

Natural Language Processing

Lecture 13:
Machine Learning: Linear and Log-Linear Models

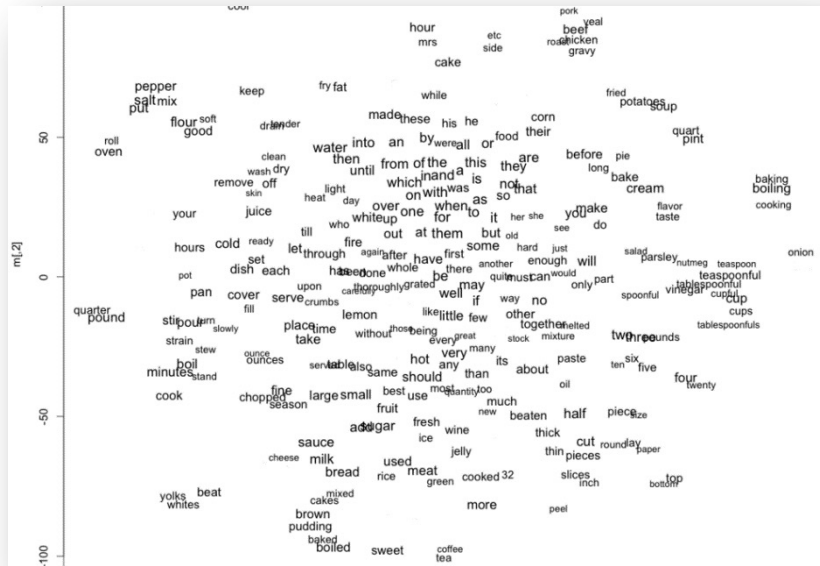
11/12/2021

COMS W4705
Yassine Benajiba

Intro

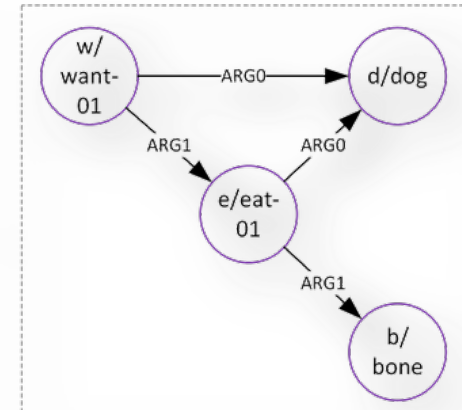


Intro



$\exists w, e, d, b$
 $\text{instance}(w, \text{want-01}) \wedge \text{instance}(d, \text{dog})$
 $\wedge \text{instance}(e, \text{eat}) \wedge \text{instance}(b, \text{bone})$
 $\wedge \text{arg0}(w, d) \wedge \text{arg1}(w, e)$
 $\wedge \text{arg0}(e, d) \wedge \text{arg1}(e, b)$

$(w / \text{want-01}$
 $\quad : \text{ARG0}(d / \text{dog})$
 $\quad : \text{ARG1}(e / \text{eat-01}$
 $\quad \quad : \text{ARG0}(d$
 $\quad \quad : \text{ARG1}(b / \text{bone}))$



Machine Learning and NLP

- We have encountered many different situations where we had to make a prediction:
 - Text classification, language modeling, POS tagging, constituency/dependency parsing,
 - These are all classification problems of some form.
- Today: Some machine learning background. Linear/log-linear models. Basic neural networks.

Generative Algorithms

- Assume the observed data is being “generated” by a “hidden” class label.
- Build a different model for each class.
- To predict a new example, check it under each of the models and see which one matches best.
- Model $P(x|y)$ and $P(y)$. Then use bases rule

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$$

Discriminative Algorithms

- Model conditional distribution of the label given the data

$$P(y|x)$$

- Learns decision boundaries that separate instances of the different classes.
- To predict a new example, check on which side of the decision boundary it falls.

Machine Learning Definition

- “Creating systems that improve from experience.”
- *“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 Mitchel, Machine Learning 1997)

Inductive Learning (a.k.a. *Science*)

- **Goal:** given a set of input/output pairs (training data), find the function $f(x)$ that maps inputs to outputs.
Problem: We did not see all possible inputs!
- Learn an approximate function $h(x)$ from the training data and hope that this function *generalize well* to unseen inputs.
- **Ockham's razor:** Choose the *simplest* hypothesis that is consistent with the training data.

Classification and Regression

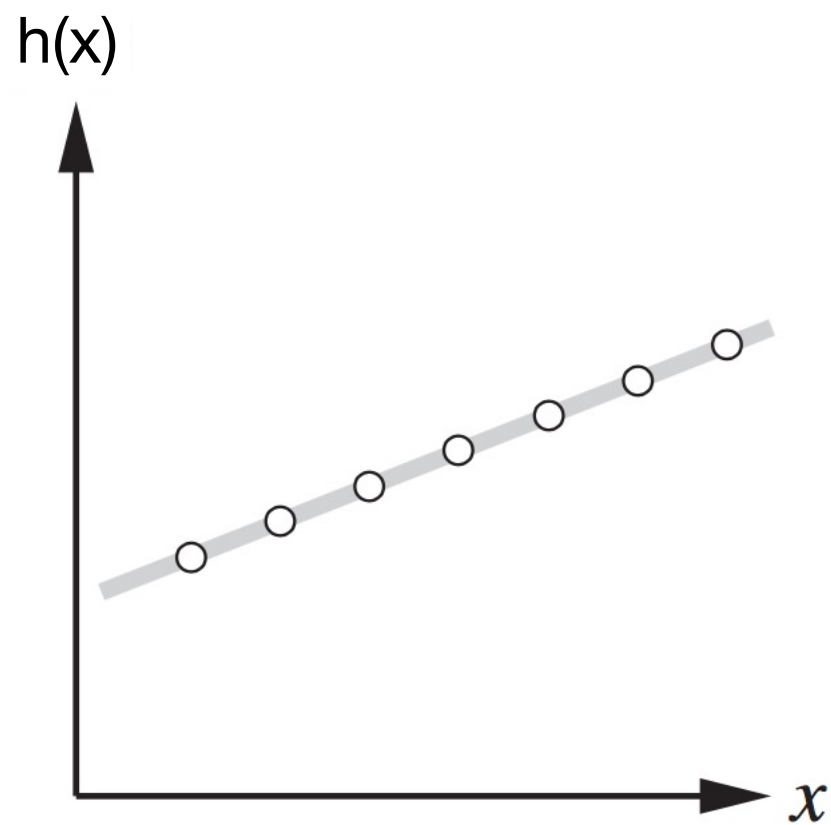
- Recall: In **supervised learning**, training data consisting of training examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, where \mathbf{x}_j is an input example (a d -dimensional vector of attribute values) and y_j is the label.
- Two types of supervised learning problems:
 - In classification: y_j is a finite, discrete set. Typically $y_j \in \{-1, +1\}$. i.e. predict a label from a set of labels. Learn a **classifier** function:

$$h : \mathbb{R}^d \rightarrow \{-1, +1\}$$

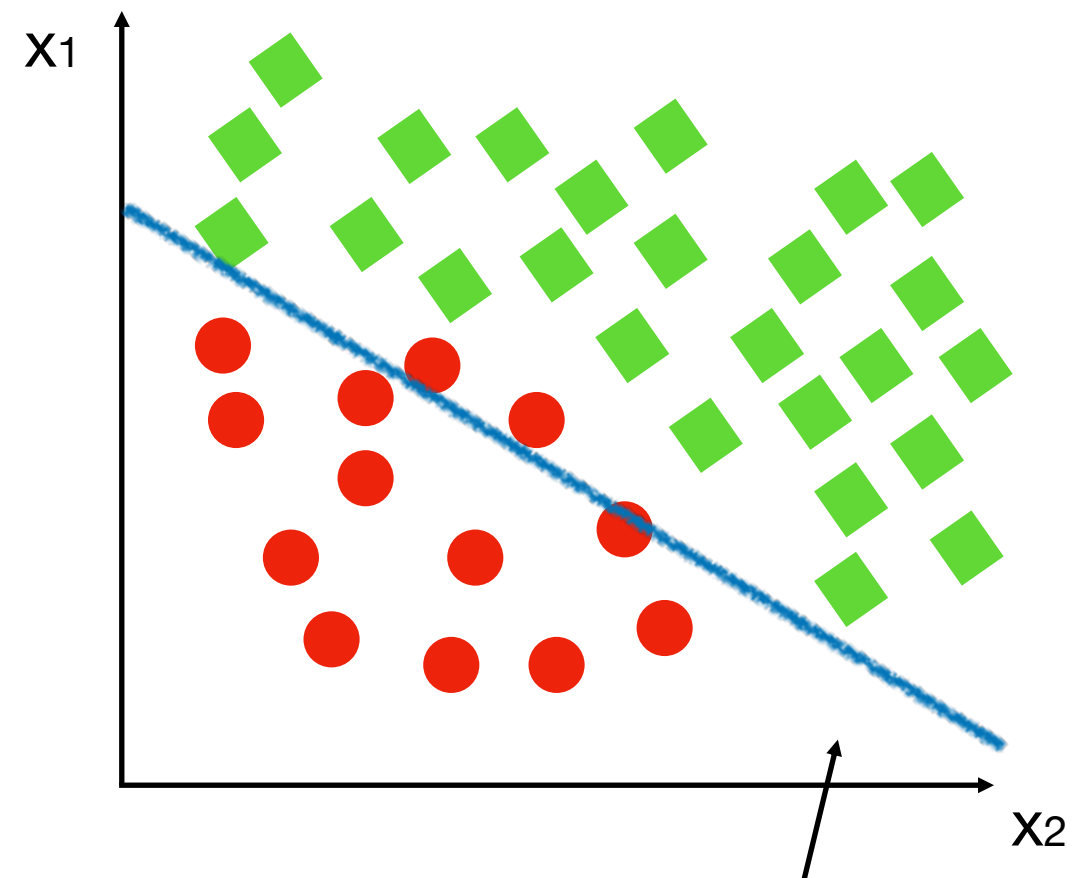
- In regression: $\mathbf{x}_j \in \mathbb{R}^d$, $y_i \in \mathbb{R}$. i.e. predict a numeric value. Learn a **regressor** function:

$$h : \mathbb{R}^d \rightarrow \mathbb{R}$$

Linear Classification and Regression



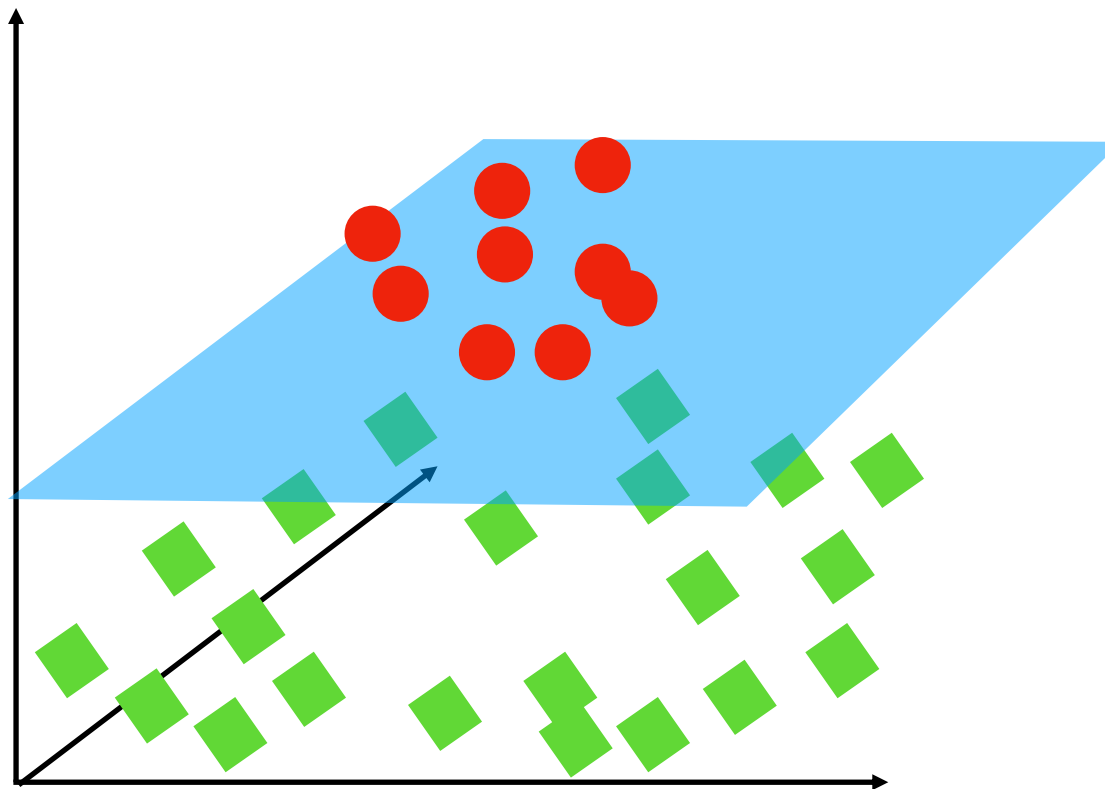
Regression



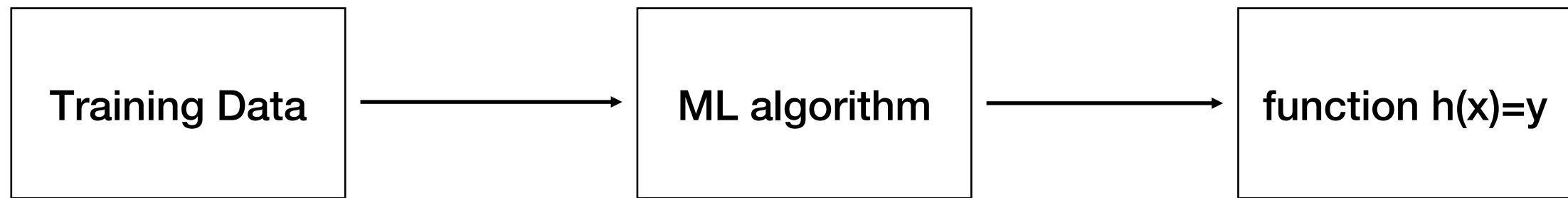
decision boundary

Classification

Linear Classification



Training ML models

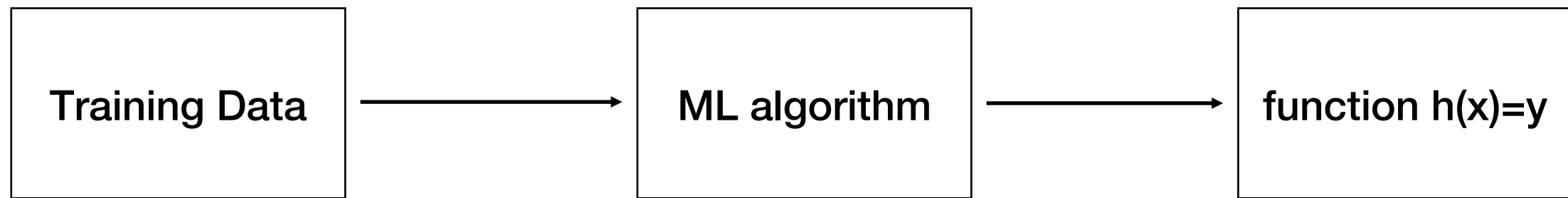


- How can we be confident about the learned function?
- Can compute empirical error/risk on the training set:

$$E_{train}(h) = \sum_{i=1}^n loss(y_i, h(x_i))$$

- Typical loss functions:
 - Least square loss (L2): $loss(y_i, h(x_i)) = (y_i - h(x_i))^2$
 - Classification error: $loss(y_i, h(x_i)) = \begin{cases} 1 & \text{if } sign(h(x_i)) \neq sign(y_i) \\ 0 & \text{otherwise.} \end{cases}$

Training ML models



- Empirical error/risk:

$$E_{train}(h) = \sum_{i=1}^n \text{loss}(y_i, h(x_i))$$

- Training aims to minimize E_{train} .
- We hope that this also minimizes E_{test} , the test error.

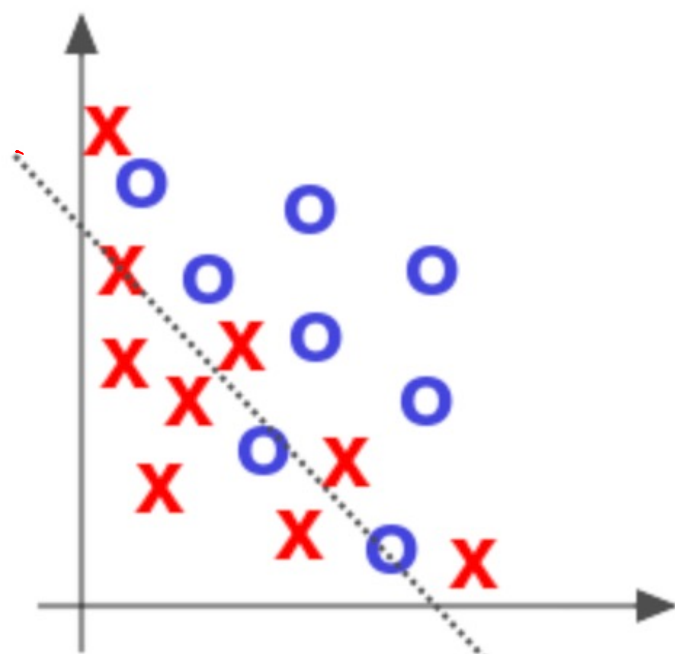
Overfitting

- Problem: Minimizing empirical risk can lead to **overfitting**.
- This happens when a model works well on the training data, but it does not **generalize** to testing data.
- Data sets can be noisy. Overfitting can model the noise in the data.

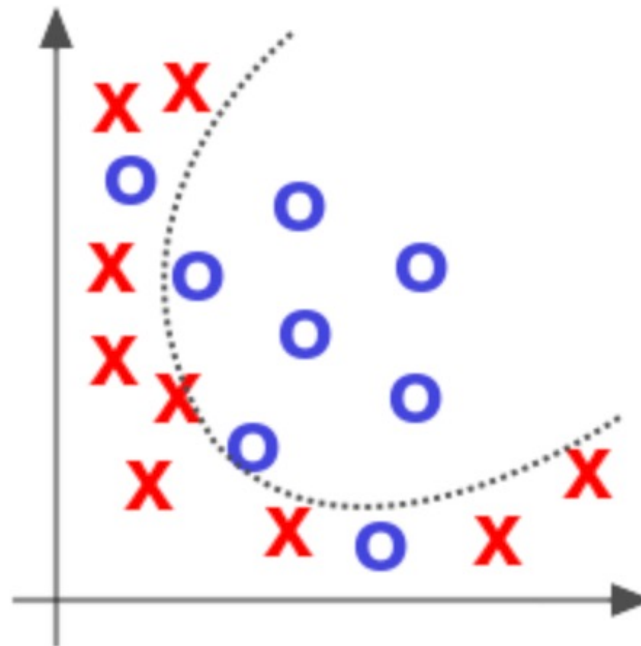
Preventing Overfitting

- Solutions: Simpler models.
 - Reduce the number of features (feature selection).
 - Model selection.
 - Regularization.
 - Cross validation.
- However: Adding wrong assumptions (bias) to the training algorithm can lead to **underfitting**!

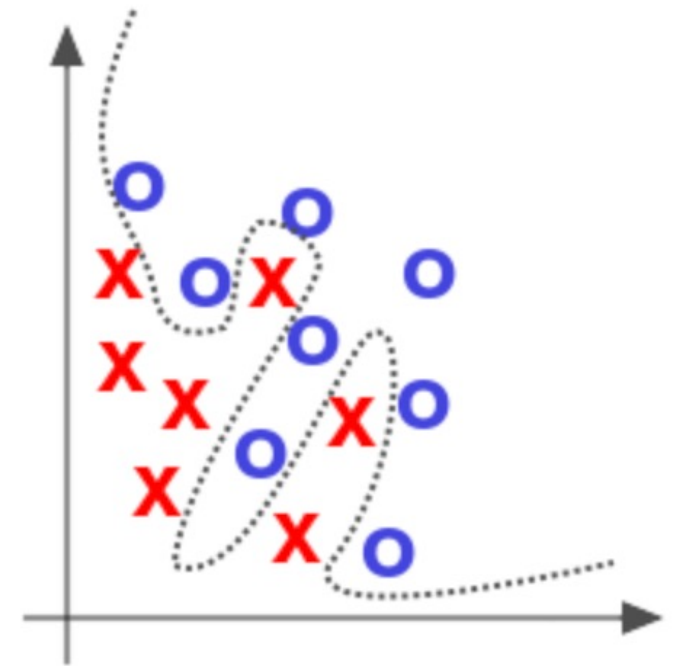
Goodness of Fit



Under Fit

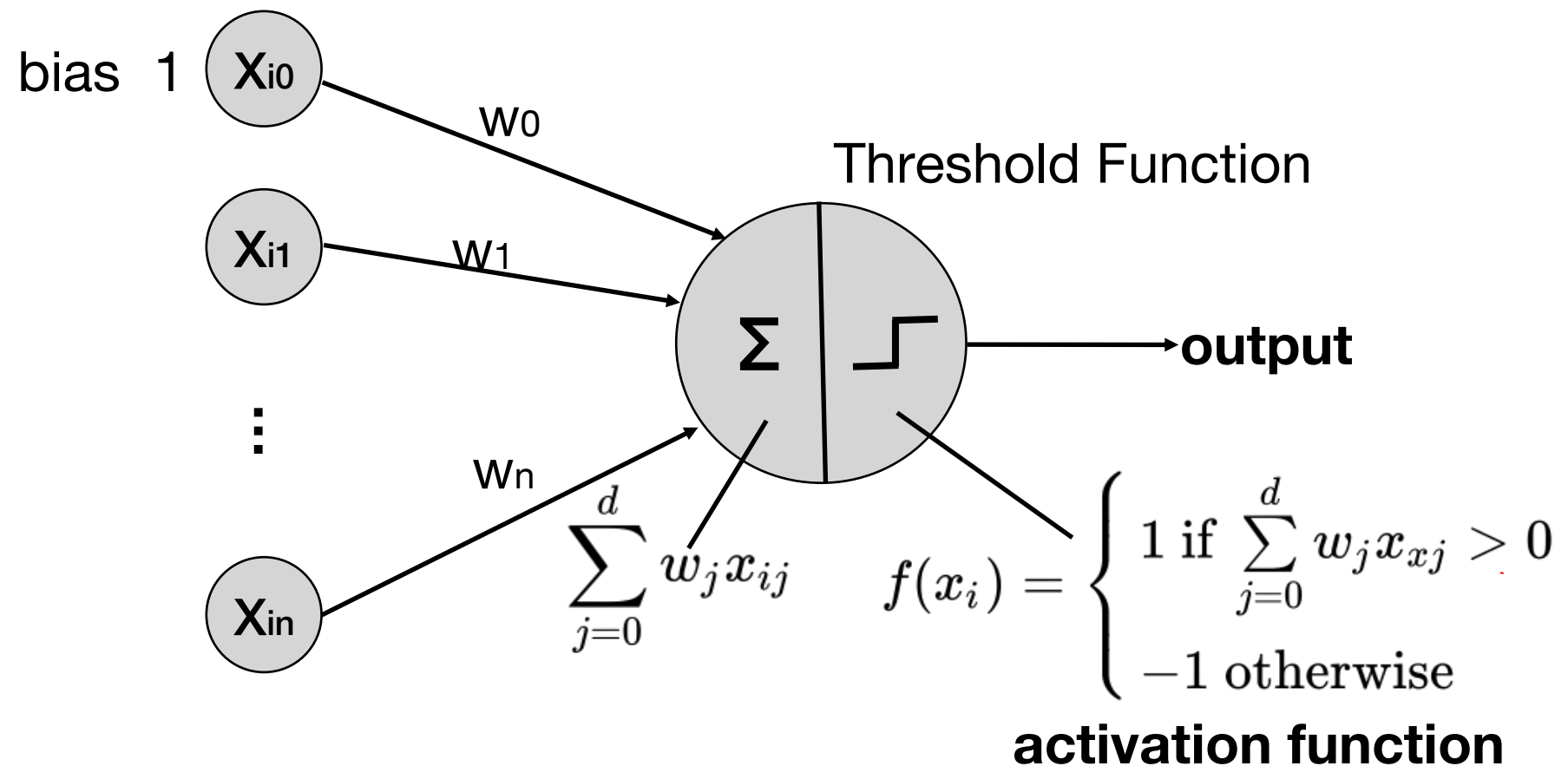


Appropriate



Over Fit

Linear Model



$$f(x_i) = \text{sign}\left(\sum_{j=0}^d w_j x_{ij}\right)$$

Linear Models

- We have chosen a function class (linear separators).
 - Specified by parameter \mathbf{w} .
- Need to estimate \mathbf{w} on the basis of the training set.
- What loss should we use? One option: minimize classification error:

$$loss(y_i, h(x_i)) = \begin{cases} 1 & \text{if } sign(h(x_i)) \neq sign(y_i) \\ 0 & \text{otherwise.} \end{cases}$$

Perceptron Learning

- Problem: Threshold function is not differentiable, so we cannot find a closed-form solution or apply gradient descent.
- Instead use iterative perceptron learning algorithm:
 - Start with arbitrary hyperplane.
 - Adjust it using the training data.
 - Update rule: $w_j \leftarrow w_j + (y - h_{\mathbf{w}}(\mathbf{x})) \times x_j$
- **Perceptron Convergence Theorem** states that any linear function can be learned using this algorithm in a finite number of iterations.

Perceptron Learning Algorithm

Input: Training examples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$

Output: A perceptron defined by (w_0, w_1, \dots, w_d)

Initialize $w_j \leftarrow 0$, for $j=0 \dots d$

while not converged:

"convergence" means that the weights don't change for one entire iteration through the training data.

 shuffle training examples.

 for each training example (\mathbf{x}_i, y_i) :

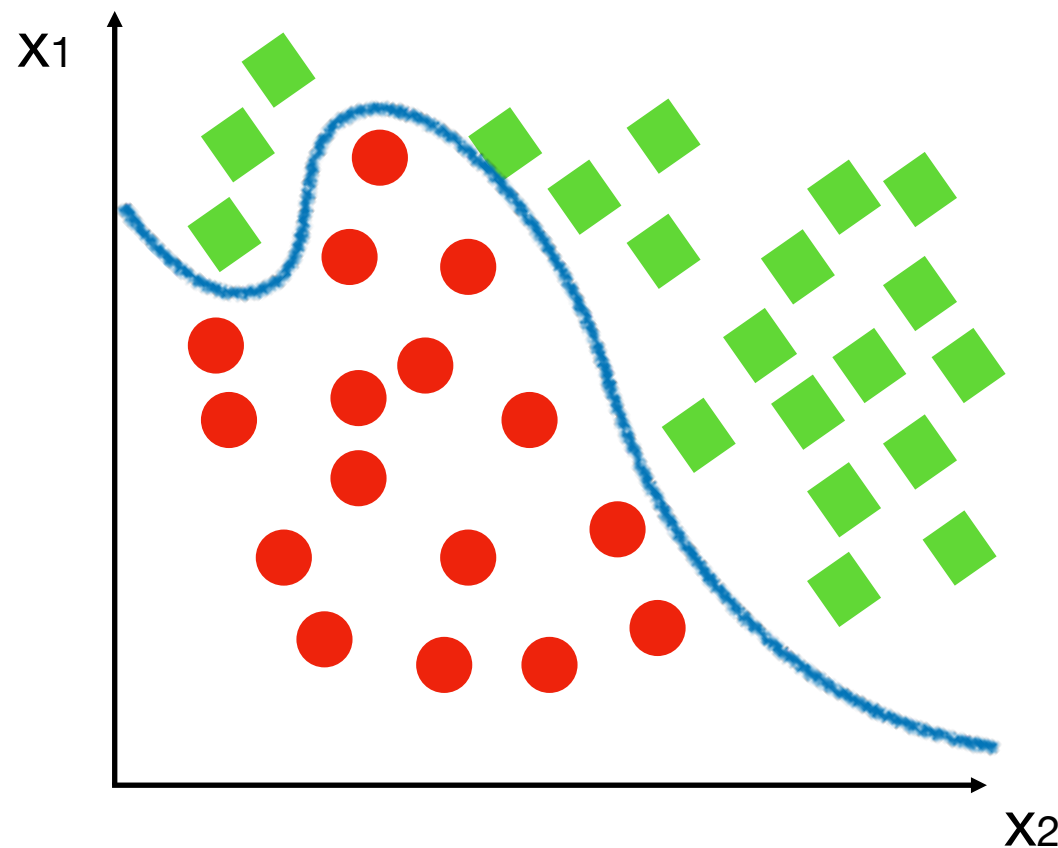
 if $\text{output} - \text{target} \neq 0$: #(output and prediction do not match)

 for each weight w_j :

$$w_j \leftarrow w_j + (y - h_{\mathbf{w}}(\mathbf{x})) \times x_j$$

Perceptron

- Simple learning algorithm. Guaranteed to converge after a finite number of steps.
- But **only** if the data is linearly separable.



perceptron cannot learn this

Feature Functions

- In NLP we often need to make multi-class decisions. Linear models provide only binary decisions.
- Use a feature function $\phi(x, y)$ where x is an input object and y is a possible output.
- The values of ϕ are d-dimensional vectors.

$$\phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$$

Log-Linear Model

(a.k.a. "Maximum Entropy Models")

- Define conditional probability $P(y|x)$

$$P(y|x; \mathbf{w}) = \frac{\exp(\mathbf{w} \cdot \phi(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w} \cdot \phi(x, y'))}$$

- $\exp(z) = e^z$ is positive for any z .

- $\sum_y P(y|x; \mathbf{w}) = 1$

- But how should we estimate \mathbf{w} ?

Log-Likelihood

- Define the log-likelihood of some model \mathbf{w} on the training data $(x_1, y_1), \dots, (x_n, y_n)$ as

$$LL(\mathbf{w}) = \sum_{i=1}^n \log P(y_i | x_i; \mathbf{w})$$

- We want to compute the maximum likelihood

$$LL^*(\mathbf{w}) = \arg \max_{\mathbf{w}} \sum_{i=1}^n \log P(y_i | x_i; \mathbf{w})$$

- Unfortunately, there is no general analytical solution. Can use gradient-based optimization.

Simple Gradient Ascent

Initialize $\mathbf{w} \leftarrow$ any setting in the parameter (weight) space

for a set number of iterations T :

for each w_i in \mathbf{w} :

$$w'_i \leftarrow w_i + \alpha \frac{\partial}{\partial w_i} LL(\mathbf{w})$$

update each w_i to w'_i

- Follow the gradients (partial derivatives) to find a parameter setting that maximizes $LL(\mathbf{w})$
- $\alpha > 0$ is the **learning rate** or **step size**.

Partial Derivative of the Log Likelihood

$$\begin{aligned}\frac{\partial}{\partial_j} LL(\mathbf{w}) &= \frac{\partial}{\partial_j} \sum_{i=1}^n \log P(y_i | x_i; \mathbf{w}) \\ &= \sum_i \phi_j(x_i, y_i) - \sum_i \sum_y P(y | x_i; \mathbf{w}) \phi_j(x_i, y)\end{aligned}$$

Regularization

- Problem: Parameter estimation can overfit the training data.
- Can include a regularization term. For example L_2 regularizer:

$$LL(\mathbf{w}) = \sum_{i=1}^n \log P(y_i | x_i; \mathbf{w}) - \frac{\lambda}{2} |\mathbf{w}|^2$$

$$LL(\mathbf{w}) = \sum_{i=1}^n \log P(y_i | x_i; \mathbf{w}) - \frac{\lambda}{2} |\mathbf{w}|^2$$

- $\lambda > 0$ controls the strength of the regularization.
- Since we are maximizing $\mathbf{w}^* = \arg \max_{\mathbf{w}} LL(\mathbf{w})$, there is now a trade-off between fit and model 'complexity'.

POS Tagging with Log-Linear Models

- Previously we used a generative model (HMM) for POS tagging.

- Now we want to use a discriminative model for

$$P(t_1, t_2, \dots, t_n | w_1, w_2, \dots, w_n) \\ = \prod_{i=1}^m P(t_i | t_1, \dots, t_{i-1}, x_1, \dots, x_m)$$

- Next tag is conditioned on previous tag sequence and all observed words.

Maximum Entropy Markov Models (MEMM)

- Make an independence assumption (similar to HMM):

$$\begin{aligned} &P(t_1, t_2, \dots, t_n | w_1, w_2, \dots, w_n) \\ &= \prod_{i=1}^m P(t_i | t_1, \dots, t_{i-1}, w_1, \dots, w_m) \\ &= \prod_{i=1}^m P(t_i | t_{i-1}, w_1, \dots, w_m) \end{aligned}$$

- Probability only depends on the previous tag.

MEMMs

$$\prod_{i=1}^m P(t_i | t_{i-1}, w_1, \dots, w_m)$$

- Model each term using a log-linear model

$$P(t_i | t_{i-1}, w_1, \dots, w_m) = \frac{\exp(\mathbf{w} \cdot \phi(w_1, \dots, w_m, i, t_{i-1}, t_i))}{\sum_{t'} \exp(\mathbf{w} \cdot \phi(w_1, \dots, w_m, i, t_{i-1}, t'))}$$

- ϕ is a feature function defined over:
 - the observed words w_1, \dots, w_m
 - the position of the current word
 - the previous tag t_{i-1}
 - the suggested tag for the current word t_i
- t' is a variable ranging over all possible tags.

MEMMs

$$P(t_i | t_{i-1}, w_1, \dots, w_m) = \frac{\exp(\mathbf{w} \cdot \phi(w_1, \dots, w_m, i, t_{i-1}, t_i))}{\sum_{t'} \exp(\mathbf{w} \cdot \phi(w_1, \dots, w_m, i, t_{i-1}, t'))}$$

- Training: same as any log-linear model.
- Decoding: Need to find $\arg \max_{t_1, \dots, t_m} P(t_i, \dots, t_m | t_{i-1}, w_1, \dots, w_m)$
 - Can use Viterbi algorithm!

Feature Function

(Ratnaparkhi, 1996)

- $\phi(w_1, \dots, w_m, i, t_{i-1}, t_i)$ is a feature vector of length d .
- $(w_i, t_i), (w_{i-1}, t_i), (w_{i-2}, t_i), (w_{i+1}, t_i), (w_{i+2}, t_i)$
- (t_{i-1}, t_i)
- $(w_i \text{ contains numbers}, t_i),$
 $(w_i \text{ contains uppercase characters}, t_i)$
 $(w_i \text{ contains a hyphen}, t_i)$
- $(\text{prefix}_1 \text{ of } w_i, t_i), (\text{prefix}_2 \text{ of } w_i, t_i), (\text{prefix}_3 \text{ of } w_i, t_i), (\text{prefix}_4 \text{ of } w_i, t_i)$
 $(\text{suffix}_1 \text{ of } w_i, t_i), (\text{suffix}_2 \text{ of } w_i, t_i), (\text{suffix}_3 \text{ of } w_i, t_i), (\text{suffix}_4 \text{ of } w_i, t_i)$

Feature Example

The stories about well-heeled communities and developers ...

DT NNS IN ??

- (*well-heeled*,JJ), (*about*,JJ), (*stories*,JJ), (*communities*, JJ), (*and*,JJ)
- (IN,JJ)
- (w_i contains a hyphen, JJ)
- (*w*,JJ), (*we*,JJ), (*wel*,JJ), (*well*, JJ)
(*d*,JJ), (*ed*,JJ), (*led*,JJ), (*eled*, JJ)