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)

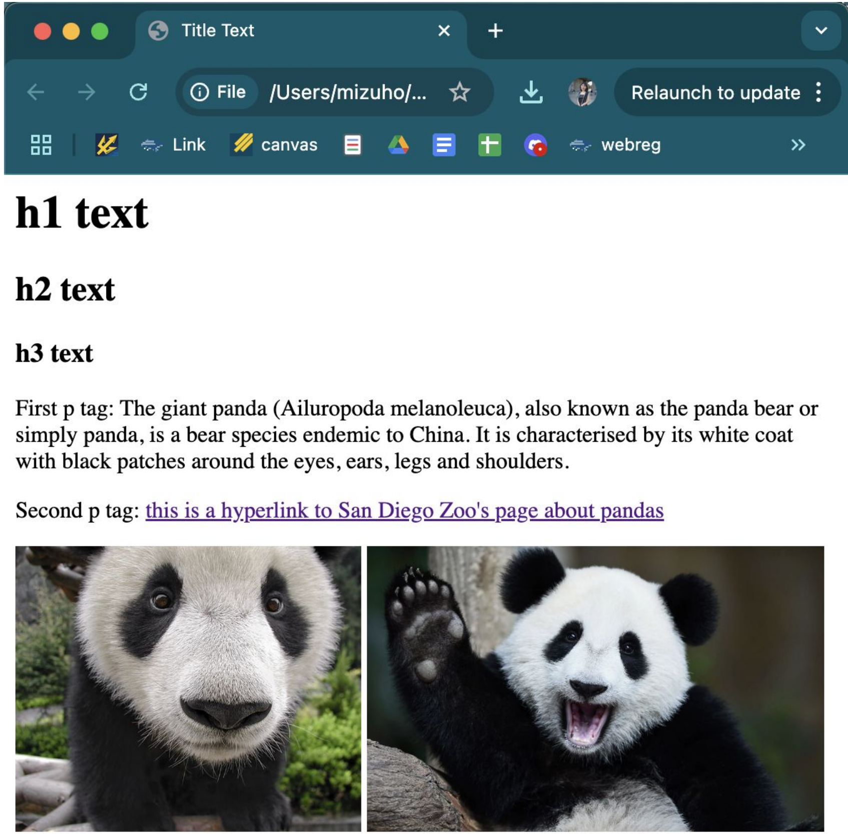
table of contents

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.
<span>	**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.
<img>	**image**: displays images. self-closing (no </img> tag required).

```html
<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>

        <img src="panda-closeup.jpg" alt="local image" height=200/>

        <img src="https://www.telegraph.co.uk/content/dam/news/2016/08/23/
        106598324PandawaveNEWS_trans_NvBQzQNjv4Bqeo_i_u9APj8RuoebjoAHt0k9u7HhRJvuo-ZLenGRumA.jpg?"
        alt="remote source" height=200/>

    </body>
</html>
```



🔗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
<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')`

regex and text features

regex functions

- `re.findall(pattern, 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

find more helpful examples here!

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

find more helpful examples here!

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

find more helpful examples here!

bag of words

→ defines a vector space in ℝ^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
0	hello world
1	good morning world
2	happy world

→

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

cosine similarity

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

for text 0 and text 1:

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} = \frac{[1,1,0,0,0]\cdot[0,1,1,1,0]}{\sqrt{2}\sqrt{3}} = \frac{1}{\sqrt{6}}$$

TF-IDF

- **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:

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

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

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

- **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 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 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?

- $X = Y$
- $X > Y > 0$
- $X > 0$ and $Y = 0$
- $X > 0$ and $Y > X$

linear regression

linear regression

Given prediction function:

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

Find the $w_0, w_1, ...w_m$ 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**.

$$w_0 = mean(y) = 3$$

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'):

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

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

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

```
▾ LinearRegression
LinearRegression()
```

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.

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

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

Prediction:

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

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

```
w0 =  -0.2059659090909062
w1 =  2.627840909090909
```

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
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

y =

	y
0	5
1	4
2	2
3	1
4	3

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 ?

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

```
▾ LinearRegression
LinearRegression()
```

Training step:

- Fitting on the entire X matrix

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

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

Prediction:

- Predicting on entire X matrix

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

```
w0 =  0.11037380241869155
w1 =  -0.284657813156235
w2 =  2.7418721532904042
w3 =  -0.0022066907491752833
```

Best parameters:

- w0 is the intercept
- w1, w2, and w3 are the coefficients for each feature of X.

how good is my linear regression model?

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

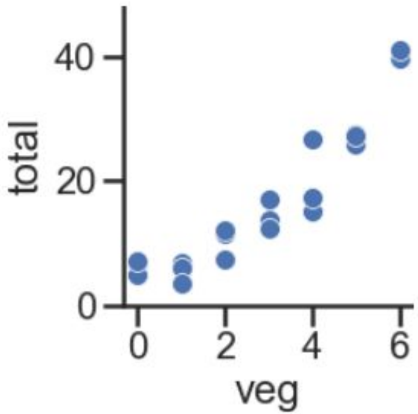
Lower
→ actual y is close to predicted y on average
= good prediction

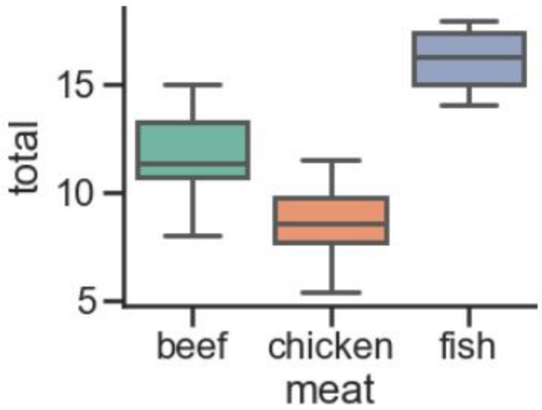
$$R^2 = \frac{\text{var(predicted } y \text{ values)}}{\text{var(actual } y \text{ values)}}$$

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 $\text{meat}{=}\text{beef}$ 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, $\text{meat}{=}\text{chicken}$ and $\text{meat}{=}\text{fish}$ 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. $H(x) = w_0 + w_1 \cdot \text{veg}$

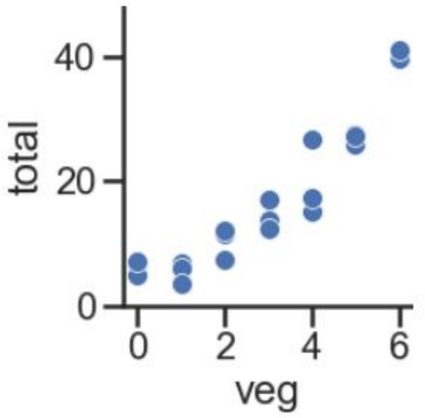
iii. $H(x) = w_0 + w_1 \cdot (\text{meat}{=}\text{chicken})$

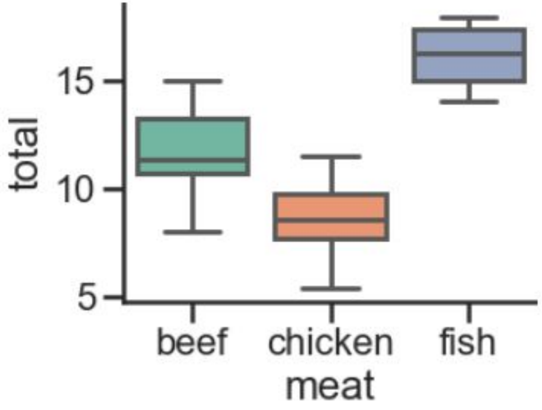
iv. $H(x) = w_0 + w_1 \cdot (\text{meat}{=}\text{beef}) + w_2 \cdot (\text{meat}{=}\text{chicken})$

v. $H(x) = w_0 + w_1 \cdot (\text{meat}{=}\text{beef}) + w_2 \cdot (\text{meat}{=}\text{chicken}) + w_3 \cdot (\text{meat}{=}\text{fish})$

DSC80 FA23 Final Question 9

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

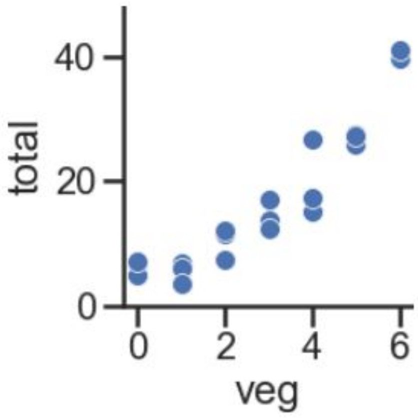


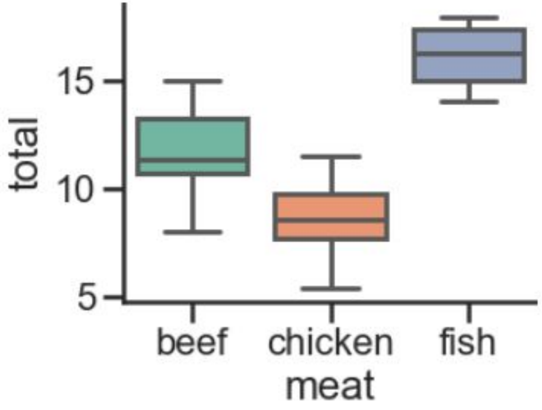


$$H(x) = w_0$$

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

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





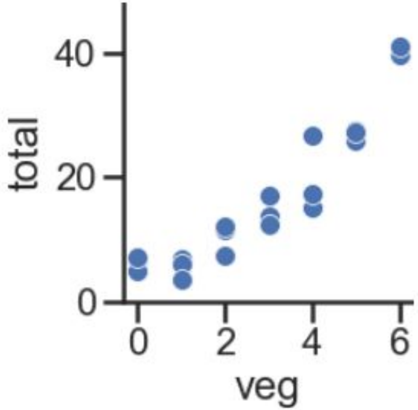
$$H(x) = w_0 + w_1 \cdot \textbf{veg}$$

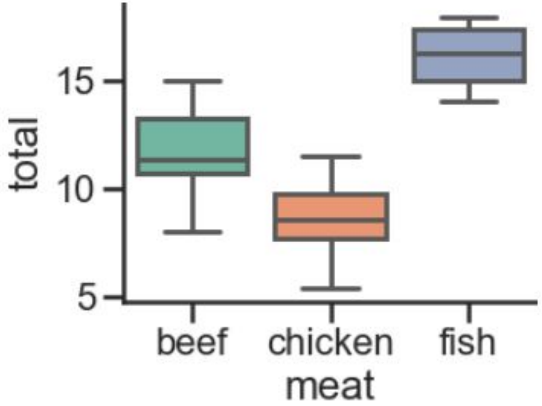
w0: positive / negative / 0 / need more information ?

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

DSC80 FA23 Final Question 9

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



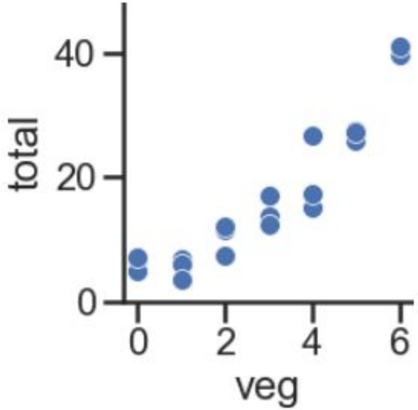


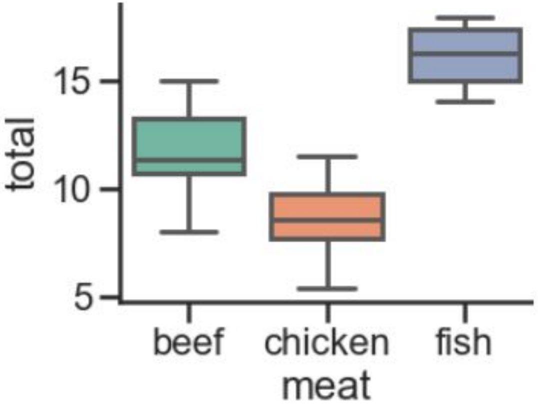
$$H(x) = w_0 + w_1 \cdot (\text{meat=chicken})$$

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

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

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





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

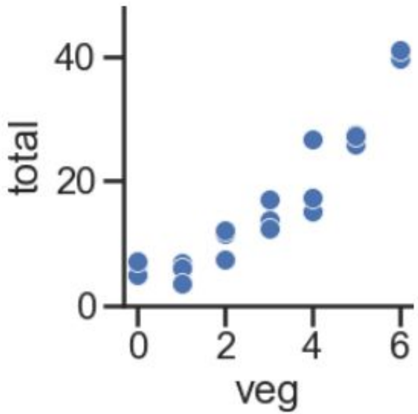
w0: positive / negative / 0 / need more information ?

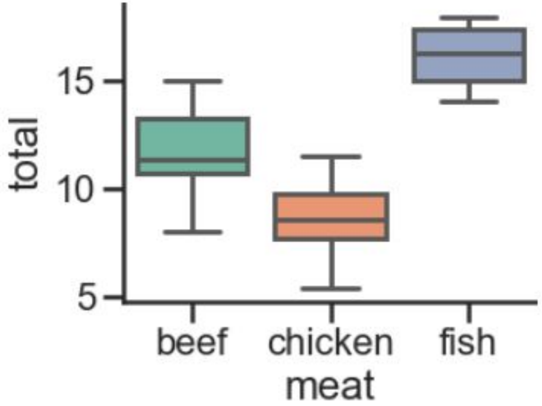
w1: positive / negative / 0 / need more information ?

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

DSC80 FA23 Final Question 9

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





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

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 ?

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:**
  - $$x_i \to \frac{x_i - \bar{x}}{\sigma_x}$$
  - 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

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

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

```
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!

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)
```

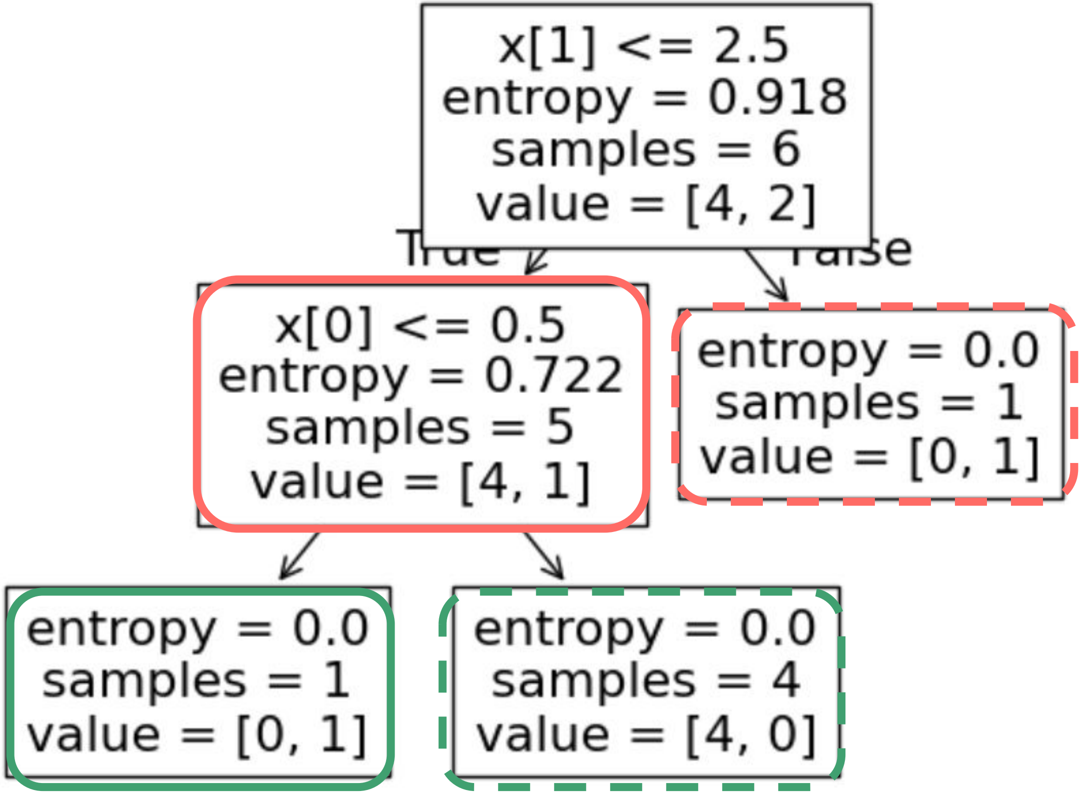
```
array([[0., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 0.],
       [0., 1.]])
```

classification

decision trees

	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

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



decision trees

pros:

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

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.

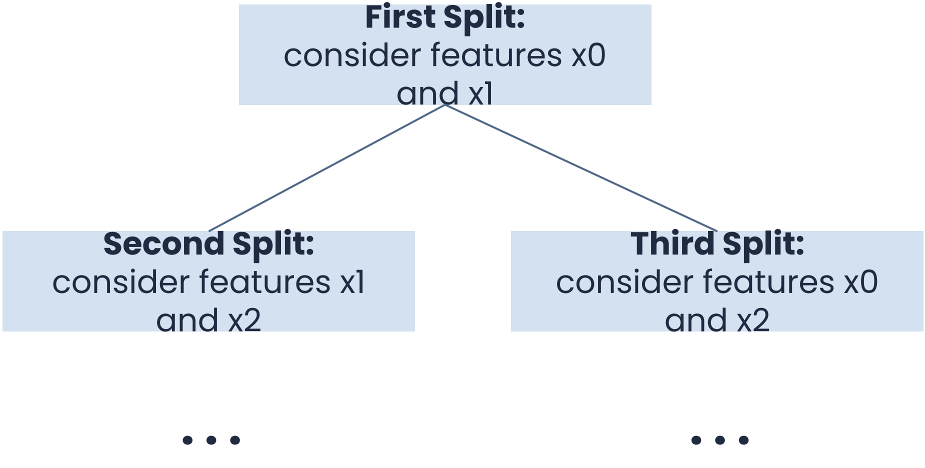
random forest

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

	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 ✅

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

$$\text{recall} = \frac{TP}{TP + FN} \qquad \text{precision} = \frac{TP}{TP + FP}$$

Problem 12

Suppose you fit a decision tree to the training set below, using the features `x0` and `x1` to predict the outcome `y`.

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

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 `x0`.

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)___ <= ___(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)___ <= ___(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)___ <= ___(ii)___`

(i):

- ○ `x0`
- ○ `x1`

(ii):

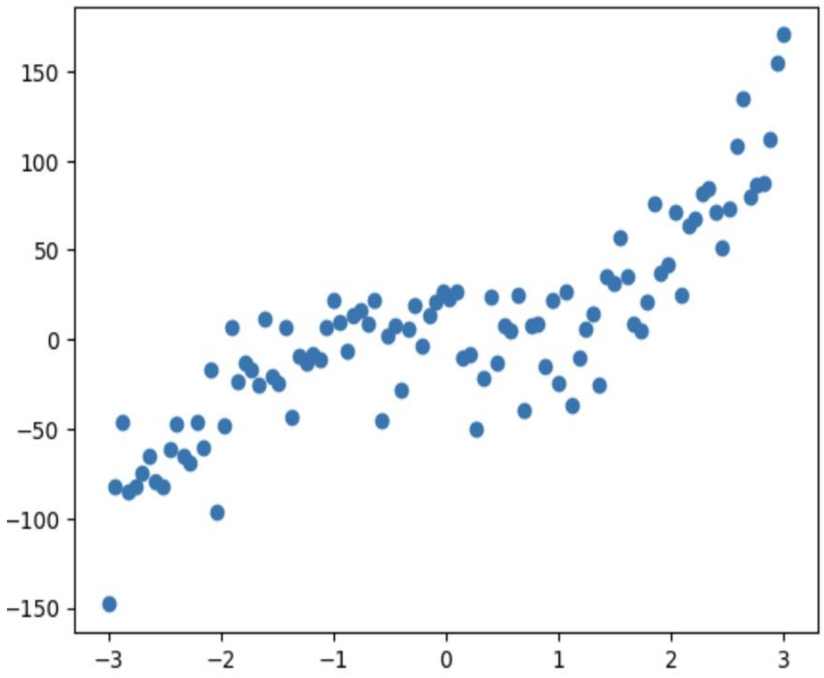
- ○ `0`
- ○ `1`
- ○ `2`
- ○ `3`

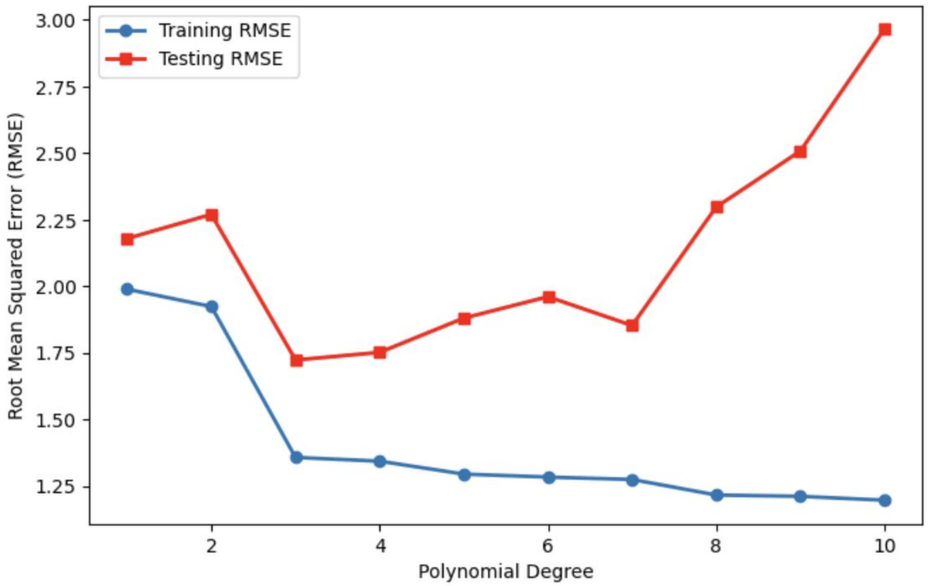
model tuning

Bias vs. Variance

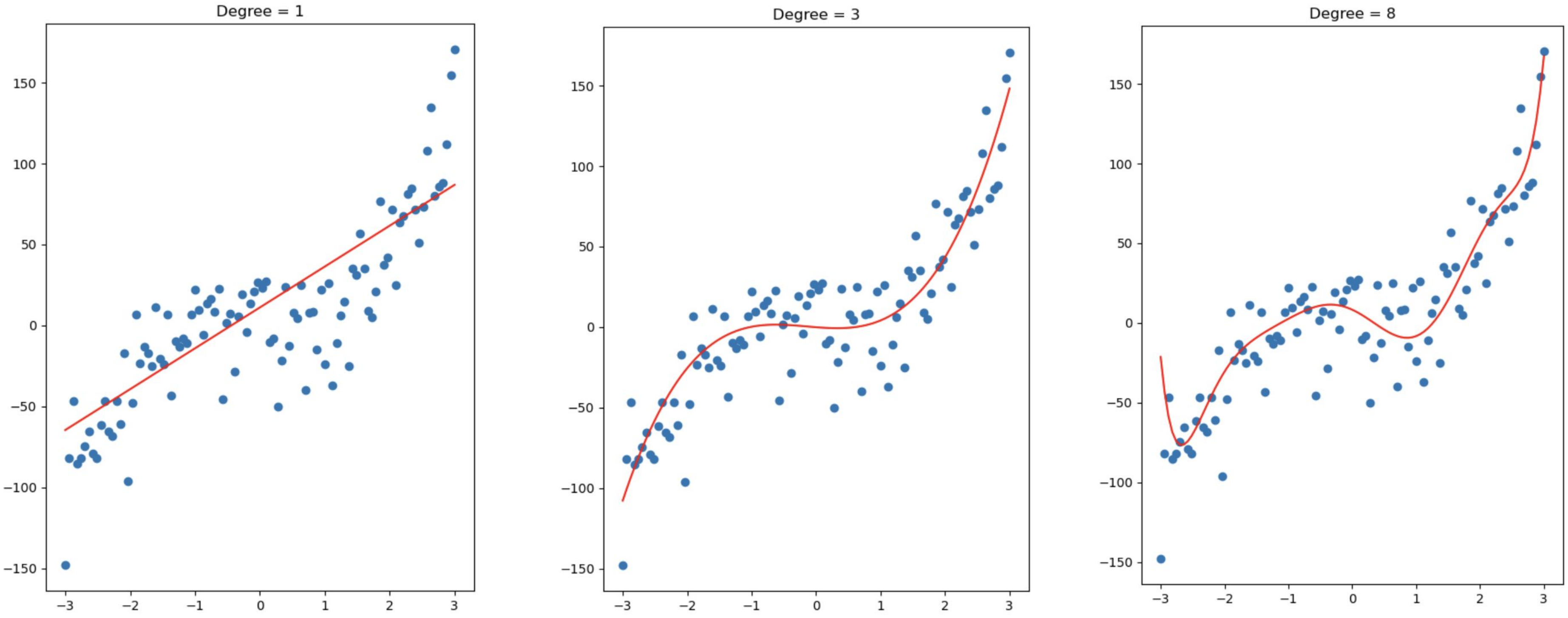
- **Bias: the expected deviation between a predicted value and the actual value**
  - low bias = good model
  - high bias = underfitting; the model fails to capture the complexity of the relationship between the features and the response variable.
- **Variance: how much does the prediction vary on different 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

in the case of polynomial regression





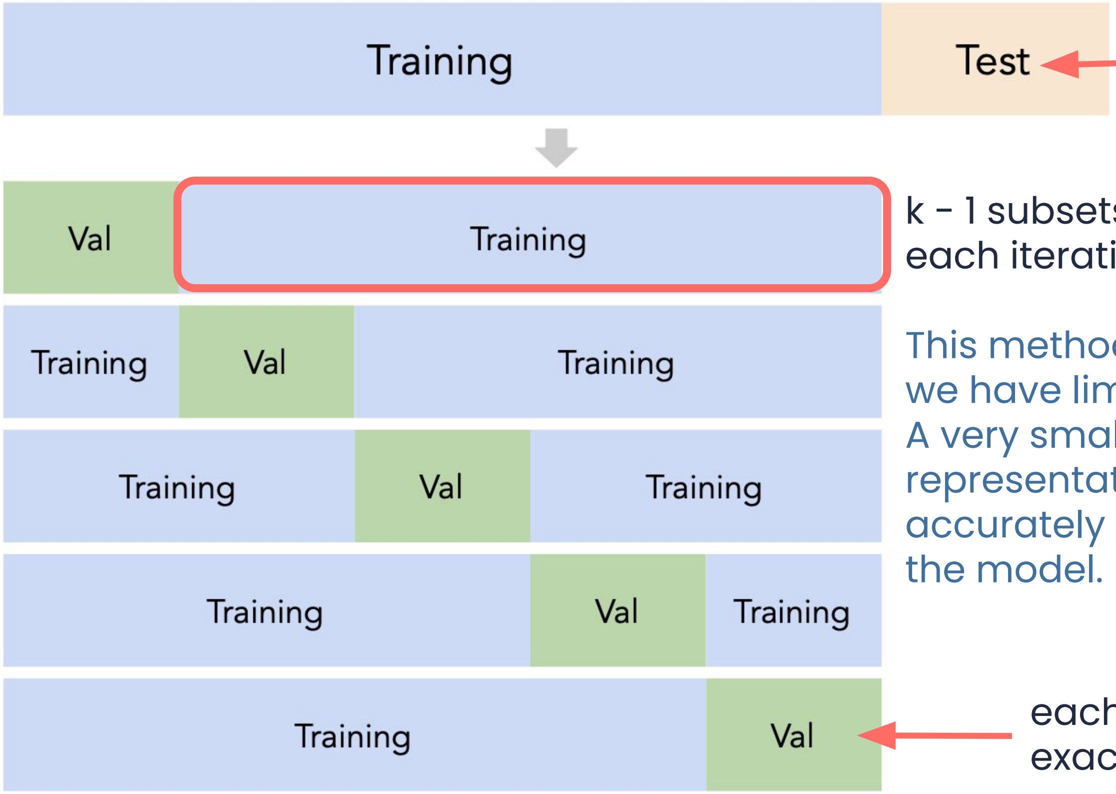
in the case of polynomial regression



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

k-fold cross-validation



always set aside test set
we use this to evaluate the performance of the final model

k - 1 subsets are used for training at each iteration

This method is especially useful when we have limited data.
A very small validation set may not be a representative sample → cannot accurately assess the performance of the model.

each training data point appears exactly once in validation set

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?

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

Grid search with k = 3.

1. How many random forests are fit in total?

1. How many decision trees are fit in total?

2. How many times in total is the first point in X_train used to train a decision tree?

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