Introduction to Machine Learning

Lecture 10
Supervised Learning

- Instance-based Learning

- Some materials are courtesy of Vibhave Gogate and Tom Mitchell.
- All pictures belong to their creators.

Instance-based Learning

- Decision Tree & Logistic regression
- Linear Regression & Perceptron & Neural Network & SVM

Recap

- (Step 1) Using your training data to train a function/classifier.
- (Step 2) When you have a new instance, apply the thing you learn.

Instance-based learning:
(Step 1) Store your training data, and have a rest.
(Step 2) When you have a new instance z, wake up and use an approach to assign a value to z. The approach is customized to z.

Instance-based Learning

Instance-based learning:
(Step 1) Store your training data, and have a rest.
(Step 2) When you have a new instance z, wake up and use an approach to assign a value to z. The approach is customized to z.

(Step 2) When you have a new instance z, wake up and use an approach to assign a value to z. The approach is customized to z.

Define the similarity between instances.
Use the instances similar to z to assign a value to z.

Different function/training data for different instances.

Parametric vs. Non-parametric

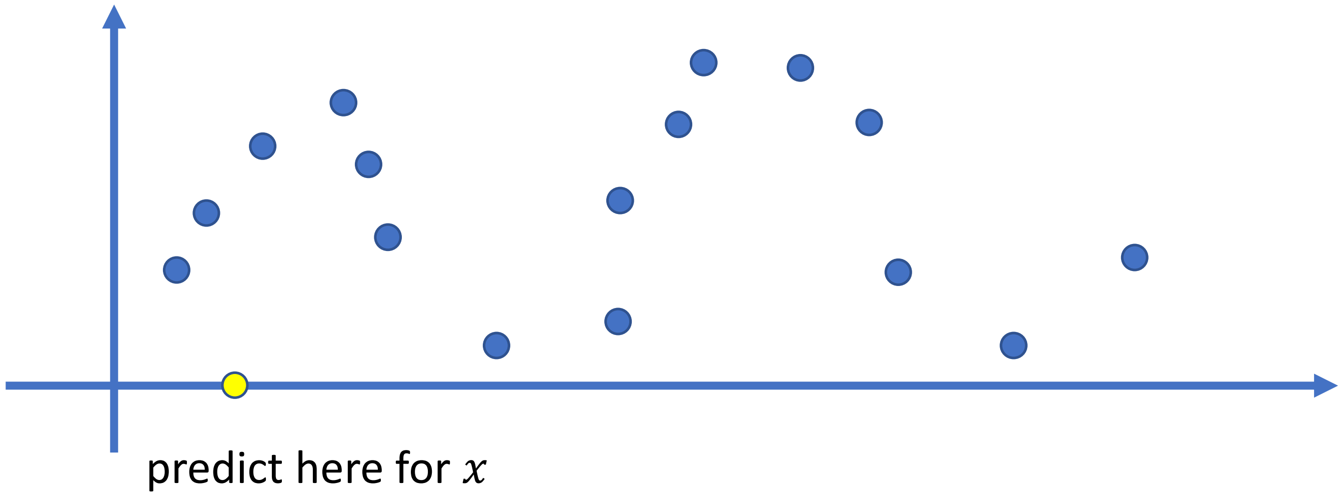
Parametric:

- A particular functional form is assumed, e.g, linear, naïve Bayes.
- Advantage of simplicity – easy to estimate and interpret
- May have high bias because the real data may not obey the assumed functional form.

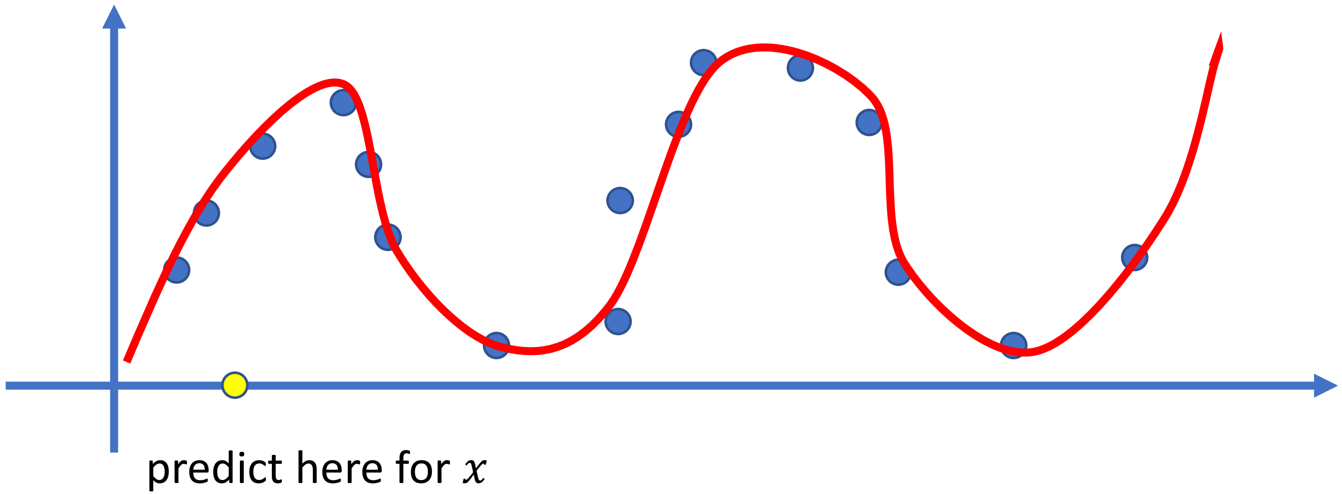
Non-parametric:

- Distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form.
- May have a high cost.

Parametric vs. Non-parametric

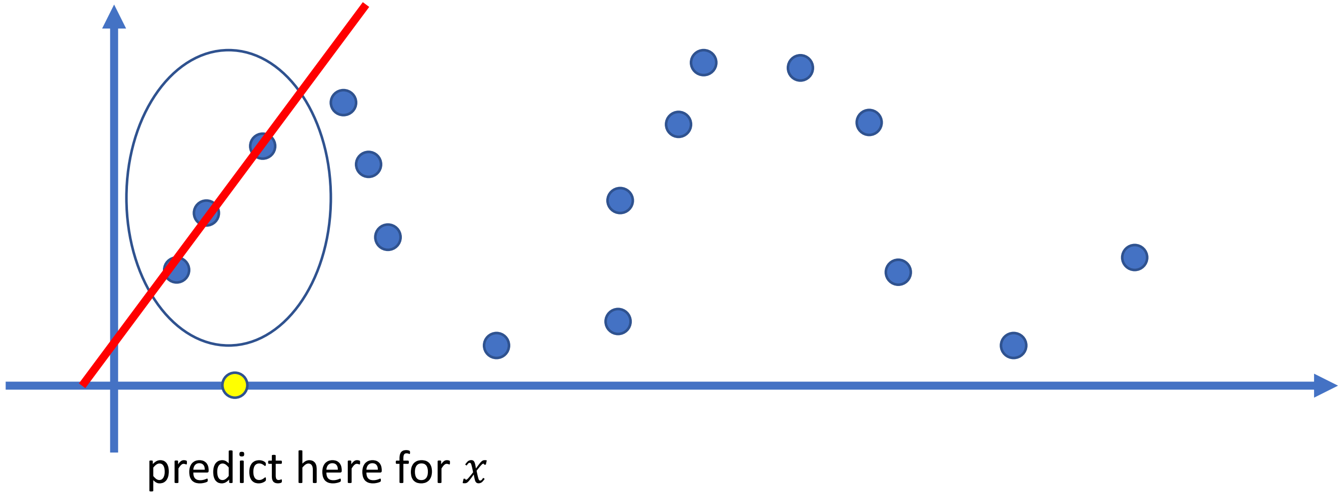


Parametric vs. Non-parametric



Parametric: learning a curve h and apply to x.

Parametric vs. Non-parametric



Instance-based: look at the local area around x. Use a simple method, e.g., linear regression.

Outline

- K-nearest-neighbor algorithm (KNN)
- Locally weighted regression
- Some issues.

K-nearest-neighbor algorithm

- Input $(\boldsymbol{x}, y)$: $\boldsymbol{x} = (x^1, \ldots, x^n)$ real-vector, y target value.

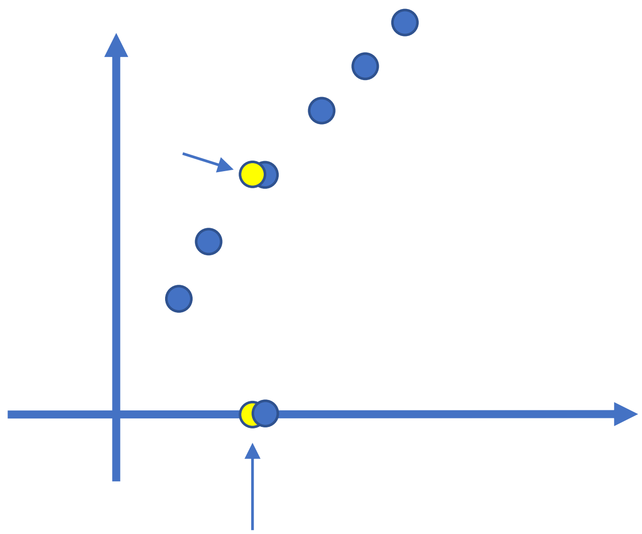
- **Nearest Neighbor Approach.**
- Suppose we can define the similarity between instances.

- Given a new instance $\boldsymbol{z}$, find the one x in D that is most similar to $\boldsymbol{z}$.
- Assign $\boldsymbol{z}$ the same value as **x**.

K-nearest-neighbor algorithm

- Input $(\boldsymbol{x}, y)$: $\boldsymbol{x} = (x^1, \dots, x^n)$ real-vector, y target value.

- **Nearest Neighbor Approach.**



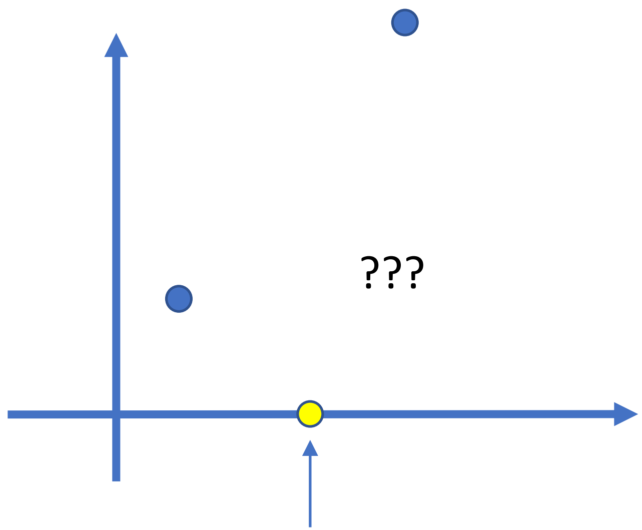
$(\boldsymbol{x}, y)$
Distance $|\boldsymbol{x_1} - \boldsymbol{x_2}|$

If your training data is dense and correct.

K-nearest-neighbor algorithm

- Input $(\boldsymbol{x}, y)$: $\boldsymbol{x} = (x^1, \dots, x^n)$ real-vector, y target value.

- **Nearest Neighbor Approach.**



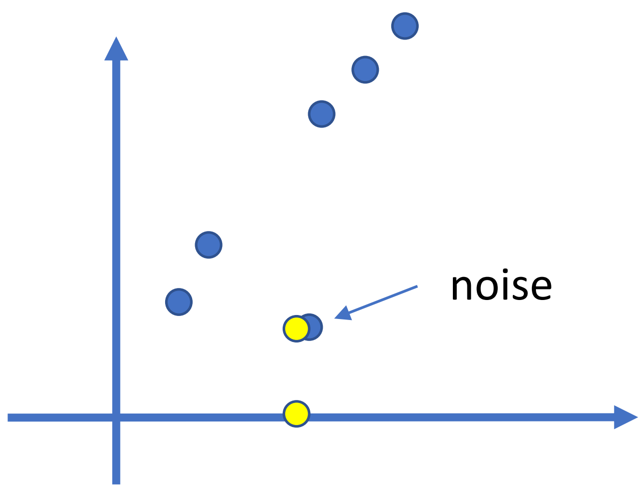
$(\boldsymbol{x}, y)$
Distance $|\boldsymbol{x_1} - \boldsymbol{x_2}|$

If your training data is sparse…

K-nearest-neighbor algorithm

- Input $(\boldsymbol{x}, y)$: $\boldsymbol{x} = (x^1, \dots, x^n)$ real-vector, y target value.

- **Nearest Neighbor Approach.**



$(\boldsymbol{x}, y)$
Distance $|\boldsymbol{x_1} - \boldsymbol{x_2}|$

If your training data has noise…

How to make it robust?

K-nearest-neighbor algorithm

- Input $(\boldsymbol{x}, y)$: $\boldsymbol{x} = (x^1, \ldots, x^n)$ real-vector, y target value.

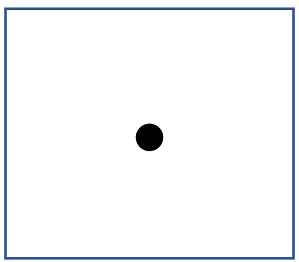
- Given a new instance $\boldsymbol{z}$, find $\{\boldsymbol{x_i}, y_i\}_{i=1}^{k}$ k training instances that are the nearest to $\boldsymbol{z}$.
- If classification problem, return the majority of the class among the k selected instances.
- If regression problem, return the mean $\frac{\sum y_i}{k}$.

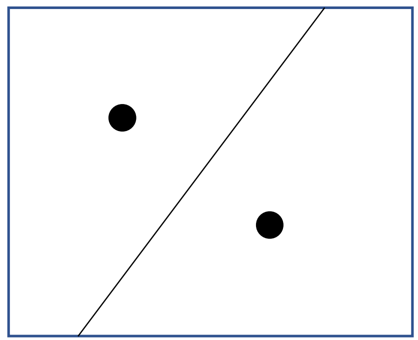
- Euclidian Distance between two vectors: $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{\sum\left(x_1^i - x_2^i\right)^2}$.

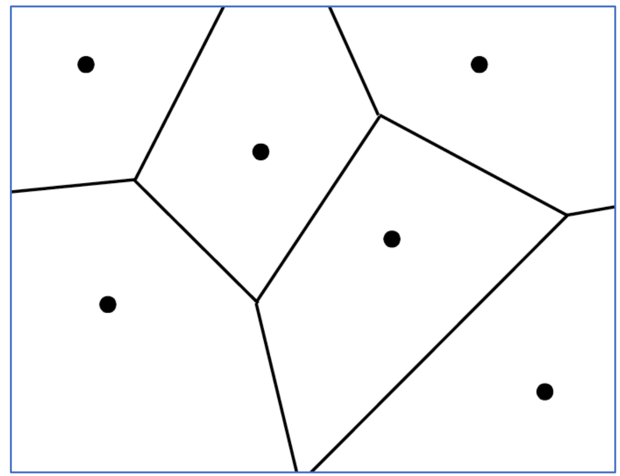
K-nearest-neighbor algorithm

- Decision Boundary of 1-NN.

Recall multi-class perceptron.







Voronoi diagram

K-nearest-neighbor algorithm

- Each instance in $\{\boldsymbol{x_i}, y_i\}_{i=1}^{k}$ is treated equally.
- Probability the neatest one should have a larger impact.

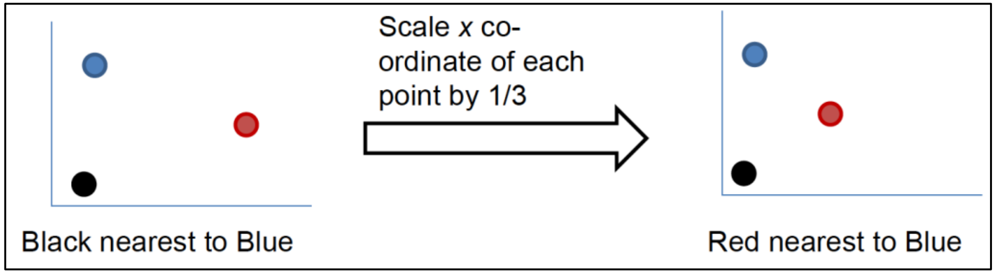
- Weighted prediction $\frac{\sum w_i \cdot y_i}{\sum w_i}$
- How to decide the weight w_i?
- $w_i = \frac{1}{d(\boldsymbol{x_i}, \boldsymbol{z})^2}$

- If it is weighted, we may consider all the instances. (global)
- Can be slow.

K-nearest-neighbor algorithm

- A note on Euclidian distance.

- Attributes may have different units.
- Sometimes scaling does not change the underlying relationship, but it changes the Euclidian distance.



- Normalize the attributes.
- E.g. by the standard deviation (try to make each attribute equally important)

K-nearest-neighbor algorithm

- A note on Euclidian distance.

- Normalize the attributes.
- E.g. by the standard deviation (try to make each attribute equally important)
- If you believe they are not equally important?
- Put a weight then.

- $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{\sum w_i \left(x_1^i - x_2^i\right)^2}$

General Distance

- Metric Space:
- A metric $d(\boldsymbol{x_1}, \boldsymbol{x_2})$ is a real-valued function defined over the pairs in the space.
- It must satisfy:
- (Positive) $d(\boldsymbol{x_1}, \boldsymbol{x_2}) \geq 0$
- (Reflective) $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = 0$ iff $\boldsymbol{x_1} = \boldsymbol{x_2}$
- (Symmetric) $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = d(\boldsymbol{x_2}, \boldsymbol{x_1})$

- Minkowski distance, L_k form
- $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \left(\sum\left(x_1^i - x_2^i\right)^k\right)^{\frac{1}{k}}$

L_2 form: Euclidian Form

Irrelevant Feature

- What if some attribute is irrelevant.
- Decision Tree: use a subset of attributes.
- K-NN always uses all the attributes
- $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{\sum \left(x_1^i - x_2^i\right)^2}$

- Suppose you know which attribute is not good.
- Put a weight then.
- $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{\sum w_i \left(x_1^i - x_2^i\right)^2}$

Day	Outlook	Temperature
D1	Sunny	Hot
D2	Sunny	Hot
D3	Overcast	Hot
D4	Rain	Mild
D5	Rain	Cool
D6	Rain	Cool
D7	Overcast	Cool
D8	Sunny	Mild
D9	Sunny	Cool
D10	Rain	Mild
D11	Sunny	Mild
D12	Overcast	Mild
D13	Overcast	Hot
D14	Rain	Mild

Irrelevant Feature

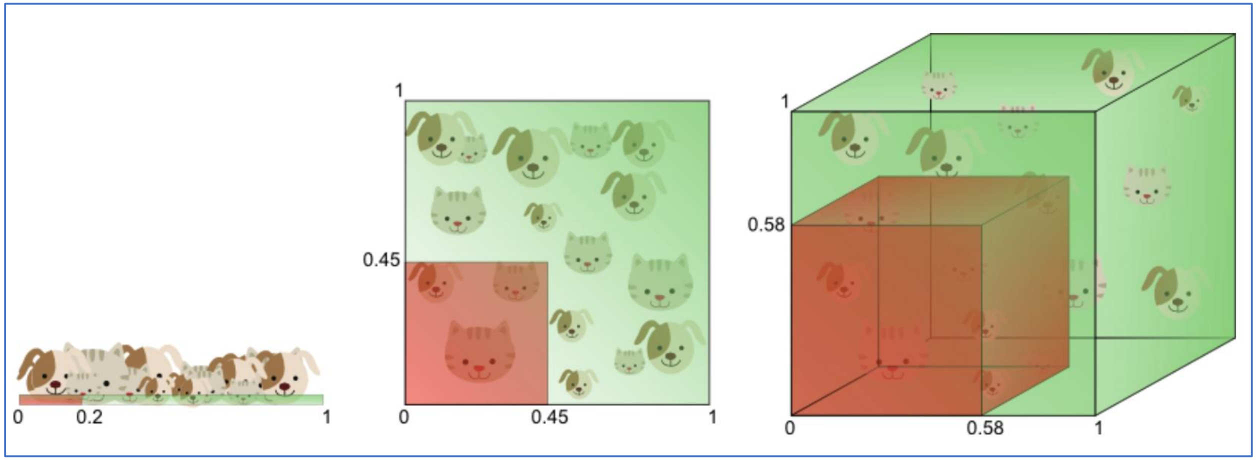
- If you **do not** know which attribute is not good.
- Put a weight and learn it.
- $d(\boldsymbol{x_1}, \boldsymbol{x_2}) = \sqrt{\sum w_i \left(x_1^i - x_2^i\right)^2}$

- Setting some initial values for w_i
- Repeat
  - Partition your dataset into training set and testing set.
  - Update the w_i so that the error on testing set is minimized.

- Note: Who comes when accuracy increases? Over-fitting.

Curse of Dimensionality – Sparse Sample

- Higher Dimensions <=> More features
- More features may be good, but a higher dimension may not be.



- The number of samples to cover half of the space grows exponentially with the increase of dimension.
- Link: http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

Efficient Implementations - Kd-trees

- How to implement a KNN?
- Find the k nearest neighbor?

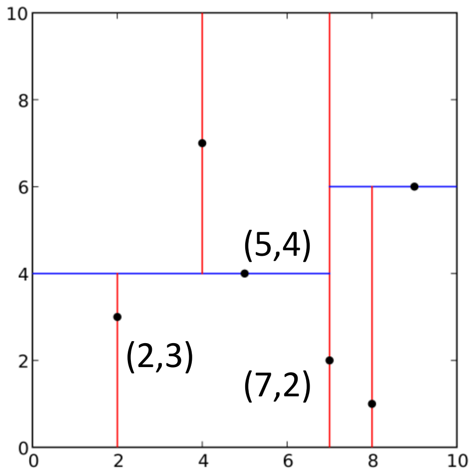
- Naïve method: compute the distance, sort the distances, find the k nearest.

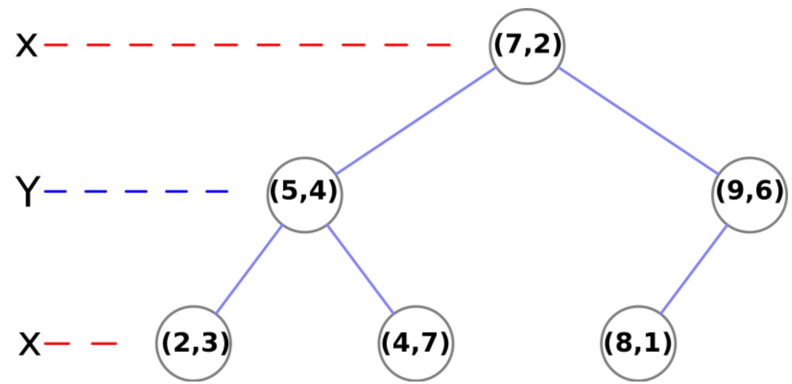
- A Kd-tree is a data structure storing the training data. For a new point, it can search the nearest instance in the tree.

- Better method: build a Kd-tree and find the k nearest neighbors.

Efficient Implementations - Kd-trees

- Example (from Wiki, KiwiSunset and MYguel)
- A kd-tree splits the space using the median value along the dimension having the highest variance, and points are stored at the leaves.





- KD tree materials: [link 1], [link 2], [wiki].

Reduce the Training Data Size.

- Storing all of the training examples can require a huge amount of memory. Select a subset of points that still give good classifications.

- **Incremental deletion.** Loop through the training data and test each point to see if it can be correctly classified given the other points. If so, delete it from the data set.
- (If it is correctly classified, the information it can provide has been stored in the training data)

- **Incremental growth.** Start with an empty data set. Add each point to the data set only if it is not correctly classified by the points already stored.

KNN Summary.

- Efficient Learning (if you can find the neighbors fast).
- No strong prior knowledge needed. (however, you believe there is relationship between the distance and target value)

$$|f(x_1) - f(x_2)| \to 0 \text{ if } dist(x_1, x_2) \to 0$$

- How to design distance?
- Cannot handle so many features.

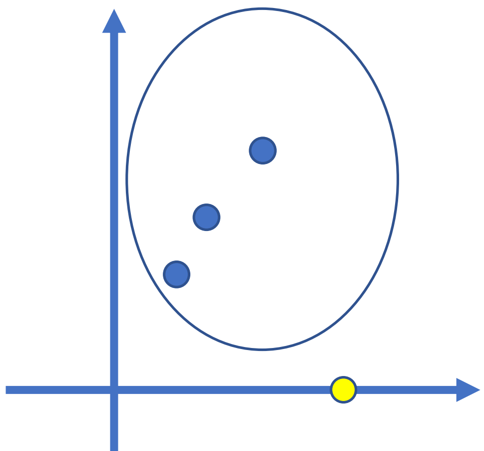
Locally Weighted Regression

- KNN: majority of the class value, or weighted sample mean.
- Somehow: a point estimation.
- More general: construct a local approximate function f around in new instance.

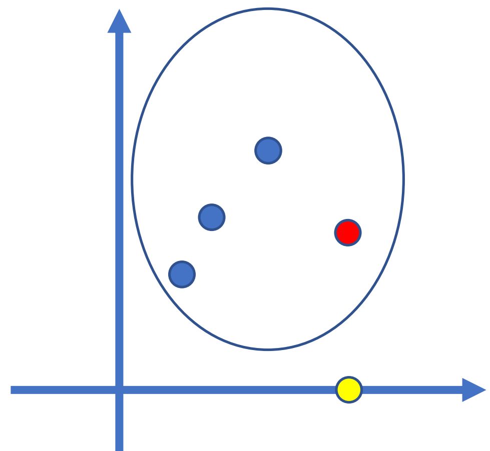
Locally Weighted Regression

- KNN: majority of the class value, or weighted sample mean.
- Somehow: a point estimation.
- More general: construct a local approximate function f around in new instance.

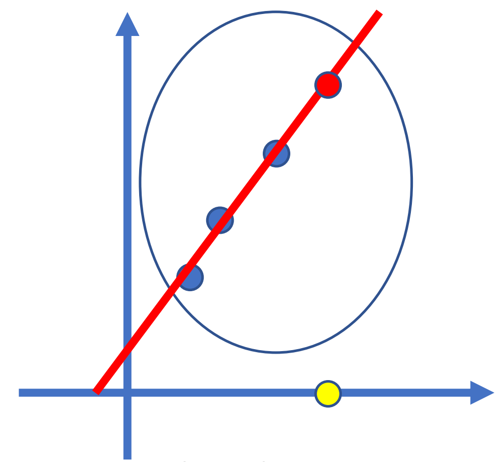
Which one is better?



predict here for x



KNN



a local regression

Locally Weighted Regression

- KNN: majority of the class value, or weighted sample mean.
- Somehow: a point estimation.
- More general: construct a local approximate function f around in new instance.

- f can be linear, quadratic, or other form you believe is good.

- Given a new instance z.
- Step 1: select a form for f
- Step 2: decide the weight of the neighbors of z
- Step 3: compute the parameter by using the weighted neighbors.

Locally Weighted Regression

- Locally Weighted Linear Regression

- Given a new instance z, assume
- $f(\boldsymbol{z}) = \omega_0 + \omega_1\, z^1 + \cdots + \omega_n\, z^n$

- $K(d(\boldsymbol{x}, \boldsymbol{z}))$ kernel function. Monotone decreasing. Assign a weight to $\boldsymbol{x}$.
- K is larger -> x and z are "close"
- E.g. $K\big(d(\boldsymbol{x}, \boldsymbol{z})\big) = \frac{1}{d(\boldsymbol{x}, \boldsymbol{z})^2}$

Locally Weighted Regression

- Locally Weighted Linear Regression

- Given a new instance z, assume
- $f(\boldsymbol{z}) = \omega_0 + \omega_1\, z^1 + \cdots + \omega_n\, z^n$
- $K(d(\boldsymbol{x}, \boldsymbol{z}))$ kernel function. Monotone decreasing. Assign a weight to $\boldsymbol{x}$.

- $Error(\boldsymbol{z}) = \frac{1}{2} \sum_{x_i \in k \text{ nearest of } \boldsymbol{z}} \left(y_i - f(\boldsymbol{x_i})\right)^2 K(d(\boldsymbol{x_i}, \boldsymbol{z}))$

- Find the $\boldsymbol{\omega}$ can minimize the error. Gradient descent.

Instance Based Learning

- KNN algorithm
- Distance
- Locally Weighted Regression
  - Locally weighted linear regression
  - Can you do locally “other” regression?
  - Remark: other forms are usually not considered.
    - Cost is high
    - Linear performs well. Why? [Theoretically, continuous functions can be locally approximated by linear function.]

Lazy vs Eager Methods

- It is all about how to do generalization.

- **Lazy methods**: decide the generalization when new instance comes.
  - Training is less needed and predicting requires more computation

- **Eager methods**: the generalization has been decided before new instance comes.
  - Training is computationally costly but predicting can be efficient.