Exam 2 Review

Cathy Poliak
cpoliak@central.uh.edu

Department of Mathematics
University of Houston

MATH 4322

Exam structure.

- First three problems will present you with a data example and ask you an array of modeling/interpretation questions about that data. (Short answer questions)
- Five (5) multiple choice questions. No partial credit for the muliple choice questions.
- 90 minutes
- May use one-page notes front/back can be typed if desired.
- Can type in the boxes or write up the answers on a separate paper and upload no later than 15 minutes after you submitted the test.

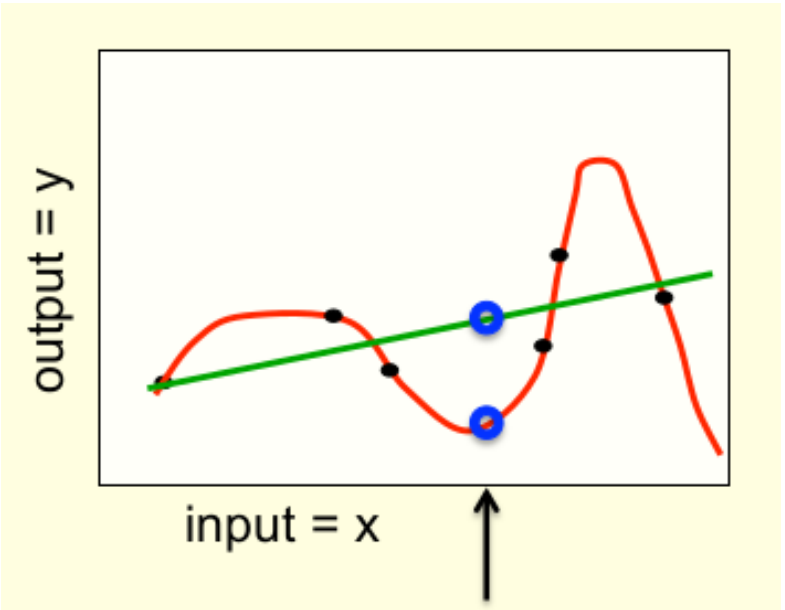
Topics Covered

- Types of statistical learning
- Cross-Validation
- Leave-One-Out Cross-Validation
- K-Fold Cross Validation
- Bootstrap Methods
- Regression Tree
- Classification Tree
- Bagging
- Random Forests
- Boosting
- Single Layer Linear Neural Network

Training error, overfitting.

Training error - not a good metric of model performance. (Why?)

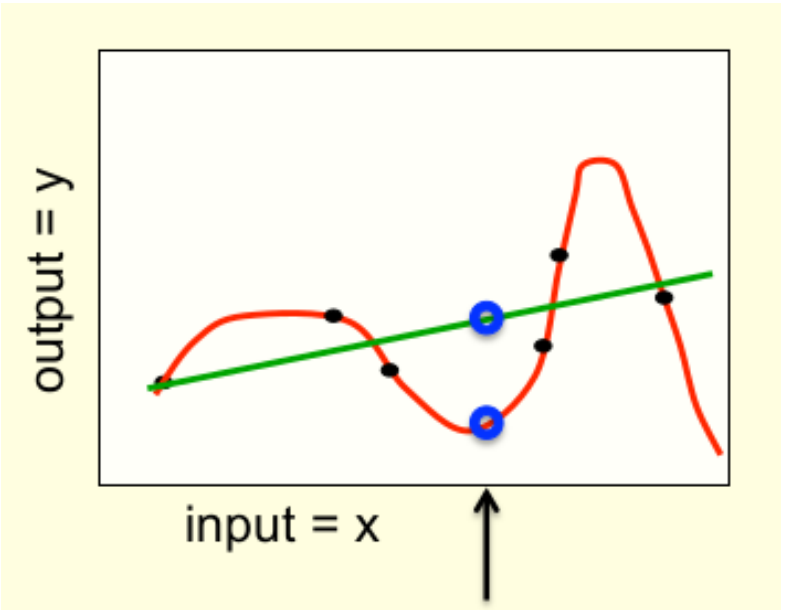
Sometimes good training test performance is more indicative of overfitting ($\Longrightarrow$ fitting the noise instead of true signal) rather than of a good generalizable model.



Training error, overfitting.

Training error - not a good metric of model performance. (Why?)

Sometimes good training test performance is more indicative of overfitting ($\Longrightarrow$ fitting the noise instead of true signal) rather than of a good generalizable model.



Instead, we need to estimate **test** error (out-of-sample error). This class covered two ways to do it:

- Validation Set Approach,
- K-fold Cross-Validation

I excluded Leave-One-Out CV (LOOCV) as it's just a special case of K-fold CV. LOOCV $K = n$ # of observation

Validation Set Approach.

One way is **validation** (or **hold-out**) **set** approach:

1. Randomly divide data set into two parts:
   - Training set
   - Validation set
2. Fit the model on training data, and use the fitted model to predict responses $\hat{y}_i$ for validation data.
3. Calculate validation set error rate:

$$\sum_{i \in \{\text{validation set}\}} (\hat{y}_i - y_i)^2 \approx \text{test error rate}$$

regression

Classification: confusion matrix

test error rate = % of misclassification

mean(observed != Predicted)

Validation Set Approach.

One way is **validation** (or **hold-out**) **set** approach:

1. Randomly divide data set into two parts:
   - Training set
   - Validation set
2. Fit the model on training data, and use the fitted model to predict responses $\hat{y}_i$ for validation data.
3. Calculate validation set error rate:

$$\sum_{i \in \{\text{validation set}\}} (\hat{y}_i - y_i)^2 \approx \text{test error rate}$$

Validation set approach has two big drawbacks:

1. Test error estimates vary heavily across different splits.

	Split #1	Split #2	Split #3
CV.err	26.14142	21.7621	22.50892

2. Only a moderate subset of data used for training (less data $\implies$ worse performance $\implies$ overestimated MSE).

K-fold Cross-Validation.

A more effective alternative is **K-fold cross-validation (K-fold CV)**

1. Data is randomly divided into K subsets of $\approx$ same size n_K.
2. For **each subset** j, $j = 1, \ldots, K$, we
   - use it as a validation set, while
   - using other $K - 1$ subsets to train the model
   - Calculate $MSE_j = \frac{1}{n_k} \sum_{i \in \{validation\ set\}} (y_i - \hat{y}_i)^2$.
3. The K-fold CV (squared) test error estimate is

$$CV_{(k)} = \frac{1}{K} \sum_{j=1}^{K} MSE_j$$

For K-fold CV:

- We use more data for training compared to validation set method.
- Test error estimates are much more stable across splits:

```
          Split #1     Split #2     Split #3
CV.err    24.11699     24.35676     24.25012
```

K-fold Cross-Validation.

Illustration of random data subdivision into training and testing subsets for 5-fold CV (as opposed to LOOCV and validation set approaches):

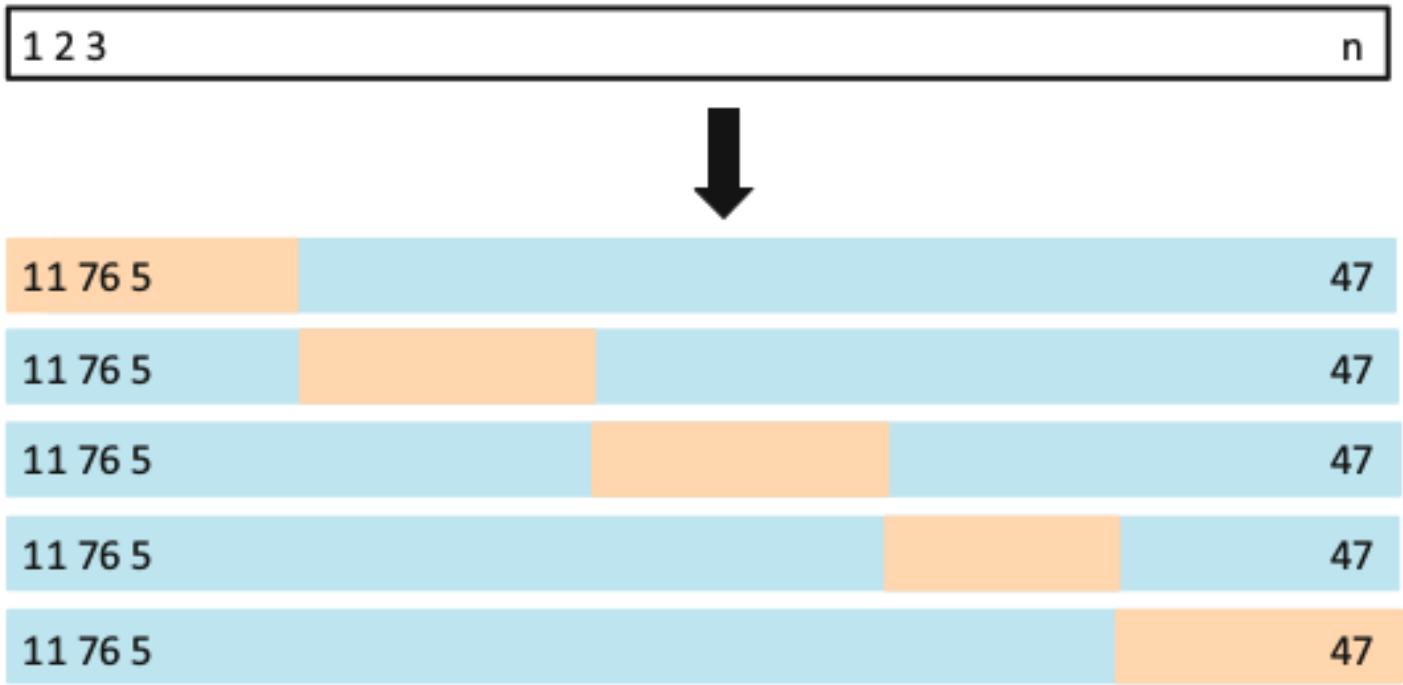


FIGURE 5.5. *A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.*

Resampling Methods: Bootstrap.

Bootstrap is a universal resampling method used to evaluate uncertainty around sample estimates for population parameter.
In particular, it helps evaluate **standard error** of an estimate.

Resampling Methods: Bootstrap.

Bootstrap is a universal resampling method used to evaluate uncertainty around sample estimates for population parameter.
In particular, it helps evaluate **standard error** of an estimate.

Presume you have a

- sample $\mathbf{x} = (x_1, \ldots, x_n)$ of n observations (e.g. *LSAT* scores),
- a sample statistic $\hat{\alpha}$ of interest (e.g. sample mean $\hat{\alpha}(\mathbf{x}) = \bar{\mathbf{x}}$)

Resampling Methods: Bootstrap.

Bootstrap is a universal resampling method used to evaluate uncertainty around sample estimates for population parameter.
In particular, it helps evaluate **standard error** of an estimate.

Presume you have a

- sample $\mathbf{x} = (x_1, \ldots, x_n)$ of n observations (e.g. *LSAT* scores),
- a sample statistic $\hat{\alpha}$ of interest (e.g. sample mean $\hat{\alpha}(\mathbf{x}) = \bar{\mathbf{x}}$)

MAIN IDEA of BOOTSTRAP (for standard error calculation):

1. Randomly resample n observations from **x with replacement** (see next slide for illustration), to get $\mathbf{z}^* = (x_1^*, \ldots, x_n^*)$,

Resampling Methods: Bootstrap.

Bootstrap is a universal resampling method used to evaluate uncertainty around sample estimates for population parameter.
In particular, it helps evaluate **standard error** of an estimate.

Presume you have a

- sample $\mathbf{x} = (x_1, \ldots, x_n)$ of n observations (e.g. *LSAT* scores),
- a sample statistic $\hat{\alpha}$ of interest (e.g. sample mean $\hat{\alpha}(\mathbf{x}) = \bar{\mathbf{x}}$)

MAIN IDEA of BOOTSTRAP (for standard error calculation):

1. Randomly resample n observations from **x with replacement** (see next slide for illustration), to get $\mathbf{z}^* = (x_1^*, \ldots, x_n^*)$,
2. Calculate and **record** value $\hat{\alpha}(\mathbf{z}^*) \equiv \hat{\alpha}^*$.

Resampling Methods: Bootstrap.

Bootstrap is a universal resampling method used to evaluate uncertainty around sample estimates for population parameter.
In particular, it helps evaluate **standard error** of an estimate.

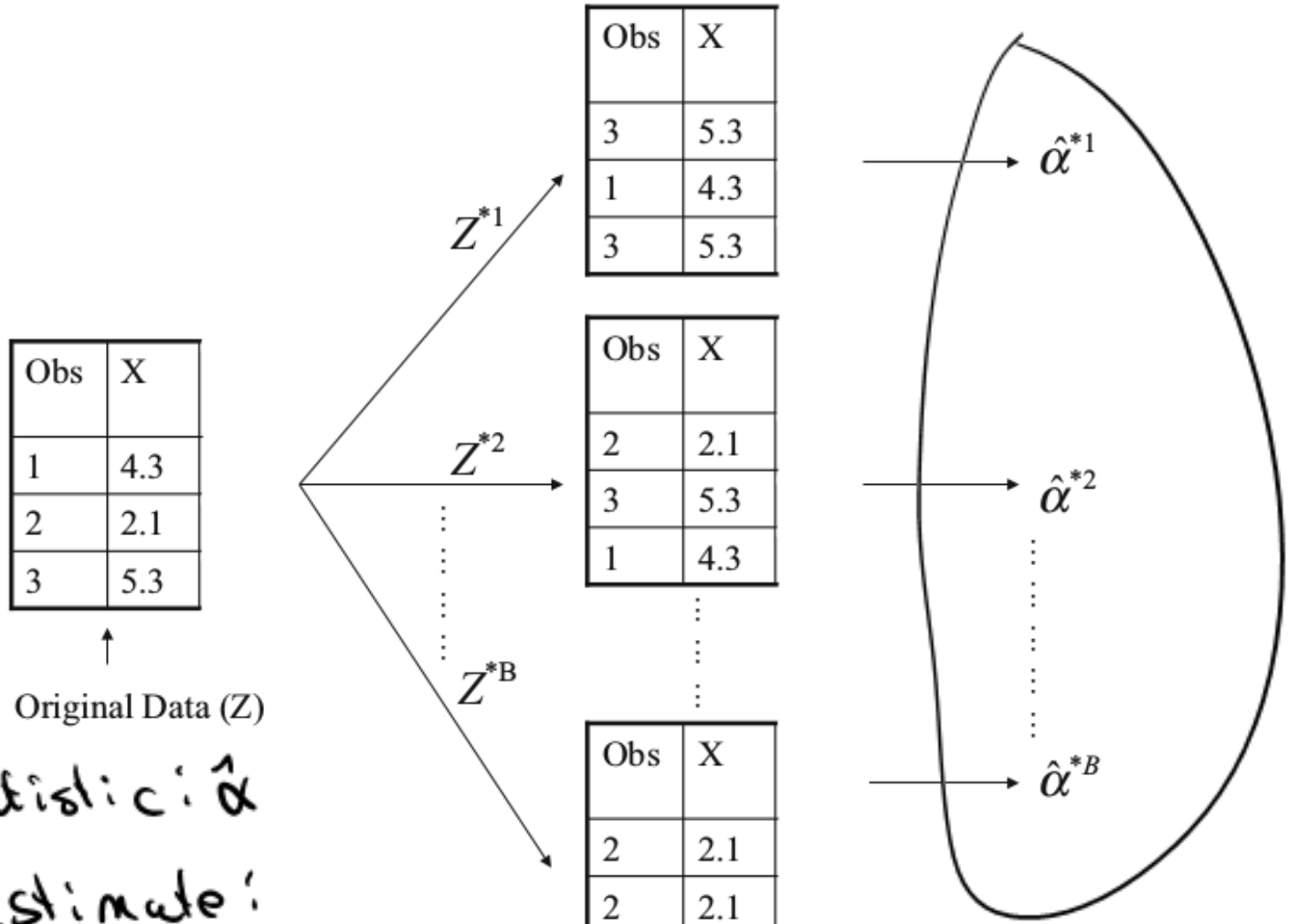
Presume you have a

- sample $\mathbf{x} = (x_1, \ldots, x_n)$ of n observations (e.g. *LSAT* scores),
- a sample statistic $\hat{\alpha}$ of interest (e.g. sample mean $\hat{\alpha}(\mathbf{x}) = \bar{\mathbf{x}}$)

MAIN IDEA of BOOTSTRAP (for standard error calculation):

1. Randomly resample n observations from **x with replacement** (see next slide for illustration), to get $\mathbf{z}^* = (x_1^*, \ldots, x_n^*)$,
2. Calculate and **record** value $\hat{\alpha}(\mathbf{z}^*) \equiv \hat{\alpha}^*$.
3. Repeat steps 1 & 2 a total of B times, to get a sample of values $\hat{\alpha}^{*,1}, \ldots, \hat{\alpha}^{*,B}$, calculate **standard deviation** of that sample.

Bootstrap: Illustration.



sample statistic: $\hat{\alpha}$

Bootstrap estimate:

mean $(\hat{\alpha}^*) = \hat{\alpha} + \text{bias}$

Tree-based Models: Decision Trees.

We've covered the following tree-based models: decision trees, bagging, random forests.

Two types of decision trees: regression trees and classification trees.

Tree-based Models: Decision Trees.

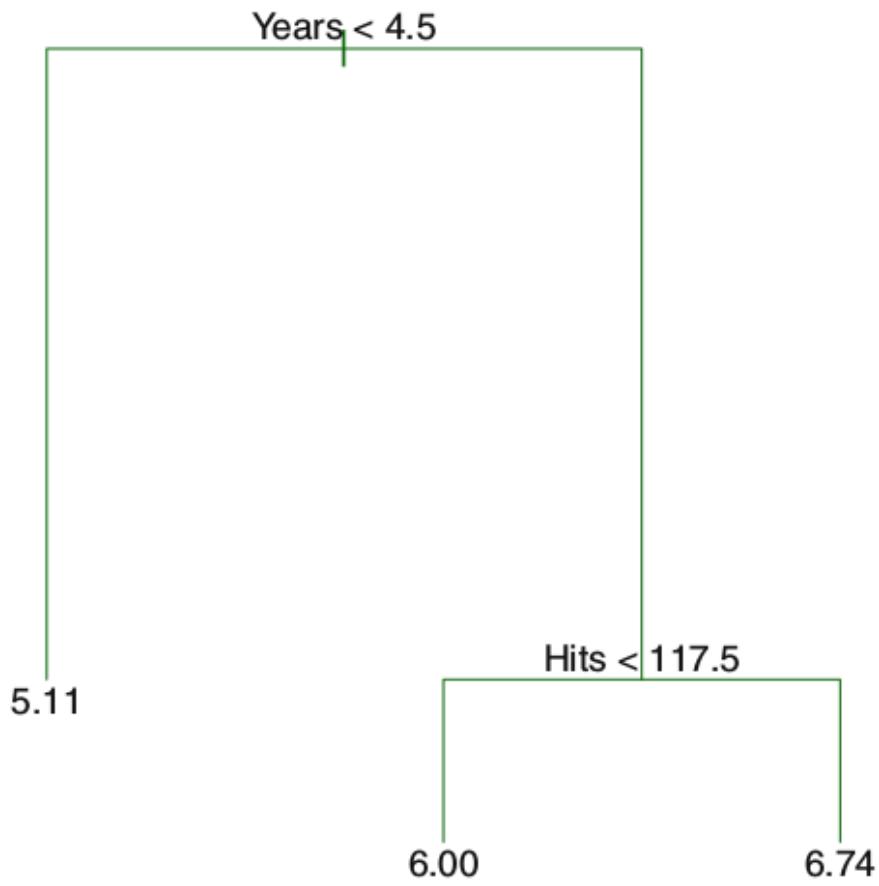
We've covered the following tree-based models: decision trees, bagging, random forests.

Two types of decision trees: regression trees and classification trees.

Example. For *Hitters* data on baseball hitters, we'd like to predict baseball player's *Salary* (qua**NT**.) based on

- *Years* - # years he played in major leagues, and
- *Hits* - # hits made in the previous year.

An example of a fitted **regresssion** tree looks as follows (plot on the right):



Regression Trees: *Hitters* example.

Example (cont'd):

– Predictor space (range of values for *Years* and *Hits* variables) got segmented into **3** regions (terminal nodes):

- $R_1 = (Years < 4.5)$
- $R_2 = (Years \geq 4.5) \ \ \& \ \ (Hits < 117.5)$,
- $R_3 = (Years \geq 4.5) \ \ \& \ \ (Hits \geq 117.5)$

Regression Trees: *Hitters* example.

Example (cont'd):

– Predictor space (range of values for *Years* and *Hits* variables) got segmented into **3** regions (terminal nodes):

- $R_1 = (Years < 4.5)$
- $R_2 = (Years \geq 4.5)$ & $(Hits < 117.5)$,
- $R_3 = (Years \geq 4.5)$ & $(Hits \geq 117.5)$

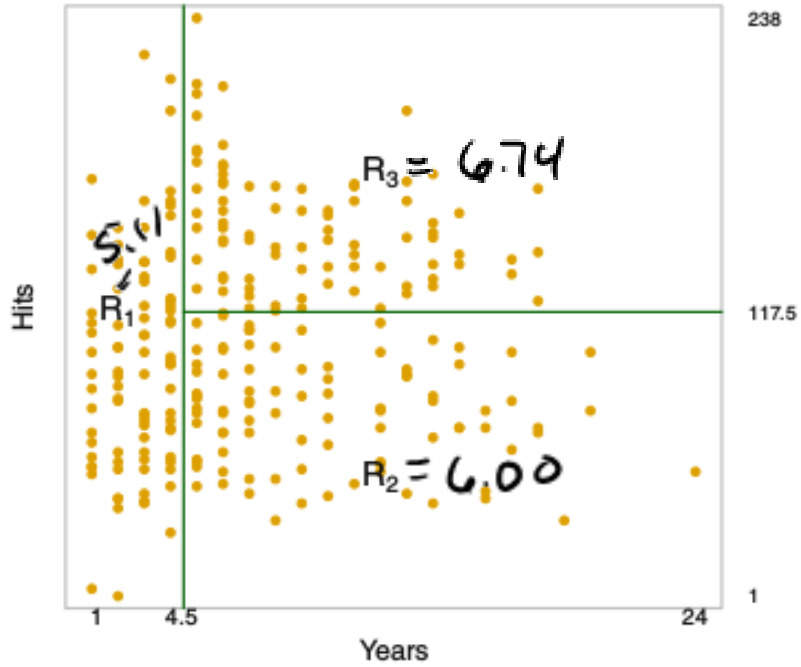
– *Salary* prediction in region R_i = (**mean** *Salary* of **all** hitters $\in \boldsymbol{R_i}$):

- Any hitter in R_1 (< 4.5 years of experience) is predicted to make $e^{5.11} \approx 165k$.
- For hitters with over 4.5 years of experience:
  - if hitter $\in R_2$ (< 117.5 hits), he is projected to make $e^{6.00} \approx 403k$,
  - while for R_3 (≥ 117.5 hits) - $e^{6.74} \approx 845k$

Back to *Hitters*: How do we segment?

Example (cont'd). How do we construct the regions $R_1, \ldots, R_J$?

1. Regions are built via splitting range of a variable. E.g. range of *Years* is split **at value** 4.5 into $(Years < 4.5)$ and $(Years \geq 4.5)$.

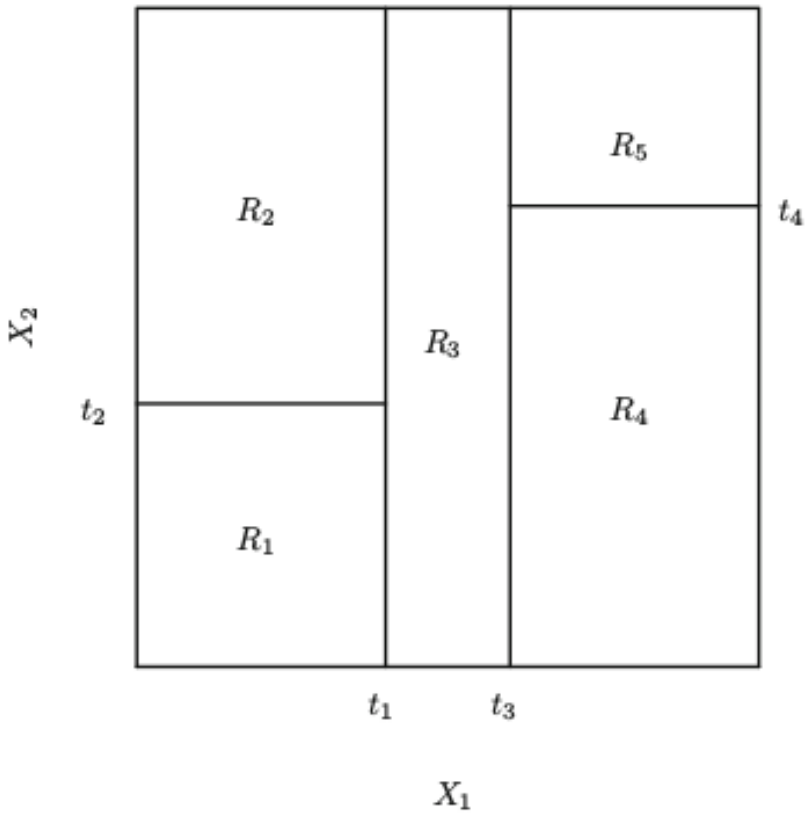


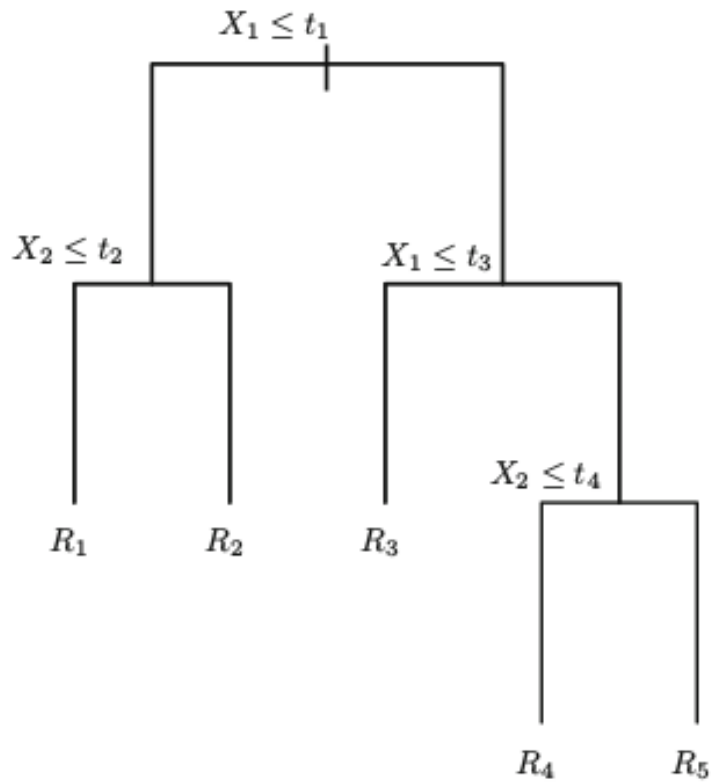
Results are boxes, or high-dimensional rectangles, for the sake of

- simplicity,
- ease of interpretation

Recursive Binary Splitting: Simulated Example.

Example. Below is the output of recursive binary splitting applied to a simulated two-dimensional example with full predictor space $\{(X_1, X_2)\}$. It resulted into five regions, which can be seen on the left, and the corresponding tree is on the right.





Overfitting Issue: Tree Pruning.

Recursive binary splitting usually results into large trees that

- produce good predictions on training data, but
- tends to overfit and perform poorly on test data.

A smaller tree with fewer splits ($\leftrightarrow$ fewer regions $R_1, \ldots, R_J$) might have better test data performance and better interpretation.

Overfitting Issue: Tree Pruning.

Recursive binary splitting usually results into large trees that

- produce good predictions on training data, but
- tends to overfit and perform poorly on test data.

A smaller tree with fewer splits ($\leftrightarrow$ fewer regions $R_1, \ldots, R_J$) might have better test data performance and better interpretation.

Solution:

1. Grow a large tree T_0, but then
2. **Prune** it back in order to obtain a subtree.

Pruning - selecting a subtree (of the large tree) which will yield the lowest **test** error rate.

For each subtree, test error is estimated via cross-validation).

Tree Pruning: *Carseats* example.

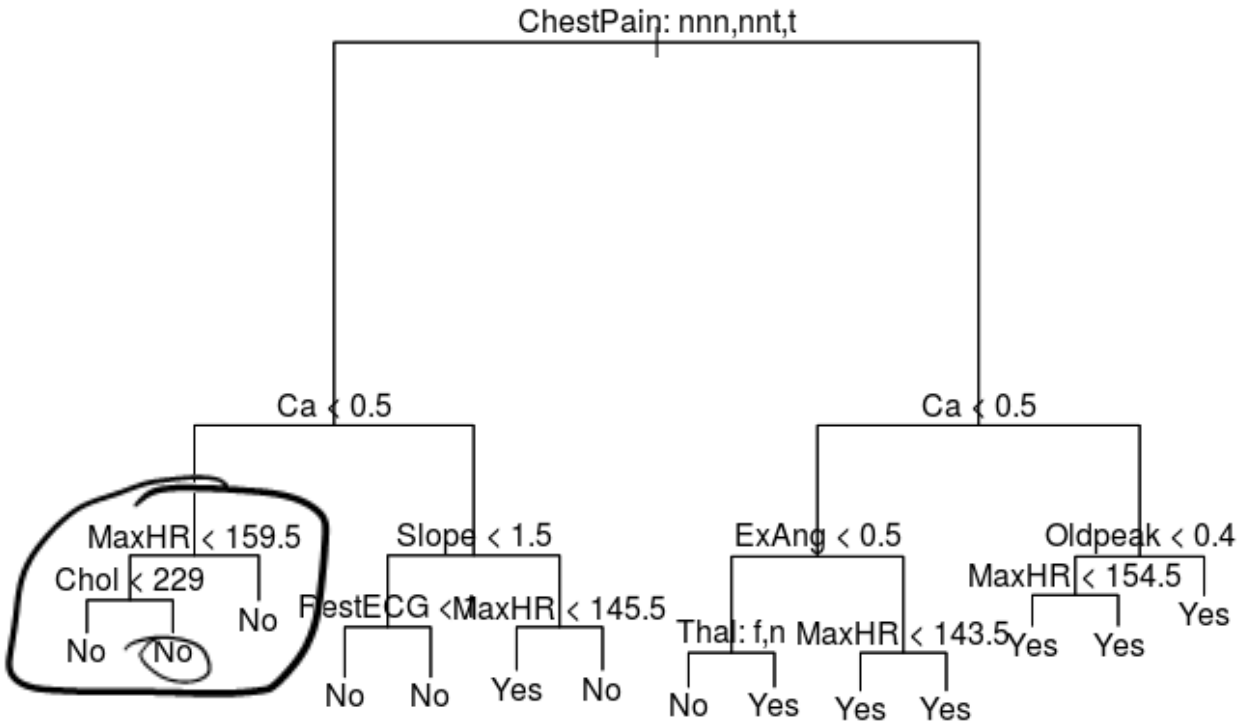
Example.
Results of pruning the classification tree for *Carseats* data (*Sales* = *High*/*Low*) from Lab #5.


```
>  cv.carseats$size
[1] 19 17 14 13  9  7  3  2  1
> cv.carseats$dev
[1] 55 55 53 52 50 56 69 65 80
```

Look for tree size corresponding to **smallest CV error**. Here, smallest CV error is 50 $\Longrightarrow$ best tree size is 9.

Classification Tree Example: *Heart* data.

Example. *Heart* data contains info on patients with chest pains, and we'd like to classify if a patient has a heart disease (*HD*) depending on multiple factors (*Age*, *Sex*, *Chol*). Below is the large classification tree:



Notice **qualitative predictors** and how they are being split.

Interpreting Splits and Node Summaries.

Example (cont'd).

1. How would you interpret this terminal node?

   For observations with $Thal \in \{f, n\}$ we predict "No", for obs. with $Thal \notin \{f, n\}$ we predict "Yes".

2. How would you interpret summary of terminal node 17?

```
8) MaxHR < 159.5 19   16.570 No ( 0.84211 0.15789 )
  16) Chol < 229 9     0.000 No ( 1.00000 0.00000 ) *
  17) Chol > 229 10   12.220 No ( 0.70000 0.30000 ) *
```

   Task: What's *Chol* > 229? What's number 10? What is 12.220? What is *No*? What's $(0.70, 0.30)$?

   Answer: All obs. in this branch have *Chol* > 229; 10 - total of these observations; 12.220 - deviance (Gini index); "No" - node prediction; 70% of "No" and 30% of "Yes" observations.

Classification Tree Example: *Heart* data, node purity.

Example (cont'd). Surprising characteristic of the resulting tree: some splits yield two terminal nodes with **same** predicted value.

E.g., $Chol < 229$ in the bottom left results into a *No* prediction for **both** nodes. Why perform it?

Classification Tree Example: *Heart* data, node purity.

Example (cont'd). Surprising characteristic of the resulting tree: some splits yield two terminal nodes with **same** predicted value.

E.g., *Chol* < 229 in the bottom left results into a *No* prediction for **both** nodes. Why perform it? It leads to increased node **purity**:

```
8) MaxHR < 159.5 19   16.570 No ( 0.84211 0.15789 )
  16) Chol < 229 9    0.000 No ( 1.00000 0.00000 ) *
  17) Chol > 229 10   12.220 No ( 0.70000 0.30000 ) *
```

- in the left node (*Chol* < 229), $9/9$ obs. have a *No* response,
- in the right node (*Chol* ≥ 229), it's just $7/10$.

Hence, the *Chol* < 229 split improves node purity

BUT

it does NOT reduce the classification error. That it means it is **likely to be eliminated during tree pruning**.

Node Interpretation: Another Example.

Example. From *Carseats* data in Lab #5:

```
1) root 400 541.500 No ( 0.59000 0.41000 )
  2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
    ...
      17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
```

For each particular branch of the tree, *R* displays:

- the split branch criterion (e.g. $Price < 92.5$),
- the total number of observations,
- the deviance (Gini index),
- the overall prediction (*Yes* or *No*), and
- fraction of obs. that take on values *Yes* and *No*, respectively.

Task: Interpret the terminal node 17: what's $CompPrice > 110.5$? What's number 5? What is 6.730? What is *Yes*? What's $(0.40, 0.60)$?

Answer: All obs. in this branch have $CompPrice > 110.5$; 5 - total observations; 6.730 - deviance (Gini index); "Yes" - node prediction; 40% of "No" and 60% of "Yes" observations.

Decision Trees: Issues & Their Solution (Bagging).

Decision trees from previous lecture do suffer from **two issues**:

- They struggle with prediction accuracy.

Decision Trees: Issues & Their Solution (Bagging).

Decision trees from previous lecture do suffer from **two issues**:

- They struggle with prediction accuracy.
- They suffer from **high variance**: different subsets of same data could yield drastically different results.

To tackle those issues, we defer to

- **Bagging**, and
- **Random Forests**.

Bagging, or bootstrap aggregation, is a general-purpose procedure for **reducing the variance** of a statistical learning method. Here, bootstrap is used differently compared to when we were evaluating standard errors of estimates:

1. Bootstrapped samples produce different **large** trees $T_1, \ldots, T_B$,
2. which, in their turn, produce predictions $Z_1, \ldots, Z_B$ for a given $\mathbf{x}$,
3. which are then **averaged** to **reduce variance** of the prediction.

Bagging for Decision Trees.

How do we apply bagging to decision trees?

1. Generate B bootstrapped training sets $\mathbf{x}_1^*, \ldots, \mathbf{x}_B^*$.
2. Construct B **large** (**not** pruned) regression trees for those sets, obtain predictions $z_1, \ldots, z_B$ from those trees.
3. Average the resulting predictions $\bar{\mathbf{z}} = \frac{1}{B}\sum_{b=1}^{B} z_b$ from those trees.

Bagging for Decision Trees.

How do we apply bagging to decision trees?

1. Generate B bootstrapped training sets $\mathbf{x}_1^*, \ldots, \mathbf{x}_B^*$.
2. Construct B **large** (**not** pruned) regression trees for those sets, obtain predictions $z_1, \ldots, z_B$ from those trees.
3. Average the resulting predictions $\bar{\mathbf{z}} = \frac{1}{B}\sum_{b=1}^{B} z_b$ from those trees.

To produce a single prediction across all B trees via bagging,

- for regression trees - we take **average** of all B predictions (e.g. for *Hitters* example, our final log-Salary prediction for each player was an average of $B = 100$ bootstrap predictions for that player).

Bagging for Decision Trees.

How do we apply bagging to decision trees?

1. Generate B bootstrapped training sets $\mathbf{x}_1^*, \ldots, \mathbf{x}_B^*$.
2. Construct B **large** (**not** pruned) regression trees for those sets, obtain predictions $z_1, \ldots, z_B$ from those trees.
3. Average the resulting predictions $\bar{\mathbf{z}} = \frac{1}{B}\sum_{b=1}^{B} z_b$ from those trees.

To produce a single prediction across all B trees via bagging,

- for regression trees - we take **average** of all B predictions (e.g. for *Hitters* example, our final log-Salary prediction for each player was an average of $B = 100$ bootstrap predictions for that player).
- for classification task (qualitative response)?

Bagging for Decision Trees.

How do we apply bagging to decision trees?

1. Generate B bootstrapped training sets $\mathbf{x}_1^*, \ldots, \mathbf{x}_B^*$.
2. Construct B **large** (**not** pruned) regression trees for those sets, obtain predictions $z_1, \ldots, z_B$ from those trees.
3. Average the resulting predictions $\bar{\mathbf{z}} = \frac{1}{B}\sum_{b=1}^{B} z_b$ from those trees.

To produce a single prediction across all B trees via bagging,

- for regression trees - we take **average** of all B predictions (e.g. for *Hitters* example, our final log-Salary prediction for each player was an average of $B = 100$ bootstrap predictions for that player).
- for classification task (qualitative response)? Take **majority vote** (the **most frequently** predicted class)

Variable Importance Measures.

While bagging improves prediction performance compared to single decision tree, it comes with a relative drawback: it is **difficult to interpret** the resulting model.

Variable Importance Measures.

While bagging improves prediction performance compared to single decision tree, it comes with a relative drawback: it is **difficult to interpret** the resulting model.

Partial solution: relative **variable importance** measures.

- If variable is important, the tree split over that variable causes the RSS (or Gini index for classification trees) to decrease the most.
- Hence, the measure of variable's importance $\equiv$ total amount by which RSS (Gini) **decreased** after each split over that variable.
- **Larger** value $\Longrightarrow$ **more** importance to the variable.

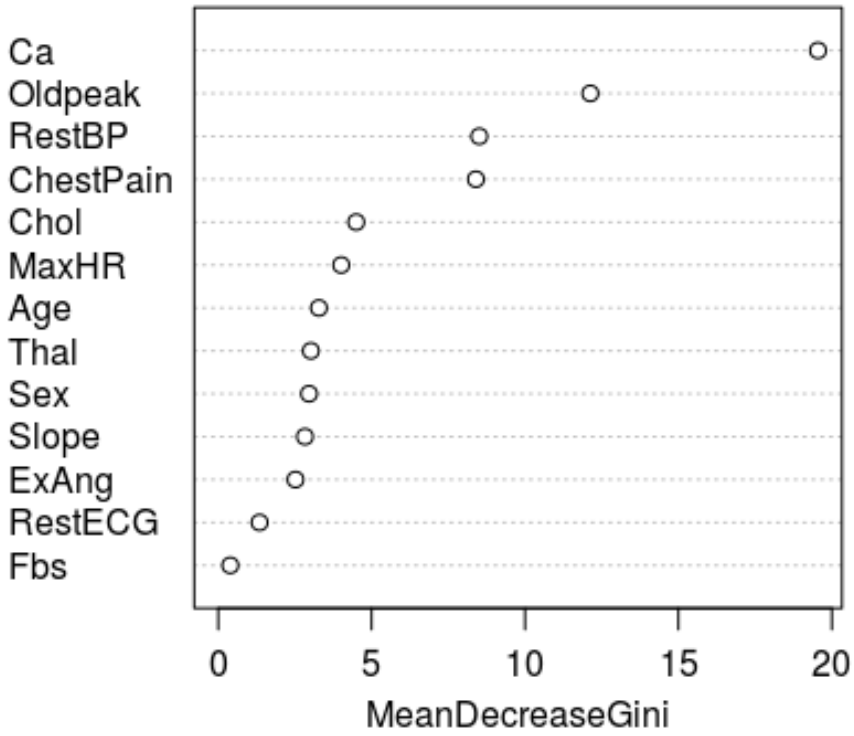
The reason for us specifying option *importance* $=$ *T* of *randomForest*() function was to be able to plot those importance measures as:

```
> varImpPlot(bag.model)
```

Variable Importance Measure: *Heart* example.

Example. Plot of variable importances for *Heart* data contains variable hierarchy w. r. t. mean decrease in Gini index (right).

Largest decreases were recorded for *Ca* variable, with *Oldpeak*, *RestBP* and *ChestPain* being in the Top-4.



JUST REMEMBER - THE LARGER THE VALUE (be it *DecreaseGini* or *PurityIncrease*, etc)

$\Longrightarrow$

THE MORE IMPORTANT THE VARIABLE IS.

Random Forests.

Random forests add a small tweak to further improve on bagging:

1. Random forests also grow B large **un**-pruned trees, **BUT**
2. each time a tree split is considered, it picks a **random** subset of m ($\approx \sqrt{p}$) predictors from the full set of p predictors.

Random Forests.

Random forests add a small tweak to further improve on bagging:

1. Random forests also grow B large **un**-pruned trees, **BUT**
2. each time a tree split is considered, it picks a **random** subset of m ($\approx \sqrt{p}$) predictors from the full set of p predictors.

That tweak leads to

1. Decorrelating the bagged trees (why?),
   - Less of a chance to have same variable dominating each bagged tree (which would've led to high correlation between estimates),

Random Forests.

Random forests add a small tweak to further improve on bagging:

1. Random forests also grow B large **un**-pruned trees, **BUT**
2. each time a tree split is considered, it picks a **random** subset of m ($\approx \sqrt{p}$) predictors from the full set of p predictors.

That tweak leads to

1. Decorrelating the bagged trees (why?),
   - Less of a chance to have same variable dominating each bagged tree (which would've led to high correlation between estimates),
2. hence stabilizing the variance of the estimate

In general, we would expect:

- Test error for **random forest** to be smaller than for **bagging**, while
- both **random forest** & **bagging** test errors should be smaller than those for a **single decision tree**.

Mathematical Model of a (Linear) Neuron.

Similar to the biological neuron structure, ANNs define the neuron as a:

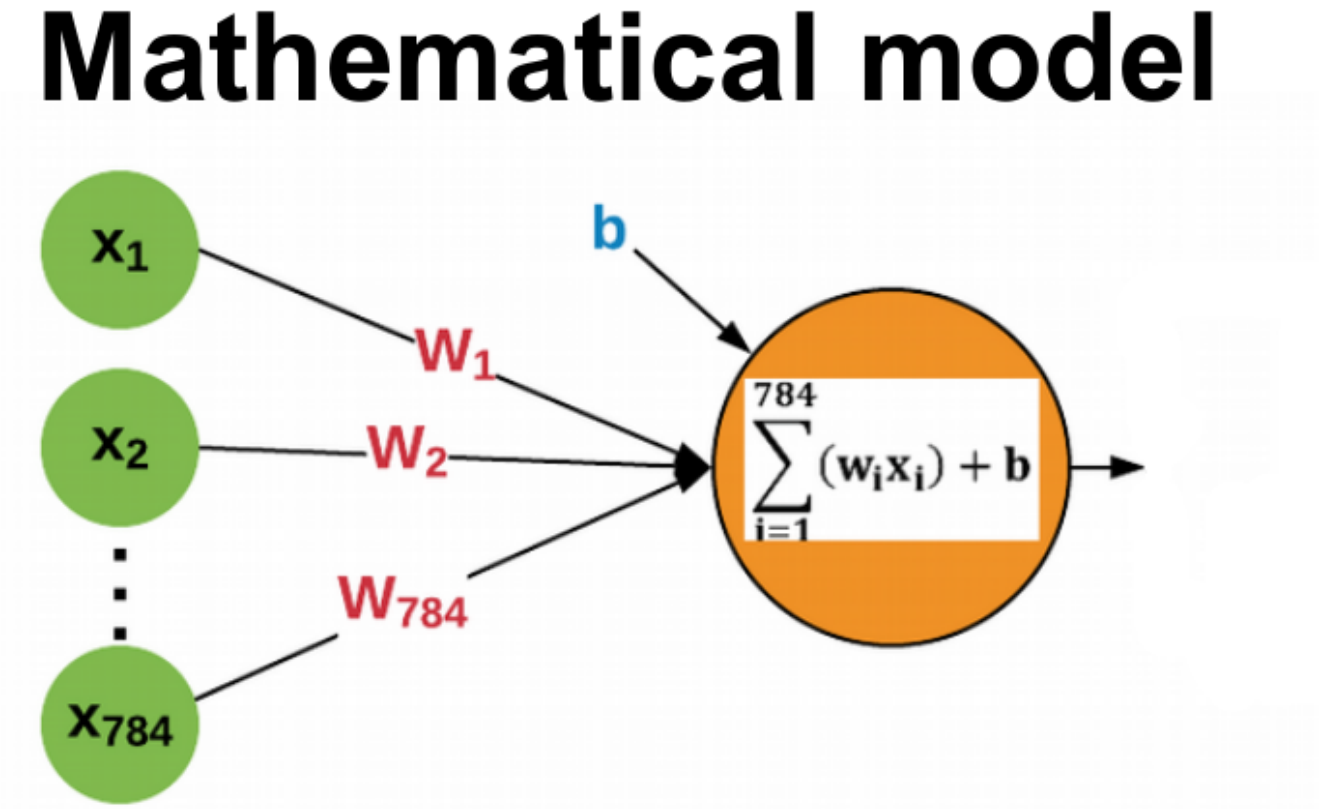
"Central processing unit that performs a **mathematical operation** to generate **output** from a set of **inputs**."

Mathematical Model of a (Linear) Neuron.

Similar to the biological neuron structure, ANNs define the neuron as a:

"Central processing unit that performs a **mathematical operation** to generate **output** from a set of **inputs**."

For a **linear** neuron, mathematical model would look as follows:

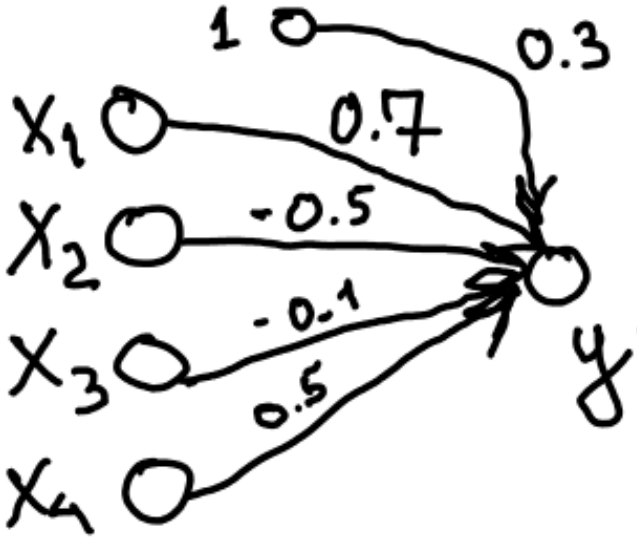


Linear Neuron: Kindergarten Example.

Example. Presume we have a linear neuron with

- inputs x_1, x_2, x_3, x_4,
- weights $w_1 = 0.7$, $w_2 = -0.5$, $w_3 = -0.1$, $w_4 = 0.5$
- for bias node ($\equiv 1$), weight is $b = 0.3$

Draw the corresponding neuron structure. Calculate the output for $x_1 = 3$, $x_2 = 7$, $x_3 = -3$, $x_4 = 10$.



$$y = \sum_{i=1}^{4} w_i x_i + b \cdot (1)$$

$$\hat{y} = 0.7 \cdot 3 + (-0.5) \cdot 7 + (-0.1) \cdot (-3) + 0.5 \cdot 10 + 0.3 \cdot 1 =$$
$$= 2.1 - 3.5 + 0.3 + 5 + 0.3 = 4.2$$