



DSC 80 winter 2025 final review

TA: Mizuho Fukuda
Tutors: Gabriel Cha, Ylesia Wu





final logistics

- thursday, 3/20
- 11:30am-2:30pm in SOLIS 104
- see seating chart [here](#)
- 180 minutes paper exam
- 2 sheets of hand-written notes (front + back)
- all lectures (first half is midterm redemption)



table of contents

01

[HTML and web scraping](#)

02

[regex and text features](#)

03

[linear regression](#)

04

[feature engineering](#)

05

[classification](#)

06

[model tuning](#)



HTML and web scraping



HTML tags

element	description
<html>	root element: enclose the whole document (indicates this is an html file).
<head>	metadata: contains metadata you want to apply across whole document.
<body>	document content: contains all visible content of the page.
<div>	block-level container: groups a block of content; usually used for layout purposes.
	inline container: used to manipulate a portion of text or elements within a line.
<p>	paragraph: starts on a new line and adds margin before and after the block.
<a>	hyperlink: creates a clickable link
<h1>, <h2>, ...	different level headings: gets smaller as you go from h1 ... h6.
	image: displays images. self-closing (no tag required).

```
<html>
  <head>
    <title>Title Text</title>
  </head>
  <body>
    <h1>h1 text</h1>

    <h2>h2 text</h2>

    <h3>h3 text</h3>

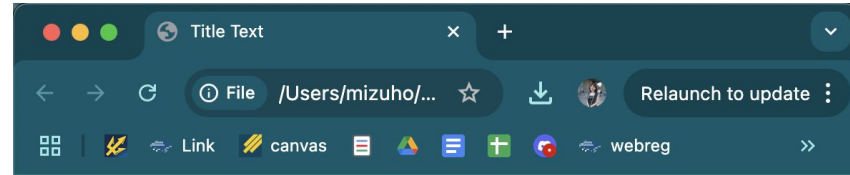
    <p>
      First p tag: The giant panda (Ailuropoda melanoleuca), also known as the panda bear or
      simply panda, is a bear species endemic to China. It is characterised by its white coat
      with black patches around the eyes, ears, legs and shoulders.
    </p>

    <p>
      Second p tag:
      <a href="https://sdzwildlifeexplorers.org/stories/precious-pandas">
        this is a hyperlink to San Diego Zoo's page about pandas
      </a>
    </p>

  </body>
</html>
```



h1 text

h2 text

h3 text

First p tag: The giant panda (*Ailuropoda melanoleuca*), also known as the panda bear or simply panda, is a bear species endemic to China. It is characterised by its white coat with black patches around the eyes, ears, legs and shoulders.

Second p tag: [this is a hyperlink to San Diego Zoo's page about pandas](https://sdzwildlifeexplorers.org/stories/precious-pandas)



[feel free to access this html file and local image here!](#)

parsing HTML

```
soup = bs4.BeautifulSoup(HTML string)
```

→ parsed document where you can access elements of the HTML

- `soup.find(tag, attrs={})`

finds the **first occurrence** of a tag (matching the specified attributes)

- `soup.find_all(tag, attrs={})`

finds **all occurrences** of a tag (matching the specified attributes) as a list

- `soup.find(tag).text`: returns only the **text** that is part of the tag
- `soup.find(tag).attrs`: lists **all attributes** of the tag
- `soup.find(tag).get(attr)`: returns the **value** of specified attribute

```
<html>
  <head>
    <title>ZOMBO</title>
  </head>

  <body>
    <h1>Welcome to Zombo.com</h1>
    <div id="greeting">
      <ul>
        <li>This is Zombo.com, welcome!</li>
        <li>This is Zombo.com</li>
        <li>Welcome to Zombo.com</li>
        <li>You can do anything at Zombo.com -- anything at all!</li>
        <li>The only limit is yourself.</li>
      </ul>
    </div>
    <div id="footnotes" class="faded">
      <h3>Footnotes</h3>

      <ol id="footnotes">
        <li>Please consider <a href="paypal.html">donating!</a></li>
        <li>Made in California with <a href="https://reactj.org">React.</a></li>
      </ol>
    </div>
  </body>
</html>
```

Problem 3.2

The page shown above contains five greetings, each one a list item in an unordered list. The first greeting is This is Zombo.com, welcome!, and the last is The only limit is yourself.

Suppose we have parsed the HTML into a BeautifulSoup object stored in the variable named soup.


Which of the following pieces of code will produce a list of BeautifulSoup objects, each one representing a single greeting list item? Mark all which apply.

- soup.find('div').find_all('li')
- soup.find_all('li', id='greeting')
- soup.find('div', id='greeting').find_all('li')
- soup.find_all('ul/li')

soup.find_all('div', id="greeting") [0] . . . not real syntax



regex and text features



regex functions

- `re.findall(regexpattern, str)`:
returns a list containing all matches
- `re.search(pattern, str)`:
returns a match object if there is a match anywhere in the string
- `re.split(pattern, str)`:
returns a list where the string has been split at each match
- `re.sub(pattern, replace, str)`:
replaces all matches with the replace text

f

[find more helpful examples here!](#)

$(abc) \rightarrow 'abc'$

$[abc] \rightarrow 'a', 'b', \text{or}' 'c'$

$(\dots)(\dots\$)$

Character	Description	Example pattern
[]	a set of characters	"[A-Z#]"
.	any single character (except newline)	"he..o"
^	starts with (i.e. check beginning of string)	"^pandas"
\$	ends with (i.e. check end of string)	"pandas\$"
*	zero or more occurrences of the preceding character	"lo*k"
+	one or more occurrences of the preceding character	"lo+k"
?	zero or one occurrence of the preceding character	"colou?r"
{ }	specify exactly the number of occurrences	"ap{2}le"
	"or"; matches either the pattern before or after	"this that"
()	capture / group	"#(tag)"

pat:
 $()\$$
 $(\$)$
"pandas blah"
~~"hi pandas"~~
↑

[find more helpful examples here!](#)

Character	Description	Example pattern
<code>\b</code>	word break	<code>"\bword"</code>
<code>\d</code>	digits (numbers 0-9)	<code>"\d{4}"</code>
<code>\s</code>	space	<code>"hi\shello"</code>
<code>\w</code>	alphanumeric and <code>"_"</code>	<code>"\w+@\w+.com"</code>
<code>[a-z]</code>	lower case characters from a to z	<code>"[a-h]+"</code>
<code>[^abc]</code>	NOT a, b, or c	<code>"[^a-z]+"</code>

bag of words

→ defines a vector space in $\mathbb{R}^{\text{number of unique words}}$

e.g. if you have 3 separate strings with 10 total unique words across all of them, you get a bag of words matrix of size 3x10

	text		hello	world	good	morning	happy
0	hello world	0	1	1	0	0	0
1	good morning world	1	0	1	1	1	0
2	happy world	2	0	1	0	0	1

cosine similarity

if $\cos \theta$ is close to 1:
2 documents are similar

if $\cos \theta$ is close to 0:
they are different

	hello	world	good	morning	happy
0	1	0	0	0	0
1	0	1	1	1	0
2	0	1	0	0	1

$$(1)(0) + (1)(1) + (0)(1) + (0)(1) + (0)(0) = 1$$

for text 0 and text 1:

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \frac{[1, 1, 0, 0, 0] \cdot [0, 1, 1, 1, 0]}{\sqrt{1^2 + 1^2} \sqrt{0^2 + 1^2 + 1^2 + 1^2}} = \frac{1}{\sqrt{6}}$$

TF-IDF

i like data science
i hate data science

- **term frequency** (tf) of a word **t** in a document **d**:

$$tf(t, d) = \frac{\# \text{ of occurrences of } t \text{ in } d}{\text{total } \# \text{ of words in } d}$$

if $tf(t, d)$ is large, then t occurs often in d .

- **inverse document frequency** (idf) of a word **t** in a set of documents:

$$idf(t) = \log\left(\frac{\text{total } \# \text{ of documents}}{\# \text{ of documents in which } t \text{ appears}}\right)$$

if $idf(t)$ is large, then t does not occur often in the set of documents

$$\text{TF-IDF}(t, d) = \underbrace{tf(t, d) \cdot idf(t)}$$

- **TF-IDF** quantifies how well a word **t** summarizes document **d**.
- This value is largest when:
 - both $tf(t,d)$ and $idf(t)$ are large
 - **t** occurs often in **d** but does not occur often in other documents.
 - **good summary**
- When $tf(t,d)$ is large but $idf(t)$ is small
 - **t** occurs often in general; not a good word to summarize **d**
- When $idf(t)$ is large but $tf(t,d)$ is small
 - **t** does not occur often in **d** or in general; just a random word

Problem 5.3

Chen downloaded 4 independent reviews of a new vacuum cleaner from Amazon (as shown in the 4 sentences below).

Sentence 1: 'if i could give this vacuum zero stars i would'

Sentence 2: 'i will not order again this vacuum is garbage'

~~Sentence 3: 'Love Love Love i love this product'~~

Sentence 4: 'this little vacuum is so much fun to use i love it'

$$X = \frac{1}{10} \cdot \log \left(\frac{4}{3} \right)$$

total # of d

X is the 'Term frequency-Inverse Document Frequency (TF-IDF)' of the word 'vacuum' in sentence 1.

Chen replaces sentence 3 with the following new sentence/review.

New Sentence 3: 'Love Love Love i love this vacuum'

$$Y = \frac{1}{10} \cdot \log \left(\frac{4}{4} \right) \rightarrow 0$$
$$Y = 0$$

Y is the 'TF-IDF' of the word 'vacuum' in sentence 1 after the sentence 3 is replaced by the new sentence 3.

Given the above information, which of the following statements is true?

$X = Y$ ~~X~~

$X > Y > 0$ ~~X~~

$X > 0$ and $Y = 0$

$X > 0$ and ~~$Y > X$~~



linear regression



linear regression

Given prediction function:

$$H(\mathbf{x}_i) = w_0 + w_1(x_1) + w_2(x_2) + \dots w_m(x_m)$$

Find the $w_0, w_1, \dots w_m$ that minimize:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - H(\mathbf{x}_i))^2$$

y =

0 5

1 4

2 2

3 1

4 3

Using a single value to predict y :

$$H = \text{predicted } y = w_0$$

The single value that minimizes MSE is always the **mean**.

$$w_0 = \text{mean}(y) = 3$$

X =

	feature 1	feature 2
0	2.0	1
1	1.5	0
2	1.0	1
3	0.4	0
4	1.2	1

y =

	y
0	5
1	4
2	2
3	1
4	3

Using only one feature of X (column 'feature 1'):

$$H(x_i) = \text{predicted } y = \overbrace{w_0 + w_1(\text{feature 1})}^{b + mX}$$

What are the best parameters w_0, w_1 that minimize MSE / RMSE ?

```
lr = LinearRegression()  
lr.fit(X[['feature 1']],y)
```

LinearRegression

LinearRegression()

```
lr.predict(X[['feature 1']])
```

```
array([[5.04971591],  
       [3.73579545],  
       [2.421875  ],  
       [0.84517045],  
       [2.94744318]])
```

```
print('w0 = ', lr.intercept_[0])  
print('w1 = ', lr.coef_[0,0])
```

```
w0 = -0.20596590909090962  
w1 = 2.627840909090909
```

Training step:

- Linear Regression model takes in **X** matrix with one column 'feature 1' and target **y** and calculates intercept and coefficient that best minimizes mean square error.

$X[['feature 1']] \rightarrow \text{dataframe}$

Prediction:

- The "trained" model predicts **y** based on the parameters it learned

Best parameters:

- sklearn LinearRegression allows you to access the parameters once model is fitted.

Now let X be a 5x3 matrix.

X =

	x1	x2	x3		y
0	1	2.0	100	0	5
1	0	1.5	120	1	4
2	1	1.0	200	2	2
3	0	0.4	75	3	1
4	1	1.2	150	4	3

y =

Using all features of X:

$$H(x_i) = \text{predicted } y = w_0 + w_1(x_1) + w_2(x_2) + w_3(x_3)$$

What are the best parameters w_0, w_1, w_2, w_3 that minimize MSE / RMSE ?

```
lr = LinearRegression()  
lr.fit(X,y)
```

Training step:

- Fitting on the entire X matrix

```
LinearRegression  
LinearRegression()
```

```
lr.predict(X)
```

Prediction:

- Predicting on entire X matrix

```
array([[5.08879116],  
       [3.95837914],  
       [2.12624993],  
       [1.04162086],  
       [2.7849589 ]])
```

```
print('w0 = ', lr.intercept_[0]) ...
```

Best parameters:

- w_0 is the intercept
- w_1 , w_2 , and w_3 are the coefficients for each feature of X.

```
w0 = 0.11037380241869155  
w1 = -0.2846578713156235  
w2 = 2.7418721532904042  
w3 = -0.0022066907491752833
```


how good is my linear regression model?

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - H(\mathbf{x}_i))^2}$$

\sqrt{MSE}

Lower

→ actual y is close to predicted y on average
= good prediction

$$R^2 = \frac{\text{var}(\text{predicted } y \text{ values})}{\text{var}(\text{actual } y \text{ values})}$$

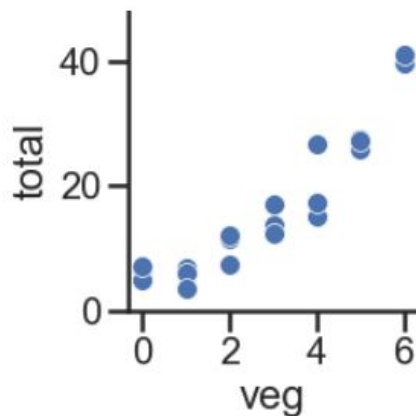
range = (0, 1)

Closer to 1

→ variation in the y is explained by the linear model
= good linear fit

Every week, Lauren goes to her local grocery store and buys a varying amount of vegetable but always buys exactly one pound of meat (either beef, fish, or chicken). We use a linear regression model to predict her total grocery bill. We've collected a dataset containing the pounds of vegetables bought, the type of meat bought, and the total bill. Below we display the first few rows of the dataset and two plots generated using the entire training set.

veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9



Problem 9.1

Suppose we fit the following linear regression models to predict 'total' using the squared loss function. Based on the data and visualizations shown above, for each of the following models $H(x)$, determine whether **each fitted model coefficient** w^* is positive (+), negative (-), or exactly 0. The notation meat=beef refers to the one-hot encoded 'meat' column with value 1 if the original value in the 'meat' column was 'beef' and 0 otherwise. Likewise, meat=chicken and meat=fish are the one-hot encoded 'meat' columns for 'chicken' and 'fish', respectively.

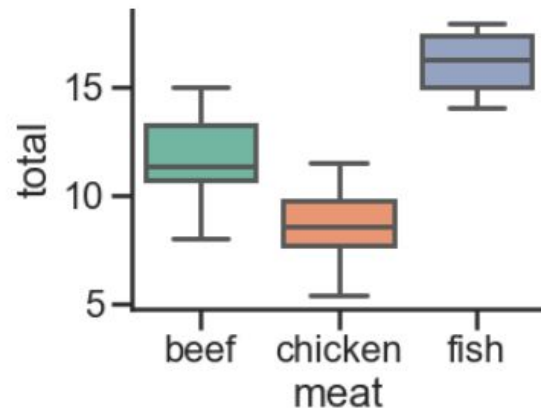
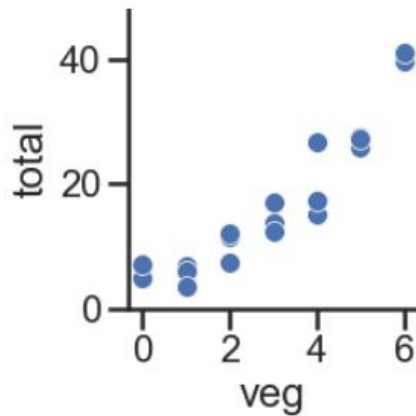
For example, in part (iv), you'll need to provide three answers: one for w_0^* (either positive, negative, or 0), one for w_1^* (either positive, negative, or 0), and one for w_2^* (either positive, negative, or 0).

or need more information

- i. $H(x) = w_0$
- ii. $H(x) = w_0 + w_1 \cdot \text{veg}$
- iii. $H(x) = w_0 + w_1 \cdot (\text{meat}=\text{chicken})$
- iv. $H(x) = w_0 + w_1 \cdot (\text{meat}=\text{beef}) + w_2 \cdot (\text{meat}=\text{chicken})$
- v. $H(x) = w_0 + w_1 \cdot (\text{meat}=\text{beef}) + w_2 \cdot (\text{meat}=\text{chicken}) + w_3 \cdot (\text{meat}=\text{fish})$



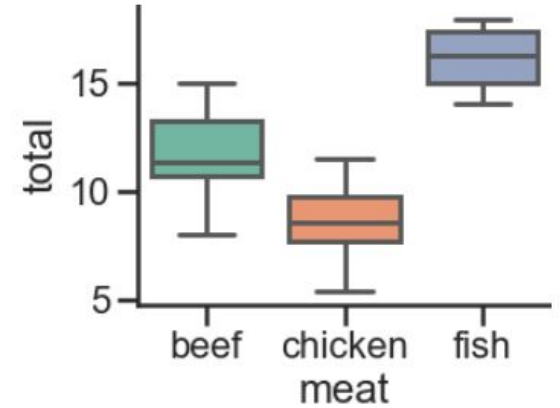
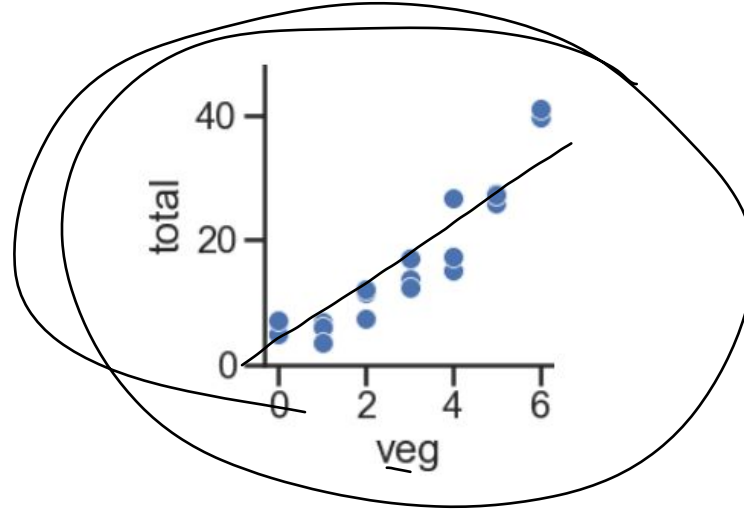
veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9



$$H(x) = w_0$$

w0: positive / negative / 0 / need more information ?

veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9

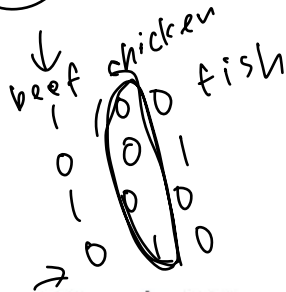
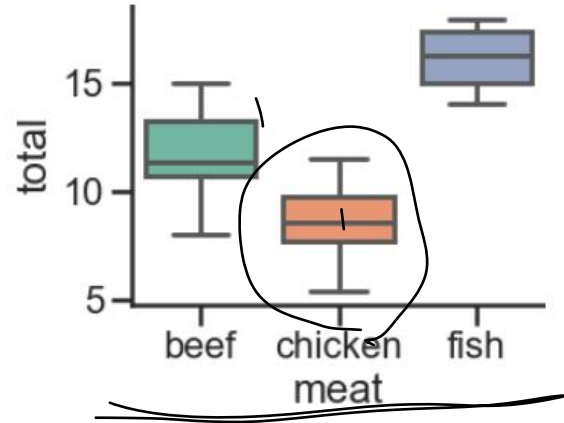
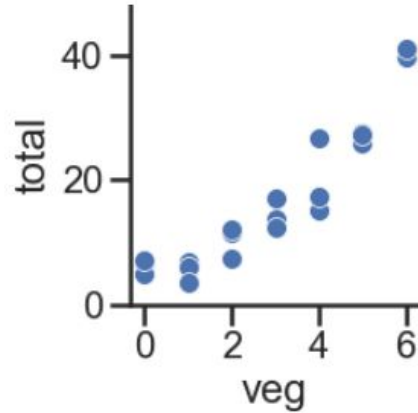


$$H(x) = w_0 + w_1 \cdot \text{veg} \rightarrow \text{line: } b + mx$$

w_0 : positive / negative / 0 / need more information ?

w_1 : positive / negative / 0 / need more information ?

veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9



$(0, 1)$

$$H(x) = w_0 + w_1 \cdot \text{(meat=chicken)}$$

w_0 : positive / negative / 0 / need more information?

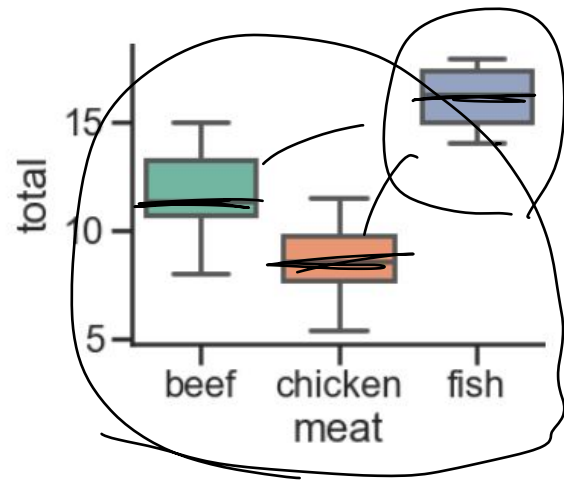
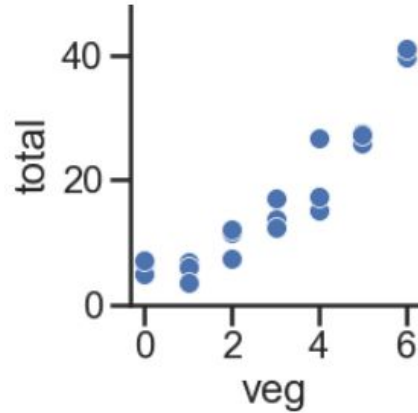
w_1 : positive / negative / 0 / need more information?

$$\hat{\text{total}} = w_0 + w_1 \text{ when chicken}$$

$$\hat{\text{total}} = w_0 \text{ no chicken}$$

$$w_0 > w_0 + w_1$$

veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9



$$H(x) = w_0 + w_1 \cdot (\text{meat}=\text{beef}) + w_2 \cdot (\text{meat}=\text{chicken})$$

w0: positive / negative / 0 / need more information?

w1: positive / negative / 0 / need more information?

w2: positive / negative / 0 / need more information?

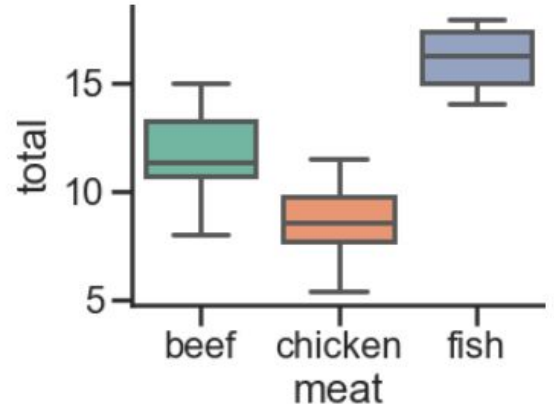
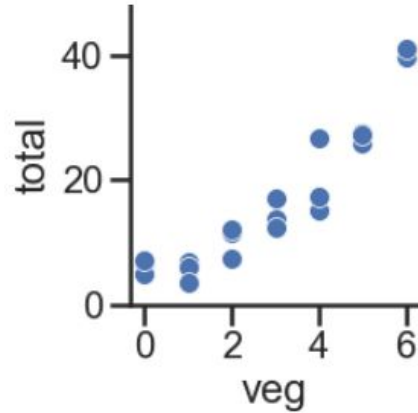
$$\hat{\text{total}} = w_0 + w_1 \quad (\text{beef})$$

$$\hat{\text{total}} = w_0 + w_2 \quad (\text{chicken})$$

$$\hat{\text{total}} = w_0 \quad (\text{fish})$$

$$w_0 > w_0 + w_1 \quad \text{or} \quad w_0 + w_2$$

veg	meat	total
1	beef	13
3	fish	19
2	beef	16
0	chicken	9



$$H(x) = w_0 + w_1 \cdot \overset{(0,1)}{\text{meat=beef}} + w_2 \cdot \overset{(0,1)}{\text{meat=chicken}} + w_3 \cdot \overset{(0,1)}{\text{meat=fish}}$$

w0: positive / negative / 0 / need more information?

w1: positive / negative / 0 / need more information?

w2: positive / negative / 0 / need more information?

w3: positive / negative / 0 / need more information?

beef: $\hat{\text{total}} = w_0 + w_1$

chicken: $\hat{\text{total}} = w_0 + w_2$

fish: $\hat{\text{total}} = w_0 + w_3$

multicollinearity

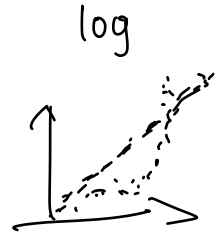


feature engineering



feature transformations

- **One hot encoding:**
 - turns categorical features into multiple binary features
 - # of binary features = number of unique values in the original column
- **Standardization:**
 - $x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$ *z-score*
 - ensures all quantitative features are on the same scale
- **Linearization:**
 - apply a non-linear transformation to data to make it linear
- **Binarization:**
 - turn quantitative data into 2 groups (1s and 0s) based on a threshold
- **Discretization**
 - turn quantitative data into percentiles or quantiles



One-Hot-Encoding

	meat
0	Beef
1	Chicken
2	Chicken
3	Fish
4	Beef
5	Fish

```
ohe = OneHotEncoder()  
ohe.fit(df)  
ohe.transform(df)
```

	beef	chicken	fish
([1., 0., 0.]	0.	0.
	[0., 1., 0.]	1.	0.
	[0., 1., 0.]	1.	0.
	[0., 0., 1.]	0.	1.
	[1., 0., 0.]	0.	0.
	[0., 0., 1.]	0.	1.]
)			

Standardization

	x1	x2	x3
0	1	2.0	100
1	0	1.5	120
2	1	1.0	200
3	0	0.4	75
4	1	1.2	150

weight

```
scaler = StandardScaler()  
scaler.fit(X)  
scaler.transform(X).round(2)
```



[0.82,	1.47,	-0.67]
[-1.22,	0.53,	-0.21]
[0.82,	-0.41,	1.64]
[-1.22,	-1.55,	-1.25]
[0.82,	-0.04,	0.49]

Multicollinearity

- Occurs when we have redundant features or features that are highly correlated with each other
 - makes coefficients of the regression model uninterpretable
 - **does not** impact the model's prediction though!

minimize MSE

instead of this:

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

do this:

```
ohe = OneHotEncoder(drop='first', sparse_output=False)  
ohe.fit(df)  
ohe.transform(df)
```

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



classification

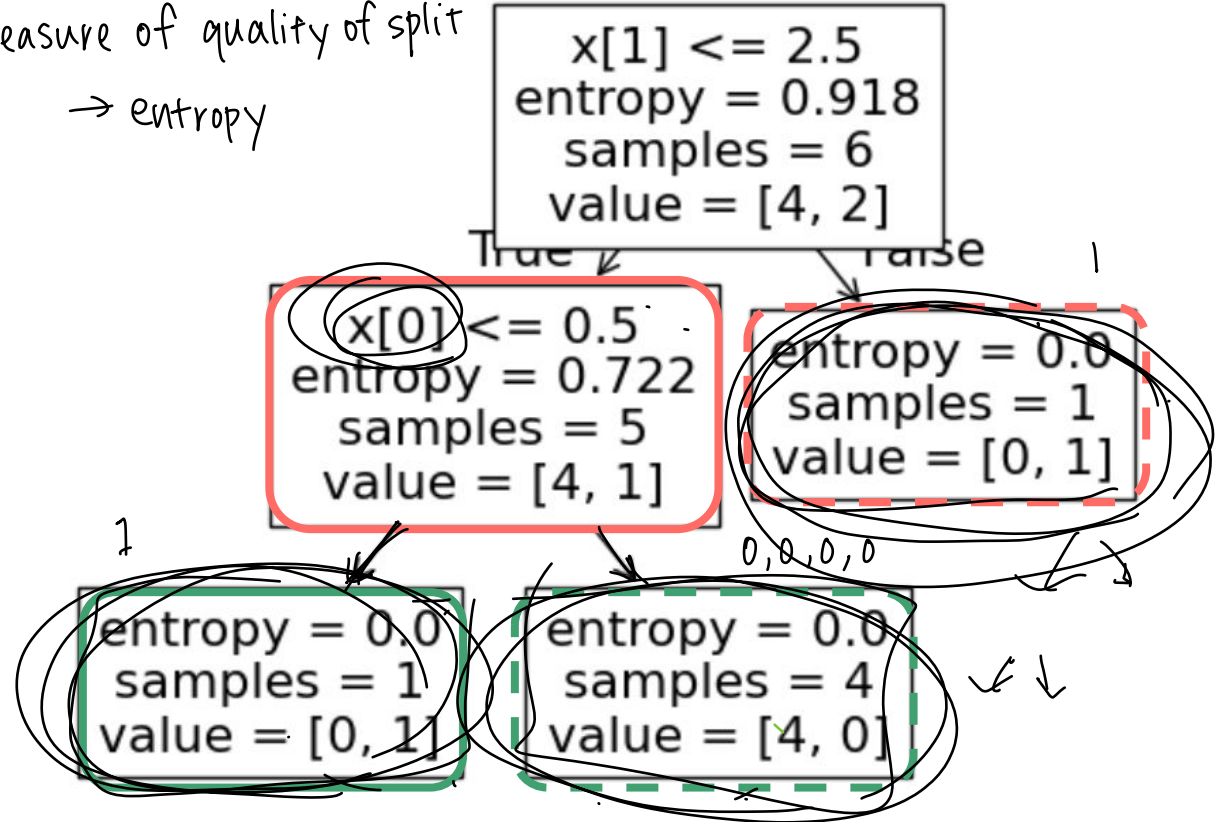


decision trees

```
dt = DecisionTreeClassifier(criterion='entropy') 'gini'  
dt.fit(data[['x0', 'x1']], data['y'])  
sklearn.tree.plot_tree(dt)  
plt.show()
```

	x0	x1	y
0	0	1	1
1	1	1	0
2	1	1	0
3	2	0	0
4	2	2	0
5	3	3	1

measure of quality of split
→ entropy



decision trees

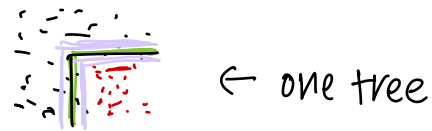
pros:

- Fast training and prediction.
- Ignores irrelevant features (will not choose a bad split).
- Not affected by different scales of data.

	target
1.5	1
1.6	0
1.8	0

test set:

1.4	1
1.55	1
1.7	0



cons:

- Complete tree almost always overfits to the training data (high variance).
- Not the best at predicting on unseen data.

random forest

- addresses the challenges of decision trees by introducing randomness.
- uses a random subset of features (also bootstrap by default).
- also able to find feature importance.

at each split

extra perk

tree

random forest

fits multiple trees, each with a random subset of features.

$n_{\text{estimator}} \approx 100$

	x0	x1	x2	y
--	----	----	----	---

0	0	1	3	1
---	---	---	---	---

1	1	1	3	0
---	---	---	---	---

2	1	1	2	0
---	---	---	---	---

3	2	0	2	0
---	---	---	---	---

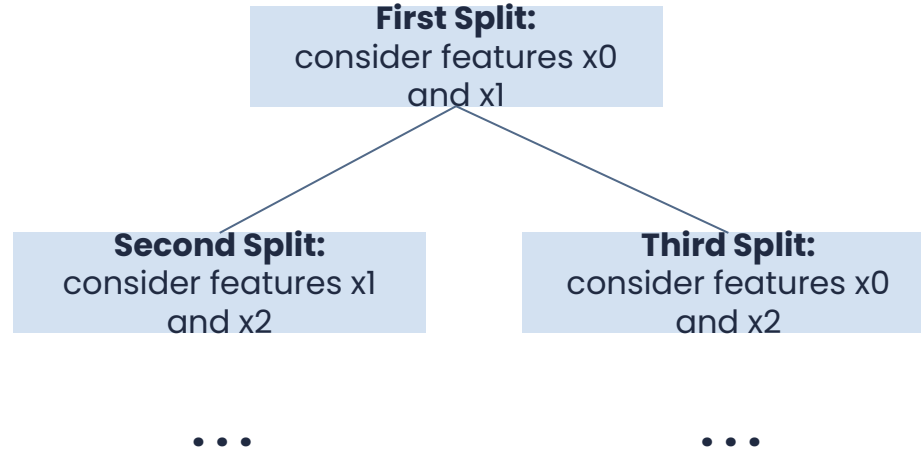
4	2	2	1	0
---	---	---	---	---

5	3	3	1	1
---	---	---	---	---

...

Bootstrap a subset of the data

→ fit tree with random subset of features at each split



→ repeat for n trees

classifier evaluation

Predicted Negative Predicted Positive

Actually Negative TN ✓

FP ✗

Actually Positive FN ✗

TP ✓

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

true positive rate

$$\text{recall} = \frac{TP}{TP + FN}$$

↑ all positive

$$\text{precision} = \frac{TP}{TP + FP}$$

← correct pos
← all predicted positive

Problem 12

Suppose you fit a decision tree to the training set below, using the features x_0 and x_1 to predict the outcome y .

x_0	x_1	y
0	0	0
0	1	0
0	2	1
1	0	0
1	1	1
2	0	1
2	1	1
3	0	1

Write the first four splitting rules that are created by the decision tree when fitting this training set (using weighted entropy). Assume that the tree is constructed in a depth-first order. If two candidate splits have the same weighted entropy, choose the one that splits on x_0 .

x0	x1	y
0	0	0
0	1	0
0	2	1
1	0	0
1	1	1
2	0	1
2	1	1
3	0	1

The first splitting rule is: (i) ___ <= ___ (ii) ___

(i):

x0

x1

(ii):

0

1

2

3

x0	x1	y
0	0	0
0	1	0
0	2	1
1	0	0
1	1	1
2	0	1
2	1	1
3	0	1

The second splitting rule is: (i) \leq (ii)

(i):

- x0
- x1

(ii):

- 0
- 1
- 2
- 3

x0	x1	y
0	0	0
0	1	0
0	2	1
1	0	0
1	1	1
2	0	1
2	1	1
3	0	1

The third splitting rule is: (i) \leq (ii)

(i):

- x0
- x1

(ii):

- 0
- 1
- 2
- 3

x0	x1	y
0	0	0
0	1	0
0	2	1
1	0	0
1	1	1
2	0	1
2	1	1
3	0	1

The fourth splitting rule is: (i) \leq (ii)

(i):

x0

x1

(ii):

0

1

2

3





model tuning



Bias vs. Variance

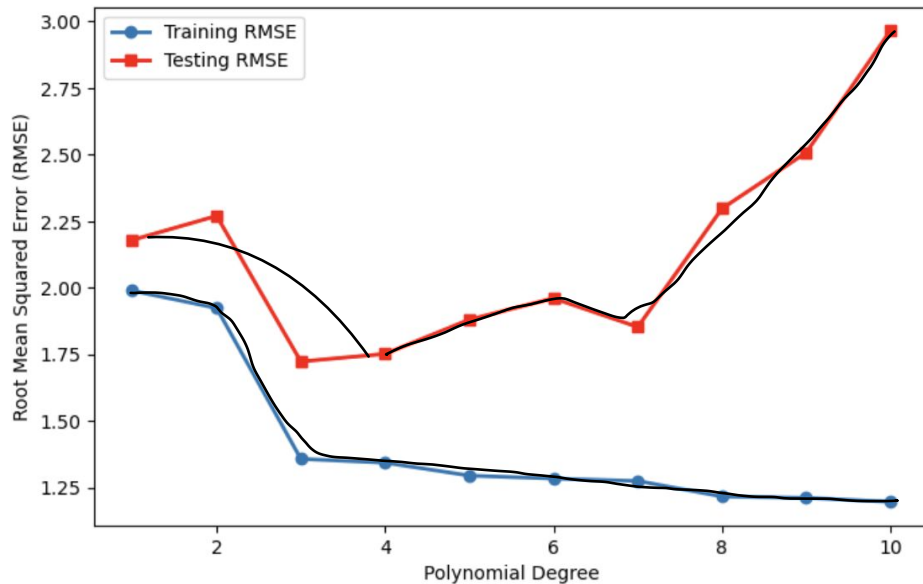
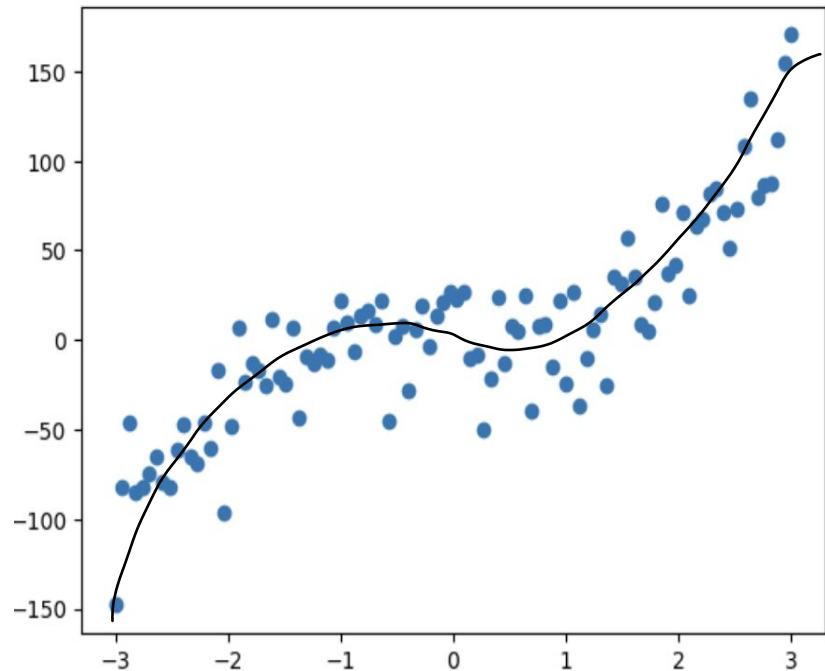
- **Bias: the expected deviation between a predicted value and the actual value**
 - low bias = good model
 - high bias = underfitting; the model fails to capture the complexity of the relationship between the features and the response variable.
- **Variance: how much does the prediction vary on different datasets**
 - low variance = good model
 - high variance = overfitting; the model is too complicated and is fitting too much of the noise in the training set

training error

testing error

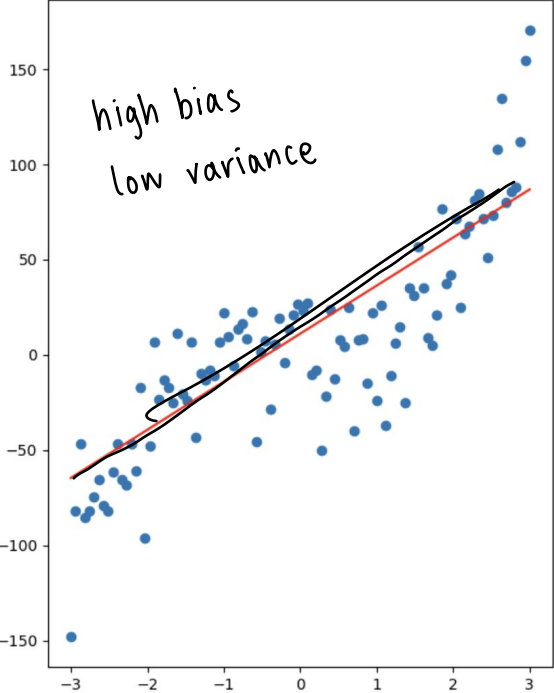
in the case of polynomial regression \neq multivariate LR

$$X^m + X^{m-1} + \dots + X^1$$

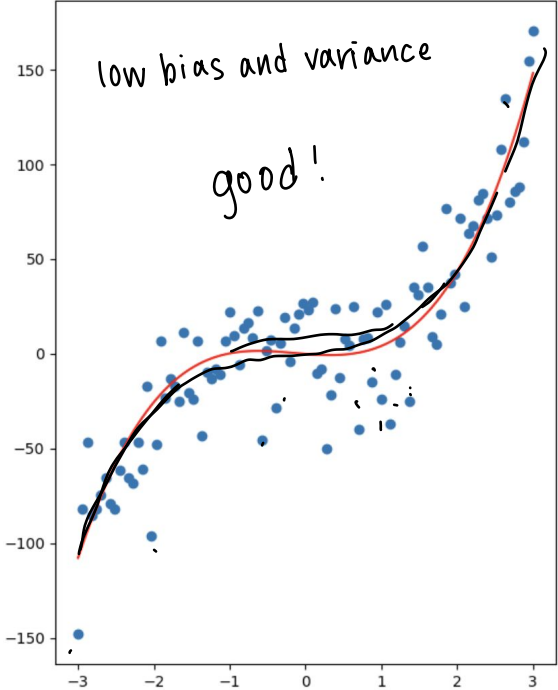


in the case of polynomial regression

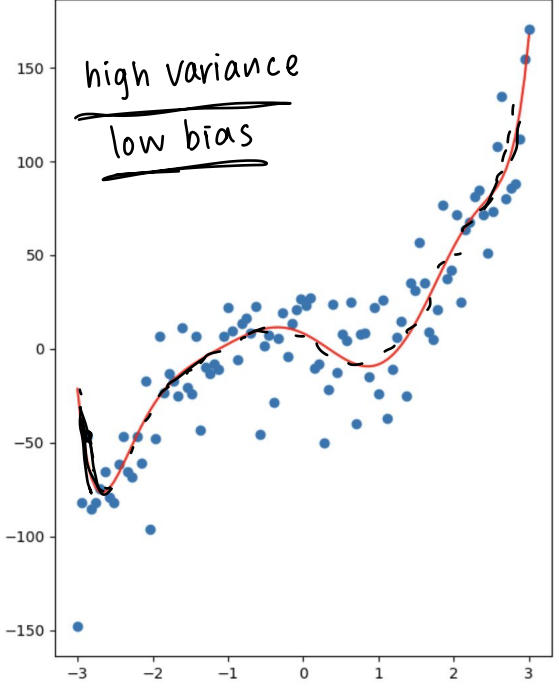
Degree = 1



Degree = 3



Degree = 8



examples of hyperparameters:

- **Decision trees**

- max tree depth

determines when to stop splitting; smaller = less overfitting

- min_samples_split

determines the minimum number of samples required in an internal node; larger = less overfitting

- criterion

'gini', 'entropy' etc.; different functions to measure quality of split

- **Random forest**

- all the hyperparameters for decision trees also apply

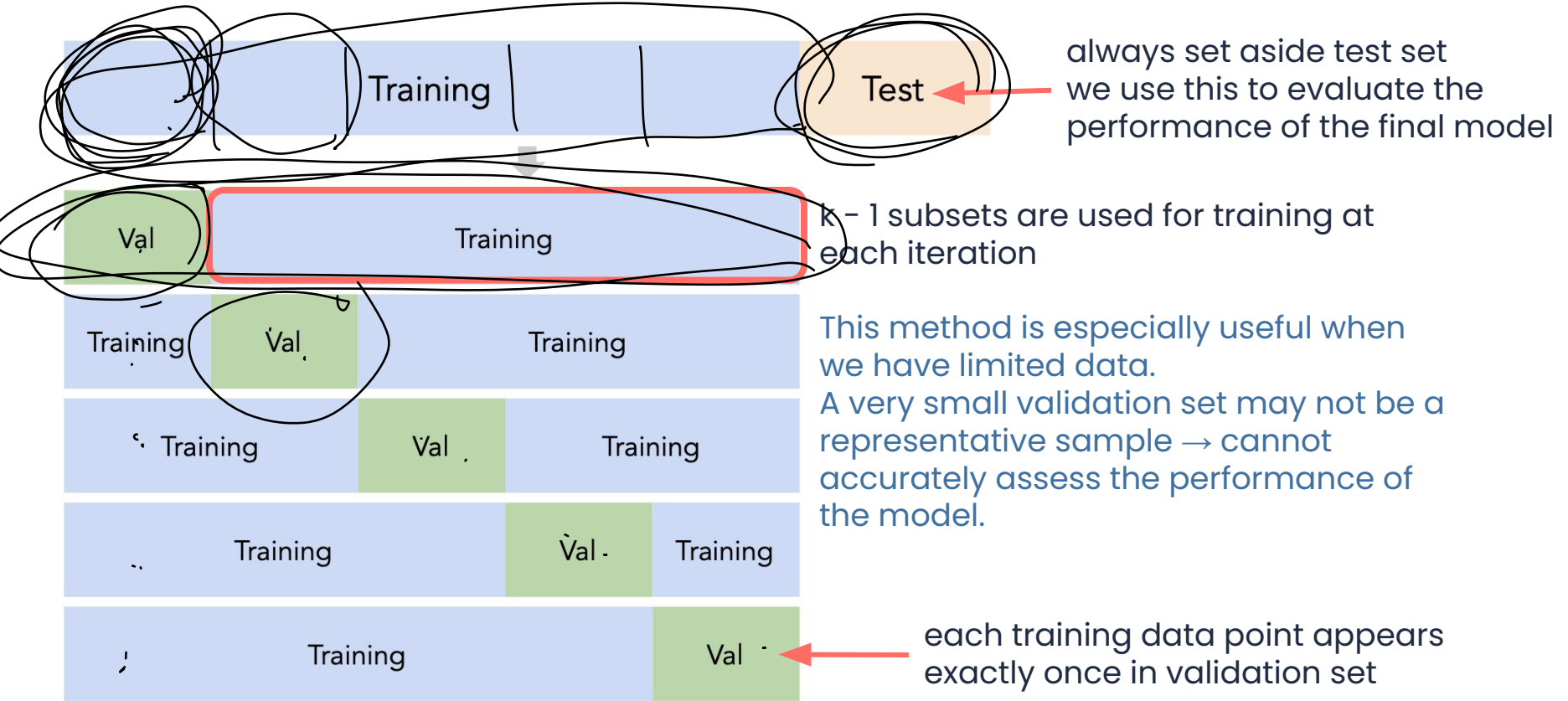
- n_estimators

how many trees to fit; more trees = less variance but more costly

- max_features

how many features to consider at each split $\sqrt{\text{total \# of features}}$

k-fold cross-validation



Suppose we write the following code:

```
hyperparameters = {  
    'n_estimators': [10, 100, 1000], # number of trees per forest  
    'max_depth': [None, 100, 10]    # max depth of each tree  
}  
grids = GridSearchCV(  
    RandomForestClassifier(), param_grid=hyperparameters,  
    cv=3, # 3-fold cross-validation  
)  
grids.fit(X_train, y_train)
```

Answer the following questions with a single number.

1. How many random forests are fit in total?
2. How many decision trees are fit in total?
3. How many times in total is the first point in X_train used to train a decision tree?

```
hyperparameters = {
  → 'n_estimators': [10, 100, 1000], # number of trees per forest
  'max_depth': [None, 100, 10] # max depth of each tree
} } 9 combos
```

Grid search with $k = 3$.

1. How many random forests are fit in total?

$$3 \text{ n_estimators} \times 3 \text{ max_depth} \times k=3 = 27 \text{ random forests}$$

2. How many decision trees are fit in total?

$$3 \text{ fold} \left(10 \text{ trees} \cdot 3 \text{ max_depth} + 100 \cdot 3 + 1000 \cdot 3 \right) = 3 \cdot 3330 = \underline{9990 \text{ trees}}$$

3. How many times in total is the first point in X_{train} used to train a decision tree?

$$(k-1) \cdot 3330 = 2 \cdot 3330 = 6660 \text{ times in training}$$

good luck on
the final!

CREDITS: This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)

Please keep this slide for attribution