Lecture 2: K-nearest neighbours CS480/680 Intro to Machine Learning

2023-1-12

Pascal Poupart
David R. Cheriton School of Computer Science

UNIVERSITY OF WATERLOO

Inductive Learning (recap)

- Induction
  - Given a training set of examples of the form $(x, f(x))$
    - x is the input, $f(x)$ is the output
  - Return a function h that approximates f
    - h is called the hypothesis

Supervised Learning

- Two types of problems
  1. **Classification**
  2. **Regression**

- NB: The nature (categorical or continuous) of the domain (input space) of f does not matter

Classification Example

- Problem: Will you enjoy an outdoor sport based on the weather?
- Training set:

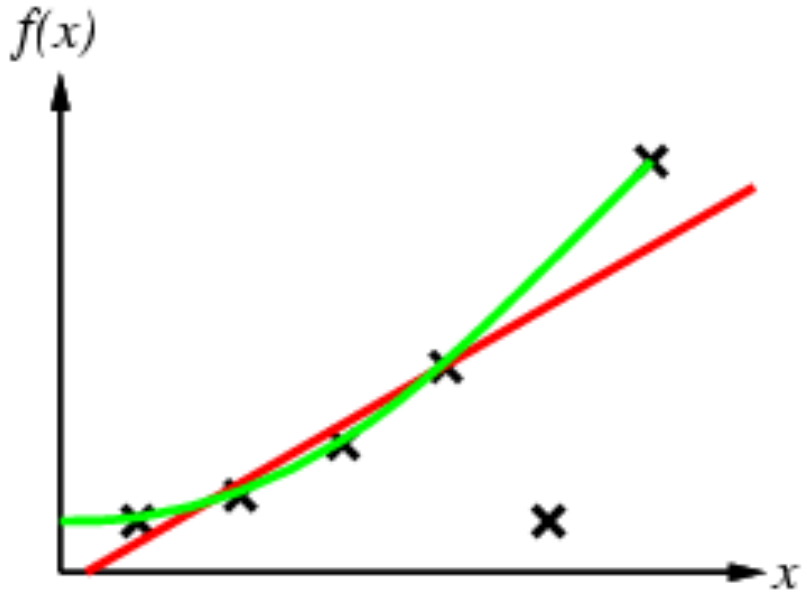
Sky	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Normal	Strong	Warm	Same	yes
Sunny	High	Strong	Warm	Same	yes
Sunny	High	Strong	Warm	Change	no
Sunny	High	Strong	Cool	Change	yes

x $f(x)$

- Possible Hypotheses:
  - $h_1: S = sunny \rightarrow enjoySport = yes$
  - $h_2: Wa = cool$ or $F = same \rightarrow enjoySport = yes$

Regression Example

- Find function h that fits f at instances x



More Examples

Problem	Domain	Range	Classification / Regression
Spam Detection			
Stock price prediction			
Speech recognition			
Digit recognition			
Housing valuation			
Weather prediction			

Hypothesis Space

- Hypothesis space H
  - Set of all hypotheses h that the learner may consider
  - Learning is a search through hypothesis space
- Objective: find h that minimizes
  - Misclassification (or more generally some error function)

  with respect to the training examples

- But what about unseen examples?

Generalization

- A good hypothesis will generalize well
  - i.e., predict unseen examples correctly

- Usually …
  - Any hypothesis h found to approximate the target function f well over a **sufficiently large set of training examples** will also approximate the target function well over any unobserved examples

Inductive Learning

- Goal: find an h that agrees with f on training set
  - h is **consistent** if it agrees with f on all examples

- Finding a consistent hypothesis is not always possible
  - Insufficient hypothesis space:
    - E.g., it is not possible to learn exactly $f(x) = ax + b + xsin(x)$ when H = space of polynomials of finite degree
  - Noisy data
    - E.g., in weather prediction, identical conditions may lead to rainy and sunny days

Inductive Learning

- A learning problem is **realizable** if the hypothesis space contains the true function otherwise it is **unrealizable**.
  - Difficult to determine whether a learning problem is realizable since the true function is not known
- It is possible to use a very large hypothesis space
  - For example: H = class of all Turing machines
- But there is a **tradeoff** between **expressiveness** of a hypothesis class and the **complexity** of finding a good hypothesis

Nearest Neighbour Classification

- Classification function: $h(x) = y_{x^*}$
  where y_{x^*} is the label associated with the nearest neighbour

$$x^* = argmin_{x'}\, d(x, x')$$

- Distance measures: $d(x, x')$

  L_1: $d(x, x') = \sum_j^M |x_j - x'_j|$

  L_2: $d(x, x') = \left(\sum_j^M |x_j - x'_j|^2\right)^{1/2}$

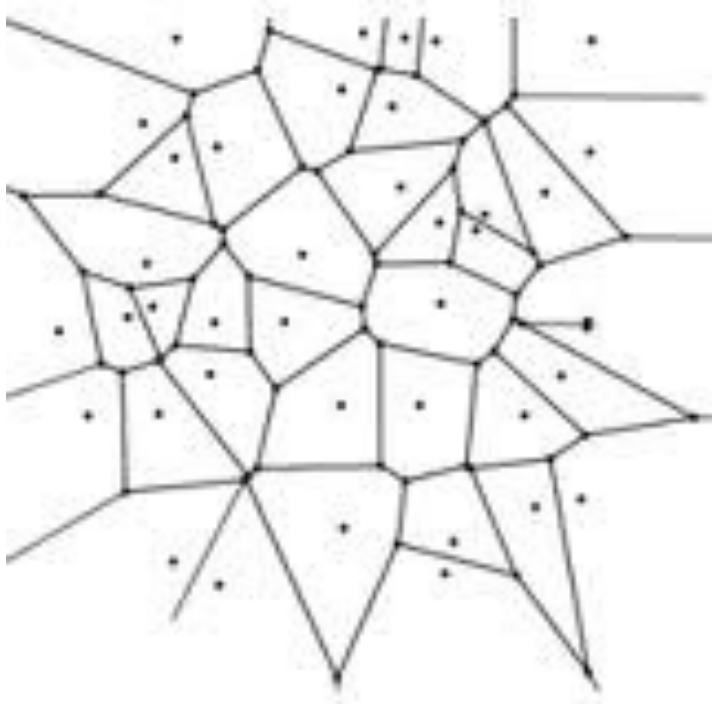
  ...

  L_p: $d(x, x') = \left(\sum_j^M |x_j - x'_j|^p\right)^{1/p}$

  Weighted dimensions: $d(x, x') = \left(\sum_j^M c_j |x_j - x'_j|^p\right)^{1/p}$

Voronoi Diagram

- Partition implied by nearest neighbor fn h
  - Assuming Euclidean distance



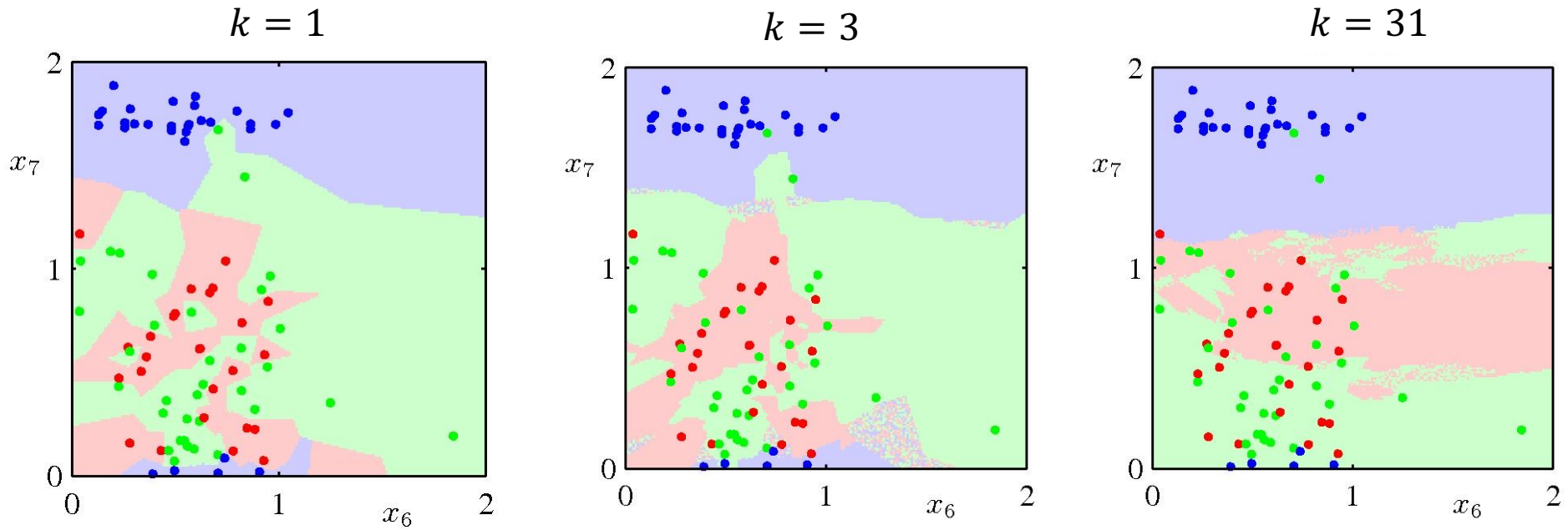
K-Nearest Neighbour

- Nearest neighbour often instable (noise)

- Idea: assign most frequent label among k-nearest neighbours
  - Let $knn(x)$ be the k-nearest neighbours of x according to distance d
  - Label: $y_x \leftarrow mode(\{y_{x'} | x' \in knn(x)\})$

Effect of K

- K controls the degree of smoothing.
- Which partition do you prefer? Why?



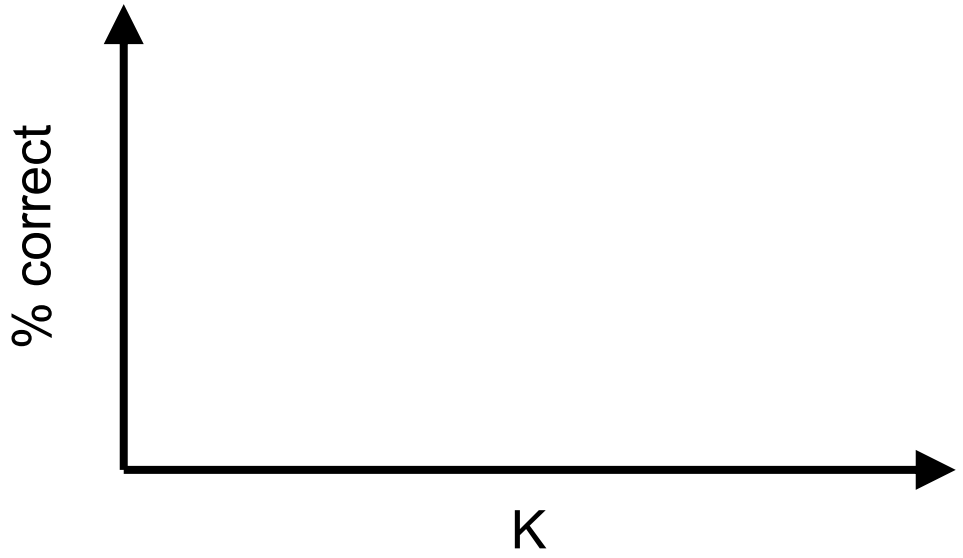
Performance of a learning algorithm

- A learning algorithm is good if it produces a hypothesis that does a good job of predicting classifications of unseen examples
- Verify performance with a **test set**

1. Collect a large set of examples
2. Divide into 2 disjoint sets: training set and test set
3. Learn hypothesis *h* with training set
4. Measure percentage of correctly classified examples by *h* in the test set

The effect of K

- Best K depends on
  - Problem
  - Amount of training data



Underfitting

- **Definition:** underfitting occurs when an algorithm finds a hypothesis h with training accuracy that is lower than the future accuracy of some other hypothesis h'

- Amount of underfitting of h:

$$max\ \{0, \max_{h'} futureAccuracy(h') - trainAccuracy(h)\}$$
$$\approx max\ \{0, \max_{h'} testAccuracy(h') - trainAccuracy(h)\}$$

- Common cause:
  - Classifier is not expressive enough

Overfitting

- **Definition:** overfitting occurs when an algorithm finds a hypothesis h with higher training accuracy than its future accuracy.

- Amount of overfitting of h:

$$\max\{0, trainAccuracy(h) - futureAccuracy(h)\}$$
$$\approx \max\{0, trainAccuracy(h) - testAccuracy(h)\}$$

- Common causes:
  - Classifier is too expressive
  - Noisy data
  - Lack of data

Choosing K

- How should we choose K?
  - Ideally: select K with highest future accuracy
  - Alternative: select K with highest test accuracy

- **Problem:** since we are choosing K based on the test set, the test set effectively becomes part of the training set when optimizing K. Hence, we cannot trust anymore the test set accuracy to be representative of future accuracy.

- Solution: split data into **training, validation and test sets**
  - **Training set:** compute nearest neighbour
  - **Validation set:** optimize hyperparameters such as K
  - **Test set:** measure performance

Choosing K based on Validation Set

Let k be the number of neighbours
For $k = 1$ to max # of neighbours
    $accuracy_k \leftarrow eval(k, trainingData, validationData)$
$k^* \leftarrow argmax_k\ accuracy_k$
$accuracy \leftarrow eval(k^*, trainingData \cup validationData, testData)$
Return $k^*, accuracy$

$eval(k, trainingData, dataset)$
    $error \leftarrow 0$
    For each $(x, y) \in dataset$
        Find $\{x_1, x_2, \dots, x_k | (x_i, y_i) \in trainingData\}$ closest to x
        $y^* \leftarrow mode(\{y_1, y_2, \dots, y_k\})$
        if $y \neq y^*$ then $error_k \leftarrow error_k + 1$
    $accuracy \leftarrow \frac{|dataset| - error}{|dataset|}$
    return $accuracy$

Robust validation

- How can we ensure that validation accuracy is representative of future accuracy?
  - Validation accuracy becomes more reliable as we increase the size of the validation set
  - However, this reduces the amount of data left for training

- Popular solution: **cross-validation**

Cross-Validation

- Repeatedly split training data in two parts, one for training and one for validation. Report the average validation accuracy.
- ***k*-fold cross validation**: split training data in k equal size subsets. Run k experiments, each time validating on one subset and training on the remaining subsets. Compute the average validation accuracy of the k experiments.
- Picture:

Selecting the Number of Neighbours by Cross-Validation

Let k be the number of neighbours

Let k' be the number of $trainingData$ splits

For $k = 1$ to max # of neighbours

    For $i = 1$ to k' do (where i indexes $trainingData$ splits)

        $accuracy_{ki} \leftarrow eval(k, trainingData_{1..i-1,i+1..k'}, validationData_i)$

    $accuracy_k \leftarrow average(\{accuracy_{ki}\}_{\forall i})$

$k^* \leftarrow argmax_k \, accuracy_k$

$accuracy \leftarrow eval(k^*, trainingData_{1..k'}, testData)$

Return $k^*, accuracy$

Selecting the Hyperparameters by Cross-Validation

Let ϕ be a hyperparameter
Let k' be the number of $trainingData$ splits
Let h be a hypothesis obtained by training
For $\phi \in \{hyperparameter\ values\}$
    For $i = 1$ to k' do (where i indexes $trainingData$ splits)
        $h_{\phi i} \leftarrow train(\phi, trainingData_{1..i-1,i+1..k'})$
        $accuracy_{\phi i} \leftarrow test(h_{\phi i}, trainingData_i)$
    $accuracy_\phi \leftarrow average(\{accuracy_{\phi i}\}_{\forall i})$
$\phi^* \leftarrow argmax_\phi\ accuracy_\phi$
$h \leftarrow train(\phi^*, trainingData_{1..k'})$
$accuracy \leftarrow test(h, testData)$
Return $\phi^*, h, accuracy$

Weighted K-Nearest Neighbour

- We can often improve K-nearest neighbours by weighting each neighbour based on some distance measure

$$w(x, x') \propto \frac{1}{distance(x, x')}$$

- Label $y_x \leftarrow argmax_y \sum_{\{x' | x' \in knn(x) \wedge y = y_{x'}\}} w(x, x')$

  where $knn(x)$ is the set of K nearest neighbours of x

K-Nearest Neighbour Regression

- We can also use KNN for regression
- Let y_x be a real value instead of a categorical label

- K-nearest neighbour regression:

$$y_x \leftarrow average(\{y_{x'} | x' \in knn(x)\})$$

- Weighted K-nearest neighbour regression:

$$y_x \leftarrow \frac{\sum_{x' \in knn(x)} w(x, x') y_{x'}}{\sum_{x' \in knn(x)} w(x, x')}$$