Review-3

Perceptron Learning Algorithm

Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$

Output: A classifying hyperplane $\vec{w}$

Randomly initialize $\vec{w}$;

while *model* $\vec{w}$ *makes errors on the training data* **do**

  for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**

    $\boxed{\text{Let } \hat{y} = sign(\vec{w} \cdot \vec{x}_i);}$

    if $\hat{y} \neq y_i$ **then**

      $\vec{w} = \vec{w} + y_i \vec{x}_i;$

    end

  end

end

Perceptron Learning Algorithm

Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$
Output: A classifying hyperplane $\vec{w}$
Randomly initialize $\vec{w}$;
while *model* $\vec{w}$ *makes errors on the training data* **do**
  for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**
    Let $\hat{y} = sign(\vec{w} \cdot \vec{x}_i)$;
    if $\hat{y} \neq y_i$ **then**
      $\vec{w} = \vec{w} + y_i \vec{x}_i$;
    end
  end
end

Perceptron Learning Algorithm

Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$

Output: A classifying hyperplane $\vec{w}$

Randomly initialize $\vec{w}$;

while *model* $\vec{w}$ *makes errors on the training data* **do**
  for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**
    Let $\hat{y} = sign(\vec{w} \cdot \vec{x}_i)$;
    if $\hat{y} \neq y_i$ **then**
      $\boxed{\vec{w} = \vec{w} + y_i \vec{x}_i;}$
    end
  end
end

Perceptron Learning Algorithm

Input: A list T of training examples $\langle \vec{x}_0, y_0 \rangle \ldots \langle \vec{x}_n, y_n \rangle$ where
$\forall i : y_i \in \{+1, -1\}$

Output: A classifying hyperplane $\vec{w}$

Randomly initialize $\vec{w}$;

while *model* $\vec{w}$ *makes errors on the training data* **do**

  for $\langle \vec{x}_i, y_i \rangle$ *in* T **do**

    Let $\hat{y} = sign(\vec{w} \cdot \vec{x}_i)$;

    if $\hat{y} \neq y_i$ **then**

      $\vec{w} = \vec{w} + y_i \vec{x}_i$;

    end

  end

end

Converges if the training set is ***linearly separable***

May not converge if the training set is ***not*** *linearly separable*

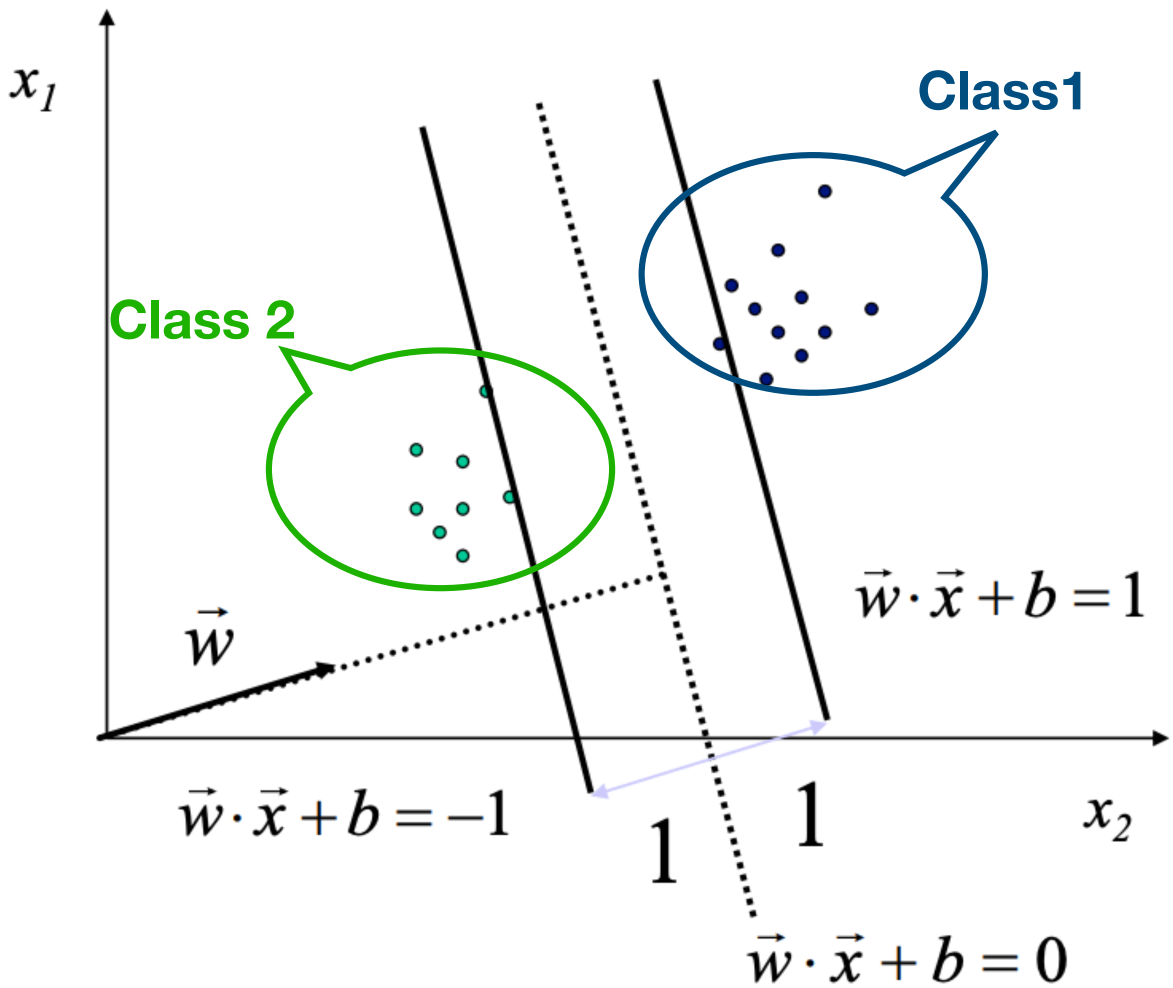
Support Vector Machines

- A learning method which *explicitly calculates the maximum margin hyperplane* by solving a gigantic quadratic programming minimization problem.

- Among the very highest -performing traditional machine learning techniques.

- But it’s relatively slow and quite complicated.

Setting Up the Optimization Problem



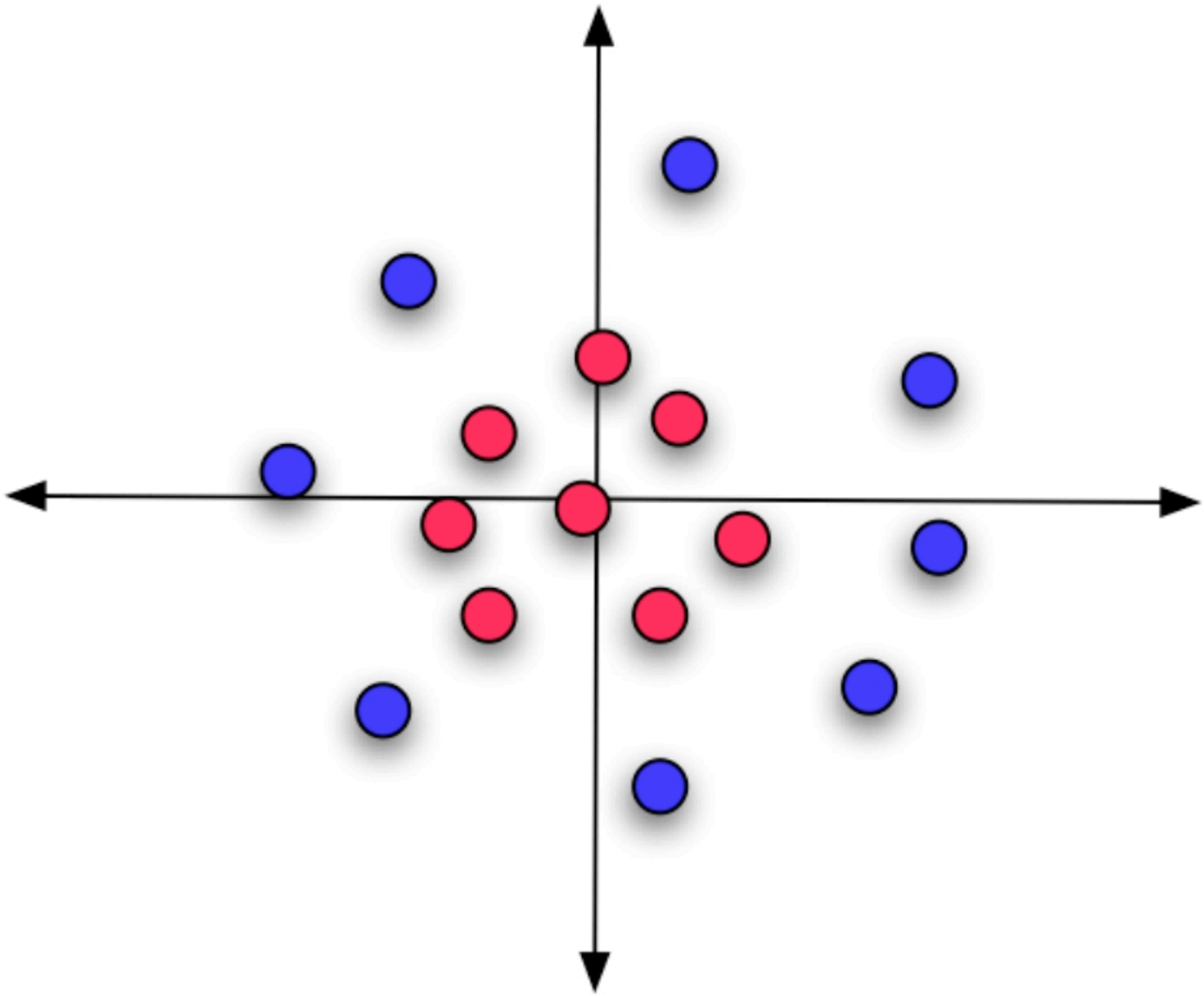
The maximum margin can be characterized as a solution to an optimization problem:

$$\max.\ \frac{2}{\|w\|}$$

$$s.t.\ (w\cdot x+b)\geq 1,\ \ \forall x \text{ of class 1}$$

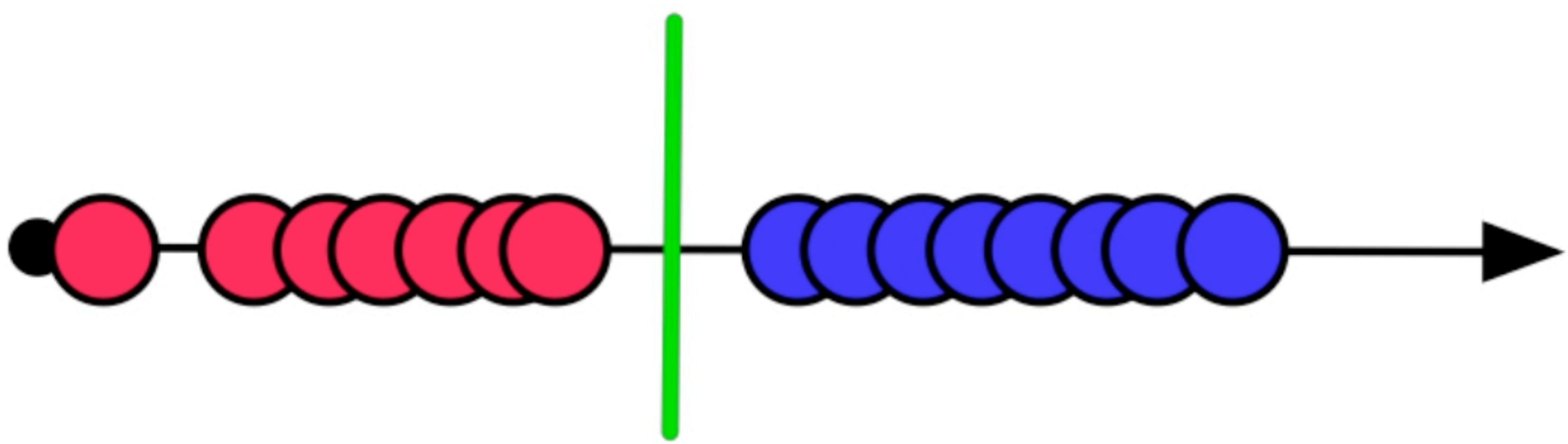
$$(w\cdot x+b)\leq -1,\ \ \forall x \text{ of class 2}$$

What if it isn’t separable?



Project it to someplace where it is!

$$\phi(\langle x, y \rangle) = x^2 + y^2$$



Multiclass Logistic Regression

$$P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$$

sum over output space to normalize

- Compare to binary:

$$P(y=1|x) = \frac{\exp(w^\top f(x))}{1+\exp(w^\top f(x))}$$

negative class implicitly had *f*(*x*, *y*=0) = the zero vector

Multiclass Logistic Regression

Softmax function

$$P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$$

sum over output space to normalize

Multiclass Logistic Regression

$$P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$$

sum over output space to normalize

- Training: maximize $\mathcal{L}(x,y) = \sum_{j=1}^{n} \log P(y_j^*|x_j)$

$$= \sum_{j=1}^{n} \left(w^\top f(x_j, y_j^*) - \log \sum_{y} \exp(w^\top f(x_j, y)) \right)$$

Training

- Multiclass logistic regression $P_w(y|x) = \dfrac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$
- Likelihood $\mathcal{L}(x_j, y_j^*) = w^\top f(x_j, y_j^*) - \log \sum_y \exp(w^\top f(x_j, y))$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \frac{\sum_y f_i(x_j, y)\exp(w^\top f(x_j, y))}{\sum_y \exp(w^\top f(x_j, y))}$$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \sum_y f_i(x_j, y) P_w(y|x_j)$$

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = \boxed{f_i(x_j, y_j^*)} - \boxed{\mathbb{E}_y[f_i(x_j, y)]}$$

gold feature value

model's expectation of feature value

Training

$$\frac{\partial}{\partial w_i}\mathcal{L}(x_j, y_j^*) = f_i(x_j, y_j^*) - \sum_y f_i(x_j, y) P_w(y|x_j)$$

too many drug trials, too few patients

y^* = Health

$f(x, y =$ Health $) =$ [1, 1, 0, 0, 0, 0, 0, 0, 0]

$f(x, y =$ Sports $) =$ [0, 0, 0, 1, 1, 0, 0, 0, 0]

$P_w(y|x)$ = [0.21, 0.77, 0.02]

gradient: [1, 1, 0, 0, 0, 0, 0, 0, 0] — 0.21 [1, 1, 0, 0, 0, 0, 0, 0, 0]
— 0.77 [0, 0, 0, 1, 1, 0, 0, 0, 0] — 0.02 [0, 0, 0, 0, 0, 0, 1, 1, 0]

= [0.79, 0.79, 0, -0.77, -0.77, 0, -0.02, -0.02, 0]

update $w^\top$:

[1.3, 0.9, -5, 3.2, -0.1, 0, 1.1, -1.7, -1.3] + [0.79, 0.79, (
= [2.09, 1.69, 0, 2.43, -0.87, 0, 1.08, -1.72, 0]

$$P_w(y|x) = \frac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$$

Softmax function

new $P_w(y|x)$ = [0.89, 0.10, 0.01]

Logistic Regression

- Model: $P_w(y|x) = \dfrac{\exp\left(w^\top f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(w^\top f(x,y')\right)}$
- Inference: $\mathrm{argmax}_y P_w(y|x)$
- Learning: gradient ascent on the discriminative log-likelihood

$$f(x,y^*) - \mathbb{E}_y[f(x,y)] = f(x,y^*) - \sum_y [P_w(y|x) f(x,y)]$$

"towards gold feature value, away from expectation of feature value"

Hidden Markov Models

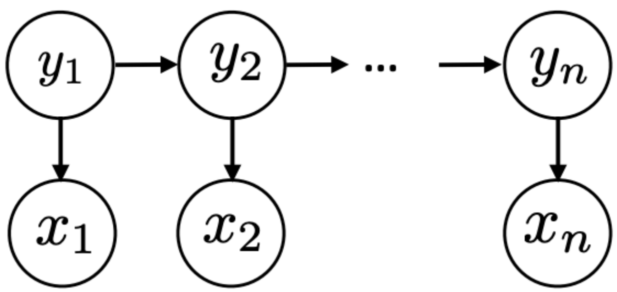
- Input $\mathbf{x} = (x_1, ..., x_n)$ Output $\mathbf{y} = (y_1, ..., y_n)$
- Model the sequence of *y* as a Markov process (dynamics model)
- Markov property: future is conditionally independent of the past given the present

$$y_1 \rightarrow y_2 \rightarrow y_3 \qquad P(y_3|y_1, y_2) = P(y_3|y_2)$$

- Lots of mathematical theory about how Markov chains behave
- If *y* are tags, this roughly corresponds to assuming that the next tag only depends on the current tag, not anything before

Inference in HMMs

- Input $\mathbf{x} = (x_1, ..., x_n)$ Output $\mathbf{y} = (y_1, ..., y_n)$



$$P(\mathbf{y}, \mathbf{x}) = P(y_1) \prod_{i=2}^{n} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)$$

- Inference problem: $\text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \text{argmax}_{\mathbf{y}} \dfrac{P(\mathbf{y}, \mathbf{x})}{P(\mathbf{x})}$

Viterbi Algorithm

1. Initial: For each state s, calculate

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

2. Recurrence: For i = 2 to n, for every state s, calculate

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

$$= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} \text{score}_{i-1}(y_{i-1})$$

3. Final state: calculate

$$\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B) = \max_{s} \text{score}_n(s)$$

π: Initial probabilities
A: Transitions
B: Emissions

This only calculates the max. To get final answer (*argmax*),

- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

Example

Sentence: “Learning changes people”
Tagset: NN, VB

				q_F
VB	$9*10^{-4}$ (q_0)	$3.6*10^{-7}$ (VB)	$7.2*10^{-11}$	$1.44*10^{-12}$
NN	$2*10^{-4}$ (q_0)	$10.8*10^{-6}$	$7.2*10^{-8}$ (VB)	$7.2*10^{-9}$ (NN)
Input	Learning	changes	People	
Output	VB	VB	NN	

$P(NN|VB)= 4*10^{-1}$
$P(NN|NN)= 1*10^{-1}$
$P(VB|VB)= 1*10^{-1}$
$P(VB|NN)= 3*10^{-1}$

$P(VB|q_0)= 3*10^{-1}$
$P(NN|q_0)= 2*10^{-1}$

$P(q_f|NN)=1*10^{-1}$
$P(q_f|VB)= 2*10^{-2}$

$P(Learning|VB)= 3*10^{-3}$
$P(Learning|NN)= 1*10^{-3}$
$P(People|NN)= 5*10^{-2}$
$P(People|VB)= 2*10^{-4}$
$P(changes|NN)= 3*10^{-3}$
$P(changes|VB)= 4*10^{-2}$

Conditional Random Fields

- HMMs: $P(\mathbf{y}, \mathbf{x}) = P(y_1)P(x_1|y_1)P(y_2|y_1)P(x_2|y_2)\ldots$

- CRFs: discriminative models with the following globally-normalized form:

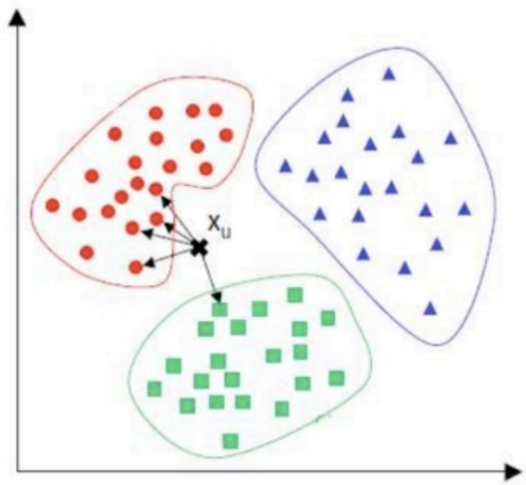
$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_k \exp(\phi_k(\mathbf{x}, \mathbf{y}))$$

normalizer

any real-valued scoring function of its arguments

K-Nearest Neighbor

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



  - For classification: assign the majority class label (majority voting)
  - For regression: assign the average response
- The algorithm requires:
  - Parameter K: number of nearest neighbors to look for
  - Distance function: To compute the similarities between examples

K-Nearest Neighbor Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

Note: K-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, K-Nearest Neighbors doesn't learn an explicit mapping f from the training data
- It simply uses the training data at the test time to make predictions

K-NN: Feature Normalization

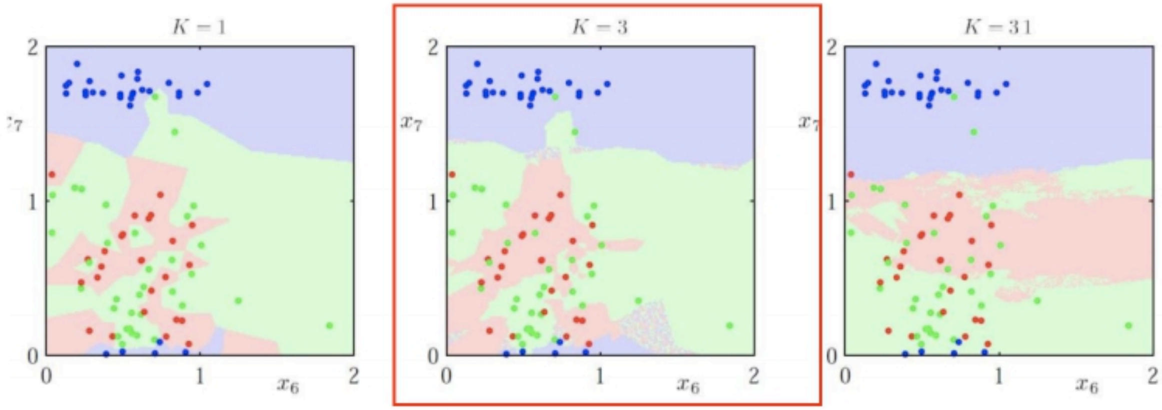
- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x_m})}{\sigma_m}$ (make them zero mean, unit variance)
  - $\bar{x_m} = \frac{1}{N}\sum_{i=1}^{N} x_{im}$: empirical mean of m^{th} feature

K-NN: Other Distance Measure

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree

- Mixed feature types (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part

K-NN: Choice of K



- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit
- Large K
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
- Choosing K
  - Often data dependent and heuristic based
  - Or using cross-validation (using some **held-out data**)
  - In general, a K too small or too big is bad!