CAI 4104/6108 – Machine Learning Engineering:

Decision Trees

Prof. Vincent Bindschaedler

Spring 2024

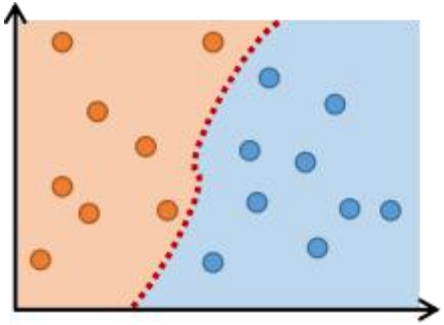
Administrivia

- Midterm is coming up!
  - Monday 2/19 and Wednesday 2/21 during class time (10:40 - 11:30) in FLG 0220
    - Topics: everything until 2/16
  - Duration: 50 minutes
  - Format: **closed-books**, (blank) scratch paper, and physical calculator are allowed (no phones!)
  - Questions: short answers + problems

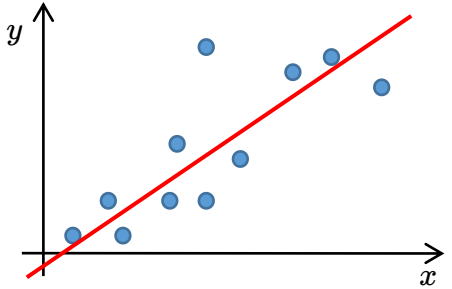
- Schedule:
  - **CAI6108**: take the exam on **2/19**
  - **CAI4104**: take the exam on **2/21**
  - Lecture that week (topic: Unsupervised Learning) will be pre-recorded

Reminder: Supervised Learning

- Classification
  - Task: predict the corresponding label
  - Different types:
    - Binary classification: there are only two classes (0,1; +,-, etc.)
    - Multiclass: more than two classes
    - Multi-label: each instance can belong to more than one class
    - One-class: there is only one class, we want to distinguish it from everything else



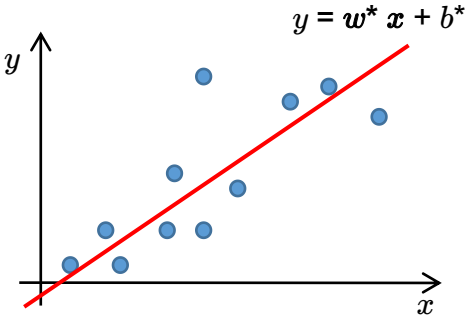
- Regression
  - Task: predict the corresponding value (typically a real number) or target
    - E.g.: you want to predict a person's future income based on their high school GPA



- Others:
  - Sequence-to-sequence, similarity learning/metric learning, learning to rank, etc.

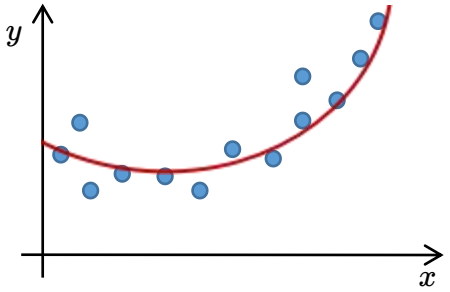
Reminder: Linear Regression

- Dataset
  - Matrix $\mathbf{X}$ $(n \times m)$ and the target vector $\mathbf{y}$ $(n \times 1)$
    - Let $\boldsymbol{x_i}$ be the feature vector for example i and $y_i \in \mathbb{R}$ is the corresponding target/value
- Prediction task:
  - Given a feature vector $\boldsymbol{x}$, predict the target/value $y \in \mathbb{R}$ as accurately as possible
- Linear Regression:
  - The model is: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = h_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}\,\boldsymbol{x} + b$
  - The prediction is: $y = h_{\boldsymbol{\theta}}(\boldsymbol{x})$
- Training:
  - We want to minimize the Mean Squared Error (MSE) [this is called OLS]
    - $\text{MSE}(\boldsymbol{w},b) := \text{MSE}(h_{\boldsymbol{w},b}, \boldsymbol{X}, \boldsymbol{y}) = 1/n \sum_i [h_{\boldsymbol{w},b}(\boldsymbol{x_i}) - y_i]^2 = 1/n \sum_i [\boldsymbol{w}\,\boldsymbol{x_i} + b - y_i]^2$
  - Optimal parameters: $\boldsymbol{\theta}^* = (\boldsymbol{w}^*, b^*) = \text{argmin}_{\boldsymbol{w},b}\, \text{MSE}(\boldsymbol{w},b)$
  - Remark: MSE is the *expected* squared error loss
    - Squared Error Loss (L_2 loss): $L(\boldsymbol{\theta}) = [y - h_{\boldsymbol{\theta}}(\boldsymbol{x})]^2$



Reminder: Polynomial Regression

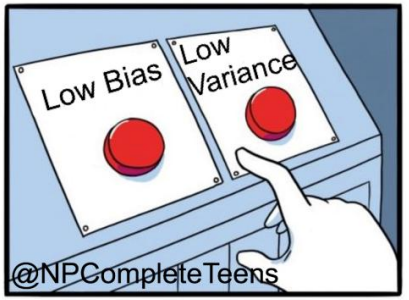
- What if the data is non-linear?
  - Then a linear model won't fit (it will have high bias)
- Can we still use linear regression?
  - Yes, we can fit a linear model on non-linear data!
  - How? Add features that can capture non-linearity!
  - Example: suppose we have a single feature
    - The linear regression model is: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = wx + b$
    - If we add x^2 as a feature, then the model is: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = w_1 x + w_2 x^2 + b$



- Polynomial regression
  - If we have several features, say x, y, z, then we can consider all combinations of features up to some degree. That is:
    - x^3, y^3, z^3, x^2y, x^2z, y^2x, y^2z, z^2x, z^2y, xyz (and x, y, z, 1)
  - Q: If we have m features and want all combinations up to degree k, how many features do we get?
    - m+k choose k: $C(m+k, k) = (m+k)! / (m!\ k!)$

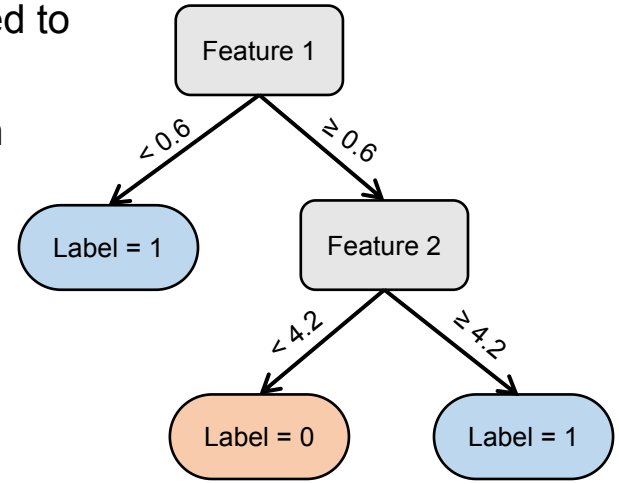
Reminder: Bias-Variance Tradeoff

- Generalization error (aka out-of-sample error or risk)
  - Prediction error on *unseen* data
  - Related to overfitting
    - If the model overfits, then the generalization error will be large
- Bias-Variance Tradeoff
  - Generalization error: $\texttt{bias}^2 + \texttt{variance} + \texttt{irreducible error}$
    - For more details:
      - Geman et al. "*Neural networks and the bias/variance dilemma.*" Neural computation (1992)
      - Kohavi et al. "Bias plus variance decomposition for zero-one loss functions." ICML, 1996.
  - Why is it a tradeoff?
    - Increasing model complexity => lower bias
    - Decreasing model complexity => lower variance
    - Note: there has been some debate of whether this applies to neural networks
      - E.g.: see Neal et al. "A modern take on the bias-variance tradeoff in neural networks." arXiv, 2018.



Decision Trees

- A decision tree is
  - An *acyclic* graph (i.e., a directed rooted tree) that can be used to make predictions
  - Nonparametric model suitable for classification or regression
    - What is the other nonparametric model we have seen?
  - Prediction:
    - Start at the root
    - Traverse the tree (branching according to feature values)
    - The leaf gives the predicted label or value/target



  - How is the tree constructed from the training data?
    - There are many algorithms and many different kinds of decision trees!

Learning Decision Trees

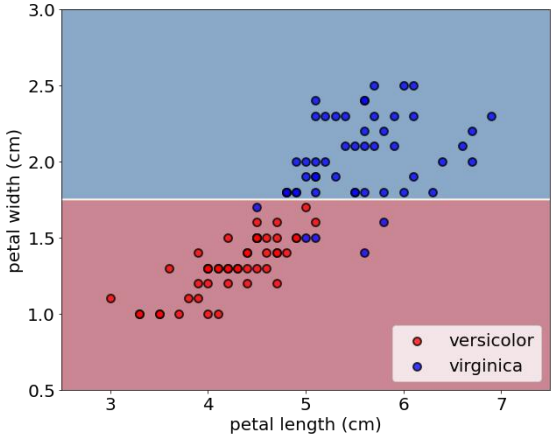
- A simplified (generic) training algorithm:
  - Input: set of examples $\mathcal{S}=\{(\boldsymbol{x_1}, y_1),(\boldsymbol{x_2}, y_2),\ldots,(\boldsymbol{x_n}, y_n)\}$
  - If stopping criterion is met:
    - (For classification:) set leaf label to be the majority label of examples in $\mathcal{S}$
    - (For regression:) set leaf value to be mean value of examples in $\mathcal{S}$
  - Find the next best feature j and threshold t for split
  - Split $\mathcal{S}$ into $\mathcal{S}_<$ and $\mathcal{S}_\geq$
    - $\mathcal{S}_<$ contains examples with $x_j < t$ and $\mathcal{S}_\geq$ contains examples with $x_j \geq t$
  - Recurse on $\mathcal{S}_<$ and $\mathcal{S}_\geq$

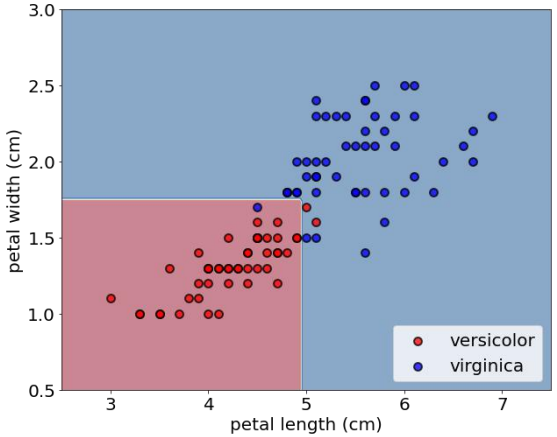
- There are many algorithms to train decision trees
  - Popular examples: ID3, C4.5, CART (used by scikit-learn)
  - They differ in stopping criterion, ways to find the best split, how they are regularized

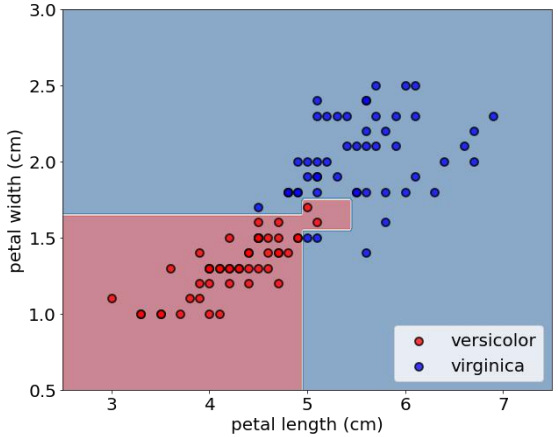
Learning Decision Trees

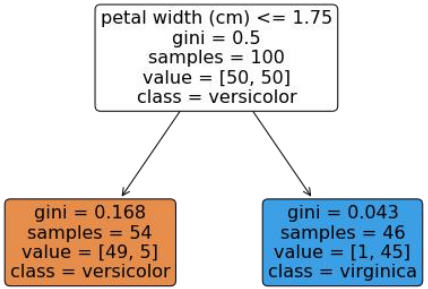
- How to find the best split?
  - Intuition: split so as to maximally distinguish between the classes
    - Ideal case: $\mathcal{S}_<$ has all examples of one class; $\mathcal{S}_\geq$ has all examples of the other class
  - Metrics:
    - Gini impurity ($\neq$ as Gini coefficient): $G_j = 1 - \sum_c [\, p_j(c) \,]^2$ where $p_j(c)$ is the probability of class c in node j
    - Entropy: $H_j = - \sum_c p_j(c) \log_2 p_j(c)$
  - For example (CART): Split to minimize $1/n \, [\, n_< G_< + n_\geq G_\geq \,]$
    - Note: CART only ever splits a node in two so that the tree is a binary tree (in contrast to some other algorithms)
- When to stop?
  - If any of the conditions are met:
    - All examples in leaves are classified correctly
    - Tree reaches some max depth
    - Cannot find a feature to split
    - Split does not significantly improve Gini impurity or entropy

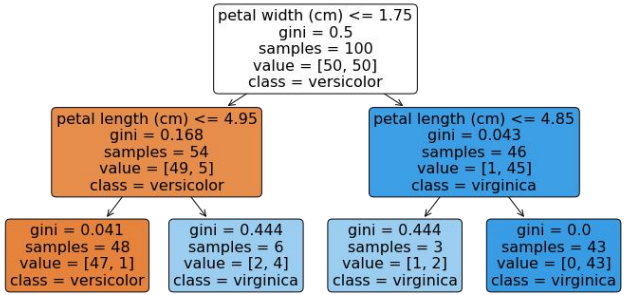
Examples

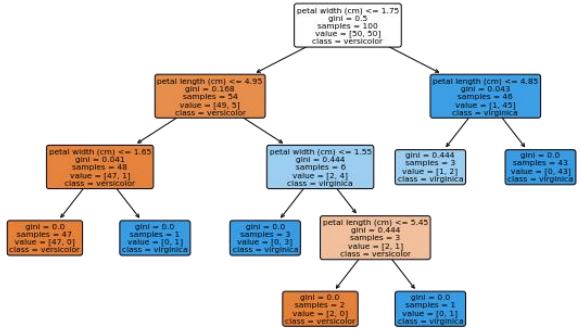




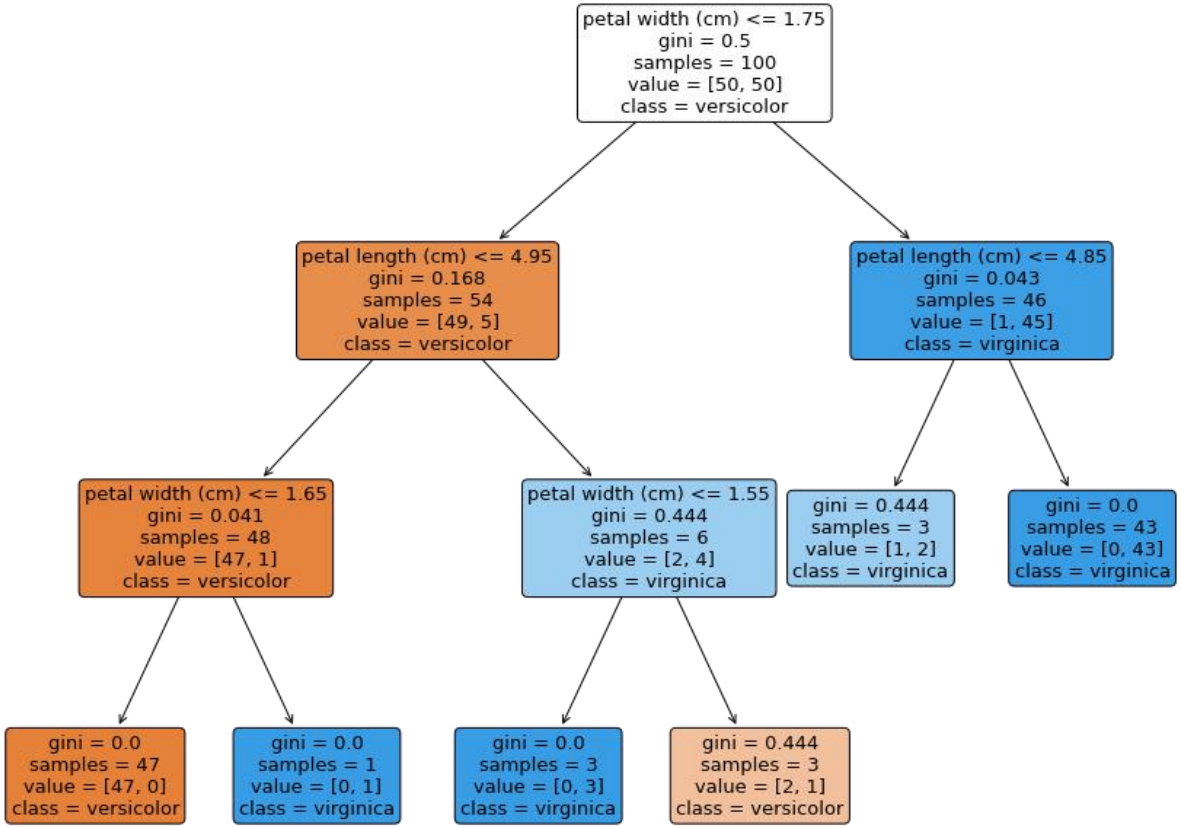


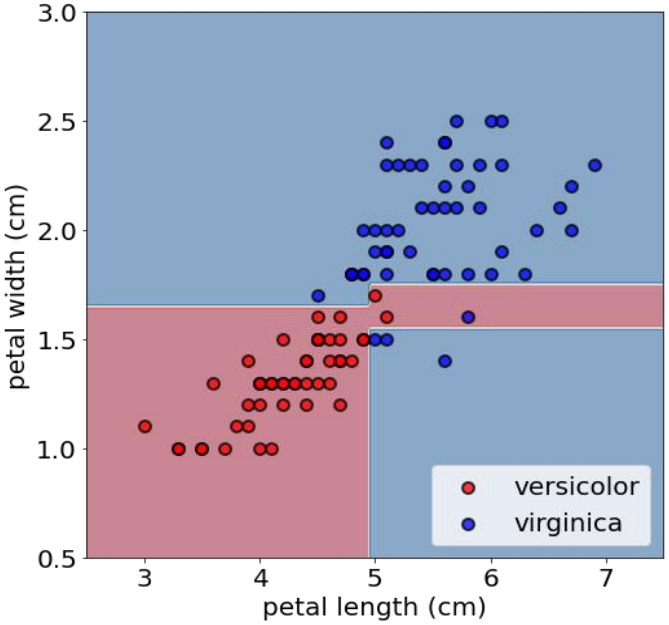






Example





Overfitting & Regularization

- Decision trees make almost no assumption about the data
  - So unless we control complexity, the tree structure will be made to (over)fit the data!

- How do we avoid overfitting?
  - Prevent the tree from growing too deep (e.g., set a maximum depth)
  - Restrict splitting (e.g., set a minimum number of examples to split)
  - Pruning: after the tree is created, prune branches that do not significantly reduce the error

- Regularization hyperparameters
  - Example: Scikit-learn CART
    - `max_depth`: maximum depth of tree (default = unlimited)
    - `min_samples_split`: minimum number of examples to split a node (default = 2)
    - `min_samples_leaf`: minimum number of examples for a leaf (default = 1)

Decision Trees: Advantages

- Scales to very large datasets easily
  - Training complexity: $O(m\ n^2\ d)$ with depth $d = O(\log_2(n))$ (if tree is balanced)
    - Note: smarter implementations can achieve $O(m\ n\ \log_2(n))$
  - Prediction complexity: $O(d)$
- Applicable to many supervised learning tasks
  - Supports classification, regression, even multi-output/multi-target
- Decisions are easy to understand and interpret
  - You can even visualize the tree (in theory at least...)
- Almost no data preprocessing / feature engineering required
  - Can handle both categorical and numerical features
  - No need to do feature scaling (Q: why?)
  - Irrelevant features typically don't get used (so not need to remove them)
  - (Some) algorithms can deal with missing feature values

Decision Trees: Drawbacks

- Finding the optimal decision tree is an NP-complete problem
  - But in practice greedy algorithms perform well
- Can create complex trees that overfit the data
  - Mitigation: constrain the structure of the tree somehow
- Trees are unstable to small variation in the data (high variance)
  - Adding/removing a single training example can change splits and thus the entire structure of the tree
- If data is imbalanced, trees may be biased

Next Time

- Wednesday (2/7): Lecture

- Upcoming:
  - Homework 2 will be out soon (due 2/14) by 11:59pm