DATA MINING CLASSIFICATION

Neural Networks

Word Embeddings

Classification Issues

Classification Evaluation

Supervised Learning Pipeline

NEURAL NETWORKS

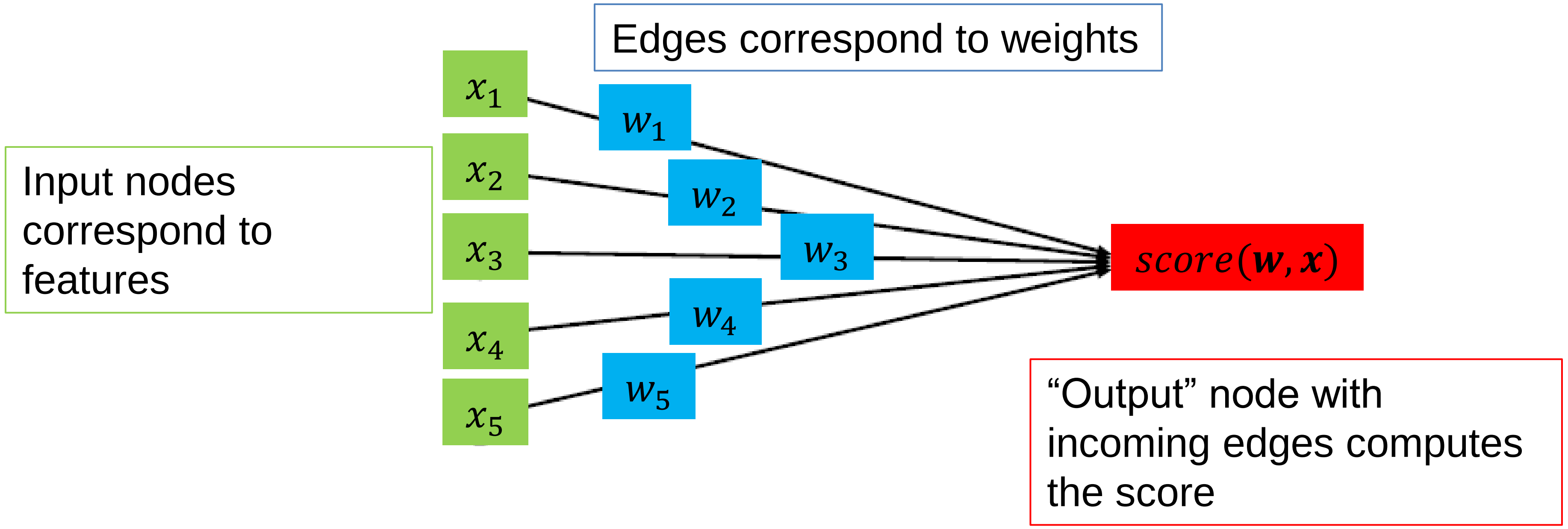
(Thanks to Philipp Koehn for the material borrowed from his slides)

Linear Classification

- A simple model for classification is to take a linear combination of the feature values and compute a score.
- Input: Feature vector $\boldsymbol{x} = (x_1, \dots, x_n)$
- Model: Weights $\boldsymbol{w} = (w_1, \dots, w_n)$
- Output: $score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$
- Make a decision depending on the output score.
  - E.g.: Decide “Yes” if $score(\boldsymbol{w}, \boldsymbol{x}) > 0$ and “No” if $score(\boldsymbol{w}, \boldsymbol{x}) < 0$
- The perceptron classification algorithm

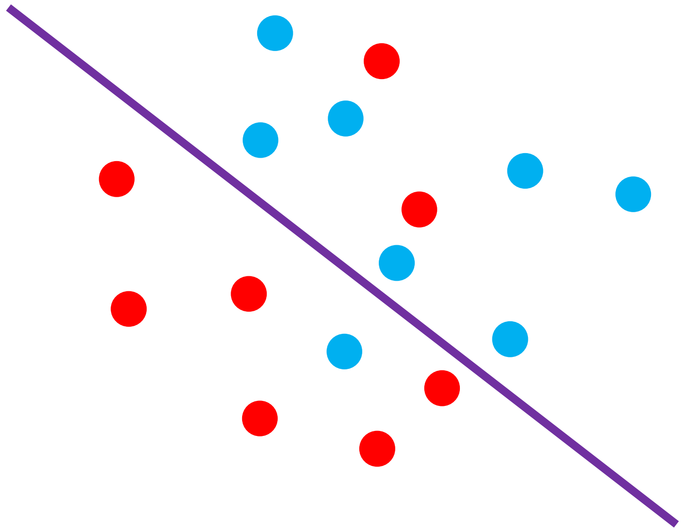
Linear Classification

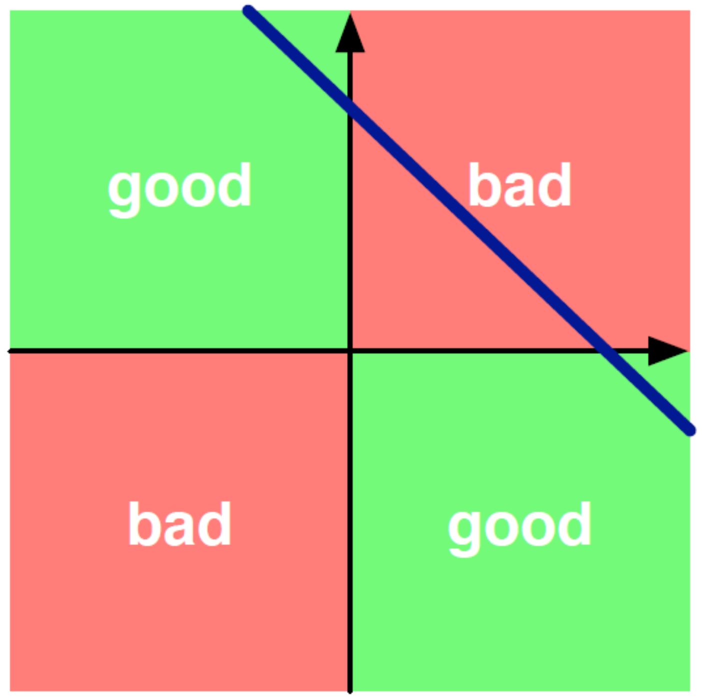
- We can represent this as a network


Edges correspond to weights
x_1
x_2
x_3
x_4
x_5
w_1
w_2
w_3
w_4
w_5
Input nodes correspond to features
$score(\boldsymbol{w}, \boldsymbol{x})$
"Output" node with incoming edges computes the score

Linear models

- Linear models partition the space according to a hyperplane

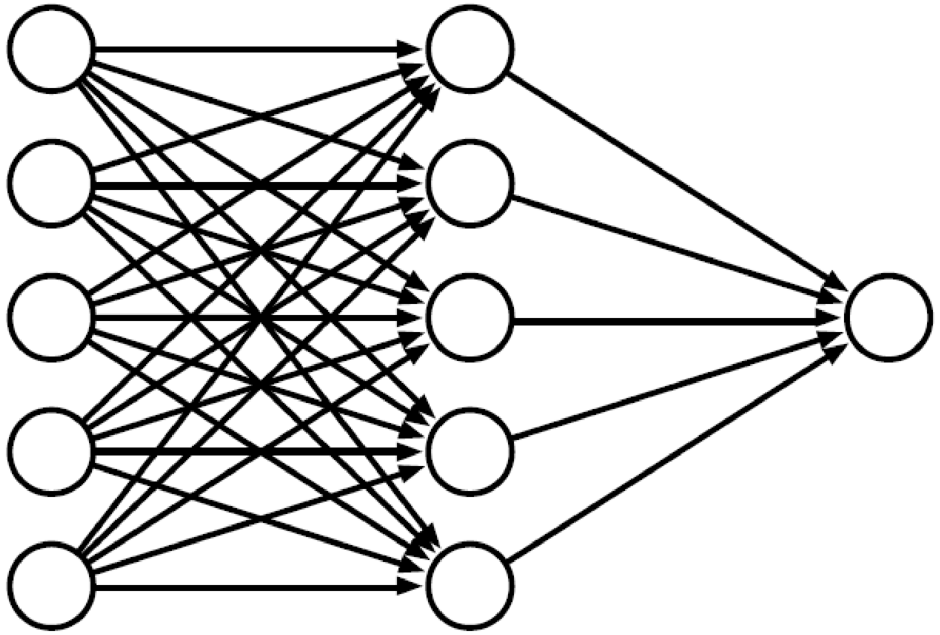




- But they cannot model everything

Multiple layers

- We can add more layers:
  - Each arrow has a weight
  - Nodes compute scores from incoming edges and give input to outgoing edges



Did we gain anything?

Non-linearity

- Instead of computing a linear combination

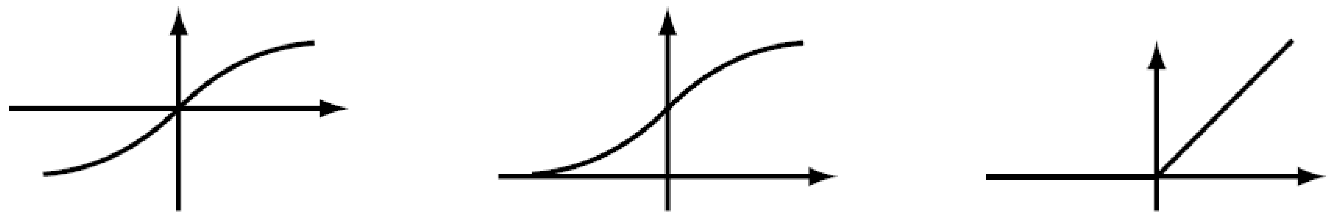
$$score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$$

- Apply a non-linear function on top:

$$score(\boldsymbol{w}, \boldsymbol{x}) = g\left(\sum_i w_i x_i\right)$$

- Popular functions:

$\tanh(x)$ $\quad$ $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ $\quad$ $\text{relu}(x) = \max(0, x)$

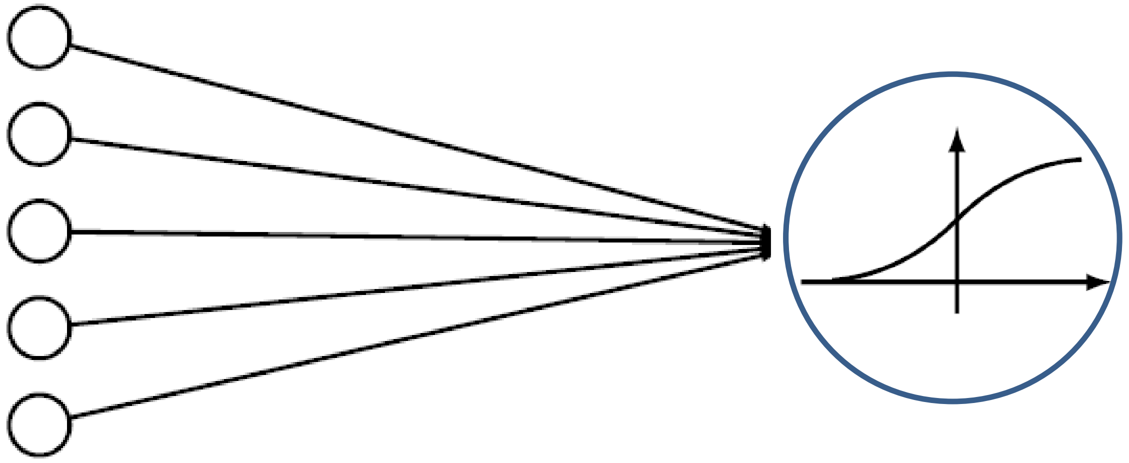


(sigmoid is also called the "logistic function")

These functions play the role of a soft "switch" (threshold function)

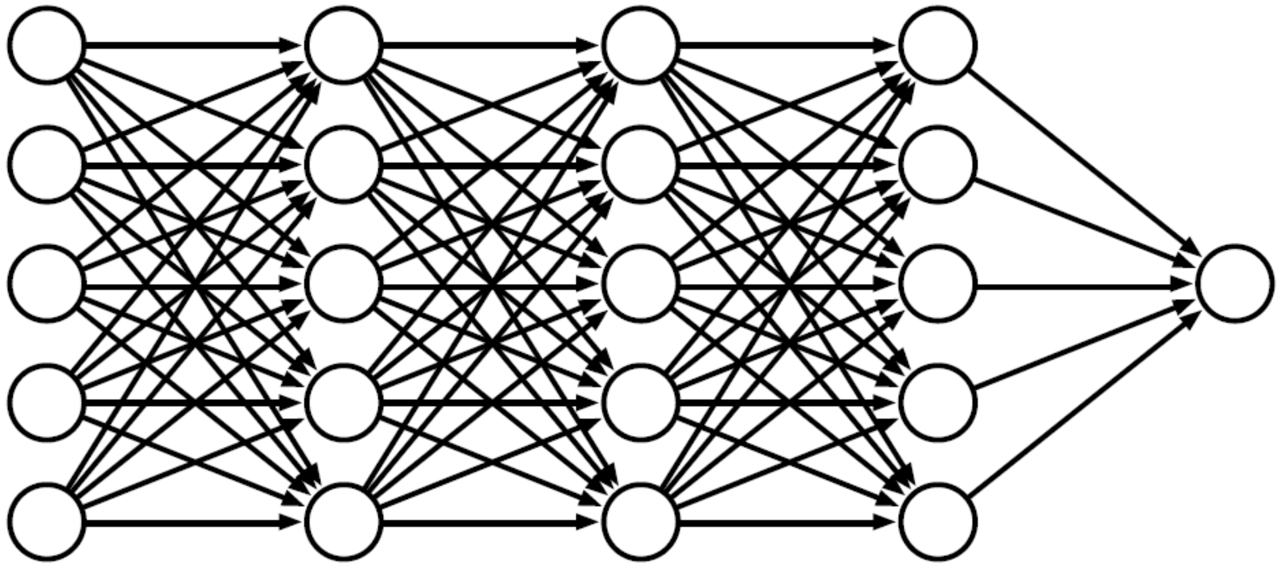
Side note

- Logistic regression classifier:
  - Single layer with a logistic function



Deep learning

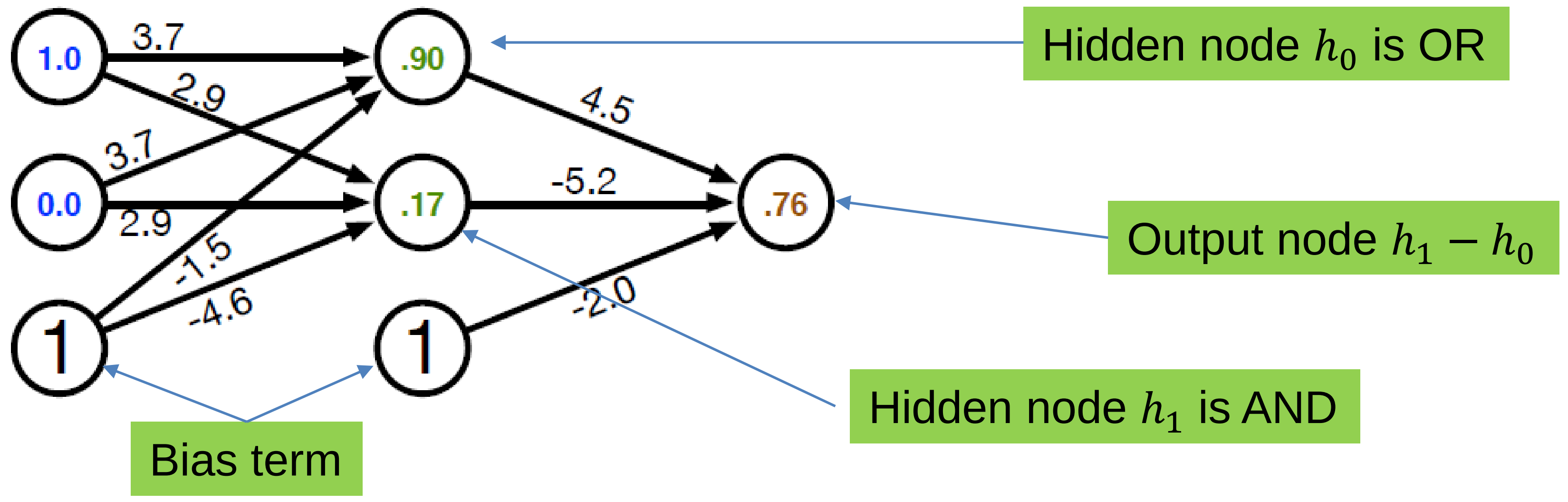
- Networks with multiple layers



- Each layer can be thought of as a processing step
- Multiple layers allow for the computation of more complex functions

Example

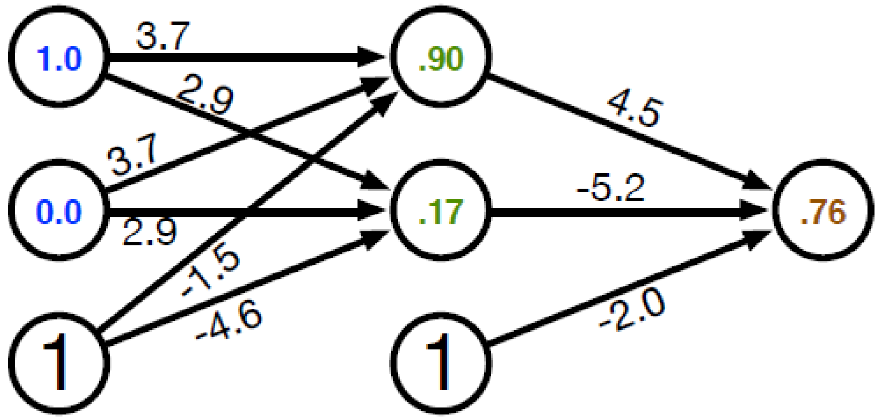
- A network that implements XOR



Input x_0	Input x_1	Hidden h_0	Hidden h_1	Output y_0
0	0	0.12	0.02	$0.18 \to 0$
0	1	0.88	0.27	$0.74 \to 1$
1	0	0.73	0.12	$0.74 \to 1$
1	1	0.99	0.73	$0.33 \to 0$

Error

- The computed value is 0.76 but the correct value is 1
  - There is an error in the computation

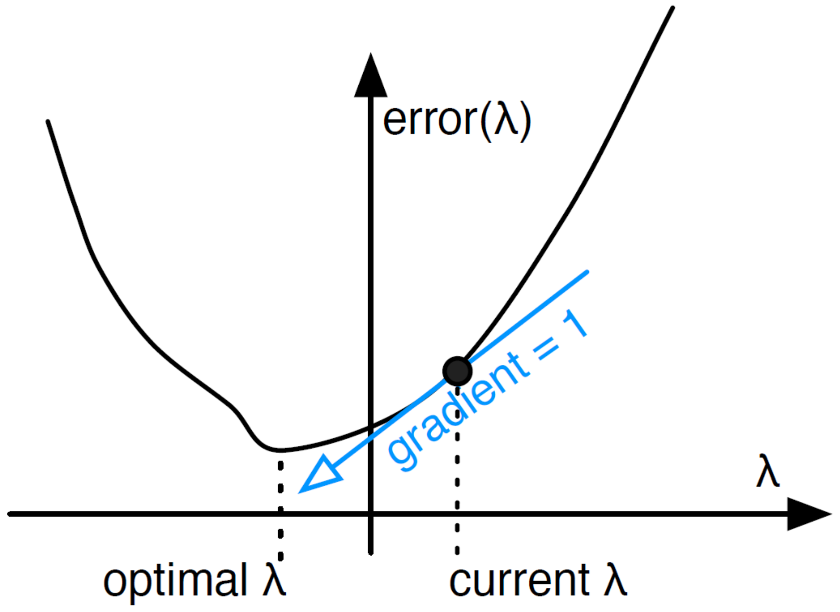


  - How do we set the weights so as to minimize this error?

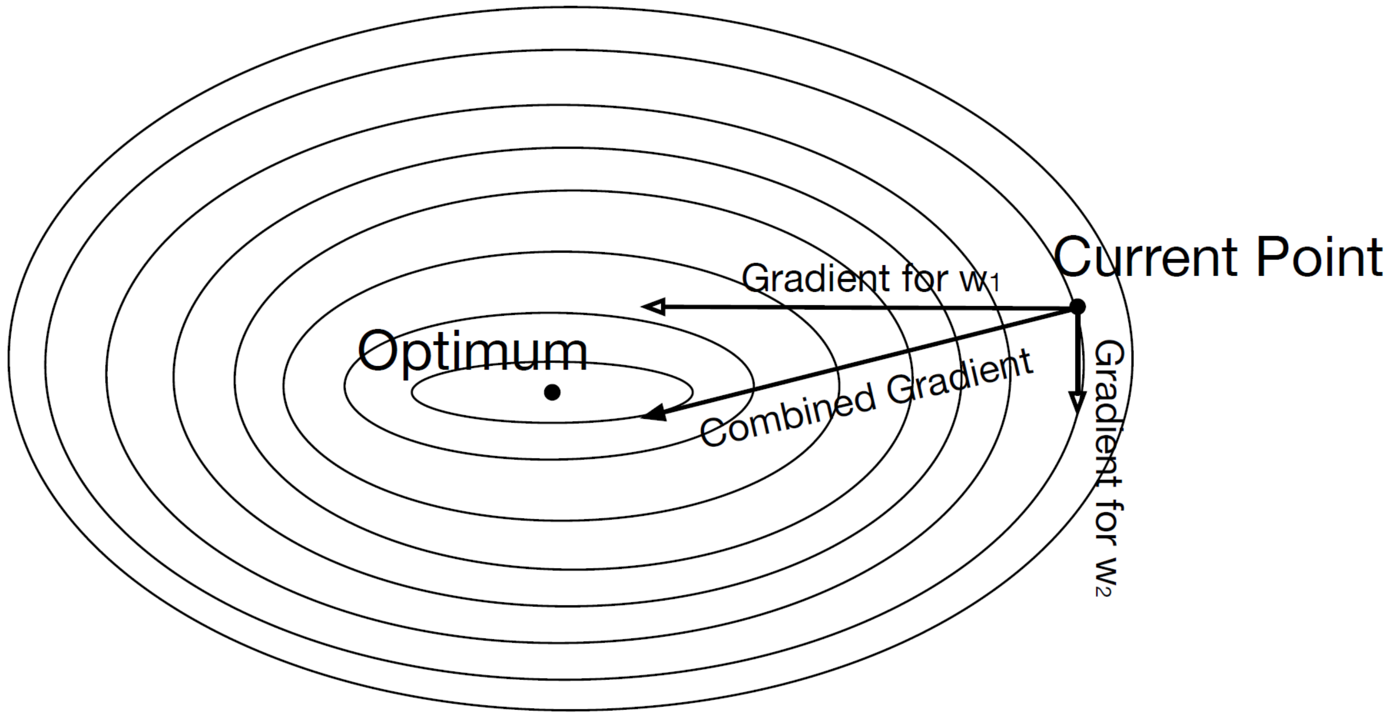
Gradient Descent

- The error is a function of the weights
- We want to find the weights that minimize the error
- Compute gradient: gives the direction to the minimum
- Adjust weights, moving at the direction of the gradient.

Gradient Descent

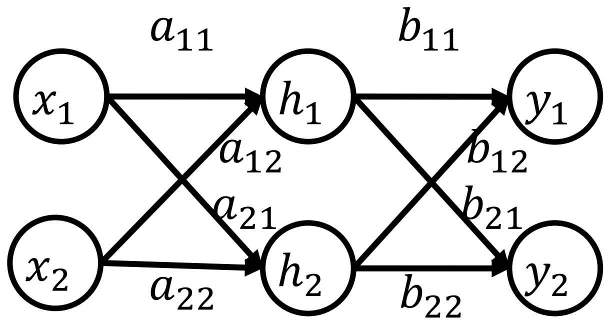


Gradient Descent



Backpropagation

- How can we compute the gradients? Backpropagation!
- Main idea:
  - Start from the final layer: compute the gradients for the weights of the final layer.
  - Use these gradients to compute the gradients of previous layers using the chain rule
  - Propagate the error backwards
- Backpropagation essentially is an application of the chain rule for differentiation.



Notation:
Activation function: g

$s_{y_1} = b_{11}h_1 + b_{12}h_2\ ,\ y_1 = g(s_{y_1})$

$s_{y_2} = b_{21}h_1 + b_{22}h_2\ ,\ y_2 = g(s_{y_2})$

$s_{h_1} = a_{11}x_1 + a_{12}x_2\ ,\ h_1 = g(s_{h_1})$

$s_{h_2} = a_{21}x_1 + a_{22}x_2\ ,\ h_2 = g(s_{h_2})$

Error: $E = \|y - t\|^2 = (y_1 - t_1)^2 + (y_2 - t_2)^2$

$$\frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial s_{y_1}} \frac{\partial s_{y_1}}{\partial b_{11}} = \delta_{y_1} h_1$$

$$\boxed{\delta_{y_1} = \frac{\partial E}{\partial s_{y_1}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial s_{y_1}} = 2(y_1 - t_1)g'(s_{y_1})}$$

$$\frac{\partial E}{\partial b_{21}} = \delta_{y_2} h_1$$

$$\boxed{\delta_{y_2} = \frac{\partial E}{\partial s_{y_2}} = 2(y_2 - t_2)g'(s_{y_2})}$$

$$\frac{\partial E}{\partial b_{12}} = \delta_{y_1} h_2 \qquad \frac{\partial E}{\partial b_{22}} = \delta_{y_2} h_2$$

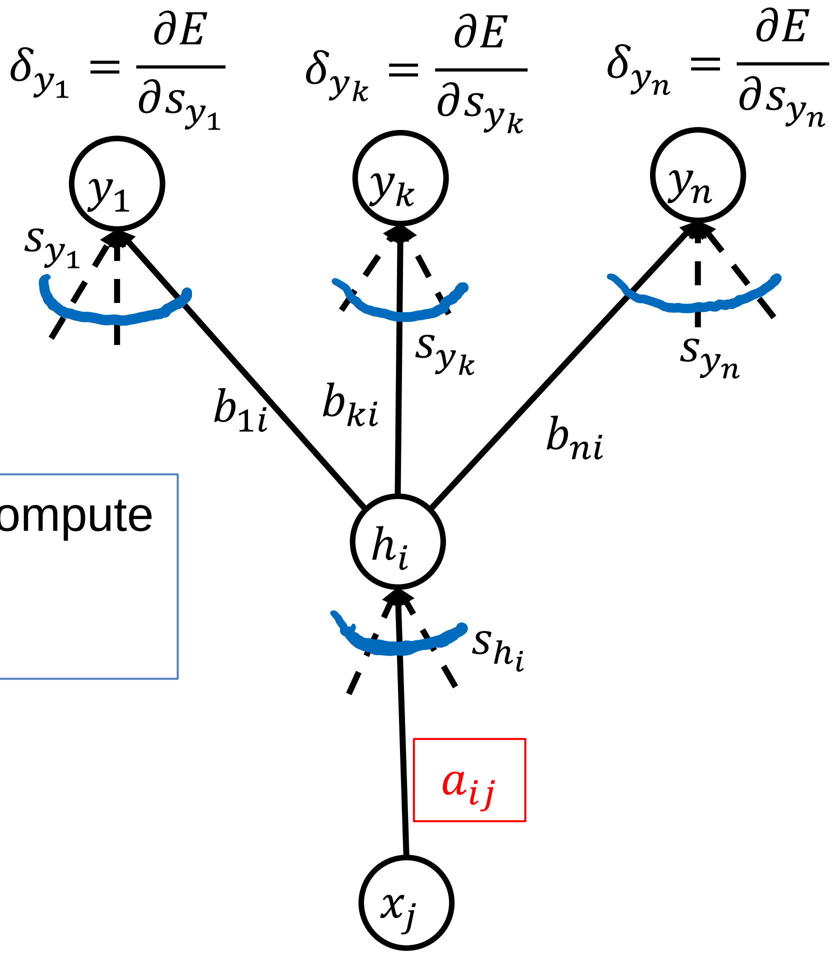
$$\frac{\partial E}{\partial a_{11}} = \frac{\partial E}{\partial s_{h_1}}\frac{\partial s_{h_1}}{\partial a_{11}} = \delta_{h_1} x_1 \qquad \frac{\partial E}{\partial a_{22}} = \frac{\partial E}{\partial s_{h_2}}\frac{\partial s_{h_2}}{\partial a_{22}} = \delta_{h_2} x_2 \qquad \frac{\partial E}{\partial a_{21}} = \delta_{h_1} x_2 \qquad \frac{\partial E}{\partial a_{12}} = \delta_{h_2} x_1$$

$$\delta_{h_1} = \frac{\partial E}{\partial s_{h_1}} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial s_{h_1}} = \left(\frac{\partial E}{\partial s_{y_1}}\frac{\partial s_{y_1}}{\partial h_1} + \frac{\partial E}{\partial s_{y_2}}\frac{\partial s_{y_2}}{\partial h_1}\right) g'(s_{h_1}) = (\delta_{y_1} b_{11} + \delta_{y_2} b_{21}) g'(s_{h_1})$$

$$\delta_{h_2} = (\delta_{y_1} b_{12} + \delta_{y_2} b_{22}) g'(s_{h_2})$$

Backpropagation

We have already computed the δ_{y_k}'s



$\delta_{y_1} = \frac{\partial E}{\partial s_{y_1}}$
$\delta_{y_k} = \frac{\partial E}{\partial s_{y_k}}$
$\delta_{y_n} = \frac{\partial E}{\partial s_{y_n}}$
y_1
y_k
y_n
s_{y_1}
s_{y_k}
s_{y_n}
b_{1i}
b_{ki}
b_{ni}
h_i
s_{h_i}
a_{ij}
x_j

We want to compute
$$\frac{\partial E}{\partial a_{ij}}$$

$$\boxed{\frac{\partial E}{\partial a_{ij}} = \sum_{k=1}^{n} \delta_{y_k} b_{ki}\, g'\left(s_{h_i}\right) x_j}$$

For the sigmoid activation function:
$$g(t) = \frac{1}{1+e^{-t}}$$

The derivative is:
$$g'(t) = g(t)(1 - g(t))$$

This makes it easy to compute it. We have:
$$g'\left(s_{h_i}\right) = h_i(1 - h_i)$$

Therefore
$$\frac{\partial E}{\partial a_{ij}} = \sum_{k=1}^{n} \delta_{y_k} b_{ki}\, h_i(1 - h_i) x_j$$

Stochastic gradient descent

- Ideally the loss should be the average loss over all training data.
- We would need to compute the loss for all training data every time we update the gradients.
  - However, this is expensive.
- Stochastic gradient descent: Consider one input point at the time. Each point is considered only once.
- Intermediate solution: Use mini-batches of data points.

WORD EMBEDDINGS

Thanks to Chris Manning for the slides

Basic Idea

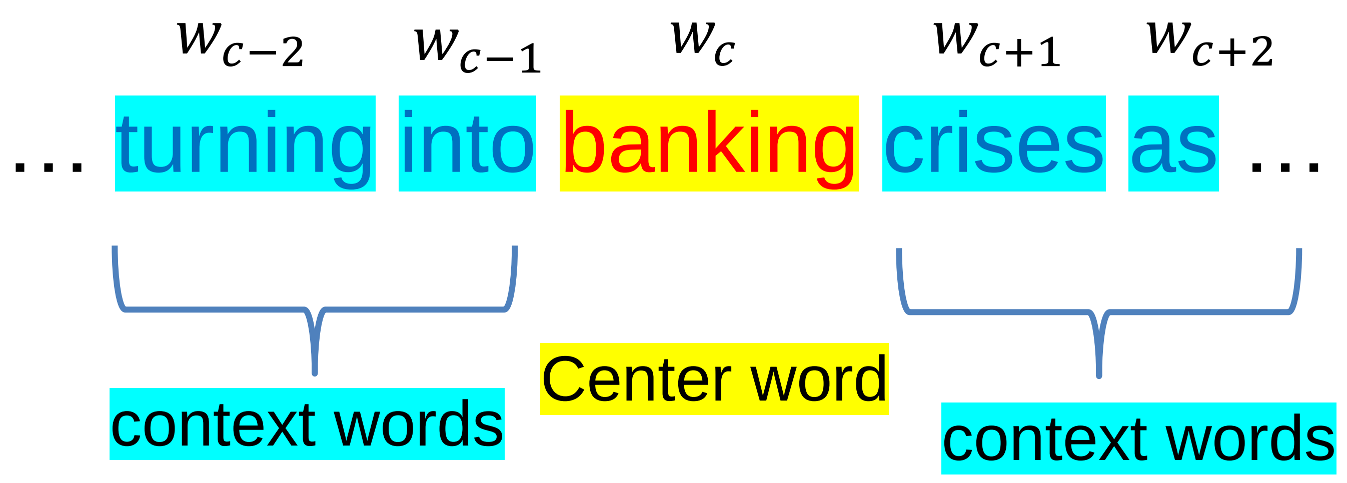
- You can get a lot of value by representing a word by means of its neighbors
- “You shall know a word by the company it keeps”
    - (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Basic idea

Define a model that aims to predict between a center word w_c and context words in some window of length m in terms of word vectors



window of size 2 each side

Word2Vec

Predict between every word and its context words

Two **algorithms**

1. Skip-grams (SG)

   Predict context words given the center word

   $P(w_{c-1}|w_c),\ P(w_{c-2}|w_c),\ P(w_{c+1}|w_c),\ P(w_{c+2}|w_c)$

2. Continuous Bag of Words (CBOW)

   Predict center word from a bag-of-words context

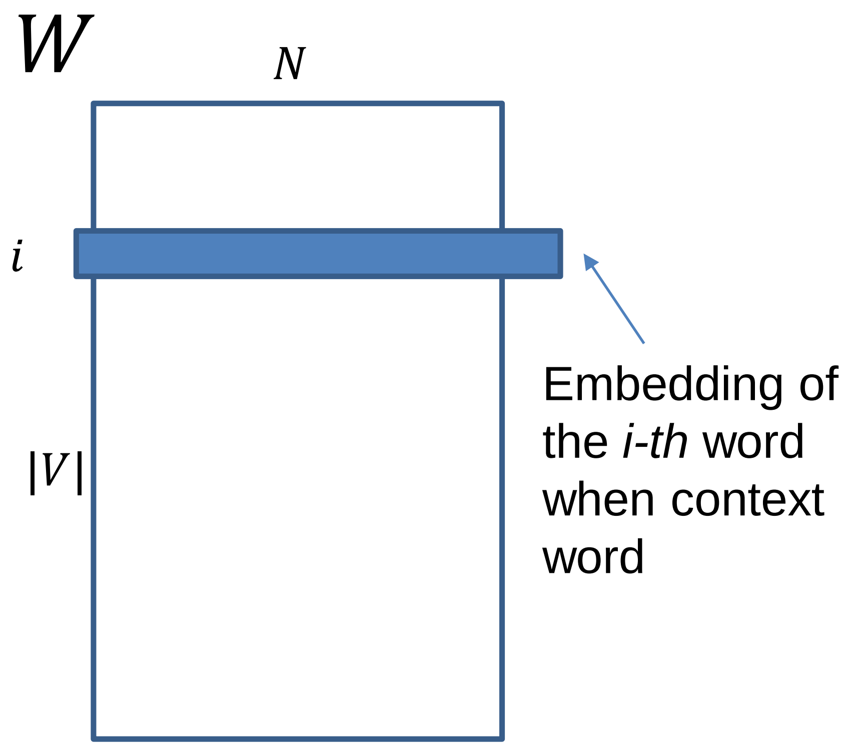
   $$P\left(w_c \middle| w_{c-2}\ , w_{c-1},\ w_{c+1}, w_{c+2}\right)$$

Position independent (do not account for distance from center)

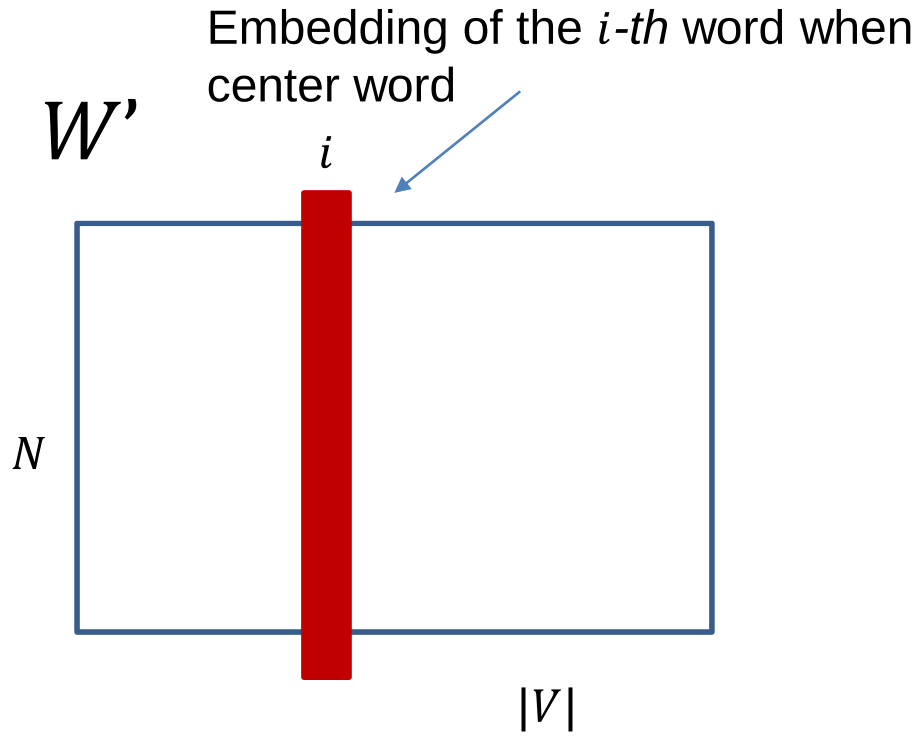
CBOW

Use a window of context words to predict the center word

Learn two matrices (N size of embedding, $|V|$ number of words)



$|V|$ x N context embeddings when *input*



N x $|V|$ center embeddings when *output*

CBOW

Given window size *m*, $x^{(c)}$ one hot vector for context words, *y* one hot vector for the center word

1. Input: the *one hot vectors* for the *2m* context words
$x^{(c-m)}, \ \ldots, \ x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)}$

2. Compute the *embeddings of the context words*
$v_{c-m} = Wx^{(c-m)}, \ \ldots, v_{c-1} = Wx^{(c-1)}, v_{c+1} = Wx^{(c+1)}, \ldots, v_{c+m} = Wx^{(c+m)}$

3. *Average* these vectors: $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\cdots v_{c+m}}{2m}$, $\hat{v} \in R^N$

4. Generate a *score vector:* $z = W' \hat{v}$

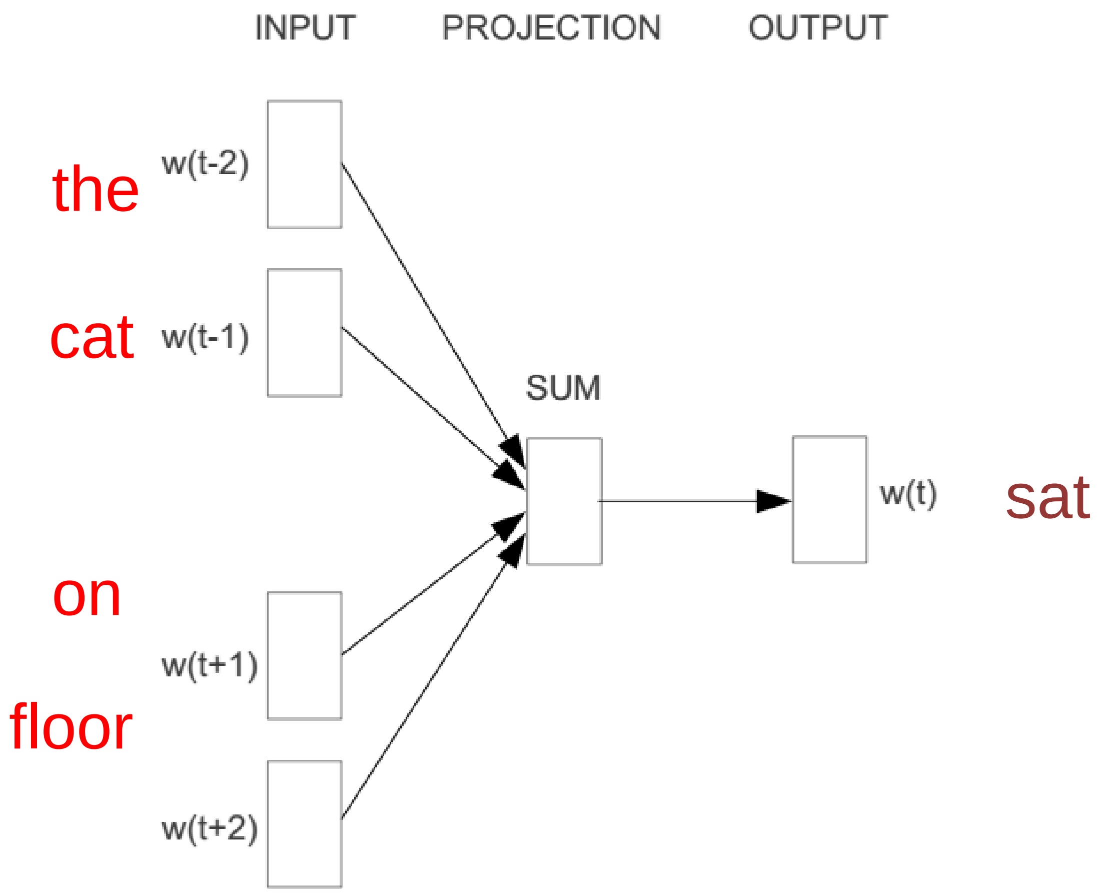
5. Turn the *score vector to probabilities:* $\hat{y}$ = *softmax(z)*

Softmax

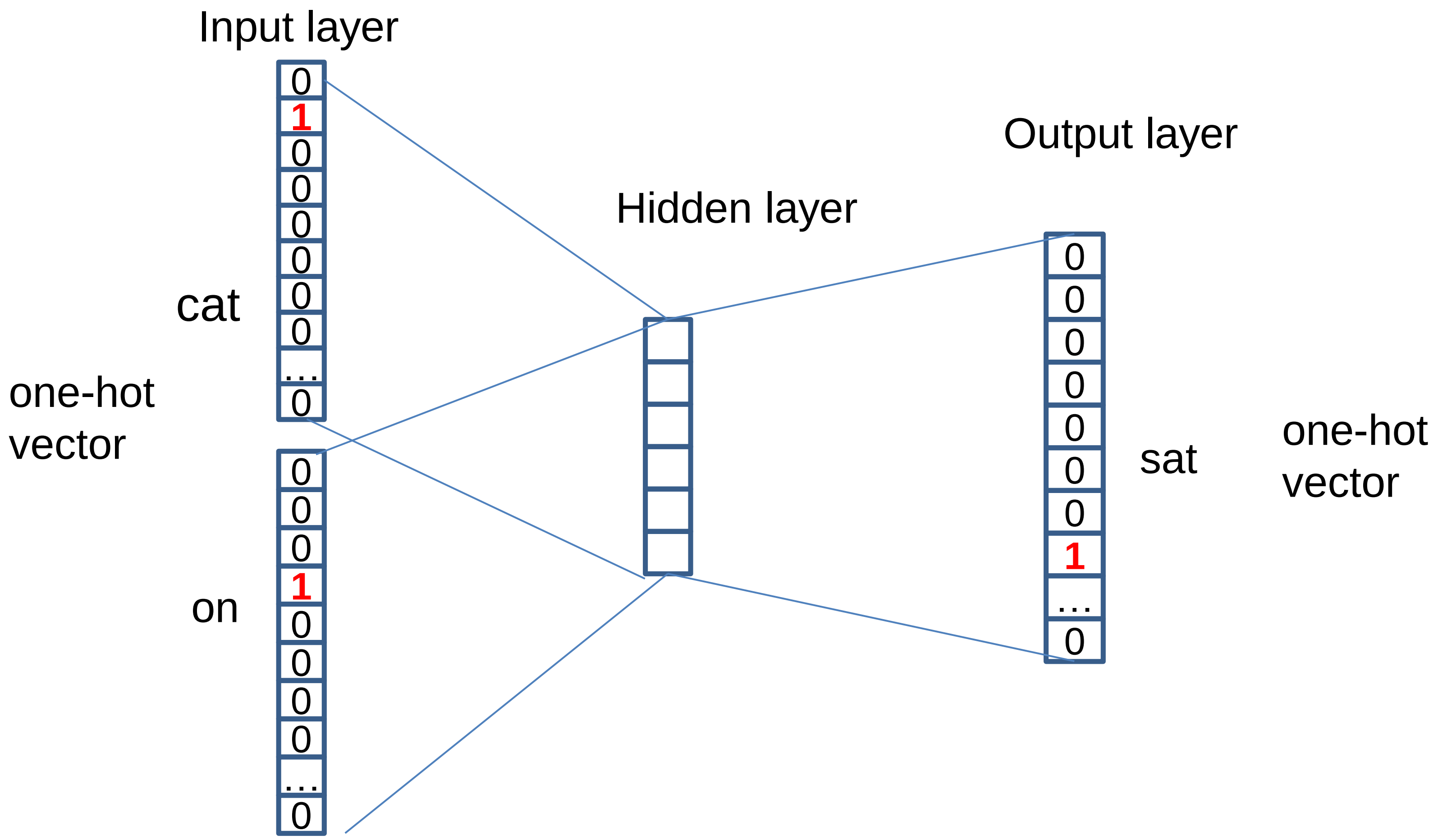
$$p_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

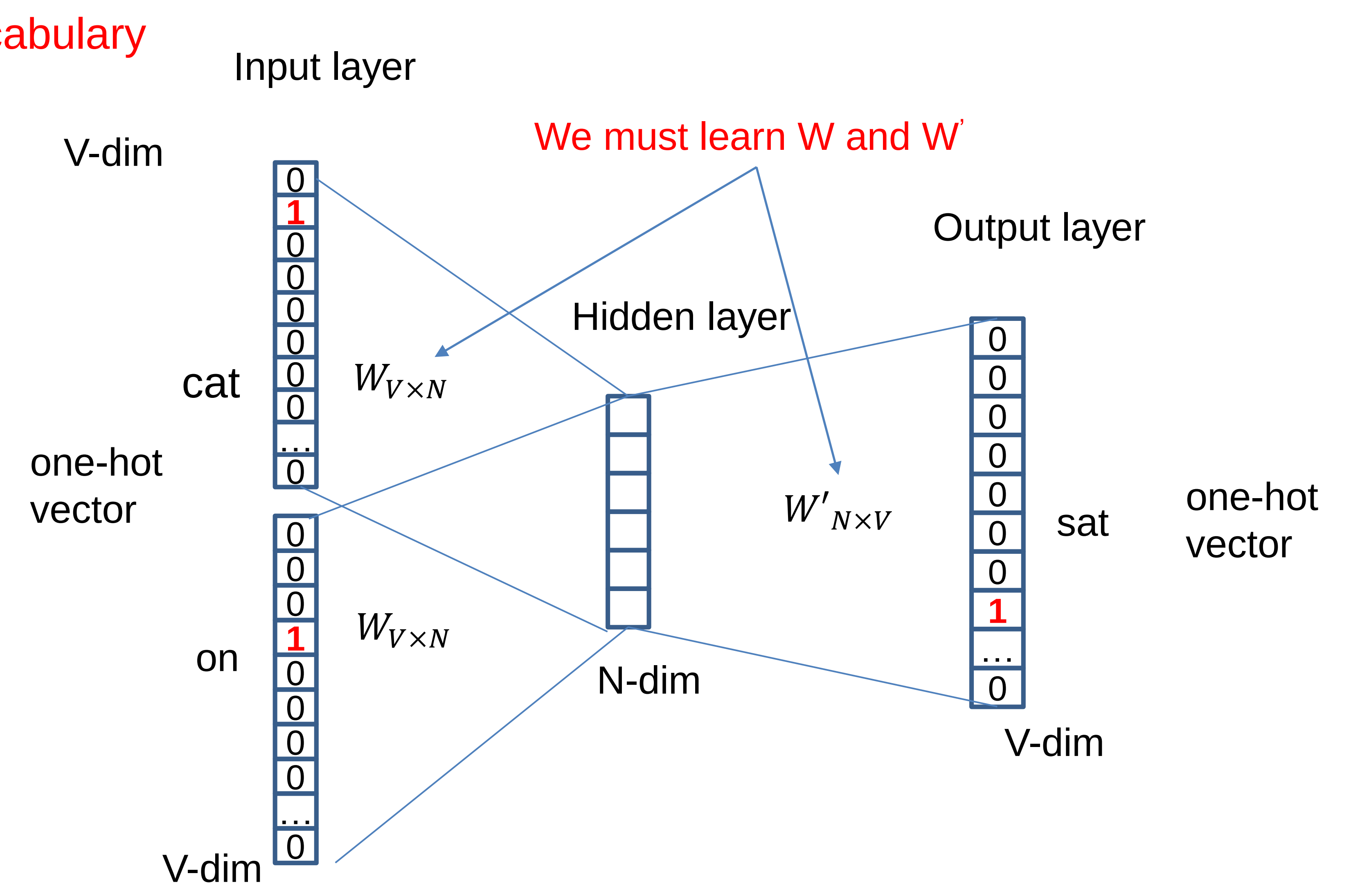
We want this to be close to 1 for the center word

- E.g. “The cat sat on floor”
  - Window size = 2



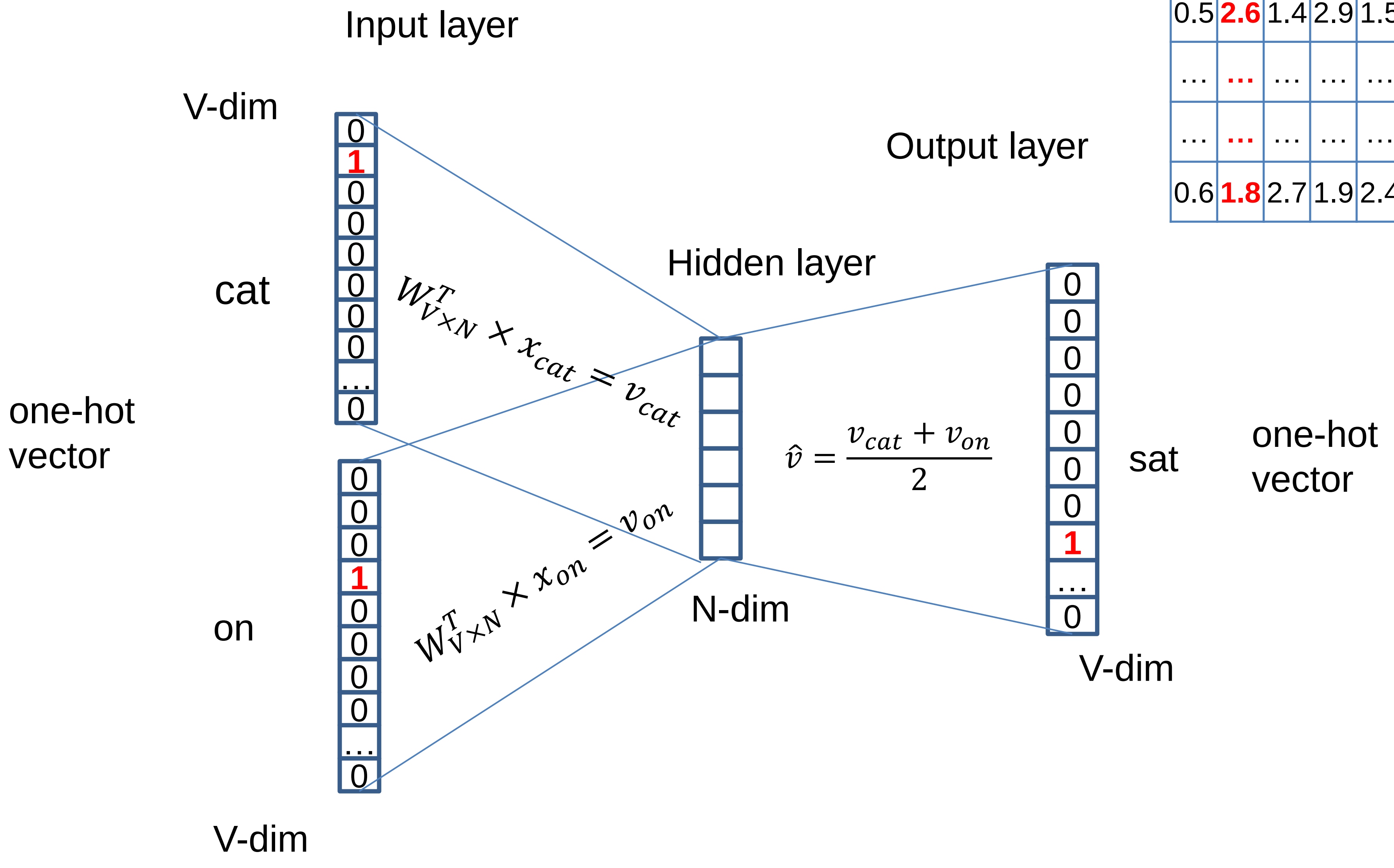
Index of cat in vocabulary




Index of cat in vocabulary
Input layer
V-dim
We must learn W and W'
Output layer
Hidden layer
cat
$W_{V \times N}$
one-hot vector
0 1 0 0 0 0 0 0 … 0
$W'_{N \times V}$
sat
one-hot vector
0 0 0 1 0 0 0 0 … 0
on
$W_{V \times N}$
0 0 0 0 0 0 0 1 … 0
N-dim
V-dim
V-dim

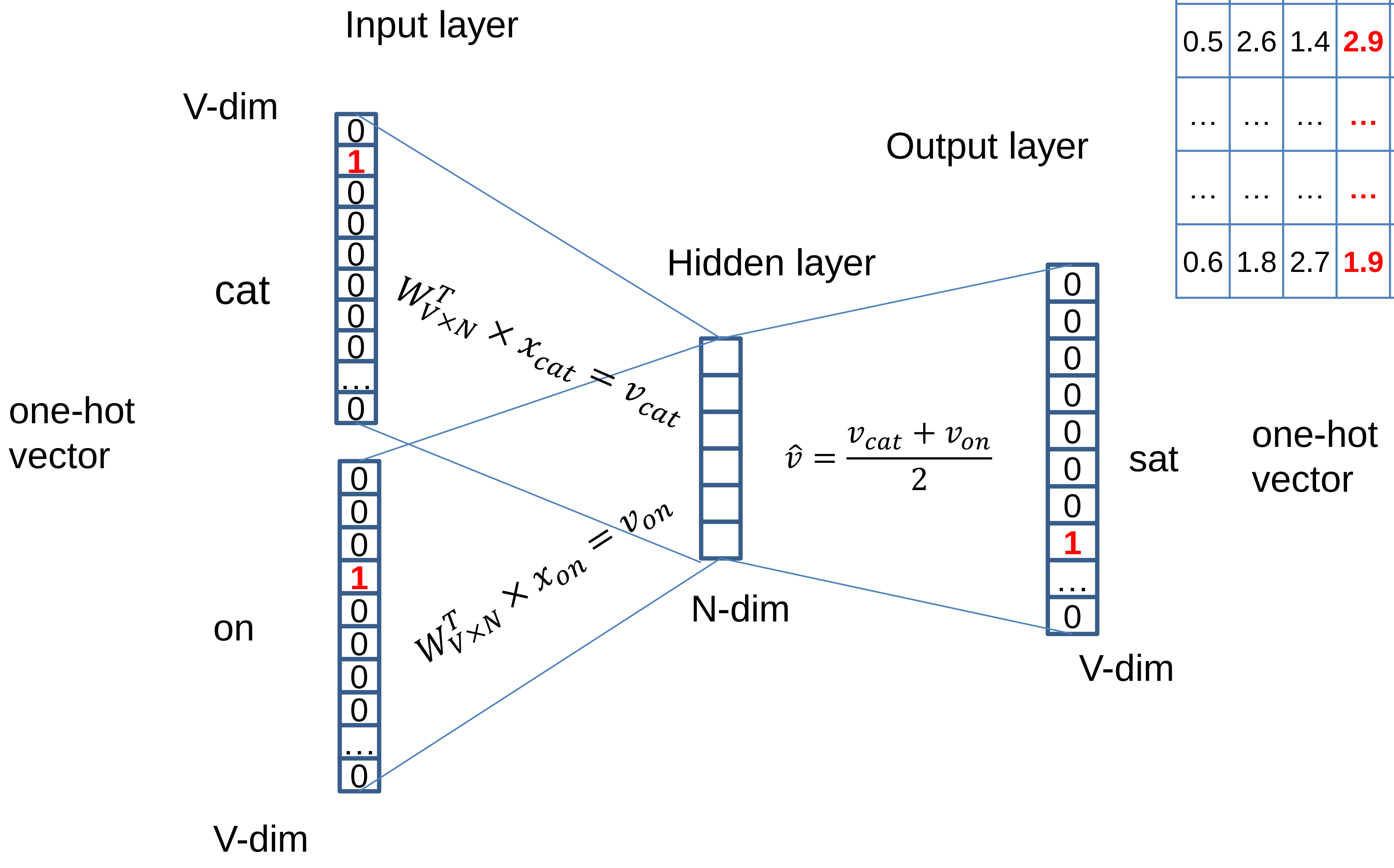
$$W^T_{V\times N} \times x_{cat} = v_{cat}$$

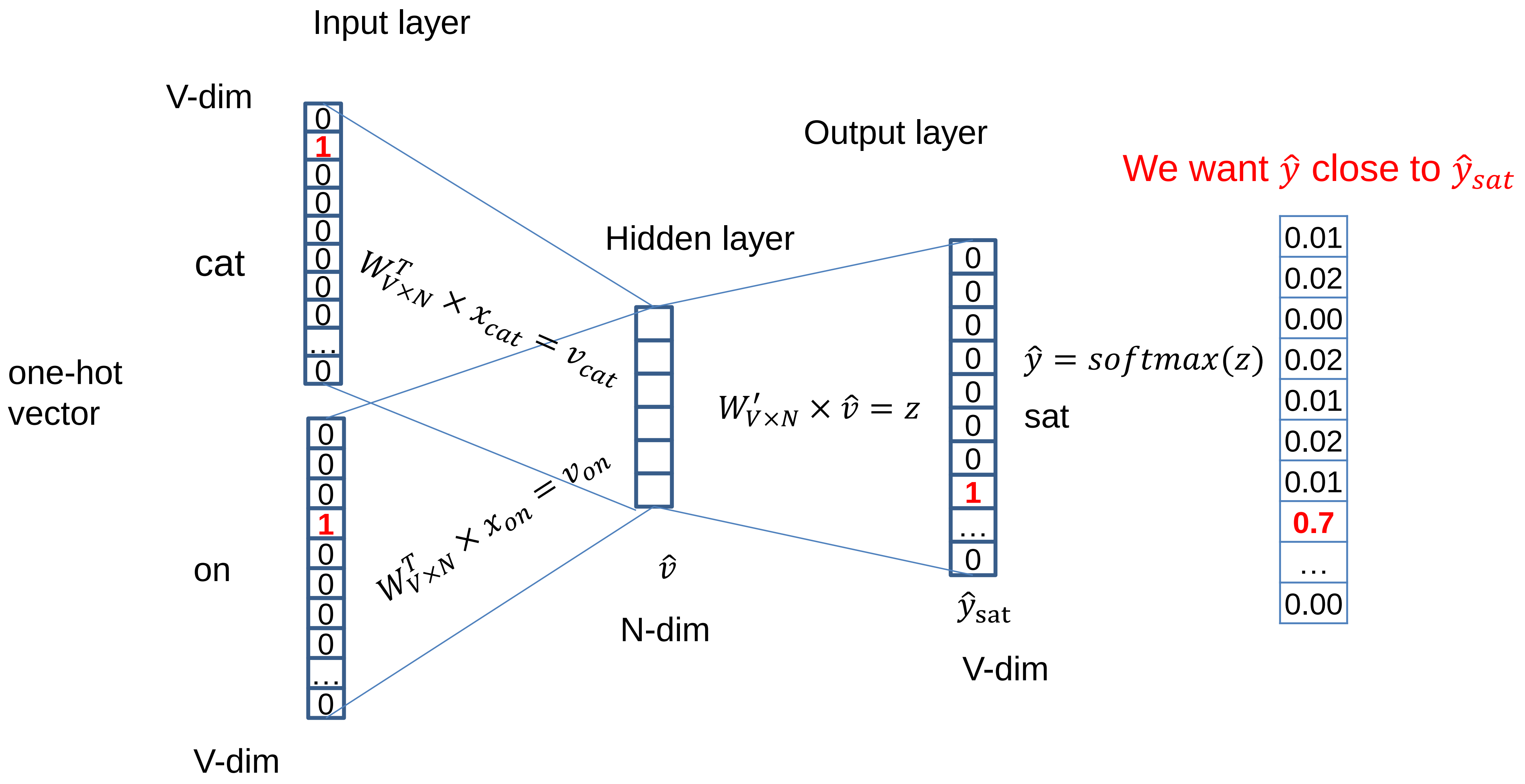
$$\begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 2.6 \\ \dots \\ \dots \\ 1.8 \end{bmatrix}$$

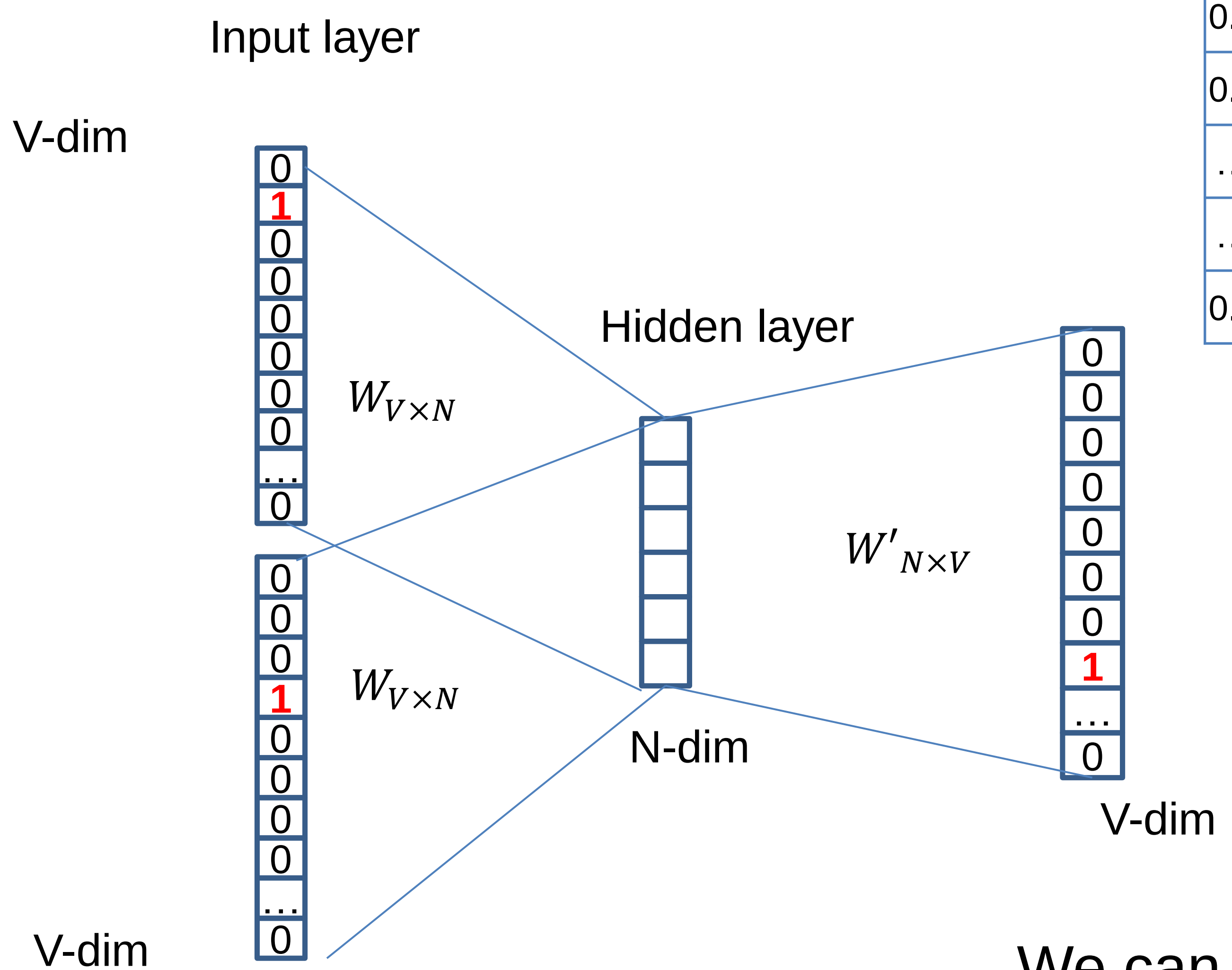

Input layer
V-dim
cat
one-hot vector
on
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
$W^T_{V\times N} \times x_{on} = v_{on}$
Hidden layer
N-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Output layer
sat
one-hot vector
V-dim

$$W_{V\times N}^{T} \times x_{on} = v_{on}$$

$$\begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.9 \\ \dots \\ \dots \\ 1.9 \end{bmatrix}$$


Input layer
V-dim
cat
one-hot vector
on
V-dim
$W_{V\times N}^{T} \times x_{cat} = v_{cat}$
$W_{V\times N}^{T} \times x_{on} = v_{on}$
Hidden layer
N-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Output layer
sat
one-hot vector
V-dim


Input layer
V-dim
cat
one-hot vector
on
V-dim
$W_{V\times N}^{T} \times x_{cat} = v_{cat}$
$W_{V\times N}^{T} \times x_{on} = v_{on}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
sat
$\hat{y}_{sat}$
V-dim
We want $\hat{y}$ close to $\hat{y}_{sat}$
0
1
0
0
0
0
0
0
…
0
0
0
0
1
0
0
0
0
…
0
0
0
0
0
0
0
0
1
…
0
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
…
0.00



$W_{V\times N}$

0.1	**2.4**	1.6	1.8	0.5	0.9	…	…	…	3.2
0.5	**2.6**	1.4	2.9	1.5	3.6	…	…	…	6.1
…	**…**	…	…	…	…	…	…	…	…
…	**…**	…	…	…	…	…	…	…	…
0.6	**1.8**	2.7	1.9	2.4	2.0	…	…	…	1.2

The word embeddings

We can consider either W (context) or W' (center) as the word's representation.
Or even take the average.

Skipgram

Given the center word,   predict (or, generate) the context words

W: $N \times |V|$, input matrix, word representation as center word
W': $|V| \times N$, output matrix, word representation as context word

$y^{(j)}$ one hot vector for context words

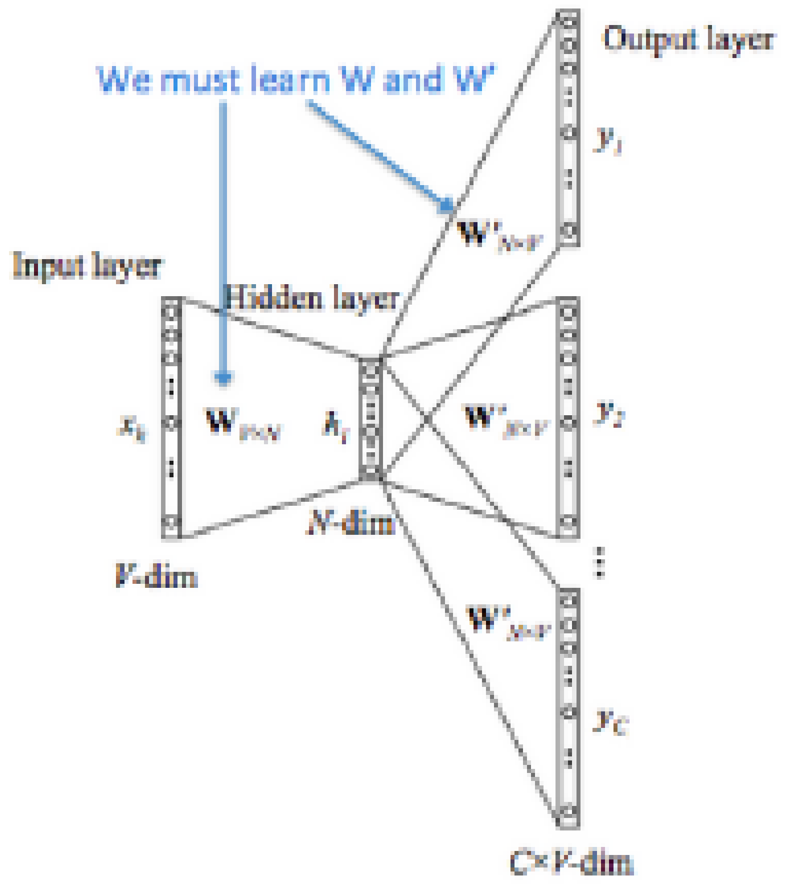
1. Get *one hot vector* of the center word x^c

2. Get the *embedding of the center word:* $v_c = W\, x^c$

3. Get the *embedding of all context words:* $z = W'\, v_c$

5. Turn the *score vector into probabilities:* $\hat{y}$ = *softmax(z)*

We want this to be close to 1 for the context words



Skipgram

- For each word *t* = 1 … *T*, predict surrounding words in a window of "radius" *m* of every word.
- Objective function: Maximize the probability of any context word given the current center word:

likelihood

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p\left(w_{t+j} \middle| w_t; \theta\right)$$

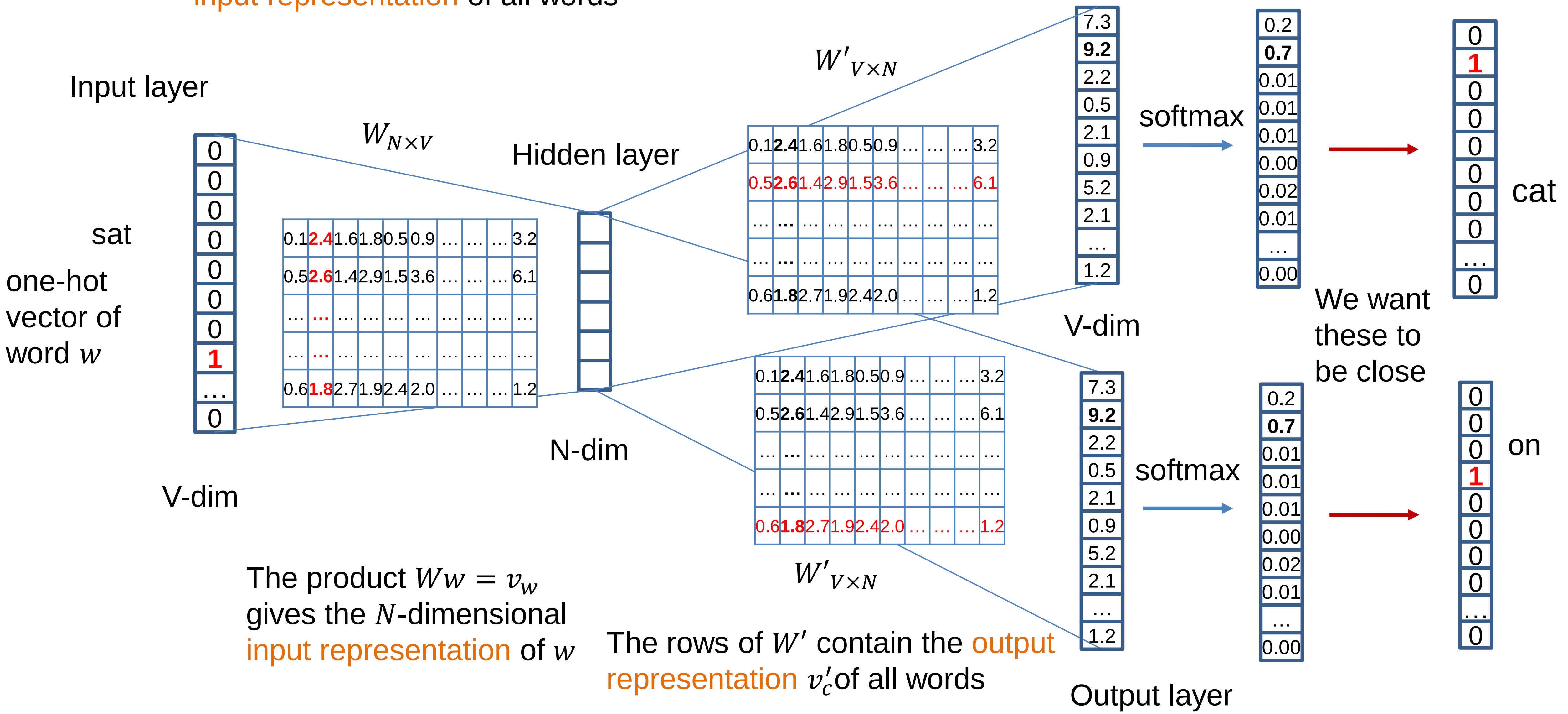
Negative
Log Likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p\left(w_{t+j} \middle| w_t; \theta\right)$$

where θ represents all variables we will optimize

The columns of W contain the input representation of all words

The product $W'v_w$ gives the dot product $v_c'v_w$ between the input presentation of w and output representation of c, for all c



The product $Ww = v_w$ gives the N-dimensional input representation of w

The rows of W' contain the output representation v_c' of all words

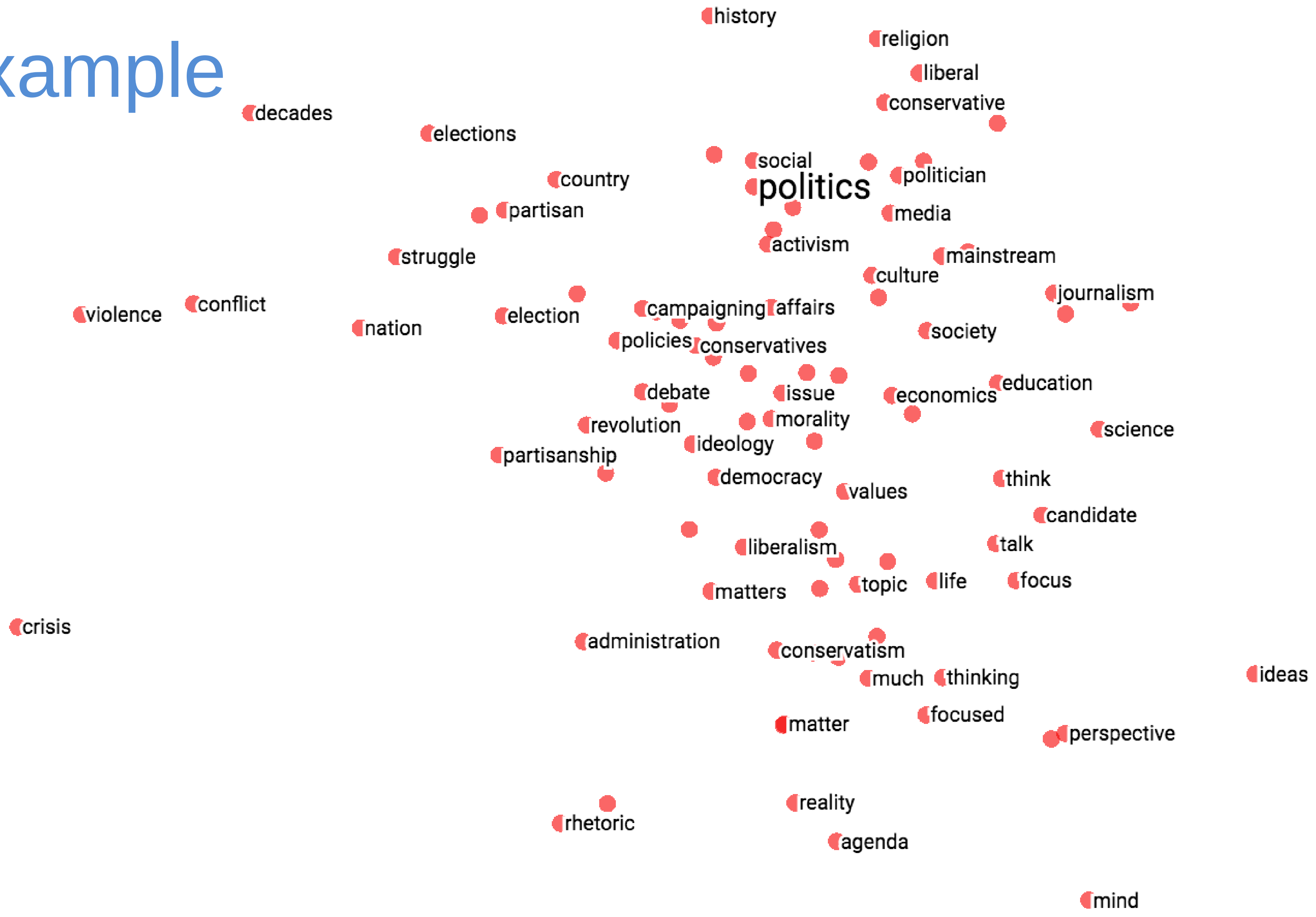
- The basic skipgram utilizes the softmax function:

$$p(c|w) = \frac{\exp\left({v'_c}^T v_w\right)}{\sum_{i=1}^{T} \exp(\,{v'_i}^T v_w)}$$

- Where:
  - T – # of words in the corpus.
  - v_w - input vector of *w*.
  - v'_w - output vector of *w*.

Word	Input	Output
King	[0.2,0.9,0.1]	[0.5,0.4,0.5]
Queen	[0.2,0.8,0.2]	[0.4,0.5,0.5]
Apple	[0.9,0.5,0.8]	[0.3,0.9,0.1]
Orang e	[0.9,0.4,0.9]	[0.1,0.7,0.2]

An example


history
religion
liberal
conservative
decades
elections
social
politics
politician
country
partisan
media
activism
mainstream
struggle
culture
journalism
violence
conflict
election
campaigning
affairs
nation
society
policies
conservatives
education
debate
issue
economics
revolution
morality
science
ideology
partisanship
democracy
values
think
candidate
talk
liberalism
matters
topic
life
focus
crisis
administration
conservatism
much
thinking
ideas
matter
focused
perspective
reality
rhetoric
agenda
mind

These representations are *very good* at encoding similarity and dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

  – $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$

  – Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  – $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$

  – $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

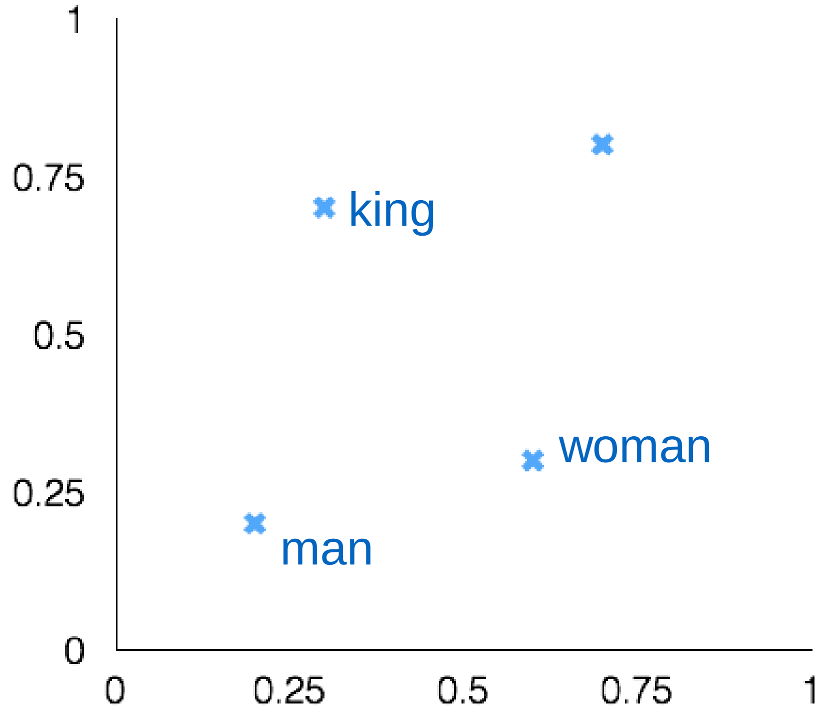
Test for linear relationships, examined by Mikolov et al.

a:b :: c:?

$$\xrightarrow{} \quad d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
–	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



OTHER CLASSIFICATION ISSUES

Expressiveness

Overfitting

Evaluation

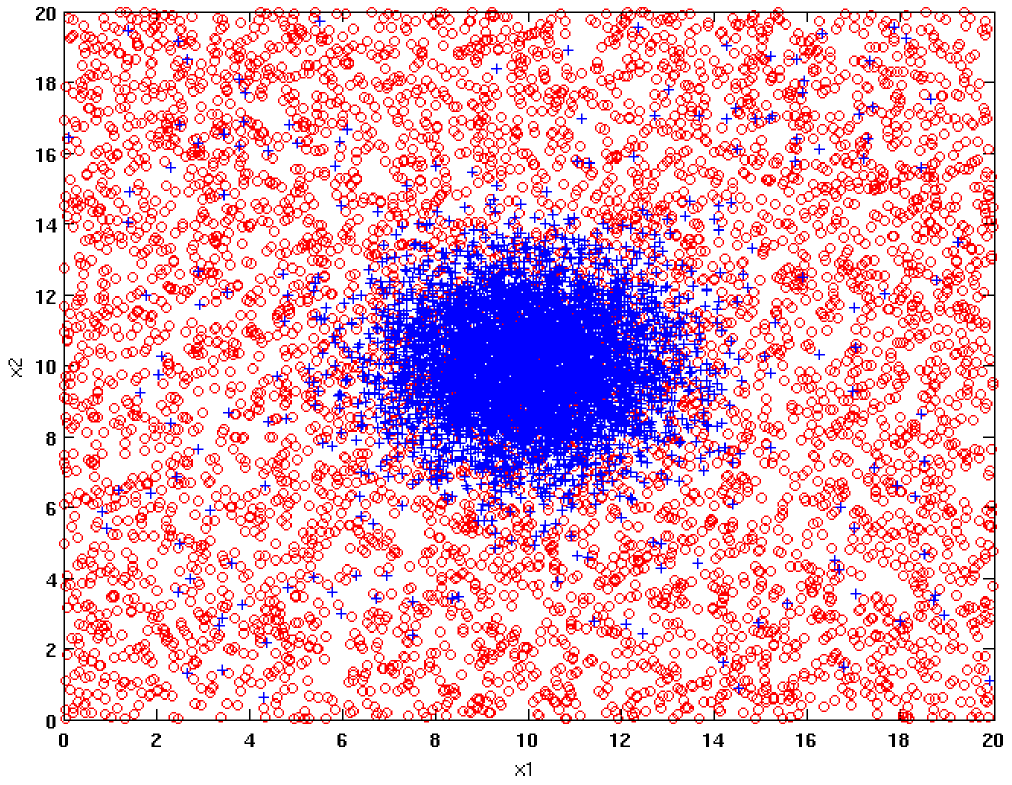
GENERALIZATION

Generalization

- Generalization refers to the ability of the classifier to correctly classify data that it has not already seen.
  - The assumption is that the new data come from the same distribution/model as the training data
- How do we measure how well a model generalizes?

- Classification errors:
  - Training errors (apparent errors): Errors committed on the training set
    - This is what we can control when creating the classifier. We hope that the training error is indicative of the generalization error, but as we will see this is not always the case
  - Test errors: Errors committed on the test set
    - This is a true measure of generalization, and this is what we want to be small, but we do not have access to the test error at training time.

Example Data Set



Two class problem:

+ : 5400 instances

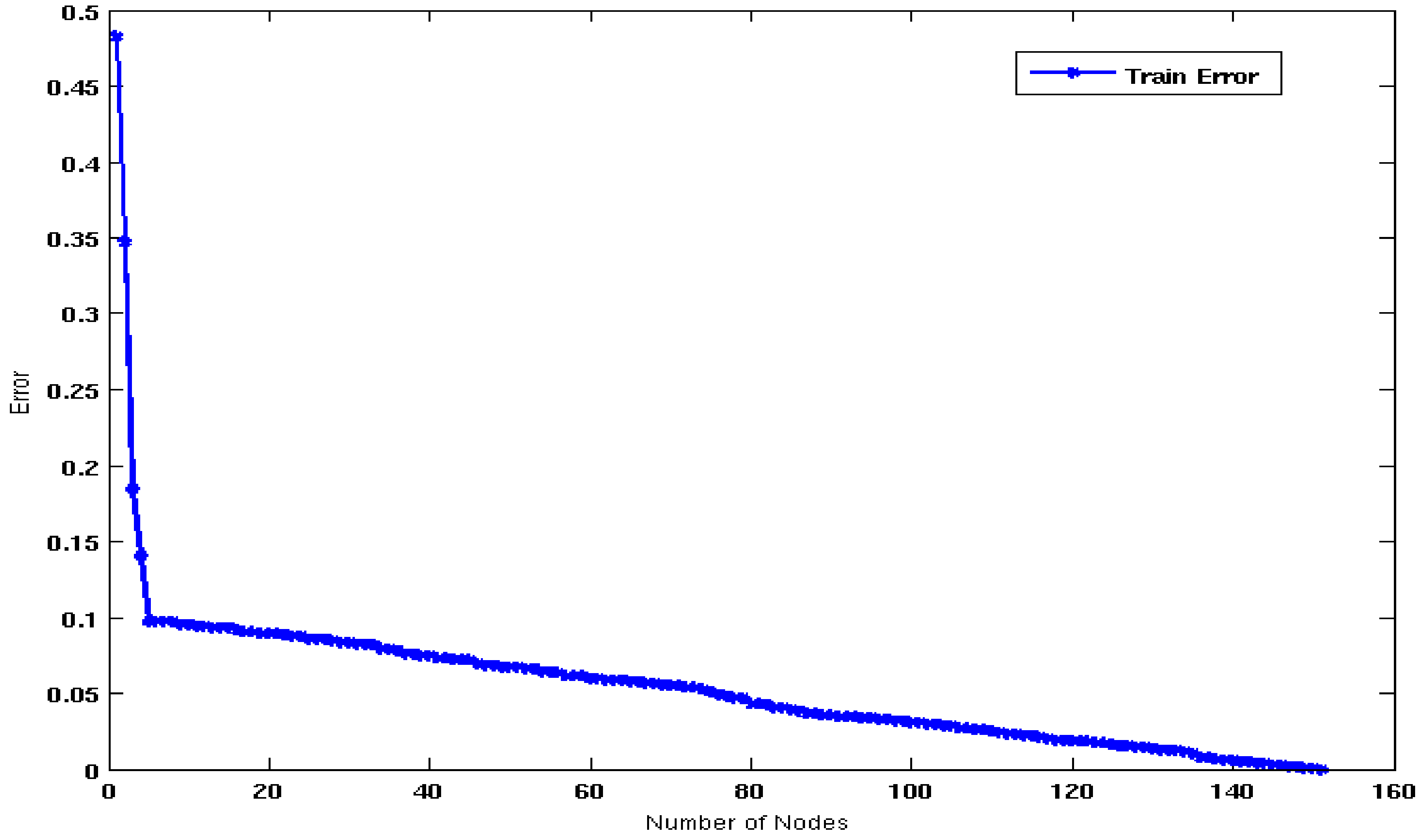
- **5000 instances generated from a Gaussian centered at (10,10)**
- **400 noisy instances added**

o : 5400 instances

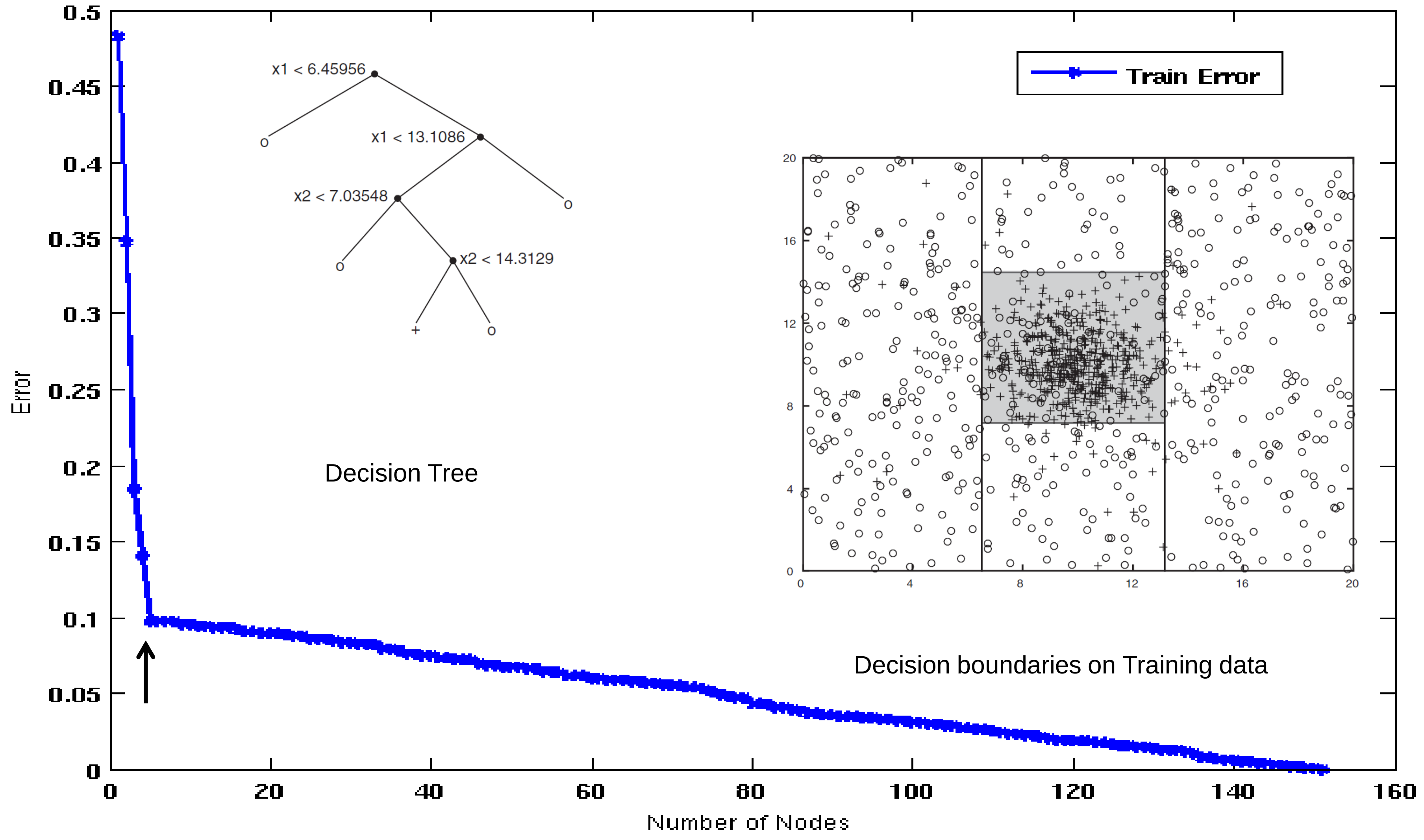
- **Generated from a uniform distribution**

10 % of the data used for training and 90% of the data used for testing

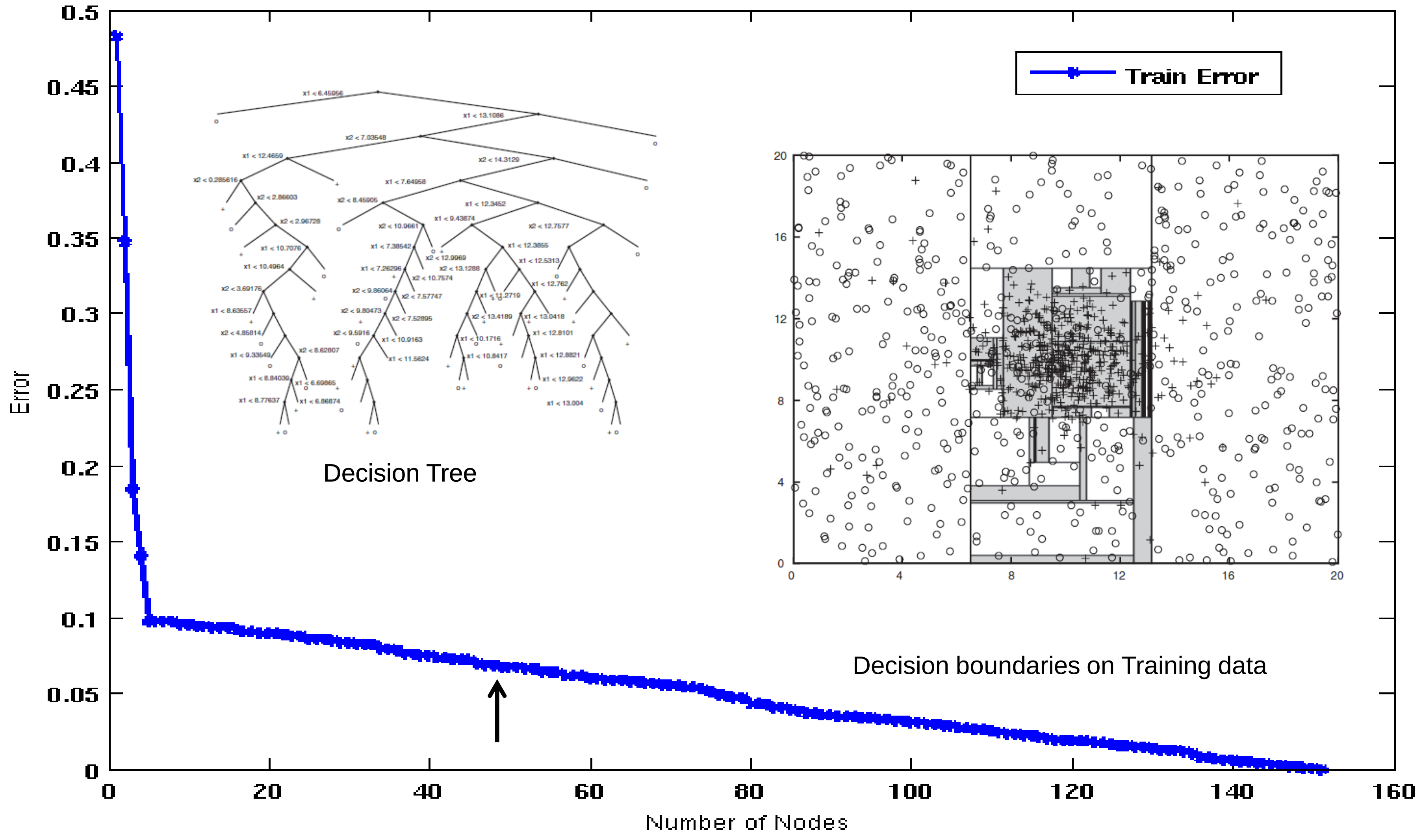
Increasing number of nodes in Decision Trees



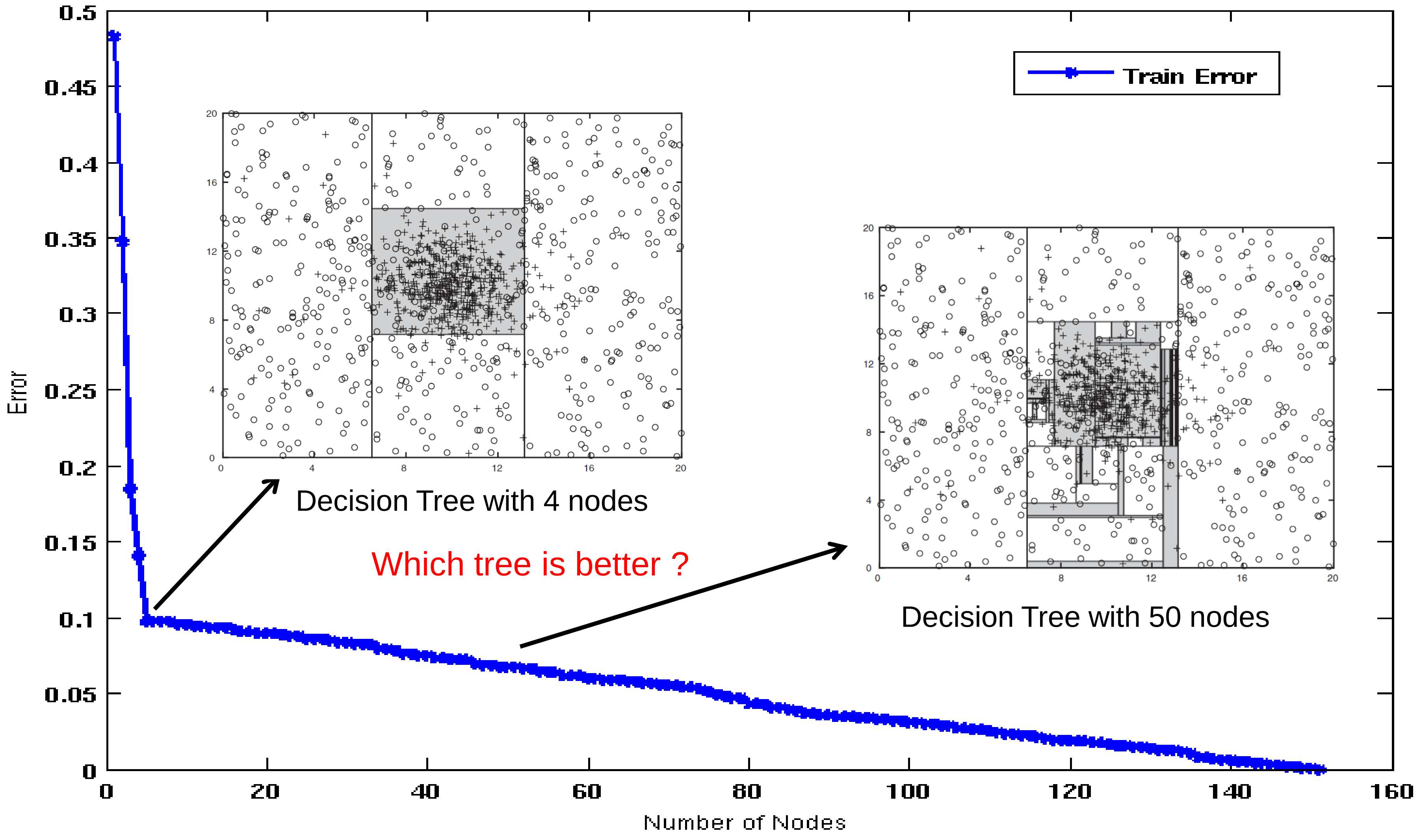
Decision Tree with 4 nodes



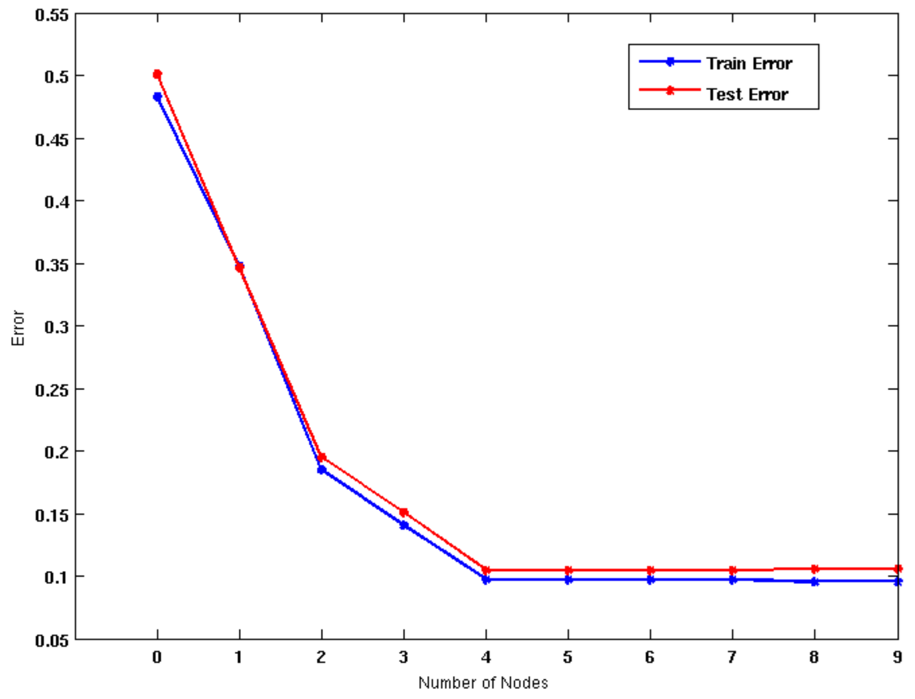
Decision Tree with 50 nodes

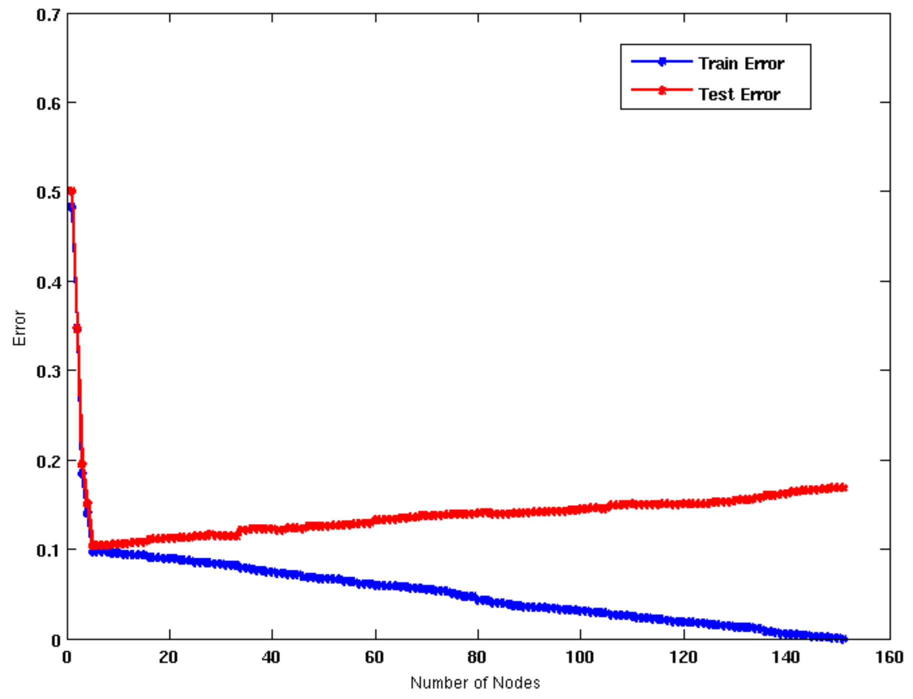


Which tree is better?



Generalization





When model is too simple, both training and test errors are large: Underfitting

When model is too complex, training error is small but test error is large: Overfitting

Bias – Variance tradeoff

- Bias: Measures how good the model is with respect to the training data
  - High Bias: Underfitting.
  - We have a poor model, for example, a tree with a single decision node, or a linear function when modeling a more complex curve

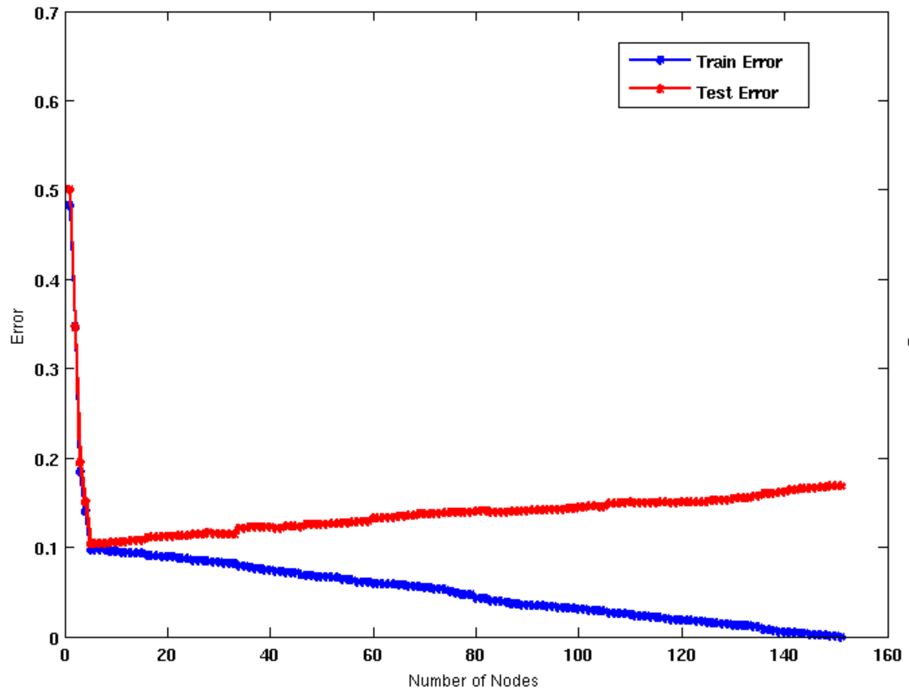
- Variance: Measures how sensitive the model error is with respect to changes in the training data
  - High Variance: Overfitting.
  - We have a very complex model, for example, a tree with a single sample per leaf, or a very high-degree polynomial curve. Small changes in the data cause errors in the model

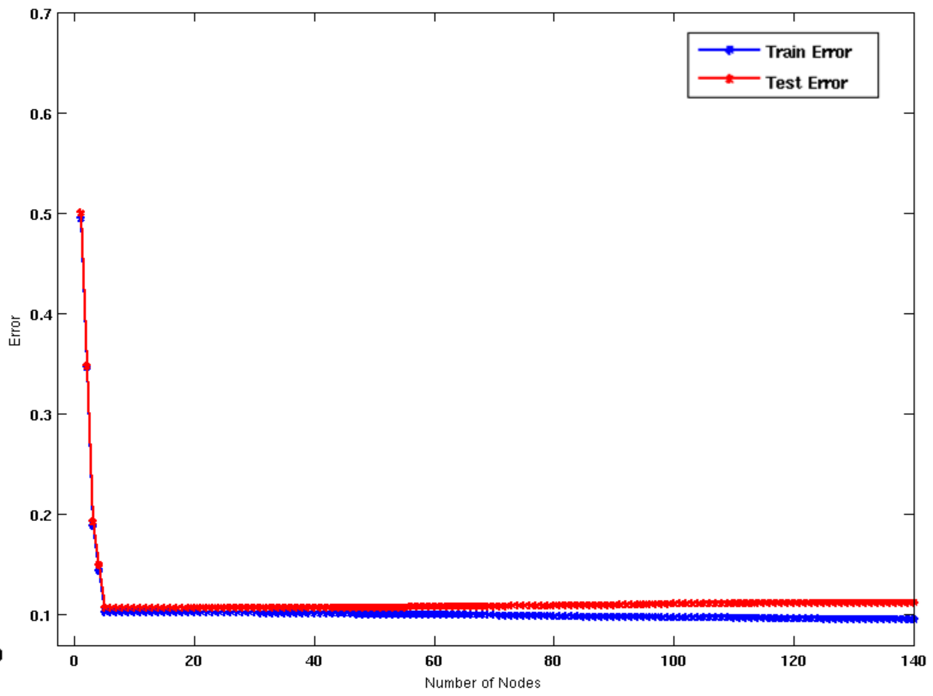
- There is a tradeoff between these two: decreasing one will increase the other.

Reasons for poor Generalization

- Not enough training data
- Not representative training data
- Erroneus training data
- A model that is too complex for the data we have.
  - Multiple Comparison Procedure

Model Overfitting

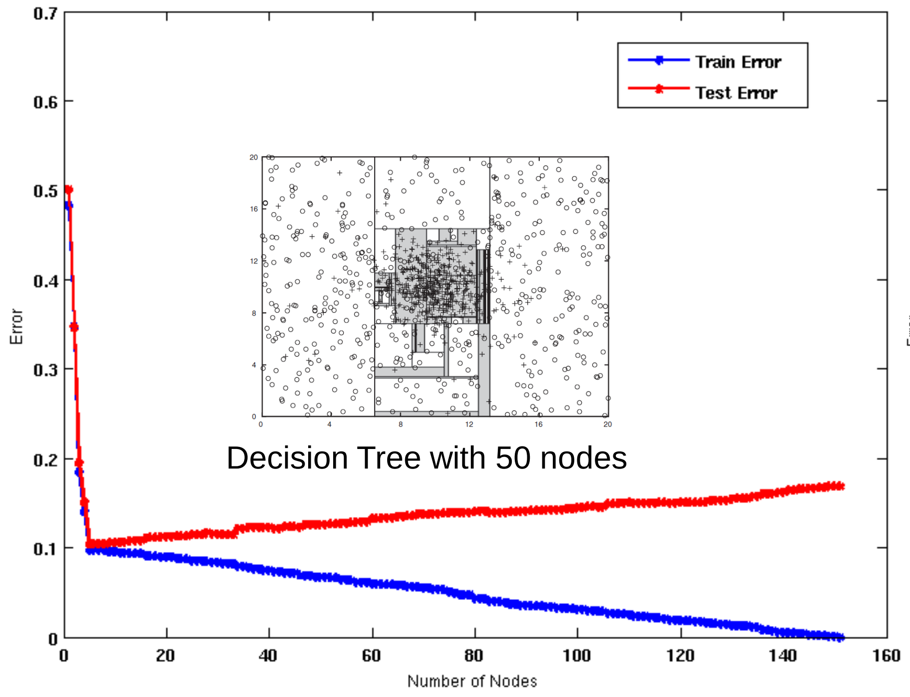


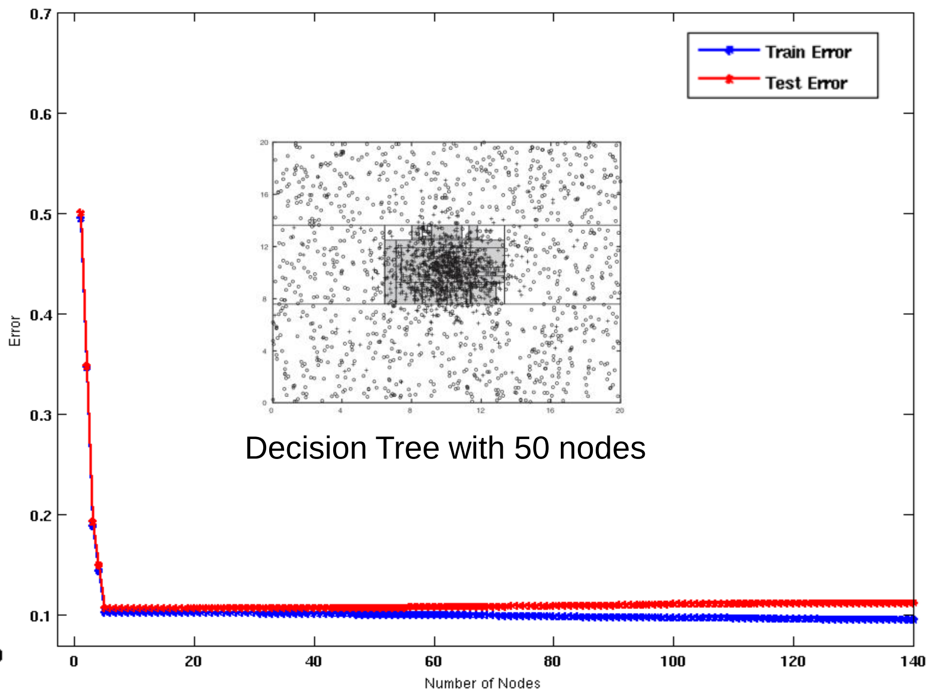


Using twice the number of data instances

- Increasing the size of training data reduces the difference between training and testing errors at a given size of model

Model Overfitting





Using twice the number of data instances

- Increasing the size of training data reduces the difference between training and testing errors at a given size of model

Effect of Multiple Comparison Procedure

- Consider the task of predicting whether stock market will rise/fall in the next 10 trading days

- Random guessing:

  $P(correct) = 0.5$

- Make 10 random guesses in a row:

$$P(\#\,correct \geq 8) = \frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0547$$

Day 1	Up
Day 2	Down
Day 3	Down
Day 4	Up
Day 5	Down
Day 6	Down
Day 7	Up
Day 8	Up
Day 9	Up
Day 10	Down

Effect of Multiple Comparison Procedure

- Approach:
  - Get 50 analysts
  - Each analyst makes 10 random guesses
  - Choose the analyst that makes the most number of correct predictions

- Probability that at least one analyst makes at least 8 correct predictions

$$P(\# correct \geq 8) = 1 - (1 - 0.0547)^{50} = 0.9399$$

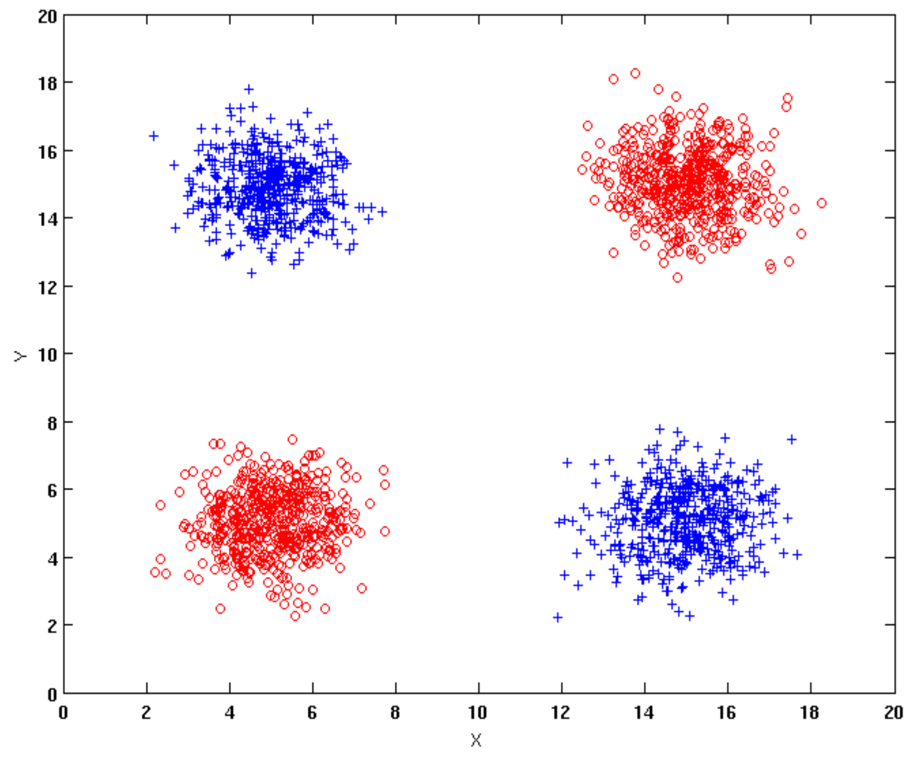
Effect of Multiple Comparison Procedure

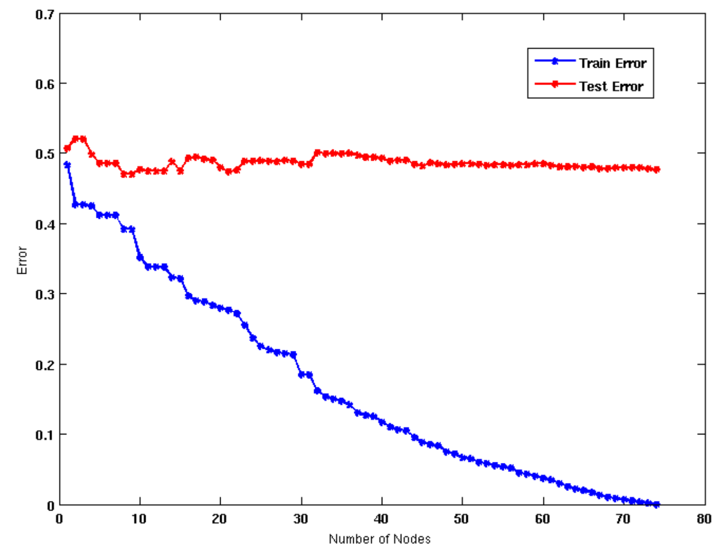
- Many algorithms employ the following greedy strategy:
  - Initial model: M
  - Alternative model: $M' = M + \gamma$,
    where γ is a component to be added to the model (e.g., a test condition of a decision tree)
  - Keep M' if improvement, $\Delta(M, M') > \alpha$

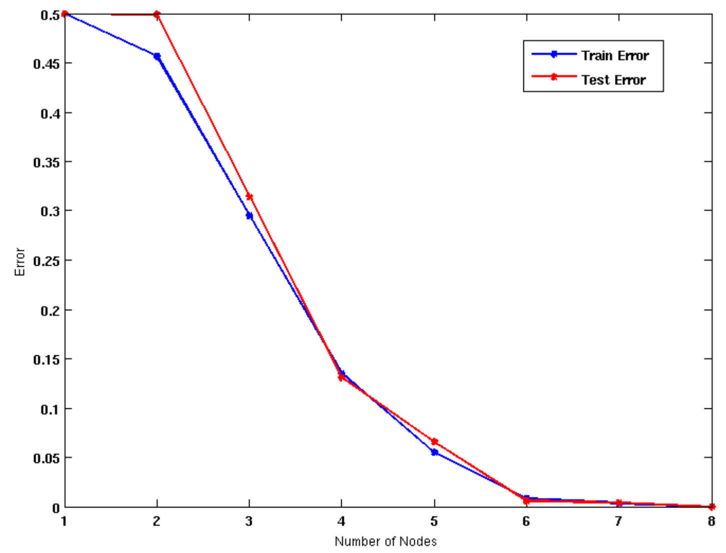
- Often times, γ is chosen from a set of alternative components, $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$

- If many alternatives are available, one may inadvertently add irrelevant components to the model, resulting in model overfitting

Effect of Multiple Comparison - Example







Using only X and Y as attributes

Use additional 100 noisy variables generated from a uniform distribution along with X and Y as attributes.

Use 30% of the data for training and 70% of the data for testing

In summary

- When we have a small number of samples, and a very large number of features then it is likely that we will create a model that overfits the data and does not generalize well.

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error no longer provides a good estimate of test error, that is, how well the tree will perform on previously unseen records
- We say that the model does not generalize well

- Need ways for estimating the generalization error and select the model.

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity

Model Selection: Using Validation Set

- Divide training data into two parts:
  - Training set:
    - Use for model building
  - Validation set:
    - Use for estimating generalization error
    - Note: validation set is not the same as test set since it affects the creation of the model (e.g. in tuning the size of the decision tree)

- Drawback:
  - Less data available for training

Model Selection: Incorporating Model Complexity

- Occam’s razor: All other things being equal, the simplest explanation/solution is the best.
  - A good principle for life as well

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model

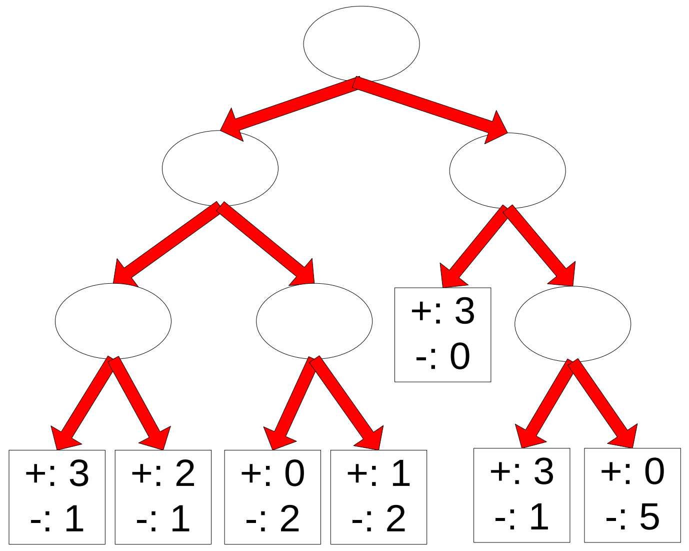
- For complex models, there is a greater chance that it was fitted accidentally

- Therefore, one should include model complexity when evaluating a model

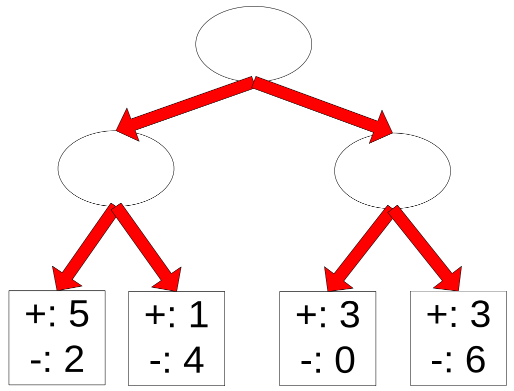
$$\boxed{\begin{aligned}\text{Gen. Error(Model)} = {} & \text{Train. Error(Model, Train. Data)} \\ & + \alpha \times \text{Complexity(Model)}\end{aligned}}$$

Estimating the Complexity of Decision Trees

- Resubstitution Estimate:
  - Using training error as an optimistic estimate of generalization error
  - Referred to as optimistic error estimate



Decision Tree, T_L



Decision Tree, T_R

$$e(TL) = 4/24$$

$$e(TR) = 6/24$$

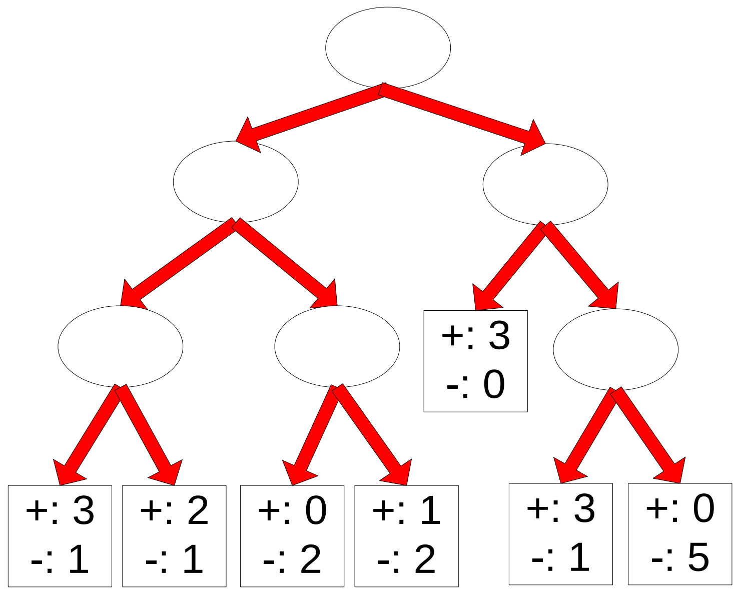
Estimating the Complexity of Decision Trees

- Pessimistic Error Estimate of decision tree T with k leaf nodes:

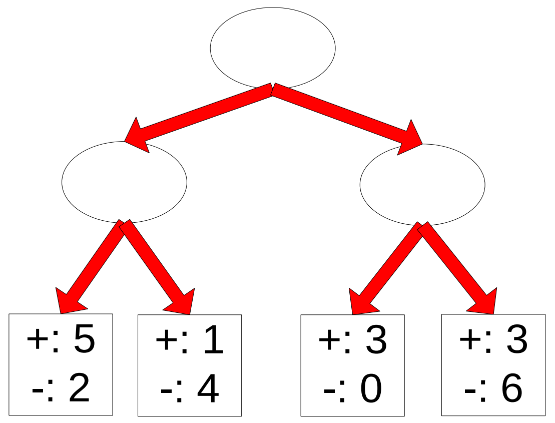
$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

  - $err(T)$: error rate on all training records
  - Ω: trade-off hyper-parameter (similar to α)
    - Relative cost of adding a leaf node
  - k: number of leaf nodes
  - N_{train}: total number of training records

Estimating the Complexity of Decision Trees: Example



Decision Tree, T_L



Decision Tree, T_R

$$e(TL) = 4/24$$

$$e(TR) = 6/24$$

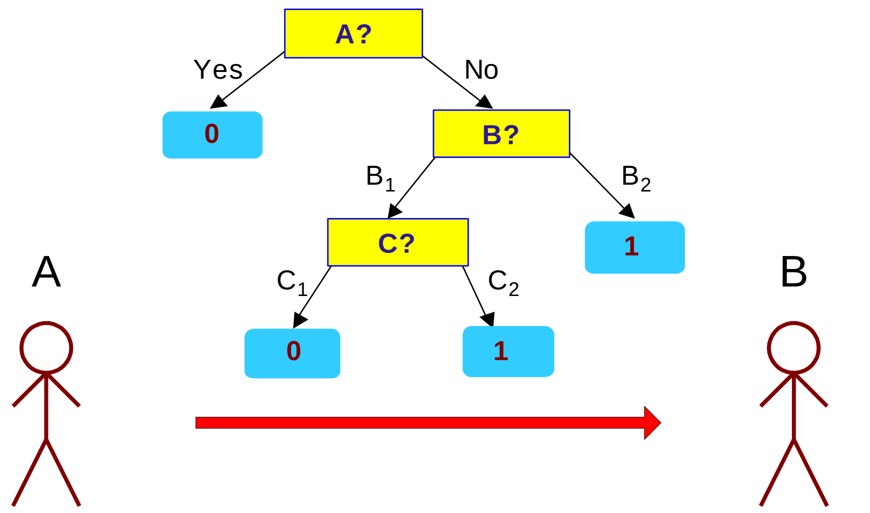
$$\Omega = 1$$

$$e_{gen}(TL) = 4/24 + 1 * 7/24 = 11/24 = 0.458$$

$$e_{gen}(TR) = 6/24 + 1 * 4/24 = 10/24 = 0.417$$

Minimum Description Length (MDL)

X	y
X_1	1
X_2	0
X_3	0
X_4	1
…	…
X_n	1



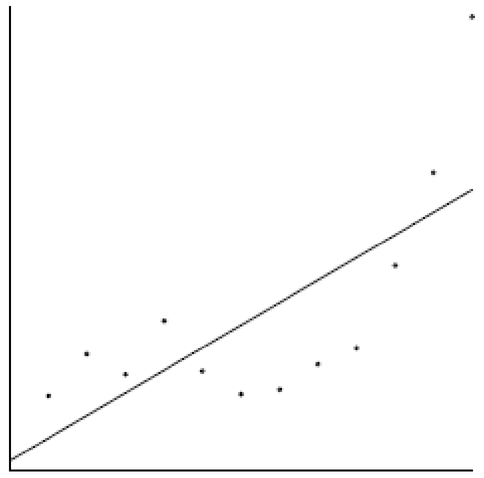
X	y
X_1	?
X_2	?
X_3	?
X_4	?
…	…
X_n	?

- Cost(Model,Data) = Cost(Model) + Cost(Data|Model)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.

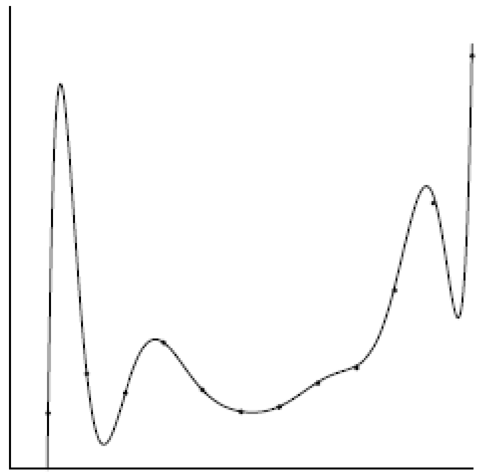
- Cost(Model) encodes the decision tree
  - node encoding (number of children) plus splitting condition encoding.
- Cost(Data|Model) encodes the misclassification errors.

Example

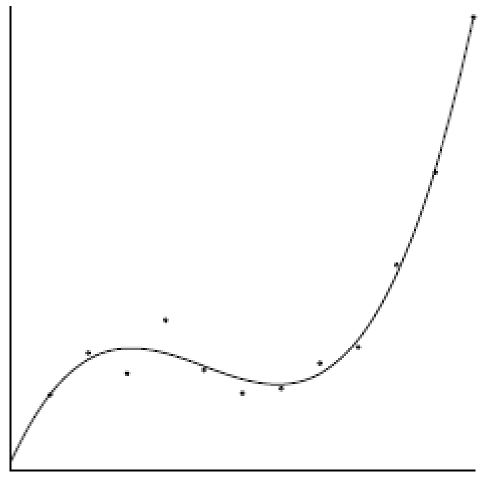
- Regression: find a polynomial for describing a set of values
  - Model complexity (model cost): polynomial coefficients
  - Goodness of fit (data cost): difference between real value and the polynomial value



Minimum model cost
High data cost



High model cost
Minimum data cost



Low model cost
Low data cost

MDL avoids overfitting automatically!

Source: Grunwald et al. (2005) *Tutorial on MDL*.

Model selection for Decision Trees

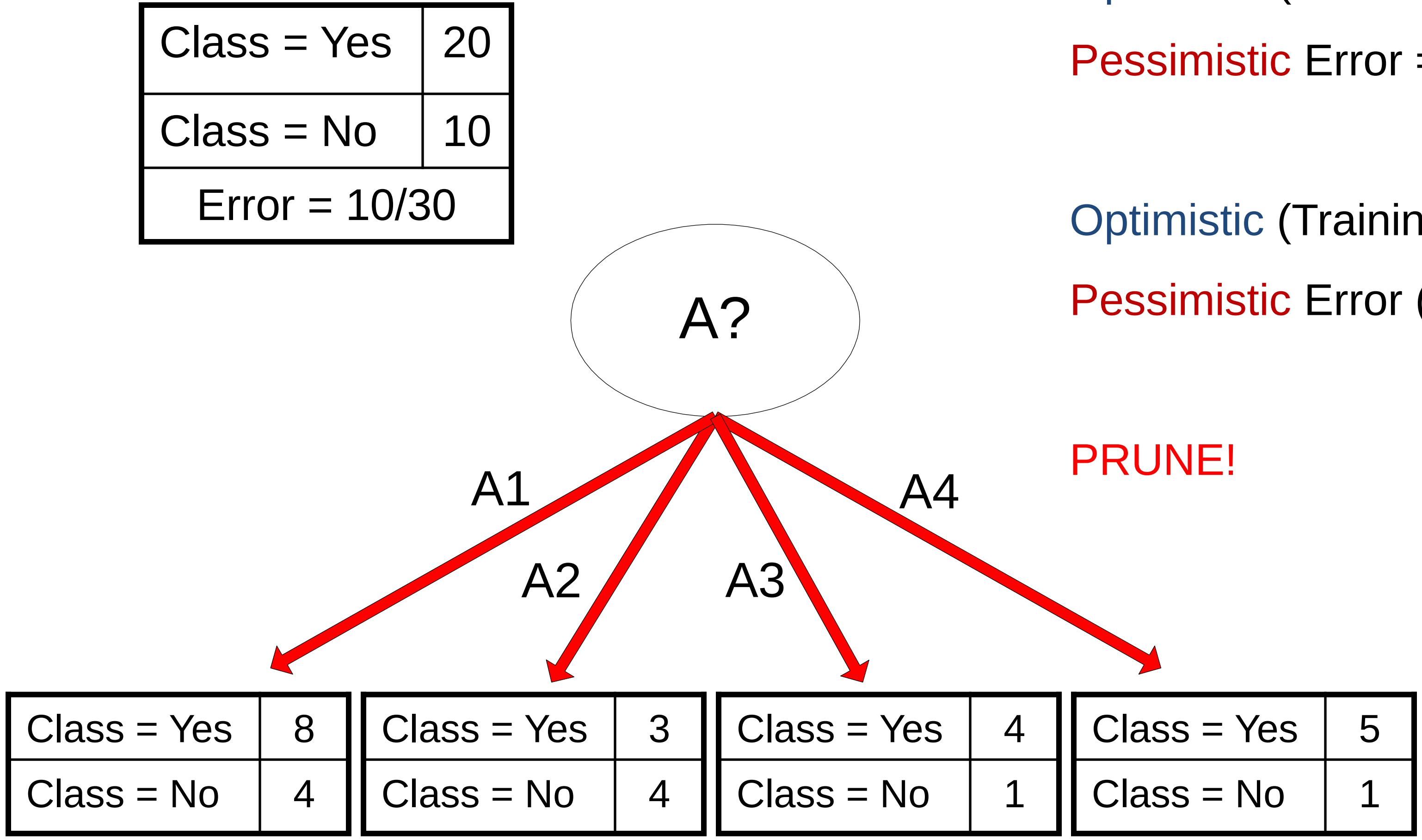
- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instance classes are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

Model selection for Decision Trees

- Post-pruning
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - If generalization error improves after trimming, replace sub-tree by a leaf node (subtree pruning) or by the most probable subtree (subtree raising).
  - Class label of leaf node is determined from majority class of instances in the sub-tree

  - Can use MDL for post-pruning
    - NP hard problem

Example of Post-Pruning



Optimistic (Training) Error (Before splitting) = 10/30

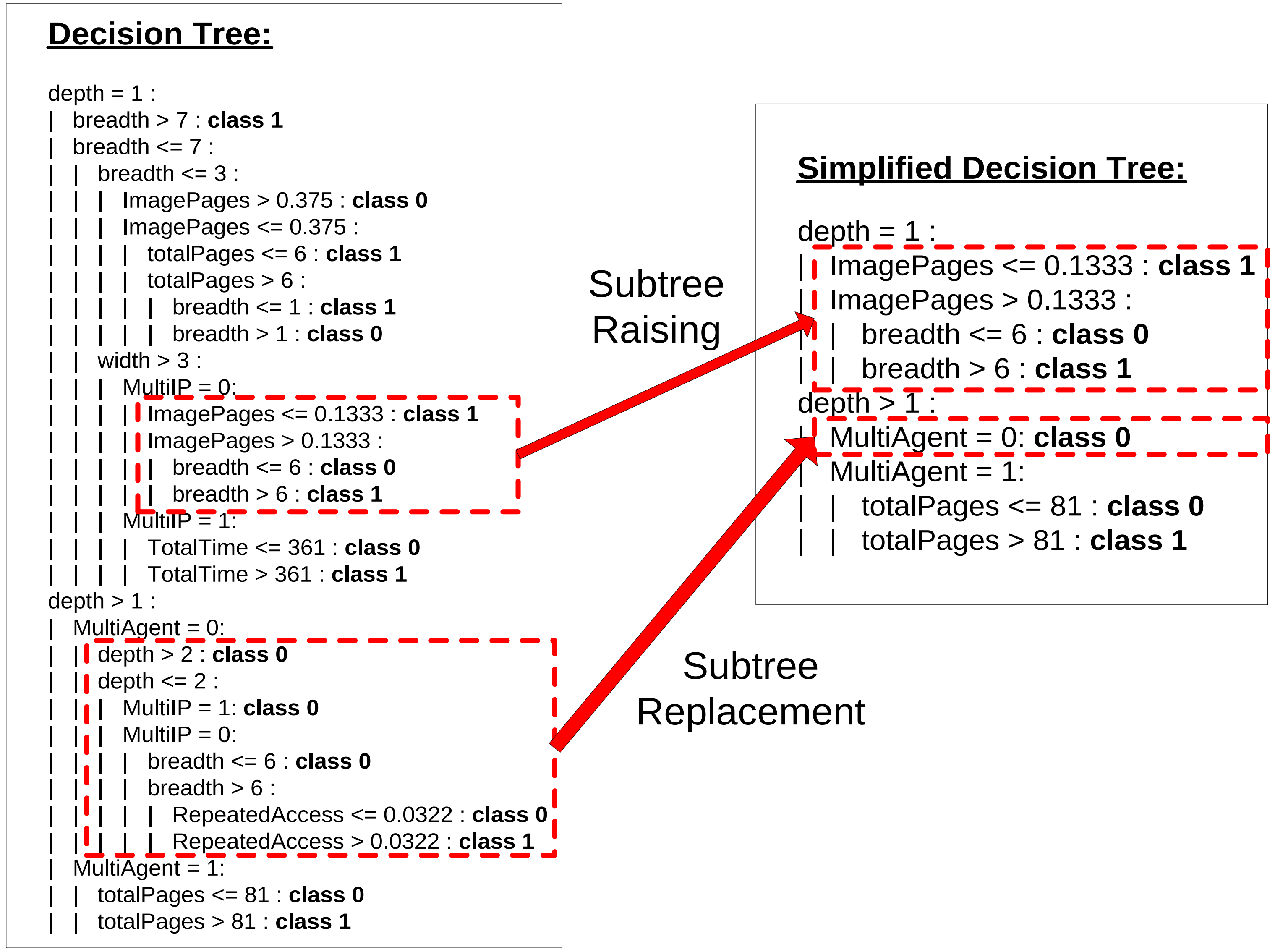
Pessimistic Error = (10 + 0.5)/30 = 10.5/30

Optimistic (Training) Error (After splitting) = 9/30

Pessimistic Error (After splitting) = (9 + 4 $\times$ 0.5)/30 = 11/30

PRUNE!

Examples of Post-pruning



Decision Tree:

```
depth = 1 :
|   breadth > 7 : class 1
|   breadth <= 7 :
|   |   breadth <= 3 :
|   |   |   ImagePages > 0.375 : class 0
|   |   |   ImagePages <= 0.375 :
|   |   |   |   totalPages <= 6 : class 1
|   |   |   |   totalPages > 6 :
|   |   |   |   |   breadth <= 1 : class 1
|   |   |   |   |   breadth > 1 : class 0
|   |   width > 3 :
|   |   |   MultiIP = 0:
|   |   |   |   ImagePages <= 0.1333 : class 1
|   |   |   |   ImagePages > 0.1333 :
|   |   |   |   |   breadth <= 6 : class 0
|   |   |   |   |   breadth > 6 : class 1
|   |   |   MultiIP = 1:
|   |   |   |   TotalTime <= 361 : class 0
|   |   |   |   TotalTime > 361 : class 1
depth > 1 :
|   MultiAgent = 0:
|   |   depth > 2 : class 0
|   |   depth <= 2 :
|   |   |   MultiIP = 1: class 0
|   |   |   MultiIP = 0:
|   |   |   |   breadth <= 6 : class 0
|   |   |   |   breadth > 6 :
|   |   |   |   |   RepeatedAccess <= 0.0322 : class 0
|   |   |   |   |   RepeatedAccess > 0.0322 : class 1
|   MultiAgent = 1:
|   |   totalPages <= 81 : class 0
|   |   totalPages > 81 : class 1
```

Simplified Decision Tree:

```
depth = 1 :
|   ImagePages <= 0.1333 : class 1
|   ImagePages > 0.1333 :
|   |   breadth <= 6 : class 0
|   |   breadth > 6 : class 1
depth > 1 :
|   MultiAgent = 0: class 0
|   MultiAgent = 1:
|   |   totalPages <= 81 : class 0
|   |   totalPages > 81 : class 1
```

EVALUATION

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	**a**	**b**
	Class=No	**c**	**d**

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Evaluation…

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision-Recall

Count	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a	b
	Class=No	c	d

$$\text{Precision (p)} = \frac{a}{a+c} = \frac{TP}{TP+FP}$$

$$\text{Recall (r)} = \frac{a}{a+b} = \frac{TP}{TP+FN}$$

$$\text{F-measure (F)} = \frac{1}{\left(\frac{1/r+1/p}{2}\right)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c} = \frac{2TP}{2TP+FP+FN}$$

Assumption: The class YES is the one we care about.

- Precision is biased towards **C(Yes|Yes) & C(Yes|No)**
- Recall is biased towards **C(Yes|Yes) & C(No|Yes)**
- F-measure is biased towards all except **C(No|No)**

More Measures of Classification Performance

	PREDICTED CLASS		
ACTUAL CLASS		Yes	No
	Yes	TP	FN
	No	FP	TN

α is the probability that we reject the null hypothesis when it is true.

This is a Type I error or a false positive (FP).

β is the probability that we accept the null hypothesis when it is false.

This is a Type II error or a false negative (FN).

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$ErrorRate = 1 - accuracy$$

$$Precision = Positive\ Predictive\ Value = \frac{TP}{TP + FP}$$

$$Recall = Sensitivity = TP\ Rate = \frac{TP}{TP + FN}$$

$$Specificity = TN\ Rate = \frac{TN}{TN + FP}$$ (recall for negative class)

$$FP\ Rate = \boxed{\alpha} = \frac{FP}{TN + FP} = 1 - specificity$$

$$FN\ Rate = \boxed{\beta} = \frac{FN}{FN + TP} = 1 - sensitivity$$

$$Power = sensitivity = 1 - \beta$$

ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms
- ROC curve plots TPR (true positive rate) (on the y-axis) against FPR (false positive rate) (on the x-axis)

Look at the positive predictions of the classifier and compute:

$$TPR = \frac{TP}{TP + FN}$$

What fraction of true positive instances are predicted correctly? (1-Type II error rate)

$$FPR = \frac{FP}{FP + TN}$$

	PREDICTED CLASS		
		Yes	No
Actual	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

What fraction of true negative instances were predicted incorrectly? (Type I error rate)

We want to strike a balance between these two

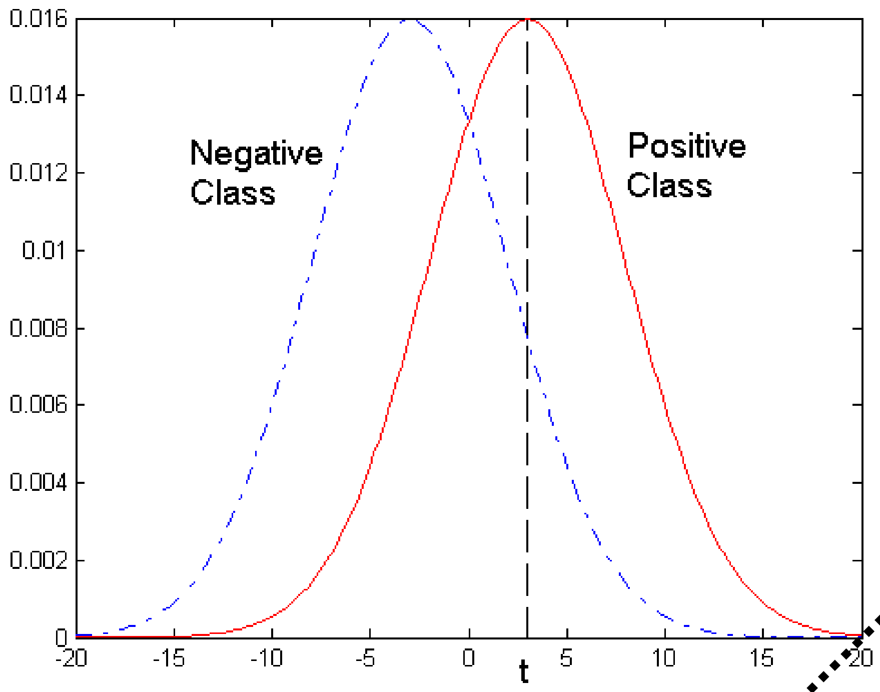
ROC (Receiver Operating Characteristic)

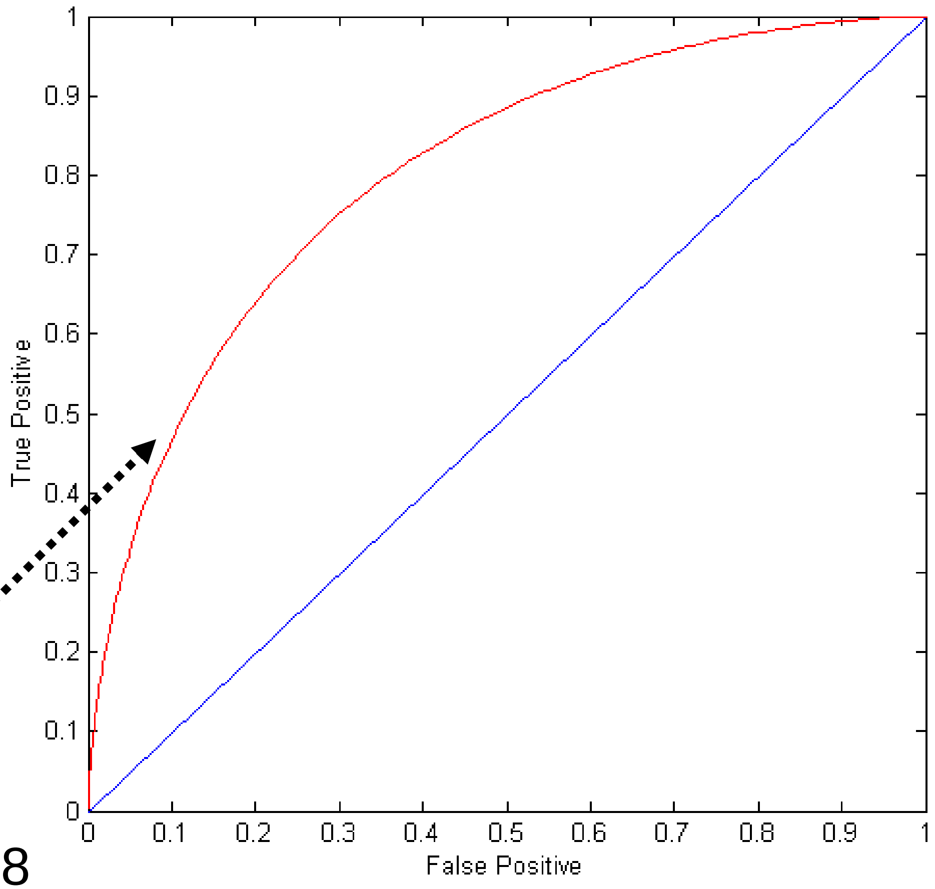
- Performance of a classifier represented as a point on the **ROC** curve
- Changing some parameter of the algorithm, sample distribution, or cost matrix changes the location of the point

ROC Curve

- **1**-dimensional data set containing **2** classes (***positive*** and ***negative***)

- any points located at **x > t** is classified as ***positive***





At threshold **t**:

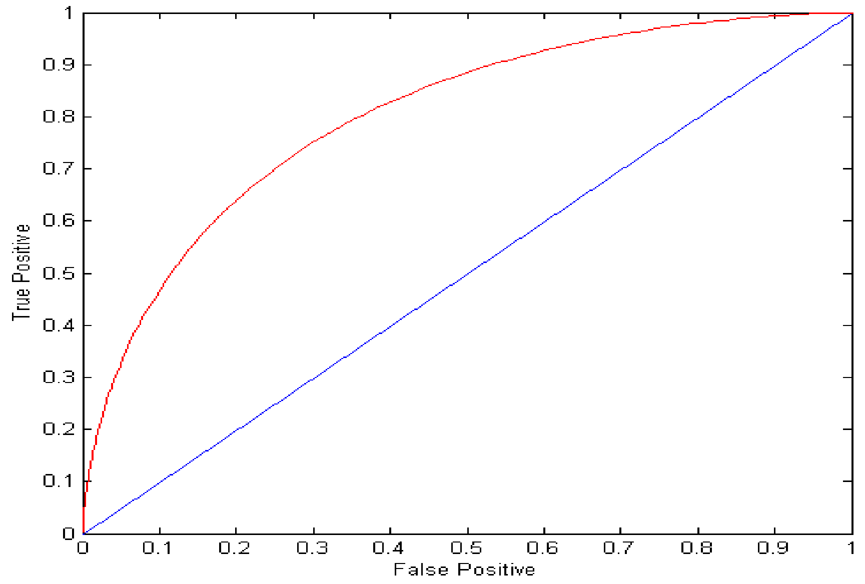
TP=0.5, FN=0.5, FP=0.12, FN=0.88

ROC Curve

(TP,FP):

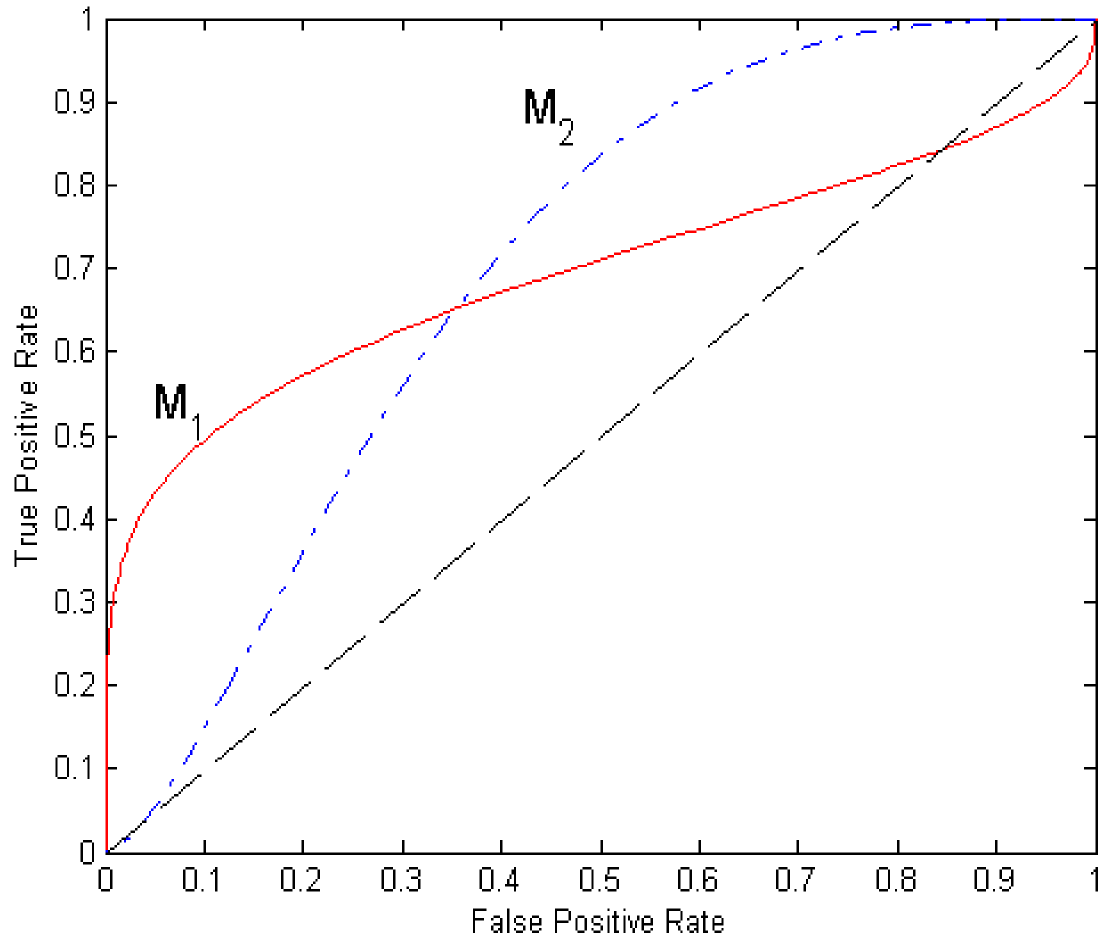
- (0,0): declare everything
  to be negative class
- (1,1): declare everything
  to be positive class
- (1,0): ideal

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



	PREDICTED CLASS		
		Yes	No
Actual	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

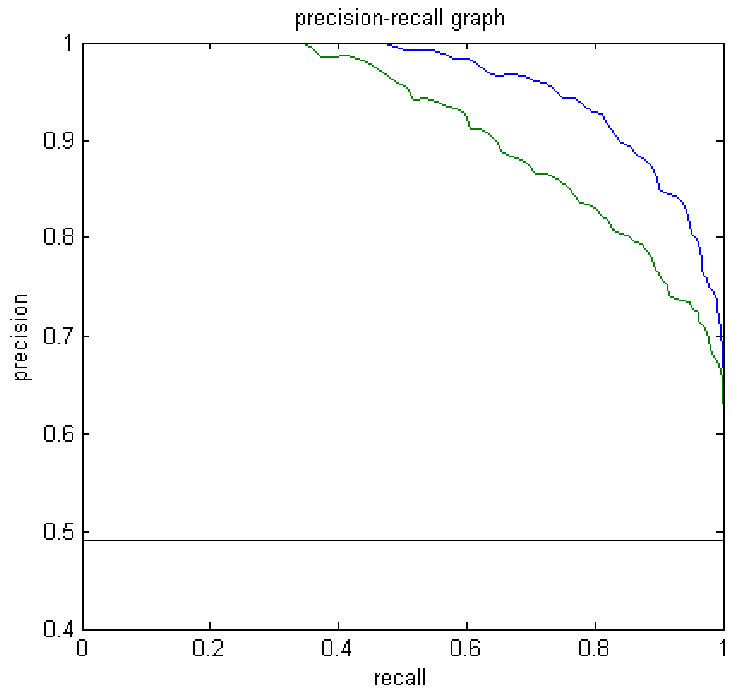
Using ROC for Model Comparison



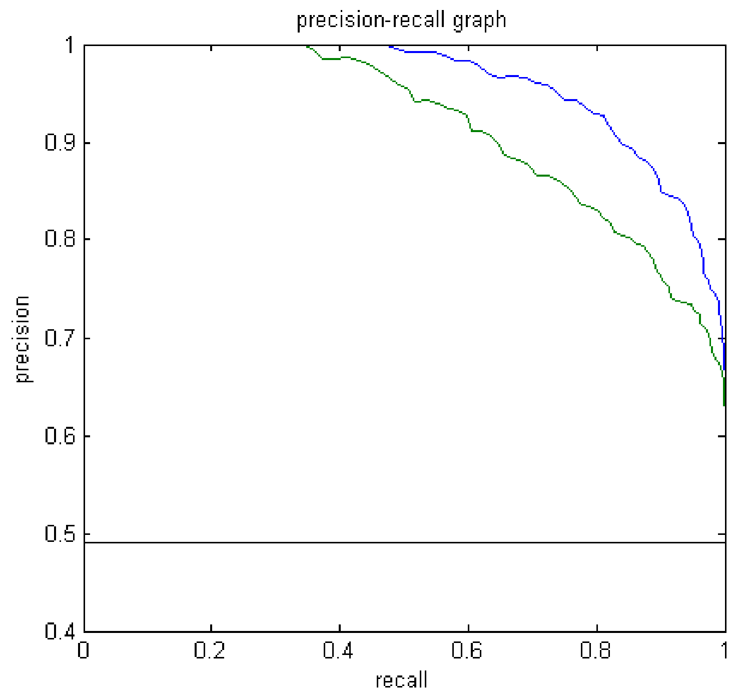
- No model consistently outperform the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve (AUC)
  - Ideal: Area = 1
  - Random guess:
    - Area = 0.5

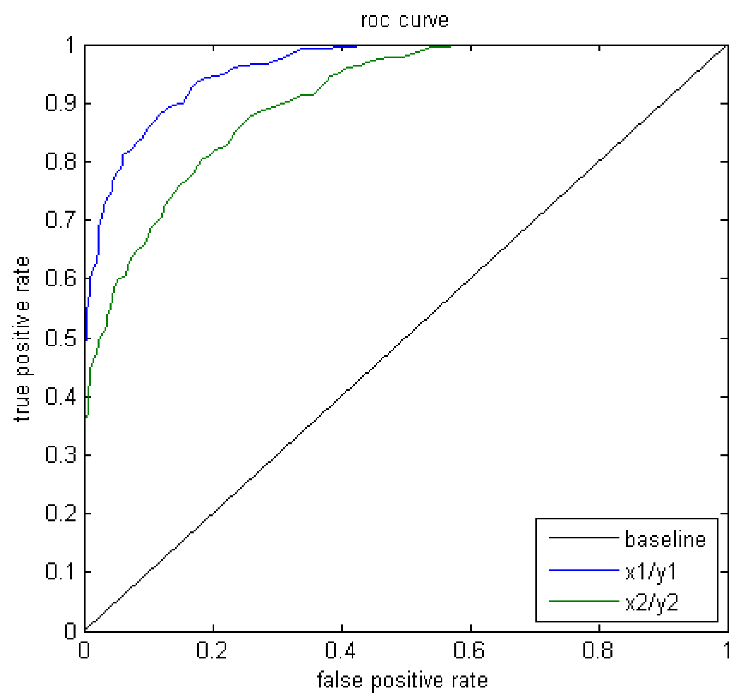
Precision-Recall plot

- Usually for parameterized models, it controls the precision/recall tradeoff



ROC curve vs Precision-Recall curve





Area Under the Curve (AUC) as a single number for evaluation

Methods of Performance Estimation

- Holdout
  - Reserve **2/3** for training and **1/3** for testing
- Random subsampling
  - One sample may be biased -- Repeated holdout
- Cross validation
  - Partition data into **k** disjoint subsets
  - **k**-fold: train on **k-1** partitions, test on the remaining one
  - Leave-one-out: **k=n**
  - Guarantees that each record is used the same number of times for training and testing
- Bootstrap
  - Sampling with replacement
  - ~63% of records used for training, ~27% for testing

Class imbalance

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example
  - Precision and recall are better measures

Dealing with class Imbalance

- Class imbalance is a problem in training:
  - If the class we are interested in is very rare, then the classifier will ignore it.
- Solution
  - We can balance the class distribution
    - Sample from the larger class so that the size of the two classes is the same
    - Replicate the data of the class of interest so that the classes are balanced
      - Over-fitting issues
  - We can modify the optimization criterion by using a cost sensitive metric

Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of classifying class **j** example as class **i**

Weighted Accuracy

CONFUSION MATRIX	PREDICTED CLASS		
		Class=Yes	**Class=No**
ACTUAL CLASS	**Class=Yes**	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

COST MATRIX	PREDICTED CLASS		
	C(i|j)	**Class=Yes**	**Class=No**
ACTUAL CLASS	**Class=Yes**	w_1 C(Yes|Yes)	w_2 C(No|Yes)
	Class=No	w_3 C(Yes|No)	w_4 C(No|No)

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i|j)	+	-
ACTUAL CLASS	+	1	100
	-	1	1

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%

Weighted Accuracy = 8.9%

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%

Weighted Accuracy= 9%

Classification Cost

CONFUSION MATRIX	PREDICTED CLASS		
ACTUAL CLASS		**Class=Yes**	**Class=No**
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

COST MATRIX	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	w_1 C(Yes|Yes)	w_2 C(No|Yes)
	Class=No	w_3 C(Yes|No)	w_4 C(No|No)

Classification Cost = $w_1 a + w_2 b + w_3 c + w_4 d$

Some weights can also be negative

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	+	-
	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

Cost vs Accuracy

Count		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

Cost		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	p	q
	Class=No	q	p

Accuracy is proportional to cost if

1. C(Yes|No)=C(No|Yes) = q
2. C(Yes|Yes)=C(No|No) = p

N = a + b + c + d

Accuracy = (a + d)/N

$$\begin{aligned}\text{Cost} &= p\,(a + d) + q\,(b + c) \\ &= p\,(a + d) + q\,(N - a - d) \\ &= q\,N - (q - p)(a + d) \\ &= N\,[q - (q-p) \times \text{Accuracy}]\end{aligned}$$

SUPERVISED LEARNING

Learning

- Supervised Learning: learn a model from the data using labeled data.
  - Classification and Regression are the prototypical examples of supervised learning tasks. Other are possible (e.g., ranking)
- Unsupervised Learning: learn a model – extract structure from unlabeled data.
  - Clustering and Association Rules are prototypical examples of unsupervised learning tasks.
- Semi-supervised Learning: learn a model for the data using both labeled and unlabeled data.

Supervised Learning Steps

- Model the problem
  - What is you are trying to predict? What kind of optimization function do you need? Do you need classes or probabilities?
- Extract Features
  - How do you find the right features that help to discriminate between the classes?
- Obtain labeled data
  - Obtain a collection of labeled data. Make sure it is large enough, accurate and representative. Ensure that classes are well represented.
- Decide on the technique
  - What is the right technique for your problem?
- Apply in practice
  - Can the model be trained for very large data? How do you test how you do in practice? How do you improve?

Modeling the problem

- Sometimes it is not obvious. Consider the following three problems
  - Detecting if an email is spam
  - Categorizing the queries in a search engine
  - Ranking the results of a web search
  - Predicting the reply to a question.

Feature extraction

- Feature extraction, or feature engineering is the most tedious but also the most important step
  - How do you separate the players of the Greek national team from those of the Swedish national team?

- One line of thought: throw features to the classifier and the classifier will figure out which ones are important
  - More features, means that you need more training data
- Another line of thought: Feature Selection: Select carefully the features using various functions and techniques
  - Computationally intensive

- Deep Neural Networks
  - They use raw data for classification
  - They learn a representation from the data

Training data

- An overlooked problem: How do you get labeled data for training your model?
  - E.g., how do you get training data for ranking web search results?
  - Chicken and egg problem
- Usually requires a lot of manual effort and domain expertise and carefully planned labeling
  - Results are not always of high quality (lack of expertise)
  - And they are not sufficient (low coverage of the space)
- Recent trends:
  - Find a source that generates the labeled data for you, or use the data themselves for the prediction task
  - Crowd-sourcing techniques

Dealing with small amounts of labeled data

- Semi-supervised learning techniques have been developed for this purpose.
- Self-training: Train a classifier on the data, and then feed back the high-confidence output of the classifier as input
- Co-training: train two “independent” classifiers and feed the output of one classifier as input to the other.
- Regularization: Treat learning as an optimization problem where you define relationships between the objects you want to classify, and you exploit these relationships
  - Example: Image restoration

Technique

- The choice of technique depends on the problem requirements (do we need a probability estimate?) and the problem specifics (does independence assumption hold? do we think classes are linearly separable?)
- For many cases finding the right technique may be trial and error
- For many cases the exact technique does not matter.

Big Data Trumps Better Algorithms

- If you have enough data then the algorithms are not so important

- The web has made this possible.
  - Especially for text-related tasks
  - Search engine uses the collective human intelligence

Google lecture: [Theorizing from the Data]

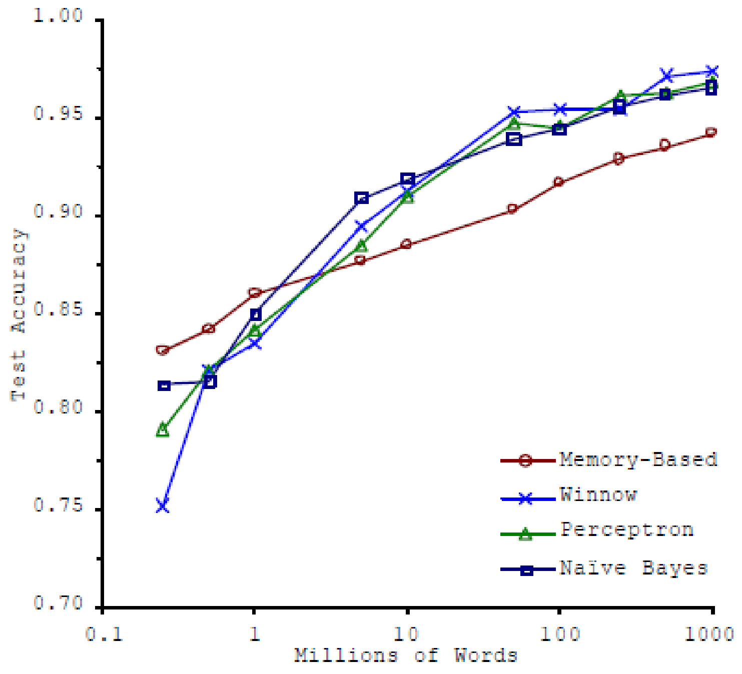


Figure 1. Learning Curves for Confusion Set Disambiguation

Apply-Test

- How do you scale to very large datasets?
  - Distributed computing – map-reduce implementations of machine learning algorithms (Mahaut, over Hadoop)

- How do you test something that is running online?
  - You cannot get labeled data in this case
  - A/B testing

- How do you deal with changes in data?
  - Active learning