

Lab 2

Chanwoo Kim



Seoul National University
Graduate School of Data Science

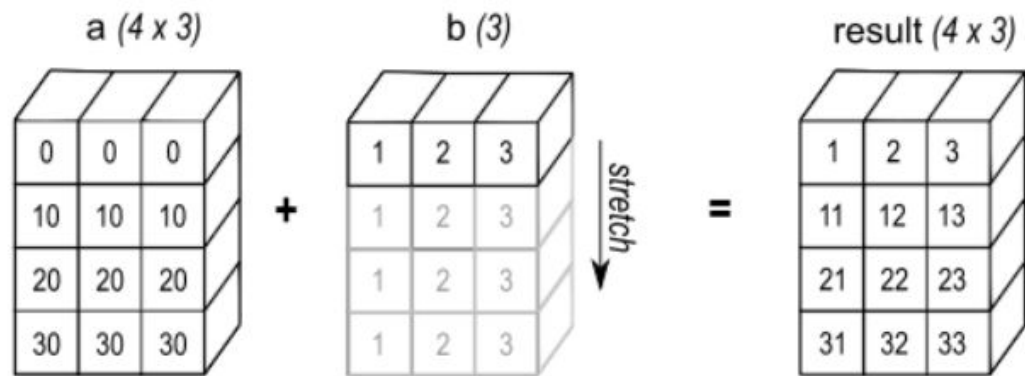


Vectorization in Numpy



Two Major Tips for Vectorization in Numpy

1. Broadcasting



2. Boolean Indexing & Fancy Indexing

```
1  a = np.array([1,2,3,4,5])
2  a[a%2==0] = 0
3  a
```

✓ 0.0s

```
array([1, 0, 3, 0, 5])
```

Regularization



Ridge Regression

- Estimation in Ridge Regression

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (\mathbf{Y} - \mathbf{X}\beta)^\top (\mathbf{Y} - \mathbf{X}\beta) + \lambda \|\beta\|_2^2$$

$$\begin{aligned} \frac{\partial \text{RSS}(\beta)}{\partial \beta} &= -\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\beta) - (\mathbf{Y} - \mathbf{X}\beta)^\top \mathbf{X} + 2\lambda\beta \\ &= -\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\beta) - \mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\beta) + 2\lambda\beta \\ &= -2\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\beta) + 2\lambda\beta = 0 \end{aligned}$$

$$\mathbf{X}^\top \mathbf{Y} - \mathbf{X}^\top \mathbf{X}\beta - \lambda\beta = 0$$

$$\beta = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{Y}$$

$d \times 1$ $n \times d$ $d \times d$ $n \times 1$

Check dimensions!

Ridge Regression

- Implement the method `fit` in the class `RidgeRegression`

Ordinary Least Squares Estimator

$$\bullet \hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y$$

Ridge Estimator

$$\bullet \hat{\beta}_R = (X^T X + \lambda I)^{-1} X^T Y$$

$d \times 1$

$n \times d$

$d \times d$

$n \times 1$

Hint:

`np.eye`

```
1  np.eye(3)
```

```
✓ 0.0s
```

```
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

Lasso Regression

- Estimation in Lasso Regression

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (\mathbf{Y} - \mathbf{X}\beta)^\top (\mathbf{Y} - \mathbf{X}\beta) + \lambda \|\beta\|_1$$

- No closed form solution.
-> Let's use gradient descent!

$$\beta^{\text{new}} = \beta^{\text{old}} - \alpha \nabla_{\beta} \mathcal{L}(\beta)$$

learning rate (pointing to α)

Gradient (pointing to $\nabla_{\beta} \mathcal{L}(\beta)$)

$$\frac{\partial}{\partial \beta} L = 2X^T(X\beta - y) + \lambda \times \text{sign}(\beta)$$

Lasso Regression

- Implement the method `fit` in the class `LassoRegression`

$$\beta^{\text{new}} = \beta^{\text{old}} - \alpha \nabla_{\beta} \mathcal{L}(\beta)$$

learning rate

Gradient

$$\frac{\partial}{\partial \beta} L = 2X^T(X\beta - y) + \lambda \times \text{sign}(\beta)$$

Hint:
`np.sign`

```
1  np.sign([-1,2,-3,4])  
✓  0.0s  
array([-1,  1, -1,  1])
```


Decision Trees



Decision Tree Classification

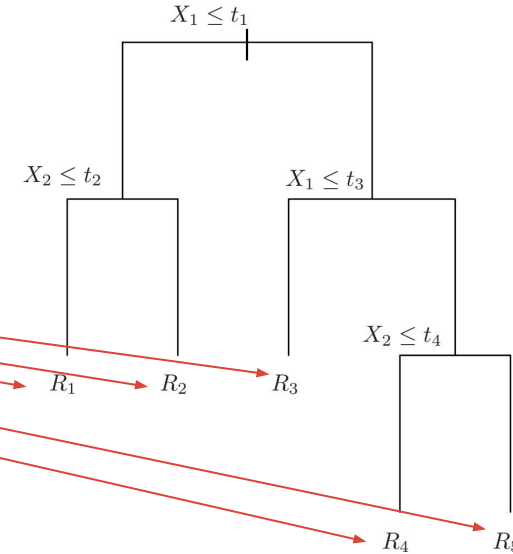
- Recursive Binary Splitting

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                      + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```

→ Three Nested For-loops

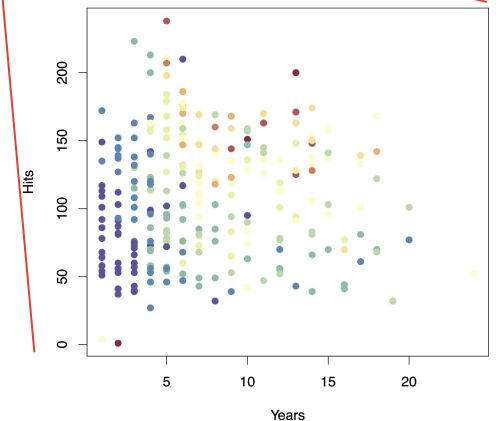
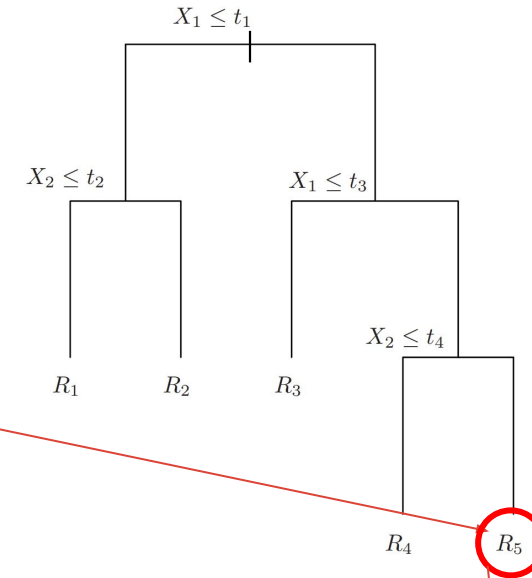
Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split  $R1(v, j, s) = v(x_j < s)$   
        split  $R2(v, j, s) = v(x_j \geq s)$   
  
        compute  $y1 = \operatorname{argmax}_k(y_i = k) \text{ for } x_i \text{ in } R1$   
        compute  $y2 = \operatorname{argmax}_k(y_i = k) \text{ for } x_i \text{ in } R2$   
  
         $\text{err}(v, j, s) = \text{impurity}(R1) / |\{x: x \in R1\}|$   
             $+ \text{impurity}(R2) / |\{x: x \in R2\}|$   
  
    pick (v, j, s) with minimum  $\text{err}(v, j, s)$   
    split node v at  $x_j = s$ 
```



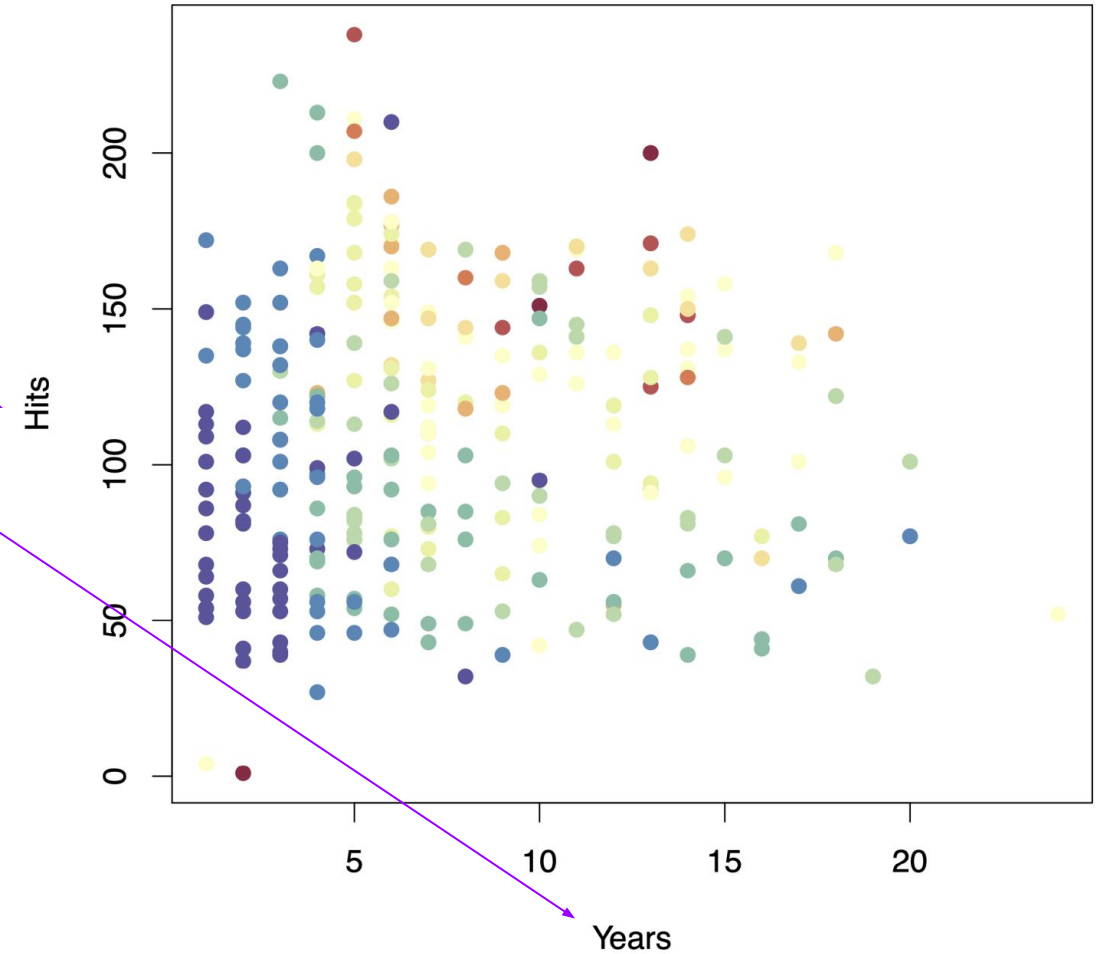
Decision Tree Classification

```
while (!stopping condition):  
  for all splittable node v, dimension j, threshold s:  
    split  $R_1(v, j, s) = v(x_j < s)$   
    split  $R_2(v, j, s) = v(x_j \geq s)$   
  
    compute  $y_1 = \operatorname{argmax}_k(y_i = k) \text{ for } x_i \text{ in } R_1$   
    compute  $y_2 = \operatorname{argmax}_k(y_i = k) \text{ for } x_i \text{ in } R_2$   
  
     $\text{err}(v, j, s) = \text{impurity}(R_1) / |\{x: x \in R_1\}|$   
                  +  $\text{impurity}(R_2) / |\{x: x \in R_2\}|$   
  
  pick  $(v, j, s)$  with minimum  $\text{err}(v, j, s)$   
  split node v at  $x_j = s$ 
```



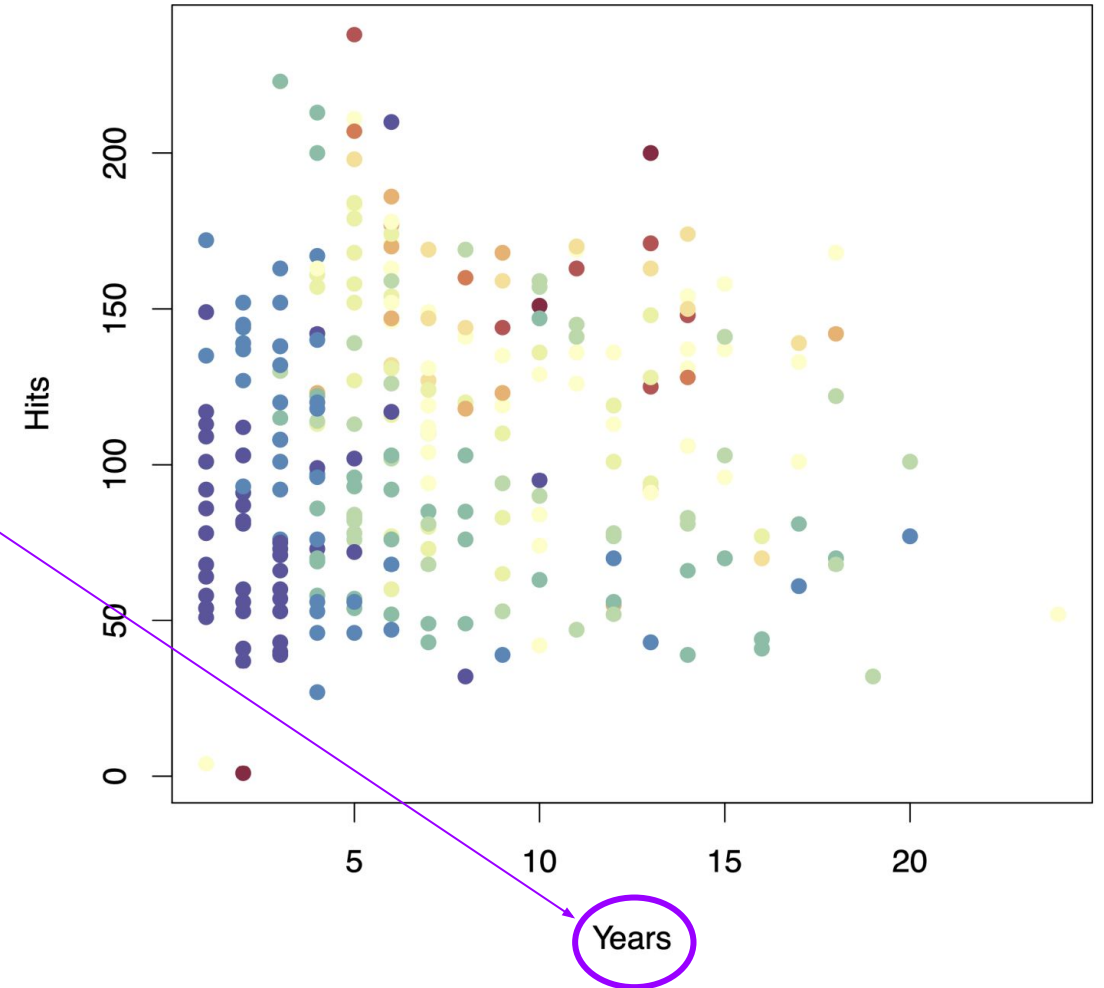
Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                    + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```



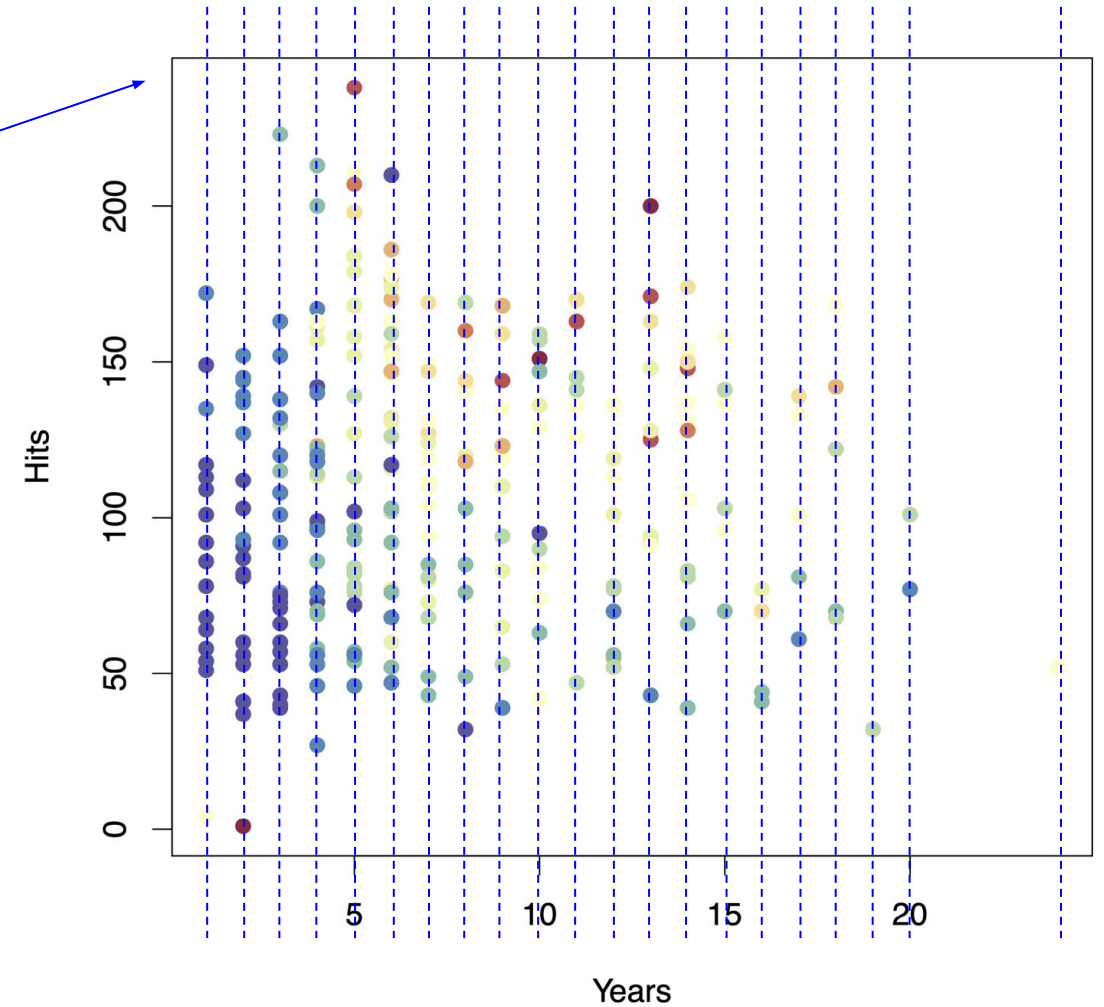
Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                    + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```



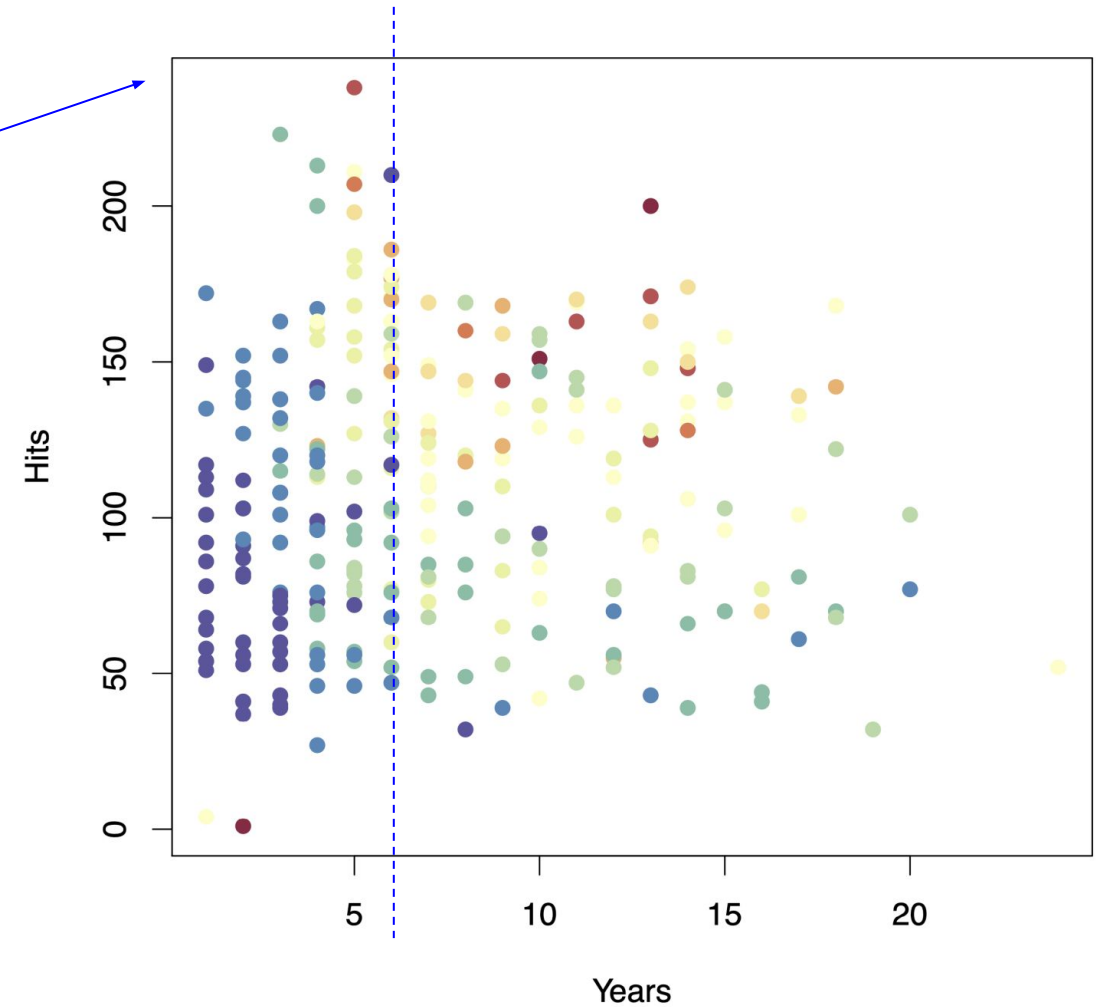
Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                    + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```



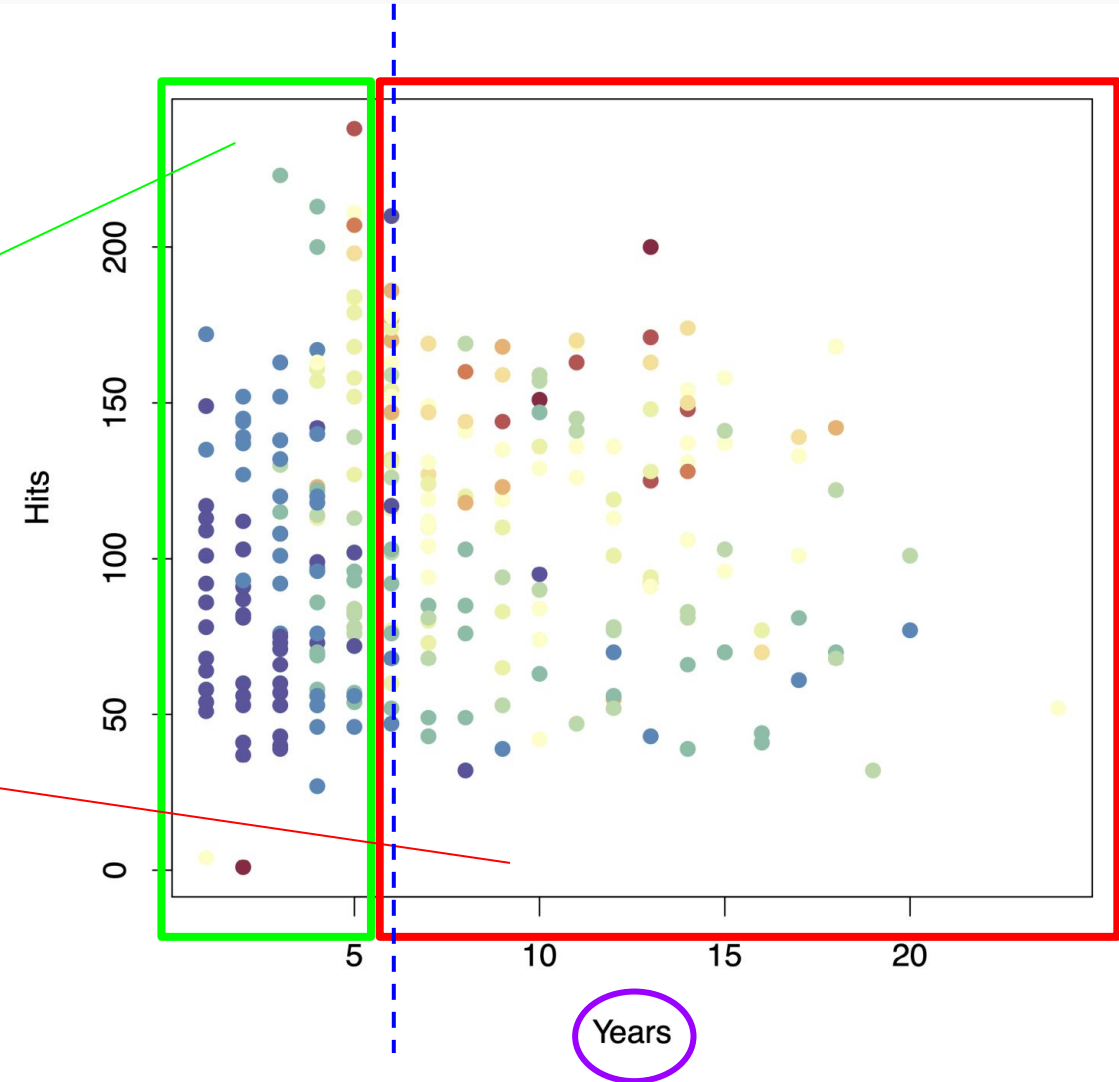
Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                    + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```



Decision Tree Classification

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold  
    s:  
        split R1(v, j, s) = v(x_j < s)  
        split R2(v, j, s) = v(x_j ≥ s)  
  
        compute y1 = argmax_k(y_i = k) for x_i in R1  
        compute y2 = argmax_k(y_i = k) for x_i in R2  
  
        err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                    + impurity(R2)/|{x: x ∈ R2}|  
  
    pick (v, j, s) with minimum err(v, j, s)  
    split node v at x_j = s
```



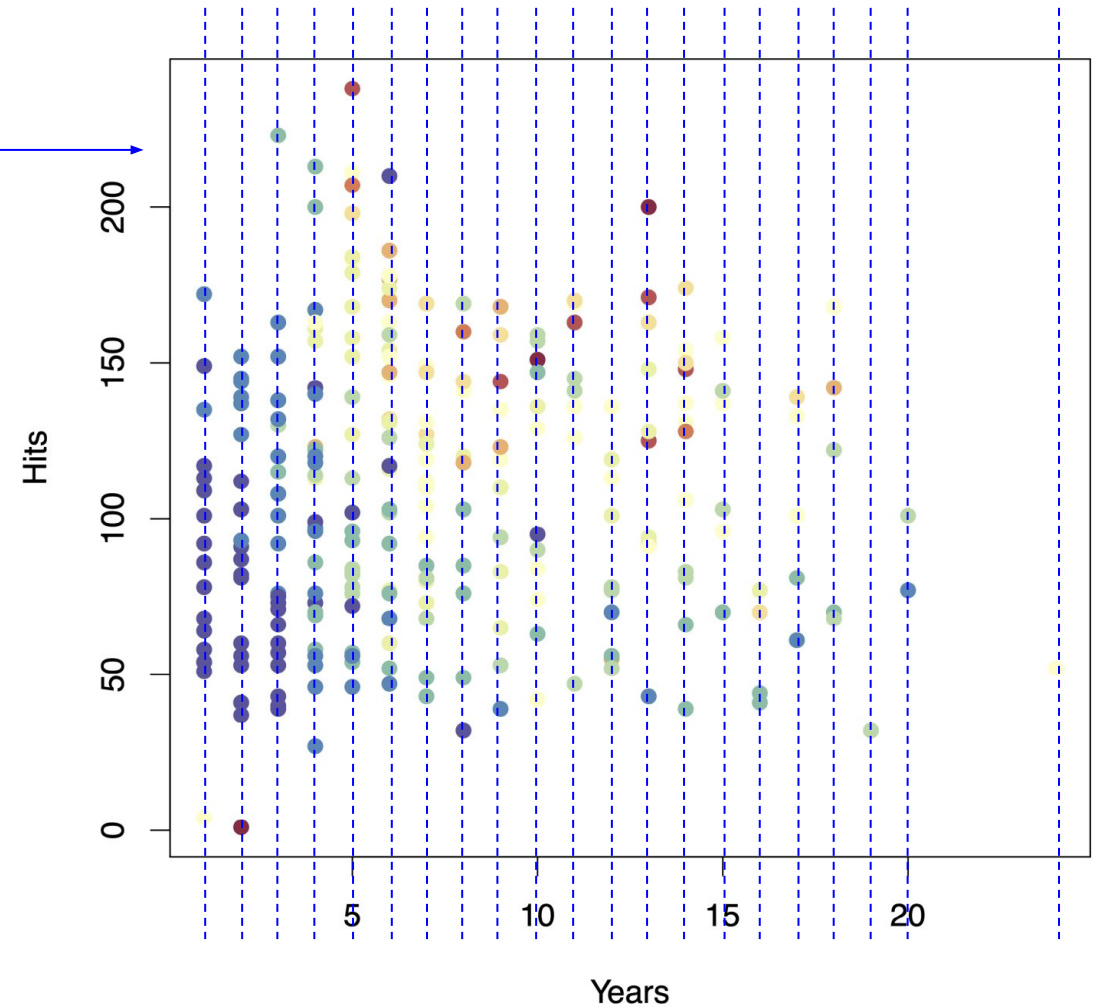
Decision Tree Classification

- Complete the method `_best_split` in the class `DecisionTree`

```
while (!stopping condition):  
    for all splittable node v, dimension j, threshold s:  
        ...
```

Hint:
`np.unique`

```
1 np.unique([1, 2, 1, 2, 3, 1])  
✓ 0.0s  
array([1, 2, 3])
```

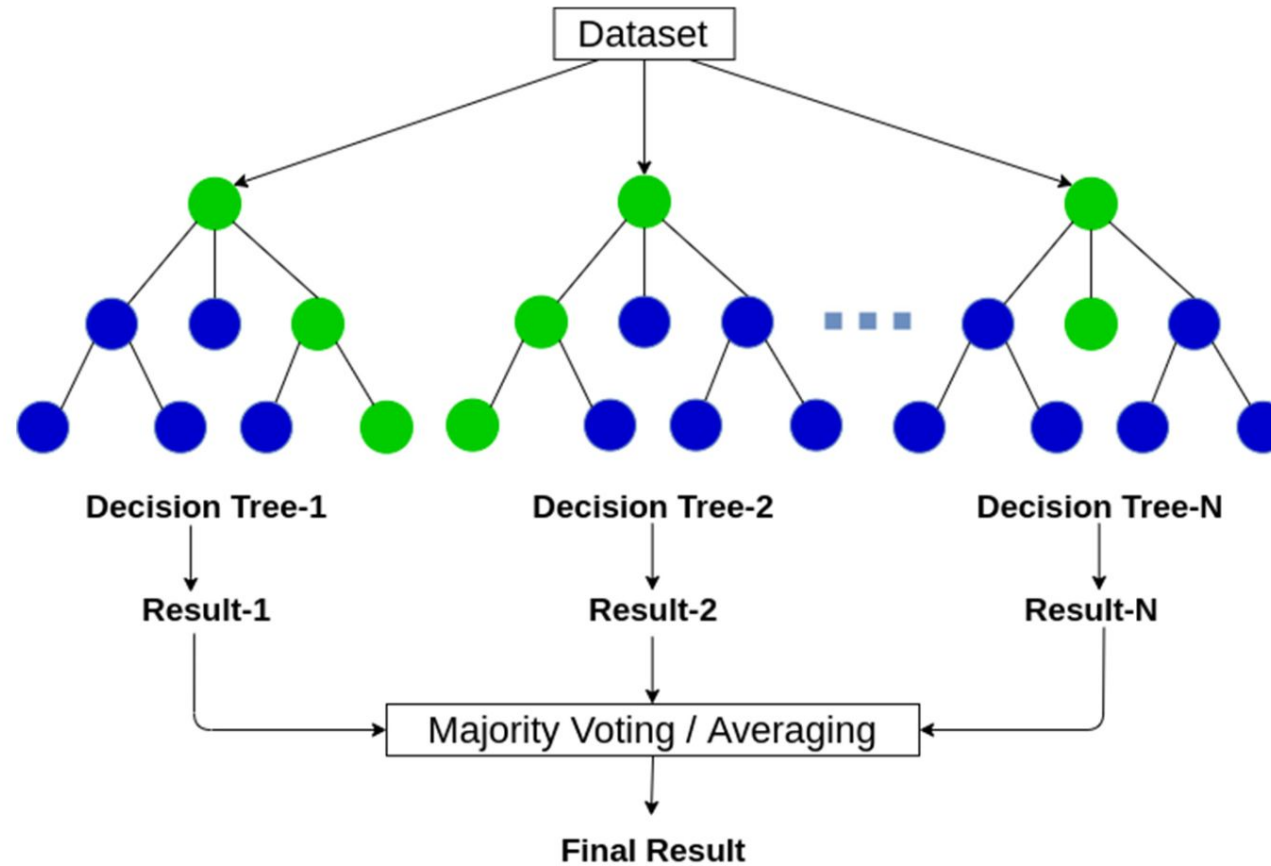


Ensemble Methods and Boosting



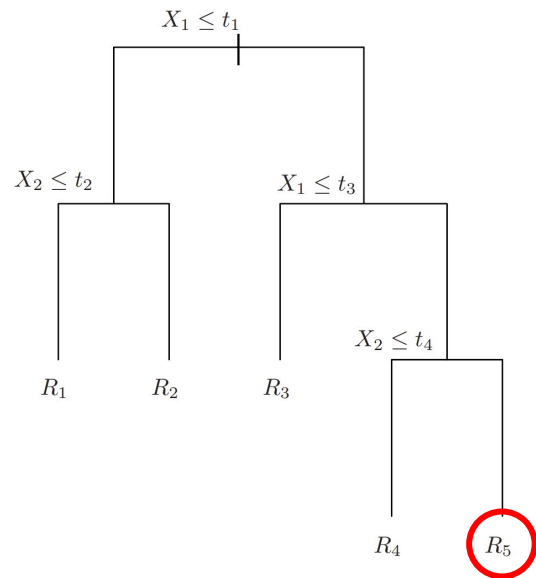
Random Forest Classification

- Bagged Trees



Random Forest Classification

- A split in a tree of Random Forest
- **Random forests** provide an improvement over bagged trees with a small tweak that **decorrelates** the trees.
 - At each time to split in training decision trees, only a subset of m predictors are considered as candidates, from the full set of p predictors.
 - A typical value of $m \approx \sqrt{p}$



To split this node,

$X_1, X_2, X_3, X_4, X_5, \dots, X_p$



Randomly choose
 m predictors!

X_2, X_5, \dots, X_{p-3}

Random Forest Classification

- A split in a tree of Random Forest

```
while (!stopping condition):  
  for all splittable node v, dimension j, threshold s:  
    split R1(v, j, s) = v(x_j < s)  
    split R2(v, j, s) = v(x_j ≥ s)  
  
    compute y1 = argmax_k(y_i = k) for x_i in R1  
    compute y2 = argmax_k(y_i = k) for x_i in R2  
  
    err(v, j, s) = impurity(R1)/|{x: x ∈ R1}|  
                  + impurity(R2)/|{x: x ∈ R2}|  
  
  pick (v, j, s) with minimum err(v, j, s)  
  split node v at x_j = s
```

$X_1, X_2, X_3, X_4, X_5, \dots, X_p$



Randomly choose
m predictors!

X_2, X_5, \dots, X_{p-3}

Random Forest Classification

- Complete the method `_best_split_with_random` in the class `DecisionTreeForRF`

$X_1, X_2, X_3, X_4, X_5, \dots, X_p$

To split this node,



Randomly choose
 \sqrt{p} predictors!

X_2, X_5, \dots, X_{p-3}

Hint:

`np.random.choice`

```
1 print(np.random.choice(5, size=2, replace=False))
2 print(np.random.choice(5, size=2, replace=False))
3 print(np.random.choice(5, size=2, replace=False))
```

✓ 0.0s

```
[0 2]
[2 1]
[1 3]
```

Ensemble Methods and Boosting



AdaBoost

- Training AdaBoost

Input: training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}$ for $i = 1, \dots, n, \mathbf{y} \in \{-1, +1\}$

- Initialize $D_1(i) = 1/n$ ← D_t : Sampling weight distribution over training samples at step t .
→ We start with **uniform distribution**.
- for $t = 1, \dots, T$: ← We train **T base learners** sequentially.
 - Train a base learner, $h_t: \mathbf{x} \rightarrow \{+1, -1\}$, with D_t ← Each base learner is trained on a training set **sampled with D_t** .
 - $\epsilon_t = p_{D_t}[h_t(\mathbf{x}_i) \neq \mathbf{y}_i]$ ← ϵ_t is the **error rate** under the current sample distribution D_t .
 - $\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$ ← α_t is **log (success rate / error rate)**, going to $-\infty$ if $\epsilon_t \rightarrow 1$, to $+\infty$ if $\epsilon_t \rightarrow 0$, and being zero if success rate = 50%.
 - $\forall i D_{t+1}(i) = D_t(i) e^{-\alpha_t \mathbf{y}_i h_t(\mathbf{x}_i)}$ ← Data distribution update for the next step:
 $\mathbf{y}_i h_t(\mathbf{x}_i) > 0$ if h_t is correct for \mathbf{x}_i , and $\mathbf{y}_i h_t(\mathbf{x}_i) < 0$ if incorrect.
For **correct** examples, weight D_{t+1} **gets smaller** than D_t .
For **incorrect** ones, weights D_{t+1} **gets larger** than D_t .
 - $\forall i D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_{j=1}^n D_{t+1}(j)}$ ← Then, **normalize D_{t+1}** so that it sums to 1.
- Output $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$ ← Final classifier is a **weighted sum of base learners**, weighted by their log success rate (α_t).

AdaBoost

- Complete the method `fit` in the class `AdaBoost`

Input: training dataset $\{\mathbf{x}_i, \mathbf{y}_i\}$ for $i = 1, \dots, n$, $\mathbf{y} \in \{-1, +1\}$

- Initialize $D_1(i) = 1/n$
- for $t = 1, \dots, T$:
 - Train a base learner, $h_t: \mathbf{x} \rightarrow \{+1, -1\}$, with D_t
 - $\epsilon_t = p_{D_t}[h_t(\mathbf{x}_i) \neq \mathbf{y}_i]$
 - $\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$
 - $\forall i D_{t+1}(i) = D_t(i) e^{-\alpha_t \mathbf{y}_i h_t(\mathbf{x}_i)}$
 - $\forall i D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_{j=1}^n D_{t+1}(j)}$

Sampling Weight Update

Hint:

```
np.exp  
np.sum
```

- Output $H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

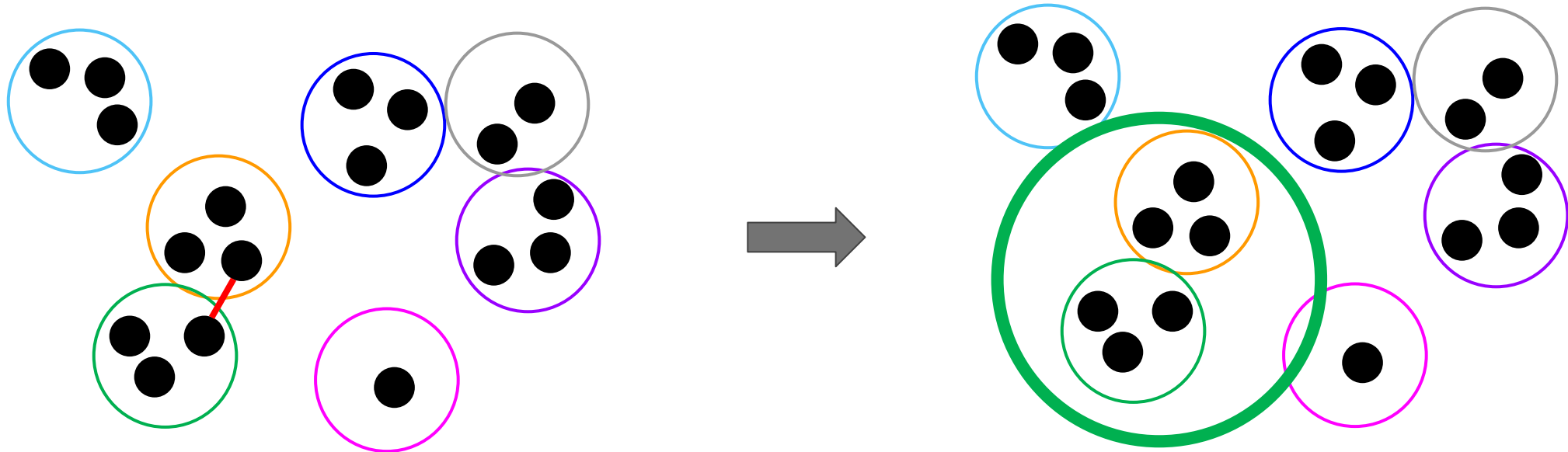
Clustering



Hierarchical Clustering

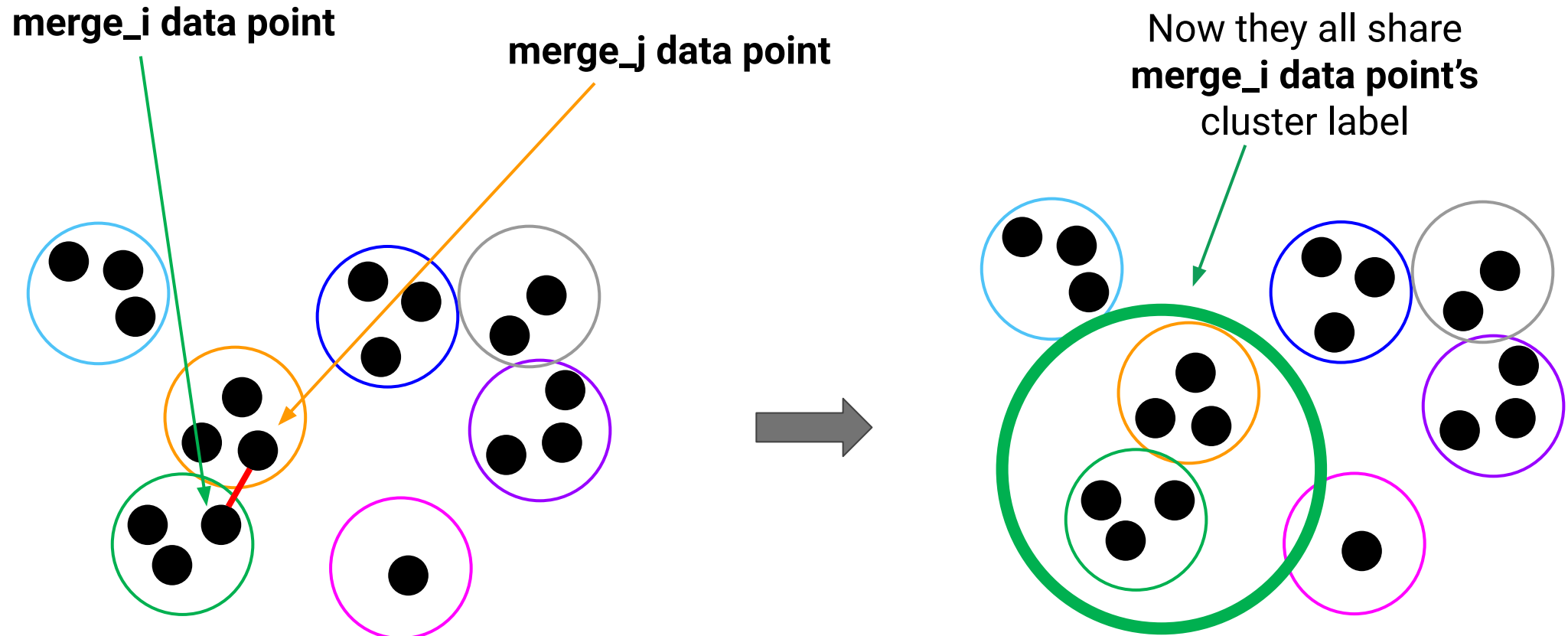
- Agglomerative Clustering

- Level 0: Start from singleton clusters (each example is a cluster).
- while there remain only N clusters:
 - Calculate the distances between all the data pairs from different cluster
 - **Find the closest data pair and merge their clusters into one cluster**



Hierarchical Clustering

- Complete the method `fit` in the class `HierarchicalClustering`



Dimension Reduction



Principal Component Analysis

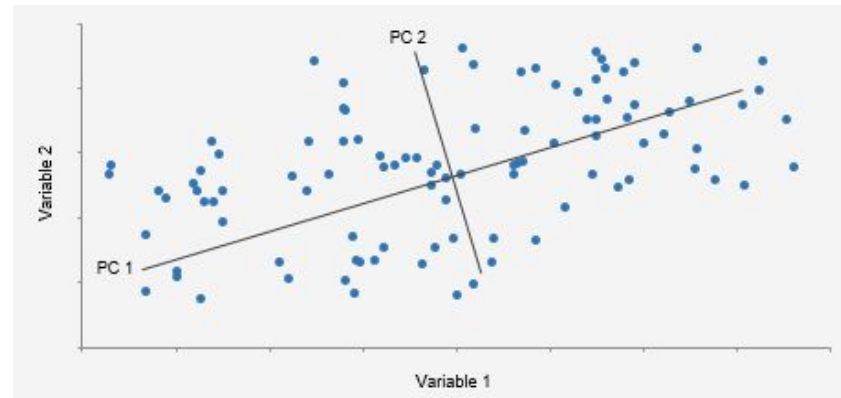
- PCA Algorithm

Step 1. Zero-center (+ and normalize) the data

Step 2. Estimate the covariance matrix

Step 3. Perform the **eigenvalue decomposition** of Σ . Then, order them by eigenvalues in decreasing order: $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$

Step 4. If we choose the first $k \ll d$ eigenvectors (with largest k eigenvalues) and discard the rest, we get a k -dimensional space such that the original data loses least amount of information (in terms of variance).



Homework 2



Homework 2

- **Due:** 4/30 Tue 6pm
 - No late penalty until 6:30 pm
- **Comments from HW1**
 - **Optimize your code** as much as you can.
 - Submit the code **with your output** and **without unnecessary output**.
 - Double check your submission.
- We **DO NOT** guarantee to run **unreasonably inefficient codes** for grading.



K-Means Clustering



Image Compression



Image Compression

- Image shape = (Height, Width, Channel)

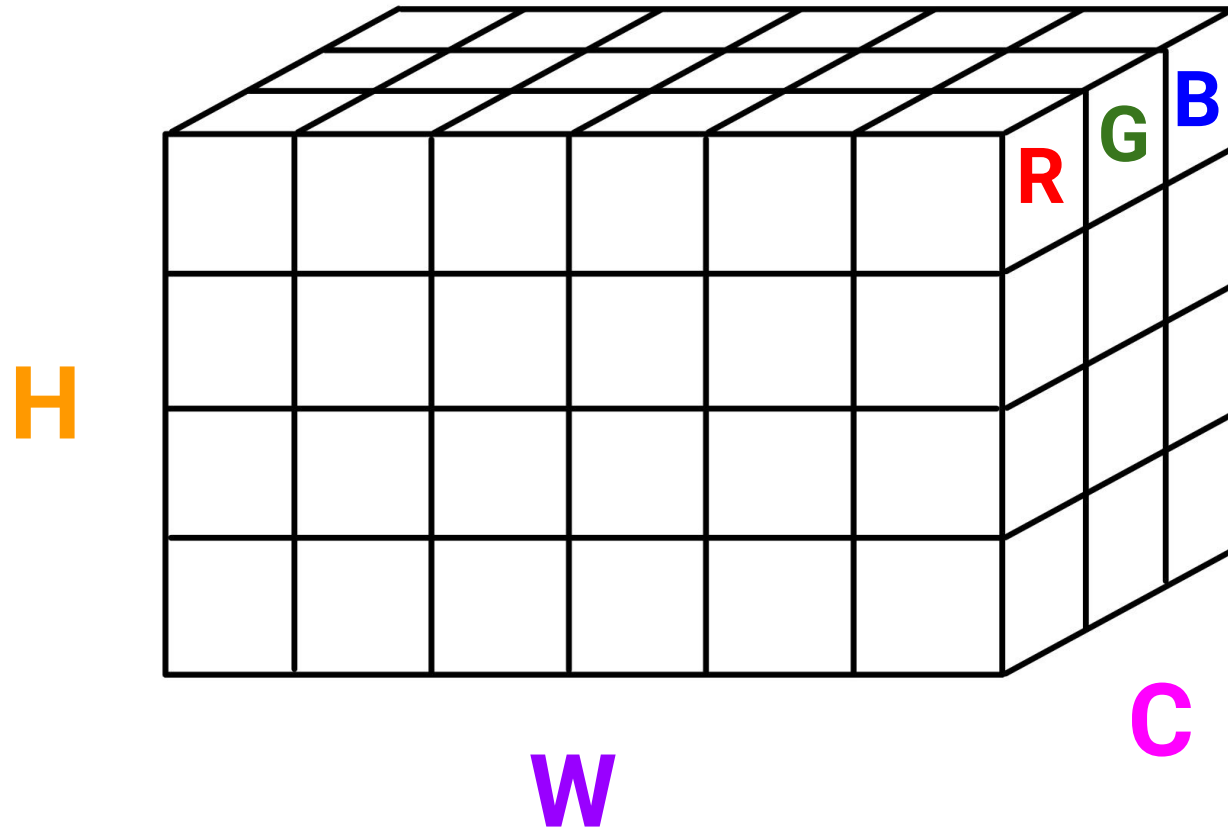
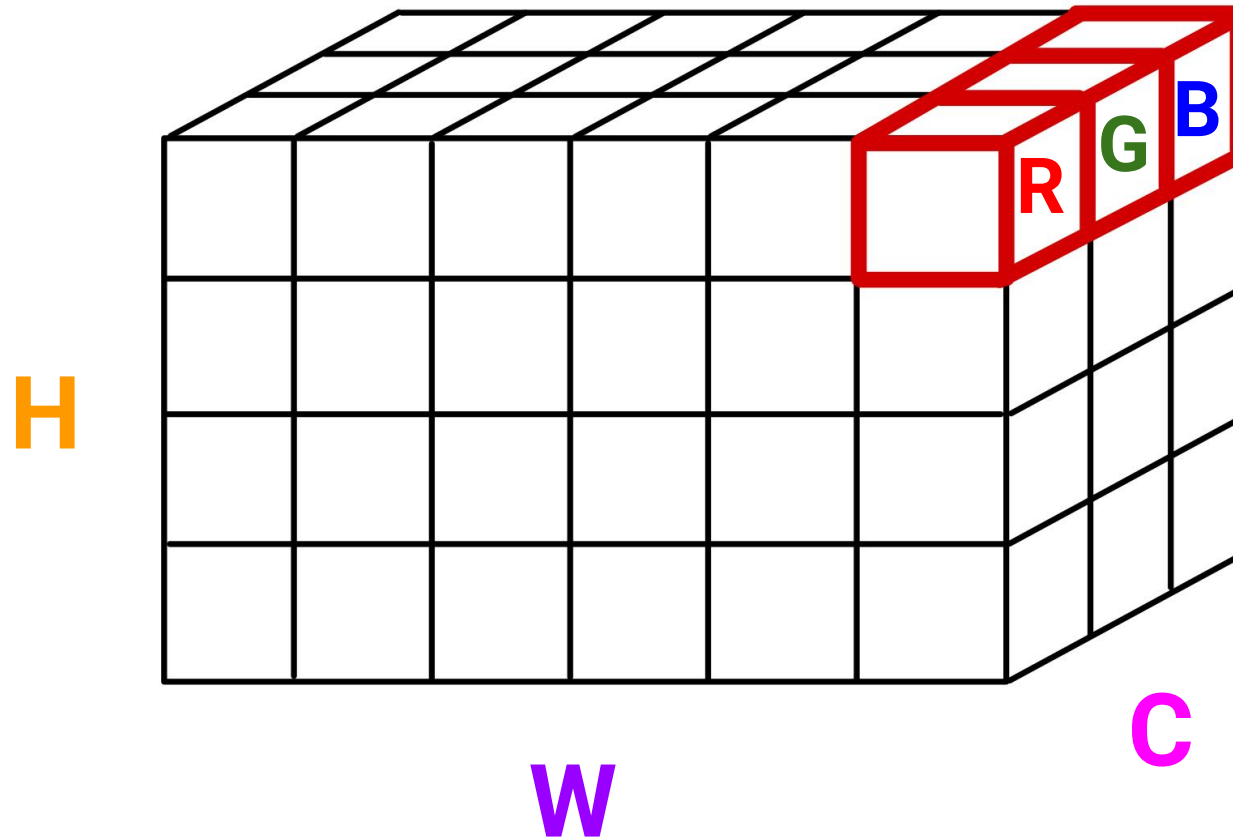


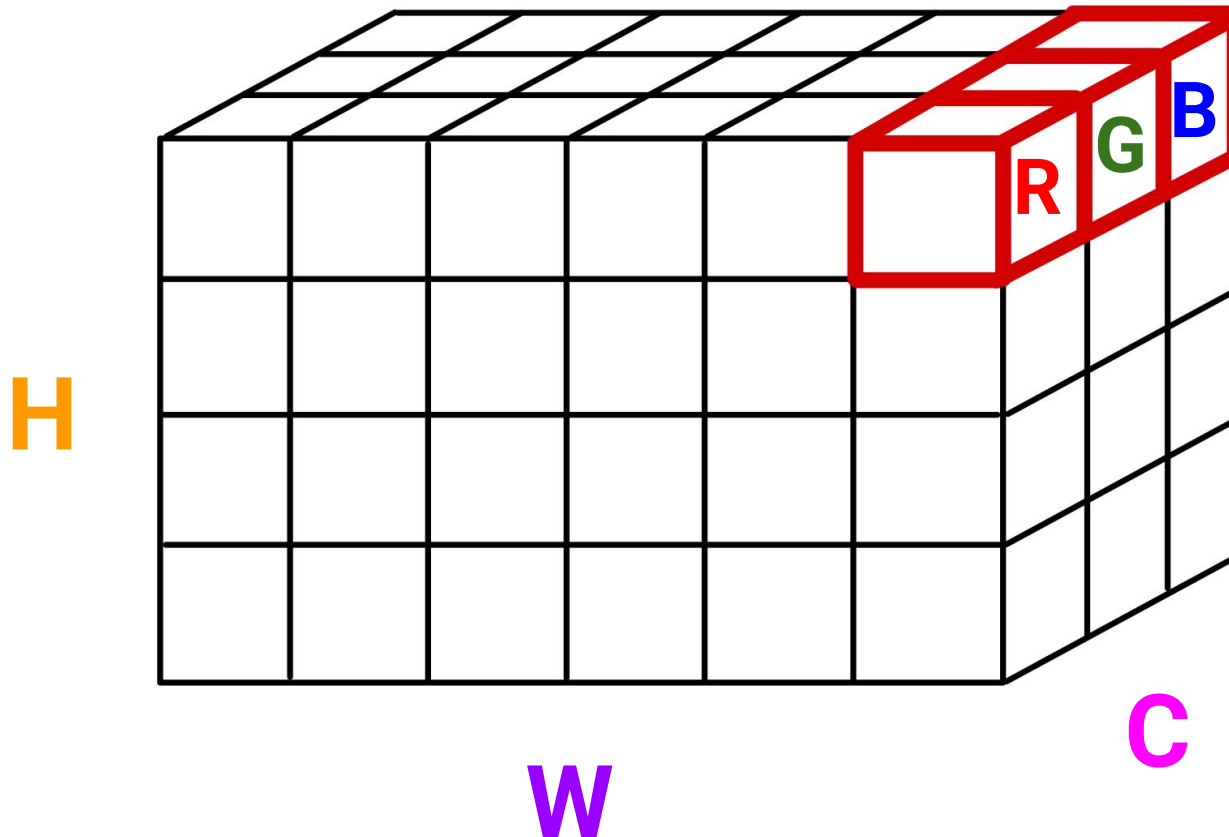
Image Compression

- One pixel value = (R, G, B)



K-Means Clustering Algorithm

1. Randomly initialize the centroids (number of centroids = K)
2. **Assign each pixel to the closest centroid**
3. **Calculate the mean of each pixel in the same cluster and update the centroid**
4. **Repeat updating with max_iter times**



K-Means Clustering Algorithm

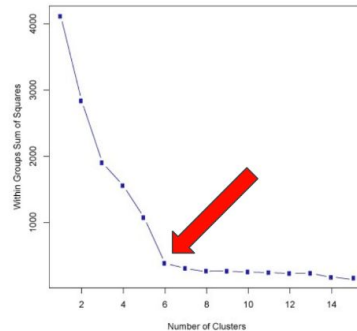
- Decide your K according to the value of J

K-Means

- How to decide K ?
 - Usually, we do not know natural distribution of the data.
 - What about choosing the K resulting in minimum value of J ?

$$J = \sum_{i=1}^N \sum_{k=1}^K r_{i,k} \|x_i - \mu_k\|^2$$

- Unfortunately, J decreases always with larger K .



- If there are natural clusters in the data, you may observe an **elbow point**. Choose it in that case.
- Life is not that easy. You may not observe such a point...

Classification



Classification

- Classifiers we have learned:
 - discriminant analysis, logistic regression, SVM, Naive Bayes, Random Forest ...
- Datasets: (1)diabetes.csv, (2)credit_score.csv
 - For each dataset, implement and train classifier models among the above.
(scikit-learn allowed)

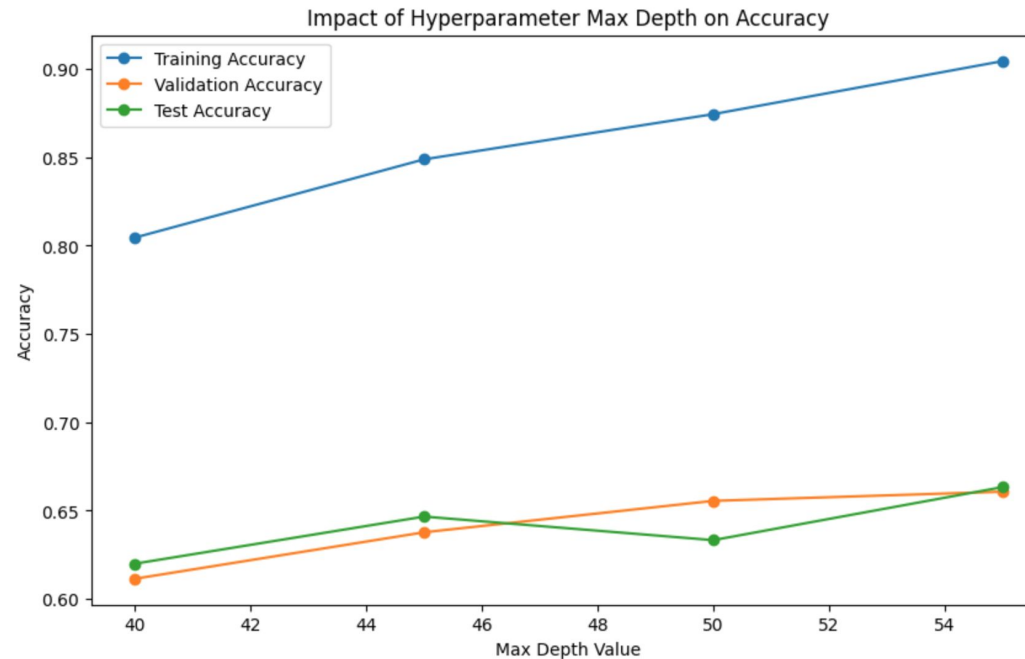
Classification

- (a), (c): Train *at least* 2 models for each dataset. Write the code and report the performances of the models.

Make sure your **codes are all run** and the evaluated **model performance is printed out!**

Classification

- (b), (d): Report the **best model** among the models you've trained.
 - What you should include:
 - **Comparison** of the models you trained in (a)/(c).
 - **Justification** on why you chose the final model.
 - Explanation on the **optimizations** you applied to your model to get the best performance.
 - e.g., regularization, hyperparameter tuning, cross validation,...
 - You will get full credit if you **plot** the process of model optimization. (example below)



Classification

- (e): Discuss the **differences or commonalities between the two datasets** with respect to *why you chose the best model for each dataset*.
 - If you chose the same model for the two datasets, why?
 - If you chose different models for the two datasets, why?