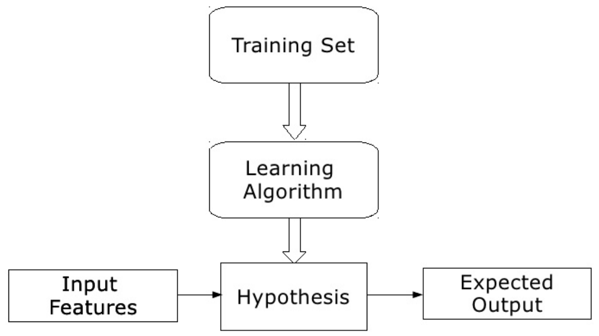
MACHINE LEARNING

By Mohamed Aziz Tousli

About



- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.“ - Tom Mitchell

- Supervised Learning: We have a data set and already know what our correct output should look like → Relationship between the input and the output (**labled data**)
  - Regression problems: Output is **continuous** or **almost continuous**

    → Linear regression
  - Classification problems: Output is **discrete**

    → Logistic regression

- Unsupervised Learning: We have no idea what our results should look like output (**unlabled data**)

Summary: Main topics

Supervised Learning
- Linear regression, logistic regression, neural networks, SVMs

Unsupervised Learning
- K-means, PCA, Anomaly detection

Special applications/special topics
- Recommender systems, large scale machine learning.

Advice on building a machine learning system
- Bias/variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.

Linear Regression with One Variable

- Training example:
  - $x^{(i)}$: input: feature
  - $y^{(i)}$: output: label
- Linear regression with one variable = Unvariante linear regression = 1 Output & 1 Input
- Hypothesis function: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$ #Map input into output #θ: Weight/Parameter
- Cost function = Squared error function = Mean squared error:
  - $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$: #Average/Mean of all inputs #$\frac{1}{2}$ for the derivative of ^{2}
  - J convex quadratic function → One minimum
  - m: number of training examples: training set
- Gradient descent = Least mean squares: $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

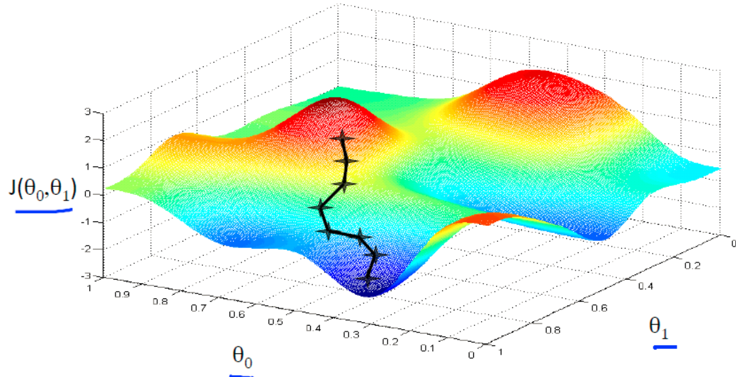
→Minimize J and update θ #α: learning rate

→Small α → Slow convergence

→Big α → Divergence

Linear Regression with Multiple Variables

- $x_j^{(i)}$ #Value of feature *j* in the i^th^ training example
- m #Number of training examples ; n #Number of features
- Hypothesis function: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x \,; x_0 = 1$
- $\rightarrow h_\theta(x) = \boldsymbol{X\theta}; X = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{pmatrix} ; \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$
- Cost function: $J(\theta_0, \theta_1, \ldots) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \boldsymbol{\frac{1}{2m}(X\theta - y)^T(X\theta - y)}$
- Gradient descent: Repeat until convergence:
  - $\theta_j = \theta_j - \alpha \frac{1}{\mathrm{m}} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \,; \boldsymbol{\theta = \theta - \frac{\alpha}{m} X^T(X\theta - y)}$
- **Vectorization**: Matrix operations (vs Loops)



Other Notes

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$, need to calculate $(X^TX)^{-1}$
Works well when n is large	Slow if n is very large

- Purpose of feature normalization: Speed up learning
- Feature scaling: $\frac{input\ value}{maximum\ value - minimum\ value} \rightarrow -1 \leq x_i \leq 1$
- Mean normalization: $input\ value - meanvalues \rightarrow \mu_i' = 0$
- $\Rightarrow x_i = \frac{x_i - \mu_i}{s_i}$ #μ: mean; s: range/standard deviation
- Debug gradient descent: Plot J in function of number of iterations: J must be decreasing
  PS: If J is increasing, decrease α (by multiple of 3)
- Automatic convergence test: Use a threshold ϵ instead of number of iterations
- Polynomial regression: Create other features by squaring actual features
  → Change the behavior of the curve of the hypothesis function
- Normal equation: Find optimum without iteration: $\boldsymbol{\theta} = \left(\boldsymbol{X^T X}\right)^{-1} \boldsymbol{X^T y}$
  If $(X^TX)^{-1}$ is noninvertible → Delete features that are linearly dependent
  PS: No need for featuring scaling here

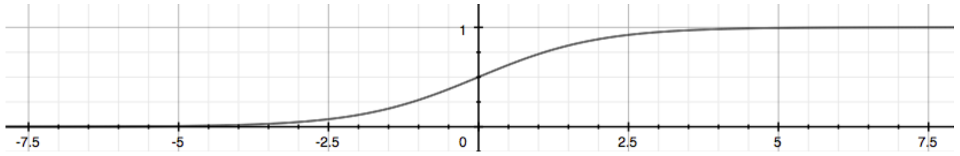
Logistic Regression

- Binary classification problem: Output y ∈ {0 « negative class »,1 « positive class »}

- Method: Use linear regression and predict > 0,5 as 1 and < 0,5 as 0
  → Doesn't work well (Classification is not linear)

- Hypothesis function = Sigmoid function = Logistic function = $h_\theta(Linear\ function)$
  - $\boldsymbol{h_\theta(x)} = \frac{\mathbf{1}}{\mathbf{1+e^{-\theta^T X}}} = P(y = 1/x; \theta) = 1 - P(y = 0/x; \theta) \in [0,1]$

- Decision boundary:
  - $h_\theta(x) \geq 0{,}5 \Leftrightarrow \theta^T x > 0 \rightarrow y = 1$
  - $h_\theta(x) < 0{,}5 \Leftrightarrow \theta^T x < 0 \rightarrow y = 0$

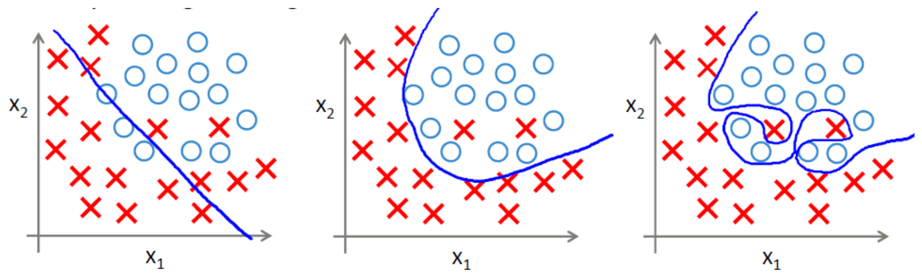


- Cost function: $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) log\left(1 - h_\theta\left(x^{(i)}\right)\right)$
  - $\boldsymbol{h = g(X\theta); J(\theta) = -\frac{1}{m}\left(-y^T \log(h) - \left(1 - y^T\right) \log(1 - h)\right)}$
  - Convex function → Guarantee the existence of the minimum

- Gradient descent: Repeat until convergence:
  - $\theta_j = \theta_j - \alpha \frac{1}{\mathrm{m}} \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$; $\boldsymbol{\theta = \theta - \frac{\alpha}{m} X^T(g(X\theta) - y)}$

- Multiclass classification: One-vs-all: Here, y ∈ {0,1,…,n}
  → Divide problem into n+1 binary classification problems
  → Choose one class and lump all the others into a single class, repeatedly → Choose highest value

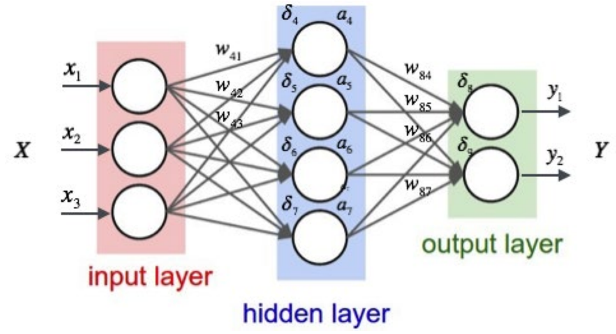
Regularization



- Overfitting (High variance) vs Underfitting (High bias)
  - Overfitting solutions: Reduce features, regularization (reduce parameters)
- New cost function: $J(\theta) = Old\ J(\theta) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j{}^2$
  - λ: Regularization parameter
  - Sum starts from 1 because we don't want to penalize θ_0
- New gradient descent: $\theta_j = Old\ \theta_j - \alpha\frac{\lambda}{m}\theta_j = \boldsymbol{\theta_j(1 - \alpha\frac{\lambda}{m})} - \alpha\frac{1}{\mathrm{m}}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$
  - $1 - \alpha\frac{\lambda}{m}$ is less than 1 → Reduce the value of θ_j by some amount on every update
- Regularization for normal equation: $\theta = (X^TX + \lambda L)^{-1}X^Ty; L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$

L: Number of layers
s_l: Number of units in layer l
K: Number of output units

$a_i^{(j)}$: Activation of unit i in layer j
$\theta^{(j)}$: matrix of weights in layer j (j+1,j)
$z^{(j)} = \theta^{(j-1)} a^{(j-1)}; a^{(j)} = g\left(z^{(j)}\right)$



Neural Networks

- Cost function of output layer: $J(\Theta) = -\frac{1}{\mathrm{m}} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} \log\left(\left(h_\Theta\left(\mathrm{x}^{(i)}\right)\right)_k\right) + \left(1 - y_k^{(i)}\right) \log\left(1 - \left(h_\Theta\left(\mathrm{x}^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{\mathrm{j,i}}^{(\mathrm{l})})^2$
  - Double sum simply adds up the logistic regression costs calculated for each cell in the output layer
  - Triple sum simply adds up the squares of all the individual Θs in the entire network.

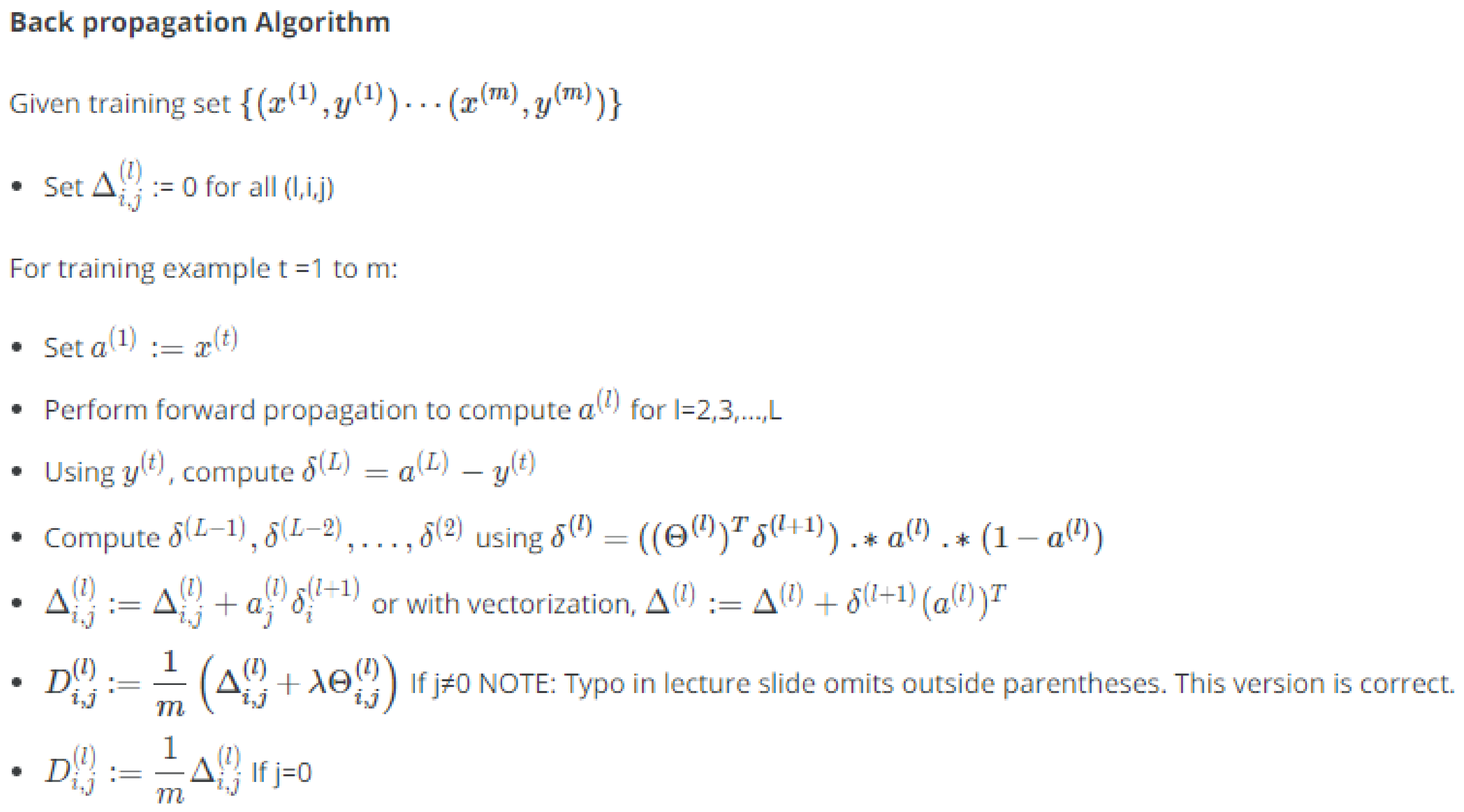
- Backpropagation = Gradient Descent:
  - Last layer: $\delta^L = a^{(L)} - y$ #$\delta_j^{(l)}$: *error of node j in layer l ⬄ derivative of cost function*
  - Other layers: $\delta^{(l)} = \left(\left(\Theta^{(l)}\right)^T \delta^{(l+1)}\right) .* \, a^{(l)} .* \left(1 - a^{(l)}\right)$

    element-wise

- Gradient checking: To check if backpropagation algorithm is correct (very slow, to be used once)
  - $\frac{\partial}{\partial \Theta_\mathrm{j}} J(\Theta) = \frac{J(\Theta_1,\ldots,\Theta_\mathrm{j}+\epsilon,\ldots,\Theta_\mathrm{n}) - J(\Theta_1,\ldots,\Theta_\mathrm{j}-\epsilon,\ldots,\Theta_\mathrm{n})}{2\epsilon}; \epsilon = 10^{-4}$

- Random initialization: $\Theta^{(l)} = 2\, \epsilon\, rand\left(L_{output}, L_{input} + 1\right) - \epsilon$
  - Initialization to zeros is not good in neural networks

Backpropagation Algorithm



Back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

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

For training example t =1 to m:

- Set $a^{(1)} := x^{(t)}$
- Perform forward propagation to compute $a^{(l)}$ for l=2,3,…,L
- Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \;.*\; a^{(l)} \;.*\; (1 - a^{(l)})$
- $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
- $D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$ If j≠0 NOTE: Typo in lecture slide omits outside parentheses. This version is correct.
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ If j=0

Neural Networks (Final)

First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers total.

- Number of input units = dimension of features $x^{(i)}$
- Number of output units = number of classes
- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
- Defaults: 1 hidden layer. If more than 1 hidden layer, then the same number of units in every hidden layer.

Training a Neural Network

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\theta(x^{(i)})$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

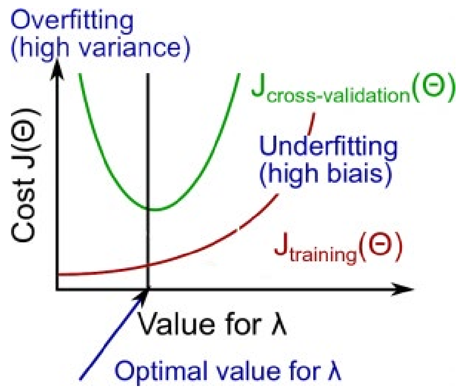
When we perform forward and back propagation, we loop on every training example:

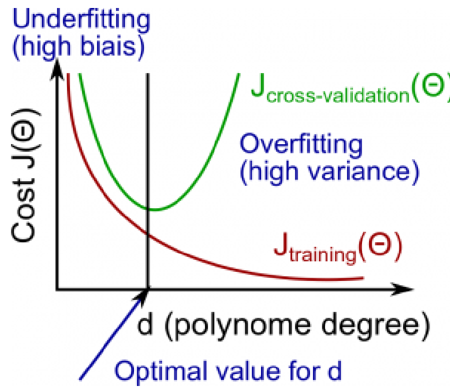
```
for i = 1:m,
   Perform forward propagation and backpropagation using example (x(i),y(i))
   (Get activations a(l) and delta terms d(l) for l = 2,...,L
```

Evaluating a Hypothesis

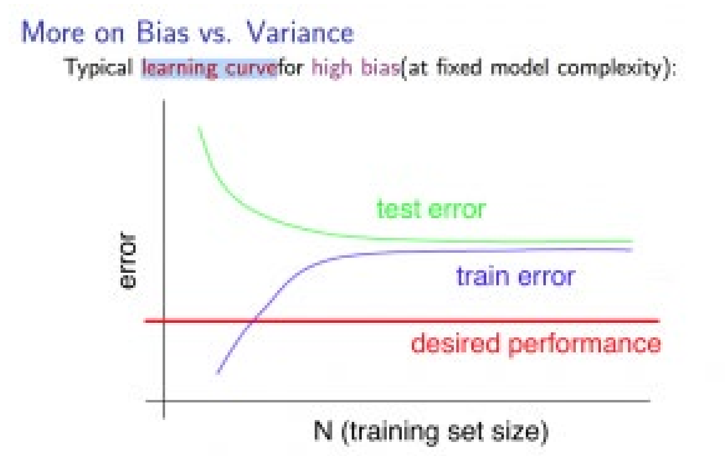
- Train set:
  1. **Train set 60%**: Learn Θ and minimize $J_{train}(\Theta)$ using the training set
  2. **CV set 20%**: Try different hyperparameters using the cross validation set
  3. **Test set 20%**: Compute the test set error $J_{test}(\Theta)$
     - We don't use regularization in the CV set (since λ is already fixed in the train set)

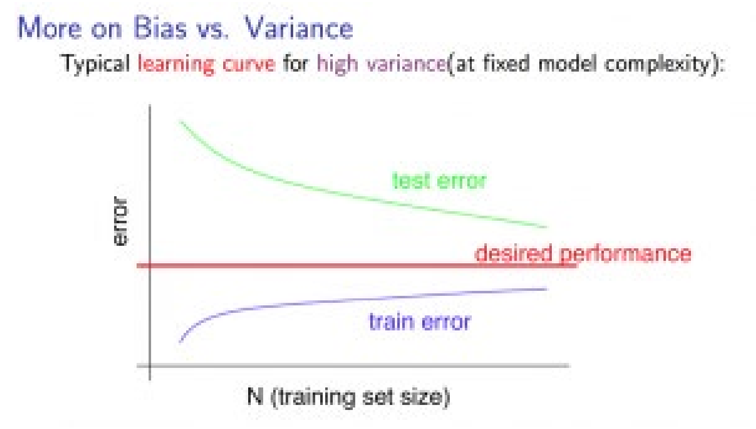
- Test set error:
  - For linear regression: $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\Theta\left(x_{test}^{(i)}\right) - y_{test}^{(i)} \right)^2$
  - For classification/misclassification error: $Test\ Error = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err\left(h_\Theta\left(x_{test}^{(i)}\right), y_{test}^{(i)} \right)$
    - $err(h_\Theta(x), y) = \begin{cases} 1 & if\ h_\Theta(x) \geq 0{,}5\ and\ y = 0\ or\ h_\Theta(x) < 0{,}5\ and\ y = 1 \\ 0 & otherwise \end{cases}$





Bias vs Variance





- High variance solutions:
  - Getting more training examples
  - Trying smaller sets of features
  - Increasing λ
  - Making neural network smaller
- High bias solutions:
  - Adding features
  - Adding polynomial features
  - Decreasing λ
  - Making neural network bigger

Model Selection

- Bias: approximation error (Difference between expected value and optimal value)
  - High Bias = UnderFitting (BU)
  - $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ both will be high and $J_{train}(\Theta) \approx J_{CV}(\Theta)$
- Variance: estimation error due to finite data
  - High Variance = OverFitting (VO)
  - $J_{train}(\Theta)$ is low and $J_{CV}(\Theta) \gg J_{train}(\Theta)$
- Intuition for the bias-variance trade-off:
  - Complex model ➔ sensitive to data ➔ much affected by changes in X ➔ high variance, low bias.
  - Simple model ➔ more rigid ➔ does not change as much with changes in X ➔ low variance, high bias.
- Regularization effects:
  - Small values of λ allow model to become finely tuned to noise leading to large variance ➔ overfitting.
  - Large values of λ pull weight parameters to zero leading to large bias ➔ underfitting.
- Model complexity effects:
  - Lower-order polynomials fit poorly consistently ➔ low variance, high bias.
  - Higher-order polynomials fit the training data well and the test data poorly ➔ high variance, low bias.

Error Metrics for Skewed Classes

- <u>Skewed classes</u>: When we have lot more examples from one class than from the other class.
- <u>Precision/Recall</u>:

Actual class / Predicted class	1	0
1	**True positive**	**False positive**
0	**False negative**	**True negative**

$$Precision = \frac{True\ positives}{Total\ number\ of\ predicted\ positives} = \frac{True\ positives}{True\ positives + False\ positives}$$

$$Recall = \frac{True\ positives}{Total\ number\ of\ actual\ positives} = \frac{True\ positives}{True\ positives + False\ negatives}$$

$$Accuracy = \frac{True\ positives + True\ negatives}{Total\ population}$$

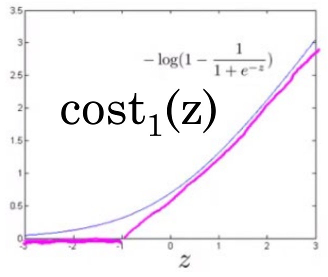
PS: We want both recall and precision to be high

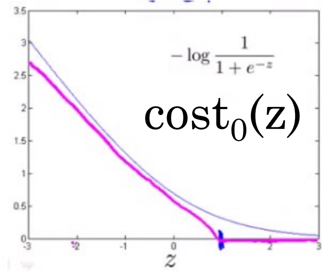
- <u>Precision/Recall tradeoff</u>:
  - Confident prediction ⇔ Large threshold (1 if > 0,7) ➔ High precision, low recall
  - Safe prediction ⇔ Small threshold (1 if > 0,3) ➔ Low precision, high recall

$\rightarrow F\ Score = F1\ Score = 2\frac{PR}{P+R}$

PS: Precision/Recall train on the CV set

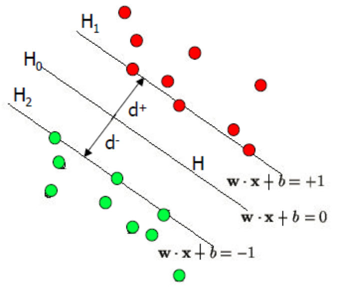
Support Vector Machine (SVM)





- Purpose: Classification machine learning algorithm
- $J(\theta) = C\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{j=1}^{n} \Theta_j^2$
    - $C = \frac{1}{\lambda}$: Convention of SVMs
    - $\frac{1}{m}$ is removed because minimizing f(x) is the same as minimizing αf(x)
    - $h_\theta(x) = \begin{cases} 1 \; if \; \Theta^T x \geq 0 \\ 0 \; otherwise \end{cases}$

      → Discriminant function: Outputs 1 or 0 instead of probability of y = 1 or 0 (in logistic regression)



H_1
H_0
H_2
d^+
d^-
H
$\mathbf{w}\cdot\mathbf{x}+b=+1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$

- Large margin classifier:
    - If y=1, we want $\Theta^T x \geq 1$ (not just ≥ 0)
    - If y=0, we want $\Theta^T x < -1$ (not just < 0)
    - **Margin**: Distance of the decision boundary to the nearest example

  ➔Decision boundary is **as far away as possible** from positive and negative examples (large margin)

  C large ➔ margin large. Reduce C for **outlier** examples

  PS: Data is **linearly separable** when a **straight line** can separate positive and negative examples

Kernels

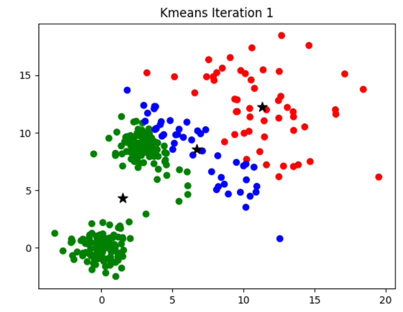
m / n	Small	Intermediate	Large
Small	-	Guassian Kernel	Linear Kernel
Large	Linear Kernel	-	-

➔ Neural Networks works well, but slower

- Purpose: Make complex, non-linear classifiers using SVMs
- Gaussian kernel: $f_i = similarity(x, l^{(i)}) = e^{-\frac{\left\|x-l^{(i)}\right\|^2}{2\sigma^2}}$; l for landmark
  - $l^{(1)} \to f_1$
  - $l^{(2)} \to f_2$ ➔ $h_\Theta(x) = \Theta_1 f_1 + \Theta_2 f_2 + \cdots$
  - …

  σ^2 can be modified to affect the **drop-off** of the feature f_i
  Feature scaling must be performed before using Gaussian kernel
  Similarity functions must satisfy "Mercer's Theorem" (Convergence of SVM)
- Cost function: $J(\theta) = C\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T \boldsymbol{f}^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T \boldsymbol{f}^{(i)}) + \frac{1}{2}\sum_{j=1}^{n} \Theta_j^2$
- About hyper parameters:
  - C large ➔ High variance, low bias
  - C small ➔ Low variance, high bias
  - σ^2 large ➔ Features f_i vary more smoothly ➔ High bias, low variance
  - σ^2 small ➔ Features f_i vary less smoothly ➔ Low bias, high variance
- PS: No kernel «Linear kernel» ➔ Standard linear classifier
- PS: Multiclass classification ➔ One-vs-all method

Clustering



- Purpose: Unsupervised learning
- K-means algorithm: Automatically group data into coherent subsets
  1. **Random initialization** of K cluster centroids
  2. **Cluster assignment**: assign all examples into groups based on which cluster centroid is closest to

     $c^{(i)} = argmin_k \left|\left|x^{(i)} - \mu_k\right|\right|^2$ #Each $c^{(i)}$ contains the index of the centroid that has minimal distance to $x^{(i)}$

     PS: Square root is to minimize more sharply and less computation
  3. **Move centroid**: compute the averages for all the points inside each of the groups, then move the cluster centroid points to those averages

     $\mu_k = \frac{1}{n}\left(x^{(k_1)} + \cdots + x^{(k_n)}\right)$ where each of $x^{(k_1)}, \ldots, x^{(k_n)}$ are the training examples assigned to group mμ_k

     PS: if μ_k has 0 points assigned to it → Re-initialize or eliminate
  4. Re-run (2) and (3) until we have found our clusters

     Stop when new iterations do not affect the clusters

$c^{(i)}$=index of cluster (1,2,...,K) to which example $x^{(i)}$ is currently assigned
μ_k=cluster centroid k
$\mu_{c^{(i)}}$=cluster centroid of cluster to which example $x^{(i)}$ has been assigned

- Cost function = Distortion of training examples: $J\left(c^{(i)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k\right) = \frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)} - \mu_{c^{(i)}}\right|\right|^2$

  Cluster assignment: Minimize J with $c^{(1)},\ldots,c^{(m)}$ (holding $\mu_1,\ldots,\mu_k$ fixed)
  Move centroid: Minimize J with $\mu_1,\ldots,\mu_k$
  PS: J always decrease unless stuck at a bad local optimum

- Random initialization: Randomly pick K training examples and set $\mu_1,\ldots,\mu_k$ equal to them

  PS: K-means **can get stuck in local optima** ➔ Run the algorithm on different random initializations

- Choosing the Number of Clusters:
  - **The elbow method**: Plot J(K). Choose K at the point where J starts to flatten out (although the curve is very gradual)
  - **Downstream purpose**: Choose K that proves to be the most useful for some goal you're trying to achieve

Principal Component Analysis (PCA)

- Purpose: Dimensionality reduction → Speed up supervised learning algorithm

  Data Compression: Reduce the dimension of our features if we have a lot of redundant data

  →Make a new single line of two highly correlated features

  →Reduce computer memory and speed up learning algorithm.

  PS: We are reducing our features rather than our number of examples (m fixed, n decreases)

  Visualization: Reduce the dimension to 3 or less in order to plot it

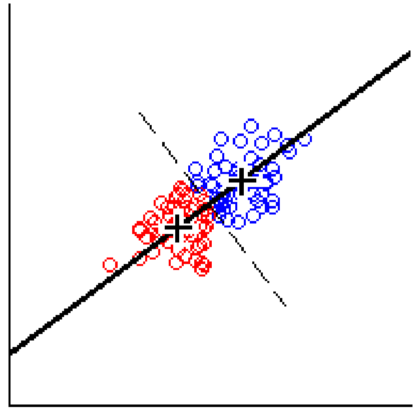
  →Need to find new features, that can effectively **summarize** all the other features

- Principle: Given two features, we want to find a single line that describes both features at once. We then map our old features onto this new line to get a new single feature.

  →Reduce from n-dimension to k-dimension: Find k vectors onto which to project the data so as to minimize the projection error

- Linear regression vs PCA:
  - In linear regression, we minimize the **squared error** from every point to our predictor line. These are vertical distances.
  - In PCA, we minimize the **shortest distance**, or shortest *orthogonal* distances, to our data points.

PCA Algorithm



- Data preprocessing: Feature scaling + Mean normalization
- Algorithm:
  1. Compute **covariance matrix**: $\Sigma = \frac{1}{\mathrm{m}} \sum_{i=1}^{m} \left(x^{(i)}\right)\left(x^{(i)}\right)^T$; $n \times n$ *matrix*
  2. Compute **eigenvectors** of covariance matrix Σ
  3. Take the first k columns of the matrix and compute z: $z^{(i)} = U_{reduce}^T . x^{(i)}$; $k \times 1$ *vector*
- Reconstruction from Compressed Representation: $x_{approx}^{(i)} = U_{reduce} . z^{(i)}$

  PS1: We can only get approximations of our original data
  PS2: $\mathrm{U}_{\mathrm{reduce}}$ is Unitary Matrix (Orthogonal Matrix)
- Choosing the Number of Principal Components:
  1. Calculate the average squared projection error: $\mathrm{A} = \frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} - x_{approx}^{(i)} \right\|^2$
  2. Calculate the total variation in the data: $\mathrm{B} = \frac{1}{m} \sum_{i=1}^{m} \left\| x^{(i)} \right\|^2$
  3. Choose k to be the smallest value such that: $\frac{A}{B} \leq 0{,}01$

  →99% of the variance is retained
- PS1: PCA is applied only on the training set and not on the cross-validation or test sets
- PS2: Don't use PCA to prevent overfitting

Anomaly Detection

- "Model" p(x) → probability that x_{test} is anomalous → Use a **threshold** ϵ
  - If anomaly detector flags **too many** anomalous examples → Decrease ϵ

- "Model" p(x) → probability that x_{test} is anomalous → Use a **threshold** ϵ

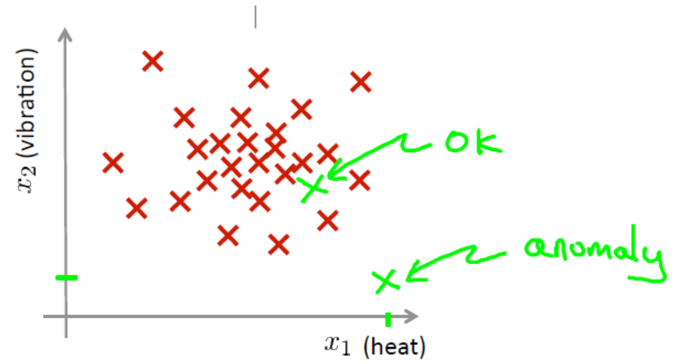
- Gaussian distribution: Probability of distribution of x $Gaussian \Leftrightarrow x \sim N(\mu, \sigma^2)$
  - ➢ $p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
  - ➢ Mean: Center of the curve: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
  - ➢ Standard deviation: Width of the curve: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$

- Independence assumption: $p(x) = \prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2)$

- Algorithm: Compute μ and σ² → Compute p(x) → Anomaly if p(x)<ϵ

- Data set:
  - Non-anomalous: 60/20/20 → train/CV/test
  - Anomalous: 50/50 → CV/test

  PS: CV to choose the threshold ϵ



Multivariate Gaussian distribution:

$$p(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{-\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}}$$

- Transformations on features: $\log(x), \log(x+c), \sqrt{x}, \sqrt[3]{x}$ → Do not have bell-shaped curve

Anomaly Detection	Supervised Learning
Small number of positive examples and large number of negative examples	Large number of both positive and negative examples
Different "types" of anomalies and it is hard for the algorithm to learn from positive examples what the anomalies look like	Enough positive examples for the algorithm to get a sense of what new positives examples look like

Recommender Systems

- <u>Example</u>: We are trying to recommend movies to customers

- <u>Notations</u>:
  - n_u = number of users
  - n_m = number of movies
  - r(i,j) = 1 if user j has rated movie i
  - y(i,j) = rating given by user j to movie I
  - x_1 = romance rate in a movie
  - x_2 = action rate in a movie
  - $\Theta^{(j)}$ = parameter vector for user j
  - $x^{(i)}$ = feature vector for movie i
  - $m^{(j)}$ = number of movies rated by user j
  - For user j, movie i, predicted rating: $\left(\boldsymbol{\theta}^{(j)}\right)^{\boldsymbol{T}}\left(\boldsymbol{x}^{(i)}\right)$ (<u>Linear regression</u>)

- <u>Learn parameter θ</u>: $\min_{\theta^{(1)},\dots,\theta^{(n_u)}} = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left(\left(\boldsymbol{\theta}^{(j)}\right)^{\boldsymbol{T}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$

  Choose i such that r(i,j) = 1

  Eleminate the constant $\frac{1}{m}$

- <u>Lean feature x</u>: $\min_{x^{(1)},\dots,x^{(n_m)}} = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left(\left(\boldsymbol{\theta}^{(j)}\right)^{\boldsymbol{T}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$

  → It is difficult to find features such as “romance rate” and “action rate in a movie

  Randomly guess the values for θ to guess the features repeatedly

- <u>Collaborative filtering</u>: $J(x,\theta) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\left(\boldsymbol{\theta}^{(j)}\right)^{\boldsymbol{T}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$

  Initialization of x’s and θ’s to small values

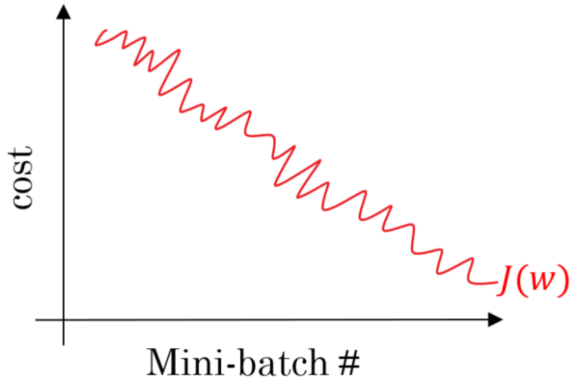
- PS: For new users, we use mean normalization → New user won’t rank all the movies 0

Large Datasets

- Large dataset > Good algorithm → Computational problems
- Large scale machine learning: Datasets can approach m = 100,000,000. Gradient descent will make a summation over m
- Stochastic gradient descent:
  1. Randomly shuffle dataset
  2. Repeat { for i=1,…,m: { $\theta_j = \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_j^{(i)}$ $(for\ j = 0, \ldots, n)$ } }

  →Modify parameters after each training example

  PS: Doesn't reach the global minimum. 1 epoch will be sufficient. Faster if m is large



- Mini-batch gradient descent: Use (b=$\frac{1}{10}$) examples in each iteration
  - Repeat { i=1,11,…,99; $\theta_j = \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9} cost$ $(for\ j = 0, \ldots, n)$ }
- Batch: **Vectorization** / Mini-batch: **Vectorization + Speed** / Stochastic: **Speed**
- Tip: Plot cost averaged over the last 1000 iterations

○ Average over more examples → Smaller plot

○ Smaller α → Avoid divergence

○ Add rate decay to guarantee convergence of stochastic gradient descent

Online Learning

→Continuous stream of data

- $p(y = 1/x; \theta)$ #Learn probability → Optimize price → Choose our service
- Repeat forever { Get (x[user, price],y[=0 they chose, =1 they didn't]) → $\theta_j = \theta_j - \alpha(h_\theta(x) - y)x_i$; $(j = 0, \dots, n)$ }
- PS: Learn and discard it → Can adapt to change user preface
- Map reduce & data parallelism: Parallelize the computations

  →Split data into p parts; p number of machines

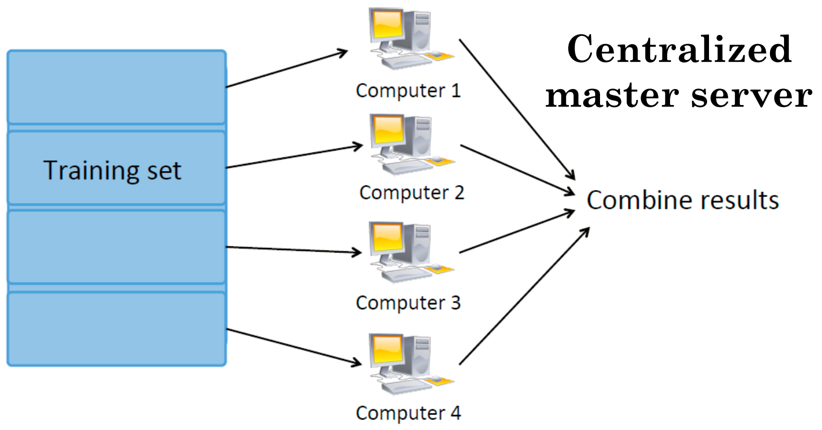
  →Each machine has $\frac{1}{p}$ to calculate

  $\theta_j = \theta_j - \alpha \frac{1}{n}(machine1 + machine2 + \cdots)$

  →Speed learning slightly less than $\times\, p$ times

  PS: If we have summation of big numbers, we use map reduce

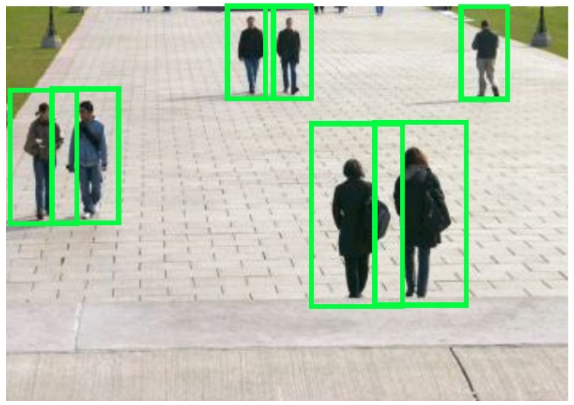
  There are also multi-core machines



Optical Character Recognition

→Extract text from images

- <u>Steps in pipeline</u>:
  1. **Text detection** → Sliding window detection with step stride / size
     1. White region that may contain text
     2. Expansion operator: Turn nearby pixels to white
     3. Choose the white ones that has dimensions of a text
  2. **Character segmentation**
     → 1D sliding window for character segmentation
     → Detect gaps and make lines to split between two
  3. **Character classification**



Artificial Data Synthesis

→Generate new images from scratch (not real data)

PS: Distortions should be representing the type of noise in the data

PS: Do not add random meaningless noise to the data

Tip: Make sure you have a low bias classifier

Question: "How much work to get 10x as much data as we currently have?"

<u>Techniques</u>:

1. **Artificial data synthesis**: Make distortions and modifications on data
2. Collect/label data by yourself → Calculate number of hours needed?
3. **Crowd source**: Hire people to label data for you (Eg. Amazon Mechanical Turk)

Ceiling Analysis

→Estimate the importance of each composition in the pipeline

1. Check accuracy of the whole system
2. Check accuracy of the system by giving it perfect next component labels
3. …
4. Check the difference between the different accuracies and choose the biggest ones