Nearest neighbor classifier

Subhransu Maji

CMPSCI 689: Machine Learning

29 January 2015

3 February 2015

Topics of interest

hw00 poll

- NLP 13
- Deep learning, neural networks 8
- Computer vision 8
- Information retrieval 8
- Databases, systems, networking 4
- AI 3
- Reinforcement learning 3
- Robotics 3
- These got 1 or 2 mentions:
  - complexity, logic, large scale learning, speech, cross modality, biology, neuroscience, graphics, recommender systems, semi-supervised learning, programming languages, virtual reality, privacy, security

Topics of interest

hw00 poll

- NLP 13
- Deep learning, neural networks 8
- Computer vision 8
- Information retrieval 8
- Databases, systems, networking 4
- AI 3
- Reinforcement learning 3
- Robotics 3
- These got 1 or 2 mentions:
  - complexity, logic, large scale learning, speech, cross modality, biology, neuroscience, graphics, recommender systems, semi-supervised learning, programming languages, virtual reality, privacy, security

"To pass the class with a B+"

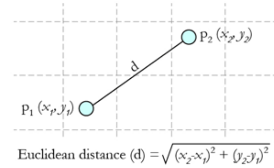
Nearest neighbor classifier

Nearest neighbor classifier

- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.

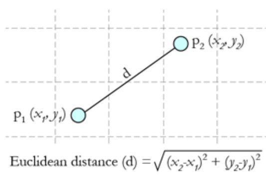
Nearest neighbor classifier

- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity



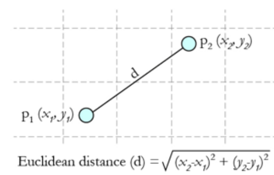
Nearest neighbor classifier

- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity
- Data to feature vectors



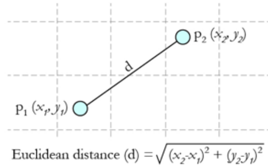
Nearest neighbor classifier

- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity
- Data to feature vectors
  - **Binary:** e.g. AI? {no, yes}
    - {0,1}
    - or {-20, 2}



Nearest neighbor classifier

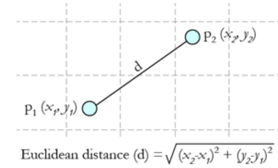
- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity



- Data to feature vectors
  - **Binary:** e.g. AI? {no, yes}
    - {0,1}
    - or {-20, 2} ✗

Nearest neighbor classifier

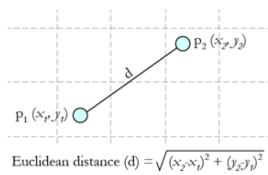
- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity



- Data to feature vectors
  - **Binary:** e.g. AI? {no, yes}
    - {0,1}
    - or {-20, 2} ✗
  - **Nominal:** e.g. color = {red, blue, green, yellow}
    - $\{0,1\}^n$
    - or {0,1,2,3}

Nearest neighbor classifier

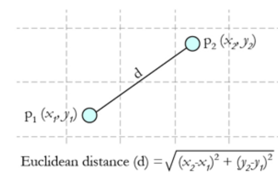
- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity



- Data to feature vectors
  - **Binary:** e.g. AI? {no, yes}
    - {0,1}
    - or {-20, 2} ✗
  - **Nominal:** e.g. color = {red, blue, green, yellow}
    - $\{0,1\}^n$
    - or {0,1,2,3} ✗

Nearest neighbor classifier

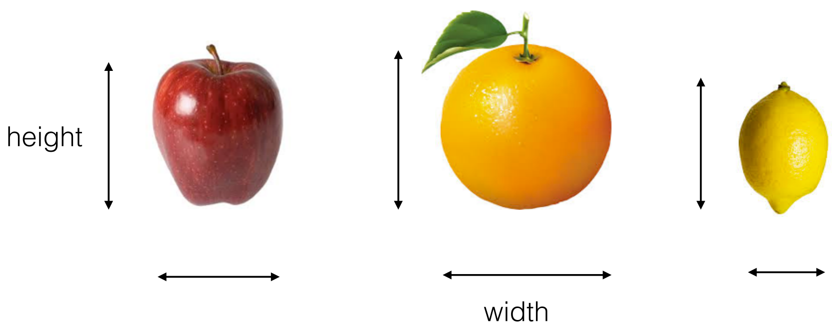
- Will Alice like AI?
  - Alice and James are similar and James likes AI. Hence, Alice must also like AI.
- It is useful to think of data as feature vectors
  - Use Euclidean distance to measure similarity



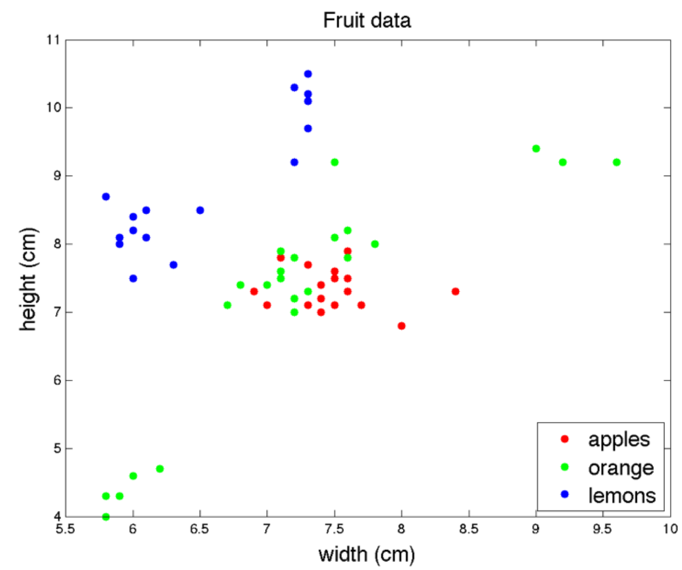
- Data to feature vectors
  - **Binary:** e.g. AI? {no, yes}
    - {0,1}
    - or {-20, 2} ✗
  - **Nominal:** e.g. color = {red, blue, green, yellow}
    - $\{0,1\}^n$
    - or {0,1,2,3} ✗
  - **Real valued:** e.g. temperature
    - copied
    - or {low, medium, high}

Nearest neighbor classifier

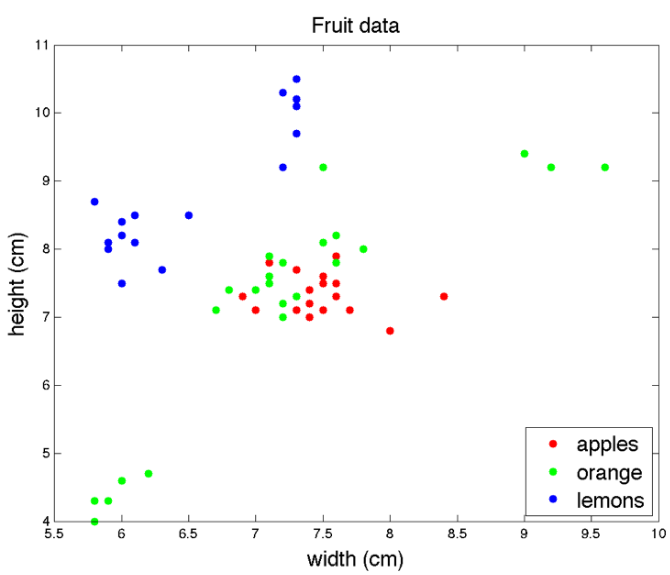
- Training data is in the form of $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
- Fruit data:
  - label: {apples, oranges, lemons}
  - attributes: {width, height}
- Euclidean distance $d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_i (\mathbf{x}_{1,i} - \mathbf{x}_{2,i})^2}$



Nearest neighbor classifier



Nearest neighbor classifier

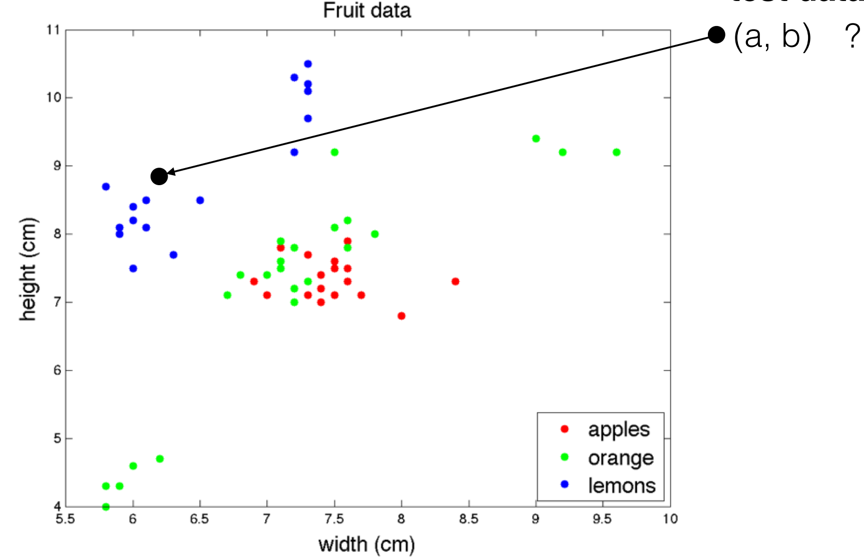


test data

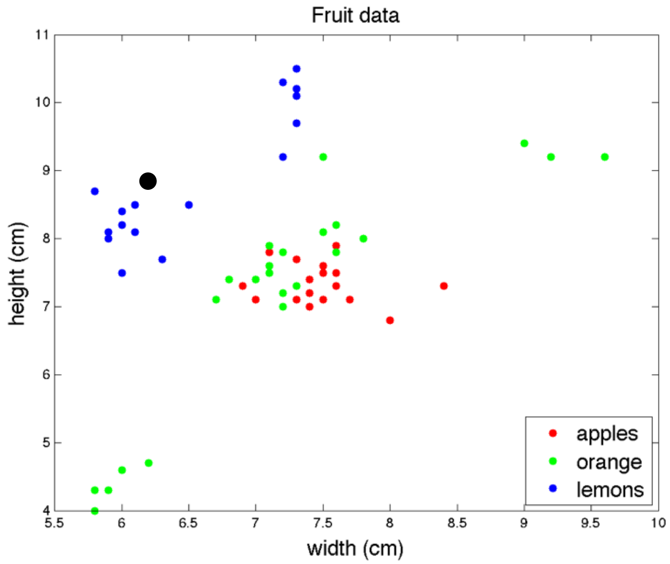
● (a, b) ?

Nearest neighbor classifier

test data



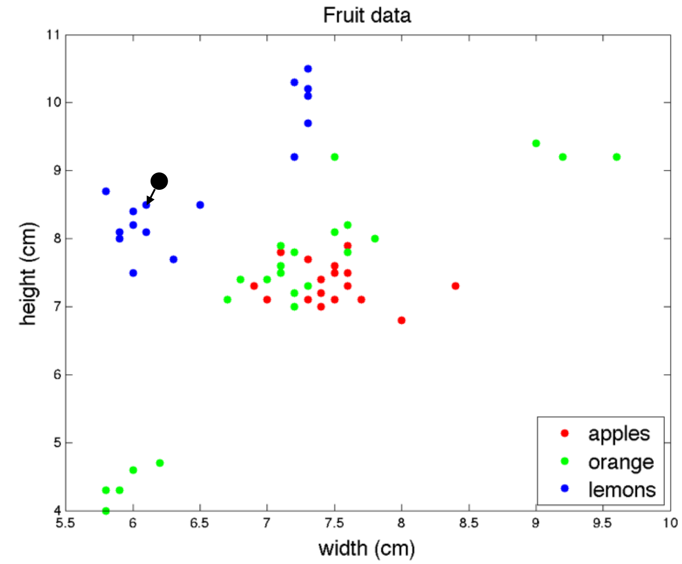
Nearest neighbor classifier



test data

- (a, b) ?

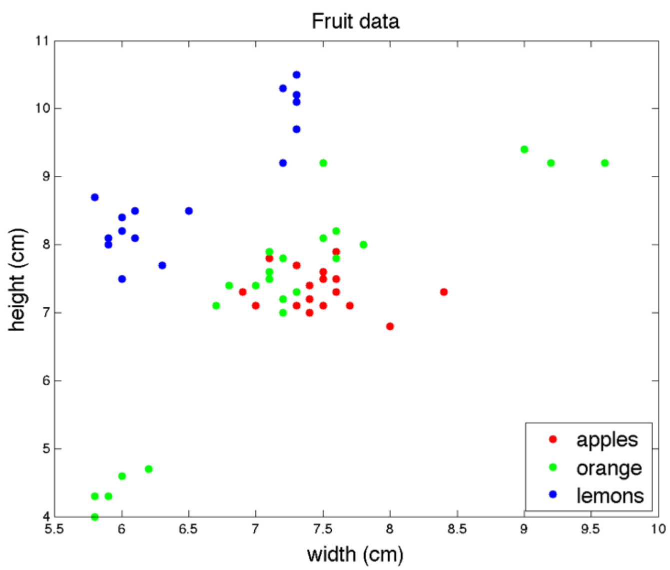
Nearest neighbor classifier



test data

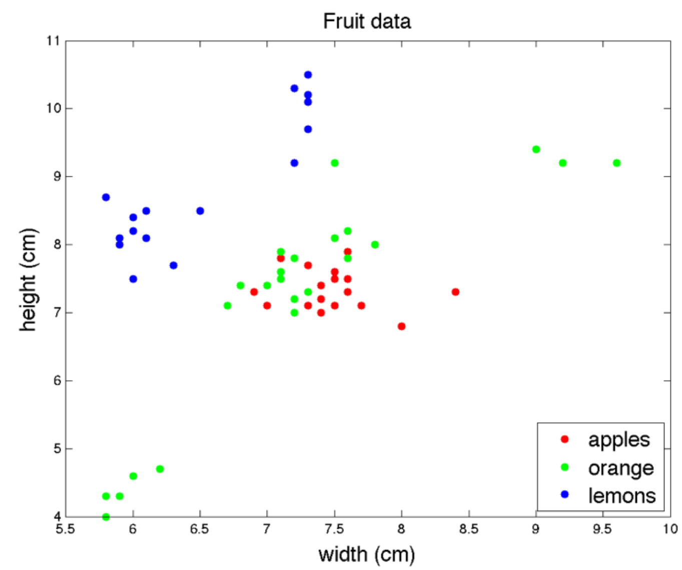
- (a, b) ? lemon

Nearest neighbor classifier



test data

Nearest neighbor classifier



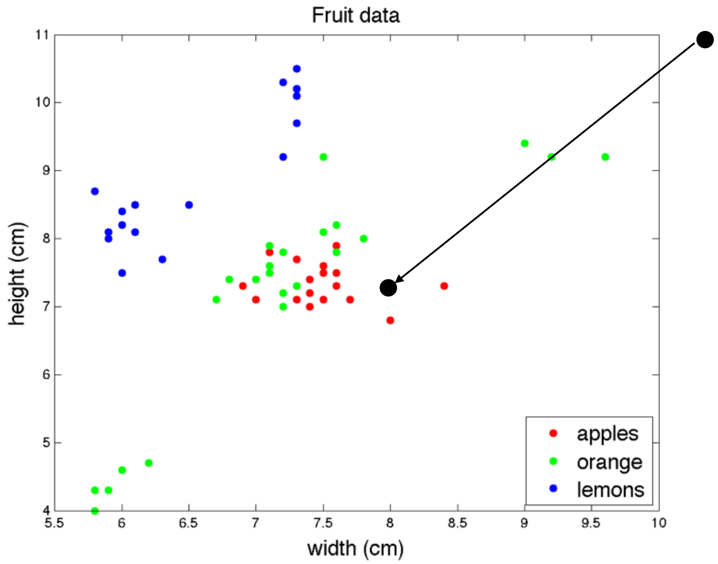
test data

- (c, d) ?

Nearest neighbor classifier

test data

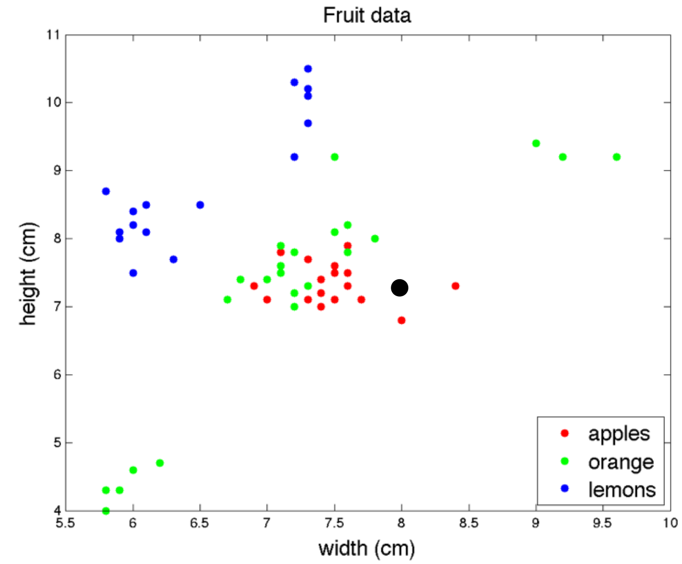
● (c, d) ?



Nearest neighbor classifier

test data

● (c, d) ?

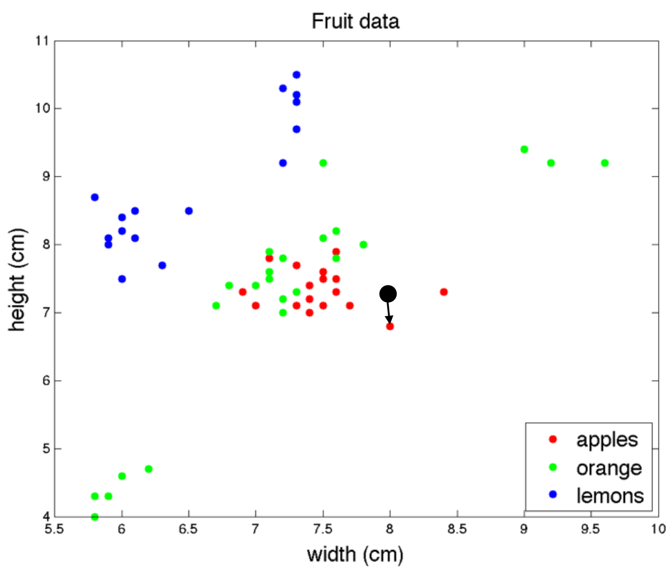


Nearest neighbor classifier

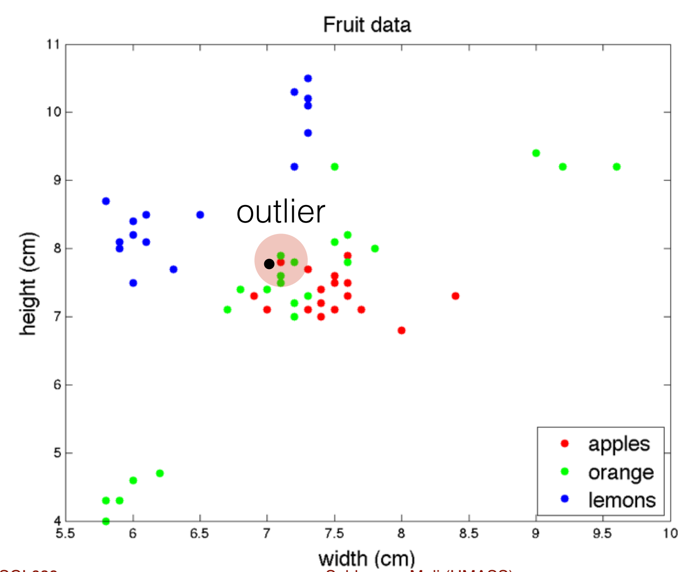
test data

● (c, d) ?

apple

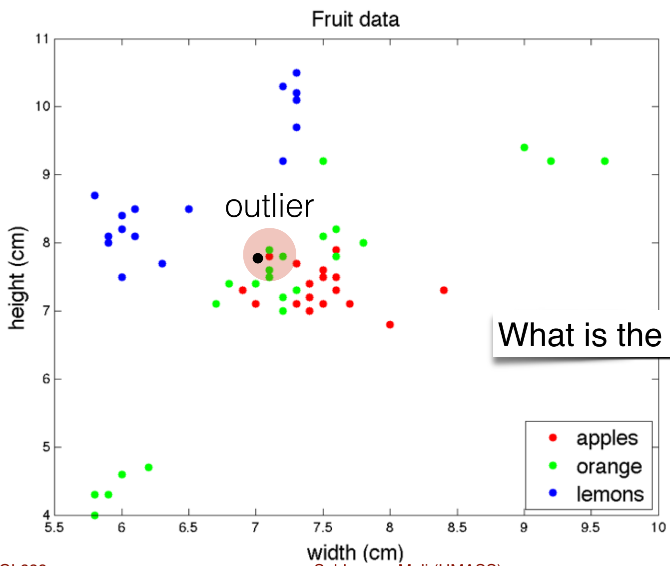


k-Nearest neighbor classifier



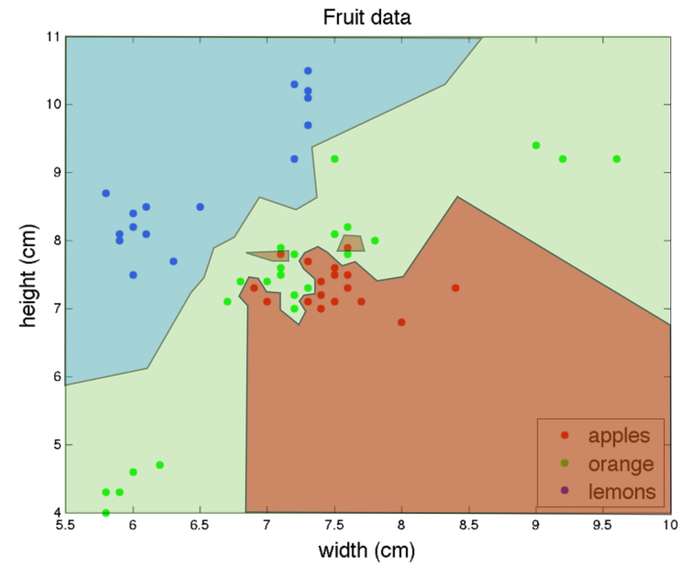
k-Nearest neighbor classifier

Take majority vote among the k nearest neighbors



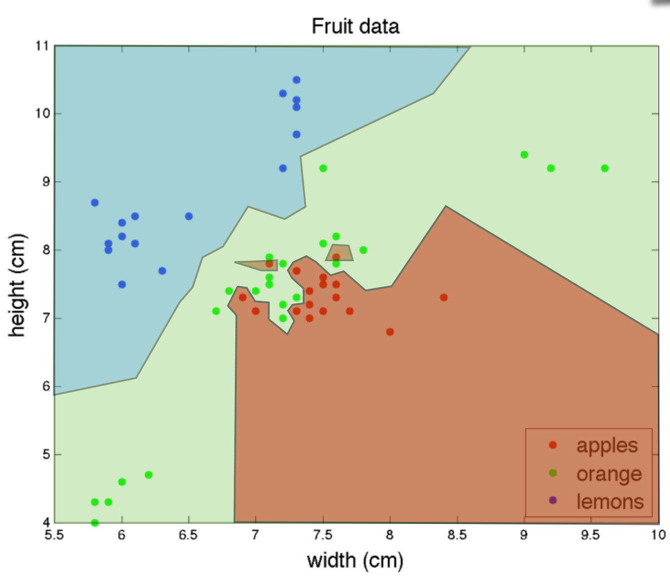
What is the effect of k?

Decision boundaries: 1NN

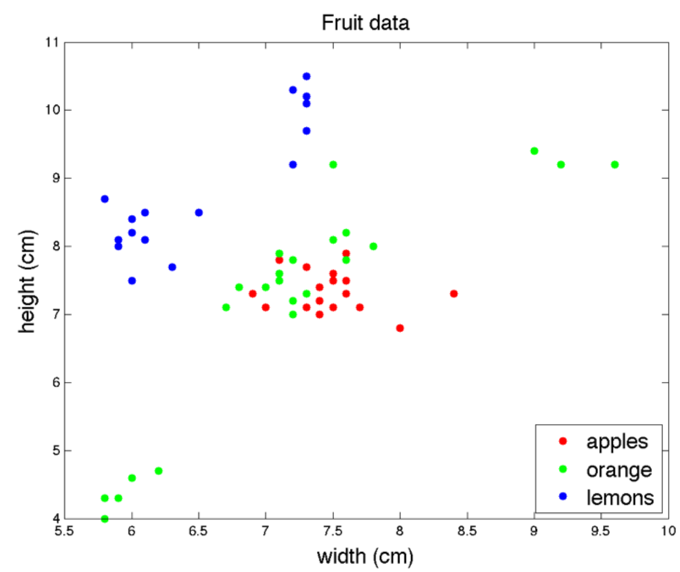


Decision boundaries: 1NN

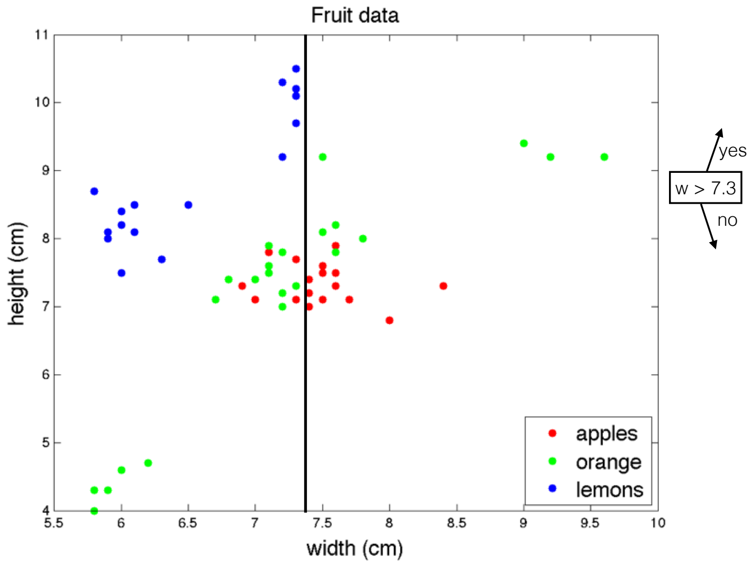
What is the effect of k?



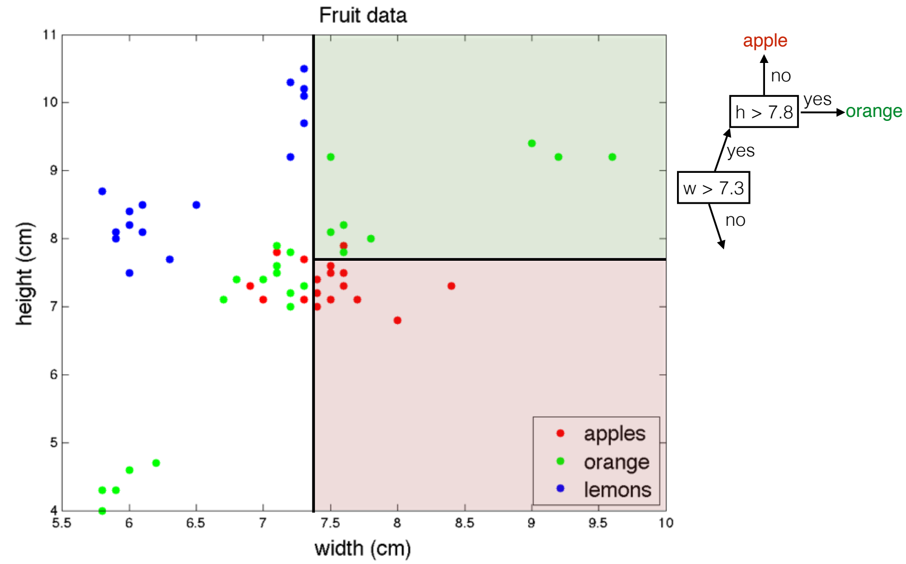
Decision boundaries: DT



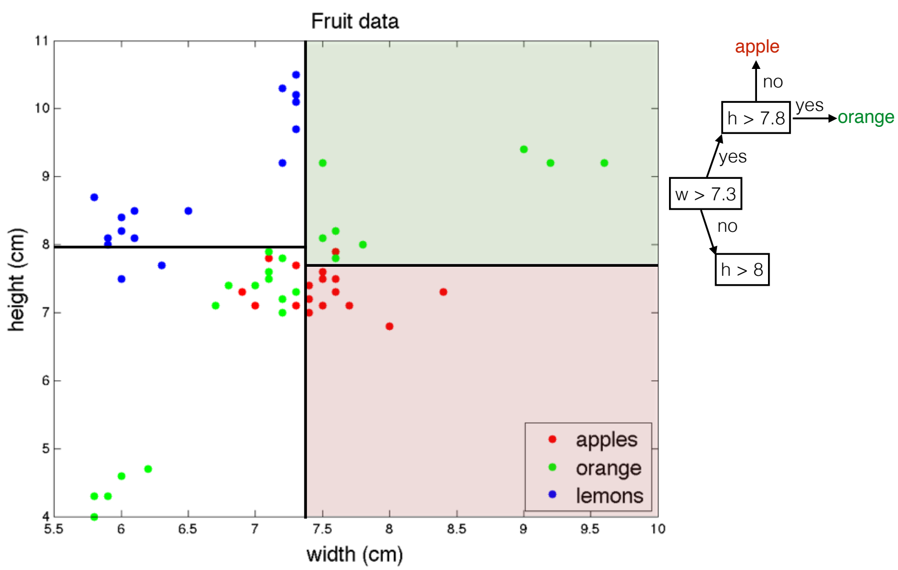
Decision boundaries: DT



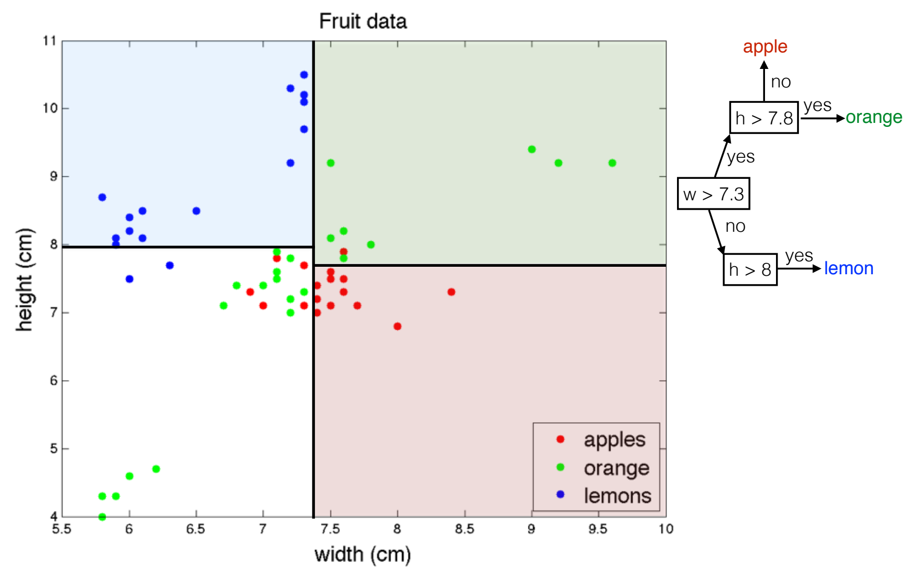
Decision boundaries: DT



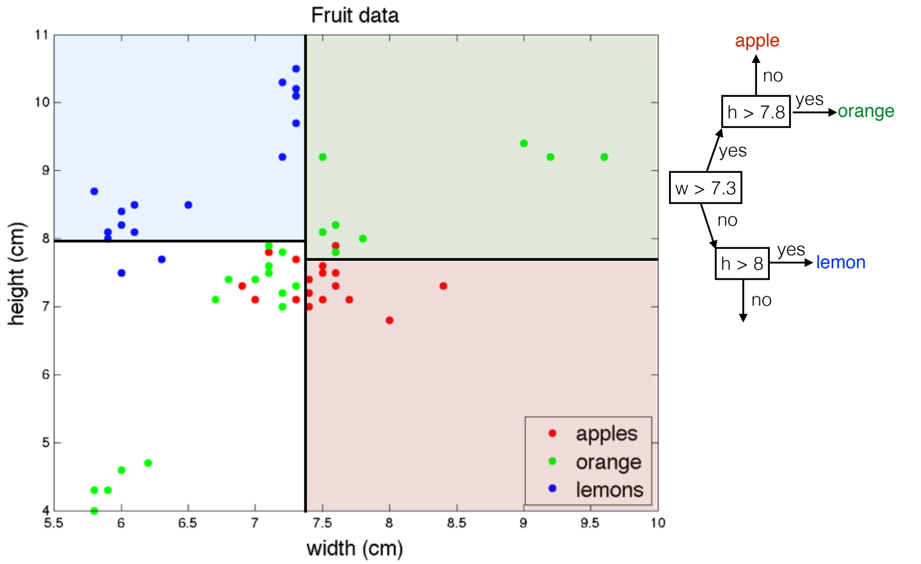
Decision boundaries: DT



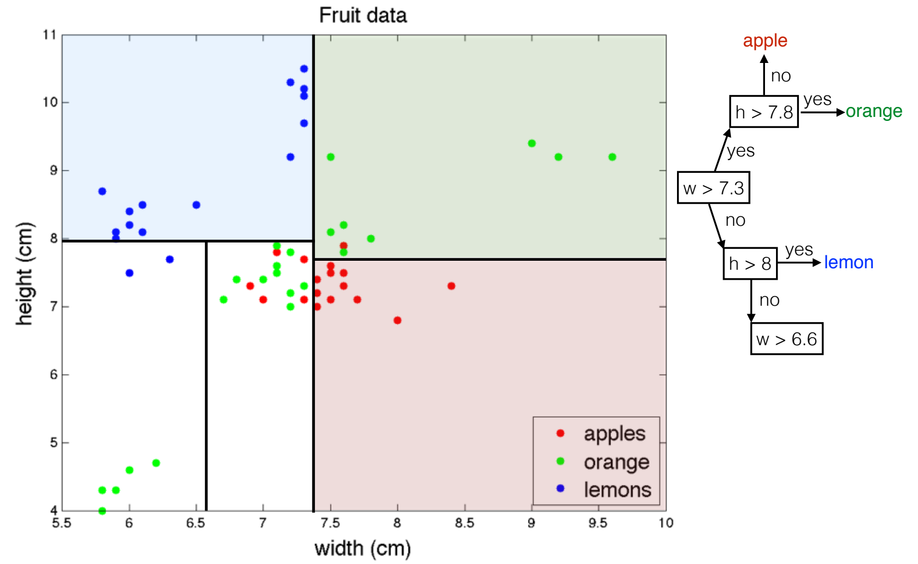
Decision boundaries: DT



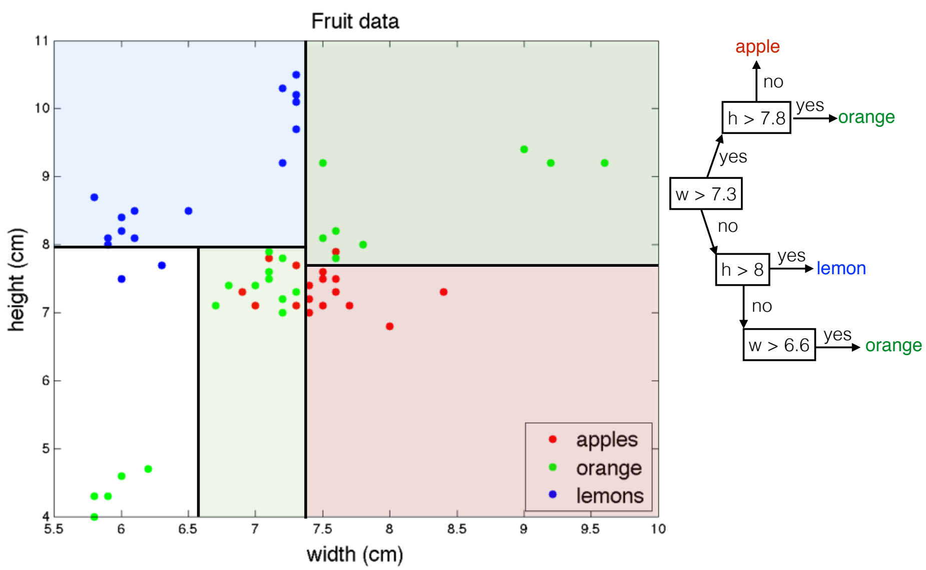
Decision boundaries: DT



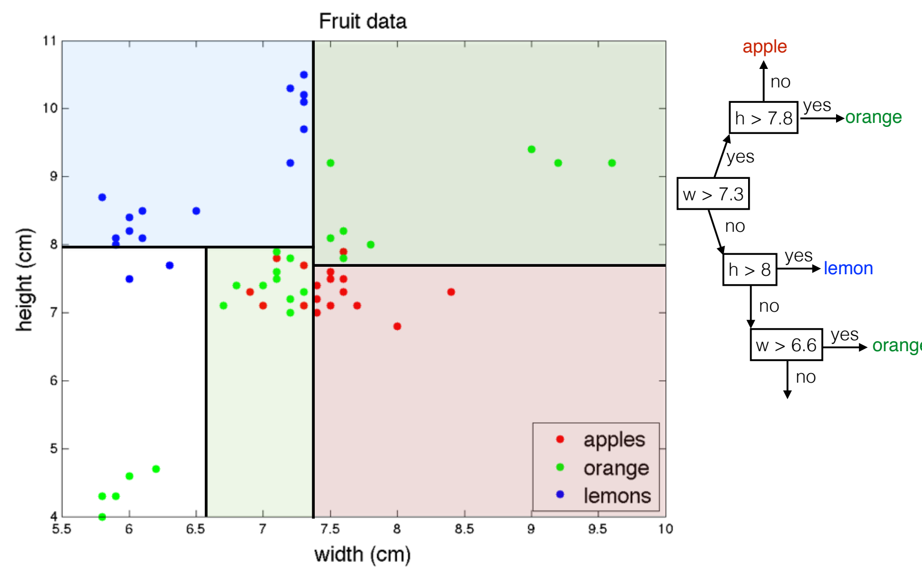
Decision boundaries: DT



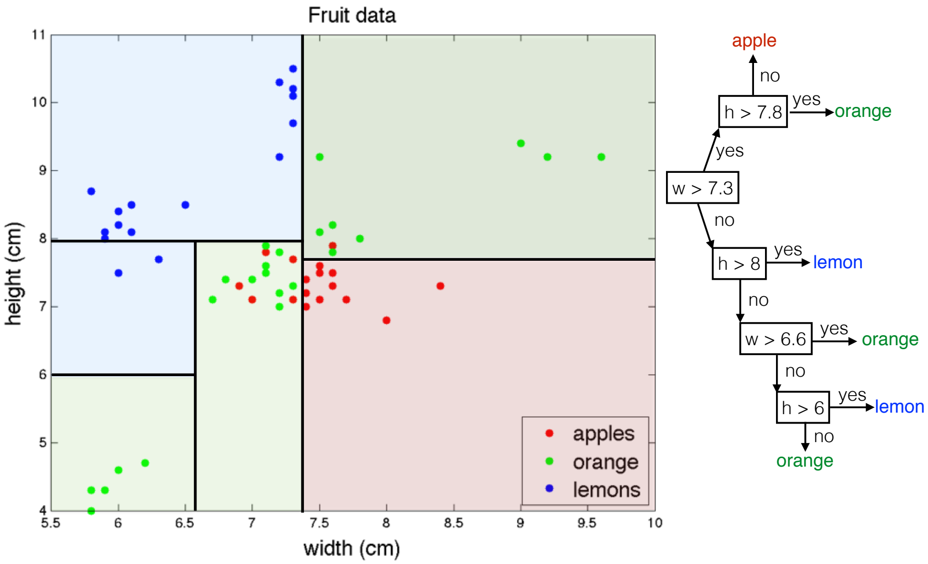
Decision boundaries: DT



Decision boundaries: DT

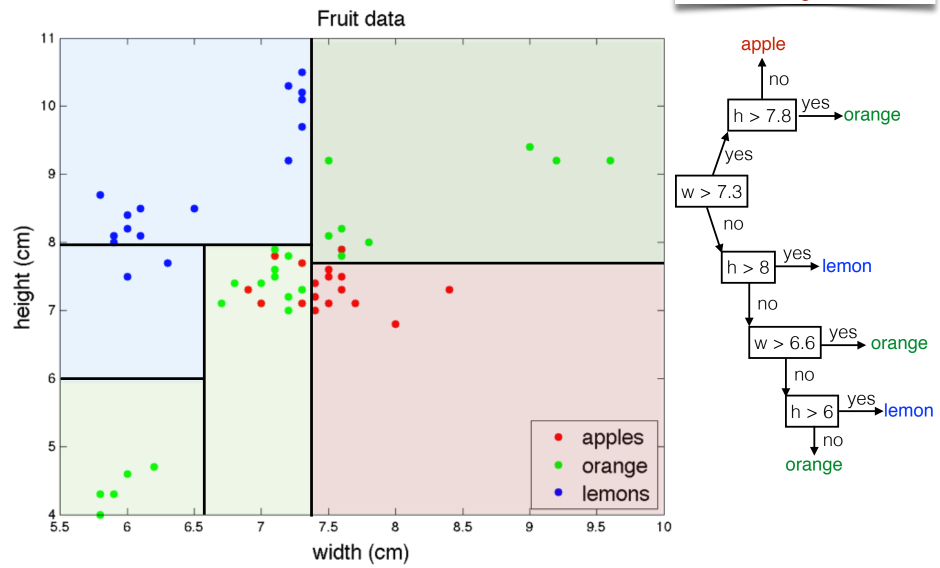


Decision boundaries: DT



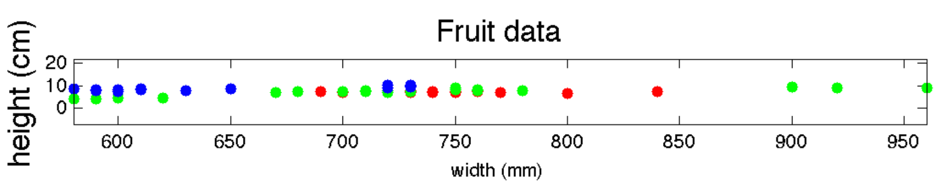
Decision boundaries: DT

The decision boundaries are axis aligned for DT



Inductive bias of the kNN classifier

- Choice of features
  - We are assuming that all features are equally important
  - What happens if we scale one of the features by a factor of 100?
- Choice of distance function
  - Euclidean, cosine similarity (angle), Gaussian, etc …
  - Should the coordinates be independent?
- Choice of k



An example

- "Texture synthesis" [Efros & Leung, ICCV 99]

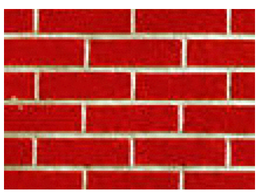
An example

- "Texture synthesis" [Efros & Leung, ICCV 99]

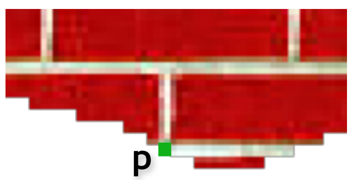
An example

- "Texture synthesis" [Efros & Leung, ICCV 99]

An example: Synthesizing one pixel



input image

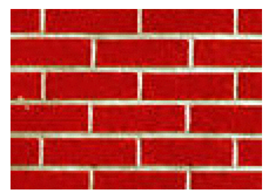


synthesized image

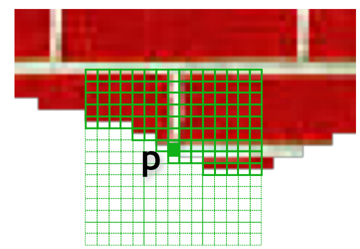
- What is $P(\mathbf{x}|\text{neighborhood of pixels around x})$?
- Find all the windows in the image that match the neighborhood
- To synthesize **x**
  - pick one matching window at random
  - assign **x** to be the center pixel of that window
- An **exact** match might not be present, so find the **best** matches using **Euclidean distance** and randomly choose between them, preferring better matches with higher probability

Slide from Alyosha Efros, ICCV 1999

An example: Synthesizing one pixel



input image

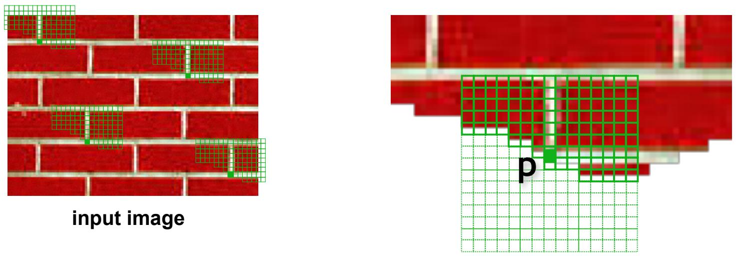


synthesized image

- What is $P(\mathbf{x}|\text{neighborhood of pixels around x})$?
- Find all the windows in the image that match the neighborhood
- To synthesize **x**
  - pick one matching window at random
  - assign **x** to be the center pixel of that window
- An **exact** match might not be present, so find the **best** matches using **Euclidean distance** and randomly choose between them, preferring better matches with higher probability

Slide from Alyosha Efros, ICCV 1999

An example: Synthesizing one pixel



input image

synthesized image

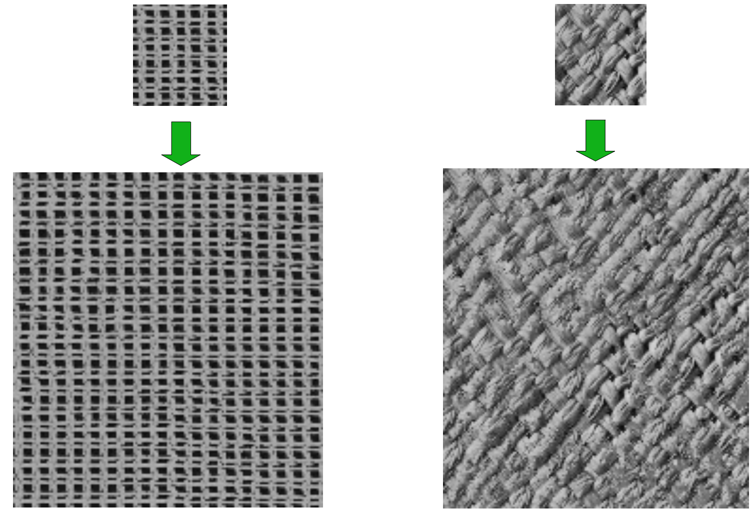
- What is $P(\mathbf{x}|\text{neighborhood of pixels around x})$?
- Find all the windows in the image that match the neighborhood
- To synthesize **x**
  - pick one matching window at random
  - assign **x** to be the center pixel of that window
- An **exact** match might not be present, so find the **best** matches using **Euclidean distance** and randomly choose between them, preferring better matches with higher probability

Slide from Alyosha Efros, ICCV 1999

An example: Synthesis results

french canvas

rafia weave

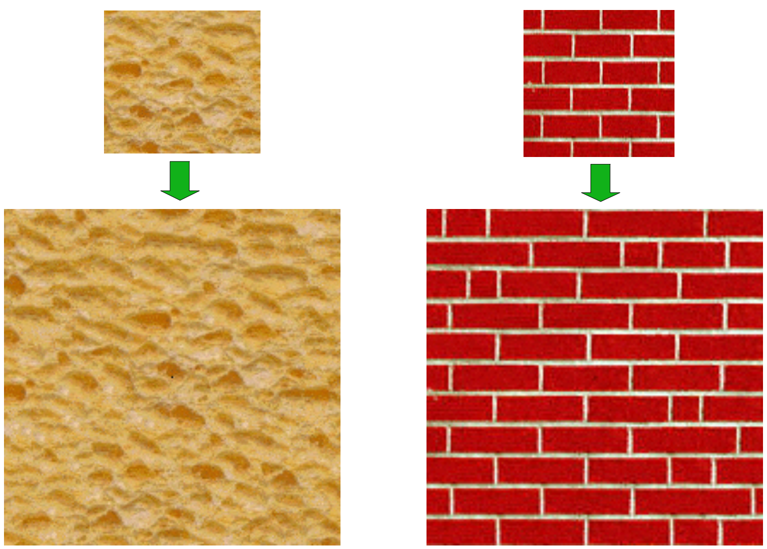


Slide from Alyosha Efros, ICCV 1999

An example: Synthesis results

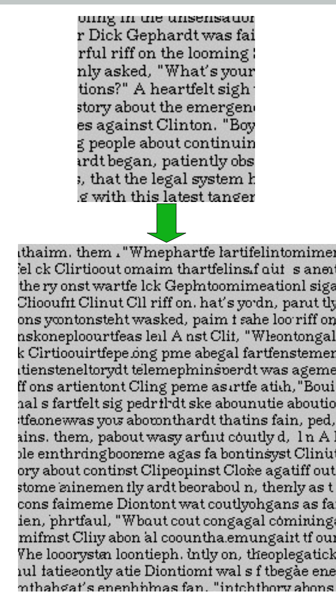
white bread

brick wall



Slide from Alyosha Efros, ICCV 1999

An example: Synthesis results



Slide from Alyosha Efros, ICCV 1999

An example: Growing Texture

- Starting from the initial image, "grow" one pixel at a time
  - **Application:** remove an object from the image

Slide from Alyosha Efros, ICCV 1999

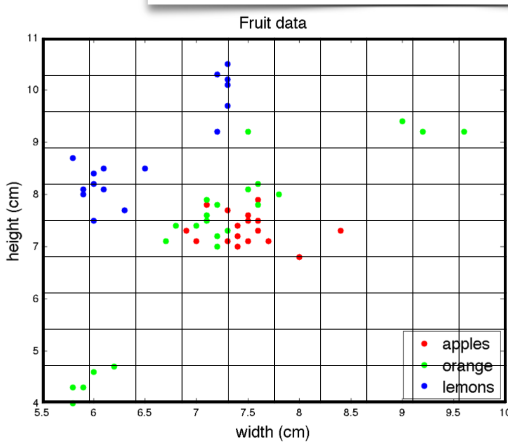
Practical issues when using kNN

- Curse of dimensionality
- Speed

Practical issues when using kNN

- **Curse of dimensionality**
- Speed

How many neighborhoods are there?

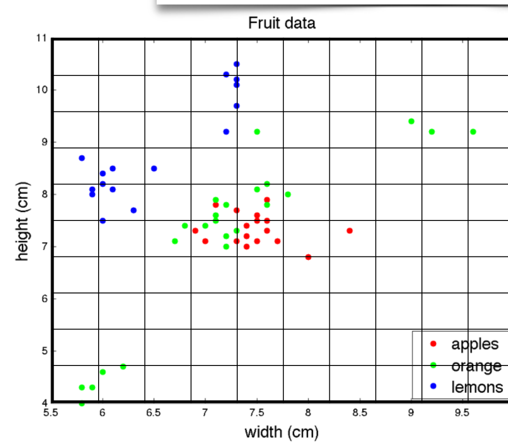


#bins = 10x10
d = 2

Practical issues when using kNN

- **Curse of dimensionality**
- Speed

How many neighborhoods are there?



#bins = 10x10
d = 2

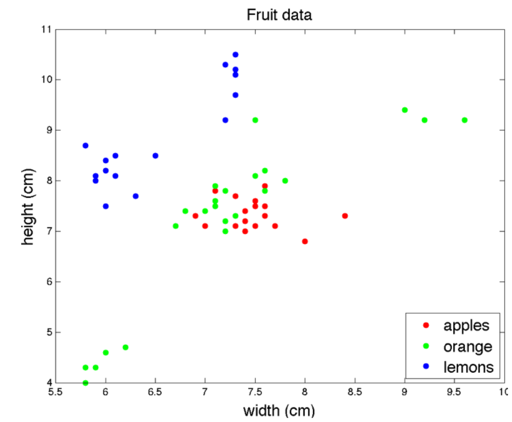
#bins = 10^d
d = 1000

Atoms in the universe
~ 10^{80}

Practical issues when using kNN

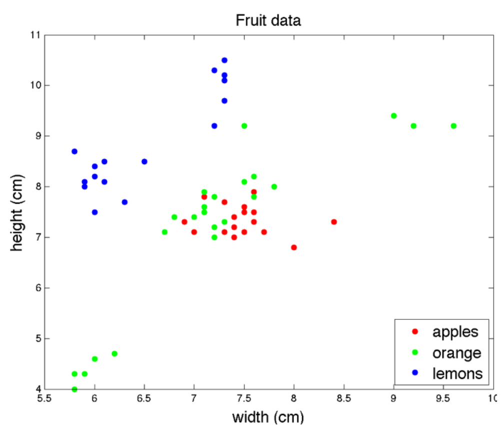
- Curse of dimensionality
- **Speed**
  - Time taken by kNN for N points of D dimensions
    - time to compute distances: O(ND)
    - time to find the k nearest neighbor
      - O(k N) : repeated minima
      - O(N log N) : sorting
      - O(N + k log N) : min heap
      - O(N + k log k) : fast median
    - Total time is dominated by distance computation
  - We can be faster if we are willing to sacrifice exactness

Approximate kNN



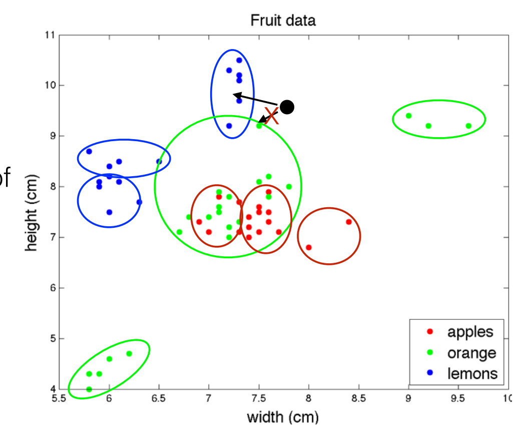
Approximate kNN

- Simplest idea is to cluster the data
  - Class → 3 clusters
  - Cluster → mean of points
  - Label of a test is the label of the nearest cluster mean



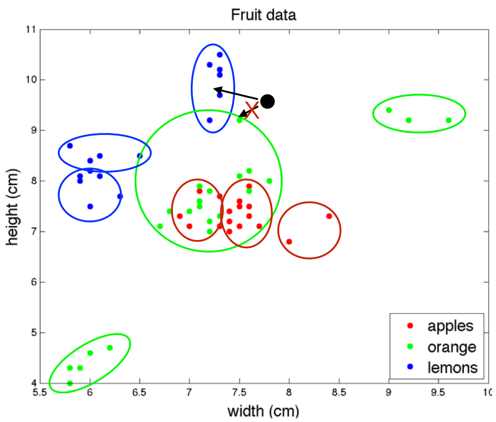
Approximate kNN

- Simplest idea is to cluster the data
  - Class → 3 clusters
  - Cluster → mean of points
  - Label of a test is the label of the nearest cluster mean



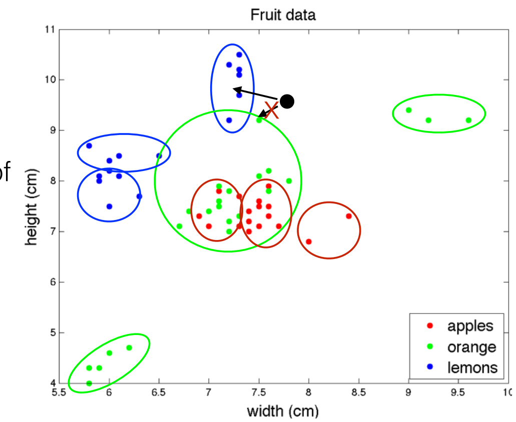
Approximate kNN

- Simplest idea is to cluster the data
  - Class → 3 clusters
  - Cluster → mean of points
  - Label of a test is the label of the nearest cluster mean
- Run time memory
  - ~~O(ND)~~ O(CD)
    - ➡ C << N



Approximate kNN

- Simplest idea is to cluster the data
  - Class → 3 clusters
  - Cluster → mean of points
  - Label of a test is the label of the nearest cluster mean
- Run time memory
  - ~~O(ND)~~ O(CD)
    - ➡ C << N



How do we cluster the data?

Clustering using k-means

Given $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, k-means clustering aims to partition the **n** observations into **k (≤ n)** sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares.

In other words, its objective is to find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

cluster center

http://en.wikipedia.org/wiki/K-means_clustering

Clustering using k-means

Given $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, k-means clustering aims to partition the **n** observations into **k (≤ n)** sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares.

In other words, its objective is to find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

cluster center

Easy to compute $\boldsymbol{\mu}$ given **S** and vice versa.

http://en.wikipedia.org/wiki/K-means_clustering

Lloyd's algorithm for k-means

- Initialize k centers by picking k points randomly
- Repeat till convergence (or max iterations)
  - Assign each point to the nearest center (assignment step)

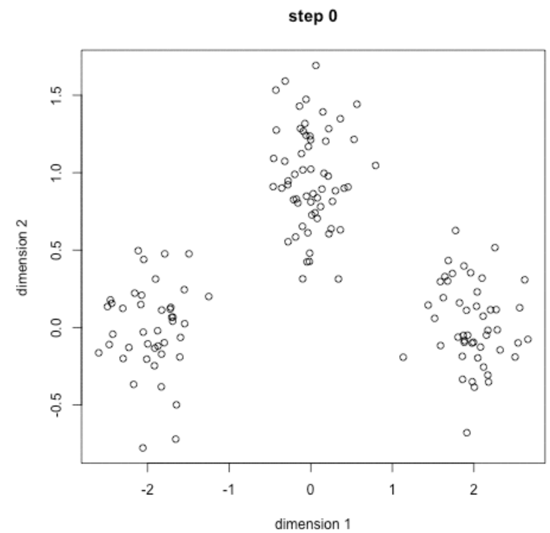
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \underline{\sum_{\mathbf{x}\in S_i}} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

  - Estimate the mean of each group (update step)

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \|\mathbf{x} - \underline{\boldsymbol{\mu}_i}\|^2$$

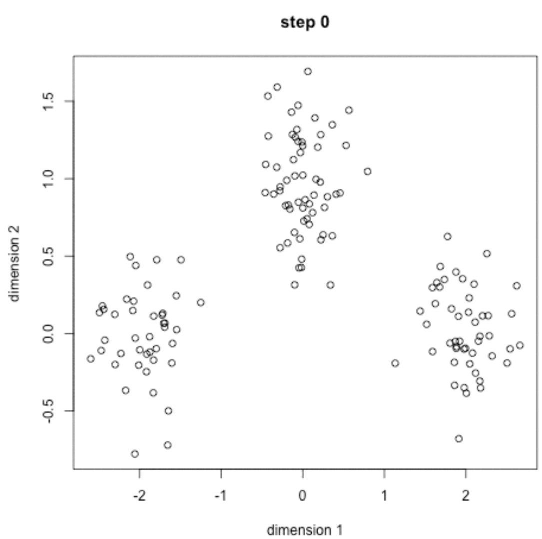
- Simple and works well in practice
  - Multiple initializations
  - Provably fast

K-means in action



http://simplystatistics.org/2014/02/18/k-means-clustering-in-a-gif/

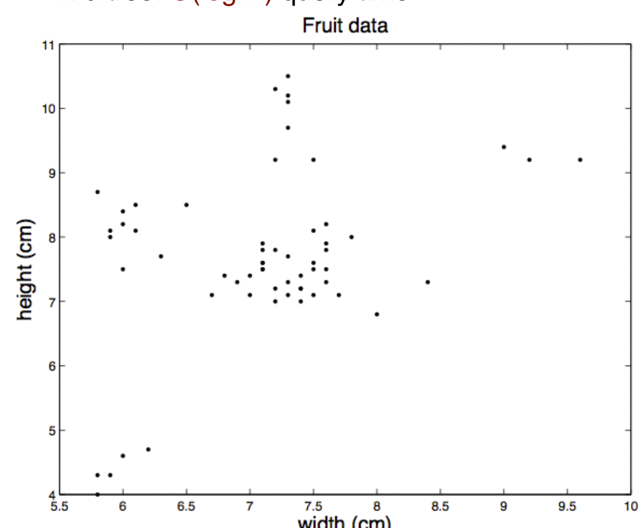
K-means in action



http://simplystatistics.org/2014/02/18/k-means-clustering-in-a-gif/

Approximate kNN

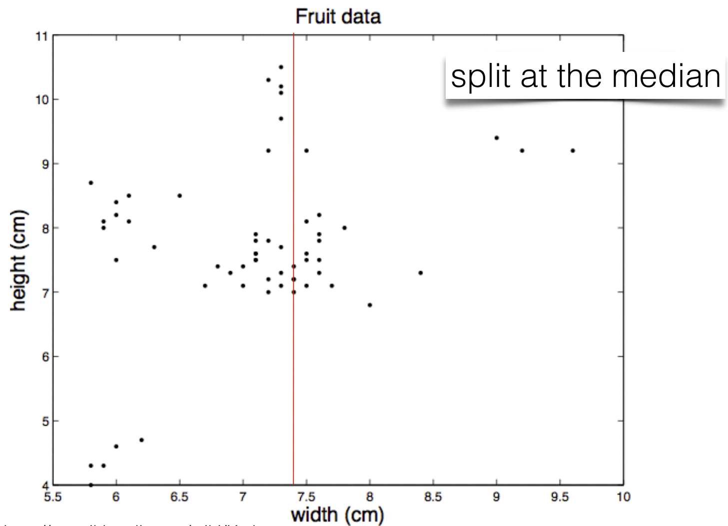
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

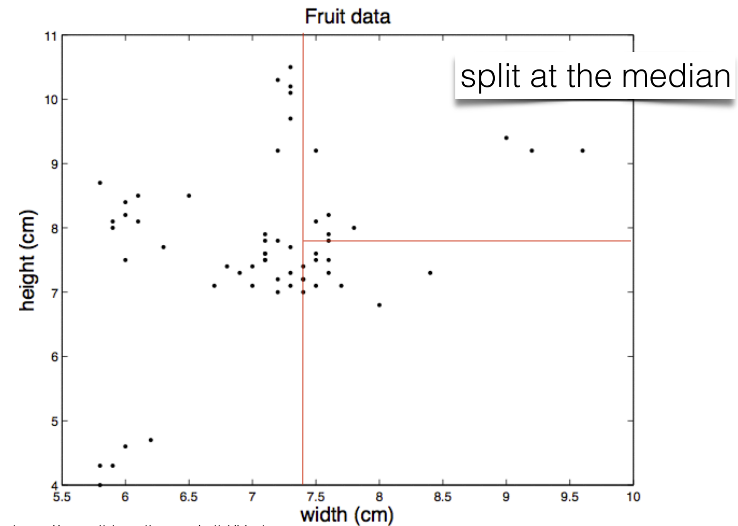
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

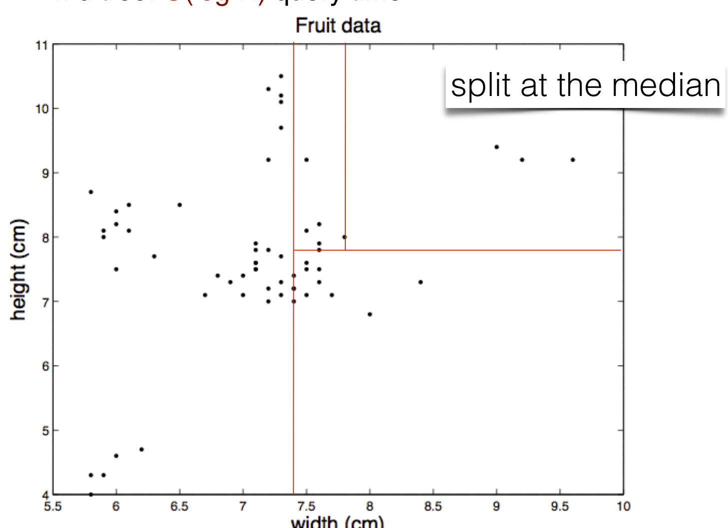
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

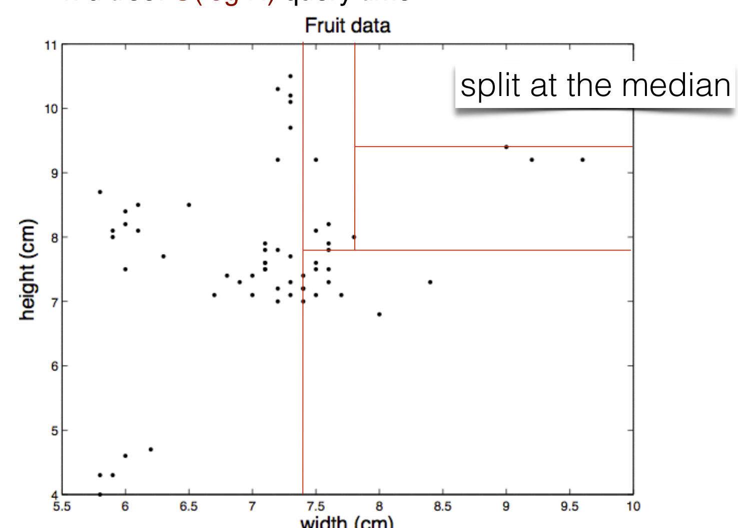
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

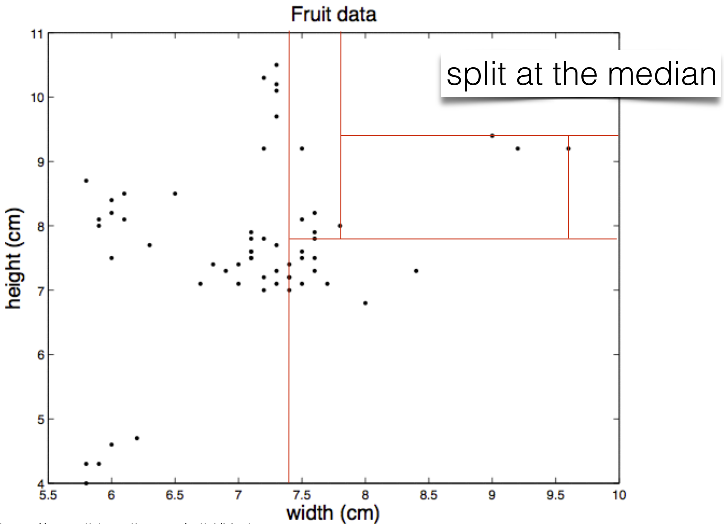
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

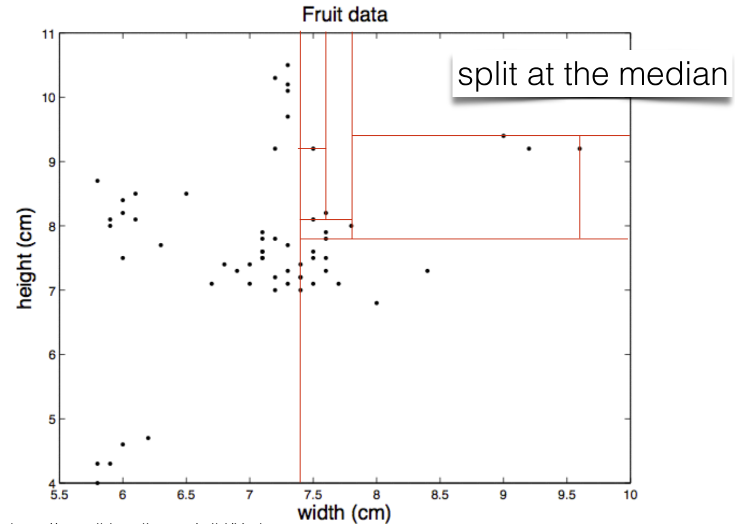
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

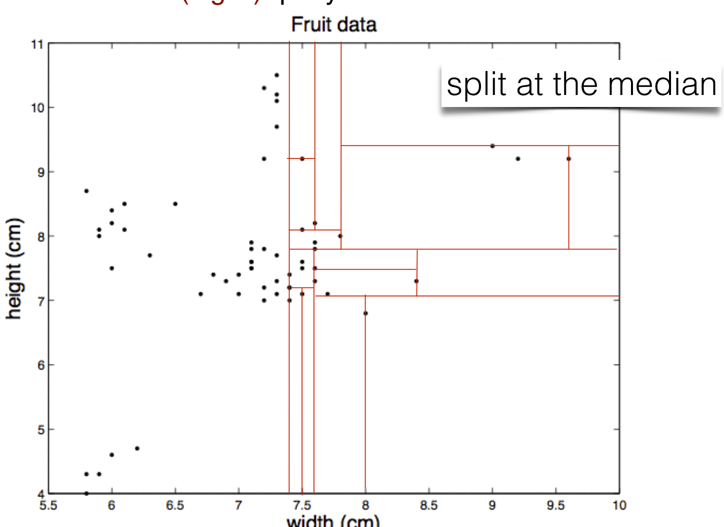
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

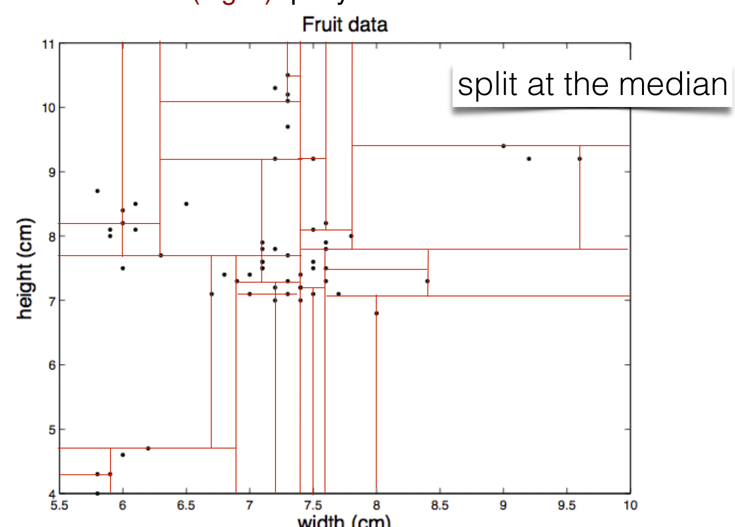
- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

- k-d tree: O(log N) query time

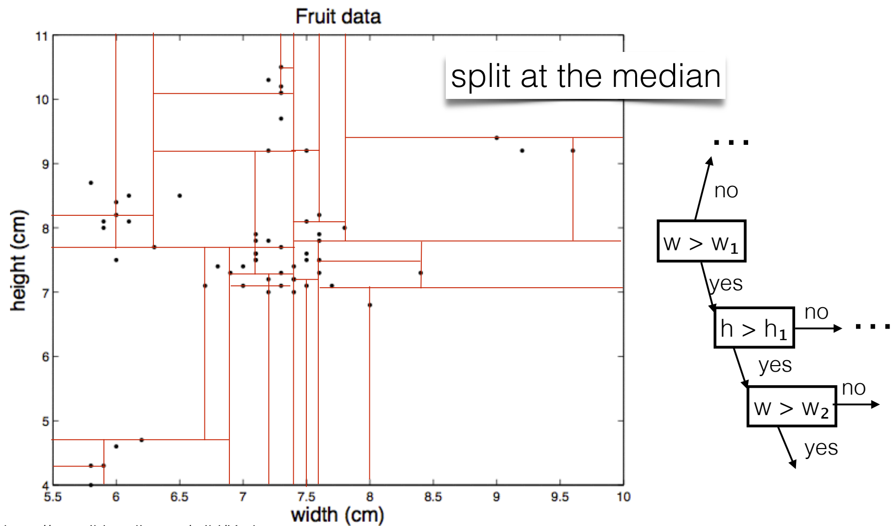


http://en.wikipedia.org/wiki/K-d_tree

Approximate kNN

Decision trees?

- k-d tree: O(log N) query time



http://en.wikipedia.org/wiki/K-d_tree

Summary of kNN

- Very simple setup
  - Training: none
  - Testing: find k nearest neighbors and take the majority class label
- An example of a non-parametric classifier: the number of parameters of the classifier *grow* with the size of the training data
- Practical issues
  - Curse of dimensionality: worst case dataset size grows $O(n^d)$
  - Speed: clustering (using k-means) and k-d trees as approximations
- kNN is likely to be competitive when:
  - the number of features are relatively small (< 20)
  - the distance metric is good
  - the dataset is large
- Research questions:
  - Learning a good metric
  - Testing speed: RP trees, locality sensitive hashing (LSH), ….

Not everything is learnable

- It may not be possible to get perfect classification on data
  - Measurement noise: sensors may be inaccurate
  - Information gap: Sometimes we just don't have enough information to make accurate predictions
    - e.g. Class ratings have high variance
      - Will students like AI? (70% yes, 30% no)
    - e.g. Image

3 or 7?

2 or 7?

The best error you can get is called the Bayes error

Lets do a bit of learning theory …

Bayes optimal classifier and error

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data $\ell(y, \hat{y})$: loss function

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data $\ell(y, \hat{y})$: loss function

$\epsilon(\hat{y}) = \mathbb{E}_{(\mathbf{x},y)\sim D}\left[\ell(y, \hat{y})\right]$: expected error of a predictor

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data $\quad \ell(y, \hat{y})$: loss function

$\epsilon(\hat{y}) = \mathbb{E}_{(\mathbf{x},y)\sim D}[\ell(y, \hat{y})]$: expected error of a predictor

$\epsilon(\mathbf{x}, \hat{y}) = \mathbb{E}_{y\sim D(y;\mathbf{x})}[\ell(y, \hat{y})]$: expected error of a predictor at $\mathbf{x}$

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data $\quad \ell(y, \hat{y})$: loss function

$\epsilon(\hat{y}) = \mathbb{E}_{(\mathbf{x},y)\sim D}[\ell(y, \hat{y})]$: expected error of a predictor

$\epsilon(\mathbf{x}, \hat{y}) = \mathbb{E}_{y\sim D(y;\mathbf{x})}[\ell(y, \hat{y})]$: expected error of a predictor at $\mathbf{x}$

$y^*(\mathbf{x}) = \arg\min_{\hat{y}} \epsilon(\mathbf{x}, \hat{y})$: Bayes optimal classifier

$\epsilon^*(\mathbf{x}) = \epsilon(\mathbf{x}, y^*)$: Bayes error

Bayes optimal classifier and error

$(\mathbf{x}, y) \sim D(\mathbf{x}, y)$: training data $\quad \ell(y, \hat{y})$: loss function

$\epsilon(\hat{y}) = \mathbb{E}_{(\mathbf{x},y)\sim D}[\ell(y, \hat{y})]$: expected error of a predictor

$\epsilon(\mathbf{x}, \hat{y}) = \mathbb{E}_{y\sim D(y;\mathbf{x})}[\ell(y, \hat{y})]$: expected error of a predictor at $\mathbf{x}$

$y^*(\mathbf{x}) = \arg\min_{\hat{y}} \epsilon(\mathbf{x}, \hat{y})$: Bayes optimal classifier

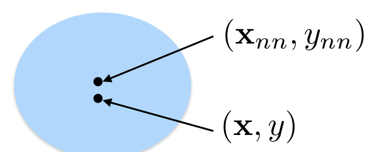
$\epsilon^*(\mathbf{x}) = \epsilon(\mathbf{x}, y^*)$: Bayes error

Binary classification $y \in \{0, 1\}$ $\quad \ell(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{otherwise} \end{cases}$

$$y^*(\mathbf{x}) = \arg\min_{\hat{y}} \left[D(y=0;\mathbf{x})\ell(0,\hat{y}) + D(y=1;\mathbf{x})\ell(1,\hat{y})\right]$$

$$y^*(\mathbf{x}) = \begin{cases} 0 & \text{if } D(y=0;\mathbf{x}) \geq 0.5 \\ 1 & \text{if } D(y=0;\mathbf{x}) < 0.5 \end{cases} \qquad \epsilon^*(\mathbf{x}) = 1 - D\left(y^*(\mathbf{x});\mathbf{x}\right)$$

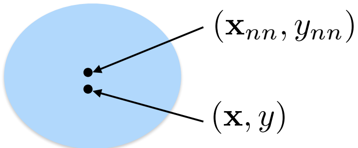
NN classifier is nearly optimal



As $n \to \infty$
$D(y_{nn};\mathbf{x}_{nn}) \to D(y;\mathbf{x})$

$$\begin{aligned}
\epsilon^1_{nn}(\mathbf{x}) &= P(y=1, y_{nn}=0;\mathbf{x},\mathbf{x}_{nn}) + P(y=0, y_{nn}=1;\mathbf{x},\mathbf{x}_{nn}) \\
&= D(y=1;\mathbf{x})D(y_{nn}=0;\mathbf{x}_{nn}) + D(y=0;\mathbf{x})D(y_{nn}=1;\mathbf{x}_{nn}) \\
&= 2D(y=1;\mathbf{x})D(y=0;\mathbf{x}) \\
&\leq 2\min\left(D(y=1;\mathbf{x}), D(y=0;\mathbf{x})\right) \\
&= 2\epsilon^*(\mathbf{x})
\end{aligned}$$

NN classifier is nearly optimal

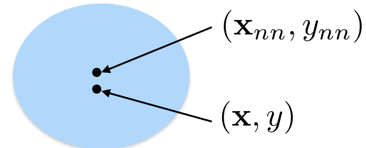


As $n \to \infty$
$D(y_{nn}; \mathbf{x}_{nn}) \to D(y; \mathbf{x})$

$$\begin{aligned}
\epsilon^1_{nn}(\mathbf{x}) &= P(y=1, y_{nn}=0; \mathbf{x}, \mathbf{x}_{nn}) + P(y=0, y_{nn}=1; \mathbf{x}, \mathbf{x}_{nn}) \\
&= D(y=1;\mathbf{x})D(y_{nn}=0;\mathbf{x}_{nn}) + D(y=0;\mathbf{x})D(y_{nn}=1;\mathbf{x}_{nn}) \\
&= 2D(y=1;\mathbf{x})D(y=0;\mathbf{x}) \\
&\le 2\min\left(D(y=1;\mathbf{x}), D(y=0;\mathbf{x})\right) \\
&= 2\epsilon^*(\mathbf{x})
\end{aligned}$$

$\epsilon^* \le \epsilon^1_{nn} \le 2\epsilon^*$ Cover-Hart, 1967

NN classifier is nearly optimal



As $n \to \infty$
$D(y_{nn}; \mathbf{x}_{nn}) \to D(y; \mathbf{x})$

$$\begin{aligned}
\epsilon^1_{nn}(\mathbf{x}) &= P(y=1, y_{nn}=0; \mathbf{x}, \mathbf{x}_{nn}) + P(y=0, y_{nn}=1; \mathbf{x}, \mathbf{x}_{nn}) \\
&= D(y=1;\mathbf{x})D(y_{nn}=0;\mathbf{x}_{nn}) + D(y=0;\mathbf{x})D(y_{nn}=1;\mathbf{x}_{nn}) \\
&= 2D(y=1;\mathbf{x})D(y=0;\mathbf{x}) \\
&\le 2\min\left(D(y=1;\mathbf{x}), D(y=0;\mathbf{x})\right) \\
&= 2\epsilon^*(\mathbf{x})
\end{aligned}$$

$\epsilon^* \le \epsilon^1_{nn} \le 2\epsilon^*$ Cover-Hart, 1967

For any $k \ge 5, \epsilon^* \le \epsilon^k_{nn} \le \epsilon^*\left(1 + \sqrt{\frac{2}{k}}\right)$ Devroye, 1981

Machine learning solved?

Not really …

- kNN is nearly optimal when there is infinite training data
  - Says nothing about the finite sample case
  - Note: not all classifiers are (nearly) optimal even with infinite data
- Bayes error is a function of features (**x**)
  - We can get better Bayes error if we choose different the features
    - If we had color in addition to the width and height, we would be able classify the fruits more accurately.
- How do we understand the performance of learners for the finite sample case?
  - Bias-variance decomposition

Bias-variance decomposition

- Standard way to decompose squared loss $\ell(y, \hat{y}) = (y - \hat{y})^2$

$$y = f(\mathbf{x}) + \epsilon \qquad \epsilon \sim N(0; \sigma^2)$$

true function   noise

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n) \rightarrow \hat{f}(\mathbf{x})$$ training algorithm

$$\bar{f}(\mathbf{x}) = \mathbb{E}\hat{f}(\mathbf{x})$$ expectation of the learned function

expectation is over datasets

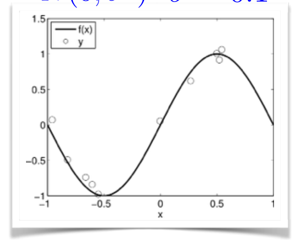
$$\mathbb{E}\left[\left(y - \hat{f}(\mathbf{x})\right)^2\right] = \mathbb{E}\left[\left(f(\mathbf{x}) - \bar{f}(\mathbf{x})\right)^2\right] + \mathbb{E}\left[\left(\hat{f}(\mathbf{x}) - \bar{f}(\mathbf{x})\right)^2\right] + \sigma^2$$

bias²   variance   noise

Example: curve fitting

$y = f(x) + \epsilon \quad f(x) = \sin(\pi x)$

$\epsilon = N(0, \sigma^2) \quad \sigma = 0.1$

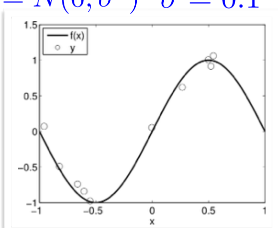


figures from https://theclevermachine.wordpress.com/tag/estimator-variance/

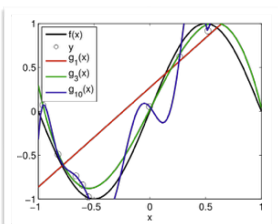
Example: curve fitting

$y = f(x) + \epsilon \quad f(x) = \sin(\pi x)$

$\epsilon = N(0, \sigma^2) \quad \sigma = 0.1$



$g_n(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$

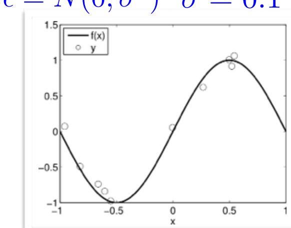


figures from https://theclevermachine.wordpress.com/tag/estimator-variance/

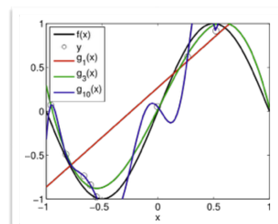
Example: curve fitting

$y = f(x) + \epsilon \quad f(x) = \sin(\pi x)$

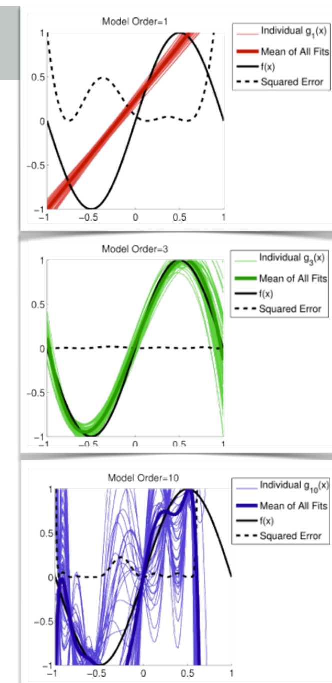
$\epsilon = N(0, \sigma^2) \quad \sigma = 0.1$



$g_n(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$

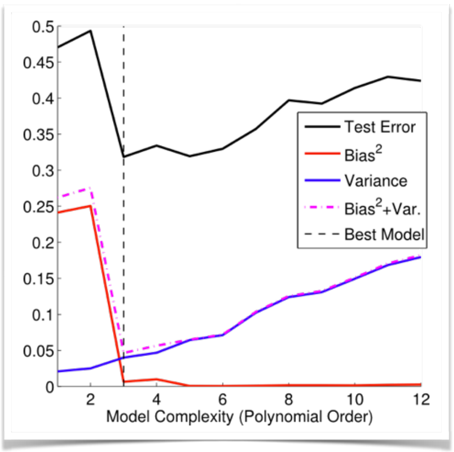


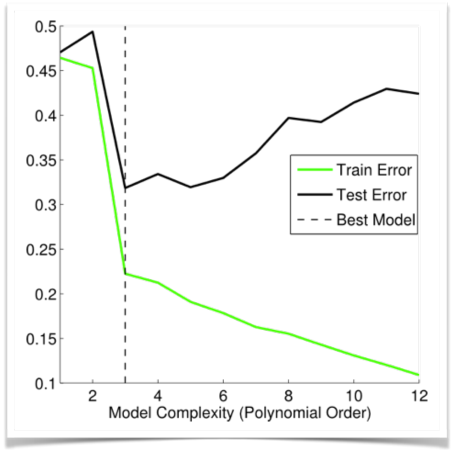
50 samples



figures from https://theclevermachine.wordpress.com/tag/estimator-variance/

Example: curve fitting





figures from https://theclevermachine.wordpress.com/tag/estimator-variance/

Bias-variance decomposition proof

$$y = f + \epsilon$$
$$\epsilon \sim N(0; \sigma^2)$$
$$\bar{f} = \mathbb{E}\hat{f}$$

$$\begin{aligned}
\mathbb{E}\left[\left(y - \hat{f})\right)^2\right] &= \mathbb{E}\left[\left(f + \epsilon - \hat{f}\right)^2\right] \\
&= \mathbb{E}\left[\left(f - \hat{f}\right)^2\right] + \sigma^2 \\
&= \mathbb{E}\left[\left(f - \bar{f} + \bar{f} - \hat{f}\right)^2\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\mathbb{E}\left[(f - \bar{f})\left(\bar{f} - \hat{f}\right)\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\mathbb{E}\left[\left(f\bar{f} - f\hat{f} - \bar{f}\bar{f} + \bar{f}\hat{f}\right)\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\left(f\bar{f} - f\bar{f} - \bar{f}\bar{f} + \bar{f}\bar{f}\right) + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + \sigma^2
\end{aligned}$$

Bias-variance decomposition proof

$$y = f + \epsilon$$
$$\epsilon \sim N(0; \sigma^2)$$
$$\bar{f} = \mathbb{E}\hat{f}$$

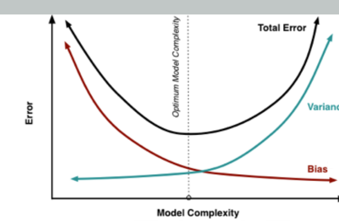
$$\begin{aligned}
\mathbb{E}\left[\left(y - \hat{f})\right)^2\right] &= \mathbb{E}\left[\left(f + \epsilon - \hat{f}\right)^2\right] \\
&= \mathbb{E}\left[\left(f - \hat{f}\right)^2\right] + \sigma^2 \\
&= \mathbb{E}\left[\left(f - \bar{f} + \bar{f} - \hat{f}\right)^2\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\mathbb{E}\left[(f - \bar{f})\left(\bar{f} - \hat{f}\right)\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\mathbb{E}\left[\left(f\bar{f} - f\hat{f} - \bar{f}\bar{f} + \bar{f}\hat{f}\right)\right] + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + 2\left(f\bar{f} - f\bar{f} - \bar{f}\bar{f} + \bar{f}\bar{f}\right) + \sigma^2 \\
&= \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + \sigma^2
\end{aligned}$$

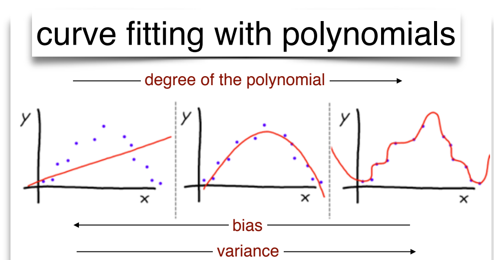
Similar decomposition can be obtained for the 0/1 loss

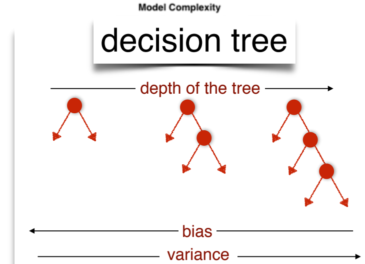
Bias-variance tradeoff for learners

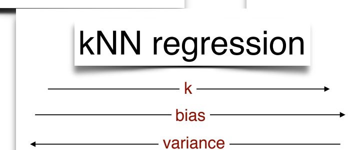
$$\mathbb{E}\left[\left(y - \hat{f}\right)^2\right] = \mathbb{E}\left[(f - \bar{f})^2\right] + \mathbb{E}\left[\left(\bar{f} - \hat{f}\right)^2\right] + \sigma^2$$

error = bias + variance + noise









Summary

- kNN classifiers
  - geometry, metric, decision boundaries
  - effect of k
  - practical issues
    - curse of dimensionality
    - speed: clustering using k-means, k-d trees
- Theory
  - Bayes optimality
  - kNN is nearly Bayes optimal as training dataset size goes to infinity
  - Bias-variance decomposition
  - Understanding overfitting and underfitting

Slides credit

- The fruit classification dataset is from Iain Murray at University of Edinburgh — http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/.
- The slides on texture synthesis are from Efros and Leung's ICCV 2009 presentation.
- Figures of the bias-variance tradeoff are from https://theclevermachine.wordpress.com/tag/estimator-variance/.