

CSE 417T

Introduction to Machine Learning

Lecture 16

Instructor: Chien-Ju (CJ) Ho

Logistics

- Homework 4 is due April 1 (Friday)
- Keep track of your own late days
 - Gradescope doesn't allow separate deadlines
 - Your submissions **won't be graded** if you exceed the late-day limit

Recap

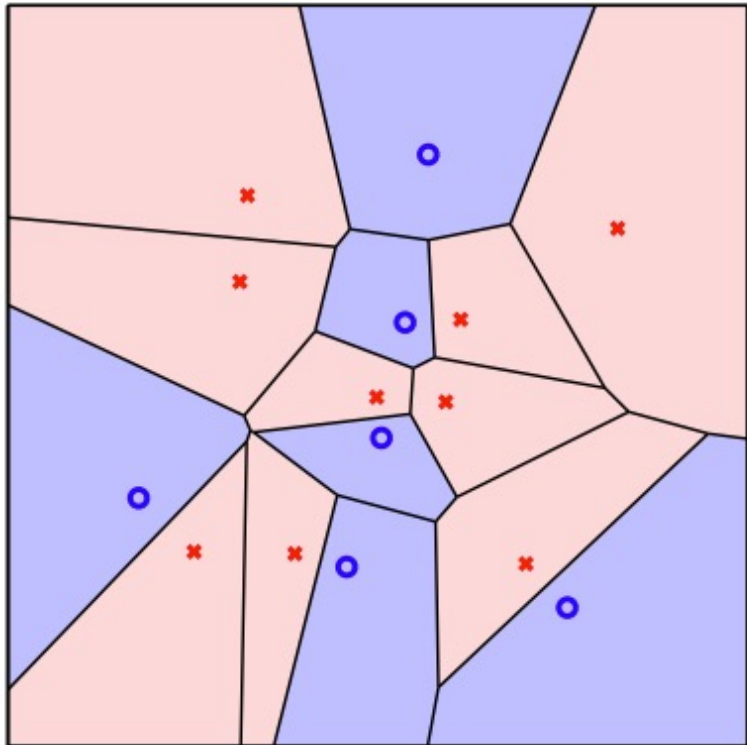
Nearest Neighbor

- Predict \vec{x} according to its nearest neighbor
 - Given $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}; y_n \in \{+1, -1\}$
 - Let $\vec{x}_{[1]}$ be \vec{x} 's nearest neighbor, i.e., the closest point to \vec{x} in D
 - Let $y_{[i]}(\vec{x})$ or $y_{[i]}$ be the label of $\vec{x}_{[i]}$
- Nearest neighbor hypothesis

$$g(\vec{x}) = y_{[1]}(\vec{x})$$

Nearest Neighbor

$g(\vec{x})$ looks like a Voronoi diagram



- Properties of Nearest Neighbor (NN)
 - No training is needed
 - Good interpretability
 - In-sample error $E_{in} = 0$
 - VC dimension is ∞
- This seems to imply bad learning models from what we talk about so far? Why we care?
- What we really care about is E_{out}
 - VC analysis: $E_{out} \leq E_{in} + \text{Generalization error}$
 - We can infer E_{out} through E_{in} and model complexity
 - NN has nice guarantees outside of VC analysis

Nearest Neighbor is 2-Optimal

- Given mild conditions, for nearest neighbor, when $N \rightarrow \infty$, with high probability,

$$E_{out} \leq 2E_{out}^*$$

- That is, we can not infer E_{out} from E_{in} , but we know it cannot be much worse than the **best anyone can do**.

Informal intuitions:

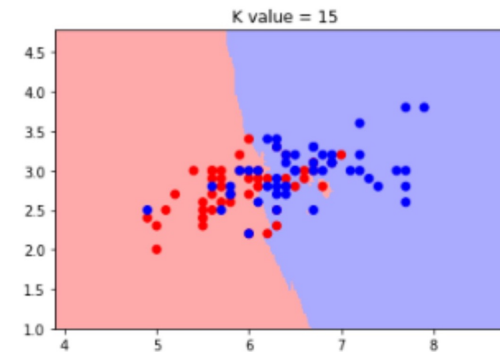
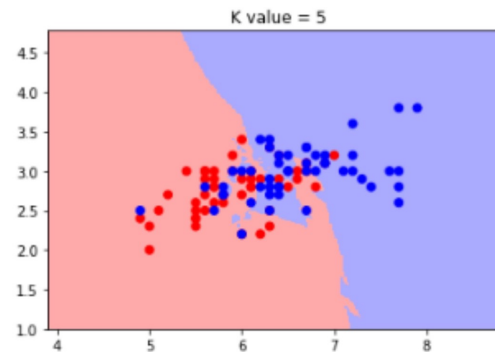
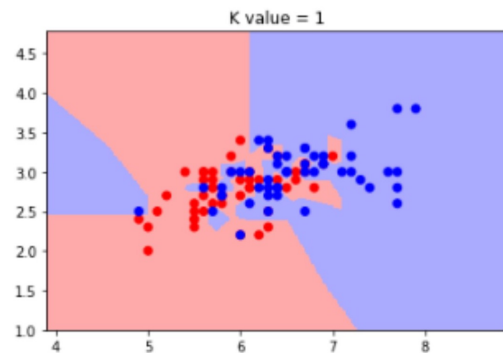
Assumption for nearest neighbor:

- Target function is continuous
- That is, nearby points have similar label distributions

As N goes large, there are “close enough” points for prediction

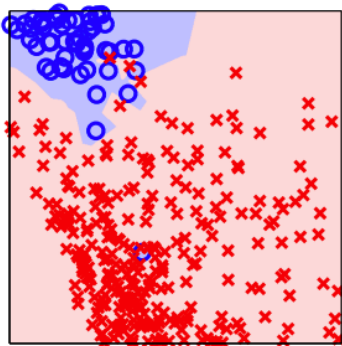
k -Nearest Neighbor (K-NN)

- Nearest neighbor
 - Influenced heavily by noisy data
- k -nearest neighbor (K-NN)
 - $g(\vec{x}) = \text{sign}\left(\sum_{i=1}^k y_{[i]}(\vec{x})\right)$
 - (k is often odd for binary classification)

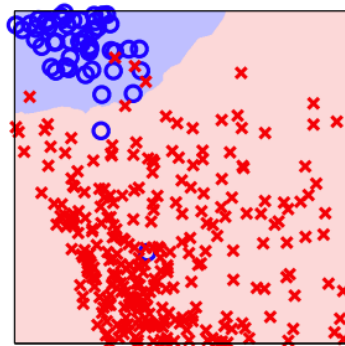


Impacts of k in k -Nearest Neighbor

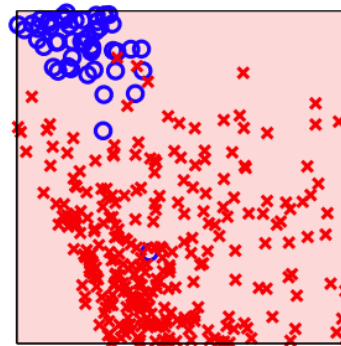
- $k = 1$: the nearest neighbor hypothesis
 - many, complicated decision boundaries; may overfit
- $k = N$, g predicts the most common label in the training dataset
 - no decision boundaries; may underfit
- k controls the complexity of the hypothesis set
 - k affects how well the learned hypothesis will generalize



1-NN rule



21-NN rule



127-NN rule

How to Choose k

- Making the choice of k a function of N , denoted by $k(N)$
- Theorem:
 - For $N \rightarrow \infty$, if $k(N) \rightarrow \infty$ and $\frac{k(N)}{N} \rightarrow 0$
 - Then $E_{in}(g) \rightarrow E_{out}(g)$ and $E_{out}(g) \rightarrow E_{out}(g^*)$
- Example: $k(N) = \sqrt{N}$ satisfies the condition

Informal intuitions:

$\frac{k(N)}{N} \rightarrow 0$: all the nearest neighbors are next to \vec{x}

$k(N) \rightarrow \infty$: select infinitely many neighbors

=> We can almost reconstruct the target function

How to Choose k

- Making the choice of k a function of N , denoted by $k(N)$
- Theorem:
 - For $N \rightarrow \infty$, if $k(N) \rightarrow \infty$ and $\frac{k(N)}{N} \rightarrow 0$
 - Then $E_{in}(g) \rightarrow E_{out}(g)$ and $E_{out}(g) \rightarrow E_{out}(g^*)$
- Example: $k(N) = \sqrt{N}$ satisfies the condition
- Practical rules of thumb:
 - Small k (e. g., $k = 3$) is often a good enough choice
 - Using validation to choose k

Summary of k -NN So Far

- Pros
 - Simple algorithm
 - Good interpretations
 - Nice theoretical guarantee
 - Easy to adapt to regression (average of nearest neighbors) and multi-class classification (majority voting)
- Cons
 - Curse of dimensionality
 - Computational issue
 - each prediction requires $O(N)$ computation

Today's Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.
Let me know if you spot errors.

Curse of Dimensionality

- Generally, higher dimensionality implies harder learning (think VC)
- Things are worse with similarity-based methods
 - Rely on assumptions that nearby points have similar labels
 - As the dimension grows, most of the points will not be nearby to each other...

Illustration of Curse of Dimensionality

- Think about Euclidean distance: $d(\vec{x}, \vec{x}') = \|\vec{x} - \vec{x}'\|$
- Illustration
 - Consider the space $[0,1]^d$ (a hypercube with length of each side = 1)
 - What's the side length ℓ of a hypercube that takes up 1% of the space?
 - $d = 1$: $\ell = 0.01$
 - $d = 2$: $\ell = 0.1$
 - ...
 - $d = 100$: $\ell^d = 0.01 \Rightarrow \ell \approx 0.95$
 - $d = 1000$: $\ell^d = 0.01 \Rightarrow \ell \approx 0.9954$

Illustration of Curse of Dimensionality

- Consider the distance to the origin when $d = 100$
 - Consider the case that the value of each dimension is uniformly drawn
 - Only 1% of the points will be in the hypercube $[0,0.95]^{100}$
 - Most of the points will be far away from the origin
 - Most of the points will be far away from each other
- No simple solutions....
 - Getting more data
 - Reduce dimensions (see LFD 9.2)

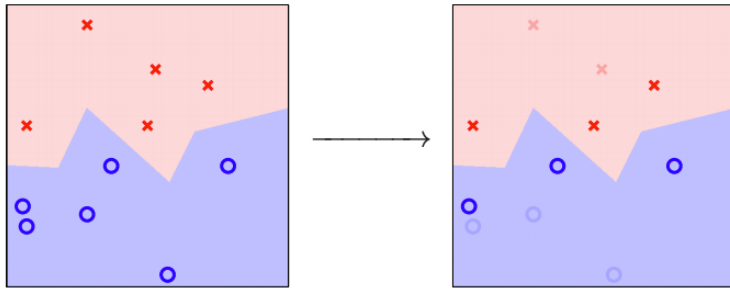
Computational Issues for k-NN

Computational Issues

- k -Nearest Neighbor is computationally demanding
 - Need to store all data points: space complexity $O(Nd)$
 - For each prediction for \vec{x}
 - Calculate the distance to every point in D
 - Find the k closest points
 - Time complexity $O(Nd + N \log k)$
- Two general approaches:
 - Reduce the number of data points
 - Store the data in some data structure to speed up searching
 - See LFD 6.2.3 for more discussion

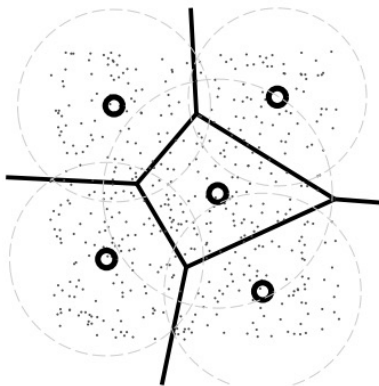
Computational Issues: Potential Solutions

- Reduce the number of data points



- Intuition: remove points that will not impact the decision boundary.
- Generally a hard problem. But there are some heuristic approaches.

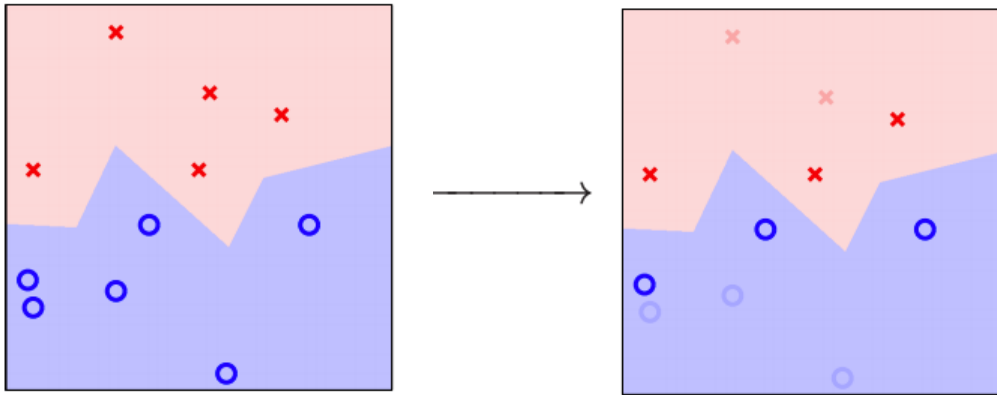
- Store the data in some data structure to speed up searching



- Intuition: Cluster data points
- For a new data point, first find a nearest cluster. Then find the nearest points within that cluster

Reduce the Amount of Data

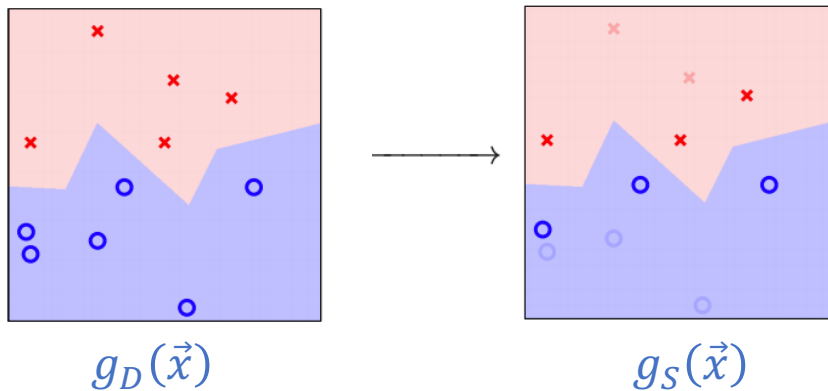
- Consider the following example



- Goal: Drop data that doesn't impact the decision boundary
 - Find $S \subseteq D$, such that $g_S(\vec{x}) = g_D(\vec{x}) \forall \vec{x}$
 - Generally a hard goal to achieve

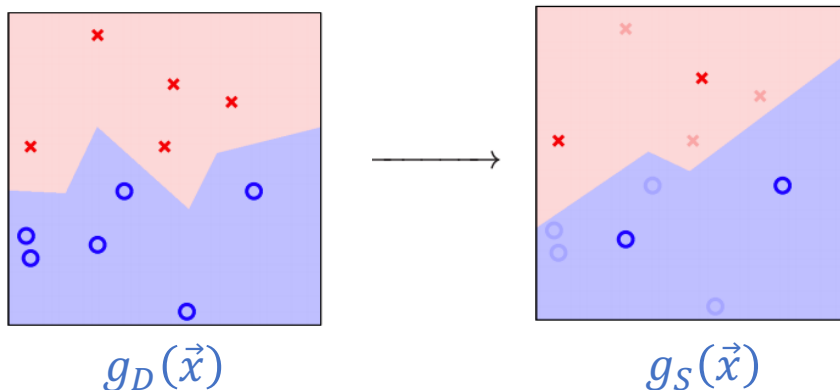
A Probably Easier Goal

- Instead of making the **decision boundary** consistent



Find $S \in D$, such that $g_S(\vec{x}) = g_D(\vec{x}) \forall \vec{x}$

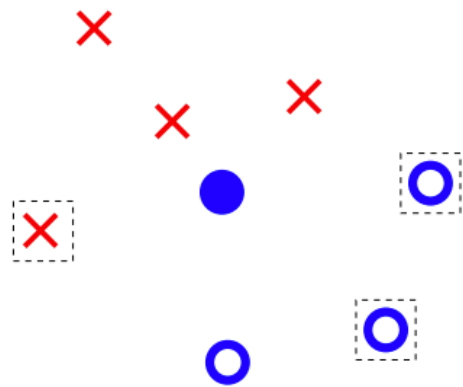
- Aim for making the prediction on the **training data** consistent



Find $S \in D$, such that $g_S(\vec{x}) = g_D(\vec{x}) \forall \vec{x} \in D$

Condensed Nearest Neighbor (CNN)

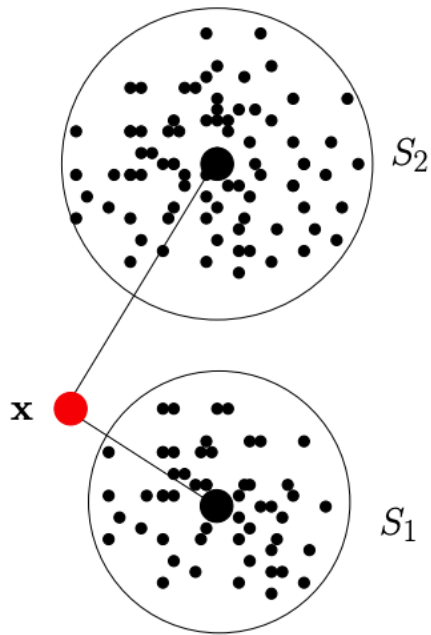
- Goal: Find $S \in D$, such that $g_S(\vec{x}) = g_D(\vec{x}) \forall \vec{x} \in D$



- CNN: An iterative algorithm
 - Random initialize a subset of data points S
 - While (there exists point $\vec{x}_* \in D$ such that $g_S(\vec{x}_*) \neq g_D(\vec{x}_*)$)
 - Let y_* be $g_D(\vec{x}_*)$
 - Find the nearest point $\vec{x}' \in D \setminus S$ with the label y_*
 - Insert \vec{x}' to S
- No theoretical guarantees
- Reasonable empirical performance

Speed Up the Search for Nearest Neighbor

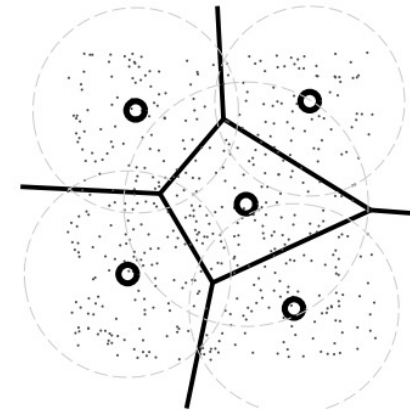
- For each prediction for \vec{x} , we need to find the nearest point in D
- Can we speed up this process?



- S_1 and S_2 are two clusters
 - Centers $\vec{\mu}_1, \vec{\mu}_2$ and radii r_1, r_2
- Let S_1 be the nearest cluster for \vec{x}
- Let $\vec{x}'_{[1]}$ be the nearest point in S_1
- Distance from \vec{x} to any point in S_2 is at least $\|\vec{x} - \vec{\mu}_2\| - r_2$
- If $\|\vec{x} - \vec{x}'_{[1]}\| \leq \|\vec{x} - \vec{\mu}_2\| - r_2$
 - we don't need to search points in S_2

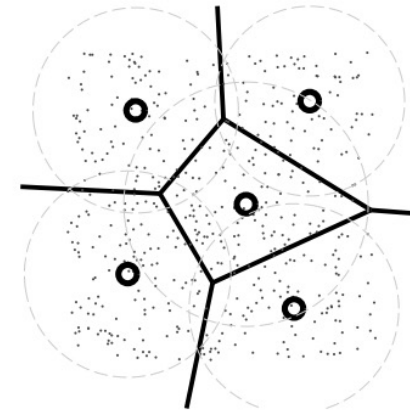
Lloyd's Algorithm: Construct Clusters

- Goal: Cluster data into K clusters
 - Again, hard in general, but we can use greedy-based approaches



Lloyd's Algorithm: Construct Clusters

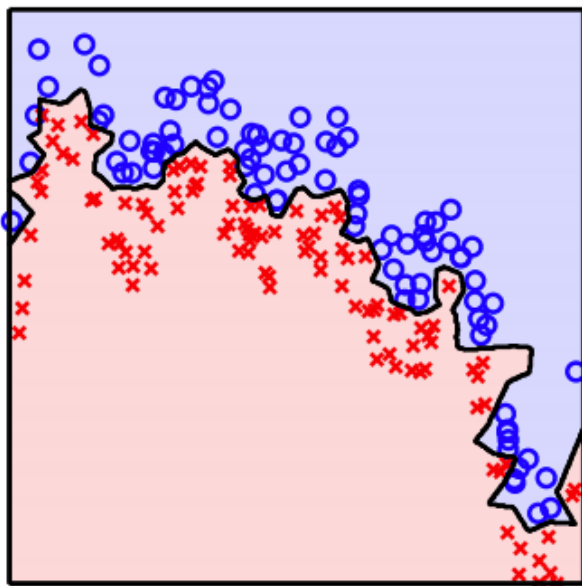
- Goal: Cluster data into K clusters
 - Again, hard in general, but we can use greedy-based approaches
- Lloyd's Algorithm
 1. Randomly pick K points as centers
 2. Create the Voronoi regions as clusters
 3. Update the centers (calculating the mean)
 4. Update the region
 5. Repeat 3 and 4
- This is the first (and probably the only) unsupervised learning algorithm we talk about in this course.



Final Remark:

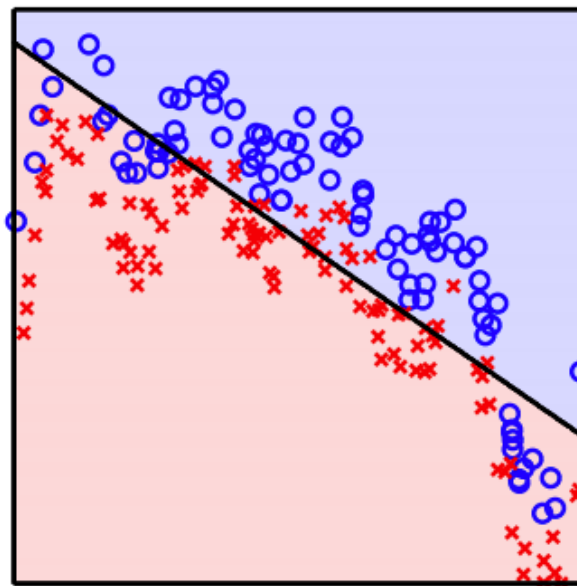
Nearest Neighbor is Nonparametric

Nearest Neighbor



no parameters
expressive/flexible
 $g(\mathbf{x})$ needs data
generic, can model anything

Linear models



$(d + 1)$ parameters
rigid, always linear
 $g(\mathbf{x})$ needs only weights
specialized

Radial Basis Functions (RBF)

Instance-Based Learning

- Make predictions based on data instances
- k -nearest neighbor (K-NN)
 - $g(\vec{x}) = \text{sign}(\sum_{i=1}^k y_{[i]}(\vec{x}))$
 - Predict according to the nearest neighbors
- Radial Basis Functions (RBF)
 - Focus of today
- Kernel SVM (Topic of Next Week)
 - $g(\vec{x}) = \text{sign}(\sum_{\alpha_n^* > 0} \alpha_n^* y_n K(\vec{x}_n, \vec{x}) + b^*))$
 - Predict according to support vectors

Radial Basis Functions

- Think about k -nearest neighbor (K-NN) again
 - $g(\vec{x}) = \text{sign}(\sum_{i=1}^k y_{[i]}(\vec{x}))$
 - Make predictions based on k nearest data points
 - Each of the k data points has the same weight
- Natural questions:
 - Can we use more (or even all) data?
 - **Weight** them based on **how close** data points are to \vec{x}

Radial Basis Functions

- Given dataset $D = \{\vec{x}_1, \dots, \vec{x}_N\}$
- Task: Make a prediction on \vec{x}

- Radial Basis Function:

- $g(\vec{x}) = \frac{1}{Z(\vec{x})} \sum_{n=1}^N \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right) y_n$

- $\phi(s)$: a monotonically decreasing function

- This is for regression. We can take a sign and make it a binary classification.
- $Z(\vec{x}) = \sum_{m=1}^N \phi\left(\frac{\|\vec{x} - \vec{x}_m\|}{r}\right)$ is for normalization

- It's called **radial** basis function since it takes the **distance** to the points as the basis function

Radial Basis Functions

- Radial Basis Function:

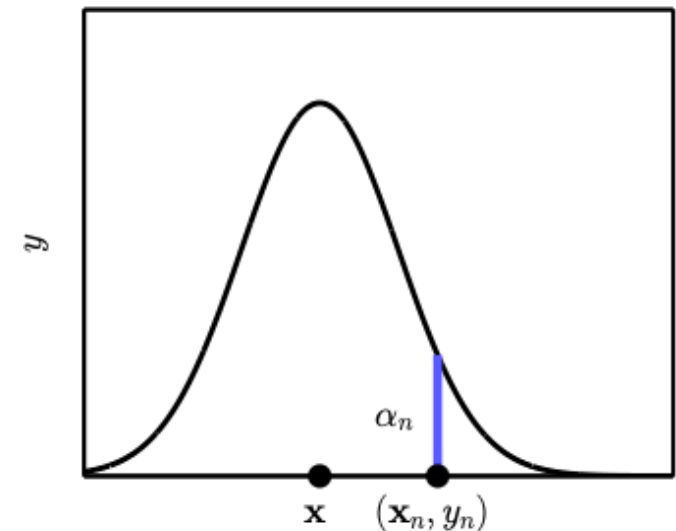
- $g(\vec{x}) = \sum_{n=1}^N \frac{1}{Z(\vec{x})} \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right) y_n$

- Example of ϕ

- Gaussian RBF: $\phi(s) = e^{-s}$

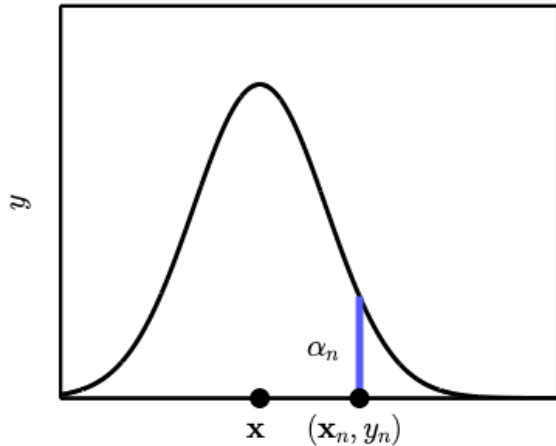
- Intuitions

- The impact of \vec{x}_n to \vec{x} is higher if it's closer to \vec{x}
 - The role of r is similar to k in k -NN
 - $r \rightarrow 0$: 1-NN
 - $r \rightarrow \infty$: N -NN (i.e., $k = N$)



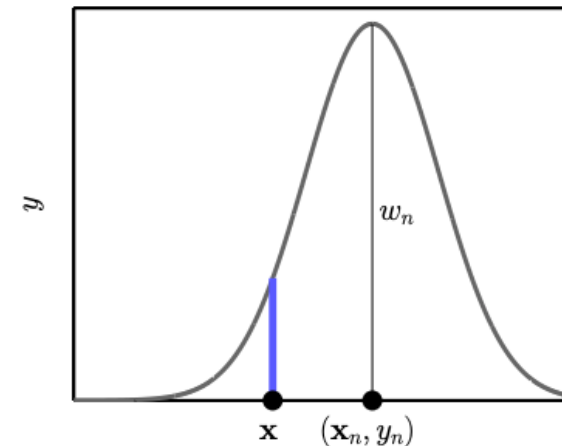
Changing the Viewpoints

- ϕ centered around \vec{x}



$$g(\vec{x}) = \sum_{n=1}^N \frac{y_n}{Z(\vec{x})} \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$$

- ϕ centered around \vec{x}_n



$$g(\vec{x}) = \sum_{n=1}^N w_n(\vec{x}) \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$$

From Nonparametric to Parametric RBF

- Nonparametric RBF

- $g(\vec{x}) = \sum_{n=1}^N \frac{y_n}{Z(\vec{x})} \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$
- $g(\vec{x}) = \sum_{n=1}^N w_n(\vec{x}) \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$
- The hypothesis is defined by dataset

- Parametric RBF hypothesis set

- $h(\vec{x}) = \sum_{n=1}^N w_n \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$
- Learn w_n from data

Parametric RBF => Linear Models

- Parametric RBF is linear model with nonlinear transformation

- $h(\vec{x}) = \sum_{n=1}^N w_n \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$

- The projection $\Phi(\vec{x}) = \begin{bmatrix} \phi\left(\frac{\|\vec{x} - \vec{x}_1\|}{r}\right) \\ \phi\left(\frac{\|\vec{x} - \vec{x}_2\|}{r}\right) \\ \vdots \\ \phi\left(\frac{\|\vec{x} - \vec{x}_N\|}{r}\right) \end{bmatrix}$

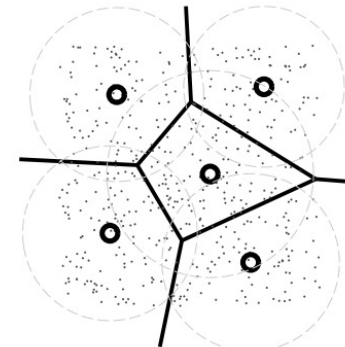
- We can apply what we learned in linear models to learn w_n
- However, this seems to be **overfitting** (N parameters for N points)

From N points to K points

- Use only K points $(\vec{\mu}_1, \dots, \vec{\mu}_K)$

- $h(\vec{x}) = \sum_{k=1}^K w_k \phi\left(\frac{\|\vec{x} - \vec{\mu}_k\|}{r}\right)$

- Which K points?
 - We can find K representative points
 - Use clustering algorithms, e.g., Lloyd algorithm as introduced earlier
 1. Randomly pick K points as centers
 2. Create the Voronoi regions as clusters
 3. Update the centers (calculating the mean)
 4. Update the region
 5. Repeat 3 and 4



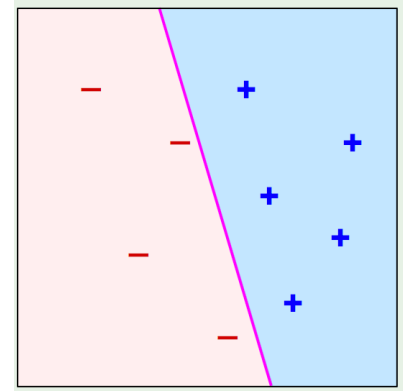
More Discussion on RBF

- $h(\vec{x}) = \sum_{k=1}^K w_k \phi\left(\frac{\|\vec{x} - \vec{\mu}_k\|}{r}\right)$
- Connection to linear models
 - Parametric RBF is essentially linear model with nonlinear transformation
- Connection to nearest neighbor
 - Radial Basis Function is defined by “similarity”
 - A prediction for a point is based on the “similarity” of the points to be predicted and other points

Support Vector Machines (SVM)

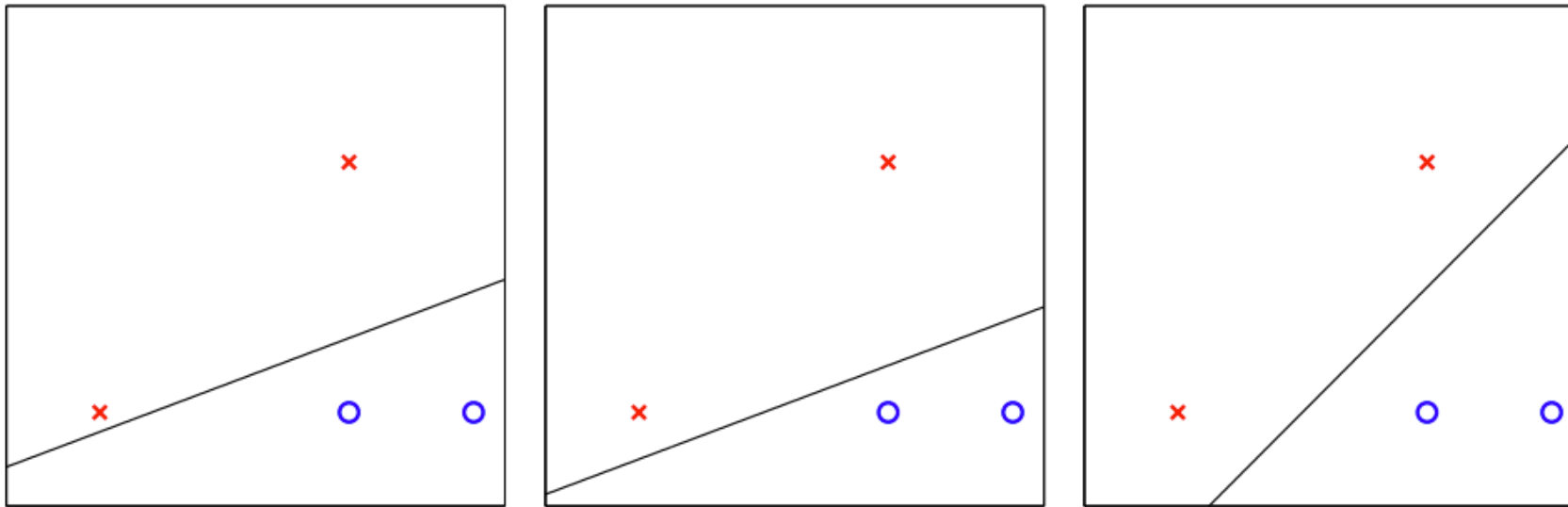
What Do We Know about Linear Classification?

- What we discussed so far:
 - PLA: Find a linear separator that separates the data within finite steps, if data is linear separable.
 - Pocket algorithm: empirically keep the best separator during PLA.
 - Surrogate loss: Using logistic regression for linear classification.
- Challenges
 - Binary classification error is hard to optimize
 - We cannot use “gradient descent” type of algorithm minimize E_{in} .
- Support vector machines (SVM) tries to look at things a bit differently.



Linear Classification

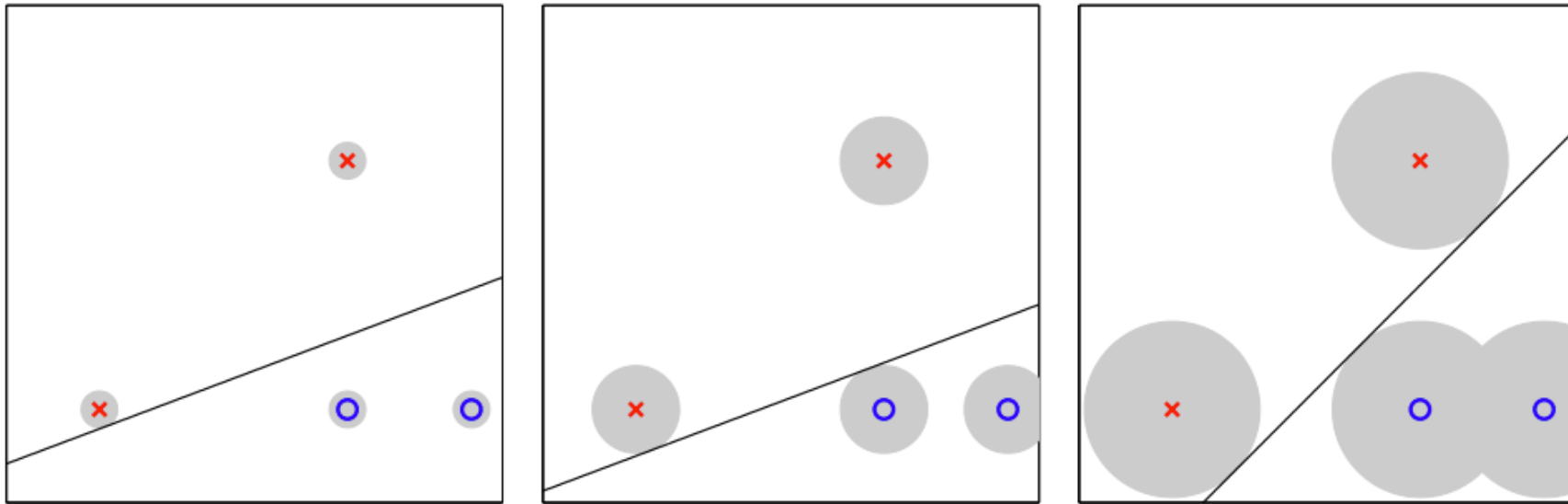
- Which separator would you choose?



Probably the right one.
Why?

Linear Classification

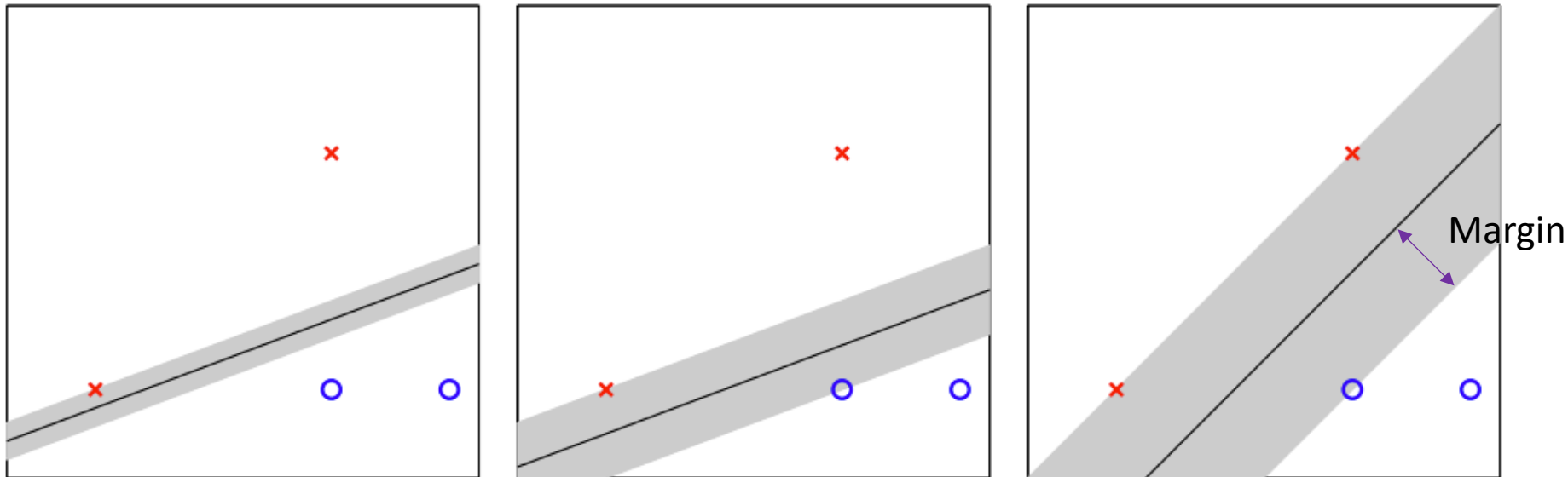
- Which separator would you choose?



More robust to noise (e.g., measurement error of \vec{x})

Linear Classification

- Which separator would you choose?



Margin: shortest distance from the separator to the points in D
(Informal argument)

Higher margin \Rightarrow more “constrained” hypothesis \Rightarrow lower VC dimension

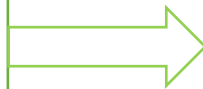
Support Vector Machine

- Goal:
 - Find the **max-margin** linear separator that separates the data
 - Recall the goal of PLA: Find the linear separator that separates the data

- Notations:

Notations we used so far:

- $\vec{x} = (x_0, x_1, \dots, x_d)$
- $\vec{w} = (w_0, w_1, \dots, w_d)$
- Linear separator
$$h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})$$



Notations we will use in SVM

- $\vec{x} = (x_1, \dots, x_d)$
- $\vec{w} = (w_1, \dots, w_d)$
- Linear separator
$$h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$$

Separating the bias/intercept b is important for us to characterize the margin.

We will use (\vec{w}, b) to characterize the hypothesis