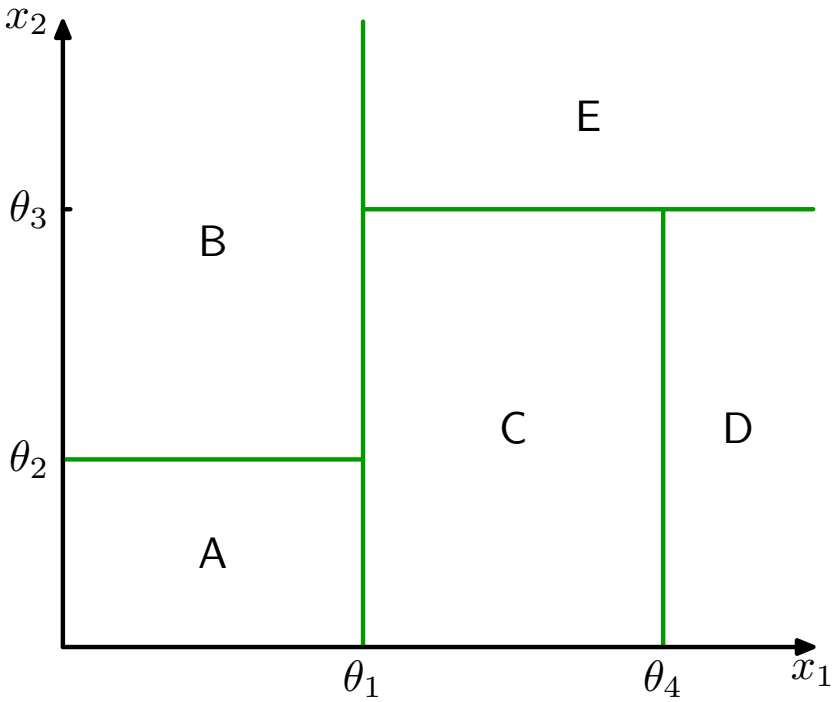
Decision trees (recap), Nearest Neighbors

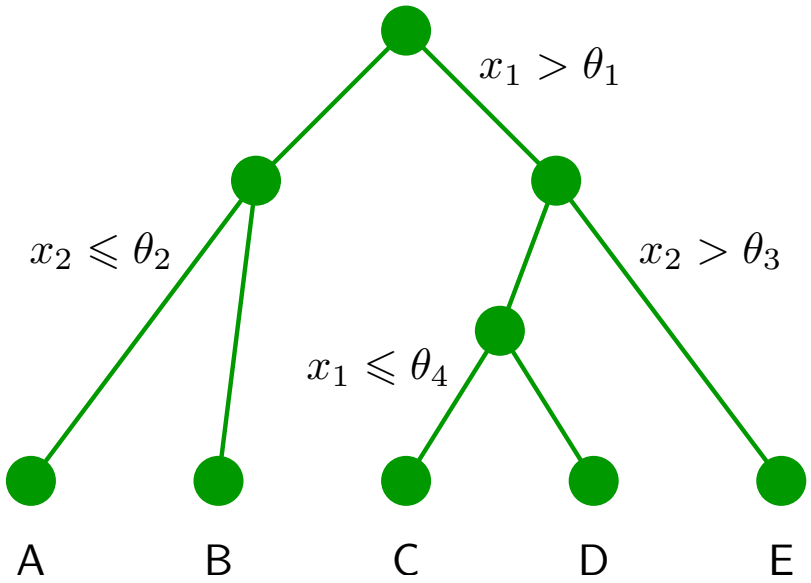
Sriram Sankararaman

The instructor gratefully acknowledges Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Outline

A tree partitions the feature space





Decision tree learning

Setup

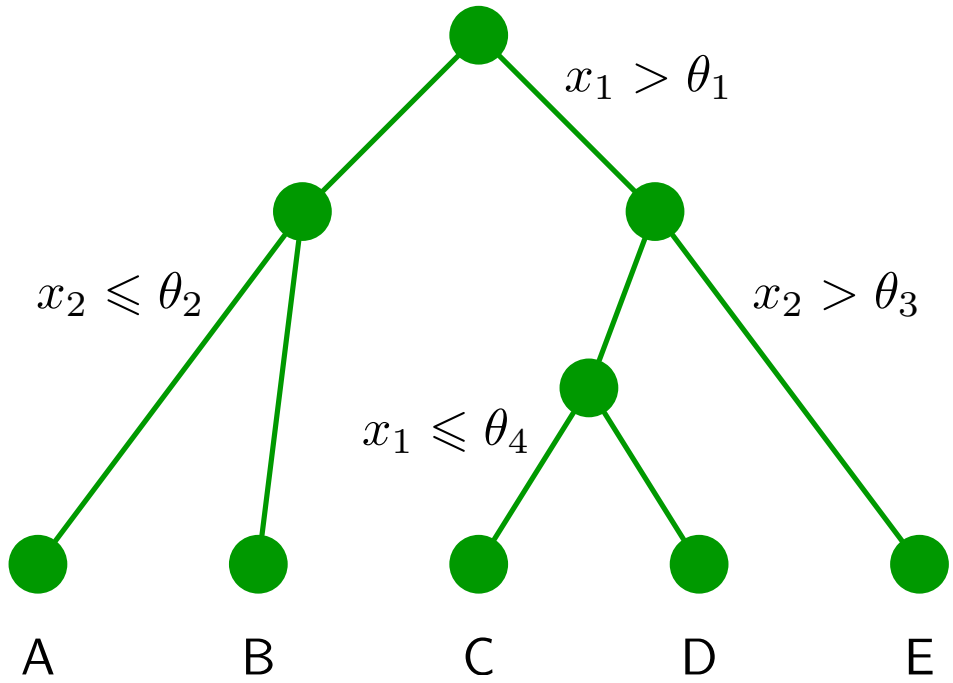
- Set of possible instances $\mathbb{X}$
  - Each instance $\boldsymbol{x} \in \mathbb{X}$ is a feature vector
- Set of possible labels $\mathbb{Y}$
  - $\mathbb{Y}$ is discrete valued
- Unknown target function $f : \mathbb{X} \rightarrow \mathbb{Y}$
- Model/Hypotheses: $H = \{h | h : \mathbb{X} \rightarrow \mathbb{Y}\}$.
- Each hypothesis h is a decision tree

Goal: Train/induce/learn a function h that maps instance to label.

Learning a tree model

Three things to learn:

1. The structure of the tree.
2. The threshold values (θ_i).
3. The values for the leaves ($A, B, \ldots$).



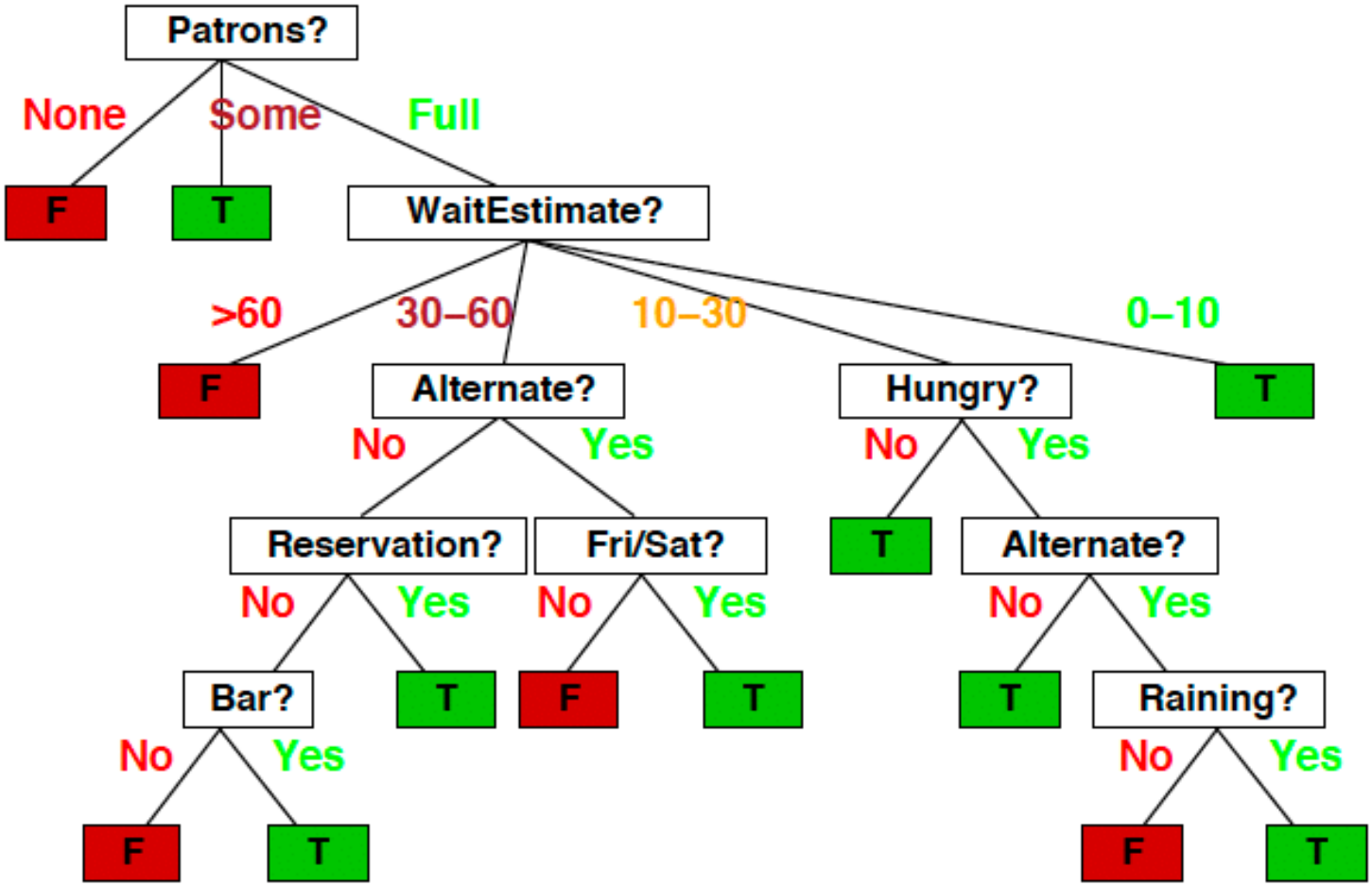
Choosing a restaurant

(Example from Russell & Norvig, AIMA)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Classification of examples is positive (T) or negative (F)

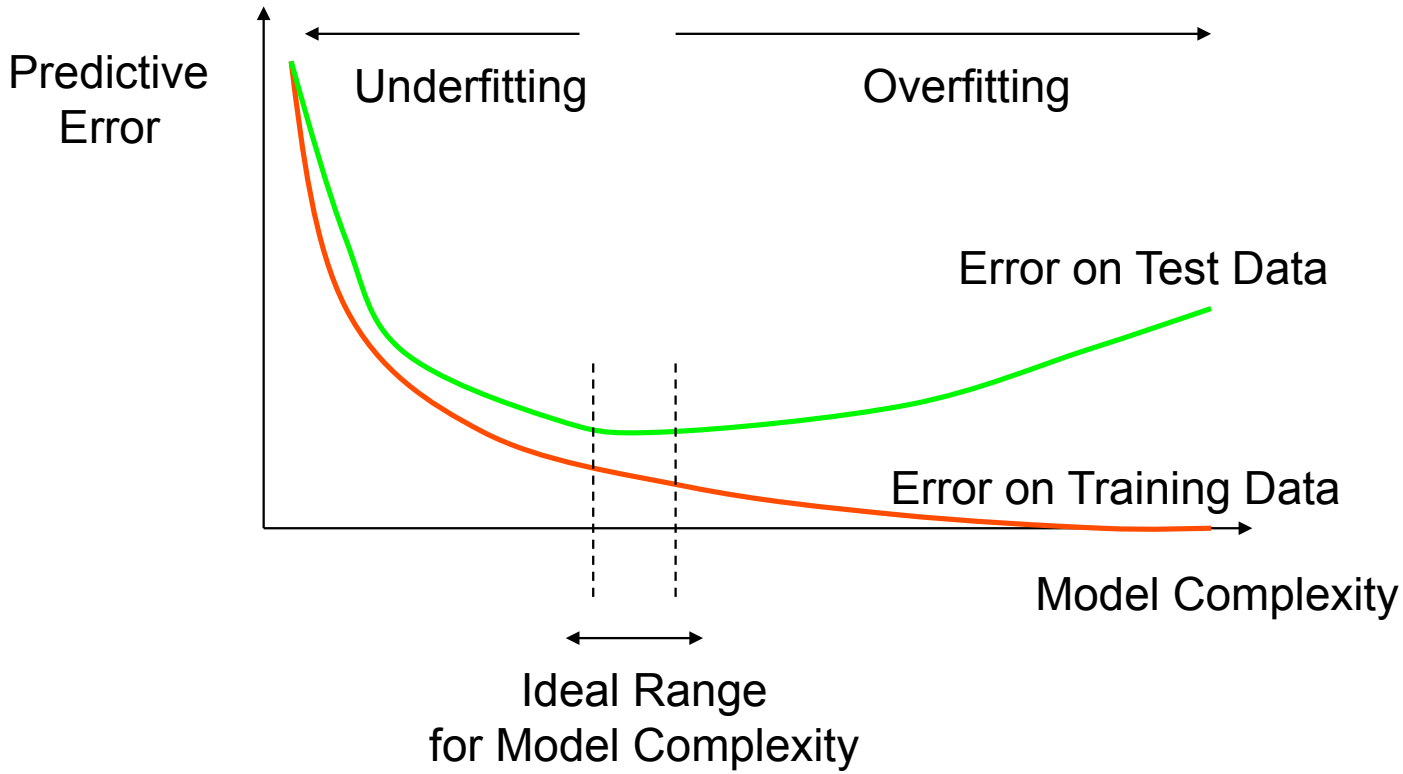
Greedily we build the tree and get this



What is the optimal Tree Depth?

- We need to be careful to pick an appropriate tree depth
- If the tree is too deep, we can overfit (memorize the training data).
- If the tree is too shallow, we underfit (not learn enough).
- Max depth is a hyperparameter that should be tuned by the data
  - A parameter that controls the other parameters of the tree.
  - Max depth of 0: underfitting
  - Max depth of ∞: overfitting.

Overfitting



Outline

Hyperparameters in Decision tree

Max depth of the tree

- Critical to choose max depth carefully for good generalization.
- Choosing max depth (as well as any other hyperparameters) based on empirical studies.

Hyperparameters in Decision tree

Max depth of the tree

- Critical to choose max depth carefully for good generalization.
- Choosing max depth (as well as any other hyperparameters) based on empirical studies.

How do we choose max depth?

Hyperparameter selection (tuning) by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Hyperparameter selection (tuning) by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{M}}, y_{\text{M}})\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Validation (or development) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{L}}, y_{\text{L}})\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

Recipe

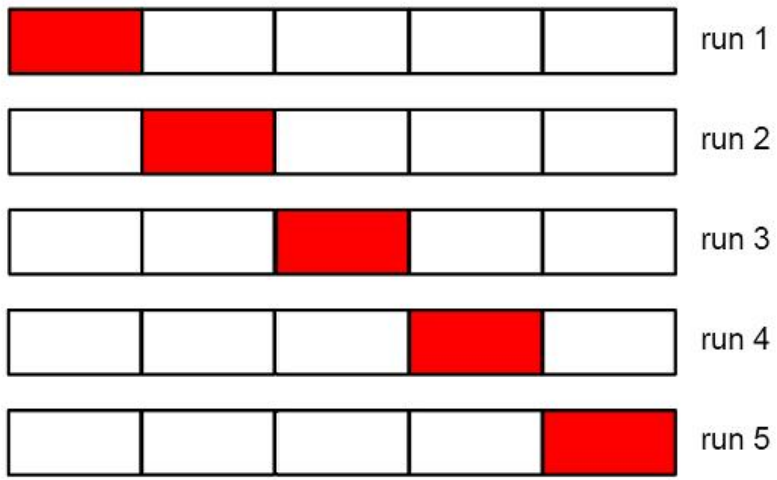
- For each possible value of the hyperparameter (say Max depth $= 1, 3, \cdots$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Cross-validation (CV)

What if we do not have validation data?

- We split the training data into K equal parts (termed folds or splits).
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that *on average*, the model performing the best

$K = 5$: 5-fold cross validation



Special case: when K = N, this will be leave-one-out (LOO).

Recipe

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$

Recipe

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say Max depth $= 1, 3, \cdots$)
  - for every $k \in [1, \text{K}]$
    - ★ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
  - Average the K performance metrics

Recipe

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say Max depth $= 1, 3, \cdots$)
  - for every $k \in [1, \text{K}]$
    - ★ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
  - Average the K performance metrics
- Choose the hyperparameter corresponding to the best averaged performance

Recipe

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say Max depth $= 1, 3, \cdots$)
  - for every $k \in [1, \text{K}]$
    - ★ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
  - Average the K performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$

Recipe

- Split the training data into K equal parts. Denote each part as $\mathcal{D}_k^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say Max depth $= 1, 3, \cdots$)
  - for every $k \in [1, \text{K}]$
    - ★ Train a model using $\mathcal{D}_{\setminus k}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_k^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_k^{\text{TRAIN}}$
  - Average the K performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Summary

- Model/hypotheses: Trees
- Have parameters: Features at each internal node, values along edges and labels at leaves.
- Have hyperparameters: max depth

Summary

Advantages of using trees

- Easily interpretable by humans (as long as the tree is not too big)
- Computationally efficient
- Handles both numerical and categorical data

Disadvantages

- Heuristic training techniques
  - Finding partition of space that minimizes empirical error is NP-hard
  - We resort to greedy approaches with limited theoretical underpinnings

Summary

- You should now be able to use decision trees to do machine learning.
- Given data, use training, development and test splits (or cross-validation).
- Use training and development to tune for Max depth that trades off overfitting and underfitting.
- Use test to get an estimate of generalization or accuracy on unseen data.
- Read 1.3,1.4-1.10 of CIML

Outline

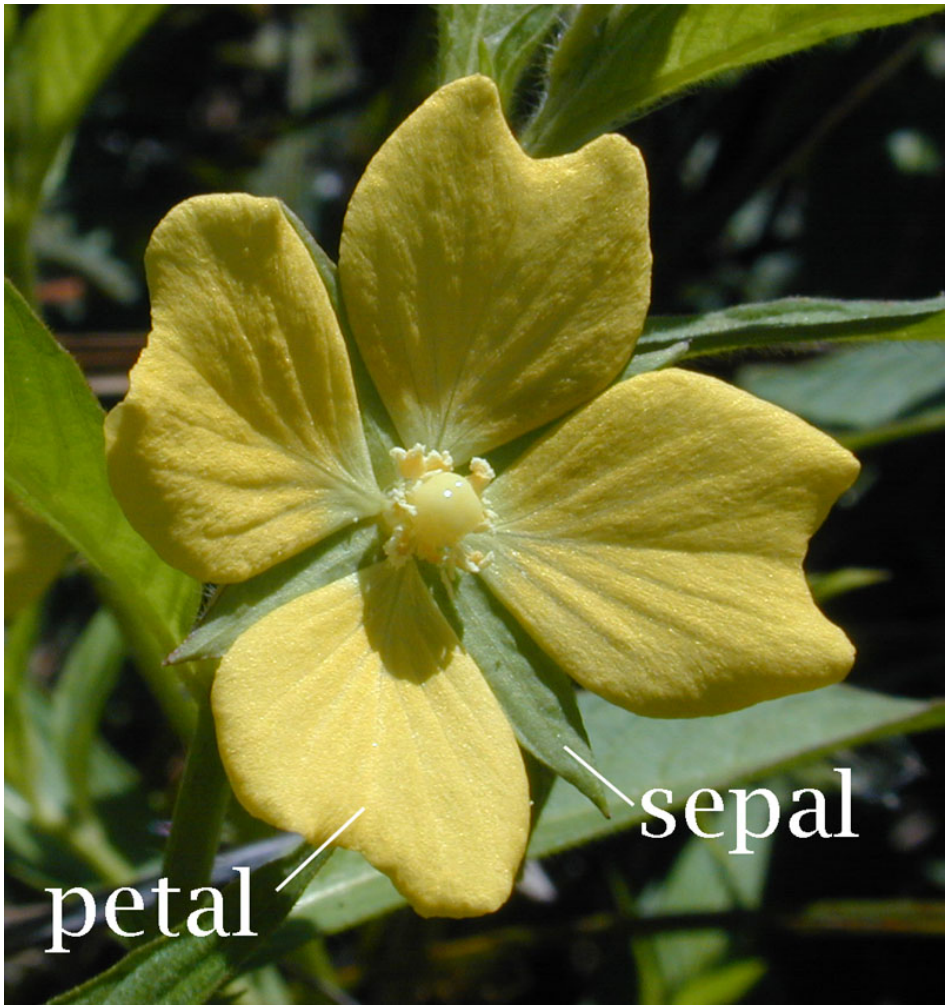
Recognizing flowers

Types of Iris: setosa, versicolor, and virginica



Measuring the properties of the flowers

Features: the widths and lengths of sepal and petal



Snapshot of Iris data

Iris data

- 4 features
- 3 classes

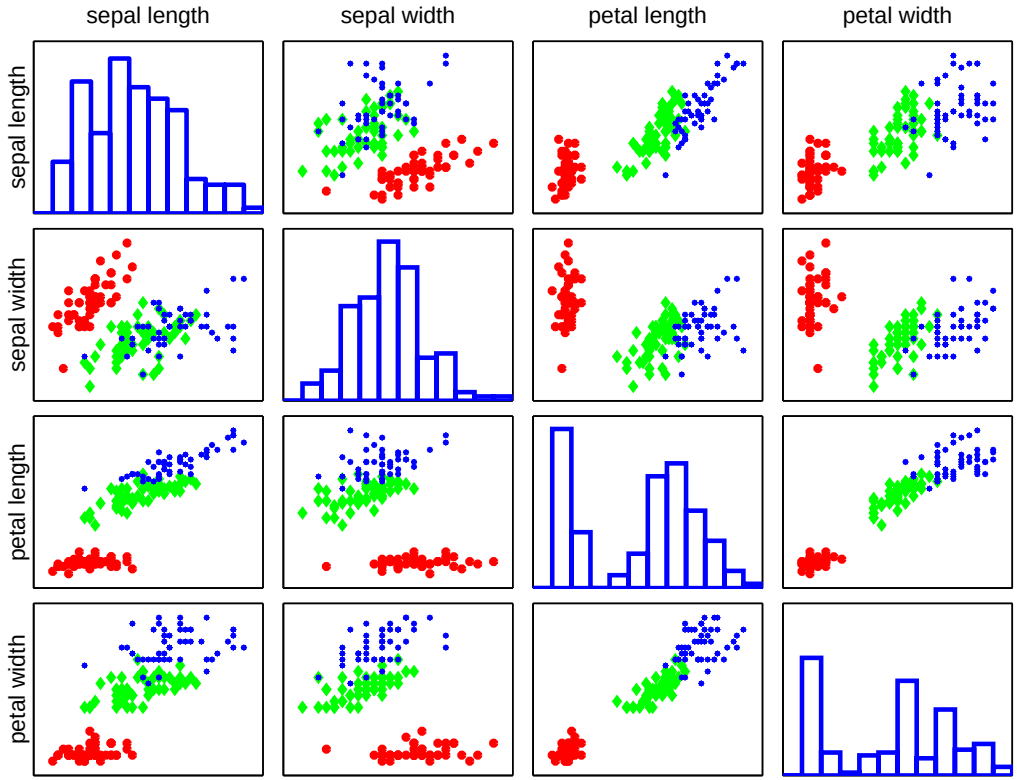
Fisher's *Iris* Data

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*

Pairwise scatter plots of 131 flower specimens

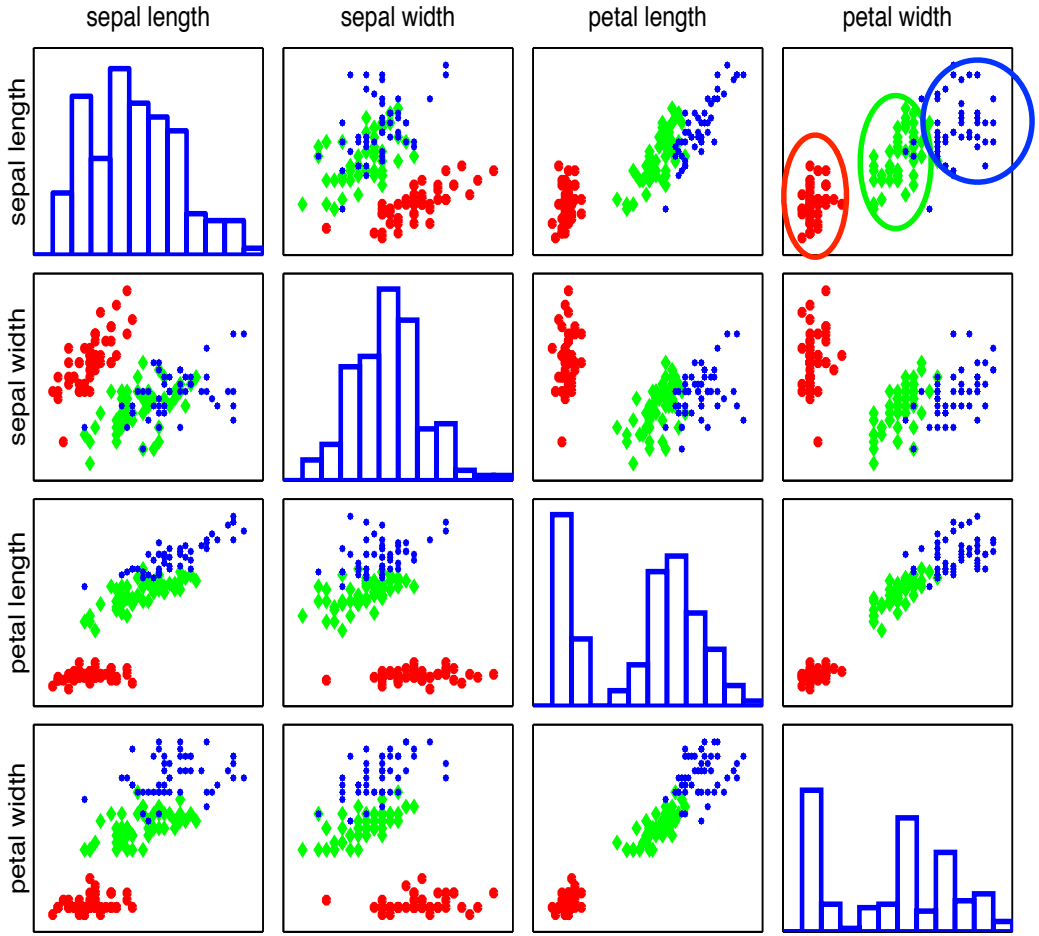
Visualization helps to identify the right learning model to use

Each colored point is an instance (flower specimen): setosa, versicolor, virginica

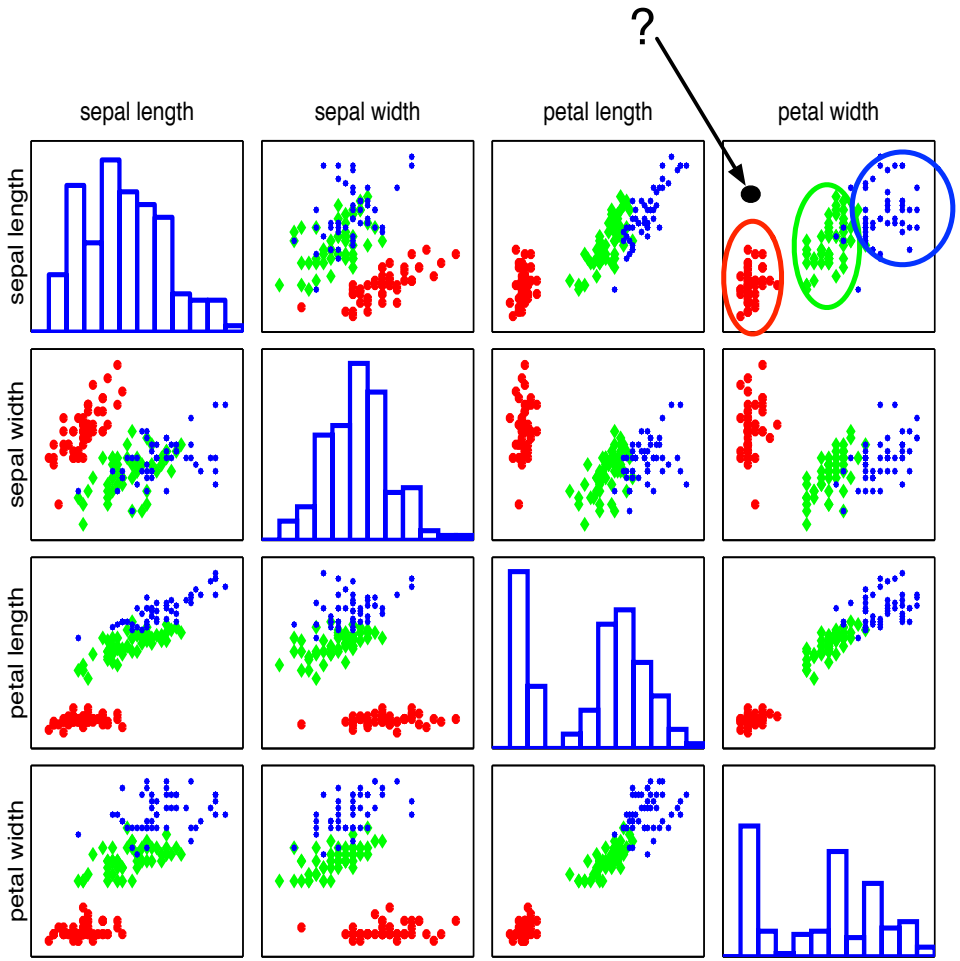


Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type



Closer to red cluster: so labeling it as setosa

Inductive bias

- Label of point (instance) is similar to the label of nearby points.

Multi-class classification

Classify data into one of the multiple categories

- Instance (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Label: $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- Learning goal: $y = h(\boldsymbol{x})$

Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

Terminology

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{M}}, y_{\text{M}})\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Training data and test data should *not* overlap: $\mathcal{D}^{\text{TRAIN}} \cap \mathcal{D}^{\text{TEST}} = \emptyset$

Nearest neighbor classification (NNC)

Training

- Store the entire training set.

Nearest neighbor classification (NNC)

Testing

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances

Nearest neighbor classification (NNC)

Testing

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances

$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 = \arg\min_{n \in [\mathsf{N}]} \sum_{d=1}^{\mathsf{D}} (x_d - x_{nd})^2$$

Nearest neighbor classification (NNC)

Testing

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances

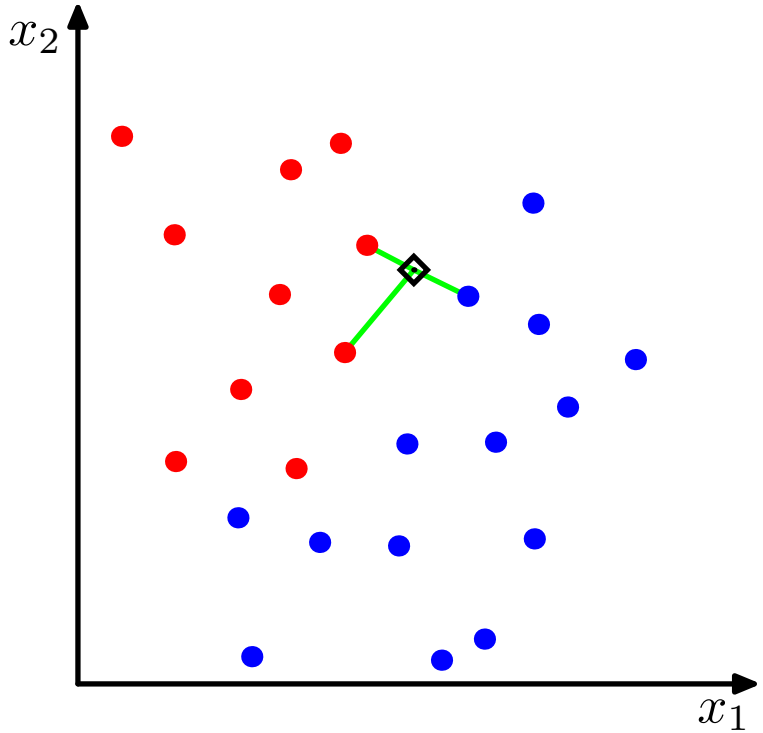
$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 = \arg\min_{n \in [\mathsf{N}]} \sum_{d=1}^{\mathsf{D}} (x_d - x_{nd})^2$$

Classification rule

$$y = h(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

Visual example

In this 2-dimensional example, the nearest point to $\boldsymbol{x}$ is a red training instance, thus, $\boldsymbol{x}$ will be labeled as red.



(a)

Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setosa
2	1.4	7.0	versicolor
3	2.5	6.7	virginica

Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setosa
2	1.4	7.0	versicolor
3	2.5	6.7	virginica

Flower with unknown category

petal width $= 1.8$ and sepal length $= 6.4$

Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setosa
2	1.4	7.0	versicolor
3	2.5	6.7	virginica

Flower with unknown category

petal width $= 1.8$ and sepal length $= 6.4$

Calculating distance $= \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	1.75
2	0.72
3	0.76

Thus, the category is *versicolor* (the real category is *virginica*)

How to measure nearness with other distances?

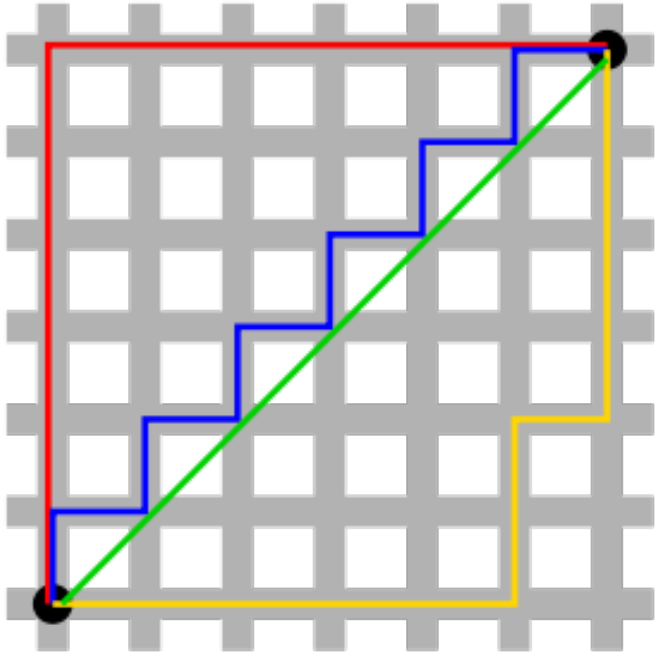
Previously, we use the Euclidean distance

$$\text{nn}(\boldsymbol{x}) = \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

We can also use alternative distances

E.g., the following L_1 distance (i.e., city block distance, or Manhattan distance)

$$\begin{aligned}\text{nn}(\boldsymbol{x}) &= \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_1 \\ &= \arg\min_{n \in [\text{N}]} \sum_{d=1}^{\text{D}} |x_d - x_{nd}|\end{aligned}$$



Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

How to compute distances with other types of features?

- Real-valued features (discussion so far).

How to compute distances with other types of features?

- Real-valued features (discussion so far).
- Binary features :

How to compute distances with other types of features?

- Real-valued features (discussion so far).
- Binary features : map to 0 and 1.
  - Example: Is flower white or colored ?

How to compute distances with other types of features?

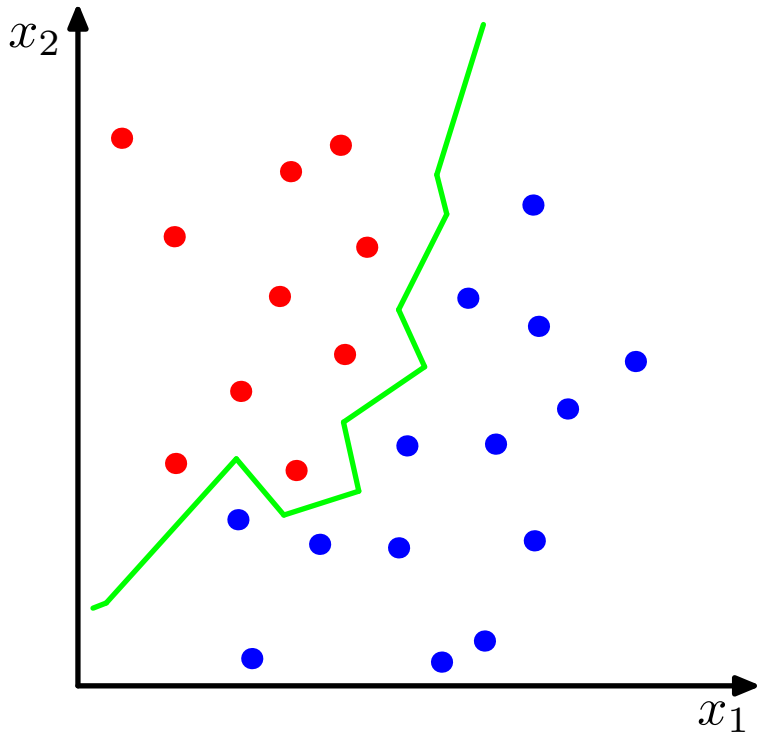
- Real-valued features (discussion so far).
- Binary features : map to 0 and 1.
  - Example: Is flower white or colored ?
- Categorical features with V values:

How to compute distances with other types of features?

- Real-valued features (discussion so far).
- Binary features : map to 0 and 1.
  - Example: Is flower white or colored ?
- Categorical features with V values: map to V-binary features.
  - Example: Color (white, blue, pink): map to three binary features IsItWhite?, IsItBlue? IsItPink?
  - Termed one-hot encoding.

Decision boundary

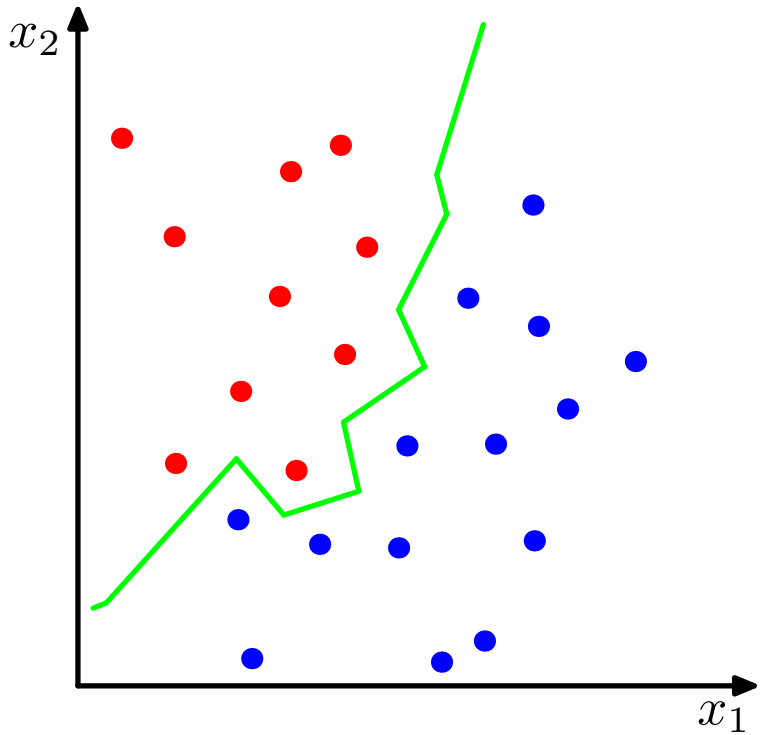
For every point in the space, determine its label using the NNC rule. Gives rise to a *decision boundary* that partitions the space into different regions.



(b)

Decision boundary

For every point in the space, determine its label using the NNC rule. Gives rise to a *decision boundary* that partitions the space into different regions.



(b)

Compare to decision boundary of decision trees

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\boldsymbol{x}) = \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x}) - \text{nn}_2(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$

K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: $\text{nn}_1(\boldsymbol{x}) = \arg\min_{n \in [\text{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 2nd-nearest neighbor: $\text{nn}_2(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$
- 3rd-nearest neighbor: $\text{nn}_3(\boldsymbol{x}) = \arg\min_{n \in [\text{N}] - \text{nn}_1(\boldsymbol{x}) - \text{nn}_2(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$

The set of K-nearest neighbors

$$\text{knn}(\boldsymbol{x}) = \{\text{nn}_1(\boldsymbol{x}), \text{nn}_2(\boldsymbol{x}), \cdots, \text{nn}_K(\boldsymbol{x})\}$$

Let $\boldsymbol{x}(k) = \boldsymbol{x}_{\text{nn}_k(\boldsymbol{x})}$, then

$$\|\boldsymbol{x} - \boldsymbol{x}(1)\|_2^2 \leq \|\boldsymbol{x} - \boldsymbol{x}(2)\|_2^2 \cdots \leq \|\boldsymbol{x} - \boldsymbol{x}(K)\|_2^2$$

How to classify with K neighbors?

How to classify with K neighbors?

Classification rule

- Every neighbor votes: suppose y_n (the label) for $\boldsymbol{x}_n$ is c, then
  - vote for c is 1
  - vote for $c' \neq c$ is 0

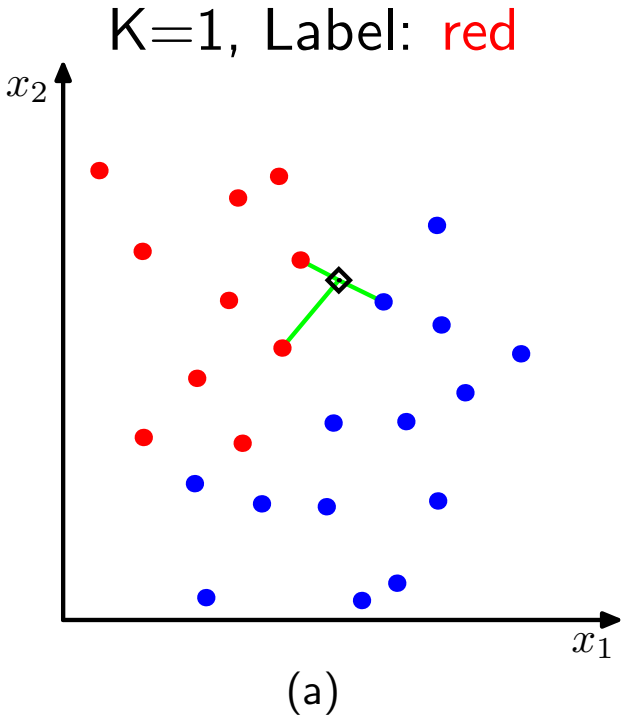
  We use the *indicator function* $\mathbb{I}(y_n == c)$ to represent.
- Aggregate everyone's vote

$$v_c = \sum_{n \in \mathsf{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\mathsf{C}]$$

- Label with the majority

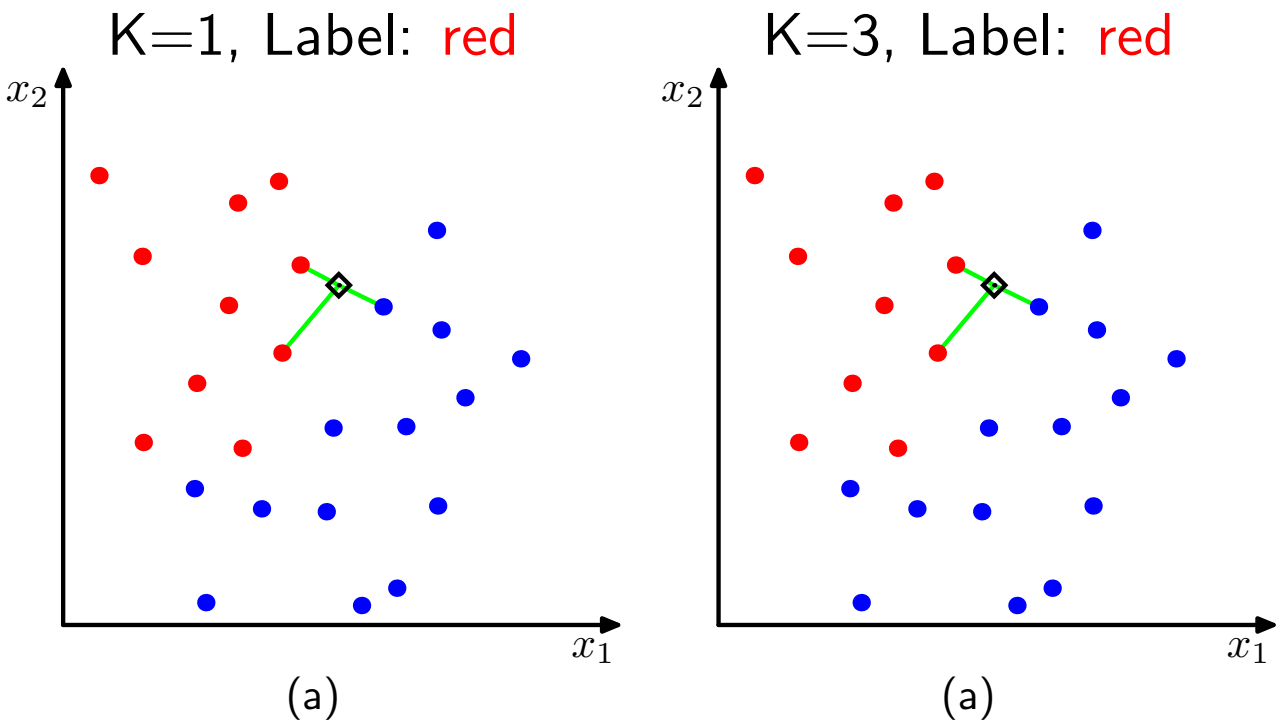
$$y = h(\boldsymbol{x}) = \arg\max_{c \in [\mathsf{C}]} v_c$$

Example

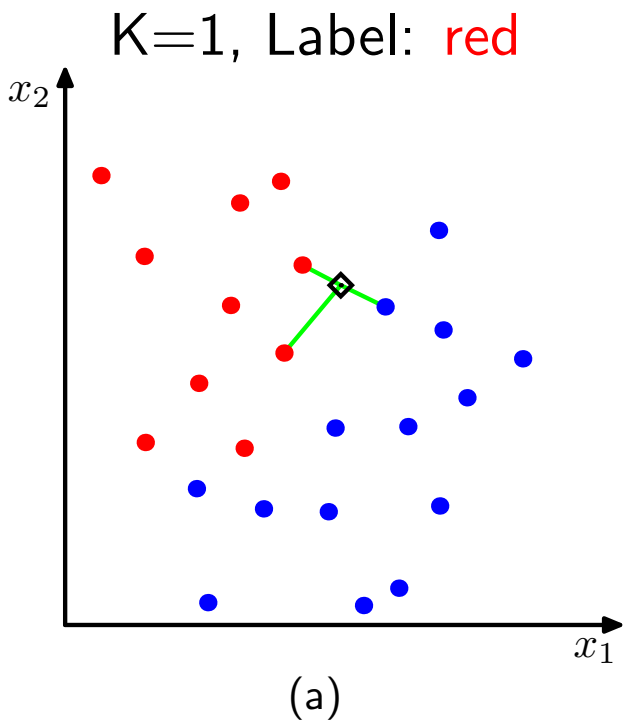


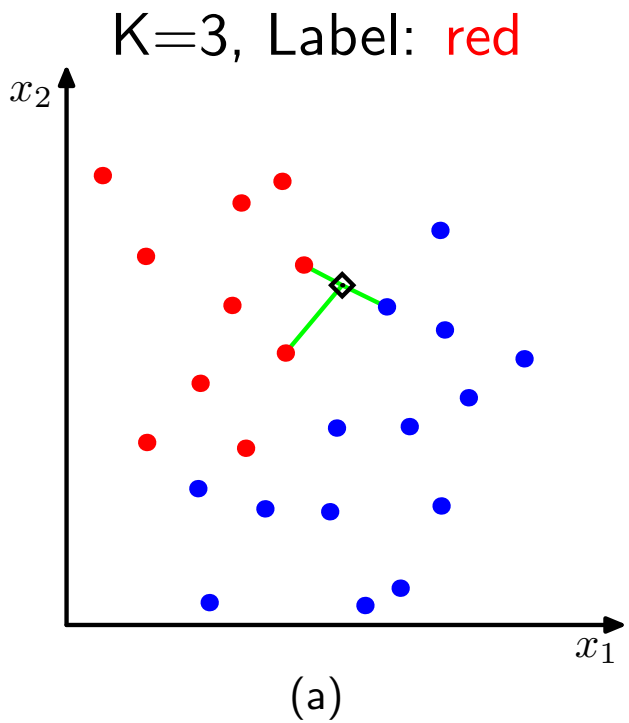
(a)

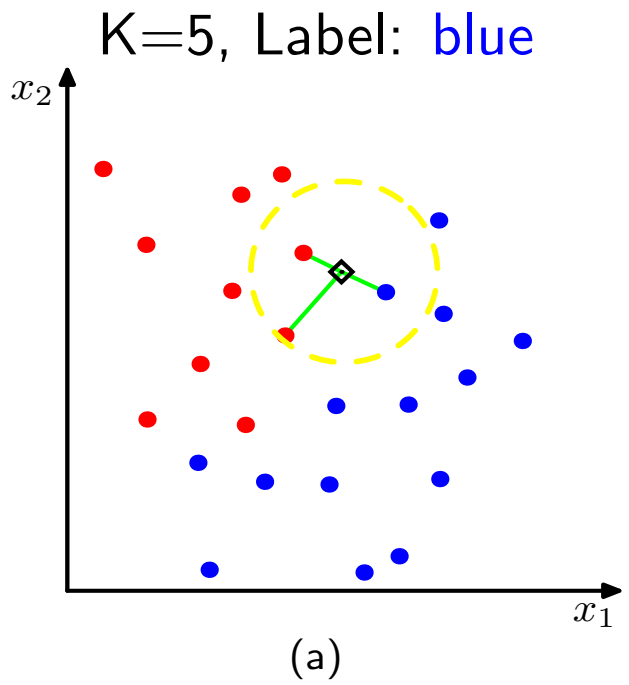
Example



Example

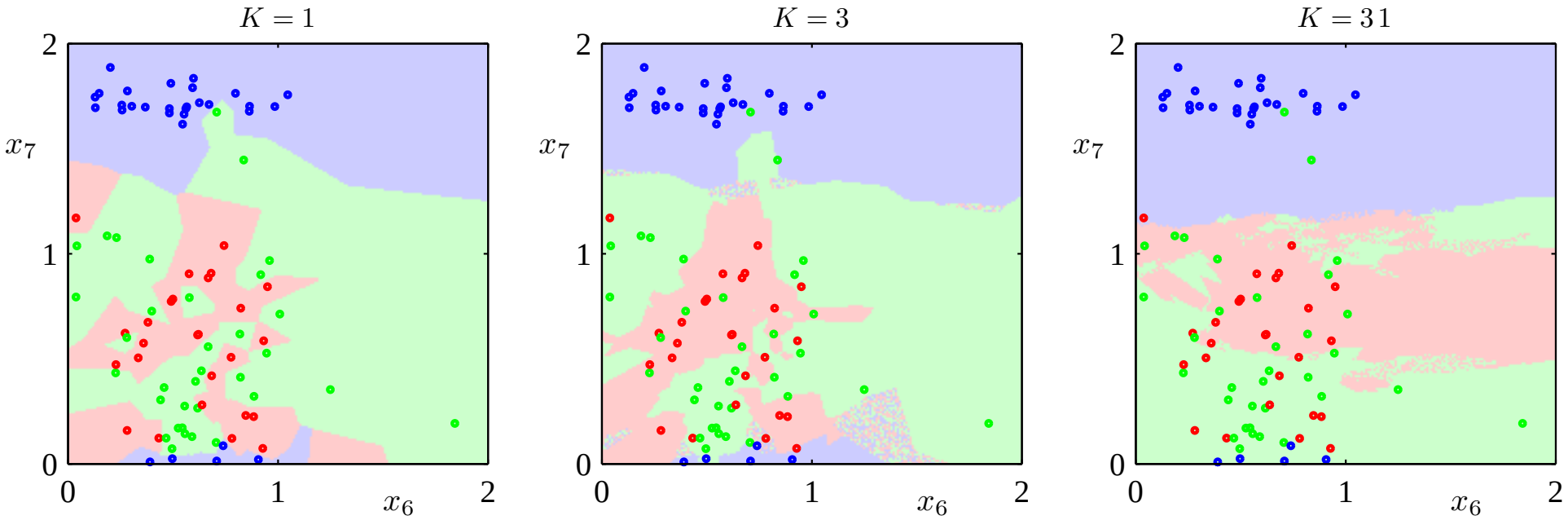






Decision boundary as a function of K

Decision boundary as a function of K



When K increases, the decision boundary becomes smooth.

Mini-summary

Advantages of NNC

- Computationally, simple and easy to implement – just computing the distance

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(\mathrm{N}D)$ for labeling a data point
- We need to "carry" the training data around. Without it, we cannot do classification. This type of method is called *nonparametric*.
- Choosing the right distance measure and K can be involved.

Mini-summary

Advantages of NNC

- Computationally, simple and easy to implement – just computing the distance

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(\mathrm{N}D)$ for labeling a data point
- We need to "carry" the training data around. Without it, we cannot do classification. This type of method is called *nonparametric*.
- Choosing the right distance measure and K can be involved.

Hypeparameters in NNC

Two practical issues about NNC

- Choosing K, i.e., the number of nearest neighbors (default is 1)
- Choosing the right distance measure (default is Euclidean distance), for example, from the following generalized distance measure

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_p = \left(\sum_d |x_d - x_{nd}|^p \right)^{1/p}$$

for $p \geq 1$.

Those are not specified by the algorithm itself — resolving them requires empirical studies and are task/dataset-specific.

Tuning by using a validation dataset

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{N}}, y_{\text{N}})\}$
- They are used for learning $h(\cdot)$

Test (evaluation) data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{M}}, y_{\text{M}})\}$
- They are used for assessing how well $h(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Development (or validation) data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\text{L}}, y_{\text{L}})\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

Recipe

- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

What if we do not have validation data?

What if we do not have validation data?

- Use cross-validation.

Yet, another practical issue with NNC

Assumes all features are equally important!

- Distances depend on units of the features.

Preprocess data

Normalize data to have zero mean and unit standard deviation in each dimension

- Compute the means and standard deviations in each feature

$$\bar{x}_d = \frac{1}{\mathrm{N}} \sum_n x_{nd}, \qquad s_d^2 = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$$

- Scale the feature accordingly

$$x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}$$

Many other ways of normalizing data — you would need/want to try different ones and pick them using (cross)validation

Outline

Summary so far

- Decision trees and Nearest Neighbor Classifier: two simple learning algorithm
  - Used intensively in practical applications.
- Discussed a few practical aspects, such as tuning hyperparameters, with (cross)validation

Summary

- You should now be able to use decision trees and nearest neighbors to do machine learning.
- Given data, use training, development and test splits (or cross-validation).
- Use training and development to tune hyperparameters that trades off overfitting and underfitting.
- Use test to get an estimate of generalization or accuracy on unseen data.
- Further reading
  - Read 1.3,1.4-1.10 of CIML for decision trees (v0.90).
  - Read 2-2.3 of CIML for nearest neighbors (v0.90).