

CSC 480/580 Principles of Machine Learning

03 k -Nearest Neighbors

Jason Pacheco



Motivation

Example Given student course survey data, predict whether Alice likes Algorithms course

Idea Find a student “similar” to Alice & has taken Algorithm course before, say Jeremy

- If Jeremy likes Algorithms, then Alice is also likely to have the same preference.
- Or even better, find *several* similar students

- Prediction = mapping inputs to outputs
- Inputs = *features* that can be viewed as points in some space (possibly high-dimensional)
- “Similarity” = “distance” in feature space
- Suggests a **geometric** view of data

k -nearest neighbor: main concept

- Train set: $S = \{ (x_1, y_1), \dots, (x_m, y_m) \}$
- **Idea**: given a new, unseen data point x , its label should resemble the labels of nearby points
- Learned function
 - Input: $x \in \mathbb{R}^d$
 - Find the k nearest points to x from S ; call it $N(x)$ E.g., Euclidean distance
 - Output: the majority vote of $\{y_i: i \in N(x)\}$
 - For regression, compute the average label.

Measuring Nearest Neighbors

- Oftentimes convenient to work with feature $x \in \mathbb{R}^d$

- Distances in \mathbb{R}^d : notation $x(f): x = (x(1), \dots, x(d))$

- Euclidean distance $d_2(x, x') = \sqrt{\sum_{f=1}^d (x(f) - x'(f))^2}$

- Manhattan distance $d_1(x, x') = \sum_{f=1}^d |x(f) - x'(f)|$

- If we shift a feature, would the distance change?

- What about scaling a feature?

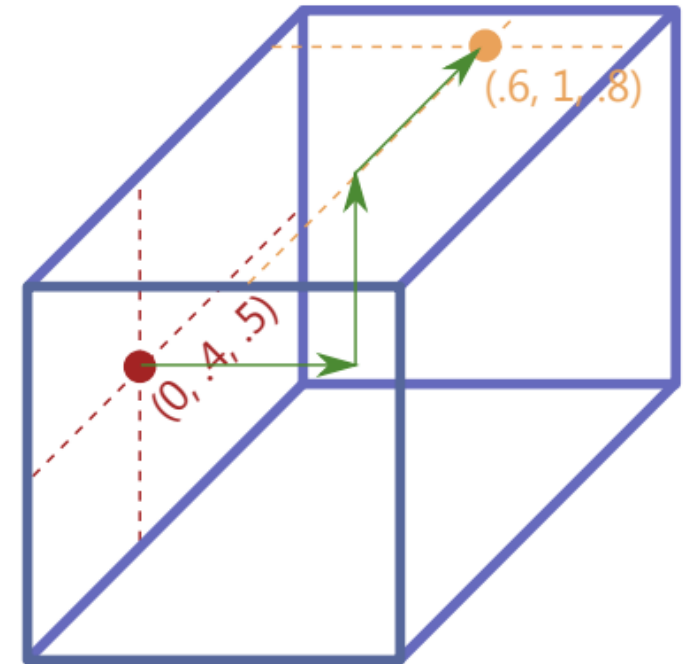
- How to extract features as **real values**?

- Boolean features: $\{Y, N\} \rightarrow \{1, 0\}$

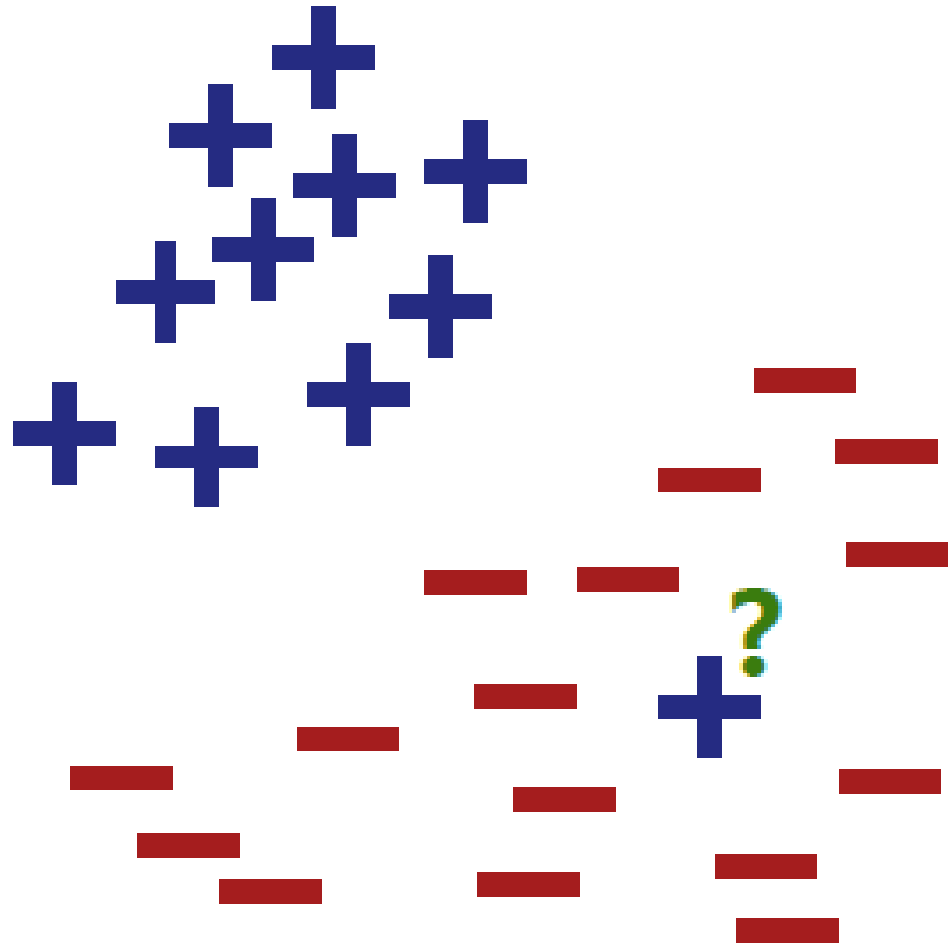
- Categorical features: $\{\text{Red, Blue, Green, Black}\}$

- Convert to $\{1, 2, 3, 4\}$?

- Better one-hot encoding: $(1, 0, 0, 0), \dots, (0, 0, 0, 1)$ (IsRed?/isGreen?/isBlue?/IsBlack?)



Nearest Neighbor Classification



Query point ? Will be classified as +
but should be -

Inductive Bias Query points belong to
same class as closest example seen in
training data

Question How can we reduce
inductive bias?

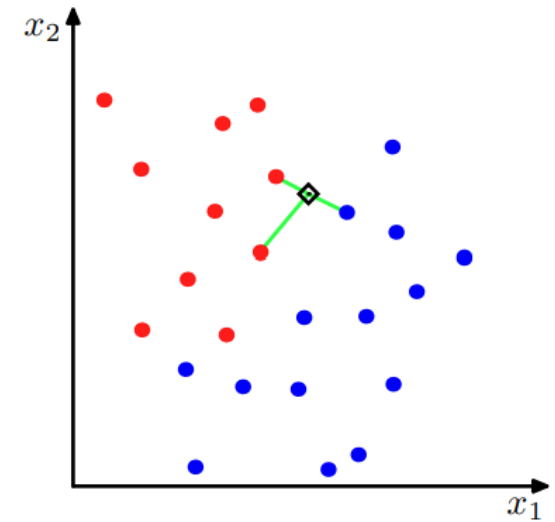
k -nearest neighbors (k -NN): main concept

Training set: $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$

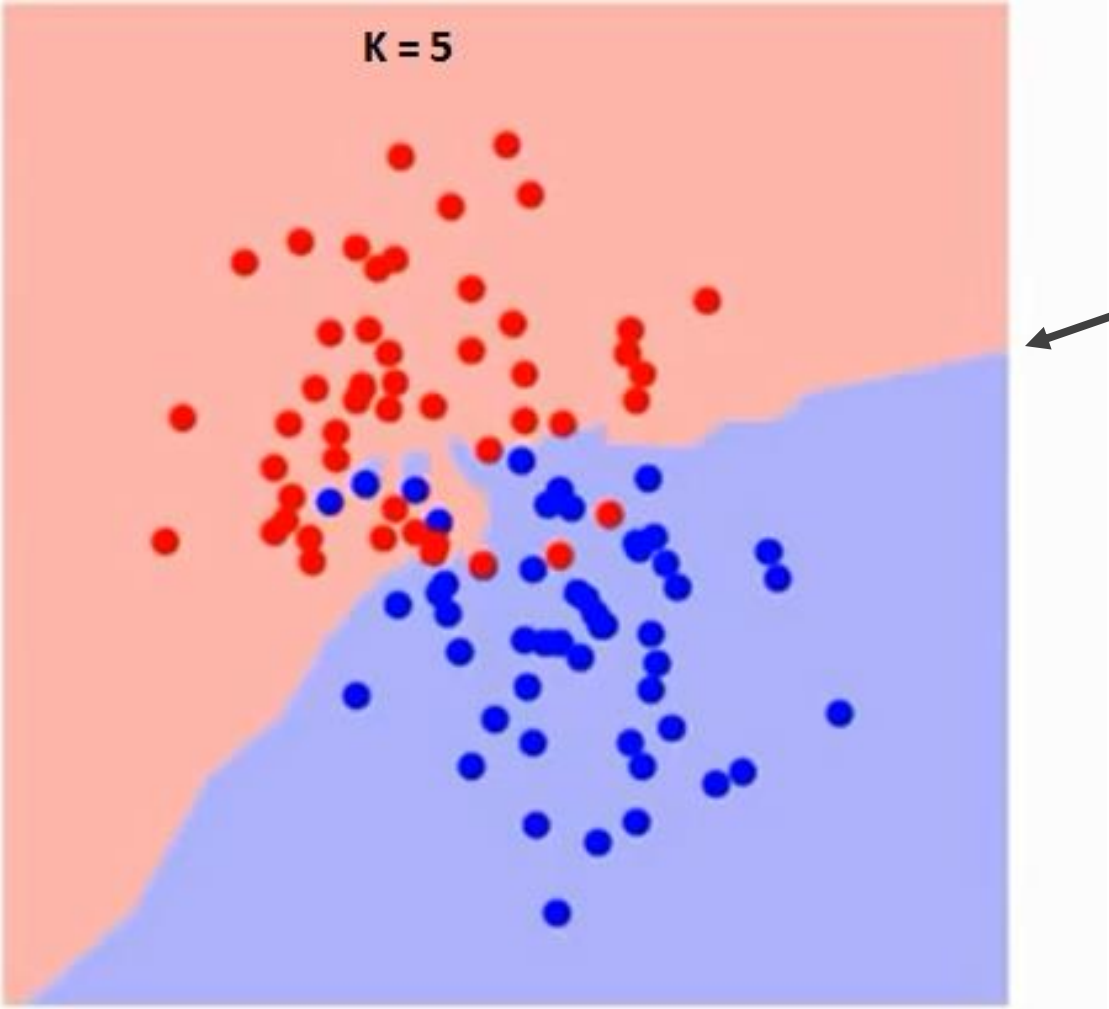
Inductive bias: given test example x , its label should resemble the labels of **nearby points**

Function

- input: x
- find the k nearest points to x from S ; call their indices $N(x)$
- output: the majority vote of $\{y_i : i \in N(x)\}$
 - For regression, the average.



k-NN classification example



k -NN classification: pseudocode

- Training is trivial: store the training set
- Test:

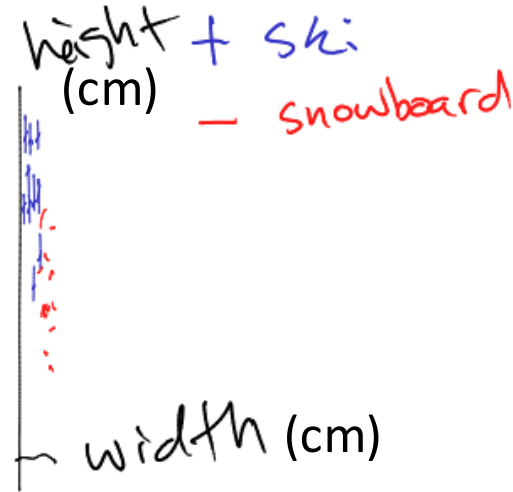
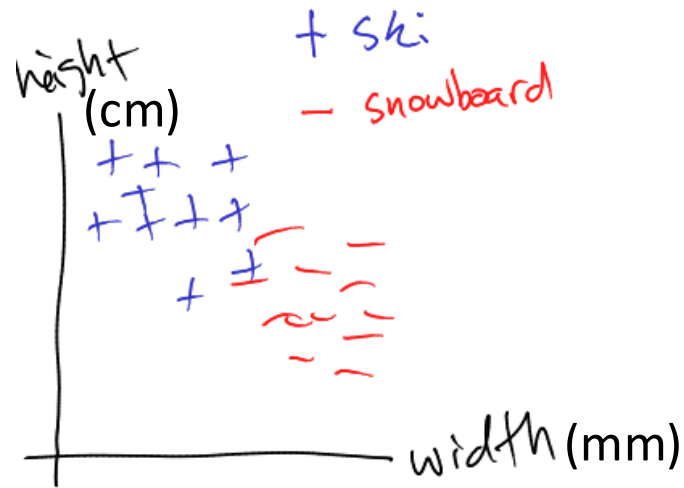
Algorithm 3 KNN-PREDICT(\mathbf{D}, K, \hat{x})

list	→	1: $S \leftarrow []$	
		2: for $n = 1$ to N do	
append to list	→	3: $S \leftarrow S \oplus \langle d(x_n, \hat{x}), n \rangle$	// store distance to training example n
		4: end for	
sort in first coordinate	→	5: $S \leftarrow \text{SORT}(S)$	// put lowest-distance objects first
		6: $\hat{y} \leftarrow 0$	
		7: for $k = 1$ to K do	
		8: $\langle \text{dist}, n \rangle \leftarrow S_k$	// n this is the k th closest data point
		9: $\hat{y} \leftarrow \hat{y} + y_n$	// vote according to the label for the n th training point
		10: end for	
Majority vote of $\{y_i: i \in N(x)\}$	→	11: return SIGN(\hat{y})	// return +1 if $\hat{y} > 0$ and -1 if $\hat{y} < 0$

- Time complexity (assuming distance calculation takes $O(d)$ time)
 - $O(m d + m \log m + k) = O(m(d + \log m))$
- Faster nearest neighbor search: k-d trees, locality sensitive hashing

Issue 1: Feature Scaling

- Features having different scale can be problematic.
- Ex: ski vs. snowboard classification



- Solution: feature standardization

Make sure features are scaled fairly

- Features having different scale can be problematic. (e.g., weights in lbs vs weight in grams)
- [Definition] **Standardization**

- For each feature f , compute $\mu_f = \frac{1}{m} \sum_{i=1}^m x_i(f)$, $\sigma_f = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i(f) - \mu_f)^2}$

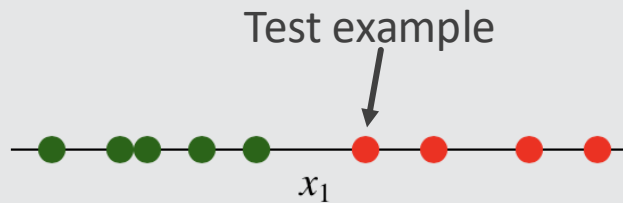
- Then, transform the data by $\forall f \in \{1, \dots, d\}, \forall i \in \{1, \dots, m\}, x_i(f) \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

after transformation, each feature has mean 0 and variance 1

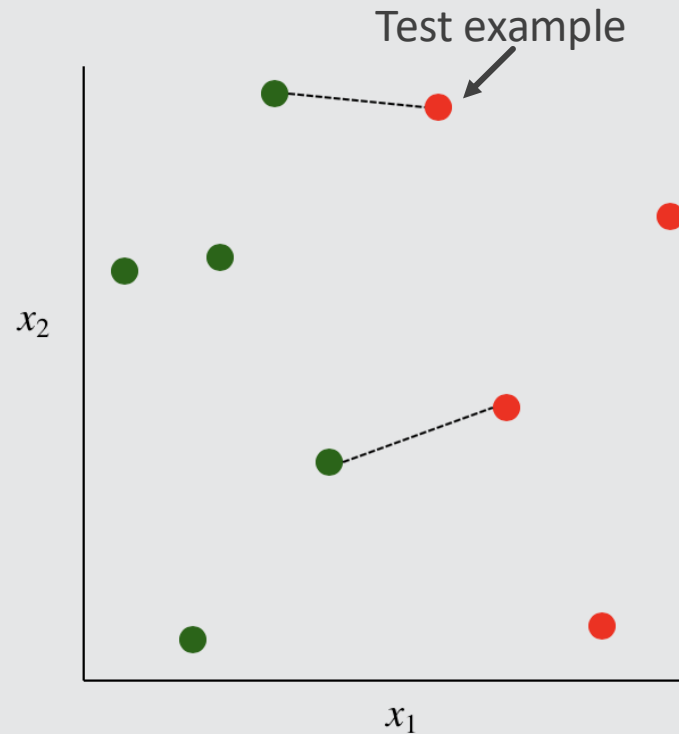
- Be sure to keep the “standardize” function and **apply it to the test points.**
 - Save $\{(\mu_f, \sigma_f)\}_{f=1}^d$
 - For test point x^* , apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}$, $\forall f$

Issue 2: Irrelevant Features

here's a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly



consider the effect of an irrelevant feature x_2 on distances and nearest neighbors



- Recall: how did we deal with these in decision trees?
- Solution: feature selection (later in the course)

Issue 3: test time complexity

- How a k-NN function work (say d -dimension):

- Compute distance to m points

$$O(dm)$$

- Sort distances

$$O(m \log m)$$

- Pick k smallest.

$$O(k)$$

- Overall $O(m(d + \log m))$

- Issue: test time complexity scales linearly with m !!

imagine an image classifier trained on 10M images to be used in smart phones.

- Solutions

- k-d tree: Exact search

for large d very likely to hit the worst case

- Best case: $O(\log(m))$ Worst case: $O(m)$

- Locality-sensitive hashing: approximate search, $O(m^\rho)$ with $\rho \in (0,1)$

sublinear time complexity!

Comparison (feature $x \in \mathbb{R}^d$)

- Interpretability
- Sensitivity to irrelevant features
- training time
- test time per example

Decision Tree

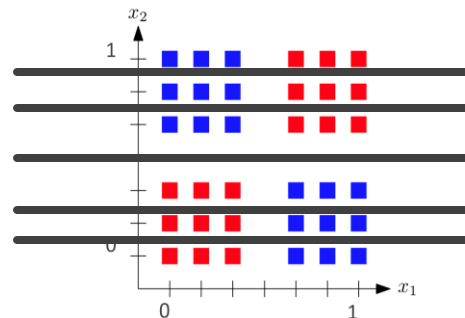
High

Low

$$O(\#nodes \cdot d \cdot (m + m \log m))$$

$$\leq \tilde{O}(d m^2) \text{ (when no two points have the same feature)}$$

$$O(\text{depth})$$



k -NN

Medium (example-based)

High

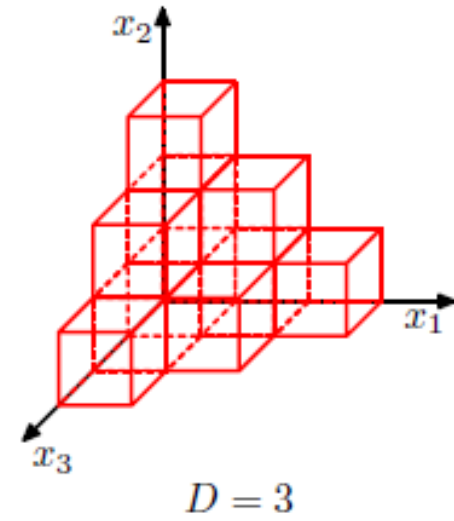
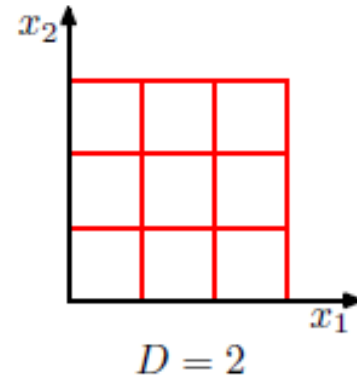
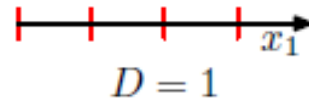
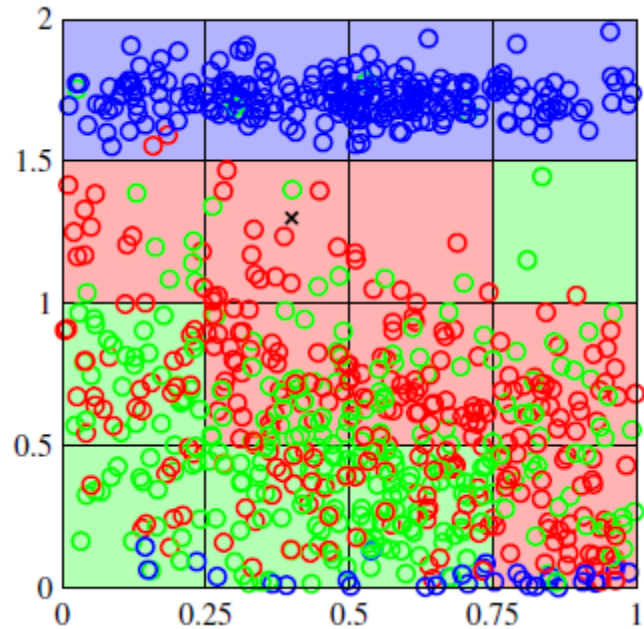
0

$$O(m(d + \log m))$$

*Can reduce this with
K-d trees or locality
sensitive hashing*

Curse of Dimensionality - Computation

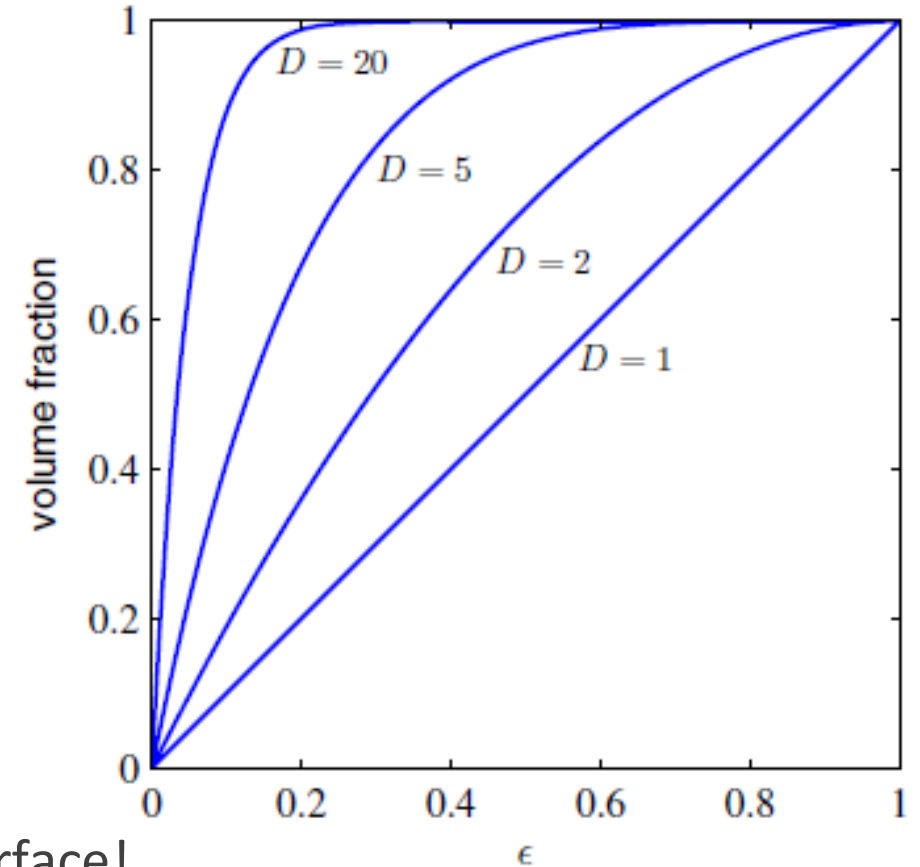
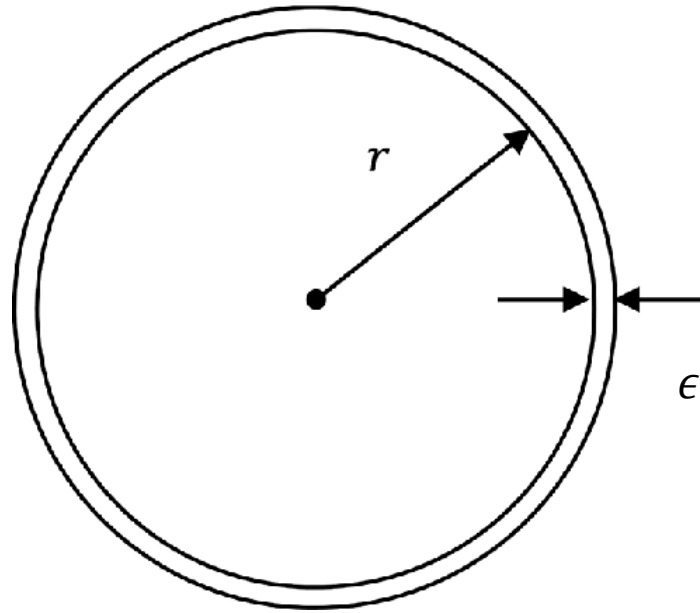
Divide space into regular intervals to avoid computing distances for each data



Number of required cells grows exponentially in dimension!

Curse of Dimensionality – Distance Weirdness

- Consider D -dimensional hypersphere of radius $r=1$
- What is the fraction of volume within shell of width ϵ ?



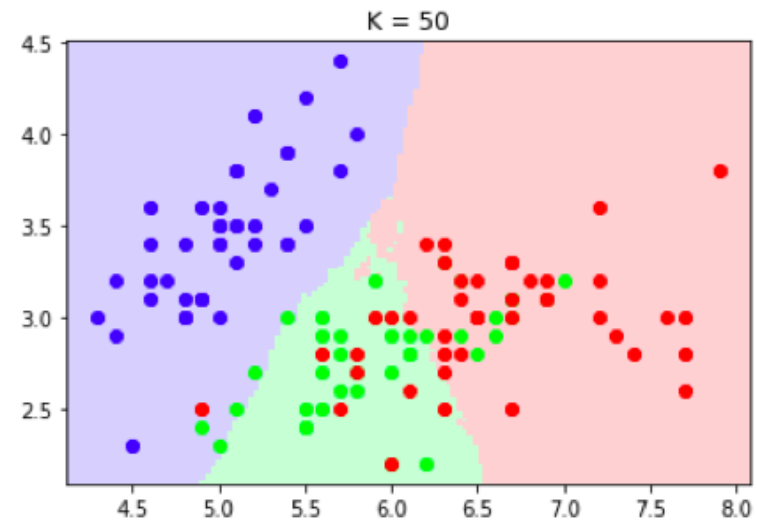
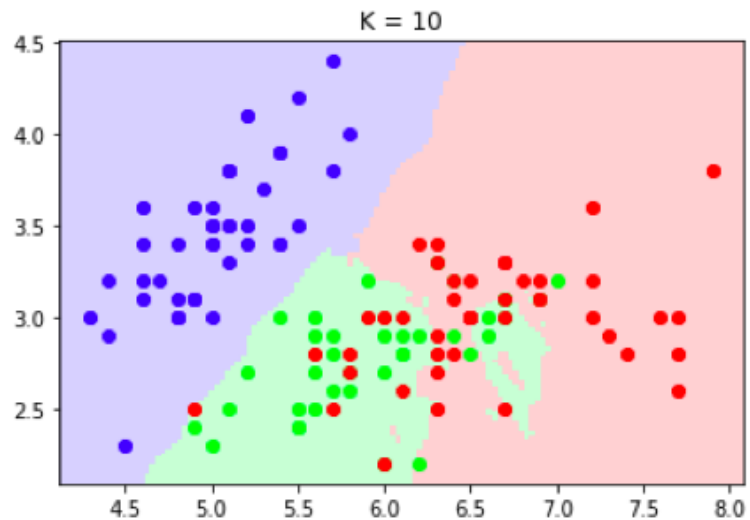
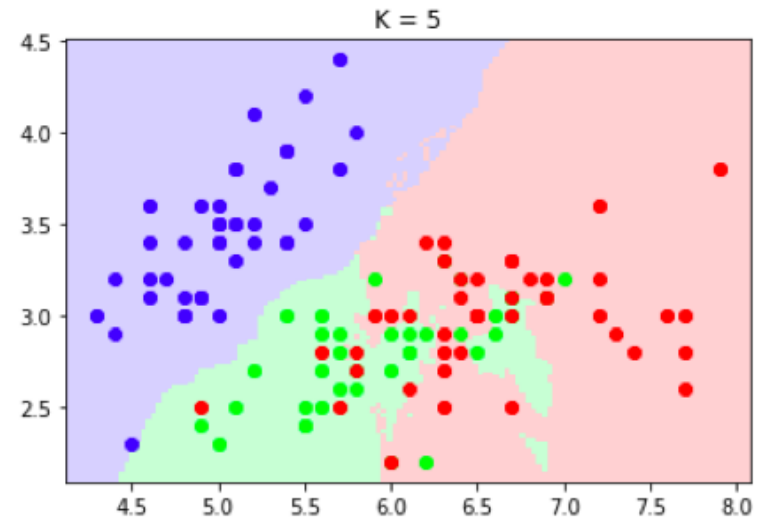
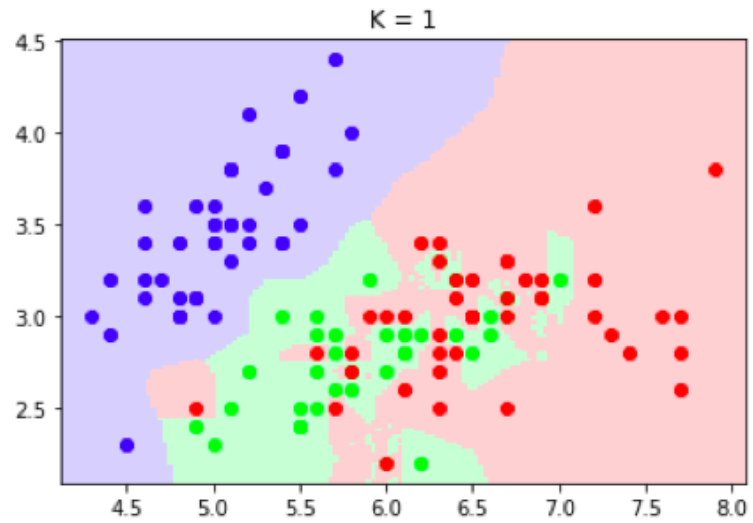
- Total volume of hypersphere concentrates onto shell at the surface!
- Distances go to zero!

Intuition about lower dimensions doesn't extend to high dimensions

Issue 3: choosing k

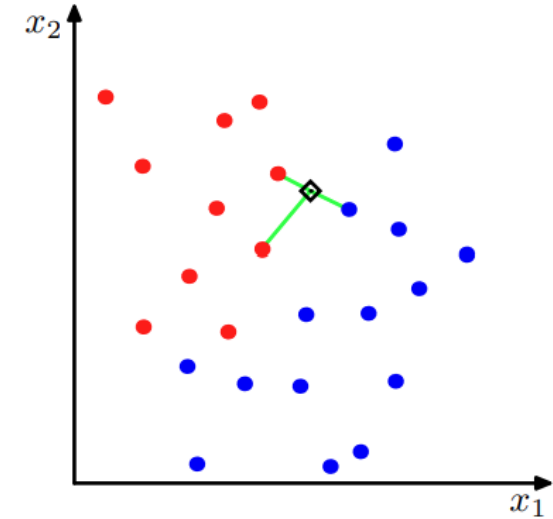
- k is not learned from data. Must be selected by practitioners.
- Q: If we set $k = m$, then which classification rule does it look like?
- Q: If we set $k = 1$, what would be the train set error (assume there is no repeated train data point)?

Hyperparameter tuning in k -NN



Hyperparameter tuning in k -NN

- Hyperparameter: k
- $k = 1$:
 - Training error = 0, overfitting
- $k = N$:
 - Output a constant (majority class) prediction, underfitting
- Can use hold-out validation to choose k



k-NN Summary

- Given: labeled data S
- Training
 - Compute and save $\{(\mu_f, \sigma_f)\}_{f=1}^d$
 - Compute and save standardization of S
- Test
 - Given x^* , apply standardization $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$
 - Compute k nearest neighbors $N(x^*)$
 - Make prediction
 - For classification: Predict by majority vote label in $N(x^*)$
 - For regression: Predict by the average label in $N(x^*)$

Variations

Recall the majority vote rule: $\hat{y} = \arg \max_{y \in \{1, \dots, C\}} \sum_{i \in \mathcal{N}(x)} 1\{y_i = y\}$




Q: Blue dot is the test point. If $k=3$, which label would it predict?

Q: Which label do you think we should predict?

Weighted version

• $\hat{y} = \arg \max_{y \in \{1, \dots, C\}} \sum_{i \in \mathcal{N}(x)} w_i 1\{y^{(i)} = y\}$

 weights that sum to 1

$$w_i \propto \exp(-\beta \cdot d(x, x^{(i)})), \beta > 0$$

~~$$w_i \propto \frac{1}{d(x, x^{(i)})^\beta}$$~~

$$w_i \propto \frac{1}{1 + d(x, x^{(i)})^\beta}$$

Q: What would be the downside of using weighted version?

tuning β is cumbersome!

Confidence

Confidence

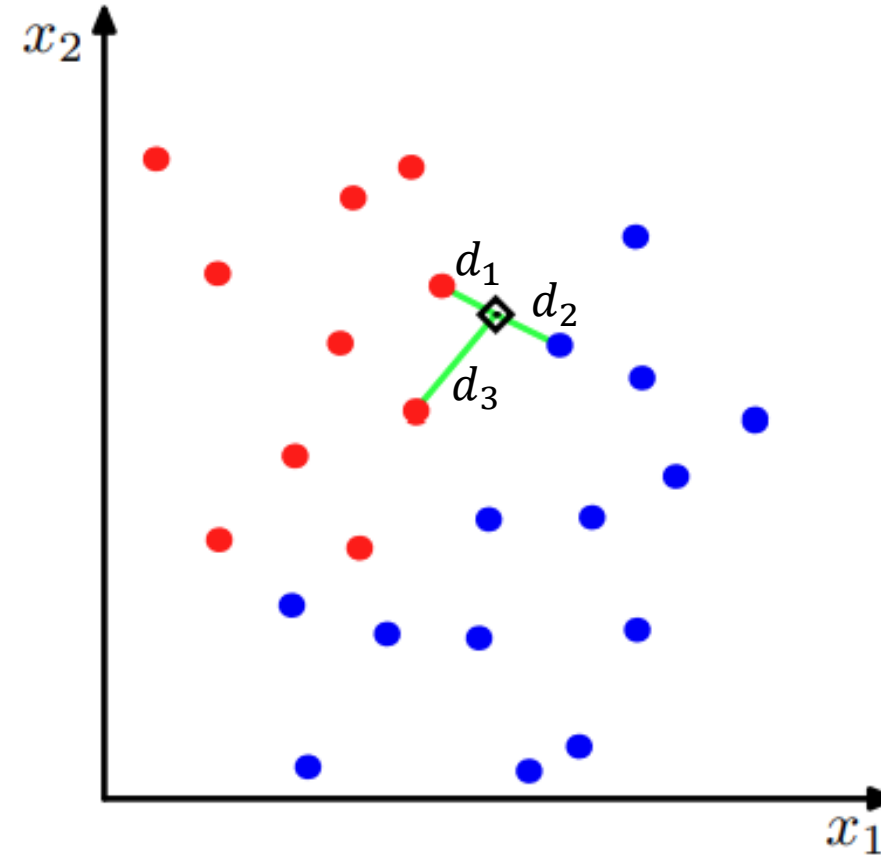
- $P(Y = y|X = x) \propto \sum_{i \in \mathcal{N}(x)} \mathbf{1}\{y^{(i)} = y\}$
- $P(Y = y|X = x) \propto \sum_{i \in \mathcal{N}(x)} w_i \mathbf{1}\{y^{(i)} = y\}$ // weighted version

Same thing applies to decision tree.

- At each leaf node, we need to record the fraction of labels, not just the majority vote label.

k-NN Regression

Predict real-valued outputs as
inverse-distance-weighted
average of nearby points



Known as *Shepard's
interpolation*

$$y^{test} = \left(\sum_j \frac{1}{d_j} \right)^{-1} \sum_i \frac{1}{d_i} y_i$$

└──────────┬──────────┘
Sum over k neighbors

Model parameters

Parameter: the variables that describe the model

Decision tree's parameter

- The entire tree structure
- What questions are being asked at each internal node
- Output (=prediction) from each leaf node

Q: What are the parameters of kNN?

It's the train set!

***Hyper-parameter**: parameters that are not learned by the algorithm. E.g., k