COMPSCI 589
Lecture 2: KNN and Decision Trees

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

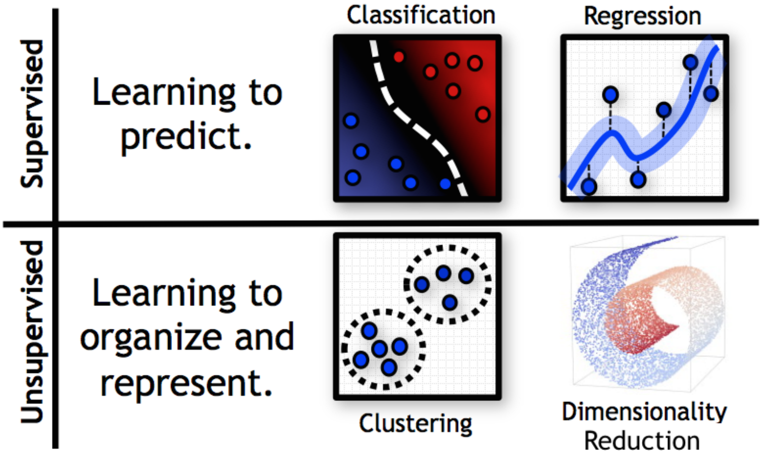
Outline

Views on Machine Learning

Mitchell (1997): “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Substitute “training data D” for “experience E.”

Machine Learning Tasks



Machine Learning Approaches

- **Non-Parametric:** Learning is accomplished by storing the training data (memorization).
- **Parametric:** Learning is accomplished by using an algorithm to adapt the parameters in a mathematical or statistical model given training data. For exmaple:

Outline

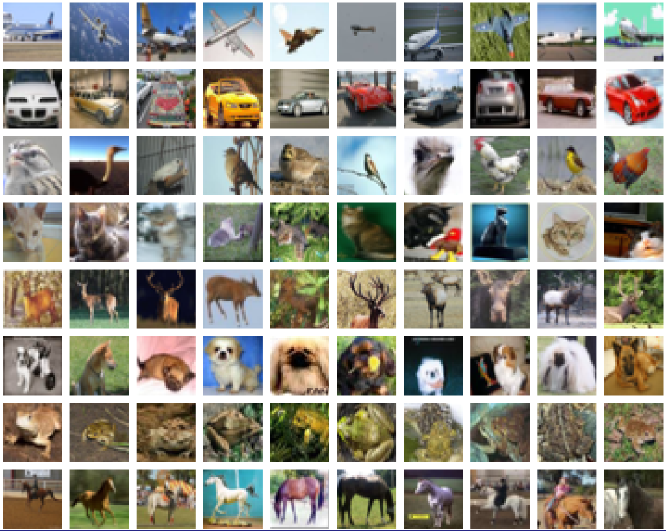
The Classification Task

Definition: The Classification Task

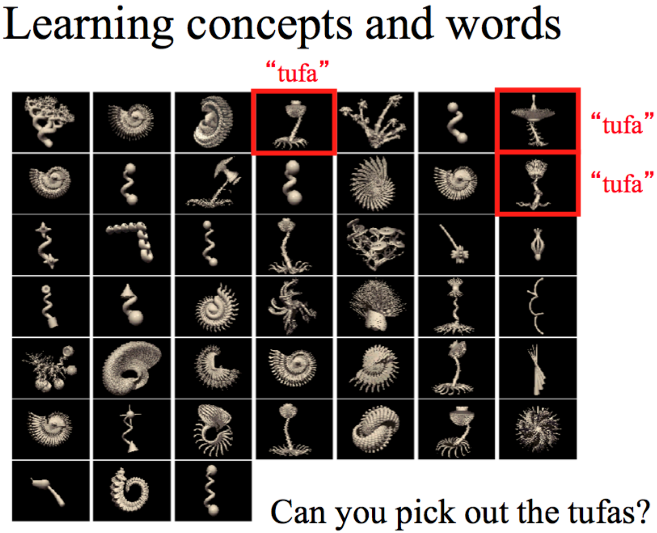
Given a feature vector $\mathbf{x} \in \mathbb{R}^D$ that describes an object that belongs to one of C classes from the set $\mathcal{Y}$, predict which class the object belongs to.

Example: Digits

Example: Natural Images



Example: Synthetic Images



The Classifier Learning Problem

Definition: Classifier Learning

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ is a feature vector and $y_i \in \mathcal{Y}$ is a class label, learn a function $f : \mathbb{R}^D \rightarrow \mathcal{Y}$ that accurately predicts the class label y for any feature vector $\mathbf{x}$.

Classification Error and Accuracy

Definition: Classification Error Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ and a function $f : \mathbb{R}^D \to \mathcal{Y}$, the classification error rate of f on $\mathcal{D}$ is:

$$Err(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i \neq f(\mathbf{x}_i))$$

Classification Error and Accuracy

Definition: Classification Error Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ and a function $f : \mathbb{R}^D \rightarrow \mathcal{Y}$, the classification error rate of f on $\mathcal{D}$ is:

$$Err(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i \neq f(\mathbf{x}_i))$$

Definition: Classification Accuracy Rate

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ and a function $f : \mathbb{R}^D \rightarrow \mathcal{Y}$, the classification accuracy rate of f on $\mathcal{D}$ is:

$$Acc(f, \mathcal{D}) = \frac{1}{N} \sum^{N} \mathbb{I}(y_i = f(\mathbf{x}_i))$$

Outline

K Nearest Neighbors Classification

The KNN classifier is a non-parametric classifier that simply stores the training data $\mathcal{D}$ and classifies each new instance $\mathbf{x}$ using a majority vote over its' set of K nearest neighbors $\mathcal{N}_K(\mathbf{x})$ computed using any distance function $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$.

K Nearest Neighbors Classification

The KNN classifier is a non-parametric classifier that simply stores the training data $\mathcal{D}$ and classifies each new instance $\mathbf{x}$ using a majority vote over its' set of K nearest neighbors $\mathcal{N}_K(\mathbf{x})$ computed using any distance function $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$.

KNN Classification Function

$$f_{KNN}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_K(\mathbf{x})} \mathbb{I}[y_i = y]$$

K Nearest Neighbors Classification

The KNN classifier is a non-parametric classifier that simply stores the training data $\mathcal{D}$ and classifies each new instance $\mathbf{x}$ using a majority vote over its' set of K nearest neighbors $\mathcal{N}_K(\mathbf{x})$ computed using any distance function $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$.

KNN Classification Function

$$f_{KNN}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_K(\mathbf{x})} \mathbb{I}[y_i = y]$$

Use of KNN requires choosing the distance function d and the number of neighbors K.

Distance and Similarity

- In general, KNN can work with any distance function d satisfying non-negativity $d(\mathbf{x}, \mathbf{x}') \geq 0$ and identity of indiscernibles $d(\mathbf{x}, \mathbf{x}) = 0$.

Distance and Similarity

- In general, KNN can work with any distance function d satisfying non-negativity $d(\mathbf{x}, \mathbf{x}') \geq 0$ and identity of indiscernibles $d(\mathbf{x}, \mathbf{x}) = 0$.
- Alternatively, KNN can work with any similarity function s satisfying non-negativity $s(\mathbf{x}, \mathbf{y}) \geq 0$ that attains it's maximum on indiscernibles $s(\mathbf{x}, \mathbf{x}) = \max_{\mathbf{x}'} s(\mathbf{x}, \mathbf{x}')$.

Distance and Similarity

- In general, KNN can work with any distance function d satisfying non-negativity $d(\mathbf{x}, \mathbf{x}') \geq 0$ and identity of indiscernibles $d(\mathbf{x}, \mathbf{x}) = 0$.
- Alternatively, KNN can work with any similarity function s satisfying non-negativity $s(\mathbf{x}, \mathbf{y}) \geq 0$ that attains it's maximum on indiscernibles $s(\mathbf{x}, \mathbf{x}) = \max_{\mathbf{x}'} s(\mathbf{x}, \mathbf{x}')$.
- However, the more structure the distance or similarity function has (symmetry, triangle inequality), the more structure you can exploit when designing algorithms.

Distance Metrics

Definition: Minkowski Distance (ℓ_p norms)

Given two data vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$, the Minkowski Distance with parameter p (the ℓ_p norm) is a proper metric defined as follows:

$$d_p(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_p$$
$$= \left(\sum_{i=1}^{D} |x_d - x'_d|^p \right)^{1/p}$$

Distance Metrics

Definition: Minkowski Distance (ℓ_p norms)

Given two data vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$, the Minkowski Distance with parameter p (the ℓ_p norm) is a proper metric defined as follows:

$$\begin{aligned} d_p(\mathbf{x}, \mathbf{x}') &= ||\mathbf{x} - \mathbf{x}'||_p \\ &= \left(\sum_{i=1}^{D} |x_d - x'_d|^p \right)^{1/p} \end{aligned}$$

Special cases include Euclidean distance ($p = 2$), Manhattan distance ($p = 1$) and Chebyshev distance ($p = \infty$).

Brute Force KNN

- Given any distance function d, brute force KNN works by computing the distance $d_i = d(\mathbf{x}_i, \mathbf{x}_*)$ from a target point $\mathbf{x}_*$ to all of the training points $\mathbf{x}_i$.

Brute Force KNN

- Given any distance function d, brute force KNN works by computing the distance $d_i = d(\mathbf{x}_i, \mathbf{x}_*)$ from a target point $\mathbf{x}_*$ to all of the training points $\mathbf{x}_i$.
- You then simply sort the distances $\{d_i, i = 1 : N\}$ and choose the data cases with the K smallest distances to form the neighbor set $\mathcal{N}_K(\mathbf{x}_*)$. Using a similarity function is identical, but you select the K most similar data cases.

Brute Force KNN

- Given any distance function d, brute force KNN works by computing the distance $d_i = d(\mathbf{x}_i, \mathbf{x}_*)$ from a target point $\mathbf{x}_*$ to all of the training points $\mathbf{x}_i$.
- You then simply sort the distances $\{d_i, i = 1 : N\}$ and choose the data cases with the K smallest distances to form the neighbor set $\mathcal{N}_K(\mathbf{x}_*)$. Using a similarity function is identical, but you select the K most similar data cases.
- Once the K neighbors are selected, applying the classification rule is easy.

KNN Variants

- Instead of giving all of the K neighbors equal weight in the majority vote, a distance-weighted majority can be used:

$$f_{KNN}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \frac{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i \mathbb{I}[y_i = y]}{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i}$$

$$w_i = \exp(-\alpha d_i)$$

KNN Variants

- Instead of giving all of the K neighbors equal weight in the majority vote, a distance-weighted majority can be used:

$$f_{KNN}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \frac{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i \mathbb{I}[y_i = y]}{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i}$$

$$w_i = \exp(-\alpha d_i)$$

- Instead of a brute force nearest neighbor search, data structures like ball trees can be constructed over the training data that support nearest neighbor search with lower amortized computational complexity.

KNN Trade-Offs

- Low bias: Converges to the correct decision surface as data goes to infinity.

KNN Trade-Offs

- Low bias: Converges to the correct decision surface as data goes to infinity.
- High variance: Lots of variability in the decision surface when amount of data is low.

KNN Trade-Offs

- Low bias: Converges to the correct decision surface as data goes to infinity.
- High variance: Lots of variability in the decision surface when amount of data is low.
- Curse of dimensionality: Everything is far from everything else in high dimensions.

KNN Trade-Offs

- Low bias: Converges to the correct decision surface as data goes to infinity.
- High variance: Lots of variability in the decision surface when amount of data is low.
- Curse of dimensionality: Everything is far from everything else in high dimensions.
- Space and Time Complexity: Need to store all training data and perform neighbor search can make the method use a lot of memory and take a lot of time.

Outline

Decision Tree

- A classical decision tree classifies data cases using a conjunction of rules organized into a binary tree structure.

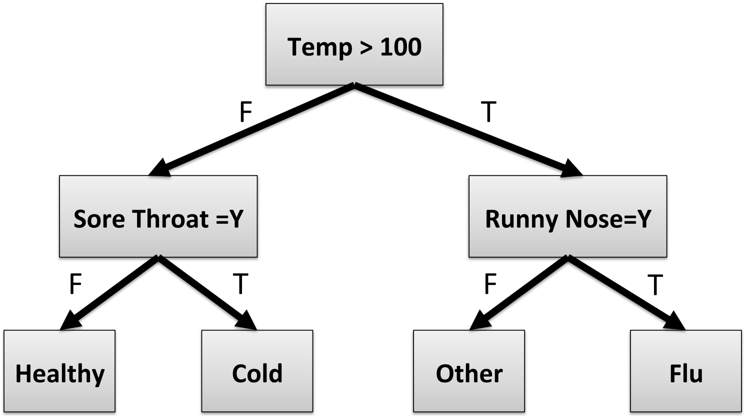
Decision Tree

- A classical decision tree classifies data cases using a conjunction of rules organized into a binary tree structure.
- Each internal node in a classical decision tree contains a rule of the form $(x_d < t)$ or $(x_d = t)$ that tests a single data dimension d against a single value t and assigns the data case to it's left or right sub-tree according to the result.

Decision Tree

- A classical decision tree classifies data cases using a conjunction of rules organized into a binary tree structure.
- Each internal node in a classical decision tree contains a rule of the form $(x_d < t)$ or $(x_d = t)$ that tests a single data dimension d against a single value t and assigns the data case to it's left or right sub-tree according to the result.
- A data case is routed through the tree from the root to a leaf. Each leaf node is associated with a class, and a data case is assigned the class of the leaf node it is routed to.

Example: Decision Tree for Flu



Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.

Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.
- The learning algorithm begins with all data cases at the root of the tree.

Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.
- The learning algorithm begins with all data cases at the root of the tree.
- The algorithm selects a variable and a threshold to split on according to a heuristic.

Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.
- The learning algorithm begins with all data cases at the root of the tree.
- The algorithm selects a variable and a threshold to split on according to a heuristic.
- The algorithm applies the chosen rule and assigns each data case to the left or right sub-tree.

Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.
- The learning algorithm begins with all data cases at the root of the tree.
- The algorithm selects a variable and a threshold to split on according to a heuristic.
- The algorithm applies the chosen rule and assigns each data case to the left or right sub-tree.
- The algorithm then recurses on the child nodes until a given stopping condition is satisfied.

Decision Tree Learning Algorithm

- Decision trees are learned using recursive greedy algorithms that select the variable and threshold at each node from top to bottom.
- The learning algorithm begins with all data cases at the root of the tree.
- The algorithm selects a variable and a threshold to split on according to a heuristic.
- The algorithm applies the chosen rule and assigns each data case to the left or right sub-tree.
- The algorithm then recurses on the child nodes until a given stopping condition is satisfied.
- When the stopping condition is satisfied, the current node is a leaf in the tree. It is typically assigned a label that corresponds to the most common label of the data cases it contains.

Decision Tree Learning

- The main stopping criteria used are all data cases assigned to a node have the same label, the number of data cases assigned to the node falls below a threshold, and the node is at the maximum allowable depth.

Decision Tree Learning

- The main stopping criteria used are all data cases assigned to a node have the same label, the number of data cases assigned to the node falls below a threshold, and the node is at the maximum allowable depth.
- Given sufficient depth, a decision tree can approximate any classification function to arbitrary accuracy.

Decision Tree Learning

- The main stopping criteria used are all data cases assigned to a node have the same label, the number of data cases assigned to the node falls below a threshold, and the node is at the maximum allowable depth.
- Given sufficient depth, a decision tree can approximate any classification function to arbitrary accuracy.
- There are a number of heuristics for selecting variables and thresholds to split on. In general, these heuristics are aimed at producing splits of the training data that are as homogeneous as possible in terms of the labels.

Decision Tree Trade-Offs

- Interpretability: The learned model is easy to understand as a collection of rules.

Decision Tree Trade-Offs

- Interpretability: The learned model is easy to understand as a collection of rules.
- Test Complexity: Shallow trees can be extremely fast classifiers at test time.

Decision Tree Trade-Offs

- Interpretability: The learned model is easy to understand as a collection of rules.
- Test Complexity: Shallow trees can be extremely fast classifiers at test time.
- Train Complexity: Finding optimal trees is NP-complete, thus need for greedy heuristics.

Decision Tree Trade-Offs

- Interpretability: The learned model is easy to understand as a collection of rules.
- Test Complexity: Shallow trees can be extremely fast classifiers at test time.
- Train Complexity: Finding optimal trees is NP-complete, thus need for greedy heuristics.
- Representation: Splitting on single variables can require very large trees to accurately model non-axis aligned decision boundaries.