hhu
Heinrich Heine
Universität Düsseldorf

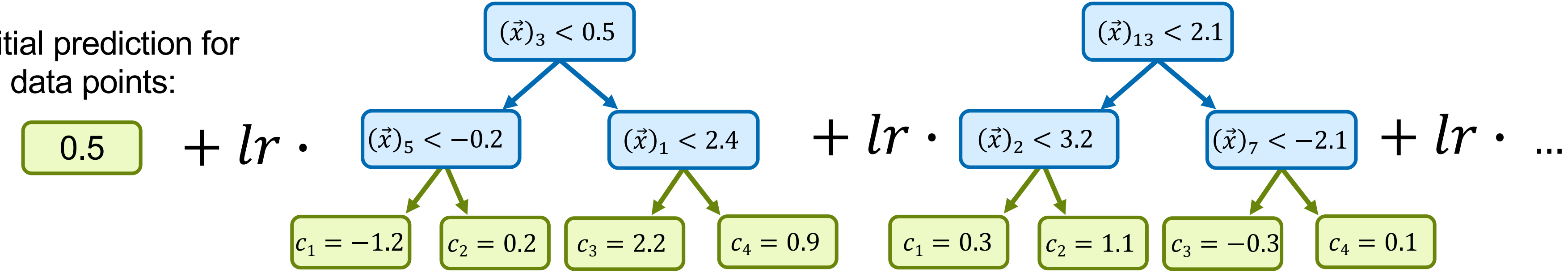
Gradient Boosting Decision Tree Models

What are Gradient Boosting Decision Tree Models?

hhu
Heinrich Heine
Universität Düsseldorf

- Let‘s assume we have input vectors $\vec{x} \in \mathbb{R}^d$. How do we get a prediction $y \in \mathbb{R}$?

Initial prediction for all data points:

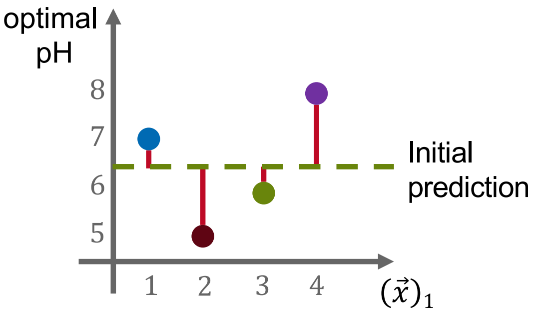


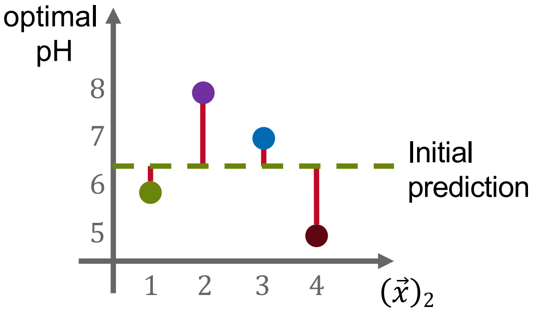
- Example:
  - learning rate $lr = 0.3$
  - We are in leave c_1 for tree 1
  - We are in leave c_3 for tree 2

$$y_{pred} = 0.5 + 0.3 \cdot (-1.2) + 0.3 \cdot (-0.3) + \ \ldots$$

Building Trees During Training

- Example training data: $\mathcal{D} = \{(\vec{x}_1,\ y_1),\ \ldots,\ (\vec{x}_n,\ y_n)\},\ \vec{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ with $n = 4, d = 2$.

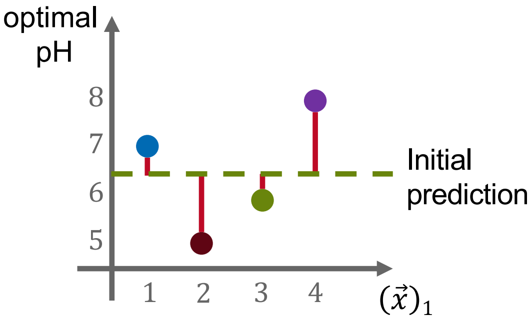


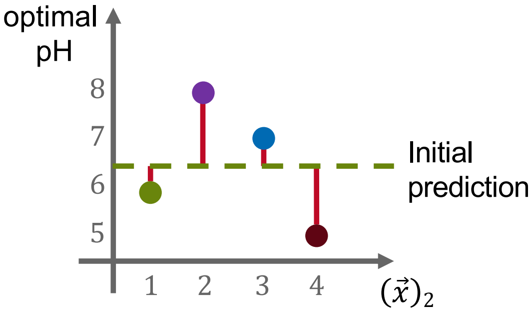


- Initial prediction for all data points: $\boxed{6.5}$
- Find decision rule $(\vec{x})_i < a$ that maximizes similarity of prediction errors e_i in two new subgroups
- Calculate similarity score for the two subgroups: $S = \frac{(\sum_i e_i)^2}{n}$
- $S_{Root} = \frac{(0.5-1.5-0.5+1.5)^2}{4} = 0$

Data point	Prediction error e_i
● (blue)	0.5
● (dark red)	-1.5
● (green)	-0.5
● (purple)	1.5

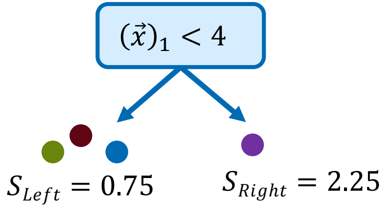
Building Trees During Training (2)





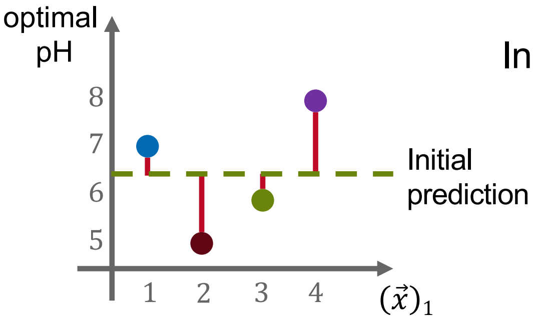
- Similarity score:
  - $S = \frac{(\sum_i e_i)^2}{n}$
  - $S_{Root} = \frac{(0.5-1.5-0.5+1.5)^2}{4} = 0$
- **Aim:** Find decision rule $(\vec{x})_i < a$ that splits the dataset in a left and a right group that maximizes the gain:
  $$Gain = S_{Left} + S_{Right} - S_{Root}$$
- We test all possible values for i and a and select the combination with the highest $Gain$

Data point	Prediction error e_i
● (blue)	0.5
● (dark red)	-1.5
● (green)	-0.5
● (purple)	1.5



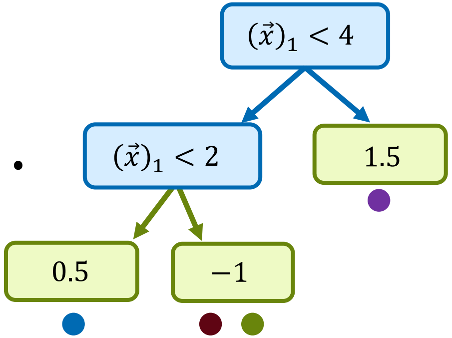
$Gain = 3$

Building Trees During Training (3)

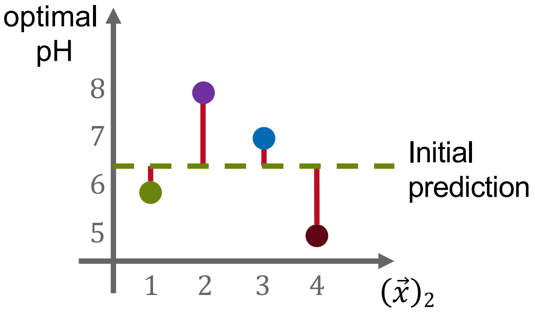


Initial prediction:

6.5 $+\ lr \cdot$



Data point	Prediction error e_i
●	0.5
●	-1.5
●	-0.5
●	1.5



For $lr = 0.5$:

Data point	New prediction	New prediction error
●	$6.5 + 0.5 \cdot 0.5 = 6.75$	0.25
●	$6.5 + 0.5 \cdot (-1) = 6.00$	-1.00
●	$6.5 + 0.5 \cdot (-1) = 6.00$	0.00
●	$6.5 + 0.5 \cdot 1.5 = 7.25$	0.75

Most important hyperparameters

- $learning\_rate$
  - Specifies how much weight the prediction of each new tree gets
  - Range: $(0,1]$
  - Default: 0.3
- reg_lambda and reg_alpha
  - Regularization coefficients that
    - penalize large predictions in the leaves
    - weaken the influence of branches that contain only few data points
  - Range: $[0, inf]$ for both
  - Default: 1 for reg_lambda and 0 for reg_alpha
- max_depth
  - Defines to maximum allowed depth for each tree.
  - Default: 6

Most important hyperparameters (2)

- $\mathrm{n\_}estimators$
  - Defines the numbers of tree that will be build in the gradient boosting model
  - Range: Any positive integer
- $\mathrm{min\_}child\_weight$
  - Tells the model when to stop splitting tree nodes when:
    - number of data points in this node is too small (regression task)
    - most data points in this node belong to one class (classification task)
  - Range: Any positive integer
  - Default: 1

- For additional hyperparameters see the XGBoost documentation: https://xgboost.readthedocs.io/en/stable/parameter.html

Gradient Boosting for binary Classification

- Gradient Boosting for binary classification tasks works very similar as for regression tasks
- Model output is a prediction score between 0 and 1:

GB model → Prediction Score

> 0.5 → classified as positive

≤ 0.5 → classified as negative

- With each tree we try to predict the prediction error:
  - $Error = True\ class\ label - Prediction\ score$

- Differences to regression prediction task:
  - Calculation of similarity scores that are required for splitting tree nodes
  - Calculating the prediction values in the leaves

Gradient Boosting for binary Classification (2)

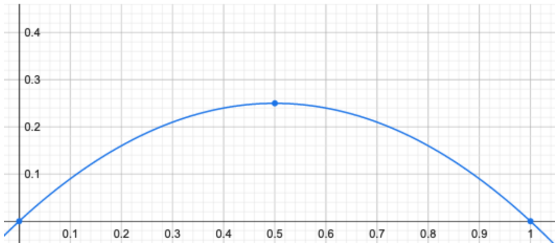
- The model predicts a $\log(odds)$ score s
  - $odds \;=\; \frac{likelihood\ of\ positive\ class}{likelihood\ of\ negative\ classs} = \frac{p}{1-p}$
  - Converting $\log(odds)$ score s to a probability score by applying the logistic function

$$sigmoid(s) = \frac{e^s}{1+e^s} = p$$

- Initial prediction: log(odds) based on the training data positive:

$$\frac{\#positive\ class\ labels}{\#\ negative\ class\ labels}$$

- With each tree we try improve the prediction error:
  - $Error \;=\; True\ class\ label - predicted\ probability$
- Computing predictions for all data points in the same leaf:



$$\frac{\sum_i Error_i}{\sum_i predicted\ probability_i \times (1 - predicted\ probability_i)}$$

Implementation with XGBoost in Python


```python
import xgboost as xgb
import numpy as np
print(xgb.__version__)
```

✓ 0.7s Python

```
2.0.3
```

```python
X = np.array([[1,3], [2,4], [3,1], [4,2]])
Y = np.array([7,5,6,8])
```

✓ 0.0s Python

```python
#fit gradient boosting model with lr = 0.3 and depth = 2 and number of trees = 2, set regularization parameters to 0
model = xgb.XGBRegressor(objective ='reg:squarederror', learning_rate = 0.5, max_depth = 2,
                         n_estimators = 2, reg_lambda = 0, reg_alpha = 0)
```

✓ 0.0s Python

```python
model.fit(X, Y)
model.predict(X)
```

✓ 0.0s Python

```
array([6.8125, 5.5   , 6.0625, 7.625 ], dtype=float32)
```

Plotting XGBoost trees


```python
from xgboost import plot_tree
import matplotlib.pyplot as plt
plot_tree(model, num_trees=0)
plot_tree(model, num_trees=1)
plt.show()
```

✓ 0.5s Python