K-Nearest Neighbours, Decision Trees, Ensemble Methods

Machine Learning 1 UE (INP.33761UF)

Thomas Wedenig

May 29, 2024

Institute of Theoretical Computer Science
Graz University of Technology, Austria

Table of contents

Parametric ML Models

- We've used **different ML models** in **supervised learning tasks**
  - Linear regression
  - Logistic regression
  - Neural Nets
- All these models had parameters $\boldsymbol{\theta} \in \mathbb{R}^D$
- Note that D is **fixed**, i.e., **independent of the size of the training data**

Non-Parametric Models

Parametric ML Models

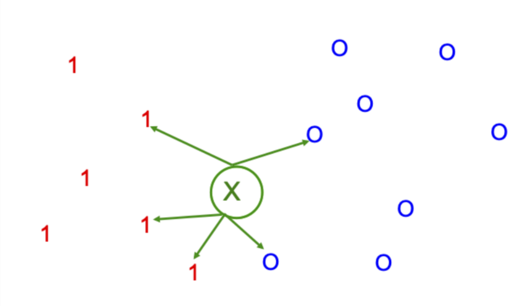
- We've used **different ML models** in **supervised learning tasks**
  - Linear regression
  - Logistic regression
  - Neural Nets
- All these models had parameters $\boldsymbol{\theta} \in \mathbb{R}^D$
- Note that D is **fixed**, i.e., **independent of the size of the training data**

Non-Parametric Models

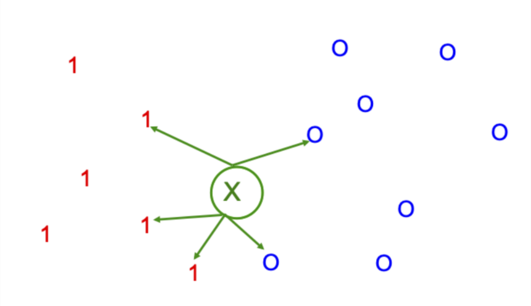
- Today, we talk about **non-parametric ML models**
  - K-Nearest Neighbours
  - Decision Tree
  - Random Forests
- They still have parameters **!**
- The **number of parameters (complexity) can grow with more training data**

K-Nearest Neighbours

- Just remember the training data $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ 😂
- Given new $\mathbf{x}$, find k *nearest* neighbours in $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$

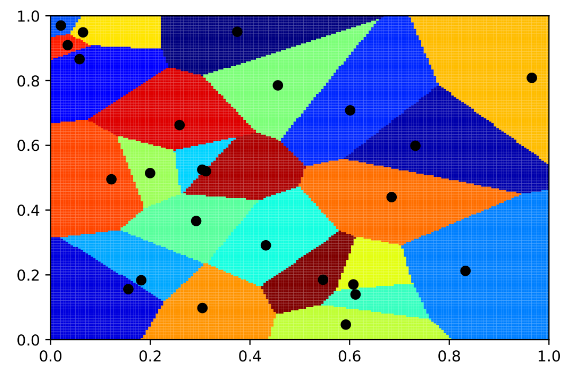


- Just remember the training data $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ 😂
- Given new $\mathbf{x}$, find k *nearest* neighbours in $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$



- **Classification:** Estimate $p(y = c|\mathbf{x}) = \frac{\text{\# of points in neighbourhood that belong to class } c}{k}$
- **Regression:** Predict (weighted) average of targets attached to the k neighbours

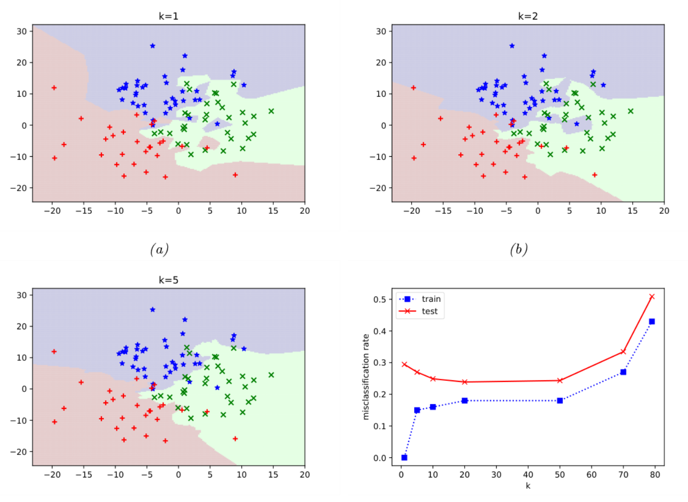
KNN WITH $k = 1$



Probabilistic Machine Learning – Chapter 16 (Kevin Murphy, 2022)

- $k = 1$ KNN induces so-called **Voronoi tesselation** of feature space
- Here, we assumed we measure distance using the **Euclidean norm** ℓ_2

Choosing k

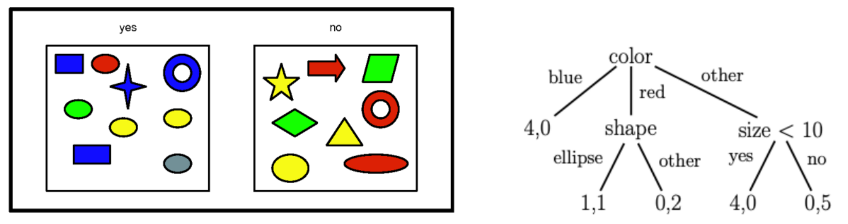


Probabilistic Machine Learning – Chapter 16 (Kevin Murphy, 2022)

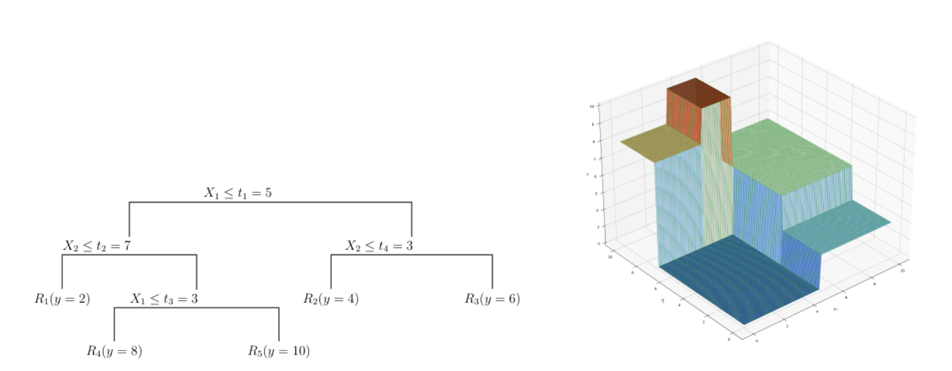
- **Very simple** predictor, **training** is easy
- Often works surprisingly well in **low dimensions**
- **Usually fails in high dimensions**
  - **Curse of dimensionality**
  - Space grows exponentially fast with number of dimensions
  - You may have to look **very** far away from some $\mathbf{x}$ to find even one neighbour...

Decision Trees

- Recursively **partition input space** by using simple **if-then rules**
- Input features can be **categorical**, **continuous** or **both**
- Can be used for **classification** and **regression**
- **Classification:** Leaf nodes predict either constant target y or distribution $p(y|\mathbf{x})$

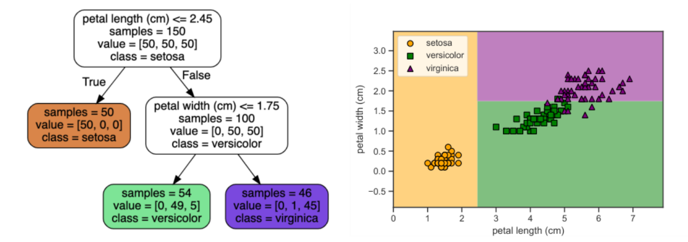


Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

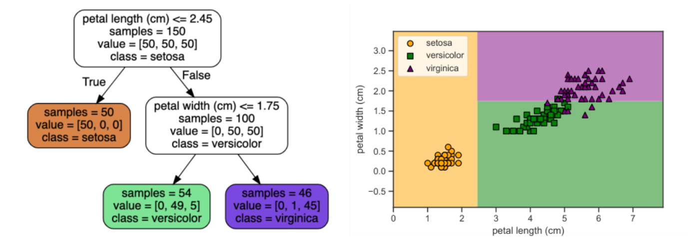


Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

- For particular region, we predict a **constant value**
- Note: **Axis-aligned partition** of input space

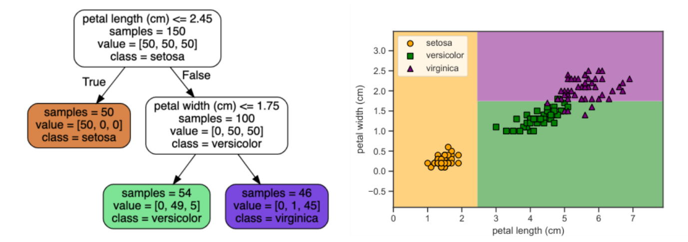


Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)



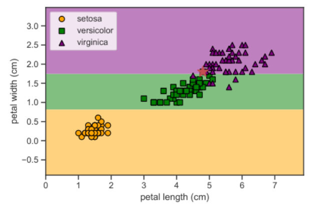
Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

- Watch what happens when we **remove a single datapoint**



Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

- Watch what happens when we **remove a single datapoint**



Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

Variance of Decision Trees

- Decision Trees seem to be **unstable w.r.t. the training data**
- **High variance models** 😩
- However, we will turn this into a **good thing** soon ☀️

Decision Trees – Pros and Cons

Pros 👍

- Easy to **interpret**
- Can handle **mixed discrete and continuous features**
- **No need to standardize** inputs
  - Makes no difference to the learning algorithm
- Perform automatic **feature selection**
- Relatively robust to **outliers**
- **Fast** to fit, **scales well** to large datasets

Cons 👎

- Predictions **not very accurate**
  - (Compared to other models)
  - **Greedy** tree construction partly responsible
- High **variance**

Ensemble Methods

- Decision Trees: High variance
- **Idea: Reduce variance by averaging multiple models** **!**

$$f(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} F_m(\mathbf{x})$$

- **Classification:**
  - Perform **majority voting**
  - Average **predicted probability mass functions**
  - (more elaborate aggregation schemes)

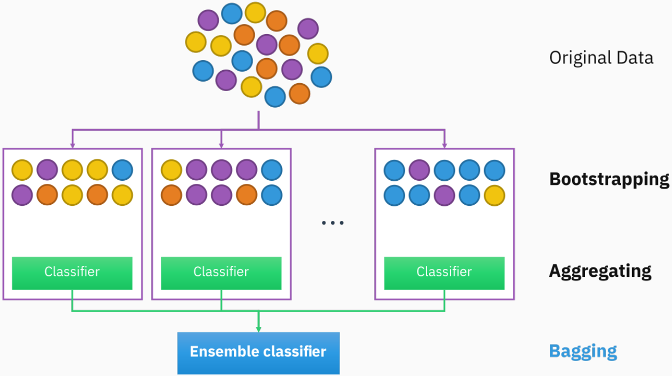
- Decision Trees: High variance
- **Idea: Reduce variance by averaging multiple models** ❗

$$f(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} F_m(\mathbf{x})$$

- **Classification:**
  - Perform **majority voting**
  - Average **predicted probability mass functions**
  - (more elaborate aggregation schemes)

- But we only have **one dataset** – how should we train **multiple models**? 🤔

Bagging



https://en.wikipedia.org/wiki/File:Ensemble_Bagging.svg

- "**Bootstrap**" new datasets and train many decision trees
- **Aggregate** trained models

- Aggregated model can't rely so much on a **single training example**
- However, learned base models are still **correlated**
- Let's try to **decorrelate them even further** 🤔

- Perform **baggging** to fit many **decision trees**
- **However,** each learned tree can only look at a **random subset of features**
- $\rightarrow$ **Random Forest Algorithm**

Pros 👍

- Usually **better predictive accuracy** than bagged decision trees
- Trees can be fit **in parallel**
  - Same for bagged decision trees

Cons 👎

- Loses **interpretability**
  - Compared to single decision tree

- Our model is

$$f_(\mathrm{x}) = \sum_{m=1}^{M} \beta_m \, F_m(\mathrm{x}; \boldsymbol{\theta}_m)$$

- Previously, we set $\beta_m = 1/M$ and fitted all F_m **independently**

- Our model is

$$f_(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \, F_m(\mathbf{x}; \boldsymbol{\theta}_m)$$

- Previously, we set $\beta_m = 1/M$ and fitted all F_m **independently**
- Given F_1, it would make sense to learn F_2 such that it **corrects the mistakes of** F_1
- That is, we want to learn f **in a stagewise fashion**
- Also, we want **to learn the weights** $\beta_1, \ldots, \beta_M$

$$f(\mathrm{x}) = \sum_{m=1}^{M} \beta_m \, F_m(\mathrm{x}; \boldsymbol{\theta}_m) \quad \text{should minimize} \quad \sum_{i=1}^{N} \ell(y^{(i)}, f(\mathrm{x}^{(i)}))$$

- At iteration m, we compute

$$(\beta_m, \boldsymbol{\theta}_m) = \operatorname*{argmin}_{\beta, \boldsymbol{\theta}} \sum_{i=1}^{N} \ell \left(y^{(i)}, \; f_{m-1}(\mathrm{x}^{(i)}) + \beta F(\mathrm{x}^{(i)}; \boldsymbol{\theta}) \right)$$

- We then set

$$f_m(\mathrm{x}) = f_{m-1}(\mathrm{x}) + \beta_m F(\mathrm{x}; \boldsymbol{\theta}_m)$$

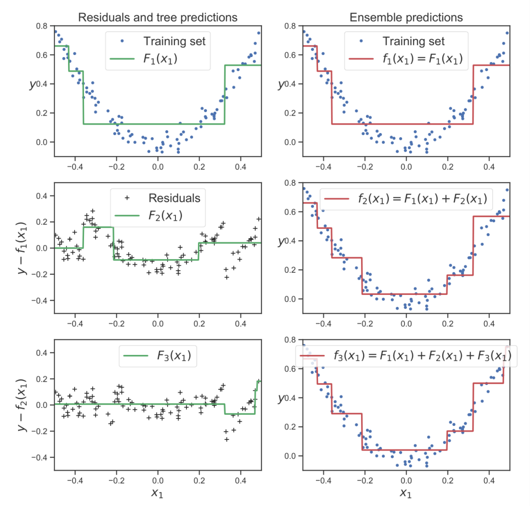
$$(\beta_m, \boldsymbol{\theta}_m) = \operatorname*{argmin}_{\beta, \boldsymbol{\theta}} \sum_{i=1}^{N} \ell\left(y^{(i)},\ f_{m-1}(\mathbf{x}^{(i)}) + \beta F(\mathbf{x}^{(i)}; \boldsymbol{\theta})\right)$$

- Let $\ell(y, \hat{y}) = (y - \hat{y})^2$
- Thus,

$$\ell\left(y^{(i)},\ f_{m-1}(\mathbf{x}^{(i)}) + \beta F(\mathbf{x}^{(i)}; \boldsymbol{\theta})\right) = (\underbrace{y^{(i)} - f_{m-1}(\mathbf{x}^{(i)})}_{\text{residual error}} - \beta F(\mathbf{x}^{(i)}; \boldsymbol{\theta}))^2$$

- We minimize this by setting $\beta = 1$ and fitting F to the **residual error** **!**

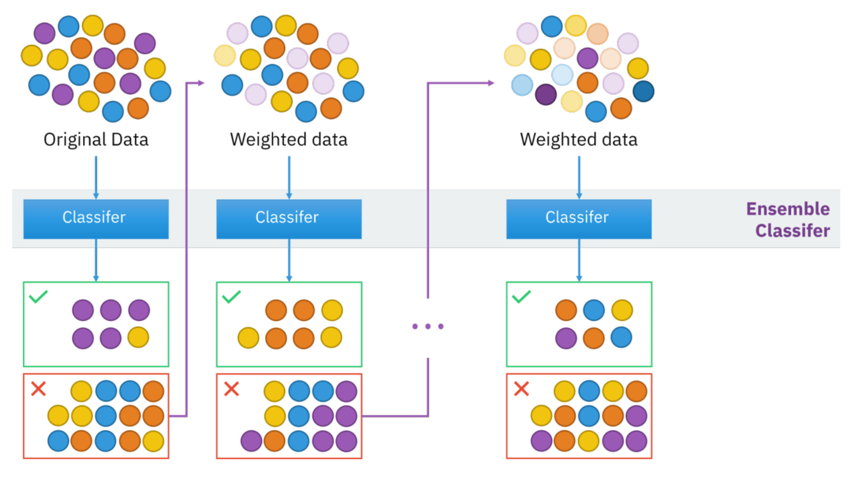
Least Squares Boosting – Example



Probabilistic Machine Learning – Chapter 18 (Kevin Murphy, 2022)

- In **least squares boosting**, we **change the dataset** by subtracting the current prediction from the true target (**residual error**)
- In **classification** settings, one can make similar calculations
- Solution: **change the dataset** by computing particular **weights** for each data sample
- Misclassified samples get **higher weights**
- **Higher weight** $\rightarrow$ **higher penalty when misclassified**

Boosting Classifiers (cont.)



- **Boosted Decision Trees work extremely well in practice** 🤯
- General framework for these methods: **Gradient Boosting**
- **Highly optimized** and flexible implementations available
  - `XGBoost` (`https://xgboost.readthedocs.io/`)
  - `CatBoost` (`https://catboost.ai/`)
  - `LightGBM` (`https://lightgbm.readthedocs.io/`)

- **Boosted Decision Trees work extremely well in practice** 🤯
- General framework for these methods: **Gradient Boosting**
- **Highly optimized** and flexible implementations available
  - `XGBoost` (`https://xgboost.readthedocs.io/`)
  - `CatBoost` (`https://catboost.ai/`)
  - `LightGBM` (`https://lightgbm.readthedocs.io/`)

TL;DR

If you do classification/regression on tabular data, try one of these libraries first ❗

Demo: KNN, Tree-based Models

Assignment 2 Questions