# Topics in Machine Learning
## (with less Magic)

Grzegorz Chrupała

Spoken Dialog Systems
Saarland University

CNGL March 2009

# Outline



# Outline



# Notes on notation

- In machine learning we learn functions from examples of inputs and outputs
- The inputs are usually denoted as $x \in \mathcal{X}$.
- The outputs are $y \in \mathcal{Y}$
- The most common and well studied scenario is classification: we learn to map some arbitrary objects into a small number of fixed classes
- Our arbitrary input object have to be represented somehow: we have to extract the features if the object which are useful for determining which output it should map to

# Feature function

Item Objects are represented using the feature function, also known as the representation function. The most commonly used representation is a $d$ dimensional vector of real values:

$$\Phi : \mathcal{X} \rightarrow \mathbb{R}^d \tag{1}$$

$$\Phi(x) = \begin{pmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_d \end{pmatrix} \tag{2}$$

- We will often simplify notation by assuming that input object are already vectors in $d$ dimensional real space

# Basic vector notation and operations

- In this tutorial vectors are written in boldface $\mathbf{x}$. An alternative notation is $\overrightarrow{x}$
- Subscripts and italic are used to refer to vector components: $x_i$
- Subscripts on boldface symbols, or more commonly superscripts in brackets are used to index whole vectors: $\mathbf{x}_i$ or $\mathbf{x}^{(i)}$
- Dot (inner) product can be written:
  - $\mathbf{x} \cdot \mathbf{z}$
  - $\langle \mathbf{x}, \mathbf{z} \rangle$
  - $\mathbf{x}^T \mathbf{z}$

# Dot product and matrix multiplication

Notation $\mathbf{x}^T$ treats the vector as a one column matrix and transposes it into a one row matrix.
This matrix can then be multiplied with a one column matrix, giving a scalar.

- Dot product:

$$\mathbf{x} \cdot \mathbf{z} = \sum_{i=1}^{d} x_i z_i$$

- Matrix multiplication:

$$(\mathbf{AB})_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

- Example

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = x_1 z_1 + x_2 z_2 + x_3 z_3$$

# Supervised learning

In supervised learning we try to learn a function $h : \mathcal{X} \to \mathcal{Y}$ where .

- Binary classification: $\mathcal{Y} = \{-1, +1\}$
- Multiclass classification: $\mathcal{Y} = \{1, \ldots, K\}$ (finite set of labels)
- Regression: $\mathcal{Y} = \mathbb{R}$
- Sequence labeling: $h : \mathcal{W}^n \to \mathcal{L}^n$
- Structure learning: Inputs and outputs are structures such as e.g. trees or graphs

We will often initially focus on binary classification, and then generalize

# Outline



# Prior, conditional, posterior

- A priori probability (prior): $P(Y_i)$
- Class-conditional
  - Density $p(x|Y_i)$ (continuous feature $x$)
  - Probability $P(x|Y_i)$ (discrete feature $x$)
- Joint probability: $p(Y_i, x) = P(Y_i|x)p(x) = p(x|Y_i)P(Y_i)$
- Posterior via Bayes formula:

$$P(Y_i|x) = \frac{p(x|Y_i)p(Y_i)}{p(x)}$$

- $p(x|Y_i)$ likelihood of $Y_i$ with respect to $x$
- In general we work with feature vectors $\mathbf{x}$ rather than single features $x$

# Loss function and risk for classification

- Let $\{Y_1..Y_c\}$ be the classes and $\{\alpha_1, \ldots, \alpha_a\}$ be the decisions
- Then $\lambda(\alpha_i|Y_j)$ describes the loss associated with decision $\alpha_i$ when the target class is $Y_j$
- Expected loss (or risk) associated with $\alpha_i$:

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda(\alpha_i|Y_j)P(Y_j|\mathbf{x})$$

- A decision function $\alpha$ maps feature vectors $\mathbf{x}$ to decisions $\alpha_1, \ldots, \alpha_a$
- The overall risk is then given by:

$$R = \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

# Zero-one loss for classification

- The zero-one loss function assigns loss 1 when a mistake is made and loss 0 otherwise
- If decision $\alpha_i$ means deciding that the output class is $Y_i$ then the zero-one loss is:

$$\lambda(\alpha_i|Y_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

- The risk under zero-one loss is the average probability of error:

$$R(\alpha_i|\mathbf{x}) = \sum_{j=0}^{c} \lambda(\alpha_i|Y_j) P(Y_j|\mathbf{x}) \tag{3}$$

$$= \sum_{j \neq i} P(Y_j|\mathbf{x}) \tag{4}$$

$$= 1 - P(Y_i|\mathbf{x}) \tag{5}$$

# Bayes decision rule

- Under Bayes decision rule we choose the action which minimizes the conditional risk. Thus we choose the class $Y_i$ which maximizes $P(Y_i|\mathbf{x})$:

**Bayes decision rule**

$$\text{Decide } Y_i \text{ if } P(Y_i|\mathbf{x}) > P(Y_j|\mathbf{x}) \qquad \text{for all } j \neq i \tag{6}$$

- Thus the Bayes decision rule gives minimum error rate classification

# Problem: risk under randomized decision rule

Suppose that we replace the deterministic function $\alpha(x)$ with a randomized rule, namely one giving the probability $P(\alpha_i|x)$ of taking action $\alpha_i$ on observing $x$.

- What is the resulting risk?

# Problem: risk under randomized decision rule

Suppose that we replace the deterministic function $\alpha(x)$ with a randomized rule, namely one giving the probability $P(\alpha_i|x)$ of taking action $\alpha_i$ on observing $x$.

- What is the resulting risk?

$$R = \int \left[ \sum_{i=1}^{a} R(\alpha_i|x) P(\alpha_i|x) \right] p(x) dx$$

# Problem: risk under randomized decision rule

Suppose that we replace the deterministic function $\alpha(x)$ with a randomized rule, namely one giving the probability $P(\alpha_i|x)$ of taking action $\alpha_i$ on observing $x$.

- What is the resulting risk?

$$R = \int \left[ \sum_{i=1}^{a} R(\alpha_i|x) P(\alpha_i|x) \right] p(x) dx$$

- Show that $R$ is minimized by choosing $P(\alpha_i|x) = 1$ for the action $\alpha_i$ associated with the minimum conditional risk $R(\alpha_i|x)$, and thus no benefit can be obtained from randomizing the decision rule.

# Discriminant functions

- For classification among $c$ classes, have a set of $c$ functions $\{g_1, \ldots, g_c\}$
- For each class $X_i$, $g_i : X \rightarrow \mathbb{R}$
- The classifier chooses the class index for $\mathbf{x}$ by solving:

$$y = \underset{i}{\operatorname{argmax}}\, g_i(\mathbf{x})$$

  That is it chooses the category corresponding to the largest discriminant
- For the Bayes classifier under the minimum error rate decision, $g_i(\mathbf{x}) = P(Y_i|\mathbf{x})$

# Choice of discriminant function

- Choice of the set of discriminant functions is not unique
- We can replace every $g_i$ with $f \circ g_i$ where $f$ is a monotonically increasing (or decreasing) function without affecting the decision.
- Examples
  - $g_i(\mathbf{x}) = p(Y_i|\mathbf{x}) = p(\mathbf{x}|Y_i)P(Y_i)/\sum_j p(\mathbf{x}|Y_j)P(Y_j)$
  - $g_i(\mathbf{x}) = p(\mathbf{x}|Y_i)P(Y_i)$
  - $g_i(\mathbf{x}) = \ln p(\mathbf{x}|Y_i) + \ln P(Y_i)$

# Dichotomizer

- A dichotomizer is a binary classifier
- Traditionally treated in a special way, using a single determinant

$$g(\mathbf{x}) \equiv g_1(\mathbf{x}) - g_2(\mathbf{x})$$

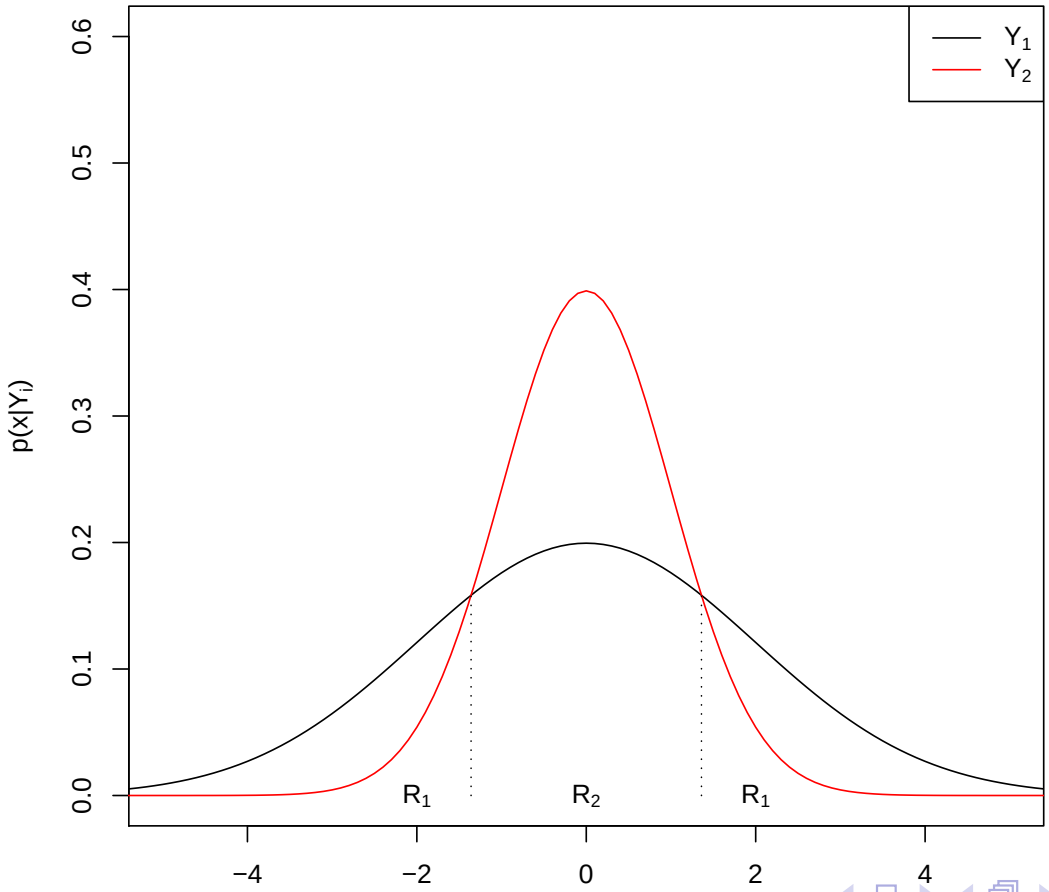
With the corresponding decision rule:

$$y = \begin{cases} Y_1 & \text{if } g(\mathbf{x}) > 0 \\ Y_2 & \text{otherwise} \end{cases}$$

- Commonly used dichotomizing discriminants under the minimum error rate criterion:
  - $g(\mathbf{x}) = P(Y_1|\mathbf{x}) - P(Y_2|\mathbf{x})$
  - $g(\mathbf{x}) = \ln p(\mathbf{x}|Y_1)/p(\mathbf{x}|Y_2) + \ln P(Y_1)/P(Y_2)$

# Decision regions and decision boundaries

- A decision rule divides the feature space into decision regions $\mathcal{R}_1, \ldots, \mathcal{R}_c$.
- If $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$ then $\mathbf{x}$ is in $\mathcal{R}_i$ (i.e. belongs to class $Y_i$)
- Regions are separated by decision boundaries: surfaces in feature space where discriminants are tied




# Outline



# Parameter estimation

- If we know the priors $P(Y_i)$ and class-conditional densities $p(\mathbf{x}|Y_i)$, the optimal classification is obtained using the Bayes decision rule
- In practice, those probabilities are almost never available
- Thus, we need to estimate them from training data
  - Priors are easy to estimate for typical classification problems
  - However, for class-conditional densities, training data is typically sparse!
- If we know (or assume) the general model structure, estimating the model parameters is more feasible
- For example, we assume that $p(\mathbf{x}|Y_i)$ is a normal density with mean $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$

# The normal density – univariate

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{7}$$

- Completely specified by two parameters, mean $\mu$ and variance $\sigma^2$

$$\mu \equiv \mathcal{E}[x] = \int_{-\infty}^{\infty} x p(x) dx$$

$$\sigma^2 \equiv \mathcal{E}[(x-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 p(x) dx$$

- Notation: $p(x) \sim N(\mu, \sigma^2)$

# Mulitvariate normal density

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right] \quad (8)$$

- Where $\boldsymbol{\mu}$ is the mean vector
- And $\Sigma$ is the covariance matrix

$$\Sigma \equiv \mathcal{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T] = \int (\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T p(\mathbf{x}) d\mathbf{x}$$

# Maximum likelihood estimation

- The density $p(\mathbf{x})$ conditioned on class $Y_i$ has a parametric form, and depends on $\boldsymbol{\theta}$
- The data set $\mathcal{D}$ consists of $n$ instances $\mathbf{x}_1, \ldots, \mathbf{x}_n$
- We assume the instances are chosen independently, thus the likelihood of $\boldsymbol{\theta}$ with respect to the training instances is:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{k=1}^{n} p(\mathbf{x}_k|\boldsymbol{\theta})$$

- In maximum likelihood estimation we will find the $\boldsymbol{\theta}$ which maximizes this likelihood

# Problem – MLE for the mean of univariate normal density

- Let $\theta = \mu$
- Maximizing log likelihood is equivalent to maximizing likelihood (but tends to be analytically easier)

$$l(\theta) \equiv \ln p(\mathcal{D}|\theta)$$

$$l(\theta) = \sum_{k=1}^{n} \ln p(x_k|\theta)$$

- Our solution is the argument which maximizes this function

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, l(\theta)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{k=1}^{n} \ln p(x_k|\theta)$$

# Normal density – log likelihood

$$p(x|\theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2\right]$$

$$\ln p(x|\theta) = \ln \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2\right] \tag{9}$$

$$= -\frac{1}{2}\ln 2\pi\sigma + \frac{1}{2\sigma^2}(x_k - \theta)^2 \tag{10}$$

We will find the maximum of the log-likelihood function by finding the point where the first derivative $= 0$

$$\frac{d}{d\theta}\sum_{k=1}^{n} \ln p(x_k|\theta) = 0$$

- Substituting the log likelihood we get

$$\frac{d}{d\theta}\sum_{k=1}^{n} -\frac{1}{2}\ln 2\pi\sigma + \frac{1}{2\sigma^2}(x_k-\theta)^2 = 0 \quad (11)$$

$$\sum_{k=1}^{n}\frac{d}{d\theta} -\frac{1}{2}\ln 2\pi\sigma + \frac{1}{2\sigma^2}(x_k-\theta)^2 = 0 \quad (12)$$

$$\sum_{k=1}^{n} 0 + \frac{1}{\sigma^2}(x_k-\theta) = 0 \quad (13)$$

$$\frac{1}{\sigma^2}\sum_{k=1}^{n}(x_k-\theta) = 0 \quad (14)$$

$$\sum_{k=0}^{n}(x_k-\theta) = 0 \quad (15)$$

- Finally we rearrange to get $\theta$

$$\sum_{k=1}^{n}(x_k - \theta) = 0 \tag{16}$$

$$-n\theta + \sum_{k=1}^{n} x_k = 0 \tag{17}$$

$$-n\theta = -\sum_{k=1}^{n} x_k \tag{18}$$

$$\theta = \frac{1}{n}\sum_{k=1}^{n} x_k \tag{19}$$

- Which is the ... mean of the training examples

# MLE for variance and for multivariate Gaussians

- Using a similar approach, we can derive the MLE estimate for the variance of univariate normal density

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k - \hat{\mu})^2$$

- For the multivariate case, the MLE estimates are as follows

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_k$$

$$\hat{\Sigma} = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{x}_k - \hat{\boldsymbol{\mu}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}})^T$$

# Exercise – Bayes classifier

In this exercise the goal is to use the Bayes classifier to distinguish between examples of two different species of iris.
We will use the parameters derived from the training examples using ML estimates.

# Outline



# Non-parametric techniques

- In many (most) cases assuming the examples come from a parametric distribution is not valid
- Non-parametric technique don't make the assumption that the form of the distribution is known
  - Density estimation – Parzen windows
  - Use training examples to derive decision functions directly: K-nearest neighbors, decision trees
  - Assume a known form for discriminant functions, and estimate their parameters from training data (e.g. linear models)

# KNN classifier

## K-Nearest neighbors idea

When classifying a new example, find $k$ nearest training example, and assign the majority label

- Also known as
  - Memory-based learning
  - Instance or exemplar based learning
  - Similarity-based methods
  - Case-based reasoning

# Distance metrics in feature space

- Euclidean distance or $L_2$ norm in $d$ dimensional space:

$$D(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}$$

- $L_1$ norm (Manhattan or taxicab distance)

$$L_1(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} |x_i - x_i'|$$

- $L_\infty$ or maximum norm

$$L_\infty(\mathbf{x}, \mathbf{x}') = \max_{i=1}^{d} |x_i - x_i'|$$

- In general, $L_k$ norm:

$$L_k(\mathbf{x}, \mathbf{x}') = \left( \sum_{i=1}^{d} |x_i - x_i'|^k \right)^{1/k}$$

# Hamming distance

- Hamming distance used to compare strings of symbolic attributes
- Equivalent to $L_1$ norm for binary strings
- Defines the distance between two instances to be the sum of per-feature distances
- For symbolic features the per-feature distance is 0 for an exact match and 1 for a mismatch.

$$\text{Hamming}(x, x') = \sum_{i=1}^{d} \delta(x_i, x'_i) \tag{20}$$

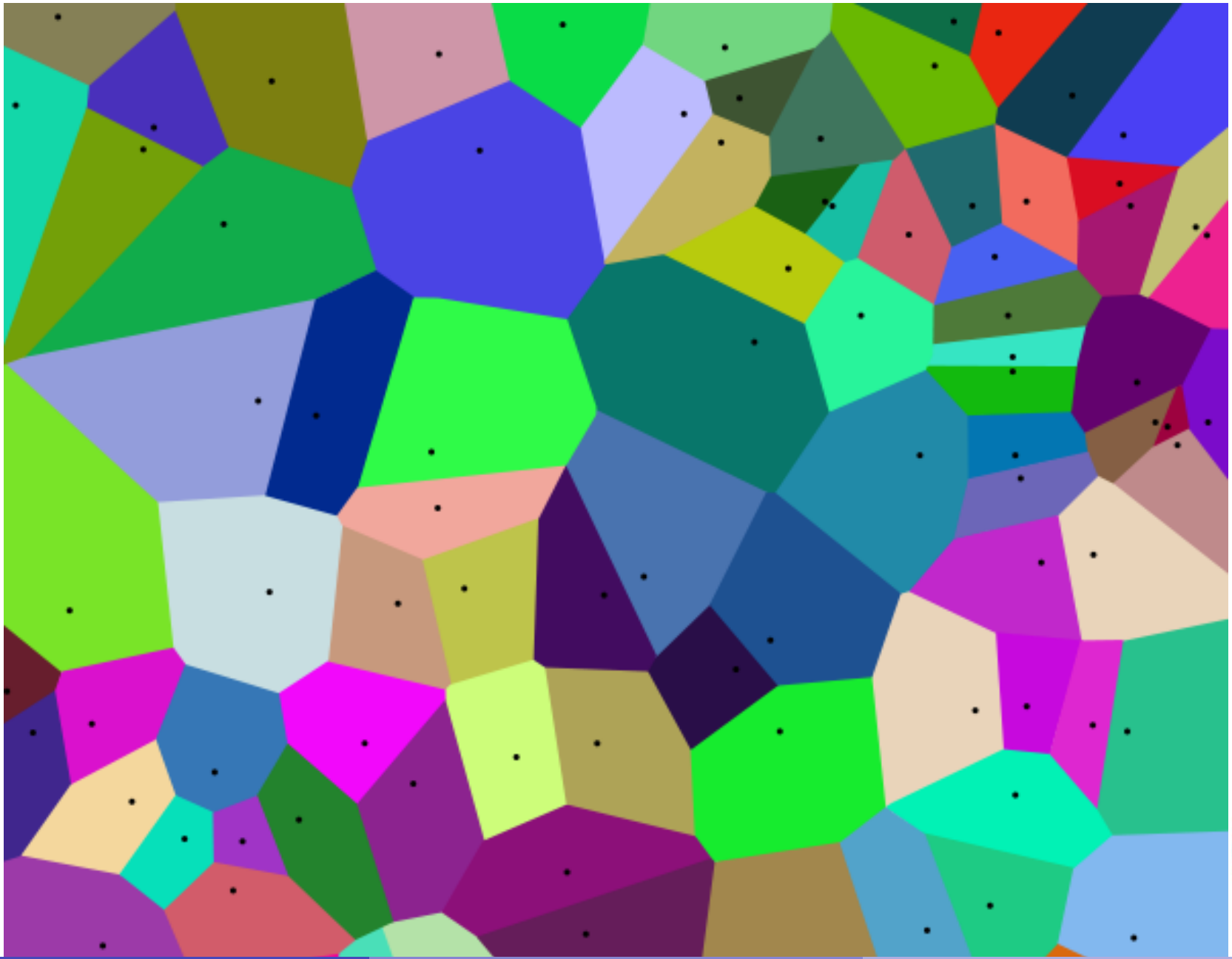
$$\delta(x_i, x'_i) = \begin{cases} 0 & \text{if } x_i = x'_i \\ 1 & \text{if } x_i \neq x'_i \end{cases} \tag{21}$$

# IB1 algorithm

For a vector with a mixture of symbolic and numeric values, the above definition of per feature distance is used for symbolic features, while for numeric ones we can use the scaled absolute difference

$$\delta(x_i, x_i') = \frac{x_i - x_i'}{max_i - min_i}. \tag{22}$$

# Nearest-neighbor Voronoi tessalation



# IB1 with feature weighting

- The per-feature distance is multiplied by the weight of the feature for which it is computed:

$$D_{\mathbf{w}}(x, x') = \sum_{i=1}^{d} w_i \delta(x_i, x'_i) \tag{23}$$

  where $w_i$ is the weight of the $i^{\text{th}}$ feature.
- [Daelemans and van den Bosch, 2005] describe two entropy-based methods and a $\chi^2$-based method to find a good weight vector $\mathbf{w}$.

# Information gain

A measure of how much knowing the value of a certain feature for an example decreases our uncertainty about its class, i.e. difference in class entropy with and without information about the feature value.

$$w_i = H(Y) - \sum_{v \in V_i} P(v) H(Y|v) \tag{24}$$

where

- $w_i$ is the weight of the $i^{\text{th}}$ feature
- $Y$ is the set of class labels
- $V_i$ is the set of possible values for the $i^{\text{th}}$ feature
- $P(v)$ is the probability of value $v$
- class entropy $H(Y) = -\sum_{y \in Y} P(y) \log_2 P(y)$
- $H(Y|v)$ is the conditional class entropy given that feature value $= v$

Numeric values need to be temporarily discretized for this to work

# Gain ratio

IG assigns excessive weight to features with a large number of values.
To remedy this bias information gain can be normalized by the entropy of the feature values, which gives the gain ratio:

$$w_i = \frac{H(Y) - \sum_{v \in V_i} P(v) H(Y|v)}{H(V_i)} \tag{25}$$

For a feature with a unique value for each instance in the training set, the entropy of the feature values in the denominator will be maximally high, and will thus give it a low weight.

# $\chi^2$

The $\chi^2$ statistic for a problem with $k$ classes and $m$ values for feature $F$:

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{m} \frac{(E_{ij} - O_{ij})^2}{E_{ij}} \tag{26}$$

where

- $O_{ij}$ is the observed number of instances with the $i^{\text{th}}$ class label and the $j^{\text{th}}$ value of feature $F$
- $E_{ij}$ is the expected number of such instances in case the null hypothesis is true: $E_{ij} = \frac{n_{\cdot j} n_{i\cdot}}{n_{\cdot\cdot}}$
- $n_{ij}$ is the frequency count of instances with the $i^{\text{th}}$ class label and the $j^{\text{th}}$ value of feature $F$
  - $n_{\cdot j} = \sum_{i=1}^{k} n_{ij}$
  - $n_{i\cdot} = \sum_{j=1}^{m} n_{ij}$
  - $n_{\cdot\cdot} = \sum_{i=1}^{k} \sum_{j=0}^{m} n_{ij}$

# $\chi^2$ example

- Consider a spam detection task: your features are words present/absent in email messages
- Compute $\chi^2$ for each word to use a weightings for a KNN classifier
- The statistic can be computed from a contingency table. Eg. those are (fake) counts of **rock-hard** in 2000 messages

| | rock-hard | $\neg$ rock-hard |
|---|---|---|
| ham | 4 | 996 |
| spam | 100 | 900 |

We need to sum $(E_{ij} - O_{ij})^2/E_{ij}$ for the four cells in the table:

$$\frac{(52-4)^2}{52} + \frac{(948-996)^2}{948} + \frac{(52-100)^2}{52} + \frac{(948-900)^2}{948} = 93.4761$$

# Problem – IG from contingency table

Use the contingency table in the previous example to compute:

- Information gain
- Gain ratio

for this the feature **rock-hard** with respect to the classes spam and ham.

# Distance-weighted class voting

- So far all the instances in the neighborhood are weighted equally for computing the majority class
- We may want to treat the votes from very close neighbors as more important than votes from more distant ones
- A variety of distance weighting schemes have been proposed to implement this idea; see [Daelemans and van den Bosch, 2005] for details and discussion.

# KNN – summary

- Non-parametric: makes no assumptions about the probability distribution the examples come from
- Does not assume data is linearly separable
- Derives decision rule directly from training data
- "Lazy learning":
  - During learning little "work" is done by the algorithm: the training instances are simply stored in memory in some efficient manner.
  - During prediction the test instance is compared to the training instances, the neighborhood is calculated, and the majority label assigned
- No information discarded: "exceptional" and low frequency training instances are available for prediction

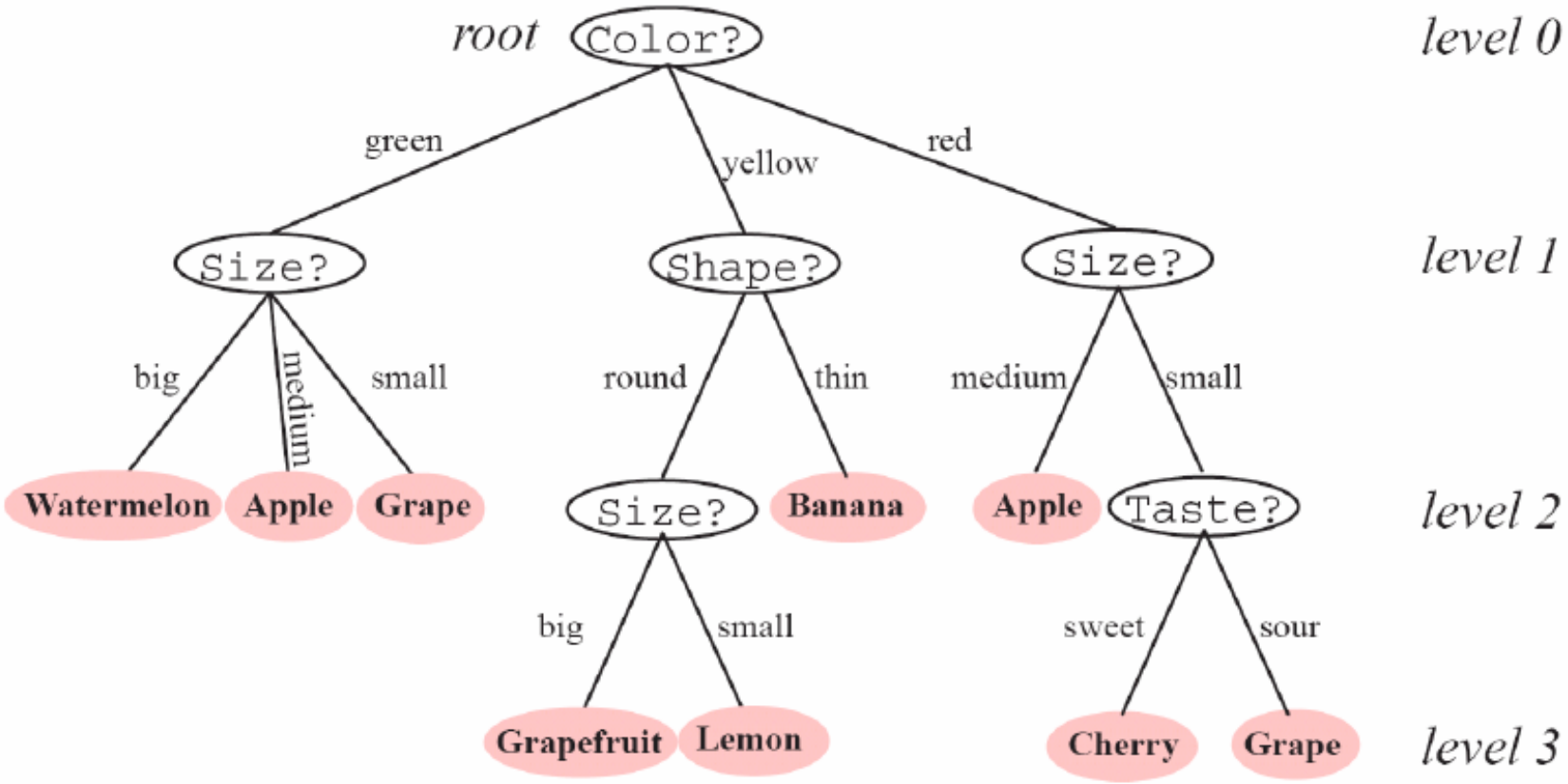
# Outline



# Decision trees

- "Nonmetric method"
- No numerical vectors of features needed
- Just attribute-value lists
- Ask a question about an attribute to partition the set of objects
- Resulting classification tree easy to interpret!

# Decision trees




# Outline



# Linear models

- With linear classifiers we assume the form of the discriminant function to be known, and use training data to estimate its parameters
- No assumptions about underlaying probability distribution – in this limited sense they are non-parametric
- Learning a linear classifier formulated as minimizing the criterion function

# Linear discriminant function

- A discriminant linear in the components of $\mathbf{x}$ has the following form:

$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{27}$$

$$g(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i + b \tag{28}$$

- Here $\mathbf{w}$ is the weight vector and $b$ is the bias, or threshold weight
- This function is a weighted sum of the components of $\mathbf{x}$ (shifted by the bias)
- For a binary classification problem, the decision function becomes:

$$f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} Y_1 \text{ if } g(\mathbf{x}) > 0 \\ Y_2 \text{ otherwise} \end{cases}$$

# Linear decision boundary

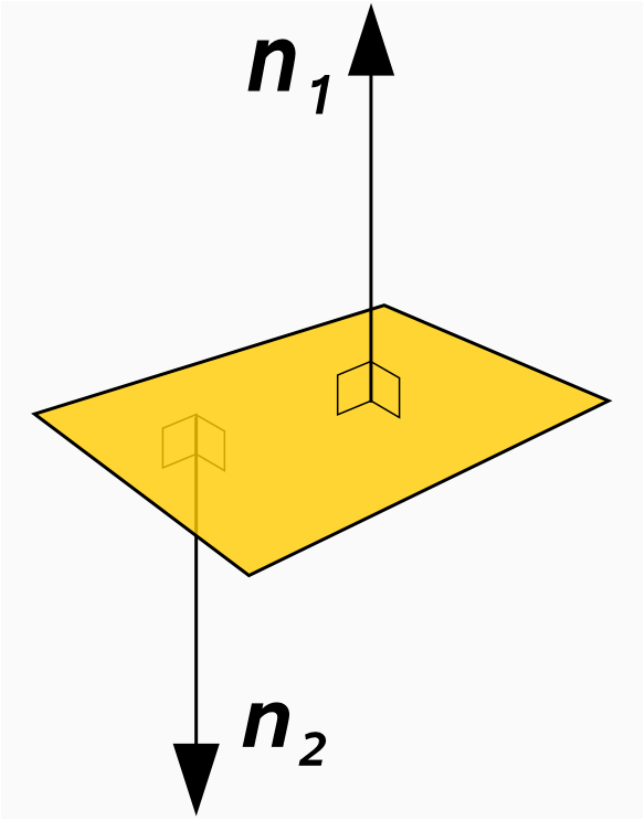
- A hyperplane is a generalization of a straight line to $> 2$ dimensions
- A hyperplane contains all the points in a $d$ dimensional space satisfying the following equation:

$$a_1 x_1 + a_2 x_2, \ldots, + a_d x_d + b = 0$$

- By identifying the components of $\mathbf{w}$ with the coefficients $a_i$, we can see how the weight vector and the bias define a linear decision surface in $d$ dimensions
- Such a classifier relies on the examples being linearly separable

# Normal vector

- Geometrically, the weight vector **w** is a normal vector of the separating hyperplane
- A normal vector of a surface is any vector which is perpendicular to it



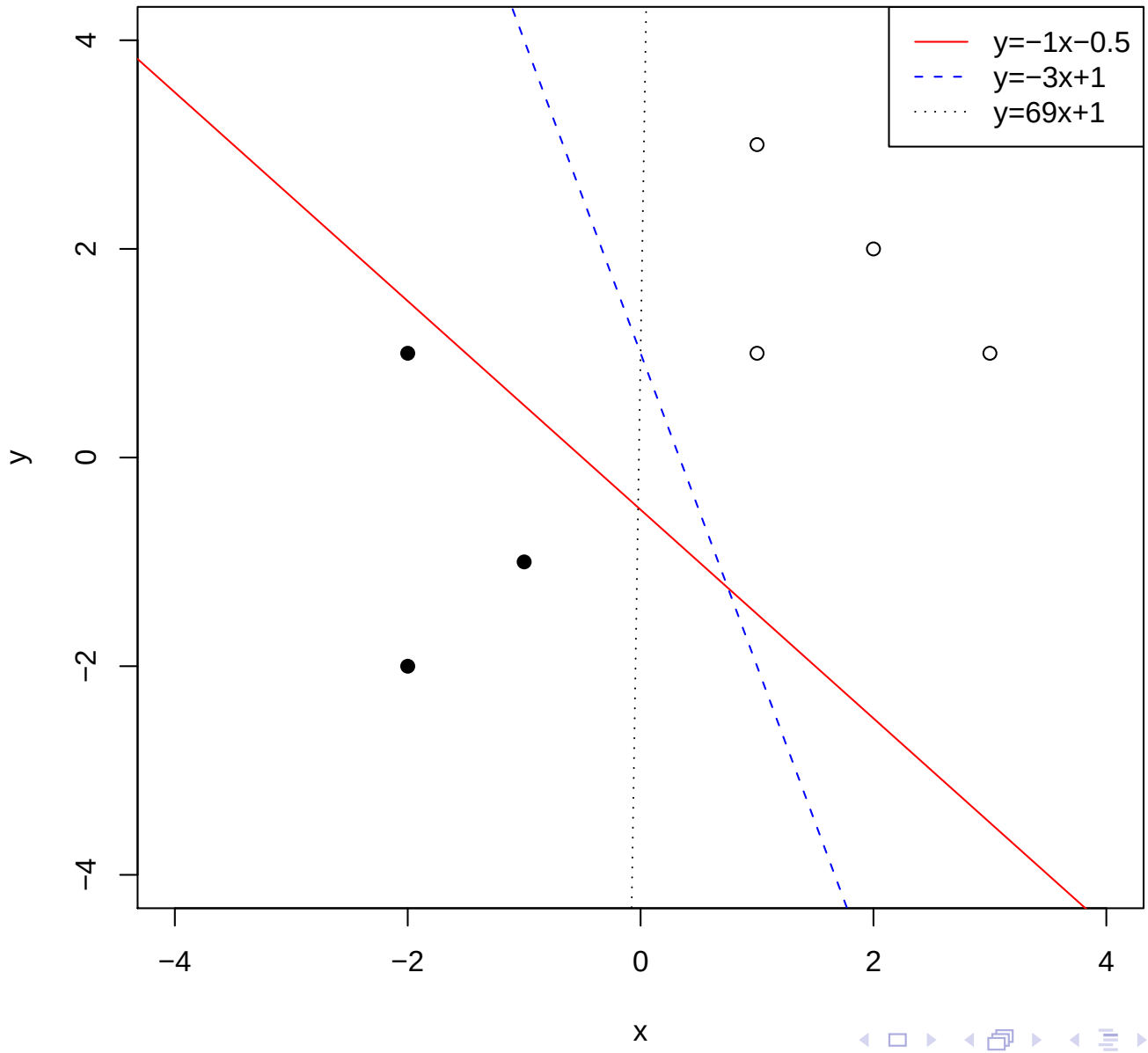

# Bias

- The orientation (or slope) of the hyperplane is determined by $\mathbf{w}$ while the location (intercept) is determined by the bias
- It is common to simplify notation by including the bias in the weight vector, i.e. $b = w_0$
- We need to add an additional component to the feature vector $\mathbf{x}$
- This component is always $x_0 = 1$
- The discriminant function is then simply the dot product between the weight vector and the feature vector:

$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \tag{29}$$

$$g(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i \tag{30}$$

# Separating hyperplanes in 2 dimensions




# Perceptron training

- How do we find a set of weights that separate our classes?
- **Perceptron**: A simple mistake-driven online algortihm

  - Start with a zero weight vector and process each training example in turn.
  - If the current weight vector classifies the current example incorrectly, move the weight vector in the right direction.
  - If weights stop changing, stop

- If examples are linearly separable, then this algorithm is guaranteed to converge on the solution vector

# Fixed increment online perceptron algorithm

- Binary classification, with classes $+1$ and $-1$
- Decision function $y' = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b)$

**PERCEPTRON**$(x^{1:N}, y^{1:N}, I)$:

1: $\mathbf{w} \leftarrow \mathbf{0}$
2: $b \leftarrow 0$
3: **for** $i = 1 \ldots I$ **do**
4:  **for** $n = 1 \ldots N$ **do**
5:  **if** $y^{(n)}(\mathbf{w} \cdot \mathbf{x}^{(n)} + b) \leq 0$ **then**
6:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(n)}\mathbf{x}^{(n)}$
7:  $b \leftarrow b + y^{(n)}$
8: **return** $(\mathbf{w}, b)$

# Weight averaging

- Although the algorithm is guaranteed to converge, the solution is not unique!
- Sensitive to the order in which examples are processed
- Separating the training sample does not equal good accuracy on unseen data
- Empirically, better generalization performance with weight averaging
  - A method of avoiding overfitting
  - As final weight vector, use the mean of all the weight vector values for each step of the algorithm

# Efficient averaged perceptron algorithm

**PERCEPTRON**$(x^{1:N}, y^{1:N}, I)$:

$\mathbf{w} \leftarrow \mathbf{0}$ ; $\mathbf{w_a} \leftarrow \mathbf{0}$
$b \leftarrow 0$ ; $b_a \leftarrow 0$
$c \leftarrow 1$
**for** $i = 1 \ldots I$ **do**
  **for** $n = 1 \ldots N$ **do**
    **if** $y^{(n)}(\mathbf{w} \cdot \mathbf{x}^{(n)} + b) \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y^{(n)}\mathbf{x}^{(n)}$ ; $b \leftarrow b + y^{(n)}$
      $\mathbf{w_a} \leftarrow \mathbf{w_a} + cy^{(n)}\mathbf{x}^{(n)}$ ; $b_a \leftarrow b_a + cy^{(n)}$
    $c \leftarrow c + 1$
**return** $(\mathbf{w} - \mathbf{w_a}/c, b - b_a/c)$

# Problem: Average perceptron

## Weight averaging

Show that the above algorithm performs weight averaging.
Hints:

- In the standard perceptron algorithm, the final weight vector (and bias) is the sum of the updates at each step.
- In average perceptron, the final weight vector should be the mean of the sum of partial sums of updates at each step

# Solution

Let's formalize:

- Basic perceptron: final weights are the sum of updates at each step:

$$\mathbf{w} = \sum_{i=1}^{n} f(\mathbf{x}^{(i)}) \tag{31}$$

# Solution

Let's formalize:

- Basic perceptron: final weights are the sum of updates at each step:

$$\mathbf{w} = \sum_{i=1}^{n} f(\mathbf{x}^{(i)}) \tag{31}$$

- Naive weight averaging: final weights are the mean of the sum of partial sums:

$$\mathbf{w} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{i} f(\mathbf{x}^{(j)}) \tag{32}$$

# Solution

Let's formalize:

- Basic perceptron: final weights are the sum of updates at each step:

$$\mathbf{w} = \sum_{i=1}^{n} f(\mathbf{x}^{(i)}) \tag{31}$$

- Naive weight averaging: final weights are the mean of the sum of partial sums:

$$\mathbf{w} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{i} f(\mathbf{x}^{(j)}) \tag{32}$$

- Efficient weight averaging:

$$\mathbf{w} = \sum_{i=1}^{n} f(\mathbf{x}^{(i)}) - \sum_{i=1}^{n} if(\mathbf{x}^{(i)})/n \tag{33}$$

- Show that equations 32 and 33 are equivalent. Note that we can rewrite the sum of partial sums by multiplying the update at each step by the factor indicating in how many of the partial sums it appears

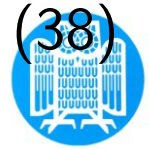
$$\mathbf{w} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} f(\mathbf{x}^{(j)}) \tag{34}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(n-i)f(\mathbf{x}^{(i)}) \tag{35}$$

$$= \frac{1}{n}\left[\sum_{i=1}^{n} nf(\mathbf{x}^{(i)}) - if(\mathbf{x}^{(i)})\right] \tag{36}$$

$$= \frac{1}{n}\left[n\sum_{i=1}^{n} f(\mathbf{x}^{(i)}) - \sum_{i=1}^{n} if(\mathbf{x}^{(i)})\right] \tag{37}$$

$$= \sum_{i=1}^{n} f(\mathbf{x}_i) - \sum_{i=1}^{n} if(\mathbf{x}^{(i)})/n \tag{38}$$



# Margins

- Intiuitively, not all solution vectors (and corresponding hyperplanes (23)) are equally good
- It makes sense for the decision boundary to be as far away from the training instances as possible
  - this improves the chance that if the position of the data points is slightly perturbed, the decision boundary will still be correct.
- Results from Statistical Learning Theory confirm these intuitions: maintaining large margins leads to small generalization error [Vapnik, 1995]

# Functional margin

The functional margin of an instance $(\mathbf{x}, y)$ with respect to some hyperplane $(\mathbf{w}, b)$ is defined to be

$$\gamma = y(\mathbf{w} \cdot \mathbf{x} + b) \tag{39}$$

- A large margin version of the Perceptron update:

$$\text{if } y(\mathbf{w} \cdot \mathbf{x} + b) \leq \theta \text{ then update}$$

  where $\theta$ is the threshold or margin parameter
- So an update is made not only on incorrectly classified examples, but also on those classified with insufficient confidence.
- Max-margin algorithms maximize the minimum margin between the decision boundary and the training set (cf. SVM)

# Perceptron – dual formulation

- We noticed earlier that the weight vector ends up being a linear combination of training examples. Instantiating the update function $f(\mathbf{x}^{(i)})$:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y \mathbf{x}^{(i)}$$

  where $\alpha_i = 1$ if the $i^{\text{th}}$ was misclassified, and $= 0$ otherwise.

- The discriminant function then becomes:

$$g(x) = \left( \sum_{i=1}^{n} \alpha_i y^{(i)} \mathbf{x}^{(i)} \right) \cdot \mathbf{x} + b \tag{40}$$

$$= \sum_{i=1}^{n} \alpha_i y^{(i)} \mathbf{x}^{(i)} \cdot \mathbf{x} + b \tag{41}$$

# Dual Perceptron training

**DualPerceptron**$(x^{1:N}, y^{1:N}, I)$:

1: $\alpha \leftarrow \mathbf{0}$
2: $b \leftarrow 0$
3: **for** $j = 1 \ldots I$ **do**
4:  **for** $k = 1 \ldots N$ **do**
5:  **if** $y^{(k)} \left[ \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(k)} + b \right] \leq 0$ **then**
6:  $\alpha_i \leftarrow \alpha_i + 1$
7:  $b \leftarrow b + y^{(k)}$
8: **return** $(\alpha, b)$

# Kernels

- Note that in the dual formulation there is no explicit weight vector: the training algorithm and the classification are expressed in terms of dot products between training examples and the test example
- We can generalize such dual algorithms to use Kernel functions
  - A kernel function can be thought of as dot product in some transformed feature space

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$$

    where the map $\phi$ projects the vectors in the original feature space onto the transformed feature space
  - It can also be thought of as a similarity function in the input object space

# Kernel – example

- Consider the following kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

$$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2 \tag{42}$$

$$= \left( \sum_{i=1}^{d} x_i z_i \right) \left( \sum_{i=1}^{d} x_i z_i \right) \tag{43}$$

$$= \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j z_i z_j \tag{44}$$

$$= \sum_{i,j=1}^{d} (x_i x_j)(z_i z_j) \tag{45}$$

# Kernel vs feature map

## Feature map $\phi$ corresponding to $K$ for $d = 2$ dimensions

$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{pmatrix} \tag{46}$$

- Computing feature map $\phi$ explicitly needs $\mathcal{O}(d^2)$ time
- Computing $K$ is linear $\mathcal{O}(d)$ in the number of dimensions

# Why does it matter

- If you think of features as binary indicators, then the quadratic kernel above creates feature conjunctions
- E.g. in NER if $x_1$ indicates that word is capitalized and $x_2$ indicates that the previous token is a sentence boundary, with the quadratic kernel we efficiently compute the feature that both conditions are the case.
- Geometric intuition: mapping points to higher dimensional space makes them easier to separate with a linear boundary

# Separability in 2D and 3D

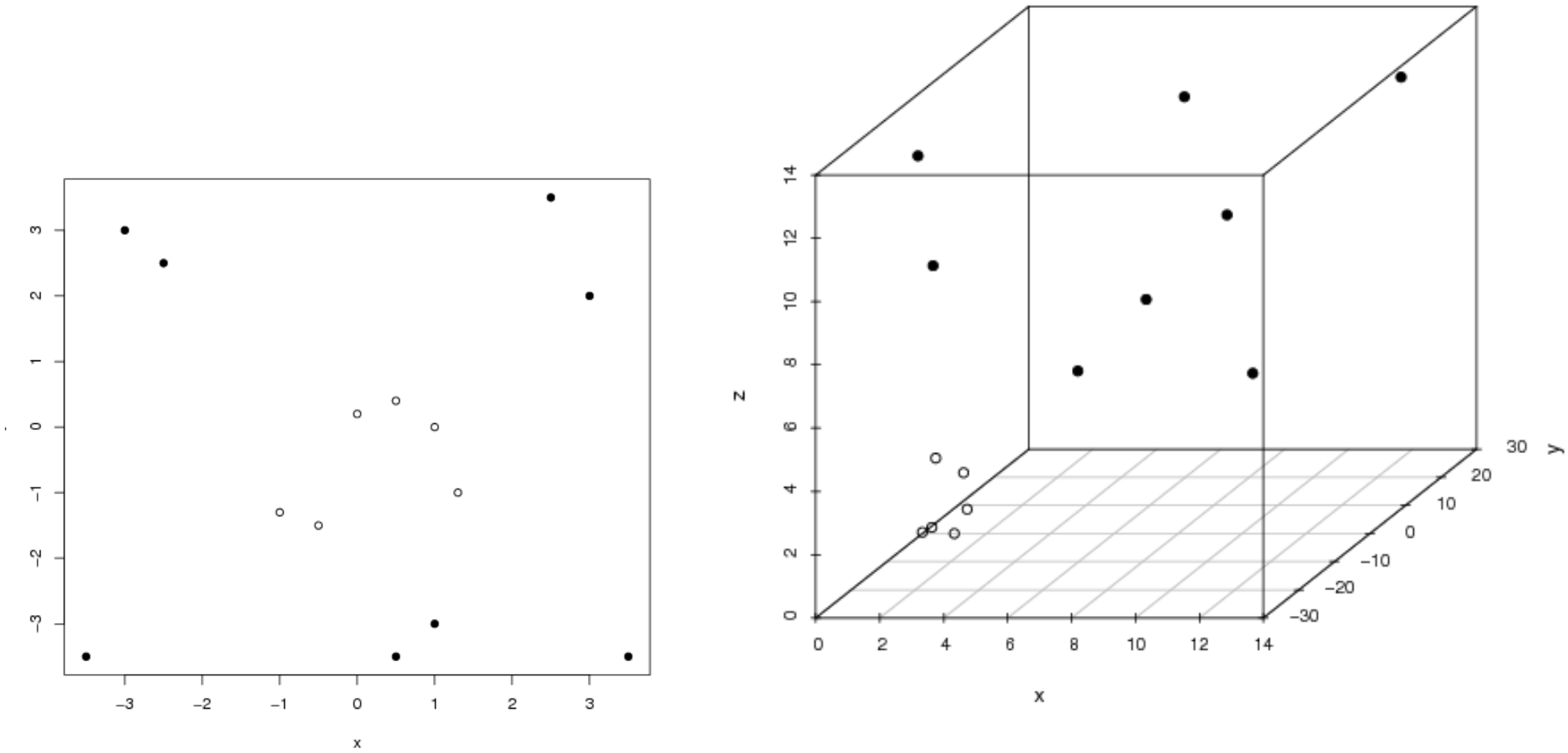



Figure: Two dimensional classification example, non-separable in two dimensions, becomes separable when mapped to 3 dimensions by $(x_1, x_2) \mapsto (x_1^2, 2x_1x_2, x_2^2)$

# Support Vector Machines

- The two key ideas, large margin, and the "kernel trick" come together in Support Vector Machines
- Margin: a decision boundary which is as far away from the training instances as possible improves the chance that if the position of the data points is slightly perturbed, the decision boundary will still be correct.
- Results from Statistical Learning Theory confirm these intuitions: maintaining large margins leads to small generalization error [Vapnik, 1995].
- A perceptron algorithm finds any hyperplane which separates the classes: SVM finds the one that additionally has the maximum margin

# Quadratic optimization formulation

- Functional margin can be made larger just by rescaling the weights by a constant
- Hence we can fix the functional margin to be 1 and minimize the norm of the weight vector
- This is equivalent to maximizing the geometric margin

For linearly separable training instances $((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$ find the hyperplane $(\mathbf{w}, b)$ that solves the optimization problem:

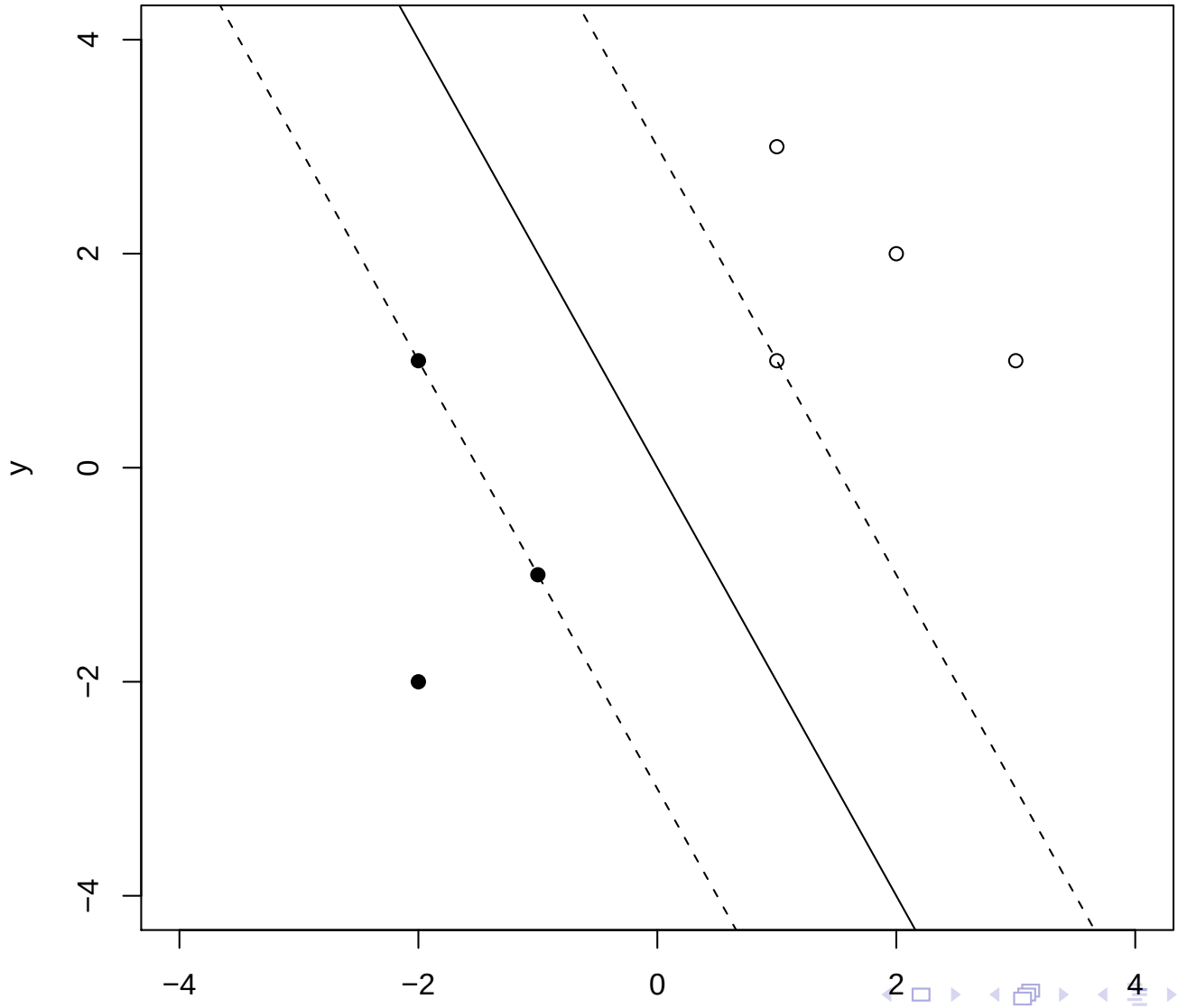
$$\begin{aligned} &\text{minimize}_{\mathbf{w},b} && \frac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to} && y_i(w \cdot \mathbf{x}_i + b) \geq 1 \quad \forall_{i \in 1..n} \end{aligned} \tag{47}$$

This hyperplane separates the examples with geometric margin $2/||\mathbf{w}||$

# Support vectors

- SVM finds a separating hyperplane with the largest margin to the nearest instance
- This has the effect that the decision boundary is fully determined by a small subset of the training examples (the nearest ones on both sides)
- Those instances are the support vectors

# Separating hyperplane and support vectors




# Soft margin

- SVM with soft margin works by relaxing the requirement that all data points lie outside the margin
- For each offending instance there is a "slack variable" $\xi_i$ which measures how much it would have move to obey the margin constraint.

$$\begin{aligned} \text{minimize}_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \forall_{i \in 1..n}\xi_i > 0 \end{aligned} \tag{48}$$

where

$$\xi_i = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$$

- The hyper-parameter $C$ trades off minimizing the norm of the weight vector versus classifying correctly as many examples as possible.
- As the value of $C$ tends towards infinity the soft-margin SVM approximates the hard-margin version.

# Dual form

The dual formulation is in terms of support vectors, where $SV$ is the set of their indices:

$$f(x, \alpha^*, b^*) = \text{sign}\left(\sum_{i \in SV} y_i \alpha_i^* (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right) \tag{49}$$

The weights in this decision function are the Lagrange multipliers $\alpha^*$.

$$\begin{aligned} &\text{minimize} \quad W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ &\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0 \qquad\qquad \forall_{i \in 1..n} \alpha_i \geq 0 \end{aligned} \tag{50}$$

The Lagrangian weights together with the support vectors determine $(\mathbf{w}, b)$:

$$\mathbf{w} = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i \tag{51}$$

$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k \text{ for any } k \text{ such that } \alpha_k \neq 0 \tag{52}$$

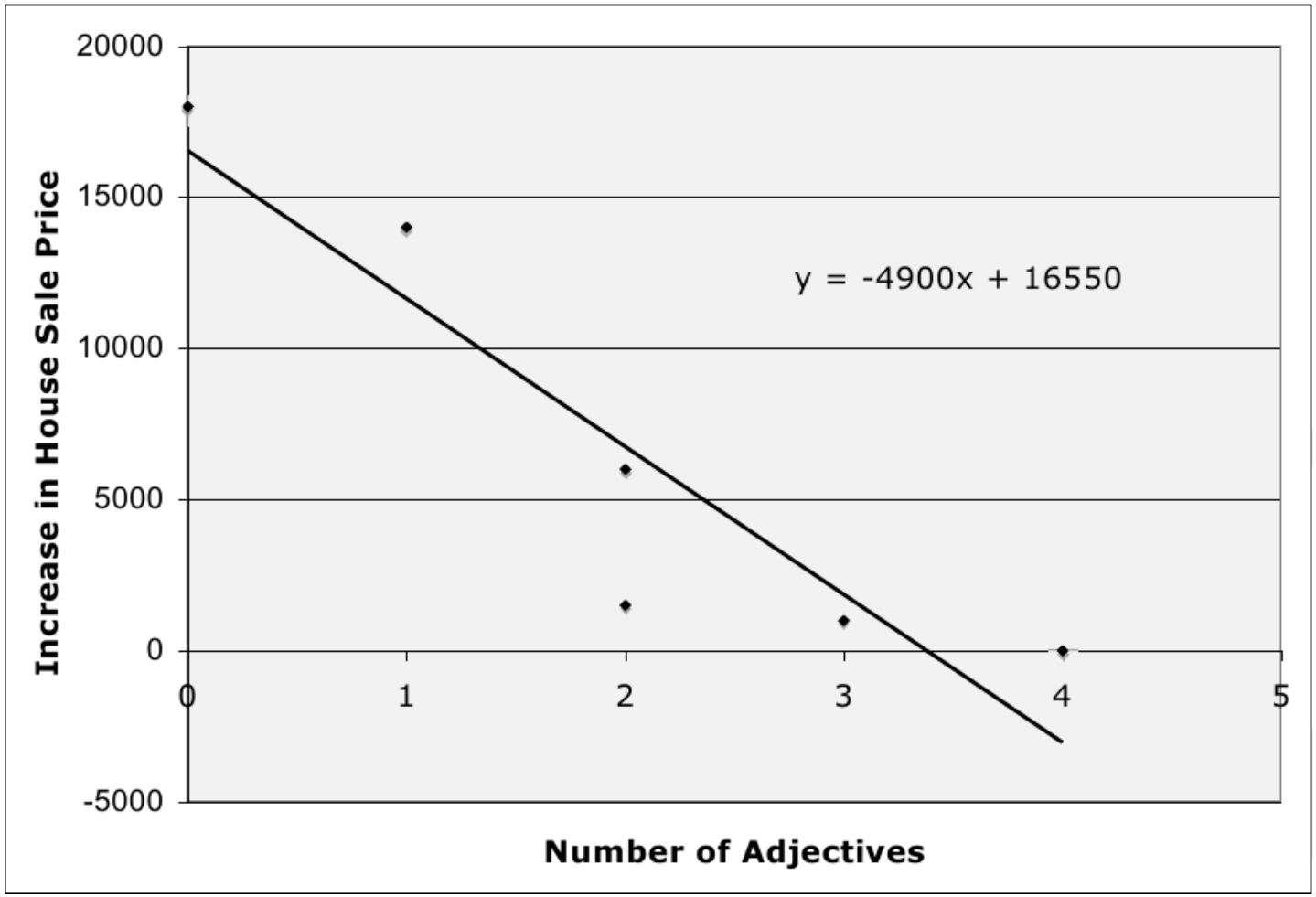
# Multiclass classification with SVM

- SVM is essentially a binary classifier.
- A common method to perform multiclass classification with SVM, is to train multiple binary classifiers and combine their predictions to form the final prediction. This can be done in two ways:
  - One-vs-rest (also known as one-vs-all): train $|Y|$ binary classifiers and choose the class for which the margin is the largest.
  - One-vs-one: train $|Y|(|Y|-1)/2$ pairwise binary classifiers, and choose the class selected by the majority of them.
- An alternative method is to make the weight vector and the feature function $\Phi$ depend on the output $y$, and learn a single classifier which will predict the class with the highest score:

$$y = \operatorname*{argmax}_{y' \in Y} \mathbf{w} \cdot \Phi(x, y') + b \tag{53}$$

# Linear Regression

- Training data: observations paired with outcomes ($n \in \mathbb{R}$)
- Observations have features (predictors, typically also real numbers)
- The model is a regression line $y = ax + b$ which best fits the observations
  - $a$ is the slope
  - $b$ is the intercept
  - This model has two parameters (or weigths)
  - One feature $= x$
  - Example:
    - $x =$ number of vague adjectives in property descriptions
    - $y =$ amount house sold over asking price


20000
15000
10000
5000
0
-5000
Increase in House Sale Price
y = -4900x + 16550
0
1
2
3
4
5
Number of Adjectives

# Multiple linear regression

- More generally $y = w_0 + \sum_{i=0}^{N} w_i f_i$, where
  - $y$ = outcome
  - $w_0$ = intercept
  - $f_1..f_N$ = features vector and $w_1..w_N$ weight vector
- We ignore $w_0$ by adding a special $f_0$ feature, then the equation is equivalent to dot product: $y = \mathbf{w} \cdot \mathbf{f}$

# Learning linear regression

- Minimize sum squared error over the training set of $M$ examples

$$cost(W) = \sum_{j=0}^{M} (y_{pred}^{(j)} - y_{obs}^{(j)})^2$$

where

$$y_{pred}^{j} = \sum_{i=0}^{N} w_i f_i^{(j)}$$

- Closed-form formula for choosing the best set of weights $W$ is given by:

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

where the matrix $X$ contains training example features, and $\overrightarrow{y}$ is the vector of outcomes.

# Logistic regression

- In logistic regression we use the linear model to do classification, i.e. assign probabilities to class labels
- For binary classification, predict $p(y = true|x)$. But predictions of linear regression model are $\in \mathbb{R}$, whereas $p(y = true|x) \in [0, 1]$
- Instead predict logit function of the probability:

$$ln\left(\frac{p(y = true|x)}{1 - p(y = true|x)}\right) = \mathbf{w} \cdot \mathbf{f} \tag{54}$$

$$\frac{p(y = true|x)}{1 - p(y = true|x)} = e^{\mathbf{w}\cdot\mathbf{f}} \tag{55}$$

- Solving for $p(y = true|x)$ we obtain:

$$p(y = true|x) = \frac{e^{\mathbf{w}\cdot\mathbf{f}}}{1 + e^{\mathbf{w}\cdot\mathbf{f}}} \tag{56}$$

$$= \frac{exp\left(\sum_{i=0}^{N} w_i f_i\right)}{1 + exp\left(\sum_{i=0}^{N} w_i f_i\right)} \tag{57}$$

# Logistic regression - classification

- Example $x$ belongs to class *true* if:

$$\frac{p(y = true|x)}{1 - p(y = true|x)} > 1 \tag{58}$$

$$e^{\mathbf{w}\cdot\mathbf{f}} > 1 \tag{59}$$

$$\mathbf{w} \cdot \mathbf{f} > 0 \tag{60}$$

$$\sum_{i=0}^{N} w_i f_i > 0 \tag{61}$$

- The equation $\sum_{i=0}^{N} w_i f_i = 0$ defines the hyperplane in $N$-dimensional space, with points above this hyperplane belonging to class *true*

# Logistic regression - learning

- Conditional likelihood estimation: choose the weights which make the probability of the observed values $y$ be the highest, given the observations $x$
- For the training set with $M$ examples:

$$\hat{\mathbf{w}} = \underset{w}{\operatorname{argmax}} \prod_{i=0}^{M} P(y^{(i)}|x^{(i)}) \tag{62}$$

- A problem in convex optimization
  - L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno method)
  - gradient ascent
  - conjugate gradient
  - iterative scaling algorithms

# Maximum Entropy model

- Logistic regression with more than two classes = multinomial logistic regression
- Also known as Maximum Entropy (MaxEnt)
- The MaxEnt equation generalizes (57) above:

$$p(c|x) = \frac{exp\left(\sum_{i=0}^{N} w_{ci} f_i\right)}{\sum_{c' \in C} exp\left(\sum_{i=0}^{N} w_{c'i} f_i\right)} \tag{63}$$

- The denominator is the normalization factor usually called $Z$ used to make the score into a proper probability distribution

$$p(c|x) = \frac{1}{Z} exp \sum_{i=0}^{N} w_{ci} f_i$$

# MaxEnt features

- In Maxent modeling normally binary freatures are used
- Features depend on classes: $f_i(c, x) \in \{0, 1\}$
- Those are indicator features
- Example $x$:
  **Secretariat/NNP is/BEZ expected/VBN to/TO race/VB tomorrow**
- Example features:

$$f_1(c, x) = \begin{cases} 1 \text{ if } word_i = race \ \& \ c = \text{NN} \\ 0 \text{ otherwise} \end{cases}$$

$$f_2(c, x) = \begin{cases} 1 \text{ if } t_{i-1} = TO \ \& \ c = \text{VB} \\ 0 \text{ otherwise} \end{cases}$$

$$f_3(c, x) = \begin{cases} 1 \text{ if } suffix(word_i) = ing \ \& \ c = \text{VBG} \\ 0 \text{ otherwise} \end{cases}$$

# Binarizing features

- Example $x$:
  **Secretariat/NNP is/BEZ expected/VBN to/TO race/VB tomorrow**
- Vector of symbolic features of $x$:

| $word_i$ | suf | $tag_{i-1}$ | is-case($w_i$) |
|---|---|---|---|
| race | ace | TO | TRUE |

- Class-dependent indicator features of $x$:

| | $word_i$=race | suf=ing | suf=ace | $tag_{i-1}$=TO | $tag_{i-1}$=DT | is-lower($w_i$)=TRUE | ... |
|---|---|---|---|---|---|---|---|
| JJ | 0 | 0 | 0 | 0 | 0 | 0 | |
| VB | 1 | 0 | 1 | 1 | 0 | 1 | |
| NN | 0 | 0 | 0 | 0 | 0 | 0 | |
| ... | | | | | | | |

# Entia non sunt multiplicanda praeter necessitatem


# Maximum Entropy principle

## Jayes, 1957

...in making inferences on the basis of partial information we must use that probability distribution which has maximum entropy subject to whatever is known. This is the only unbiased assignment we can make; to use any other would amount to arbitrary assumption of information which by hypothesis we do not have.

# Entropy

- Out of all possible models, choose the simplest one consistent with the data (Occam's razor)
- Entropy of the distribution of discrete random variable $X$:

$$H(X) = -\sum_{x} P(X = x) log_2 P(X = x)$$

- The uniform distribution has the highest entropy
- Finding the maximum entropy distribution in the set $C$ of possible distributions

$$p* = \underset{p \in C}{\text{argmax}}\, H(p)$$

- Berger et al. (1996) showed that solving this optimization problem is equivalent to finding the multinomial logistic regression model whose weights maximize the likelihood of the training data.

# Maxent principle – simple example

- Find a Maximum Entropy probability distribution $p(a, b)$ where $a \in \{x, y\}$ and $b \in \{0, 1\}$
- The only thing we know are is the following constraint:
  - $p(x, 0) + p(y, 0) = 0.6$

| $p(a, b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | ? | ? | |
| $y$ | ? | ? | |
| total | 0.6 | | 1 |

# Maxent principle – simple example

- Find a Maximum Entropy probability distribution $p(a, b)$ where $a \in \{x, y\}$ and $b \in \{0, 1\}$
- The only thing we know are is the following constraint:
  - $p(x, 0) + p(y, 0) = 0.6$

| $p(a, b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | ? | ? | |
| $y$ | ? | ? | |
| total | 0.6 | | 1 |

| $p(a, b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | 0.3 | 0.2 | |
| $y$ | 0.3 | 0.2 | |
| total | 0.6 | | 1 |

# Constraints

- The constraints imposed on the probability model are encoded in the features: the expected value of each one of $I$ indicator features $f_i$ under a model $p$ should be equal to the expected value under the empirical distribution $\tilde{p}$ obtained from the training data:

$$\forall i \in I,\ \mathcal{E}_p[f_i] = \mathcal{E}_{\tilde{p}}[f_i] \tag{64}$$

- The expected value under the empirical distribution is given by:

$$\mathcal{E}_{\tilde{p}}[f_i] = \sum_x \sum_y \tilde{p}(x,y) f_i(x,y) = \frac{1}{N} \sum_j^N f_i(x_j, y_j) \tag{65}$$

- The expected value according to model $p$ is:

$$\mathcal{E}_p[f_i] = \sum_x \sum_y p(x,y) f_i(x,y) \tag{66}$$

# Approximation

- However, this requires summing over all possible object - class label pairs, which is in general not possible.
- Therefore the following standard approximation is used:

$$\mathcal{E}_p[f_i] = \sum_x \sum_y \tilde{p}(x) p(y|x) f_i(x,y) = \frac{1}{N} \sum_j^N \sum_y p(y|x_j) f_i(x_j,y) \quad (67)$$

  where $\tilde{p}(x)$ is the relative frequency of object $x$ in the training data
- This has the advantage that $\tilde{p}(x)$ for unseen events is 0.
- The term $p(y|x)$ is calculated according to Equation 63.

# Regularization

- Although the Maximum Entropy models are maximally uniform subject to the constraints, they can still overfit (too many features)
- Regularization relaxes the constraints and results in models with smaller weights which may perform better on new data.
- Instead of solving the optimization in Equation 62, here in log form:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}), \tag{68}$$

we solve instead the following modified problem:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}) + \alpha R(\mathbf{w}) \tag{69}$$

where $R$ is the regularizer used to penalize large weights [Jurafsky and Martin, 2008b].

# Gaussian smoothing

- We can use a regularizer which assumes that weight values have a Gaussian distribution centered on 0 and with variance $\sigma^2$.
- By multiplying each weight by a Gaussian prior we will maximize the following equation:

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}) - \sum_{j=0}^{d} \frac{w_j^2}{2\sigma_j^2} \tag{70}$$

  where $\sigma_j^2$ are the variances of the Gaussians of feature weights.
- This modification corresponds to using a maximum *a posteriori* rather than maximum likelihood model estimation
- Common to constrain all the weights to have the same global variance, which gives a single tunable algorithm parameter (can be found on held-out data)

# Outline



# Sequence labeling

- Assigning sequences of labels to sequences of some objects is a very common task (NLP, bioinformatics)
- In NLP
  - POS tagging
  - chunking (shallow parsing)
  - named-entity recognition
- In general, learn a function $h : \Sigma^* \longrightarrow \mathcal{L}^*$ to assign a sequence of labels from $\mathcal{L}$ to the sequence of input elements from $\Sigma$
- The most and easily tractable case: each element of the input sequence receives one label:

$$h : \Sigma^n \longrightarrow \mathcal{L}^n$$

- In cases where it does not naturally hold, such as chunking, we decompose the task so it is satisfied.
- IOB scheme: each element gets a label indicating if it is initial in chunk X (B-X), a non-initial in chunk X (I-X) or is outside of any chunk (O).

```
So so ADV O O
führte führen VVFIN I-VC O
die d ART I-NC O
Vergewaltigung Vergewaltigung NN I-NC O
deutscher deutsch ADJA B-NC I-MISC
Frauen Frau NN I-NC O
durch durch APPR I-PC O
Angehörige Angehörige|Angehöriger NN I-NC O
der d ART B-NC O
Roten Rote NN I-NC I-ORG
Armee Armee NN I-NC I-ORG
nicht nicht PTKNEG O O
allein allein ADV O O
```

# Local classifier

- The simplest approach to sequence labeling is to just use a regular multiclass classifier, and make a local decision for each word.
- Predictions for previous words can be used in predicting the current word
- This straightforward strategy can give surprisingly good results.

# HMMs and MEMMs

- HMM POS tagging model:

$$\hat{T} = \operatorname*{argmax}_T P(T|W) \tag{71}$$

$$= \operatorname*{argmax}_T P(W|T)P(T) \tag{72}$$

$$= \operatorname*{argmax}_T \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1}) \tag{73}$$

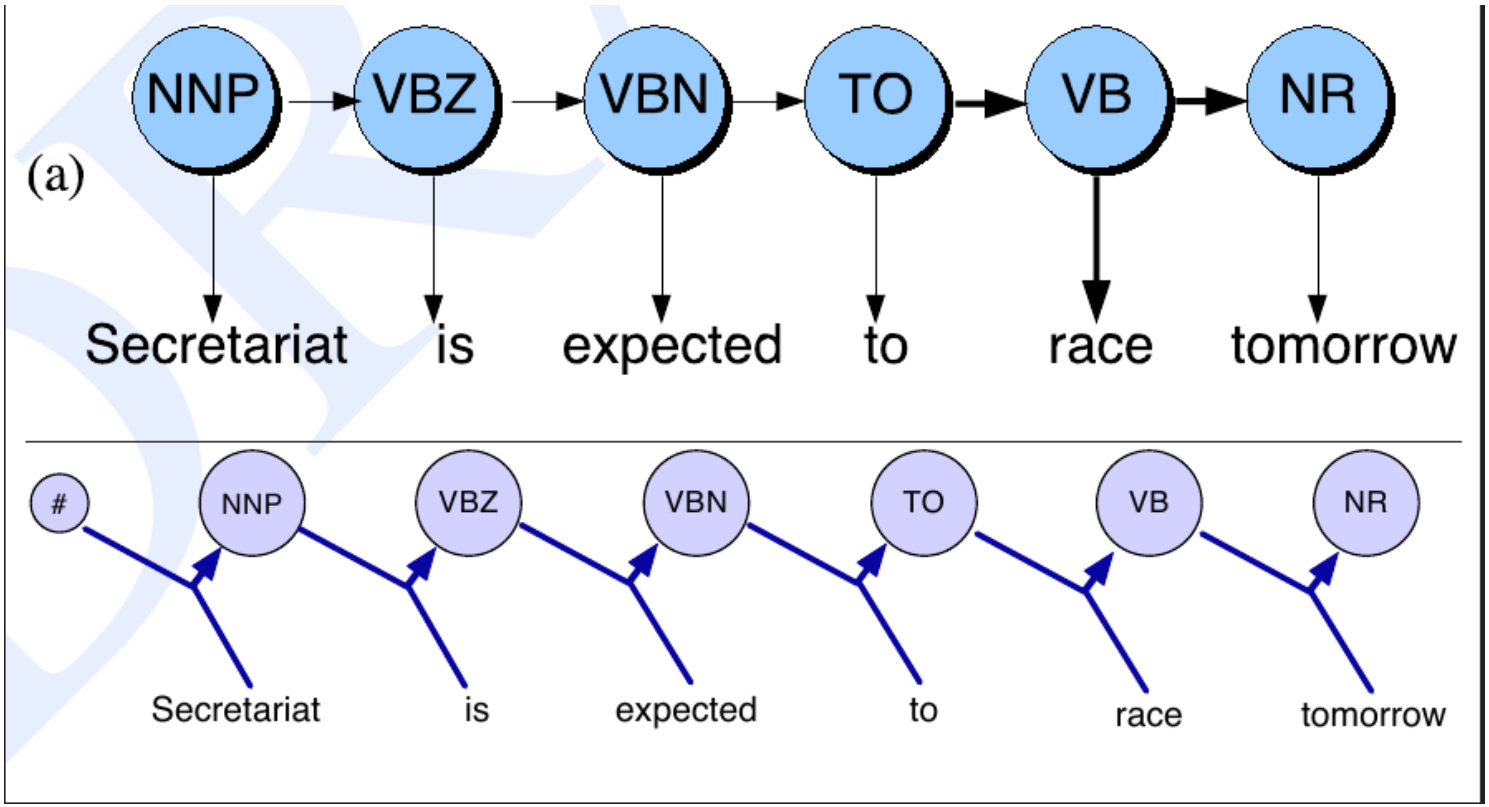
- MEMM POS tagging model:

$$\hat{T} = \operatorname*{argmax}_T P(T|W) \tag{74}$$

$$= \operatorname*{argmax}_T \prod_i P(tag_i|word_i, tag_{i-1}) \tag{75}$$

- Maximum entropy model gives conditional probabilities

# Conditioning probabilities in a HMM and a MEMM




# Viterbi in MEMMs

- Decoding works almost the same as in HMM
- Except entries in the DP table are values of $P(t_i|t_{i-1}, word_i)$
- Recursive step: Viterbi value of time $t$ for state $j$:

$$v_t(j) = \max_{i=i}^{N} v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T \tag{76}$$

# MaxEnt with beam search

$$P(t_1, ..., t_n | c_1, ..., c_n) = \prod_{i=1}^{n} P(t_i | c_i) \quad (77)$$

## Beam search

For each word $w_i$ the beam search algorithm maintains the $N$ (= beam size) highest scoring tag sequences for words $(w_1, ..., w_{i-1})$ up to the previous word.
Each of those label sequences is combined with the current word $w_i$ to create the context $c_i$, and the Maximum Entropy Model is used to obtain the $N$ probability distributions over tags for word $w_i$.
Now we find $N$ most likely sequences of tags up to and including word $w_i$ by using Equation 77, i.e. by multiplying the probability of the sequence of tags up to $w_{i-1}$ with the probability for the tag $t_i$ given the context formed using that sequence, and we proceed to word $w_{i+1}$ if $i \leq n$.

# Generic Perceptron

**Inputs**: Training examples $(x_i, y_i)$
**Initialization**: Set $\mathbf{w} = \mathbf{0}$
**Algorithm**:
  For $t = 1..T, i = 1..n$
    Calculate $y_i' = \text{argmax}_{y \in \text{GEN}(x_i)} \mathbf{w} \cdot \Phi(x_i, y)$
    If $y' \neq y$ then $\mathbf{w} = \mathbf{w} + \Phi(x_i, y_i) - \Phi(x_i, y_i')$
**Output**: Parameters $\mathbf{w}$

# Details

- Feature function $\Phi : X \times Y \rightarrow \mathbb{R}^d$
  - Classification $\Phi(\mathbf{x}, y) = (\phi(\mathbf{x})[y = Y_i])_{i=1}^{|Y|}$
  - Sequence labeling $\Phi(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \phi(x_i, y_i)$ where $n$ is the length of the sequence
- Decoding function (argmax)
  - Classification $\text{argmax}_{y \in Y} \mathbf{w} \cdot \Phi(x, y)$
  - Sequence labeling $\mathit{ViterbiPath}(\mathbf{x}; \mathbf{w})$ or $\mathit{BeamSearch}(\mathbf{x}; \mathbf{w})$ where the local score is $\mathbf{w} \cdot \phi(x_i, y_i)$

# Conditional Random Fields

- Alternative generalization of MaxEnt to structured prediction
- Main difference:
  - while MEMM uses per state exponential models for the conditional probabilities of next states given the current state
  - CRF has a single exponential model for the joint probability of the whole sequence of labels
- The formulation is:

$$p(y|x;\vec{w}) = \frac{1}{Z_{x;\vec{w}}} exp\ [\vec{w} \cdot \Phi(x,y)]$$

$$Z_{x;\vec{w}} = \sum_{y' \in \mathcal{Y}} exp\ [\vec{w} \cdot \Phi(x,y')]$$

# CRF - tractability

- Since the set $Y$ contains structures such as sequences, the challenge here is to compute the sum in the denominator.

$$Z_{x;\vec{w}} = \sum_{y' \in \mathcal{Y}} exp\ [\vec{w} \cdot \Phi(x, y')]$$

- [Lafferty et al., 2001] and [Sha and Pereira, 2003] show that given certain constraints on $Y$ and on $\Phi$ dynamic programming techniques can be used to compute it efficiently.
- Specifically $\Phi$ should obey the Markov property, i.e. no feature should depend on elements of $y$ that are more than the Markov length $l$ apart)

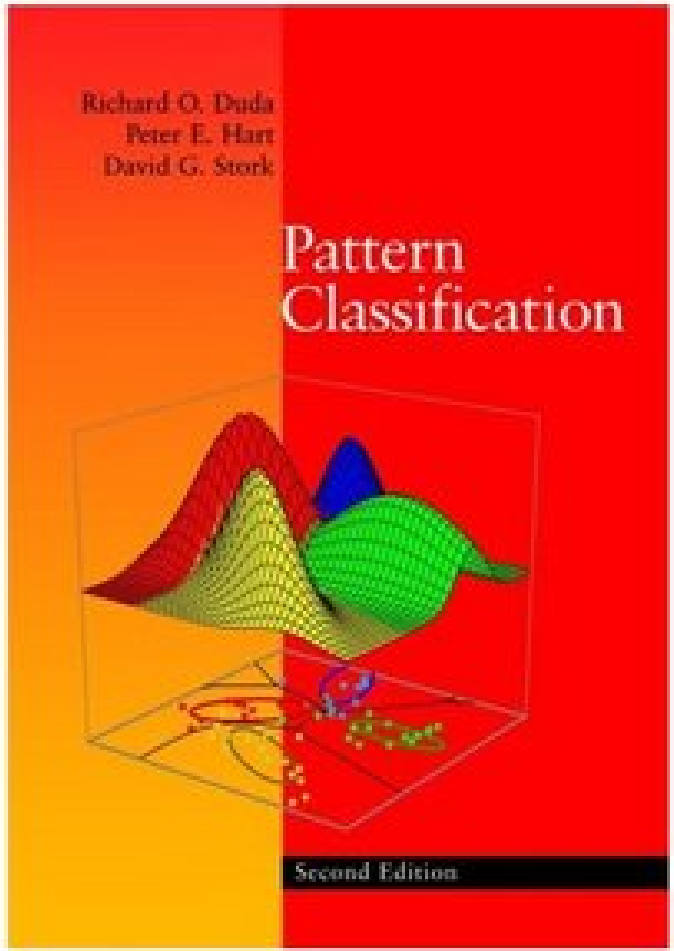
# CRF - experimental comparison to HMM and MEMMs

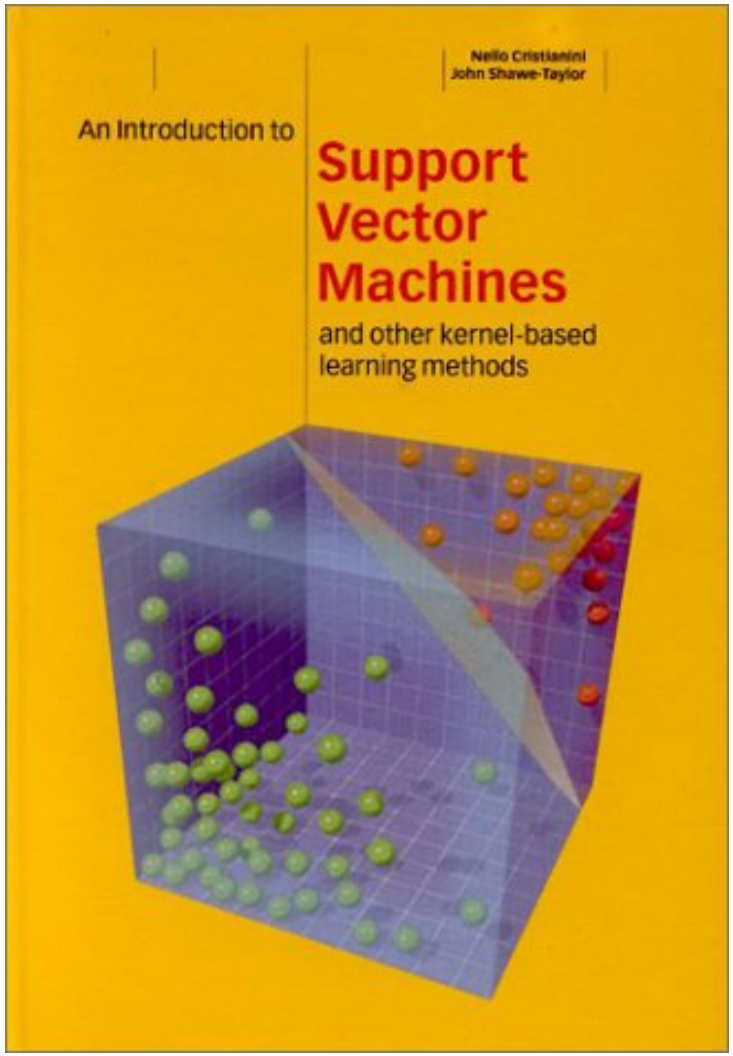
| *model* | *error* | *oov error* |
|---|---|---|
| HMM | 5.69% | 45.99% |
| MEMM | 6.37% | 54.61% |
| CRF | 5.55% | 48.05% |
| MEMM$^+$ | 4.81% | 26.99% |
| CRF$^+$ | 4.27% | 23.76% |

$^+$Using spelling features

*Figure 4.* Per-word error rates for POS tagging on the Penn treebank, using first-order models trained on 50% of the 1.1 million word corpus. The oov rate is 5.45%.

[Lafferty et al., 2001]


Richard O. Duda
Peter E. Hart
David G. Stork
Pattern Classification
Second Edition


Nello Cristianini
John Shawe-Taylor
An Introduction to
Support Vector Machines
and other kernel-based learning methods

# Still more

- Accessible intro to MaxEnt: A simple introduction to maximum entropy models for natural language processing, Ratnaparkhi (1997)
- More complete: A maximum entropy approach to natural language processing, Berger et al. (1996) in CL
- MEMM paper: Maximum Entropy Markov Models for Information Extraction and Segmentation, McCallum (2000)

# References


Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996).
A maximum entropy approach to natural language processing.
*Computational Linguistics*, 22(1):39–71.

Cristianini, N. and Shawe-Taylor, J. (2000).
*An introduction to support Vector Machines: and other kernel-based learning methods*.
Cambridge Univ Pr.

Daelemans, W. and van den Bosch, A. (2005).
*Memory-Based Language Processing*.
Cambridge University Press.

Duda, R., Hart, P., and Stork, D. (2001).
*Pattern classification*.
Wiley New York.

Jurafsky, D. and Martin, J. (2008a).
*Speech and language processing*.
Prentice Hall.

Jurafsky, D. and Martin, J. H. (2008b).
*Speech and Language Processing*.
Prentice Hall, 2 edition.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001).
Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data.
In *ICML 2001: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Manning, C., Schütze, H., and Press, M. (1999).
*Foundations of statistical natural language processing*.
MIT Press.

Sha, F. and Pereira, F. (2003).