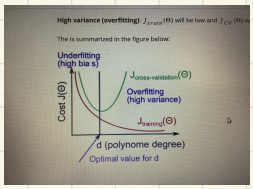
J_{cv} high

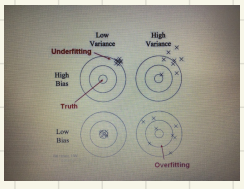
$J_{train} \approx J_{cv}$

Variance (overfit)

$J_{train}(\theta)$ will be low

$J_{cv}(\theta)$ will be >> $J_{train}(\theta)$ ← much greater than





- Regularization and bias/variance

- Large λ = high bias (underfit) → $\lambda = 1000$. $\theta_1 \approx 0$, $\theta_i \approx 0$ → $h_\theta(x) \approx \theta_0$ 
- Intermediate λ =
- Small λ = high variance (overfit) $\lambda \approx 0$ → nothing changes

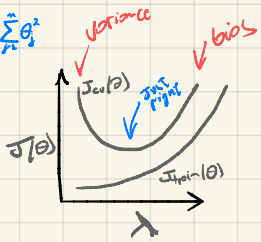
for choose the regularization parameter λ properly you can try a range (0, 0.01, 0.02 ... 10) and store each θ's to see the CV error and choose the best

Bias/variance as a function of the regularization parameter

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

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

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

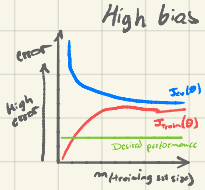


In order to get the "just right" value for λ, we should follow these steps:

1. Create a list of lambdas
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 $\lambda = 0$.
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.

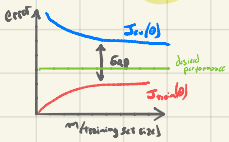
Learning Curves

High bias

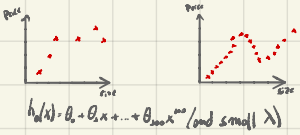


If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

High variance



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



Deciding what to try next (revisited)

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2, etc) → fixes high bias
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Week 6 - ML System Design

- Prioritizing what to work on: Spam classification Example

Error Analysis

→ Manually pick some wrong predictions in the CV data and try to figure out what happened.

Why is the recommended approach to perform error analysis using the cross validation data and to compute $J_{cv}(\theta)$ rather than the test data used to compute $J_{test}(\theta)$?

→ If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so $J_{test}(\theta)$ is no longer a good estimate of how well we generalize to new examples.

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 the errors were made.

Handling Skewed Data

- Error metrics for skewed classes → It occurs when we have a lot more examples in one class than in another

Precision/Recall

Cancer Example:

Predicted class \ Actual class	1	0
1	True Positive	False Positive
0	False Negative	True Negative

Precision: Of all predicted as 1, what are the real 1?

Precision = $\frac{\text{True Positive}}{\text{\# predicted positive}} = \frac{\text{True Positives}}{\text{True Positive + false positives}}$

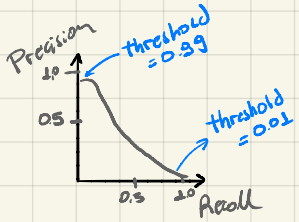
Recall: Of all patients that HAVE cancer, what fraction did we correctly detect as having cancer?

Recall = $\frac{\text{True positives}}{\text{\# actual positives}} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$

Trading Off Precision and Recall

for logistic Regression: if we want to predict y=1 (cancer) only if very confident → **Higher** precision **Lower** recall

if we want to avoid missing too many cases of cancer (Avoid False negatives) → **Higher** recall **Lower** precision



More generally: Predict 1 if $h_\theta(x) \geq$ threshold

- How to compare precision / recall numbers?

f_1 Score (F Score)

$F_1 \text{score} = 2\frac{PR}{P+R}$

Which algorithm is better?

	Precision (P)	Recall (R)	~~Average~~	F_1 Score
Algorithm 1	0.5	0.4	~~0.45~~	0.444
Algorithm 2	0.7	0.1	~~0.4~~	0.175
Algorithm 3	0.02	1.0	~~0.51~~	0.0392

Predict 1 for all the time

A reasonable way to pick the value to use for the threshold in a Logistic Regression Classifier:

Measure precision and recall on the **Cross validation set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

Data for Machine Learning

article [Banko and Brill, 2001] → Classify between confusable words.

"It's not who has the best algorithm that wins. It's who has the most data"

Large data rationale

Use a Learning algorithm with many parameters (e.g. logistic regression / linear regression with many features; Neural network with many hidden units) → Low bias → $J_{train}(\theta)$ will be small

Use a very large training set (unlike to overfit) → Low Variance

→ $J_{train}(\theta) \approx J_{test}(\theta)$

→ $J_{test}(\theta)$ will be small

Week 7 - Support Vector Machines

SVMs are considered by many to be the most powerful 'black box' learning algorithm, and by posing a cleverly-chosen optimization objective, one of the most widely used learning algorithm

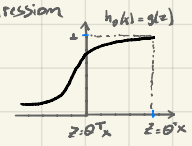
Large Margin Classification

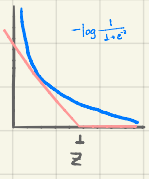
- Optimization Objective
  - Alternative view of logistic regression

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

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

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

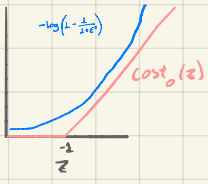




Cost example: $-(y \log h_\theta(x) + (1-y) \log(1-h_\theta(x))$

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

If we want $\theta^T x >> 0$ ($y=1$) — if we want $\theta^T x << 0$ ($y=0$)



- Support Vector Machine

Cost function of logistic Regression

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\underbrace{(-\log h_\theta(x^{(i)}))}_{cost_1(\theta^T x^{(i)})} + (1-y)\underbrace{(-\log(1-h_\theta(x^{(i)})))}_{cost_0(\theta^T x^{(i)})}\right] + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2}_{B}$$

(A)

Cost function of SVM:

$$\min_\theta \cancel{\frac{1}{m}} C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \cancel{\frac{1}{2m}}\sum_{i=0}^{m}\theta_j^2$$

(1/m just a constant)

$A + \lambda B \rightarrow C = \frac{1}{\lambda} \rightarrow CA + B$

(logistic regression → SVM)

} they give the same optimal value of θ when $C = \frac{1}{\lambda}$

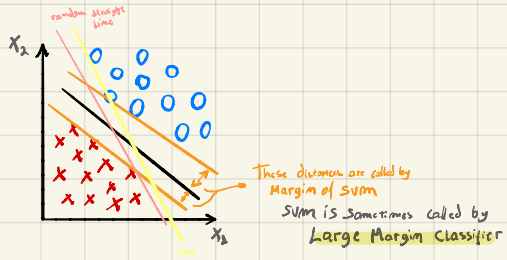
$$\boxed{\min_\theta C \sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{(j=1)}^{m}\theta_j^2}$$

hypothesis: $h_\theta(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$

• Large Margin Intuition

- SVM Decision Boundary: Linearly Separable Case

(It means that exists many different straight lines that can separate the positive and negative examples perfectly)



SVM is sometimes called by Large Margin Classifier

SVM try to separate the classes with the large margin possible.

- The mathematics behind large margin classification (optional)
- Kernels I

↳ Similarity functions

↳ SVM with Kernels

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

↳ Predict "$y=1$" if $\theta^T f \geq 0$ → $\theta_0 f_0 + \theta_1 f_1 + \ldots + \theta_m f_m$

Training:

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

• SVM PARAMETERS

$C (= \frac{1}{\lambda})$ → Large C = Lower bias, high variance → (small λ) → prone to overfitting (if using LR)

Small C = Higher bias, low variance → (Large λ) → prone to underfitting

σ^2 (Gaussian Kernel) → Large σ^2 = features f_i vary more smoothly

Higher bias, lower variance

Small σ^2 = features vary less smoothly

lower bias, higher variance

→ higher slope

Suppose you train an SVM and find it overfits your training data.

↳ The reasonable next step is: Decrease C and Increase σ^2

Using An SVM

Use SVM software packages to solve for parameters θ

→ Need to specify:

→ Choice of parameters C → Informally, the C parameter is a positive value that controls the penalty for misclassified training examples. A large C parameter tells SVM to try to classify all the examples correctly. C plays a role similar to $\frac{1}{\lambda}$, where λ is the regularization parameter that we were using previously for logistic regression.

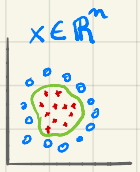
→ Choice of Kernel (similarity function)

- E.g. No Kernel is also called "linear Kernel" → Standard linear classifier → # features n large, # training sample m small
  - Predict "$y=1$" if $\theta^T x \geq 0$ — $x \in \mathbb{R}^{n+1}$
- Gaussian Kernel

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

  Need to choose σ^2 (sigma squared)

  When to use: n is small and/or m is large — n = # features, m = # training examples



To find NON-linear decision boundaries with the SVM, we need to first implement a Gaussian Kernel. You can think of the Gaussian Kernel as a similarity function that measures the "distance" between a pair of examples $(x^{(i)}, x^{(j)})$. The Gaussian Kernel is also parameterized by a bandwidth parameter, σ, which determines how fast the similarity decreases (to 0) as the examples are further apart.

Kernel (similarity) function:

Do Perform feature scaling before using the Gaussian Kernel

Landmarks on training examples as well

$\|x - l\|^2 \longrightarrow v = x - l$

$\|v\|^2 = v_1^2 + v_2^2 + \ldots + v_m^2$

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \ldots + (x_m - l_m)^2$

$(x_1 - l_1)^2$: 1000 feet^2; $(x_2 - l_2)^2$: 1-5 bedrooms

Other choices of Kernel

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

- Many off-the-shelf Kernels Available:
  - Polynomial Kernel → the similarity is $K(x, l) = (x^T l)^2$, another one: $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$ (Parameters; $+0$)

    the general form is: $(x^T l + \text{Constant})^{\text{degree}}$

    Almost Always usually performs worse than Gaussian Kernel

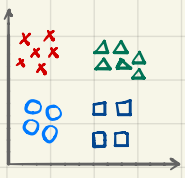
    $x^T l$ Non Negatives
  - More esoteric: String Kernel, Chi-Squared Kernel, histogram intersection Kernel, ...

    String Kernel: if your input data is text strings; histogram intersection Kernel: Similarity between objects

If you are trying to decide among a few different choices of Kernel and are also choosing parameters such as C, σ^2, etc.

You should choose whatever performs best on the cross-validation data

Multi-class classification



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

Many SVM packages already have built-in multi-class classification functionality. Otherwise, use one-vs-all method. (Train K SVMs, one to distinguish $y=i$ from the rest for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ → pick class i with largest $(\theta^{(i)})^T x$.

- Logistic regression vs. SVM's

  n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

  If n is large (relative to m): (E.g. $n \geq m$, $n = 10{,}000$, $m = 10, \ldots, 1000$)

  → Use logistic regression, or SVM without a Kernel ("linear Kernel")

  If n is small, m is intermediate: ($n = 1$-1000, $m = 10$-$10{,}000$)

  → Use SVM with Gaussian Kernel

  If n is small, m is large: ($n = 1$-1000, $m = 50{,}000+$)

  → Create/add more features, then use logistic regression or SVM without a Kernel

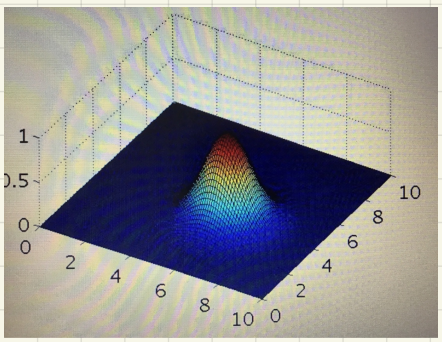
finally, Neural network likely to work well for most of those settings, but may be slower to train

The optimization problem of SVM is convex → So, you don't need to worry about local optima. It always find a global minimum or something close to.

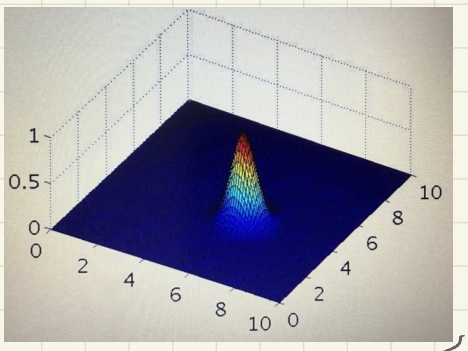
SVM Quiz → Important points

- If you have a decision boundary that is underfit to the training set, we'd like to lower bias / increase the variance of the SVM. We can do so by:
  - either increasing the parameter C or decreasing σ^2

When you are using a Gaussian Kernel, given by $similarity(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$ and you have a plot of $f_i = similarity(x, l^{(i)})$ when $\sigma^2 = 1$:



If you plot f_i with $\sigma^2 = 0.25$ then you have:



It shows a "narrower" Gaussian Kernel centered at the same location which is the effect of decreasing σ^2.

Suppose you have a dataset with $n = 10$ features and $m = 5000$ examples.
After training your logistic regression classifier with gradient descent, you find that it has underfit the training set and does not achieve the desired performance on the training or cross validation sets. For this situation might be promising steps to take as follow:

- → Use a SVM with Gaussian Kernel
  - by using a Gaussian Kernel, your model will have greater complexity and can avoid underfitting the data.
- → Create / add new polynomial features
  - When you add more features, you increase the variance of your model, reducing chances of underfitting

- The maximum value of the Gaussian Kernel (i.e., $sim(x, l^{(i)})$) is 1
  - When $x = l^{(i)}$, the Gaussian Kernel has value $\exp(0) = 1$, and it is less than 1 otherwise
- It is import to perform feature normalization before using the gaussian Kernel.
  - The similarity measure used by the gaussian Kernel expects that the data lie in approximately the SAME range.

Week 8 - Unsupervised Learning

Clustering

- Introduction to unsupervised learning → No labels
- K-Means Algorithm: Is a method to automatically cluster similar data examples together. The intuition behind K-means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.

Input:
- K (number of cluster)
- Training set $\{x^{(1)}, x^{(2)}, \dots x^{(m)}\}$ $\quad x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

Algorithm:
- Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$
- Repeat {
  - cluster assignment step: for $i = 1$ to m
    - $c^{(i)}$:= index (From 1 to K) of cluster centroid closest to $x^{(i)}$ $\quad \min_K \|x^{(i)} - \mu_K\|^2 \rightarrow c^{(i)}$
  - move centroids: for $K = 1$ to K
    - μ_K := Average (mean) of points assigned to cluster K.
  }

$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)} \rightarrow c^{(1)} = 2,\ c^{(5)} = 2,\ c^{(6)} = 2,\ c^{(10)} = 2$

$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$

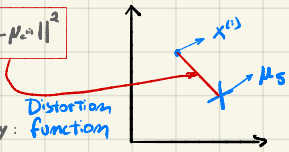
- Optimization Objective
  - $c^{(i)}$ = index of cluster $(1,2,\dots,K)$ to which example $x^{(i)}$ is currently assigned
  - μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

  $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned → $x^{(i)} \to 5$ $c^{(i)} = 5$ → $\mu_{c^{(i)}} = \mu_5$

  Optimization Objective:

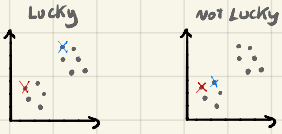
  $J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}-\mu_{c^{(i)}}\|^2$

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

  → This cost function is also called by: first minimizes J in respect to variables $c^{(i)}$ then minimizes J in respect to variables μ_K

- Random Initialization
  - → Should have $K<m$
  - → Randomly pick K training examples
  - → Set $\mu_1,\dots,\mu_K$ equal to these K examples

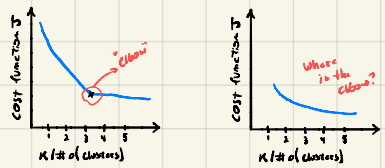
  - Local Optima → You can initialize multiple times in order to assure that you'll get the best sol.

```
For i=1 to 100 {
    Randomly initialize K-means
    Run K-means. Get c^(1),...,c^(m), μ_1,...,μ_K
    Compute cost function (distortion) → J(c^(1),...,c^(m), μ_1,...,μ_K)
}
Pick clustering that gave lowest cost J(c^(1),...,c^(m), μ_1,...,μ_K)
```

→ This is good when you have a small number of clusters K = 2-10

→ But even if have a large number of clusters it should give you a reasonable starting point to start from for finding a good set of clusters.

- Choosing the number of clusters
  - → Elbow Method



→ If you run K-means using K=3 and K=5 and you find that the cost function J is much higher for K=5 than for K=3

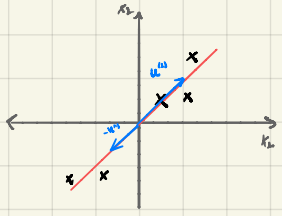
→ In the run with K=5, K-means got stuck in a bad local minimum. You should try re-running K-means with multiple random initializations.

→ Sometimes you are running K-mean to get clusters to use for some later/downstream purpose. Evaluate K-mean based on a metric for how well it performs for that later purpose.

Dimensionality Reduction

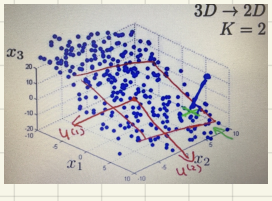
→ In this section we'll introduce PCA and show how it can be used for data compression to speed up learning algorithms as well as for visualizations of complex datasets

- Data Compression
  - Reduce the data from 2D to 1D (e.g. cm/inches) or 3D to 2D, etc
  - It makes some algorithms run much faster
- Data Visualization
- Principal Component Analysis (PCA) - Problem formulation



- Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.
- Reduce from n-dimension to K dimension:

→ Find K vectors $u^{(1)}, u^{(2)}, \dots, u^{(K)}$ onto which to project the data, so as to minimize the projection error.



- PCA is NOT LINEAR regression
  - PCA try to find a lower dimension surface onto which to project the data, so as to minimize this squared projection error.
    - In order to minimize the square distance between each point and the location of where it gets projected.

PCA Algorithm

- Apply data preprocessing

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

Preprocessing (feature scaling / mean normalization):

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

Replace each $x_j^{(i)}$ with $x_j - \mu_j$ → then will have mean = 0

If different features on different scales (e.g., x_1 = size of house, x_2 = # of bedroom), scale features to have comparable range of values.

- Algorithm

Reduce data from n-dimensions to K-dimensions

Compute the "covariance matrix":

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \qquad \text{Sigma}$$

$x^{(i)}$: $n\times 1$, $(x^{(i)})^T$: $1\times n$, Σ: $n\times n$

Compute "eigenvectors" of matrix Σ:

$[U, S, V] = \text{SVD}(\text{sigma})$ → Singular Value Decomposition (SVD)

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & & | \end{bmatrix} \qquad U \in \mathbb{R}^{n\times n}$$

$u^{(1)}, \dots, u^{(K)}$ (K columns)

$$U_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(K)} \\ | & | & & | \end{bmatrix} \longrightarrow z = U_{reduce}^T \cdot x \qquad (K\times n)\cdot(n\times 1) \to K\times 1 \to z \in \mathbb{R}^K$$

- SUMMARY

After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T$ / or the vectorized version sigma = (1/m) * X' * X

```
[U,S,V] = svd(sigma);
Ureduce = U(:,1:K);
z = Ureduce' * x;
```

Reconstruction from compressed Representation

Reconstruction of Original data

$z = U_{reduce}^T \cdot x$ → $z \in \mathbb{R} \to x \in \mathbb{R}^2$

$x_{approx} = U_{reduce} \cdot z$ ($n\times K$)($K\times 1$) → $n\times 1$

Choosing the Number of principal Components

→ Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2$

→ Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically, choose K to be smallest value so that:

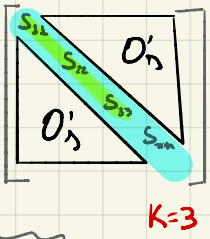
$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \le 0.01 \ (1\%)$$ → It means "99% of variance is retained"

→ Create a for loop for run PCA trying a range of K's and computing $U_{reduce}, z^{(1)}, z^{(2)}, z^{(m)}, x_{approx}^{(1)}, \dots$, and checking if the % chosen of retained variance is satisfied

→ Not efficient, because run PCA over and over

You can do it just iterating using S (diagonal matrix from SVD)

↳ [U, S, V] = svd(Sigma)

$S =$ 

K=3

for a given range of K pick the smallest value of K which:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \leq 0.01 \quad \text{OR} \quad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

Previously, we said that PCA chooses a direction $u^{(1)}$ (or K directions $u^{(1)}, \dots, u^{(k)}$) onto which to project the data so as to minimize the squared projection error. Another way to say the same is that PCA tries to minimize: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

• Advice for Applying PCA

Supervised learning speed up

↳ $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

↳ Extract inputs: Unlabeled dataset: $x^{(1)}, x^{(2)}, \dots, x^{(m)} \in \mathbb{R}^{10000}$

↓ PCA

$z^{(1)}, z^{(2)}, \dots, z^{(m)} \in \mathbb{R}^{1000}$

New training set: $(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \dots, (z^{(m)}, y^{(m)})$

Note: Mapping $x^{(i)} \to z^{(i)}$ should be defined by running PCA ONLY on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

Applications of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithms
  - Choose K by % of variance retain
- Visualization
  - We can't plot over 3 dimensions, so PCA with K=2 or K=3 can help.

Bad use of PCA: To prevent overfitting

↳ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the # of features to K<n, thus, fewer features, less likely to overfit ← BAD. Do not do this

this might work OK, but is not a good way to address overfitting.

↳ Use REGULARIZATION instead

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

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$
- Train logistic regression $\{(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \dots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

→ How about doing the whole thing without using PCA?

↳ Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$

Week 9 - Anomaly Detection

- **Gaussian Distribution or Normal Distribution**

↳ Say $x \in \mathbb{R}$ if x is distributed Gaussian with mean μ, variance σ^2 → $x \sim \mathcal{N}(\mu, \sigma^2)$ ("distributed as")

σ = std deviation
σ^2 = variance

$p(x; \mu, \sigma^2)$

$$p(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \rightarrow \text{Gaussian density}$$

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

- **Anomaly detection: Algorithm**

1- Choose features x_i that you think might be indicative of anomalous examples.

2- fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

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

3- Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

- **Developing and evaluating an anomaly detection system**

- **The importance of real-number evaluation**

When developing a learning algorithm (choosing features, etc), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous and non-anomalous examples ($y=0$ if normal, $y=1$ if anomalous).

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ (assume just normal examples / not anomalous)

Cross Validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set: $\underbrace{(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})}_{y=1}$

- **Aircraft engines example**

→ 10.000 good (normal) engines

→ 20 flawed engines (anomalous)

Training set: 6000 good engines ($y=0$)

CV set: 2000 good engines ($y=0$), 10 anomalous ($y=1$)

Test set: 2000 good engines ($y=0$), 10 anomalous ($y=1$)

- **Algorithm Evaluation**

Fit model $p(x)$ on Training set $\{x^{(1)}, \dots, x^{(m)}\}$

On a cross validation / test example x, predict:

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

Possible evaluation metrics:

→ Precision / Recall

→ F_1 Score

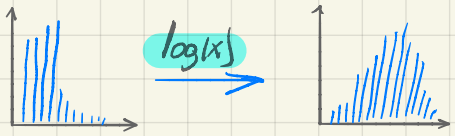
Can also use CV set to choose parameter ε.

- **Anomaly detection vs. Supervised learning**

• Very small # of $y=1$	• Large # of $y=1$, $y=0$
• large # of $y=0$	• Enough positive examples for algorithm to get sense of what positive examples are like, future positive examples likely to be similar to ones in training set.
• Many ≠ types of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like.	
• Future anomalies may look different than the actual.	

• Choosing what features to use

- It's better to have gaussian features
- In order to do it, we can try these:



other options would be:

$x_2 \leftarrow \log(x_2 + 1)$

$x_3 \leftarrow \sqrt{x_3}$

$x_1 \leftarrow \sqrt[3]{x_1}$

- Choose features that might take on usually large or small values in the event of an anomaly.
- Sometimes you can try to divide one feature by another

• Multivariate Gaussian (Normal) distribution.

- $x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \dots$, etc. separately
- Model $p(x)$ all in one go.

Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

• Anomaly detection using the multivariate Gaussian distribution

Parameters: μ $(\mu \in \mathbb{R}^n)$, Σ $(\Sigma \in \mathbb{R}^{n \times n})$

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

Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

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

1- Fit the model $p(x)$ by setting:

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

2- Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

flag an anomaly if $p(x) < \varepsilon$

Relationship to the original model

The multivariate Gaussian model is the same as original model when $\Sigma = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{bmatrix}$

Original model vs Multivariate Gaussian

$p(x_1;\mu_1,\sigma_1^2) \times \dots \times p(x_n;\mu_n,\sigma_n^2)$	$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} \lvert\Sigma\rvert^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$
• Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. $x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$	• Automatically captures correlations between features
• Computationally cheaper (scale better to large n, e.g. $n=10000$, $n=100\,000$)	• Computationally more expensive $\left(\frac{n^2}{2}\right)$
• OK even if m (training set size) is small)	• MUST have $m > n$ or else Σ is non-invertible ↳ $m > 10n$ → $m \geq 10n$

Week 9 - Recommender System

↳ They look at patterns of activities between different users and products to produce recommendations

· Problem formulation

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 (defined only if $r(i,j)=1$)

· Content-based Recommendations

Problem formulation:

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined)

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

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

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

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

Optimization Objective:

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

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

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

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

Gradient descent update:

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

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

· Collaborative filtering

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

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

→ Guess $\theta \to x \to \theta \to x \to \theta \to x \to \dots$

Algorithm:

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 gradient descent (or an advanced optimization algorithm). E.g.:

for every $j = 1, \dots, n_u$, $i = 1, \dots, n_m$:

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

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

3. for a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

· Vectorization: Low rank matrix factorization

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance Forever	5	?	?	0
Cute puppies	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. Karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Predicted ratings

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \cdots & \cdots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \cdots & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & & & & \\ (\theta^{(1)})^T(x^{(n_m)}) & \cdots & \cdots & & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \quad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ -(\theta^{(2)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

$X \cdot \Theta^T$

↳ low rank matrix factorization

- finding related movies
  - → for each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.
    - x_1 = romance, x_2 = action, ...
  - → How to find movies j related to movie i?
    - Small $\|x^{(i)} - x^{(j)}\|$ — movie j and i are "similar"
  - 5 most similar movies to movie i:
    - find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

- Mean Normalization
  - for user j, on movie i, predict: $(\theta^{(j)})^T(x^{(i)}) + \mu_i$ → this is the mean of the ratings for movie i

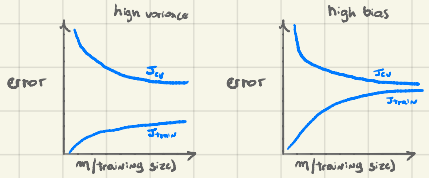
Week 10 - Large Scale Machine Learning

Knowing how to handle big data is one of the most sought after skills in most companies.

- Learning with large datasets

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

What if you have $m = 1.000.000$?



- Stochastic Gradient descent

Batch = entire training set

Batch Gradient Descent

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

Repeat {

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

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$

(for every $j = 0, \dots, n$)

}

Stocastic 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 suffle dataset
2. Repeat {

for $i = 1, \dots, m$ {

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$ → $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

(for $j = 0, \dots, n$) → $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots$

}

}

- Algorithm
  1. Randomly suffle (reorder) training
  2. Repeat { for $i := 1, \dots, m$ { $\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ (for $j = 0, \dots, n$) } } ← 1-10x (usually)

→ Batch Gradient descent

Stocastic Gradient descent — It "fails" to find a global minimum, but it find its zone of global minimum, which still a good option.

- Mini-batch Gradient descent

→ Batch Gradient descent = Use all m examples in each iteration.

→ Stocastic Gradient descent = Use 1 example in each iteration.

→ Mini-batch Gradient descent = Use "b" examples in each iteration.

↳ Usually $b = 10$

- Algorithm: Mini-batch gradient descent

  Say $b=10$, $m=1000$ → It's possible to apply vectorization

  Repeat { for $i=1, 11, 21, 31, \dots, 991$ {

  $$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

  (for every $j=0, \dots, n$) } }

Stocastic Gradient Descent - Convergence

- Checking for convergence:
  - → Batch Gradient Descent:

    Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent

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

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

    During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$

    Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

    Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 examples.

    If the cost is increasing you can try to decrease the learning rate.

    Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge (E.g. $\alpha = \frac{constant1}{iterationnumber + constant2}$)

Online Learning

- Shipping service
- Product Search (Learning to search) → CTR
- Choosing special offers to show user
- Customized selection of news articles
- Product recommendation

Map-reduce and Data parallelism

Map-reduce → Batch Gradient Descent

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Training Set

Instead of Machine 1 it could be cores

- $(x^{(i)}, y^{(i)})$ → Machine 1 → Use $(x^{(1)}, y^{(1)}), \dots, (x^{(100)}, y^{(100)})$ → $temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
- ⋮ → Machine 2 → Use $(x^{(101)}, y^{(101)}), \dots, (x^{(200)}, y^{(200)})$ → $temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
- ⋮ → Machine 3 → Use $(x^{(201)}, y^{(201)}), \dots, (x^{(300)}, y^{(300)})$ → $temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
- $(x^{(m)}, y^{(m)})$ → Machine 4 → Use $(x^{(301)}, y^{(301)}), \dots, (x^{(400)}, y^{(400)})$ → $temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

Combine: $\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$ $(j=0, \dots, n)$

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

E.g. for Advanced optimization, with logistic regression, need:

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

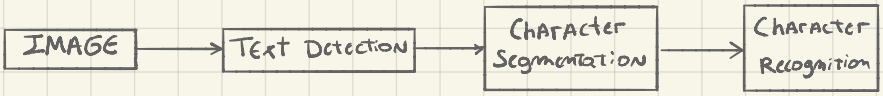
$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Week 11 - Application Example - Photo OCR

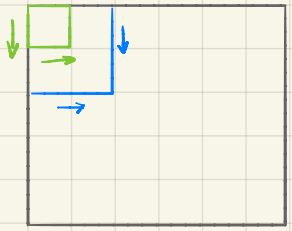
- Problem Description and pipeline:

  OCR = Optical Character Recognition

Pipeline



Sliding Window Detection



Step-size / Stride

It is used to "scan" the image with a fixed window that slides in the image

Getting lots of Data: Artificial Data Synthesis

OCR:

this Synthetic data can be obtained using font styles + Noise / distortions

The synthetic data can be produced for a lot of applications that needs more training examples

↳ E.G. speech recognition (can use the original data with added background noises or bad cell phone connection)

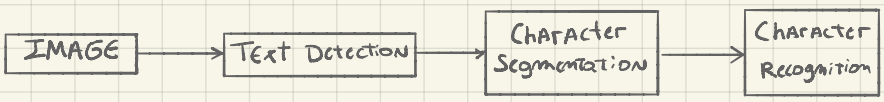
The distortion introduced in your synthetic data should be representation of the type of noise / distortions in the test set.

Usually does not help to add purely random / meaningless noise to your data.

Discussion on Getting more data

1. Make sure you have a **LOW BIAS** classifier before expending the effort. (Plot curves). E.g. Keep increasing the number of features / number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect / label it yourself → # hours?
   - "Crowd source" (E.g. Amazon Mechanical Turk)

Ceiling Analysis: What part of the pipeline to work on Next



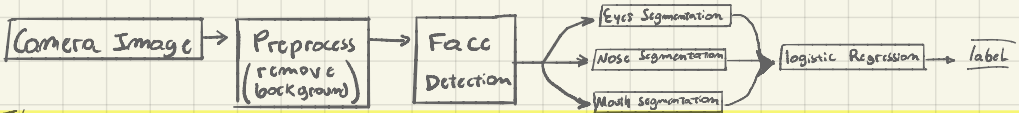
What part of the pipeline should you spend the most time trying to improve?

Truth table →

Component	Accuracy
Overall System	72%
Text Detection	89%
Character Segmentation	90%
Character Recognition	100%

+17 pp, +1 pp, +10 pp

face recognition from Images



If we got 100% of accuracy in each component, one per time, how would this affect the Overall accuracy of the system?