Data mining & Machine Learning

CS 373
Purdue University

Dan Goldwasser
dgoldwas@purdue.edu

Today's Lecture

Linear classifiers

- *Last time we showed that Naïve Bayes is a linear classifier!*
  - *Weights determined by conditional probability distributions*
- *In fact, Linear functions form a very popular* ***hypothesis space***
  - ***Many learning algorithms – Perceptron, SVM, Logistic Regression,..***
- *Today we will discuss their properties and introduce Perceptron, a learning algorithm for linear functions.*

Make sure you understand..

- Supervised vs. Unsupervised
- Discriminative vs. Generative
- Parametric vs. Non-Parametric
- Overfitting vs. Underfitting

But first, a quick wrap up of Naïve Bayes

Bayes rule for probabilistic classifier

$$y_{MAP} = \text{argmax}_{y \in Y} P(x_1, x_2, \ldots, x_n | y) P(y)$$

- Given training data we can estimate the two terms
  - Estimating P(y) is **easy**. For each value v count how many times it appears in the training data.

Question: Assume binary x_i's. How many parameters does the model require?

- However, it is not feasible to estimate $P(x_1, \ldots, x_n \mid y)$
  - In this case we have to estimate, for each target value, the probability of each instance (most of which will not occur)
- In order to use a Bayesian classifiers in practice, we *need to make assumptions* that will allow us to estimate these quantities.

Inductive bias – Naïve Bayes version

- An acute version of overfitting occurs when we try to estimate $P(Y|\mathbf{X}) = P(Y)\ P(\mathbf{X}|Y)$ directly
- It requires learning 2^n parameters, **essentially one parameter for each input instance.**
  - This is overfitting at its worst – just memorizing the data
- We encountered this problem before in decision trees – Trees that have n intermediate nodes only memorize the data.
  - How did we solve it for decision trees?
- Similarly – making independence assumptions is a way to control the complexity of the model space and prevent overfitting.

NB: Independence Assumptions

Conditional Independence:

Assume feature probabilities are independent given the label

$P(x_i|y_j) = P(x_i|x_{i-1}; y_j)$

$$P(Y|\mathbf{X}) \propto P(\mathbf{X}|Y)P(Y)$$

Bayes rule

$$\propto \prod_{i=1}^{m} P(X_i|Y)P(Y)$$

Naive assumption

Question: How many parameters do we need to estimate now?

NB: Independence Assumptions

$$p(\mathtt{y} \mid \mathbf{x}) = \frac{p(\mathtt{y})\, p(\mathbf{x} \mid \mathtt{y})}{p(\mathbf{x})}$$

NB Decision Rule

$$\mathtt{p(y)p(x|y)} = p(x_1, \ldots, x_n, \mathtt{y})$$

Joint probability definition

$$\begin{aligned}
&= p(x_1 \mid x_2, \ldots, x_n, \mathtt{y})\, p(x_2, \ldots, x_n, \mathtt{y}) \\
&= p(x_1 \mid x_2, \ldots, x_n, \mathtt{y})\, p(x_2 \mid x_3, \ldots, x_n, \mathtt{y})\, p(x_3, \ldots, x_n, \mathtt{y}) \\
&= \ldots \\
&= p(x_1 \mid x_2, \ldots, x_n, \mathtt{y})\, p(x_2 \mid x_3, \ldots, x_n, \mathtt{y}) \ldots p(x_{n-1} \mid x_n, \mathtt{y})\, p(x_n \mid \mathtt{y})\, p(\mathtt{y})
\end{aligned}$$

Chain rule

$$p(x_i \mid x_{i+1}, \ldots, x_n, \mathtt{y}) = p(x_i \mid \mathtt{y})$$

Conditional Independence

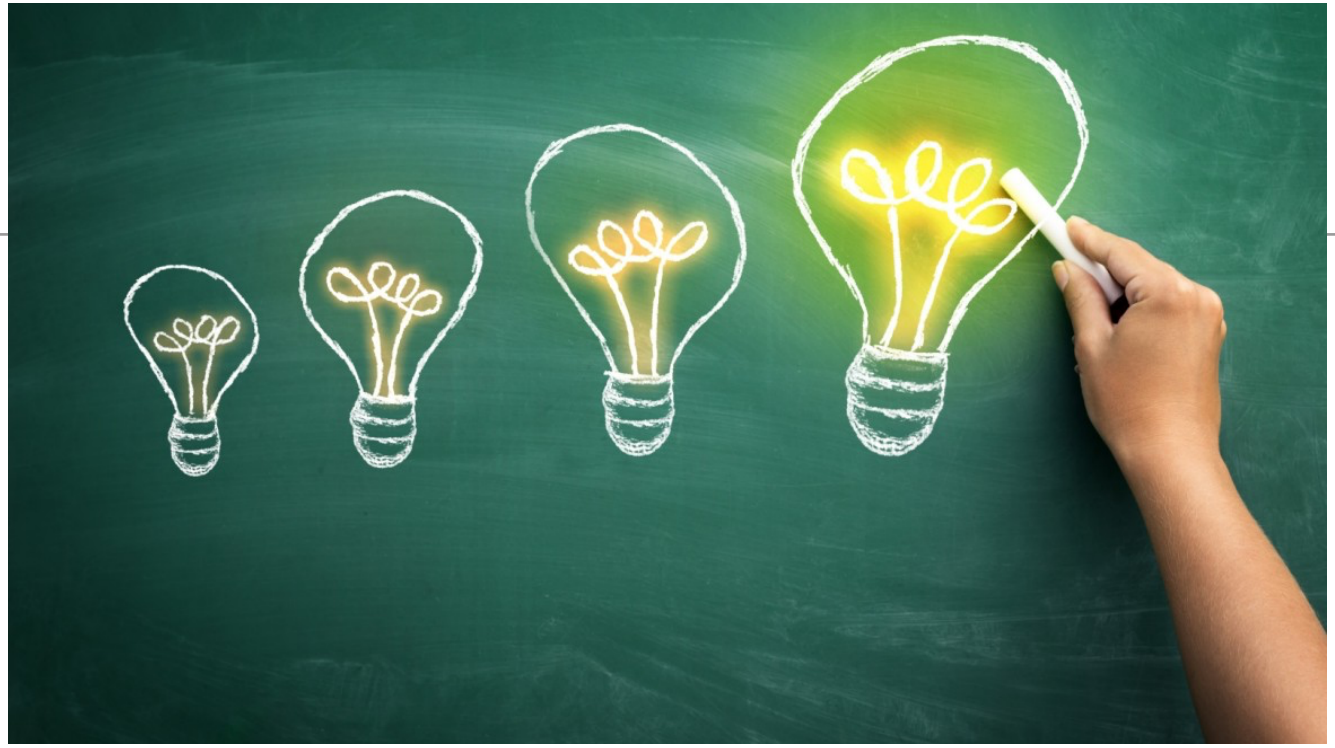
NB: Independence Assumptions

$$p(x_i \mid x_{i+1}, \ldots, x_n, \texttt{y}) = p(x_i \mid \texttt{y})$$

Conditional Independence

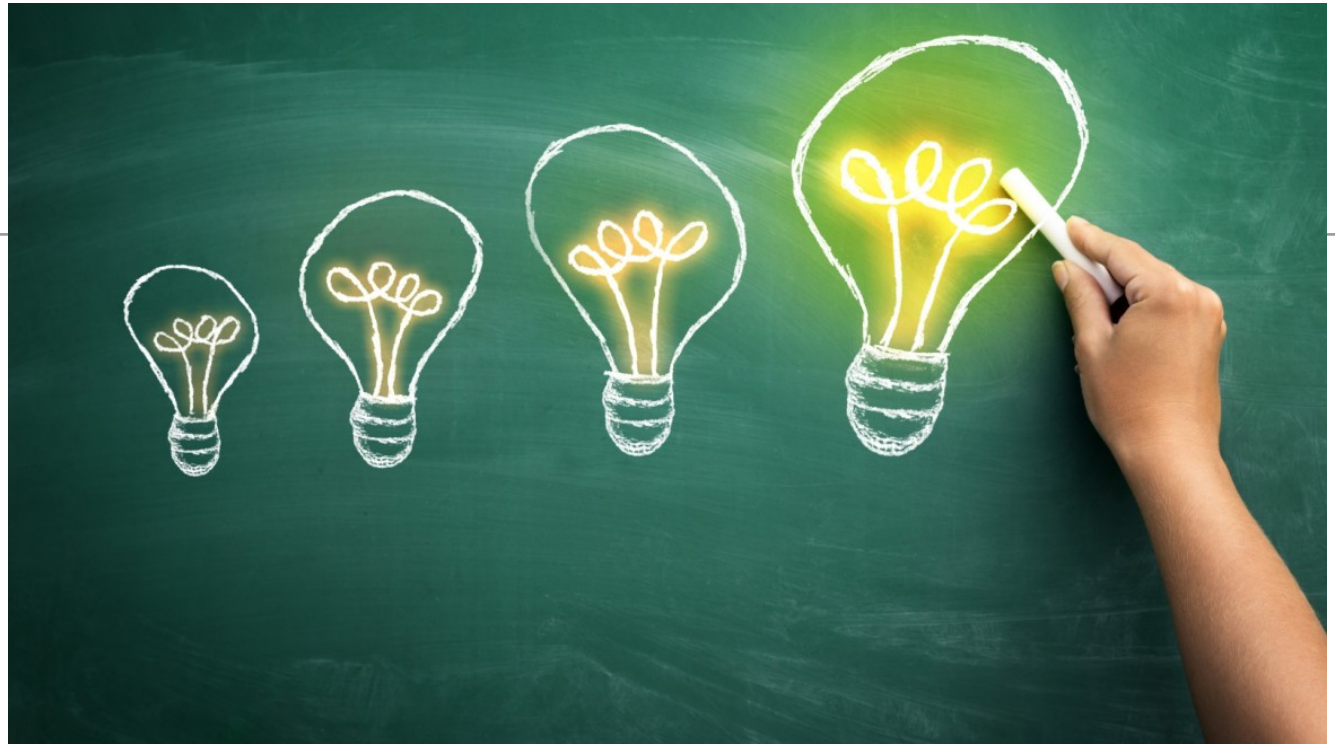
$$\begin{aligned} p(\texttt{y} \mid x_1, \ldots, x_n) &\propto p(\texttt{y}, x_1, \ldots, x_n) = \\ &= p(\texttt{y})\, p(x_1 \mid \texttt{y})\, p(x_2 \mid \texttt{y})\, p(x_3 \mid \texttt{y}) \cdots \\ &= p(\texttt{y}) \prod_{i=1}^{n} p(x_i \mid \texttt{y}). \end{aligned}$$

Apply Conditional Independence To NB Decision rule



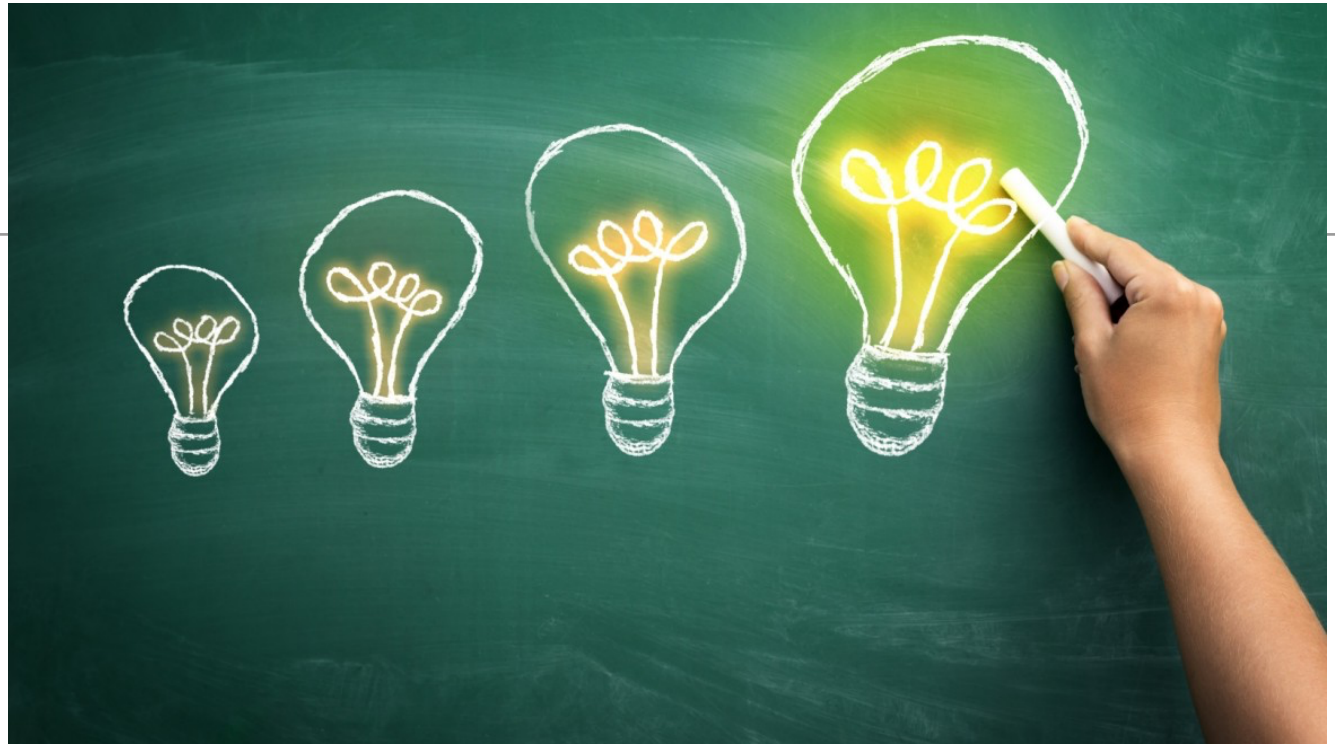
Big Idea!

Simplifying assumptions are useful even when incorrect – **they can lead to better generalization!**



Big Idea!

The actual complexity of the classifier depends on the hypothesis class we consider AND the feature decisions we make.



Big Idea!

It's important to diagnose our learner – if it's ***overfitting****, you can move to a simpler function class (e.g., use NB), simplify the feature set, or annotate more data. If it's* ***underfitting****, you can add more features*

NB: Learning

NBC learning

$$P(BC|A,I,S,CR) = \frac{P(A,I,S,CR|BC)P(BC)}{P(A,I,S,CR)}$$
$$= \frac{P(A|BC)P(I|BC)P(S|BC)P(CR|BC)P(BC)}{P(A,I,S,CR)}$$
$$\propto P(A|BC)P(I|BC)P(S|BC)P(CR|BC)P(BC)$$

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

NBC parameters = CPDs+prior

CPDs: P(A|BC)
P(I|BC)
P(S|BC)
P(CR|BC)
Prior: P(BC)

NB Learning: **Short Version**

X_1

Y	Low	Medium	High
Yes	10	13	17
No	2	13	0

NBC parameters = CPDs+prior

$$P[X_1 = \text{Low} \mid Y = \text{Yes}] = \frac{10}{(10+13+17)}$$

$$P[Y = \text{No}] = \frac{(2+13)}{(2+13+10+13+17)}$$

Now, the long version…

Likelihood

- Let $D = \{x(1), ..., x(n)\}$
- Assume the data *D* are independently sampled from the same distribution: $p(X|\theta)$
- The likelihood function represents the probability of the data as a function of the model parameters:

$$\begin{aligned} L(\theta|D) &= L(\theta|x(1), ..., x(n)) \\ &= p(x(1), ..., x(n)|\theta) \\ &= \prod_{i=1}^{n} p(x(i)|\theta) \end{aligned}$$

If instances are independent, likelihood is product of probs

NBCs: Likelihood

- NBC likelihood uses the NBC probabilities for each data instance (i.e., probability of the class given the attributes)

$$L(\theta|D) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i;\theta)$$

General likelihood

$$\propto \prod_{i=1}^{n} p(\mathbf{x}_i|y_i;\theta)p(y_i|\theta)$$

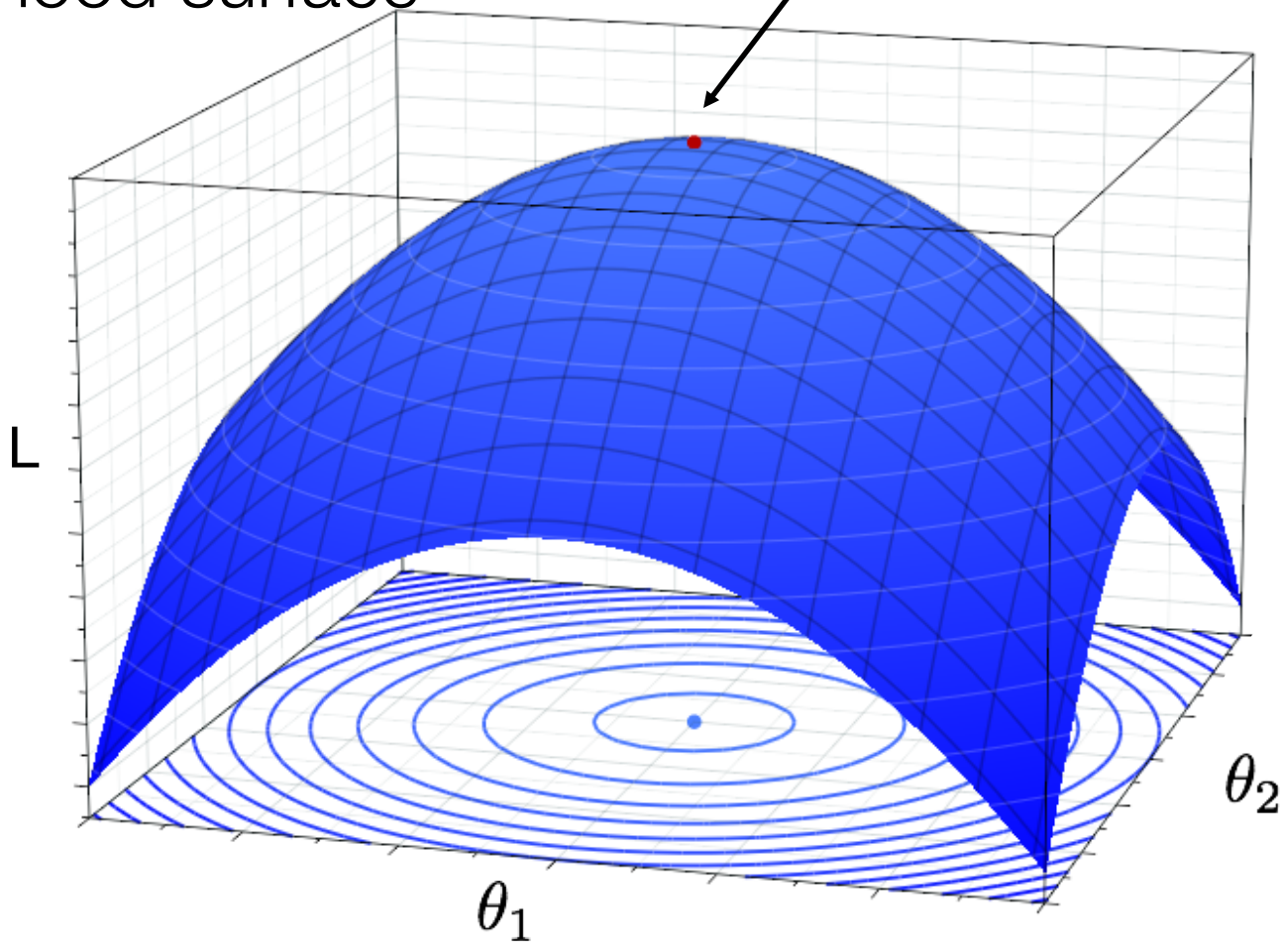
Bayes rule

$$\propto \prod_{i=1}^{n}\prod_{j=1}^{p} p(x_{ij}|y_i;\theta)p(y_i|\theta)$$

Naive assumption

$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta)$$

Likelihood surface



If the likelihood surface is convex we can often determine the parameters that maximize the function analytically

MLE for multinomials

- Let $X \in \{1, ..., k\}$ be a discrete random variable with k values, where $P(X=j)=\theta_j$
- Then P(X) is a multinomial distribution:

$$P(X|\theta) = \prod_{j=1}^{k} \theta_j^{I(X=j)}$$

  where I(X=j) is an indicator function

- The likelihood for a data set $D=[x_1, ..., x_N]$ is:

$$P(D|\theta) = \prod_{n=1}^{N} \prod_{j=1}^{k} \theta_j^{I(x_n=j)} = \prod_{j} \theta_j^{N_j}$$

- The ML estimates for each parameter are: (using Lagrange multipliers)

$$\hat{\theta}_j = \frac{N_j}{N}$$

In this case, MLE can be determined analytically by counting

Naïve Bayes:
Practical Implementation Advice

Zero counts are a problem

- If an attribute value does not occur in training example, we assign **zero** probability to that value
- How does that affect the conditional probability P[f(x) | x] ?
- It equals 0!!!
- Why is this a problem?
- Adjust for zero counts by “smoothing” probability estimates

$P[X_1 = \text{High} \mid Y = \text{No}] =$

$$\frac{0 + 1}{(2 + 13 + 0) + 3}$$

Adds uniform prior

Laplace correction
Numerator: *add 1*
Denominator: *add k*,
where k=number of possible values of X

Numerical Stability

- Recall: NB classifier: $\propto \prod_{i=1}^{m} P(X_i|Y)P(Y)$

- **Multiplying probabilities can get us into problems!** better to sum logs of probabilities rather than multiplying probabilities.

- Class with highest final un-normalized log probability score is still the most probable.

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} \log P(c_j) + \sum_{i \in positions} \log P(x_i \mid c_j)$$

Summary: Naive Bayes classifier

- **Simplifying (naive) assumption:**
  attributes are conditionally independent given the class
- **Strengths**:
  - Easy to implement
  - Often performs well even when assumption is violated
  - Learning is really fast! *(why?)*
- **Weaknesses:**
  - Class conditional assumption produces skewed probability estimates
  - Dependencies among variables cannot be modeled

Summary: NBC learning

- **Model space**
  - Parametric model with specific form
    (i.e., based on Bayes rule and assumption of conditional independence),
  - Models vary based on parameter estimates in CPDs
- **Search algorithm**
  - MLE optimization of parameters
    (convex optimization results in exact solution)
- **Scoring function**
  - Likelihood of data given NBC model form

From Naïve Bayes to Linear Classifiers

Naïve Bayes: Two classes

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_i P(x_i \mid v_j)$$

- Notice that the naïve Bayes method gives a method for predicting rather than an explicit classifier
- In the case of two classes, $v \in \{0,1\}$ we predict that $v=1$ iff:

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} P(x_i \mid v_j = 1)}{P(v_j = 0) \bullet \prod_{i=1}^{n} P(x_i \mid v_j = 0)} > 1$$

Naïve Bayes: Two classes

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_i P(x_i \mid v_j)$$

- NB Classier:

In the case of two classes, $v \in \{0,1\}$ we predict that $v=1$ iff:

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} P(x_i \mid v_j = 1)}{P(v_j = 0) \bullet \prod_{i=1}^{n} P(x_i \mid v_j = 0)} > 1$$

Let's "simplify" the notation:

$$\text{Denote: } p_i = P(x_i = 1 \mid v = 1), \quad q_i = P(x_i = 1 \mid v = 0)$$

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} p_i^{x_i} (1 - p_i)^{1-x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} q_i^{x_i} (1 - q_i)^{1-x_i}} > 1$$

Naïve Bayes: Two classes

In the case of two classes, $v \in \{0,1\}$ we predict that $v=1$ iff:

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} p_i^{x_i} (1-p_i)^{1-x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} q_i^{x_i} (1-q_i)^{1-x_i}} = \frac{P(v_j = 1) \bullet \prod_{i=1}^{n} (1-p_i) \left(\frac{p_i}{1-p_i}\right)^{x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} (1-q_i) \left(\frac{q_i}{1-q_i}\right)^{x_i}} > 1$$

Naïve Bayes: Two classes

In the case of two classes, $v \in \{0,1\}$ we predict that $v=1$ iff:

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} p_i^{x_i} (1-p_i)^{1-x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} q_i^{x_i} (1-q_i)^{1-x_i}} = \frac{P(v_j = 1) \bullet \prod_{i=1}^{n} (1-p_i) \left(\frac{p_i}{1-p_i}\right)^{x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} (1-q_i) \left(\frac{q_i}{1-q_i}\right)^{x_i}} > 1$$

Take logarithm. We predict v=1, iff:

$$\log \frac{P(v_j = 1)}{P(v_j = 0)} + \sum_i \log \frac{1-p_i}{1-q_i} + \sum_i \left(\log \frac{p_i}{1-p_i} - \log \frac{q_i}{1-q_i}\right) x_i > 0$$

Naïve Bayes: Two classes

In the case of two classes, $v \in \{0,1\}$ we predict that v=1 iff:

$$\frac{P(v_j = 1) \bullet \prod_{i=1}^{n} p_i^{x_i} (1-p_i)^{1-x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} q_i^{x_i} (1-q_i)^{1-x_i}} = \frac{P(v_j = 1) \bullet \prod_{i=1}^{n} (1-p_i)(\frac{p_i}{1-p_i})^{x_i}}{P(v_j = 0) \bullet \prod_{i=1}^{n} (1-q_i)(\frac{q_i}{1-q_i})^{x_i}} > 1$$

Take logarithm. We predict v=1, iff:

$$\log \frac{P(v_j = 1)}{P(v_j = 0)} + \sum_i \log \frac{1-p_i}{1-q_i} + \sum_i (\log \frac{p_i}{1-p_i} - \log \frac{q_i}{1-q_i}) x_i > 0$$

We get that **naive Bayes is a linear separator** with :

$$w_i = \log \frac{p_i}{1-p_i} - \log \frac{q_i}{1-q_i} = \log \frac{p_i}{q_i} \frac{1-q_i}{1-p_i}$$

$$\text{if } p_i = q_i \text{ then } w_i = 0 \text{ and the feature is irrelevant}$$

Linear Classifiers

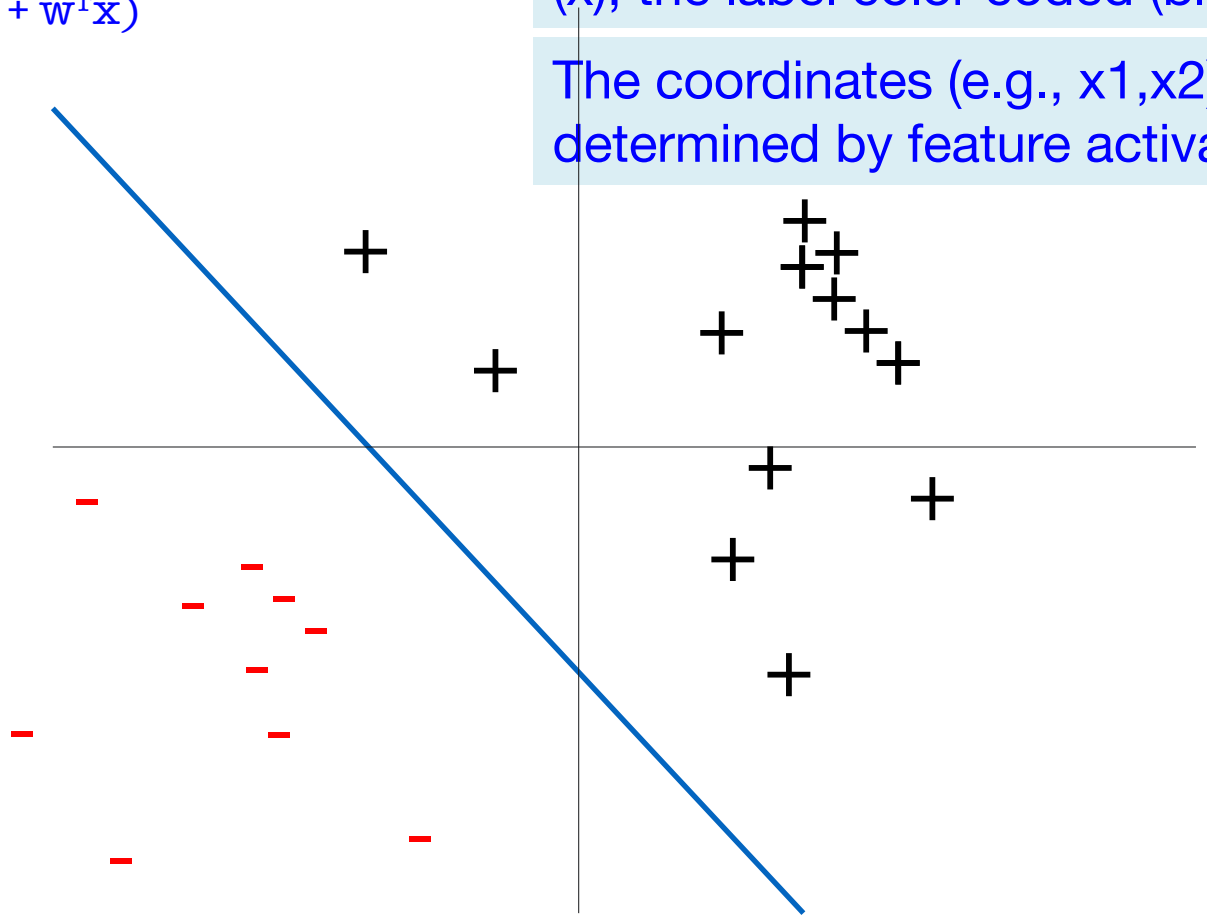
- Linear threshold functions
  - Associate a weight (w_i) with each feature (x_i)
  - **Prediction**: $\text{sign}(b + w^T x) = \text{sign}\,(b + \Sigma\, w_i\, x_i)$
    - $b + w^T x \geq 0$ predict y=1
    - Otherwise, predict y=-1
- ***NB is a linear threshold function***
  - Weight vector (w) is assigned by computing conditional probabilities
- *In fact, Linear threshold functions are a very popular representation!*

Linear Classifiers

$\text{sign}(b + w^T x)$

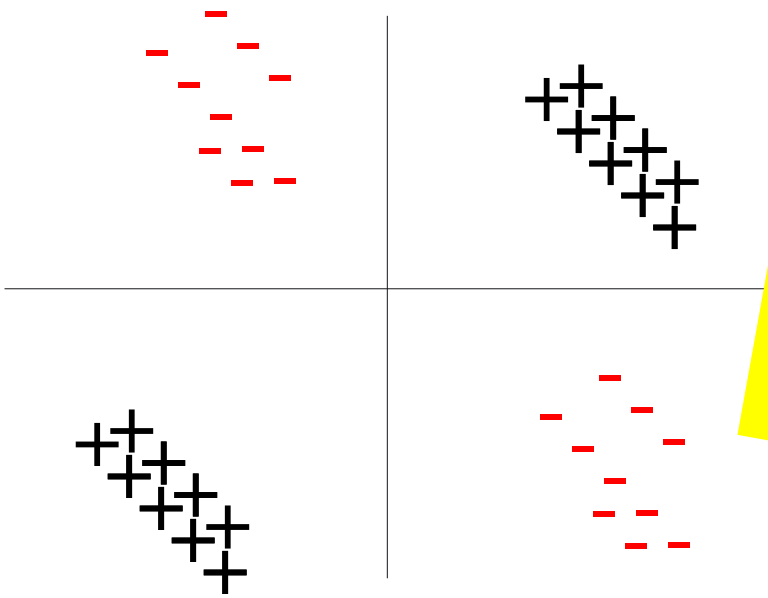
Each point in this space is an instance (x), the label color coded (black,red)

The coordinates (e.g., x1,x2), are determined by feature activations



Expressivity

- Linear functions are quite expressive
  - *Exists a linear function that is consistent with the data*
- **A famous negative examples (XOR):**



Is this a "deal breaker"?

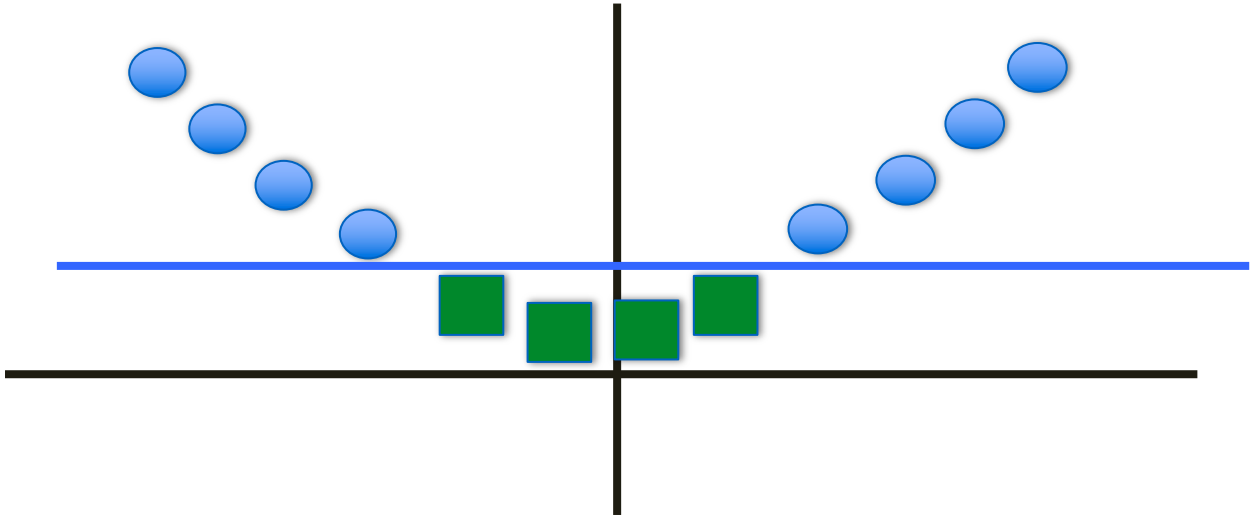
Pop Quiz!

- We want to identify car buying preferences
  - Cars can be described as a set of properties: engine size, color, #cup-holders, wheel size, gas consumption, max speed, acceleration, #seats, ..
- **We assume:** (1) there is a subset of car properties people care about (2) if the car has at least K of these properties, the costumer will buy the car.
- **Question**: Is the hypothesis space of linear functions appropriate for this problem?

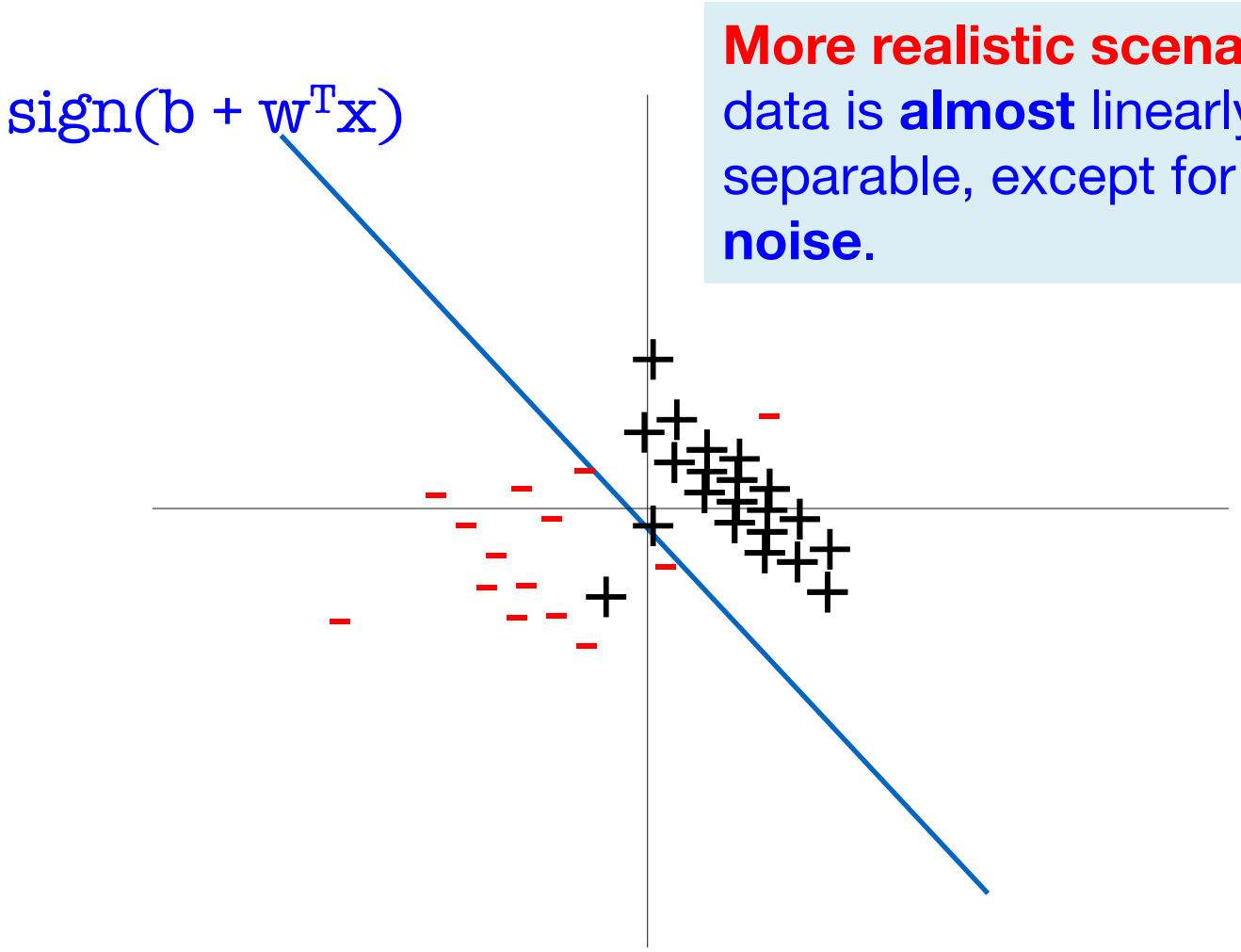
Expressivity

By transforming the feature space these functions can be made linear

Represent each point in 2D as (x,x^2)



Expressivity



More realistic scenario: the data is **almost** linearly separable, except for some **noise**.

Features

- So far we have discussed simple attribute-based representation
  - *In fact, you can use a very rich representation*
- **Broader definition**
  - Functions mapping attributes of the input to a Boolean/categorical/numeric value

$$\phi_1(x) = \begin{cases} 1 & x_1 \ \ is \ \ capitalized \\ 0 & otherwise \end{cases}$$

$$\phi_k(x) = \begin{cases} 1 & x \ \ contains \ "good" \ \ more \ than \ twice \\ 0 & otherwise \end{cases}$$

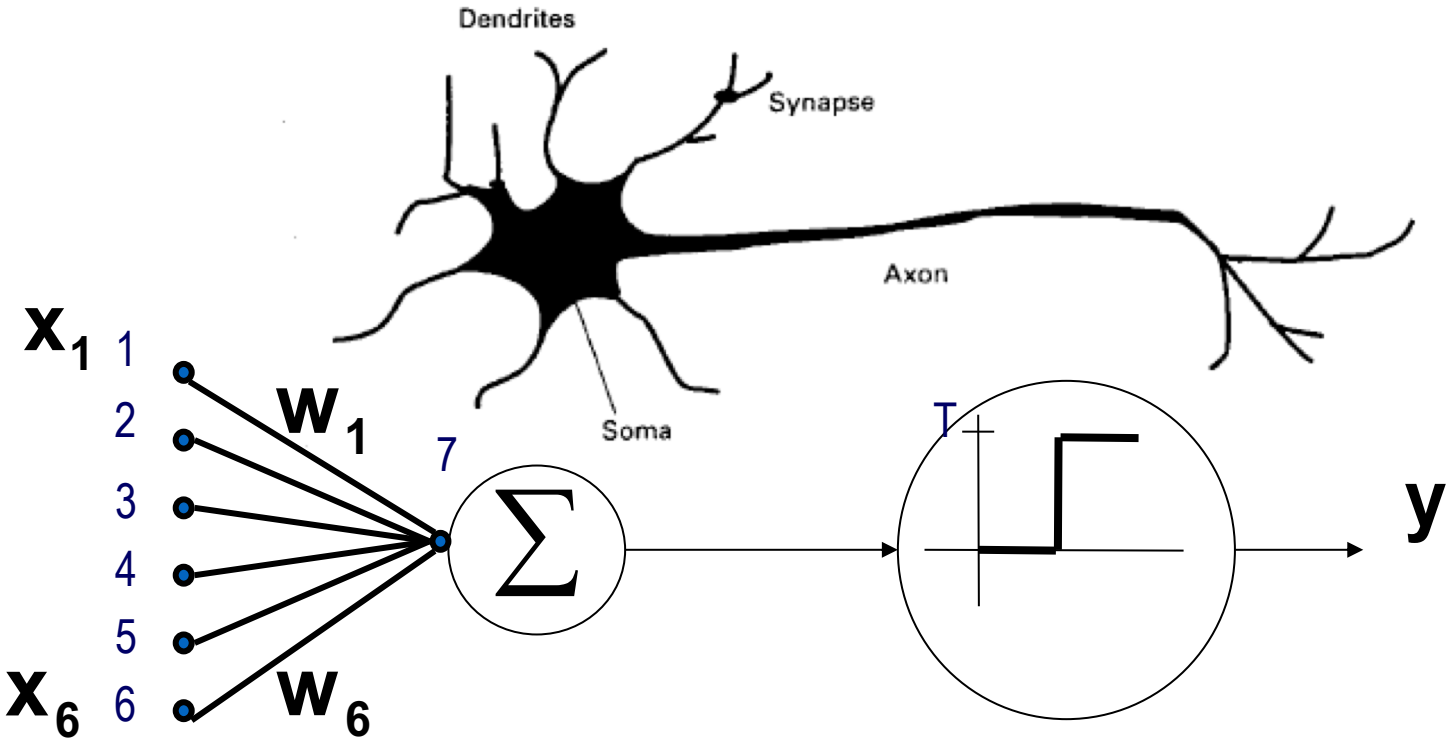
Perceptron

- One of the earliest learning algorithms
  - Introduced by Rosenblatt 1958 to model neural learning
- **Goal**: directly search for a separating hyperplane
  - If one exists, perceptron will find it
  - If not, …
- **Online** algorithm
  - Considers one example at a time (NB – looks at entire data)
- **Error driven** algorithm
  - Updates the weights only when a mistake is made

Perceptron: Biological motivation

- **Rosenblatt** suggested that when a target output value is provided for a single neuron with fixed input, it can incrementally change weights and learn to produce the output using the Perceptron learning rule

- Perceptron = Linear threshold unit



Perceptron

- We learn f:X→ {-1,+1} represented as f =sgn{w•x)
- Where X= $\{0,1\}^n$
- Given Labeled examples: $\{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$

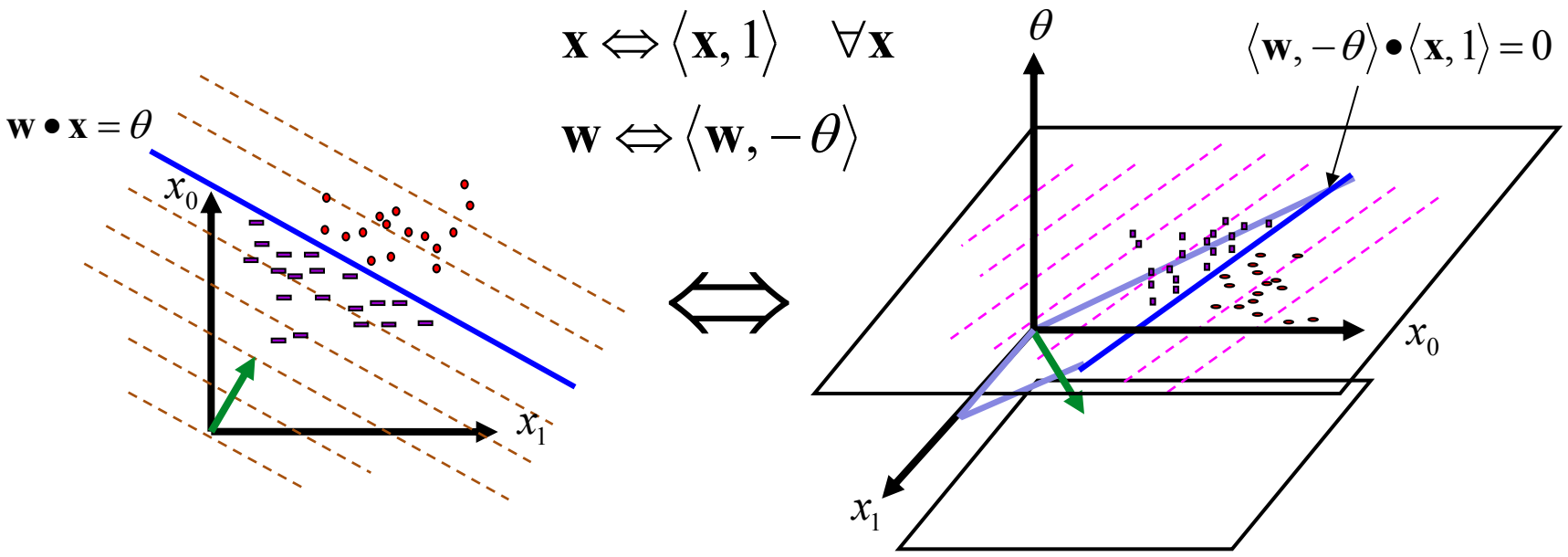
1. Initialize w=0 $\in \mathbf{R}^n$
2. Cycle through all examples, until no more errors are made
   a. **Predict** the label of instance x to be y' = sgn{w•x)
   b. If ***y'≠y***, **update** the weight vector:

      w = w + r y x (r - a constant, learning rate)

      Otherwise, if **y'=y**, leave weights unchanged.

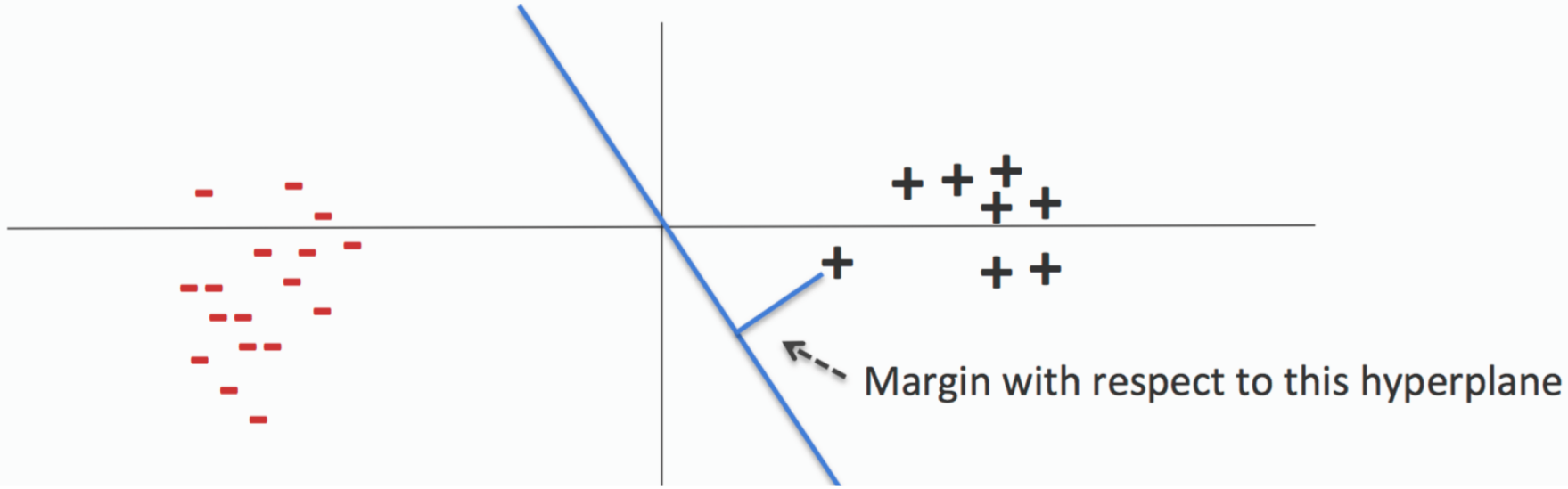
Footnote About the Threshold

- On previous slide, Perceptron has no threshold
- But we don't lose generality:



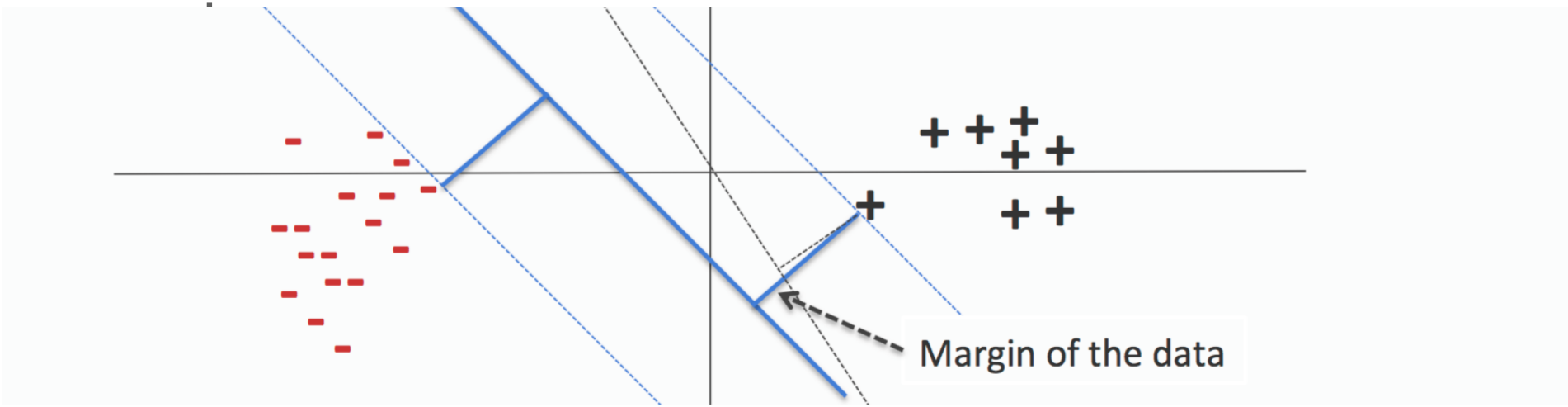
Margin

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Margin

- The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
- The margin of a data set (γ) is the maximum margin possible for that dataset using any weight



Mistake Bound for Perceptron

- Let $D=\{(x_i, y_i)\}$ be a labeled dataset that is separable
- Let $||x_i|| < R$ for all examples.
- Let γ be the margin of the dataset D.
- Then, the perceptron algorithm will make at most R^2/γ^2 mistakes on the data.

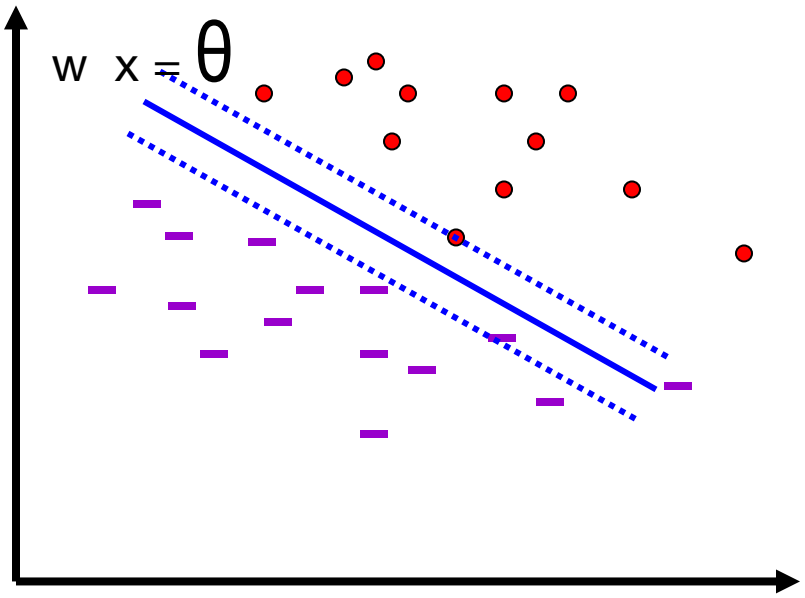
Regularization: Perceptron with Margin

- *Weights with better margin generalize better*
  - Perceptron finds *any* separating hyperplane
- **Thick Separator (aka as Perceptron with Margin)**
  - **Predict positive**

    $w \cdot x - \theta > \gamma$
  - **Predict negative**

    $w \cdot x - \theta < -\gamma$
  - **Mistake:**

    $\gamma > (w \cdot x - \theta) > -\gamma$



Regularization: Perceptron with Margin

- **Perceptron margin** : hyperparameter that has to be tuned using the validation set (try different values)
  - *In the future: the data will decide the margin*
- The impact of margin regularization in perceptron becomes smaller as *w* grows
  - Can we control the growth of w?
- In practice, **very effective**

Perceptron: Robust Variation

- *The perceptron algorithm counts later points more than earlier points*

1: (0,1,..,1,0,1)

2: (0,1,..,1,0,1)

…

100: (0,1,..,1,0,1)

…

10000: (0,1,..,1,0,1)

Makes some mistakes, update..
After 100 examples, learner stops making mistakes

We keep going…

BUT then at the 10,000 example the learner makes a mistake!

Is this a problem?

Voted Perceptron

- *Training:*
  - *Learner remembers how long each hypothesis survived (no mistakes on w)*
- Test:
  - Weighted vote of all participating hypotheses

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

- Heavy: (1) Computational effort (2) **Storage**

Averaged Perceptron

- **Training**: *Maintain a running weighted average of survived hypotheses*
- **Test**: *Predict according to the averaged weight vector*

Voted Perceptron →

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

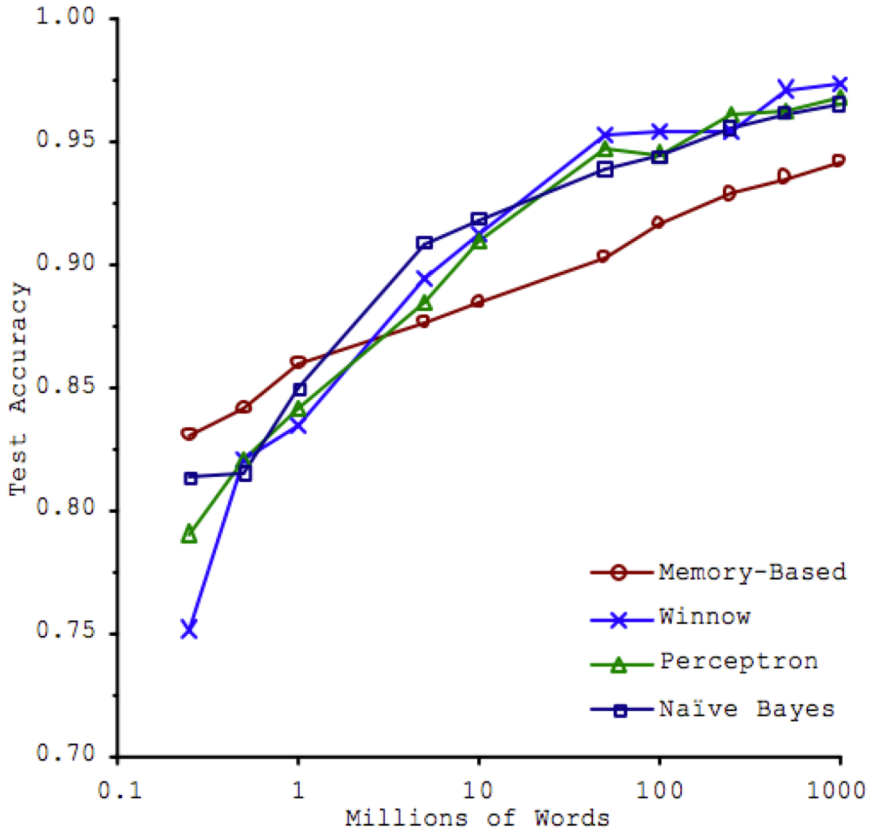
← Averaged Perceptron

- *An efficient approximation of voted perceptron*
- *Almost always better than regular perceptron!*

Practical Example

Task: ***context sensitive spelling***

"I didn't know {weather,whether} to laugh or cry"



Source: Scaling to very very large corpora for natural language disambiguation Michele Banko, Eric Brill. MSR, 2001.

Example question: Compare NBC to DT to KNN

- **Hypothesis space**
  - What type of functions are used? Which one is more expressive?
- **Scoring function**
  - How is each model scored?
- **Search**
  - Which search procedure is used?
  - Are we guaranteed to find the optimal model?
  - What is the complexity of the search procedure?