W6 - KNN

$y = f(x) + \epsilon, \quad \epsilon \sim N(0, \sigma_\epsilon^2)$

$\hat{y} = \hat{f}(x), \quad \hat{f}(x_i) = \frac{1}{K} \sum_{i \in \mathcal{K}_x} f(x_i) + \epsilon_i$

$\text{Bias}^2 = (f(x) - \frac{1}{K} \sum_{i \in \mathcal{K}_x} f(x_i))^2$

$\text{Var}(\hat{y}) = E[(\frac{1}{K} \sum_{i \in \mathcal{K}_x} \epsilon_i)^2] = \frac{\sigma_\epsilon^2}{K}$

$\text{Euclidean}(L^2): \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}$

$\text{Manhattan}(L1): \sum_{i=1}^{d} |a_i - b_i|$

Subsets to consider in exhaustive search

[0,1,2,3] [0,1,2] [0,1,3] [0,2,3] [1,2,3] [0,1] [0,2] [0,3] [1,2] [1,3] [2,3] [0] [1] [2] [3]

Subsets to consider in naive search

[0,1,2,3] → [0,1,2] [0,1,3] [0,2,3] [1,2,3] → [0,1] [0,2] [0,3] [1,2] [1,3] [2,3] → [0] [1] [2] [3]

Subsets to consider in sequential forward search (example)

[0,1,2,3] → [0,1,2] [0,1,3] [0,2,3] [1,2,3] → [0,1] [0,2] [0,3] [1,2] [1,3] [2,3] → [0] [1] [2] [3]

W7 — Decision trees and ensembles

$\text{GINI index}: \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$

$\hat{p}_{mk}$ is the proportion of training samples in R_m belong to class k.

$\text{Entropy}: -\sum_{k=1}^{K} \hat{p}_{mk} \log_2 \hat{p}_{mk}$

$\text{Conditional entropy}: \hat{E}(S|X) = \sum_{V} \frac{|S_V|}{|S|} \hat{E}(S_V)$

$\text{Information gain}: G(S,X) = \hat{E}(S) - \hat{E}(S|X)$

	name	predator	milk	domestic	feathers	toothed	category
0	piranha	1	0	0	0	1	fish
1	herring	1	0	0	0	1	fish
2	sole	0	0	0	0	1	fish
3	haddock	0	0	0	0	1	fish
4	skimmer	1	0	0	1	0	bird
5	parakeet	0	0	1	1	0	bird
6	catfish	1	0	0	0	1	fish
7	wren	0	0	0	1	0	bird
8	vulture	1	0	0	1	0	bird
9	flamingo	0	0	0	1	0	bird

First, we compute the entropy for the entire set:

$$S = \{5 \text{ bird}, 5 \text{ fish}\}, |S| = 10$$

$$E(S) = -\frac{5}{10}\log_2\frac{5}{10} - \frac{5}{10}\log_2\frac{5}{10} = 1$$

Note that entropy is maximized ($E = 1$) when the data is evenly divided between the two classes.

Now, we will compute the conditional entropy and the information gain for each of the features.

predator:

Considering the predator $= 0$ branch:

$$S_0 = \{3 \text{ bird}, 2 \text{ fish}\}, |S_0| = |S|$$

$$E(S_0) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.97095$$

Considering the predator $= 1$ branch:

$$S_1 = \{2 \text{ bird}, 3 \text{ fish}\}, |S_1| = |S|$$

$$E(S_1) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97095$$

Conditional entropy and information gain of splitting on predator:

$$E[S|\text{ predator}) = \frac{5}{10}E(S_0) + \frac{5}{10}E(S_1) = 0.97095$$

$$G(S,\text{ predator}) = E(S) - E[S|\text{ predator}) = 1 - 0.97095 = 0.02905$$

Feature	Conditional entropy	Information gain
predator	$E[S\|\text{ predator}) = 0.971$	$G(S,\text{ predator}) = 0.029$
milk	$E[S\|\text{ milk}) = 1.000$	$G(S,\text{ milk}) = 0.000$
domestic	$E[S\|\text{ domestic}) = 0.892$	$G(S,\text{ domestic}) = 0.108$
feathers	$E[S\|\text{ feathers}) = 0.000$	$G(S,\text{ feathers}) = 1.000$
toothed	$E[S\|\text{ toothed}) = 0.000$	$G(S,\text{ toothed}) = 1.000$

n↑ → more complex

① Decision tree by depth. depth↑ → bias↓, Var↑

② Decision tree by pruning parameter

$\alpha \uparrow \rightarrow$ bias↑, Var↑ less complex

③ Decision tree (A) vs. Bagged trees (B)

Bias (A) = Bias (B), Var(A) > Var(B)

④ Bagged trees by number of estimators

Ensemble↑ → Var↓

⑤ Bagged trees (A) vs. Random forest (B)

Var(A) > Var(B)

⑥ Decision tree (A) VS. AdaBoost (B)

Bias (A) > Bias (B), Var(A) > Var(B)

⑦ AdaBoost by number of iterations

iteration↑ → Bias↓, Var↓

Bagged trees: highly correlated;

Random forest: much less correlated

AdaBoost: output will be a weighted average of the predictions of all three trees

W8 - Support vector classifiers and kernels

$\underset{w,\epsilon}{\text{minimize}} \ \frac{1}{2}\sum_{j=1}^{p} w_j^2 + C\sum_{i=1}^{n} \epsilon_i$

$\text{Subject to } y_i[w_0 + \sum_{j=1}^{p} w_j x_{ij}] \geq 1 - \epsilon_i, \forall i$

$\epsilon_i \geq 0, \forall i$

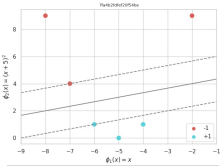
C↑ → Bias↓, Var↑

$\text{RBF parameter}: K(x,z) = \exp(-\frac{\|x-z\|_2^2}{\sigma^2})$

σ — bandwidth $= \exp(-\gamma\|x-z\|_2^2)$

σ↑ → γ↓ → Bias↑, Var↓ less complex

Sample index	x	y
1	-7	-1
2	-6	1
3	-5	1
4	-4	1
5	-8	-1
6	-2	-1



Comment The slope of the line is 1/3, and the intercept is 14/3.

The hyperplane is defined by $\phi_2(x) = 1/3\ \phi_1(x) + 14/3$, or equivalently $14 + 1\phi_2(x) + -3\phi_1(x) = 0$ (or any scaled version of this expression).

Part 2: Express the objective of the dual problem

To find α_i for the maximal margin classifier, we will solve the dual problem

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_{i'} \alpha_i\alpha_{i'} y_i y_{i'} \phi(x_i)^T \phi(x_{i'}) \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned}$$

(note: this maximal margin classifier problem is slightly different from the support vector classifier problem! There is no slack variable for the maximal margin classifier)

We know that $\alpha_i = 0$ for any sample that is not a support vector - in this case, only $i = 1, 2$ will have non-zero values. So, write out the expression

$$\sum_i \alpha_i - \frac{1}{2}\sum_i\sum_{i'} \alpha_i\alpha_{i'} y_i y_{i'} \phi(x_i)^T \phi(x_{i'})$$

in terms of α_1 and α_2, the support vectors x_1 and x_2, and their labels y_1 and y_2. (Note that $\phi(x_1)$ and $\phi(x_2)$ are vectors. The others are scalars.) In this expression, i and j are sample indices.

Then substitute the values of x_1 and x_2, y_1 and y_2, and simplify, so that the only unknowns are α_1 (alpha_1) and α_2 (alpha_2).

$-\frac{49\alpha_1^2}{2} + 46\alpha_1\alpha_2 + \alpha_1 - \frac{37\alpha_2^2}{2} + \alpha_2$

Comment

$$\begin{aligned} &\alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1\alpha_1 y_1 y_1 \phi(x_1)^T\phi(x_1) + 2\alpha_1\alpha_2 y_1 y_2 \phi(x_1)^T\phi(x_2) + \alpha_2\alpha_2 y_2 y_2 \phi(x_2)^T\phi(x_2)\right) \\ &\alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1^2\phi(x_1)^T\phi(x_1) - 2\alpha_1\alpha_2\phi(x_1)^T\phi(x_2) + \alpha_2^2\phi(x_2)^T\phi(x_2)\right) \\ &\alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1^2[-7,4]^T[-7,4] - 2\alpha_1\alpha_2[-7,4]^T[-6,1] + \alpha_2^2[-6,1]^T[-6,1]\right) \\ &\alpha_1 + \alpha_2 - \frac{1}{2}\left(65\alpha_1^2 - 92\alpha_1\alpha_2 + 37\alpha_2^2\right) \end{aligned}$$

Part 3: Use the constraint to simplify

Note that for our example, since there is one support vector in the positive class and one support vector in the negative class, the first constraint of the dual problem described above,

$$\sum_i \alpha_i y_i = 0$$

implies that $\alpha_1 = \alpha_2$.

In your expression for the objective in the previous part, substitute $\alpha_1 = \alpha$ and $\alpha_2 = \alpha$, and simplify. Your answer should be an expression in terms of α (alpha).

$-5\alpha^2 + 2\alpha$

Comment With $\alpha_1 = \alpha_2 = \alpha$, the expression

$$\alpha_1 + \alpha_2 - \frac{1}{2}\left(65\alpha_1^2 + -92\alpha_1\alpha_2 + 37\alpha_2^2\right)$$

becomes

$$\alpha + \alpha - \frac{1}{2}\left(65\alpha^2 + -92\alpha^2 + 37\alpha^2\right) = 2\alpha - 5\alpha^2$$

Part 4: Solve by setting the derivative to zero

Take the derivative of the expression from the previous part with respect to α.

$\frac{d}{d\alpha} = 2 - 10\alpha$

Then, set it equal to zero. Solve to find the value of $\alpha_1 = \alpha_2 = \alpha$.

$\alpha = 0.200$

Find the optimal values of the coefficients of the separating hyperplane, using

$$w_j = \sum_{i \in S} \alpha_i y_i \phi_j(x_i), j = 1, 2$$

where i is a sample index and j indexes the columns (in the transformed space defined by $\phi(x)$).

$w_1^* = 0.200$

$w_2^* = -0.600$

Comment

$$\frac{d}{d\alpha} 2\alpha - 5\alpha^2 = 0 \to 2 - (2)(5)\alpha = 0 \to \alpha = \frac{1}{5}$$

Then,

$$w_1 = (\tfrac{1}{5})(-1)(-7) + (\tfrac{1}{5})(1)(-6) = \frac{1}{5}$$

and

$$w_2 = (\tfrac{1}{5})(-1)(4) + (\tfrac{1}{5})(1)(1) = \frac{-3}{5}$$

Part 5: Find w_0

To find w_0, compute:

$$w_0 = \frac{1}{2}\sum_{i \in S_v} y_i - \langle [w_1, w_2], \phi(x_i) \rangle$$

where i is a sample index, and you use the w_1 and w_2 you found above.

$w_0^* = 2.800$

Comment For either support vector, $y_i - \langle [w_1, w_2], \phi(x_i) = w_0$.

For the first support vector:

$$-1 - [(\tfrac{1}{5})(-7) + (\tfrac{-3}{5})(4)] = 14/5$$

For the second support vector:

$$1 - [(\tfrac{1}{5})(-6) + (\tfrac{-3}{5})(1)] = 14/5$$

W9 - Neural networks

d. `categorical_crossentropy` Labels: `[0, 0, 1], [1, 0, 0], [0, 1, 0]`, indicating that these samples belong to class 2, class 0, and class 1, respectively.

b. `mean_squared_error function` Labels: `2, 0, 1`, each representing the value of some continuous quantity.

a. `sparse_categorical_crossentropy` Labels: `2, 0, 1`, indicating that these samples belong to class 2, class 0, and class 1, respectively.

c. `binary_crossentropy` Labels: `1, 0, 1`, each representing a binary class label.

Part 4: Update weights

Compute the updated weights for both the hidden layer and the output layer by performing one step of gradient descent. Use a learning rate of 0.4.

$W_{o1} = [w_{h1,o1}, w_{h2,o1}, w_{bh,o1}] =$ [2.058, -0.449, -1.942]

$W_{h1} = [w_{x1,h1}, w_{x2,h1}, w_{bh,h1}] =$ [1.500, 2.001, 2.501]

$W_{h2} = [w_{x1,h2}, w_{x2,h2}, w_{bh,h2}] =$ [-2.500, -1.003, 2.997]

Part 4: Update weights

1. **更新输出层权重 W_{o1}**
   - 学习率 $\alpha = 0.4$
   - $w_{h_1 o_1}^{new} = w_{h_1 o_1} - \alpha \frac{\partial L}{\partial w_{h_1 o_1}} = -2 - 0.4 \times (-0.144) = -2 + 0.0576 = 2.058$
   - $w_{h_2 o_1}^{new} = w_{h_2 o_1} - \alpha \frac{\partial L}{\partial w_{h_2 o_1}} = -0.5 - 0.4 \times (-0.128) = -0.5 + 0.0512 = -0.4$
   - $w_{bo_1}^{new} = w_{bo_1} - \alpha \frac{\partial L}{\partial w_{bo_1}} = -2 - 0.4 \times (-0.145) = -2 + 0.058 = -1.942$
   - 所以 $W_{o1} = [2.058, -0.449, -1.942]$
2. **更新隐藏层 h_1 的权重 W_{h1}**
   - $w_{x_1 h_1}^{new} = w_{x_1 h_1} - \alpha \frac{\partial L}{\partial w_{x_1 h_1}} = -1.5 - 0.4 \times 0.000 = -1.5$
   - $w_{x_2 h_1}^{new} = w_{x_2 h_1} - \alpha \frac{\partial L}{\partial w_{x_2 h_1}} = 2 - 0.4 \times (-0.003) = 2 + 0.0012 = 2.001$
   - $w_{bh_1}^{new} = w_{bh_1} - \alpha \frac{\partial L}{\partial w_{bh_1}} = 2.5 - 0.4 \times (-0.003) = 2.5 + 0.0012 = 2.501$
   - 所以 $W_{h1} = [1.500, 2.001, 2.501]$
3. **更新隐藏层 h_2 的权重 W_{h2}**
   - $w_{x_1 h_2}^{new} = w_{x_1 h_2} - \alpha \frac{\partial L}{\partial w_{x_1 h_2}} = -2.5 - 0.4 \times 0.000 = -2.5$
   - $w_{x_2 h_2}^{new} = w_{x_2 h_2} - \alpha \frac{\partial L}{\partial w_{x_2 h_2}} = -1 - 0.4 \times 0.008 = -1 - 0.0032 = -1.003$
   - $w_{bh_2}^{new} = w_{bh_2} - \alpha \frac{\partial L}{\partial w_{bh_2}} = 3 - 0.4 \times 0.008 = 3 - 0.0032 = 2.997$
   - 所以 $W_{h2} = [-2.500, -1.003, 2.997]$

W10 - Deep learning, CNN

Suppose an image with dimension 224 × 224 and 3 color channels is passed to the first layer of a neural network.

How many weights must be learned between the input layer and the first hidden layer if it is...

... a fully connected layer with 128 units?

224×224×3×128

Weights between input units and hidden units in first layer: 19267584 ✓ 100%

+

Bias terms to hidden units in first layer: 128 ✓ 100%

... a convolutional layer with 128 filters, each having filter size 3 × 3, stride 1, and 1 padding?

Weights between input units and hidden units in first layer: 3456 ✓ 100%

3×3×3×128

+

Bias terms to hidden units in first layer: 128 ✓ 100%

H10.5. Fine-tune a pre-trained model

You are developing a machine learning model to classify images of humans in various poses as one of:

```
['sitting', 'standing', 'lying', 'dancing']
```

You have loaded your training dataset into the variable `xtr`, and the associated labels into the variable `ytr`. Each image has dimension (224, 224, 3). The class labels are given as integers (i.e. as an index for the array of labels shown above).

You decide to use a pre-trained model trained on general image data, and add a new classification head for this specific task. (You'll also include a dropout layer between the pre-trained base model and your classification head, for regularization.) You'll train this for 20 epochs. After training the classification head, you will also fine-tune the last layers of the pre-trained model for the specific task, for another 10 epochs.

Use the blocks below to construct and train your classifier.

Correct answer (there may be other correct orders):

```
INPUT_IMG_SHAPE = (224, 224, 3)
b = tf.keras.applications.MobileNetV2(
  input_shape=INPUT_IMG_SHAPE,
  include_top=False,
  pooling='avg'
)

b.trainable = False

m = tf.keras.models.Sequential()

m.add(b)

m.add(tf.keras.layers.Dropout(0.5))

m.add(tf.keras.layers.Dense(
    units=5,
    activation=tf.keras.activations.softmax
))

# Longer learning rate
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
m.compile(
    optimizer=opt,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)

m.fit(Xtr, ytr, epochs=20)

b.trainable = True
for layer in b.layers[:140]:
    layer.trainable = False

# Smaller learning rate
opt = tf.keras.optimizers.Adam(learning_rate=0.00001)
m.compile(
    optimizer = opt,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)

m.fit(Xtr, ytr, epochs=10)
```

H10.4. Build a model for transfer learning

You are developing a machine learning model to classify images of flowers as one of:

```
['rose',
 'daisy',
 'lily',
 'daffodil',
 'lilac',
 'violet',
 'petunia',
 'iris',
 'sunflower']
```

You have loaded your training dataset into the variable `xtr`, and the associated labels into the variable `ytr`. Each image has dimension (224, 224, 3). The class labels are given as integers (i.e. as an index for the array of labels shown above).

You decide to use a pre-trained model trained on general image data, and add a new classification head for this specific task. (You'll also include a dropout layer between the pre-trained base model and your classification head, for regularization.)

Use the blocks below to construct and train your classifier.

Correct answer (there may be other correct orders):

```
INPUT_IMG_SHAPE = (224, 224, 3)
b = tf.keras.applications.MobileNetV2(
  input_shape=INPUT_IMG_SHAPE,
  include_top=False,
  pooling='avg'
)

b.trainable = False

m = tf.keras.models.Sequential()

m.add(b)

m.add(tf.keras.layers.Dropout(0.5))

m.add(tf.keras.layers.Dense(
    units=5,
    activation=tf.keras.activations.softmax
))

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
m.compile(
    optimizer=opt,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)

m.fit(Xtr, ytr, epochs=20)
```

Incorrect Blocks (these blocks should not be included in your solution):

```
INPUT_IMG_SHAPE = (224, 224, 3)
b = tf.keras.applications.MobileNetV2(
  input_shape=INPUT_IMG_SHAPE
)

b.trainable = True

m.add(tf.keras.layers.Dense(
    units=5,
    activation=tf.keras.activations.softmax
))
```

W11 - Unsupervised learning

K means clustering

H11.1. K means clustering

Given these initial cluster centers: $C_1 = (-3, -3)$ and $C_2 = (3, 3)$, assign each of these points to their nearest cluster. Use Euclidean distance, and in case of a tie, assign the point to the cluster with the smaller index.

x_1	x_2	cluster	
0	3	1	✓ 100%
2	-5	2	✓ 100%
-3	-1	2	✓ 100%
2	-5	2	✓ 100%
-2	3	1	✓ 100%

Then, compute the new position of the cluster centers for the next iteration. Express your answer using list notation e.g. [1,2].

C_1 [0.33, -3.67] ✓ 100%

C_2 [-1.00, 3.00] ✓ 100%

Hyperparameter for tuning couple

- a. More complex model — Decision tree with maximum number of nodes n. Higher n means...
- b. Less complex model — Logistic regression with $\alpha \sum_j |w_j|$ penalty in the objective function. Higher α means...
- b. Less complex model — Decision tree using cost complexity pruning with a penalty on the size of the tree $\alpha|T|$. Higher α means...
- a. More complex model — SVM with RBF kernel, $K_{RBF}(\mathbf{x}_i, \mathbf{x}_i) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_i\|^2\right)$. Higher γ means...
- b. Less complex model — K nearest neighbor regression using the mean of K neighbors. Higher K means...
- b. Less complex model — Feature selection using a filter method, where each feature is scored using mutual information, and only features exceeding a threshold t are used. Higher t means...
- a. Less complex model — Logistic regression with $\alpha \sum_j |w_j|$ penalty in the objective function. Higher α means...

W13 - Reinforcement learning

- f. Policy — Maps states to actions, either deterministically or as a probability distribution.
- c. Temporal Difference (TD) Learning — Updates value estimates using the immediate reward and estimated value of the next state.
- a. Q-Learning — A value-based method using TD learning.
- b. ε-greedy Policy — Chooses random actions with some probability to allow exploration.
- e. Value Function — Estimates the expected return from a state or state-action pair.
- d. Monte Carlo Method — Updates values only at the end of complete episodes, once total return is known.

Question 2: Using a fitted support vector classifier

You fit a linear support vector classifier on some data, and find that $w_0 = 1.4$ and these are the support vectors and α values:

x_1	x_2	y	α
0	4	-1	0.2
1	1	1	0.2

(note: the index on x is a column index.)

Using the expression

$$\hat{s}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} w_j x_j = w_0 + \sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{p} x_{ij} x_j$$

find the value of $\hat{s}$ for $\mathbf{x} = [-2, 16]$. (Note that i indexes the training samples, j indexes the columns.)

$\hat{s} =$ -8.6 ✓ 100%

What is the predicted class $\hat{y}$ for $\mathbf{x} = [-2, 16]$ (-1 or 1)?

-1 ✓ +1 ✓ 100%

Comment Use the given expression to find $\hat{s}$:

1.4 + (0.2)(-1)(([0, 4], [-2, 16])) + (0.2)(1)(([1, 1], [-2, 16])) = -8.6

and the sign of this value is the predicted class.

Neural network with unacceptable performance

- b. Gradient descent does not converge to the minimum because it 'steps' too far in each iteration — Decrease the learning rate
- e. Gradient descent does not converge quickly because the loss surface may be steep in some directions and much less steep in others — Use an adaptive learning rate
- d. The model lacks the capacity to learn complex features and relationships — Increase the size of the network
- c. Weights are not updated because of the vanishing gradient problem — Use ReLU activation
- a. Garbage in, garbage out — Get better data

Encoding non-numeric variables

- b. Ordinal encoding — Employment status (possible values: unemployed, employed part-time, employed full-time)
- a. One-hot encoding — Marital status (possible values: never married, married, divorced, widowed, separated)
- a. One-hot encoding — Major field of study (possible values: computer science, computer engineering, electrical engineering, transportation engineering)
- b. Ordinal encoding — Degree type (possible values: BS, MS, PhD)

Question 14: Neural autoencoder for dimensionality reduction

Autoencoder

Consider the following statements about using a neural autoencoder for dimensionality reduction.

Which statement is **true** regarding the input and output dimensions of the neural autoencoder?

(a) The input shape will be **larger than** the output shape.

(b) The input shape will be **the same as** the output shape. ✓

(c) The input shape will be **smaller than** the output shape.

✓ 100%

Which statement is **true** regarding the use of a neural autoencoder to reduce dimensionality before applying a supervised learning model?

(a) You will use the output of the **decoder** network as input to the supervised learning model.

(b) You will use the output of the **encoder** network as input to the supervised learning model. ✓

✓ 100%

Model Selection

一家大学招生办公室希望使用机器学习，根据申请人的考试成绩和其他申请材料，将他们分为"录取"或"拒绝"。由于每个新生班级的名额是固定的，因此招生办公室需要能够轻松调整"录取"决定的数量，最好无需重新训练模型。您推荐哪种技术？

- 决策树 ✗
- 逻辑回归

×0%

一家机器人公司正在开发一款用于工厂环境的机器人。它将从遍布工厂的传感器收集大量数据，并利用这些数据快速实时地做出决策，决定机器人的下一步行动。机器学习模型可以在离战舰务器或专用设备上预先训练，但实时预测必须在机器人内部廉价的计算资源上运行。您推荐哪种技术？

- K最近邻 ✗
- 神经网络

×0%

一家生物科技初创公司正在开发一套用于医院急诊室的系统，该系统可以识别哪些患者可以安全地转诊至非紧急医疗机构，而无需在急诊室接受治疗。然而，该系统必须易于医生定期审核模型的决策，并确保其行为符合当前的医疗实践。您会推荐哪种技术？

- 决策树 ✓
- 深度神经网络

✓100%

一家公司正在开发一套针对精英体操运动员的人工智能评判系统。该系统将使用 GPU 进行数百小时体操比赛录像的训练。您会推荐哪种技术？

- 卷积神经网络 ✓
- 全连接神经网络

✓100%

Train an `AdaBoostClassifier`

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

Xtr = np.load("Xtr.npy")
Xts = np.load("Xts.npy")
ytr = np.load("ytr.npy").reshape(-1,)
yts = np.load("yts.npy").reshape(-1,)
```

Fit an `AdaBoostClassifier` on the training data (according to the specifi
accuracy of the classifier's predictions, and save this value in `acc_ts`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
model = AdaBoostClassifier(n_estimators=5, random_state = 42)
model.fit(Xtr, ytr)
yhat_ts = model.predict(Xts)
acc_ts = accuracy_score(yts, yhat_ts)
```

K-fold CV for number of features with feature selection

```
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold, train_test_split

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

Since there are so many columns, the feature selection and model fitting we are about to do can be computationally intensive. Therefore, we'll only consider the first 2000 columns. The next cell will load in that feature data into `X` and the labels into `y`.

```
X = np.load('X.npy', allow_pickle=True)[:,:2000]
y = np.load('y.npy', allow_pickle=True)
```

Then, you will set aside 25% of the data for evaluating the final model at the end. Save the result in `Xtr`, `ytr`, `Xts`, and `yts`.

Use `sklearn`'s `train_test_split` with shuffling, and you will specify the `random_state = 42` so that your results will match the autograders' results.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size=0.25, random_state=42)
```

Now, you will use 10-fold cross validation (with `sklearn`'s `KFold`, no additional shuffling since you have already shuffled the data) to evaluate model candidates as follows:

- First, within each fold, compute the *absolute value* of the correlation coefficient between each column of the feature data and the target variable. (You may use `numpy`'s `corrcoef` function.) Save the results in `score_ff`, which has one entry per column per fold.
- Then, iterate over the number of columns to include in the model - the `d` values in `d_list`. In each iteration, you will use the `d` features that had the highest absolute value of correlation coefficient in the model.
- You will train an SVC model with a linear kernel, `C=10`, `random_state = 24`, and all other settings at their default values. You will evaluate the model on the validation data and save the accuracy score in `score_val`, which has one entry per `d` value per fold.

(Note: in many cases we would standardize the data before fitting an SVC, but we won't do that here.)

Write your solution in the `#grade` cell.

```
d_list = np.arange(1, X.shape[1]+1)
nd = len(d_list)
nfold = 10
score_ff = np.zeros((nfold, X.shape[1]))
score_val = np.zeros((nfold, nd))

#grade/write your code in this cell and DO NOT DELETE THIS LINE)
kf = KFold(n_splits=nfold, shuffle=False)
for i, (train_index, val_index) in enumerate(kf.split(Xtr)):
    X_train, X_val = Xtr[train_index,:], Xtr[val_index,:]
    y_train, y_val = ytr[train_index], ytr[val_index]
    for j in range(X_train.shape[1]):
        if i < nfold and j < X_train.shape[1]:
            score_ff[i, j] = np.abs(np.corrcoef(X_train[:, j], y_train)[0, 1])
    for k, d in enumerate(d_list):
        if i < nfold and k <= nd:
            top_features = np.argsort(score_ff[i])[-d:]
            X_train_selected = X_train[:, top_features]
            X_val_selected = X_val[:, top_features]
            model = SVC(kernel='linear', C=10, random_state=24)
            model.fit(X_train_selected, y_train)
            y_pred = model.predict(X_val_selected)
            score_val[i, k] = accuracy_score(y_val, y_pred)
```

Use `score_val` to find `best_d`, the optimal number of features to include in the model (best mean validation accuracy). (Compute the value - don't hard-code it.)

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
best_d = d_list[np.argmax(np.mean(score_val, axis=0))]
```

Then, find `best_d_one_se`, the optimal number of features to include according to the one-SE rule:

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
mean_scores = np.mean(score_val, axis=0)
std_scores = np.std(score_val, axis=0)
best_score = np.max(mean_scores)
best_score_se = std_scores[np.argmax(mean_scores)]
one_se_threshold = best_score - best_score_se
best_d_one_se = d_list[np.min(np.where(mean_scores >= one_se_threshold))]
```

Predict Song Popularity

Predicting song popularity

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GroupShuffleSplit, TimeSeriesSplit
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
```

Assign the value specified in the question page to the `random_state` variable. Use this variable throughout your work on this question.

```
random_state = 5
```

You are working with a dataset of Taylor Swift songs:

```
df = pd.read_csv("taylor_swift_spotify.csv", index_col = 0)
df.head()
```

	name	album	release_date	track_number	id	uri	acousticness	danceability	energy	instrumentalness	liveness	loud
0	Fortnight (feat. Post Malone)	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	1	6dODwocEuGzHAaxXqTbwHv	spotify:track:6dODwocEuGzHAaxXqTbwHv	0.5020	0.504	0.386	0.000015	0.0961	-10
1	The Tortured Poets Department	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	2	4PdLaGZubp4IgHChqoReB	spotify:track:4PdLaGZubp4IgHChqoReB	0.0483	0.604	0.428	0.000000	0.1260	-8
2	My Boy Only Breaks His Favorite Toys	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	3	7uGYWMwRy24dm7RUDDhUJD	spotify:track:7uGYWMwRy24dm7RUDDhUJD	0.1370	0.596	0.563	0.000000	0.3020	-7
3	Down Bad	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	4	1kbEbBdEgQaQaUKCJh2Bpl	spotify:track:1kbEbBdEgQaQaUKCJh2Bpl	0.5600	0.541	0.366	0.000001	0.0946	-10
4	So Long, London	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	5	7wAkQF5hJ27V8i36ZMqevQr	spotify:track:7wAkQF5hJ27V8i36ZMqevQr	0.7300	0.423	0.533	0.002640	0.0816	-11

You may add some additional code here to explore the data (the following cell is not graded).

1. Prepare data ¶

To prepare data for use by your model, create a data frame of features, `X`, including the following columns:

- `acousticness`
- `danceability`
- `energy`
- `instrumentalness`
- `liveness`
- `loudness`
- `speechiness`
- `tempo`
- `valence`
- `duration_ms`

and a data series `y` with the song's popularity (`popularity`).

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
X = df[['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']]
y = df['popularity']
```

Then, split `X` and `y` into a training and test set. Reserve 100 samples for the test set, and put the rest in the training set. Shuffle the data, and use the random state specified in the question page.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=100, shuffle=True, random_state=random_state)
```

2. Predict song popularity

Next, train a `RandomForestRegressor` to predict the popularity of each song.

- Use the random state specified in the question page
- and put 100 trees in your random forest
- leave other settings at their default values

Save the predictions of the model on the test set in `y_pred_rf`, and its R2 score in `rsq_rf`.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
rf = RandomForestRegressor(n_estimators=100, random_state=random_state)
rf.fit(X_tr, y_tr)
```

```
RandomForestRegressor(random_state=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

RandomForestRegressor
?Documentation for RandomForestRegressorFitted
RandomForestRegressor(random_state=5)

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
y_pred_rf = rf.predict(X_ts)
rsq_rf = r2_score(y_ts, y_pred_rf)
print(rsq_rf)
```

```
0.3056984471639
```

3. Group split

You notice that some songs are represented more than once in the data (e.g. in different versions). You are concerned there may be some data leakage due to this duplication, since a song that is popular in one version is likely to be also be popular in another version of the same song.

For example, both versions of the song "Fortnight" in the data are currently very popular:

```
df[df['name'].str.contains("Fortnight", case=False, na=False)]
```

	name	album	release_date	track_number	id	uri	acousticness	danceability	energy	instrumentalness	liveness	loudness
0	Fortnight (feat. Post Malone)	THE TORTURED POETS DEPARTMENT: THE ANTHOLOGY	2024-04-19	1	6dODwocEuGzHAaxXqTbwHv	spotify:track:6dODwocEuGzHAaxXqTbwHv	0.502	0.504	0.386	0.000015	0.0961	-10.976
31	Fortnight (feat. Post Malone)	THE TORTURED POETS DEPARTMENT	2024-04-18	1	2OzhQSqBEmt7hmKYxf76m	spotify:track:2OzhQSqBEmt7hmKYxf76m	0.502	0.504	0.386	0.000015	0.0961	-10.976

Given the following `clean_title` function (which you will not change):

```
import re
def clean_title(title):
    title_no_parentheticals = re.sub(r'\s*\(.*?\)', '', title)
    return re.sub(r'^\s*|\s*$', '', title_no_parentheticals).strip()
```

create a `numpy` array `base_name` which has the "cleaned" title for each song title in the data.

Different versions of the same song, e.g. "Shake It Off" and "Shake It Off (Taylor's Version)", will have the same base name.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
base_name = df['name'].apply(clean_title).values
```

Now, you will use a `GroupShuffleSplit` to create a training and test set where samples with the same `base_name` will either all be in the training set together, or will all be in the test set together.

- Use 1 split.
- Reserve 100 samples for the test set, and put the rest in the training set.
- Use the random state specified in the question page

and save the training and test data in `X_tr_grp`, `X_ts_grp`, `y_tr_grp`, `y_ts_grp`.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
gss = GroupShuffleSplit(n_splits=1, test_size=100, random_state=random_state)
for train_idx, test_idx in gss.split(X, y, groups=base_name):
    X_tr_grp = X.iloc[train_idx]
    X_ts_grp = X.iloc[test_idx]
    y_tr_grp = y.iloc[train_idx]
    y_ts_grp = y.iloc[test_idx]
```

Train a random forest (using the same configuration as before) on *this* data, and report the results in `y_pred_grp` and `rsq_grp`.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
rf_grp = RandomForestRegressor(n_estimators=100, random_state=random_state)
rf_grp.fit(X_tr_grp, y_tr_grp)
y_pred_grp = rf_grp.predict(X_ts_grp)
rsq_grp = r2_score(y_ts_grp, y_pred_grp)
print(rsq_grp)
```

```
0.14595588001194648
```

4. Time split

You ultimately want to predict the popularity of *newly* released songs based on *past* data, but you divided data into training and test sets without consideration of time.

You want to make sure that the evaluation above is not an "overly optimistic" one, due to temporal data leakage.

In the next cell,

- Make a copy of the data frame in `df_srt`.
- In `df_srt`, convert the release date to a `datetime` field.
- Sort by this field, so that the oldest songs are the first rows in `df_srt`, then by track number (e.g. for songs released on the same date, order by track number). (You may want to use `sort_values`)
- Reset the index of the data frame (and drop the old index) so that rows are re-numbered starting at zero. (You may want to use `reset_index`)

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
df_srt = df.copy()
df_srt['release_date'] = pd.to_datetime(df_srt['release_date'])
df_srt = df_srt.sort_values(by=['release_date', 'track_number'])
df_srt = df_srt.reset_index(drop=True)
```

Now, create `X_srt` and `y_srt` using the same columns as before, but they will now be sorted.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
X_srt = df_srt[['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']]
y_srt = df_srt['popularity']
```

Use them to divide the data into a training and test set *without* shuffling. (As before, leave 100 samples for the test set.)

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
X_tr_srt, X_ts_srt, y_tr_srt, y_ts_srt = train_test_split(X_srt, y_srt, test_size=100, shuffle=False, random_state=random_state)
```

Train a random forest (using the same configuration as before) on *this* data, and report the results in `y_pred_srt` and `rsq_srt`.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
rf_srt = RandomForestRegressor(n_estimators=100, random_state=random_state)
rf_srt.fit(X_tr_srt, y_tr_srt)
y_pred_srt = rf_srt.predict(X_ts_srt)
rsq_srt = r2_score(y_ts_srt, y_pred_srt)
print(rsq_srt)
```

```
-4.5305787424011937
```

5. (Bonus) Predict max popularity

This is a bonus section. If you complete this part correctly, you can earn above 100% on the overall exam. But, don't spend time on this section before you have answered the rest of the questions on the exam.

You decide that instead of predicting the popularity of *every* version of a song, you just want to predict the popularity of the most popular version of each song.

So, you create a new data frame in which:

- for each unique value of `base_name`, you only include the sample in the original data that is the most popular.
- this data is also sorted by release date, then by track number.

From this data frame, you create `X_max` and `y_max`, then `X_tr_max`, `X_ts_max`, `ytr_max`, `yts_max`. Since this is a smaller sample, you reserve only 50 samples for the test set. Once again, you do not shuffle the data.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
idx_max = df.groupby(df['name'].apply(clean_title))['popularity'].idxmax()
df_max = df.loc[idx_max]
df_max = df_max.sort_values(by=['release_date', 'track_number'])
df_max = df_max.reset_index(drop=True)
X_max = df_max[['acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', 'valence', 'duration_ms']]
y_max = df_max['popularity']
X_tr_max, X_ts_max, y_tr_max, y_ts_max = train_test_split(X_max, y_max, test_size=50, shuffle=False, random_state=random_state)
```

Train a random forest (using the same configuratoin as before) on *this* data, and report the results in `y_pred_max` and `rsq_max`.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
rf_max = RandomForestRegressor(n_estimators=100, random_state=random_state)
rf_max.fit(X_tr_max, y_tr_max)
y_pred_max = rf_max.predict(X_ts_max)
rsq_max = r2_score(y_ts_max, y_pred_max)
print(rsq_max)
```

```
-2.6356232335535182
```

Construct and compile model ¶

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation, Input
from tensorflow.keras import optimizers
import tensorflow.keras.backend as K
import tensorflow as tf
```

Construct a neural network following the problem parameters described in the question.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
model = Sequential()
model.add(Dense(64, input_dim=11, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Create an optimizer and compile the model. Select the appropriate loss function for this type of problem, and use an appropriate metric.

```
#grade/write your code in this cell and DO NOT DELETE THIS LINE)
opt = optimizers.Adam()
model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Backpropagation & NN Short answer grading

```
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def relu(x):
    return np.maximum(0, x)
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# 定义权重
w_x1h1 = 1.5
w_x2h1 = 2
w_x1h2 = -2.5
w_x2h2 = -1
w_h1o1 = 2
w_h2o1 = -0.5
w_xbh1 = 2.5
w_xbh2 = 3
w_hbh1 = -2

# 输入值
x1 = 0
x2 = 1
# sigmoid input
u_h1 = sigmoid(x1 * w_x1h1 + x2 * w_x2h1 + w_xbh1)
u_h2 = sigmoid(x1 * w_x1h2 + x2 * w_x2h2 + w_xbh2)
# relu input
# net_h1 = x1 * w_x1h1 + x2 * w_x2h1 + w_xbh1
# u_h1 = relu(net_h1)
# net_h2 = x1 * w_x1h2 + x2 * w_x2h2 + w_xbh2
# u_h2 = relu(net_h2)
# linear input
# u_o1 = u_h1 * w_h1o1 + u_h2 * w_h2o1 + w_hbh1
# sigmoid output
net_o1 = u_h1 * w_h1o1 + u_h2 * w_h2o1 + w_hbh1
u_o1 = sigmoid(net_o1)

# 真实值
y = 1
# 计算o1节点的反向传播误差，linear output
# delta_o1 = (u_o1 - y)
# 计算o1节点的反向传播误差，sigmoid output
delta_o1 = (u_o1 - y) * u_o1 * (1 - u_o1)
delta_h1 = delta_o1 * w_h1o1 * u_h1 * (1 - u_h1)
delta_h2 = delta_o1 * w_h2o1 * u_h2 * (1 - u_h2)
# relu input
# delta_h1 = delta_o1 * w_h1o1 * relu_derivative(net_h1)
# delta_h2 = delta_o1 * w_h2o1 * relu_derivative(net_h2)
```

```
grad_w_x1h1 = delta_h1 * x1
grad_w_x2h1 = delta_h1 * x2
grad_w_xbh1 = delta_h1
gradients_h1 = [grad_w_x1h1, grad_w_x2h1, grad_w_xbh1]

grad_w_x1h2 = delta_h2 * x1
grad_w_x2h2 = delta_h2 * x2
grad_w_xbh2 = delta_h2
gradients_h2 = [grad_w_x1h2, grad_w_x2h2, grad_w_xbh2]

grad_w_h1o1 = delta_o1 * u_h1
grad_w_h2o1 = delta_o1 * u_h2
grad_w_hbh1 = delta_o1
gradients_o1 = [grad_w_h1o1, grad_w_h2o1, grad_w_hbh1]

# 学习率
learning_rate = 0.4

w_x1h1 = w_x1h1 - learning_rate * grad_w_x1h1
w_x2h1 = w_x2h1 - learning_rate * grad_w_x2h1
w_xbh1 = w_xbh1 - learning_rate * grad_w_xbh1
updated_weights_h1 = [w_x1h1, w_x2h1, w_xbh1]

w_x1h2 = w_x1h2 - learning_rate * grad_w_x1h2
w_x2h2 = w_x2h2 - learning_rate * grad_w_x2h2
w_xbh2 = w_xbh2 - learning_rate * grad_w_xbh2
updated_weights_h2 = [w_x1h2, w_x2h2, w_xbh2]

w_h1o1 = w_h1o1 - learning_rate * grad_w_h1o1
w_h2o1 = w_h2o1 - learning_rate * grad_w_h2o1
w_hbh1 = w_hbh1 - learning_rate * grad_w_hbh1
updated_weights_o1 = [w_h1o1, w_h2o1, w_hbh1]

print("u_h1:", u_h1)
print("u_h2:", u_h2)
print("u_o1:", u_o1)
print("delta_o1:", delta_o1)
print("delta_h1:", delta_h1)
print("delta_h2:", delta_h2)
print("gradients_o1:", gradients_o1)
print("gradients_h1:", gradients_h1)
print("gradients_h2:", gradients_h2)
print("updated_weights_o1:", updated_weights_o1)
print("updated_weights_h1:", updated_weights_h1)
print("updated_weights_h2:", updated_weights_h2)
```

Short answer grading

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score
```

This question includes bonus points - you only need to earn 20 points of credit to get full credit, but you *can* earn up to 30 points (and therefore, up to 110% on the overall exam). You may work on whichever sections you like to earn the first 20 points, but don't spend time trying to get to 30 points until you have first answered the rest of the exam.

Use this `random_state` throughout your work:

```
random_state = 0
```

You are working with a data set that has four different short answer questions, each with a reference answer and approximately 30 student answers.

First, read in the data:

```
df = pd.read_csv("answers.csv")

df["id"].unique() # four question IDs

array([0, 1, 2, 3])

df["question"].unique() # four questions

array(['Carrie wanted to find out which was harder, a penny or a nickel, so she did a scratch test. How would this tell her which is harder?',
       'Look at the picture on the right. Label the poles on each magnet. (The bottom 2 magnets are stuck together, the others are not.) What is the rule that explains why you labeled the pole
s the way you did?',
       'A solution is a type of mixture. What makes it different from other mixtures?',
       'Katie got a guitar for her birthday. She experimented with the strings and found she could change their sounds. One way Katie could change the sound of a string was to tighten it. Desc
ribe how the sound was different when the string was tightened.'],
      dtype=object)

df["reference_answer"].unique() # four reference answers (the "official" correct answer)

array(['The harder coin will scratch the other.',
       'Like poles repel and opposite poles attract.',
       'A solution is a mixture formed when a solid dissolves in a liquid.',
       'When the string was tighter, the pitch was higher.'], dtype=object)

df.head() # note many answers per question
```

	answer	correct	question	reference_answer	id
0	You could tell if it has the hardest if most o...	0	Carrie wanted to find out which was harder, a ...	The harder coin will scratch the other.	0
1	If just the penny could scratch and the nickel...	1	Carrie wanted to find out which was harder, a ...	The harder coin will scratch the other.	0
2	Whichever one was damaged most was less hard.	1	Carrie wanted to find out which was harder, a ...	The harder coin will scratch the other.	0
3	Rub them against a crystal.	0	Carrie wanted to find out which was harder, a ...	The harder coin will scratch the other.	0
4	Which had less scratches.	1	Carrie wanted to find out which was harder, a ...	The harder coin will scratch the other.	0

1. Train a RandomForest

You will train and evaluate a separate model for each question, so in everything you do, you will start by iterating over the question IDs.

Then, for each question ID:

- get the data for that question ID
- divide the data into a training and test set, using the random state specified above, and leaving 10 random samples per question in the test set with the rest in the training set.
- Use a `CountVectorizer` with `stop_words = "english"` to create a numeric representation of the answers to that question, using the answer data in the training set to fit the `CountVectorizer`.
- train a `RandomForestClassifier` to predict whether or not the answer is correct. Use 20 trees in your forest, and use the random state specified above.
- save the result on the validation set (per question ID) in `acc_rf`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
question_ids = df["id"].unique()
acc_rf = np.zeros(len(question_ids))
# implement here
for i, question_id in enumerate(question_ids):
    question_data = df.df["id"] == question_id
    X = question_data["answer"]
    y = question_data["correct"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10, random_state=random_state)
    vectorizer = CountVectorizer(stop_words="english")
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)
    model = RandomForestClassifier(n_estimators=20, random_state=random_state)
    model.fit(X_train_vec, y_train)
    y_pred = model.predict(X_test_vec)
    acc_rf[i] = accuracy_score(y_test, y_pred)
print(acc_rf)

[0.6 0.7 0.8 0.8]
```

2. Evaluate a RandomForest with KFold CV

Since you are working with a very small data set, you are concerned that perhaps the results are highly dependent on the random draw of training vs test samples. So, you repeat the analysis above, but with a KFold CV for evaluation:

- first, you iterate over question IDs
- then, you set up a KFold CV with 5 folds. Shuffle the data and use the random state specified above.
- inside each fold,
  - Use a `CountVectorizer` with `stop_words = "english"` to create a numeric representation of the answers to that question, using the training set to fit the `CountVectorizer`.
  - train a `RandomForestClassifier` to predict whether or not the answer is correct. Use 20 trees in your forest, and use the random state specified above.
  - save the result on the validation set (per question ID) in `acc_rf_kf`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
question_ids = df["id"].unique()
n_folds = 5
acc_rf_kf = np.zeros((len(question_ids), n_folds))
# implement here
for i, question_id in enumerate(question_ids):
    question_data = df.df["id"] == question_id
    X = question_data["answer"]
    y = question_data["correct"]
    kf = KFold(n_splits=n_folds, shuffle=True, random_state=random_state)
    for j, (train_index, test_index) in enumerate(kf.split(X)):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        vectorizer = CountVectorizer(stop_words="english")
        X_train_vec = vectorizer.fit_transform(X_train)
        X_test_vec = vectorizer.transform(X_test)
        model = RandomForestClassifier(n_estimators=20, random_state=random_state)
        model.fit(X_train_vec, y_train)
        y_pred = model.predict(X_test_vec)
        acc_rf_kf[i, j] = accuracy_score(y_test, y_pred)
```

Then, get the average accuracy per question ID (across folds) and save it in `acc_rf_mean`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
acc_rf_mean = np.mean(acc_rf_kf, axis=1)
print(acc_rf_mean)

[0.61071429 0.81428571 0.77857143 0.74642857]
```

3. Clustering similar answers

The supervised learning model above is interesting, but is only useful for questions on which Professor Fund already has labeled data from previous semesters. It won't help her grade new questions that have not been used in previous semesters.

To help with brand-new questions, you will also create a supervised learning model that will group together similar answers, making them easier to grade.

In the next cell,

- first, you iterate over question IDs. For each question,
  - Use a `CountVectorizer` with `stop_words = "english"` to create a numeric representation of the answers to that question, using the answer data to fit the `CountVectorizer`.
  - then use a `kMeans` model with 3 clusters to group similar answers. Use the random state specified above, and leave all other settings at their default values.
  - save the cluster ID of each sample in `cluster_ids`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
cluster_ids = np.zeros(len(df), dtype=int)
# implement here
for i, question_id in enumerate(question_ids):
    question_data = df.df["id"] == question_id
    X = question_data["answer"]
    vectorizer = CountVectorizer(stop_words="english")
    X_vec = vectorizer.fit_transform(X)
    kmeans = KMeans(n_clusters=3, random_state=random_state)
    cluster_ids[df["id"] == question_id] = kmeans.fit_predict(X_vec)
```

If your clustering model is doing a good job of grouping together answers that will be graded similarly, we would expect that within a cluster, most of the answers should be either correct or incorrect (i.e. most answers within a cluster should have similar "correctness".)

For each question and each cluster, compute the average "correctness" within the cluster (i.e. the average value of "correct" among samples within that cluster). Save the result in `cluster_correctness`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
question_ids = df["id"].unique()
n_clusters = 3
cluster_correctness = np.zeros((len(question_ids), n_clusters))
# implement here
for i, question_id in enumerate(question_ids):
    question_data = df[df["id"] == question_id]
    for j in range(n_clusters):
        cluster_data = question_data[cluster_ids[df["id"] == question_id] == j]
        if len(cluster_data) > 0:
            cluster_correctness[i, j] = cluster_data["correct"].mean()
print(cluster_correctness)

[[1.         0.57142857 0.4       ]
 [0.05882353 0.75       0.14285714]
 [0.23333333 0.         0.5       ]
 [0.38095238 0.57142857 0.        ]]
```

relu（Rectified Linear Unit，修正线性单元）

- **公式**：$f(x) = \max(0, x)$。
- **特点**：当输入x小于 0 时，输出为 0；当x大于等于 0 时，输出等于x。计算简单高效，没有复杂的指数运算；能缓解梯度消失问题，在$x > 0$时梯度恒为 1；符合生物神经元特性，模拟大脑神经元 “兴奋 - 抑制” 机制。
- **应用场景**：广泛应用于神经网络隐藏层，尤其在图像识别等计算机视觉领域表现出色。不过存在 “神经元死亡” 问题，当某些神经元输入恒为负时，输出恒为 0，不再激活。

sigmoid

- **公式**：$f(x) = \frac{1}{1+e^{-x}}$。
- **特点**：将任意实数映射到$(0, 1)$区间，可表示概率，具有非线性特性，能帮助神经网络捕捉数据间复杂非线性关系。但在输入值很大或很小时，处于饱和状态，梯度趋近于 0，会导致梯度消失问题，影响网络训练。
- **应用场景**：常用于二分类问题，在早期神经网络中应用较多，比如逻辑回归中用于将线性模型输出转化为概率。

tanh（双曲正切函数）

- **公式**：$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。
- **特点**：把实数映射到$(-1, 1)$之间，函数形状与 sigmoid 类似，相比 sigmoid，其均值更接近 0，收敛速度可能更快。不过在输入较大或较小时，输出平滑，梯度较小，也存在梯度消失问题。
- **应用场景**：可用于需要将数据归一化到特定区间的场景，但在深层网络中也因梯度问题使用受限。

softmax

- **公式**：$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$，$z_i$表示第$i$类的输入值。
- **特点**：用于多分类任务，将输入的多个实数值转换为 0 到 1 之间的概率分布，所有输出数值之和等于 1，通过指数运算放大输入值差异，并进行归一化。
- **应用场景**：神经网络输出层处理多分类问题，如图片分类、文本分类等，将模型输出的 logits 值转换为不同类别的概率分布。

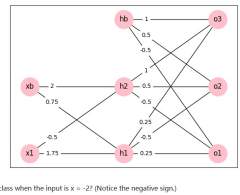
linear（线性激活函数）

- **公式**：$f(x) = x$。
- **特点**：输出与输入保持完全线性关系，无任何非线性变换，计算简单，输出范围无限制，可输出任意实数值。但不引入非线性，限制模型对复杂非线性关系的表达能力。
- **应用场景**：适用于回归问题，当预测目标值与输入特征存在线性关系时，如预测房价、温度等连续值，或作为线性回归模型、多层感知机（MLP）回归模型最后一层的激活函数。

Consider the following fully connected neural network for a three-class classification problem, with one input unit, three output units, and two hidden units. Weights are indicated on the edges. **xb** and **hb** represent bias inputs.

There is a **ReLU** activation at the hidden units, and a **softmax** activation at the output units.

Note: the ReLU activation function is $f(x) = \max(0, x)$.



What is the predicted class when the input is x = -2? (Notice the negative sign.)

3

Comment

First, we compute the input to the activation function at each of the hidden units:

$$z_{h1} = 1.75 \times -2 + 0.75 = -2.75$$

and

$$z_{h2} = -0.5 \times -2 + 2 = 3$$

Then, we compute the output of the activation function at each of the hidden units:

$$u_{h1} = \text{ReLU}(-2.75) = 0$$

and

$$u_{h3} = \text{ReLU}(3) = 3$$

Finally, at the output layer, the input to the softmax activation function will be

$$z_{o1} = 0.25 \times 0 + -0.5 \times 3 + -0.5 = -2$$
$$z_{o2} = -0.5 \times 0 + 0.5 \times 3 + 0.5 = 2$$
$$z_{o3} = 0.25 \times 0 + 1 \times 3 + 1 = 4$$

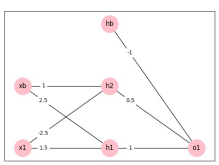
We use these values as input to the softmax function

$$g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{c=1}^{K} e^{z_c}}$$

which estimates the per-class probability as 0.0022, 0.1189, and 0.8789 for class 1, 2, and 3, respectively. The most probable class is therefore 3.

Consider the following fully connected neural network, with one input unit, one output unit, and two hidden units. Weights are indicated on the edges. **xb** and **hb** represent bias inputs.

There is a **sigmoid** activation at the hidden units, and a **linear** activation at the output unit.



What is the output of the network when the input is x = -2? (Notice the negative sign.)

$u_{o1} = -0.12$

Comment

First, we compute the input to the activation at each of the hidden units:

$$z_{h1} = 1.5 \times -2 + 2.5 = -0.5$$

and

$$z_{h2} = -2.5 \times -2 + 1 = 6$$

Then, we compute the output of the activation at each of the hidden units:

$$u_{h1} = \sigma(-0.5) = 0.3775406687981454$$

and

$$u_{h2} = \sigma(6) = 0.9975273768433653$$

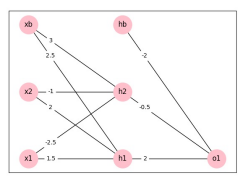
Finally, at the output unit, the input to the activation function will be

$$z_{o1} = 1 \times 0.3775406687981454 + 0.5 \times 0.9975273768433653 + -1$$

and since the output unit uses a linear activation function, the output is

$$u_{o1} = -0.12369564278017187$$

Consider a neural network for classification with two features at the input, two hidden-layer nodes in a single hidden layer, and one output node. There is a sigmoid activation function at the hidden layer nodes and at the output layer node.



Part 1: Forward pass

For the input $x_1 = 0, x_2 = 1$, do a forward pass on the network and compute the outputs.

$u_{h_1} =$ 0.989

$u_{h_2} =$ 0.881

$u_{o_1} =$ 0.386

Part 1: Forward pass

1. **计算隐藏层节点输入**
   - 对于隐藏层节点 h_1，其输入 z_{h_1} 为：
     - $z_{h_1} = w_{x_1 h_1} x_1 + w_{x_2 h_1} x_2 + w_{bh_1}$
     - 已知 $x_1 = 0, x_2 = 1, w_{x_1 h_1} = -1.5, w_{x_2 h_1} = 2, w_{bh_1} = 2.5$
     - 则 $z_{h_1} = -1.5 \times 0 + 2 \times 1 + 2.5 = 4.5$
   - 对于隐藏层节点 h_2，其输入 z_{h_2} 为：
     - $z_{h_2} = w_{x_1 h_2} x_1 + w_{x_2 h_2} x_2 + w_{bh_2}$
     - 已知 $w_{x_1 h_2} = -2.5, w_{x_2 h_2} = -1, w_{bh_2} = 3$
     - 则 $z_{h_2} = -2.5 \times 0 + (-1) \times 1 + 3 = 2$
2. **应用激活函数（sigmoid 函数 $\sigma(z) = \frac{1}{1+e^{-z}}$）**
   - 对于 h_1：
     - $u_{h_1} = \sigma(z_{h_1}) = \frac{1}{1+e^{-4.5}} \approx 0.989$
   - 对于 h_2：
     - $u_{h_2} = \sigma(z_{h_2}) = \frac{1}{1+e^{-2}} \approx 0.881$
3. **计算输出层节点输入和输出**
   - 输出层节点 o_1 的输入 z_{o_1} 为：
     - $z_{o_1} = w_{h_1 o_1} u_{h_1} + w_{h_2 o_1} u_{h_2} + w_{bo_1}$
     - 已知 $w_{h_1 o_1} = -2, w_{h_2 o_1} = -0.5, w_{bo_1} = -2$
     - $z_{o_1} = -2 \times 0.989 + (-0.5) \times 0.881 + (-2)$
     - $z_{o_1} = -2 \times 0.989 - 0.5 \times 0.881 - 2 = -4.3185$
   - 应用 sigmoid 函数：
     - $u_{o_1} = \sigma(z_{o_1}) = \frac{1}{1+e^{4.3185}} \approx 0.386$

Part 2: Compute backpropagation errors

Suppose the true value is $y = 1$ when the input is $x_1 = 0, x_2 = 1$.

Compute the backpropagation error $\delta_j = \frac{\partial L}{\partial z_j}$ at each node.

Use the loss function

$$L(W) = \frac{1}{2} \sum_n (y - u_O)^2$$

(We would usually use the binary cross entropy loss function for a classification problem, but the math is easier to work with for the L2 loss! So in this case, we'll use this L2 loss.)

Note that because of the sigmoid activation function at the output node, $u_O = \sigma(z_O)$, you will compute the backpropagation error at the output with

$$\delta_O = \frac{\partial L}{\partial z_O} = \frac{\partial L}{\partial u_O} \frac{\partial u_O}{\partial z_O}$$

$\delta_{h_1} =$ -0.145 ✓ 100%

$\delta_{h_2} =$ -0.003 ✓ 100%

$\delta_{h_3} =$ 0.008 ✓ 100%

Part 2: Compute backpropagation errors

1. **计算输出层误差 δ_{o_1}**
   - 首先，对损失函数 $L(W) = \frac{1}{2} \sum_n (y - u_o)^2$ 关于 u_o 求导：$\frac{\partial L}{\partial u_o} = -(y - u_o)$
   - 已知 $y = 1, u_o = 0.386$，则 $\frac{\partial L}{\partial u_o} = -(1 - 0.386) = -0.614$
   - sigmoid 函数的导数 $\sigma'(z) = \sigma(z)(1 - \sigma(z))$，对于 z_{o_1}，
     $\sigma'(z_{o_1}) = 0.386 \times (1 - 0.386) = 0.386 \times 0.614 = 0.237$
   - 根据 $\delta_o = \frac{\partial L}{\partial z_o} = \frac{\partial L}{\partial u_o} \frac{\partial u_o}{\partial z_o}$，$\delta_{o_1} = -0.614 \times 0.237 = -0.145$
2. **计算隐藏层误差 δ_{h_1} 和 δ_{h_2}**
   - 对于隐藏层节点 h_1：
     - $\delta_{h_1} = \delta_{o_1} w_{h_1 o_1} \sigma'(z_{h_1})$
     - 已求得 $\delta_{o_1} = -0.145, w_{h_1 o_1} = -2$，
       $\sigma'(z_{h_1}) = 0.989 \times (1 - 0.989) = 0.989 \times 0.011 = 0.010879$
     - $\delta_{h_1} = -0.145 \times (-2) \times 0.010879 = -0.003$
   - 对于隐藏层节点 h_2：
     - $\delta_{h_2} = \delta_{o_1} w_{h_2 o_1} \sigma'(z_{h_2})$
     - 已知 $\delta_{o_1} = -0.145, w_{h_2 o_1} = -0.5$，
       $\sigma'(z_{h_2}) = 0.881 \times (1 - 0.881) = 0.881 \times 0.119 = 0.104839$
     - $\delta_{h_2} = -0.145 \times (-0.5) \times 0.104839 = 0.008$

Part 3: Compute gradients of loss function with respect to weights

Now, compute the gradient of the loss function with respect to the weights.

Each of the answers below should be a vector, which you can express using list notation e.g. `[1,2,3]`.

$\frac{\partial L}{\partial W_{o1}} = [\frac{\partial L}{\partial w_{h_1 o1}}, \frac{\partial L}{\partial w_{h2,o1}}, \frac{\partial L}{\partial w_{bh,o1}}] =$ [-0.144, -0.128, -0.145]

$\frac{\partial L}{\partial W_{h1}} = [\frac{\partial L}{\partial w_{x_1 h1}}, \frac{\partial L}{\partial w_{x2,h1}}, \frac{\partial L}{\partial w_{xh,h1}}] =$ [0.000, -0.003, -0.003]

$\frac{\partial L}{\partial W_{h2}} = [\frac{\partial L}{\partial w_{x_1 h2}}, \frac{\partial L}{\partial w_{x2,h2}}, \frac{\partial L}{\partial w_{bh,h2}}] =$ [0.000, 0.008, 0.008]

Part 3: Compute gradients of loss function with respect to weights

1. **计算 $\frac{\partial L}{\partial W_{o1}}$**
   - $\frac{\partial L}{\partial w_{h_1 o_1}} = \delta_{o_1} u_{h_1} = -0.145 \times 0.989 = -0.144$
   - $\frac{\partial L}{\partial w_{h_2 o_1}} = \delta_{o_1} u_{h_2} = -0.145 \times 0.881 = -0.128$
   - $\frac{\partial L}{\partial w_{bo_1}} = \delta_{o_1} = -0.145$
   - 所以 $\frac{\partial L}{\partial W_{o1}} = [-0.144, -0.128, -0.145]$
2. **计算 $\frac{\partial L}{\partial W_{h1}}$**
   - $\frac{\partial L}{\partial w_{x_1 h_1}} = \delta_{h_1} x_1 = -0.003 \times 0 = 0.000$
   - $\frac{\partial L}{\partial w_{x_2 h_1}} = \delta_{h_1} x_2 = -0.003 \times 1 = -0.003$
   - $\frac{\partial L}{\partial w_{bh_1}} = \delta_{h_1} = -0.003$
   - 所以 $\frac{\partial L}{\partial W_{h1}} = [0.000, -0.003, -0.003]$
3. **计算 $\frac{\partial L}{\partial W_{h2}}$**
   - $\frac{\partial L}{\partial w_{x_1 h_2}} = \delta_{h_2} x_1 = 0.008 \times 0 = 0.000$
   - $\frac{\partial L}{\partial w_{x_2 h_2}} = \delta_{h_2} x_2 = 0.008 \times 1 = 0.008$
   - $\frac{\partial L}{\partial w_{bh_2}} = \delta_{h_2} = 0.008$
   - 所以 $\frac{\partial L}{\partial W_{h2}} = [0.000, 0.008, 0.008]$

Neural network - summary

Things that are “given”

For a particular problem, these are “given”:

- the number of inputs
- the number of outputs
- the activation function to use at the output
- the loss function

Things that we decide

We still need to decide:

- the number of hidden units
- the activation function to use at hidden units

Training the network

- Still need to find the $\mathbf{W}$ that minimizes $L(\mathbf{W})$.
- How?

Train a `RandomForestRegressor`

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
```

In this question, we will try to predict the time it will take for an order to be delivered.

First, we'll load the dataset:

```
df = pd.read_csv('data.csv', names=['id', 'cost', 'average_cost', 'average_time_wk', 'average_time_hr', 'has_drive_thru', 'unfulfilled_orders', 'time'], header=None, index_col='id')
```

You can add some code here to inspect the data, see the names of features, and see the data types - the cell below will not be graded.

```
df.head()
```

id	cost	average_cost	average_time_wk	average_time_hr	has_drive_thru	unfulfilled_orders	time
1375.0	9.595529	12.929266	24.808019	28.140299	1.0	2.581362	21.885150
1376.0	58.060832	38.395697	62.905777	103.955612	0.0	4.882039	85.343048
1377.0	16.364105	16.678048	36.988889	54.138861	1.0	5.846501	42.501774
1378.0	10.411111	20.756315	36.531330	59.250801	1.0	5.673834	51.074250
1379.0	6.501663	21.792520	41.203542	56.736735	1.0	5.142906	50.419929

(but, note that your code will be evaluated on *different* data organized in a data frame with the same columns - so in your solution, you should not hard-code anything specific to this data.)

Now we will split into training and test sets, using `train_test_split` !

- Reserve 20% of the data for testing.
- Use the random state specified on the question page.

The following cell should create

- `Xtr` and `Xts` as pandas data frames including *only* the features,
- and `ytr` and `yts` as either pandas data series or 1d numpy arrays containing the target variable.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
features = ['cost', 'average_cost', 'average_time_wk', 'average_time_hr', 'has_drive_thru', 'unfulfilled_orders']
target = ['time']
X = df[features]
y = df[target]
random_state = 28
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size = 0.2, random_state = random_state)
```

Now we are ready to fit the `RandomForestRegressor` . Using

- the random state specified in the question page
- and setting the number of trees in the forest to 10
- and the default settings otherwise,

fit the model on the training data. Then, use it to make predictions for the test samples, and save this prediction in `yts_hat` . Evaluate the R2 score of the model on the test data, and save this in `rsq` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
model = RandomForestRegressor(n_estimators=10, random_state = random_state)
model.fit(Xtr, ytr)
yts_hat = model.predict(Xts)
rsq = r2_score(yts, yts_hat)
```

```
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), fo
```

Train a KNN classifier

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

First, we'll load the dataset:

```
df = pd.read_csv('data.csv', index_col = 'Event_ID')
```

You can add some code here to inspect the data, see the names of features, and see the data types. For example, what is the proportion of false alarms? The cell below will not be graded.

```
df.head()
```

Event_ID	Address	Presence_of_Pets	Percentage_of_Frames_with_Human_Presence	False_Alarm	Time_of_Day	Duration_of_Motion_Event	Events_in_Last_Hour	Outdoor_Temperature
1	2422 Willow Rd	1	90.078592	0	5	19	0	53.212582
2	3036 Main Ct	1	98.169261	0	17	20	1	70.452566
3	6228 Chestnut Ln	0	25.555773	0	4	25	2	69.541186
4	5620 Maple Rd	0	26.367181	0	18	25	0	69.034840
5	9363 Oak Rd	0	99.221166	0	19	7	1	67.195680

(but, note that your code will be evaluated on *different* data organized in a data frame with the same columns - so in your solution, you should not hard-code anything specific to this data.)

Now we will split into training and test sets, using `train_test_split` !

- Reserve 20% of the data for testing.
- Use the random state specified on the question page.

The following cell should create

- `Xtr` and `Xts` as pandas data frames including *only* the features used to train the model,
- and `ytr` and `yts` as either pandas data series or 1d numpy arrays containing the target variable.

(For pandas data frames or data series, don't change the names of any columns.)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
df = df.drop('Address', axis=1)
X = df.drop('False_Alarm', axis=1)
y = df['False_Alarm']
random_state = 13
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size = 0.2, random_state = random_state)
```

Now we are ready to fit the `KNeighborsClassifier` . Using

- 9 neighbors
- and default settings for everything else,

fit the model on the training data. Then, use it to make predictions for the test samples, and save this prediction in `yts_hat` . Evaluate the accuracy score of the model on the test data, and save this i
`acc` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
knn = KNeighborsClassifier(n_neighbors=9).fit(Xtr, ytr)
yts_hat = knn.predict(Xts)
acc = accuracy_score(yts, yts_hat)
```

Train a linear SVM

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

First, we'll load the dataset:

```
df = pd.read_csv('data.csv')
```

You can add some code here to inspect the data, see the names of features, and see the data types - the cell below will not be graded.

```
df.head()
```

	Address	Walkability_Score	Parks_Nearby	Grocery_Stores_Nearby	Schools_Nearby	Public_Transit_Nearby
0	939 Maple Ave, Bloomington, TX	0	2	2	1	2
1	986 Cedar Blvd, Greenville, CA	0	2	2	1	1
2	795 Cedar Blvd, Madison, NC	1	1	3	3	2
3	742 Main St, Riverside, OR	0	2	2	2	0
4	515 Maple Ave, Brooklyn, NJ	0	2	2	1	3

(but, note that your code will be evaluated on *different* data organized in a data frame with the same columns - so in your solution, you should not hard-code anything specific to this data.)

Now we will split into training and test sets, using `train_test_split` !

- Reserve 20% of the data for testing.
- Use the random state specified on the question page.

The following cell should create

- `Xtr` and `Xts` as pandas data frames including *only* the features used to train the model,
- and `ytr` and `yts` as either pandas data series or 1d numpy arrays containing the target variable.

(For pandas data frames or data series, don't change the names of any columns.)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
df = df.drop('Address', axis=1)
X = df.drop('Walkability_Score', axis=1)
y = df['Walkability_Score']
random_state = 9
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size=0.2, random_state=random_state)
```

Now we are ready to fit the `SVC` . Using

- a linear kernel
- the random state specified in the question page
- and default settings for everything else.

fit the model on the training data. Then, use it to make predictions for the test samples, and save this prediction in `yts_hat` . Evaluate the accuracy score of the model on the test data, and save this in `acc` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
model = SVC(kernel='linear', random_state=random_state)
model.fit(Xtr, ytr)
yts_hat = model.predict(Xts)
acc = accuracy_score(yts, yts_hat)
```

Train a `KNeighborsRegressor`

```
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
```

```
Xtr = np.load("Xtr.npy")
Xts = np.load("Xts.npy")
ytr = np.load("ytr.npy")
yts = np.load("yts.npy")
```

Fit a `KNeighborsRegressor` on the training data (according to the specifi
R2 score of the model's predictions, and save this value in `r2_ts` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
knn = KNeighborsRegressor(n_neighbors = 9).fit(Xtr, ytr)
yhat_ts = knn.predict(Xts)
r2_ts = r2_score(yts, yhat_ts)
```

Train a `DecisionTreeClassifier`

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
Xtr = np.load("Xtr.npy")
Xts = np.load("Xts.npy")
ytr = np.load("ytr.npy")
yts = np.load("yts.npy")
```

Fit a `DecisionTreeClassifier` on the training data (according to the specifications in t
the accuracy of the classifier's predictions, and save this value in `acc_ts` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
model = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
model.fit(Xtr, ytr)
yhat_ts = model.predict(Xts)
acc_ts = accuracy_score(yts, yhat_ts)
```

Train a `RandomForestClassifier`

```
import pandas as pd
import numpy as np

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Load in data
df = pd.read_csv('mushrooms.csv')
X = pd.get_dummies(df.drop('class', axis=1), drop_first=True)
y = df['class']
```

Split the data into training and test sets. Use `test_size = 0.25` and the random state specified in the question page.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 42)
model = RandomForestClassifier(n_estimators=3, random_state = 42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=3, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

RandomForestClassifier
?Documentation for RandomForestClassifierFitted
RandomForestClassifier(n_estimators=3, random_state=42)

Fit a `RandomForestClassifier` on the training data (according to the specifications in the question). Then, use it to pre
the accuracy of the classifier's predictions, and save this value in `acc_ts` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE
yhat_ts = model.predict(X_test)
acc_ts = accuracy_score(y_test, yhat_ts)
```

Selecting SVM hyperparameters with grid search

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
```

First, upload the `data-svm.csv` file to the workspace. Then, load in data from that file:

```
dat = np.genfromtxt('data-svm.csv',delimiter=',')
X = dat[:,:-1]
y = dat[:,-1]
```

Now, we will split the data into training and test sets using `sklearn` 's implementation of `train_test_split` . We will:

- Reserve 25% of the data for testing, and leave 75% for training.
- Shuffle the data, and use the random state specified in the PrairieLearn question page.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
random_state = 28
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size=0.25, random_state=random_state)
```

Use the `StandardScaler` to standardize the data by removing the mean and scaling to unit variance (using the statistics of the training set only).

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
scaler = StandardScaler()
Xtr_std = scaler.fit_transform(Xtr)
Xts_std = scaler.transform(Xts)
```

We are going to use `GridSearchCV` to test the following "parameter grid" of SVM hyperparamters:

```
param_grid = [
  {'C': [1e-1, 1e1, 1e3, 1e5, 1e7], 'gamma': [1e2, 1, 1e-1, 1e-3], 'kernel': ['rbf']},
 ]
```

In the following cell, create a `GridSearchCV` and fit it using the standardized training data.

Use 3-fold CV in your grid search, and pass an `SVC` model and the parameter grid specified above. Leave other settings at their default values.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
clf = GridSearchCV(SVC(), param_grid, cv=3)
clf.fit(Xtr_std, ytr)
```

If your fitted model is in the variable `clf` , you can use the following cell to visualize the results:

```
scores = clf.cv_results_['mean_test_score'].reshape(len(param_grid[0]['C']),len(param_grid[0]['gamma']))
plt.figure(figsize=(6, 6))
plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.viridis)
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar()
plt.xticks(np.arange(len(param_grid[0]['gamma'])), param_grid[0]['gamma'])
plt.yticks(np.arange(len(param_grid[0]['C'])), param_grid[0]['C'])
plt.title('CV accuracy')
plt.show()
```

Use the `best_params_` attribute of your fitted model to find out the combination of `gamma` and `C` that had the best accuracy on the validation data.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
C_best = clf.best_params_['C']
gamma_best = clf.best_params_['gamma']
```

If you left the `refit` argument at its default setting (`True`), your `GridSearchCV` classifier was refitted on the entire training set, using the best parameters found in the grid search
model to make predictions on new data and evaluate its accuracy.

In the following cell, find the accuracy of the refitted model on the test data, and save it in `acc_ts` .

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
y_pred = clf.predict(Xts_std)
acc_ts = accuracy_score(yts, y_pred)
```

Detecting false statements

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, KFold
from sklearn.feature_selection import mutual_info_classif
```

First, load in the data:

```
df = pd.read_csv("data_factcheck.csv")
```

The data frame includes

- a `statement` field, which has the text of the statement,
- and a `label_binary` field indicating if it is false (1) or true/mostly true (0).

```
df.sample(5)
```

	statement	label_binary
619	Says Bruce Braley threatened to sue a neighbor...	1
456	Says Vladimir Putin said, President Obama and ...	1
1492	At least Obama didn't marry his cousin, as Glo...	0
356	Says Tommy Thompson favors Obamas health care ...	1
330	Says Gov. Scott Walkers repeal of a 2009 ban h...	1

We will split the data into a training and test set. Note that the data is ordered by class, so we *must* shuffle the data.

```
Xtr_str, Xts_str, ytr, yts = train_test_split(df['statement'].values, df['label_binary'].values, shuffle=True, random_state=42, test_size=0.25)
```

Part 1 - train an SVC classifier ¶

Note: Graded variables in this section: `Xtr_vec`, `Xts_vec`, `acc_score`.

First, we will train an SVC classifier.

To train a model, we will need to get this text into some kind of numeric representation. We will use a basic approach called "bag of words", that works as follows:

0. (Optional) Remove the "trivial" words that you want to ignore, such as "the", "an", "has", etc. from the text.
1. Compile a "vocabulary" - a list of all of the words in the dataset - with integer indices from 0 to $d - 1$.
2. Convert every sample into a d-dimensional vector x, by letting the j th coordinate of x be the number of occurences of the j th words in the document. (This number is often called the "term frequency".)

Now, we have a set of vectors - one for each sample - containting the frequency of each word.

We will use the `sklearn` implementation of this, which is called `CountVectorizer`. You may refer to the `CountVectorizer` documentation.

We will create an instance of a `CountVectorizer`, passing `stop_words = 'english'` and leaving other arguments at their default values.

Create `Xtr_vec` by fitting it on the training data, then using it to transform the training data. Use the already fitted vectorizer to transform the test data, to create `Xts_vec`. (This is the 'typical' data pre-processing pattern, where data pre-processing always uses the statistics of the training data only.)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
vec = CountVectorizer(stop_words='english')
Xtr_vec = vec.fit_transform(Xtr_str)
Xts_vec = vec.transform(Xts_str)
```

```
Xtr_vec.shape
```

```
(1500, 4561)
```

Next, fit an SVC classifier on the vector representation of the training data, leaving all arguments at their default values.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
clf = SVC()
clf.fit(Xtr_vec, ytr)
```

```
SVC()
```

Compute the accuracy of this model on the test data.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
acc_score = accuracy_score(yts, clf.predict(Xts_vec))
```

Part 2 - use only the most relevant features

Note: Graded variables in this section: `mi_scores`, `threshold`, `Xtr_selected`, `Xts_selected`, `acc_score_median`.

You notice that in its vector representation, the data has several thousand features. You wonder if maybe you can get comparable performance just by using the most relevant features (i.e. the words that are most related to whether or not the claim is true).

Compute a score for each feature in the training data, using `mutual_info_classif` (you can refer to the `mutual_info_classif` documentation here) as the scoring metric. Save the scores in `mi_scores`.

Then, set `threshold` to the 0.5 quantile value of the mutual information scores you computed. Use the `quantile` function in `numpy` (you can find the `quantile` documentation here.)

Let `Xtr_selected` include only the columns of `Xtr_vec` for which the mutual information score is strictly greater than the 0.5 quantile (and similarly, `Xts_selected` will only include those columns of `Xts_vec`.) Use this data to fit an SVC classifier (leaving all arguments at their default values) and compute its accuracy on the test set. Save this value in `acc_score_median`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
mi_scores = mutual_info_classif(Xtr_vec, ytr)
threshold = np.quantile(mi_scores, 0.5)
selected_indices = np.where(mi_scores > threshold)[0]
Xtr_selected = Xtr_vec[:, selected_indices]
Xts_selected = Xts_vec[:, selected_indices]

clf_selected = SVC()
clf_selected.fit(Xtr_selected, ytr)
acc_score_median = accuracy_score(yts, clf_selected.predict(Xts_selected))
```

Part 3 - find the threshold by CV

Note: Graded variables in this section: `acc_scores_kfold`, `best_quantile_mean`, `best_quantile_one_se`.

You still think this idea could work, but maybe excluding half of the features (by using the `0.5` quantile value as the threshold) wasn't exactly right. You decide to use K-fold CV to find a better threshold.

In the following cell, use K-fold CV with `n_folds = 5` to evaluate different quantiles for setting the threshold: `[0.25, 0.5, 0.75, 0.9, 0.95, 0.99]`. Iterate over the K-fold split (don't shuffle the data again!) and in each fold:

- train a `CountVectorizer` (passing `stop_words = 'english'` and leaving other arguments at their default values) using the training data for that fold, then use it to transform the training and validation data for the fold,
- compute the mutual information scores using the training data for that fold (using the default arguments to `mutual_info_classif`, as before),
- compute `threshold_list` as the list of threshold mutual information scores corresponding to the quantiles `[0.25, 0.5, 0.75, 0.9, 0.95, 0.99]`,
- then iterate over `threshold_list`! In this inner loop:
  - fit an SVC classifier using the training data for that fold, selecting only the columns with mutual information score strictly exceeding the threshold,
  - and compute the accuracy score of this model, saving the results in `acc_scores_kfold`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

n_folds = 5
quantiles = np.array([0.25, 0.5, 0.75, 0.9, 0.95, 0.99])

acc_scores_kfold = np.zeros((len(quantiles), n_folds))

# K-fold cross-validation
kf = KFold(n_folds, shuffle=False)
for i, (train_index, val_index) in enumerate(kf.split(Xtr_str)):
    X_train_fold, X_val_fold = Xtr_str[train_index], Xtr_str[val_index]
    y_train_fold, y_val_fold = ytr[train_index], ytr[val_index]

    vec_fold = CountVectorizer(stop_words='english')
    X_train_vec = vec_fold.fit_transform(X_train_fold)
    X_val_vec = vec_fold.transform(X_val_fold)

    mi_scores_fold = mutual_info_classif(X_train_vec, y_train_fold)
    for j, q in enumerate(quantiles):
        threshold_fold = np.quantile(mi_scores_fold, q)
        selected_indices_fold = np.where(mi_scores_fold > threshold_fold)[0]
        X_train_selected = X_train_vec[:, selected_indices_fold]
        X_val_selected = X_val_vec[:, selected_indices_fold]

        clf_fold = SVC()
        clf_fold.fit(X_train_selected, y_train_fold)
        acc_scores_kfold[j, i] = accuracy_score(y_val_fold, clf_fold.predict(X_val_selected))
```

Find the quantile value with the best mean accuracy across folds, and save this value in `best_quantile_mean`:

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
mean_scores = np.mean(acc_scores_kfold, axis = 1)
best_quantile_mean = quantiles[np.argmax(mean_scores)]
```

Find the quantile value that is best according to the one-SE rule (i.e. the simplest model whose mean accuracy is within one standard error of the best model), and save this value in `best_quantile_one_se`:

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
std_scores = np.std(acc_scores_kfold, axis = 1)
one_se_bound = np.max(mean_scores) - np.min(std_scores)
candidate_quantiles = quantiles[mean_scores >= one_se_bound]
best_quantile_one_se = candidate_quantiles[np.argmin(np.abs(mean_scores[mean_scores >= one_se_bound] - np.max(mean_scores)))]
```

Reproducing "Characterization of Term and Preterm Deliveries using Electrohysterograms Signatures"

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from imblearn.over_sampling import ADASYN
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import accuracy_score, recall_score, balanced_accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
mkdir: cannot create directory '/tmp/cache/matplotlib': [Errno 13] Permission denied: '/tmp/cache/matplotlib'
Matplotlib created a temporary cache directory at /tmp/matplotlib-5ot0rr3 because there was an issue with the default path (/tmp/cache/matplotlib); it is highly recommended to set the MPLCONFIGDIR environment variable to a writable directory, in particular to speed up the import of Matplotlib and to better support multiprocessing.
```

We will use the following random state throughout this notebook, in any function calls that involve randomness:

```
random_state = 83
```

In this notebook, we will reproduce the results published in

> M. U. Khan, S. Aziz, S. Ibraheem, A. Butt and H. Shahid, "Characterization of Term and Preterm Deliveries using Electrohysterograms Signatures," 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2019, pp. 0899-0905, doi: 10.1109/IEMCON.2019.8936292, PDF

Preterm birth occurs when a baby is born before 37 weeks of pregnancy. This notebook aims to reproduce the results of a study exploring the use of electrohysterograms (EHG) signatures to predict whether a delivery will be term or preterm.

In this notebook, we will first use an *incorrect* approach described in Khan et al., then we will fix their error and evaluate a correct version of their model.

Read in data

The data we will use comes from the Term-Preterm EHG Database, a collection of EHG signals obtained from 1997 to 2005 at the University Medical Centre Ljubljana, Department of Obstetrics and Gynecology.

Electrohysterogram signature is obtained by placing four electrodes on the abdomen of the mother, to measure the electrical activity of the uterine muscle.

The TPEHG database consists of EHG records obtained from 262 women who had full-term pregnancies and 38 whose pregnancies ended prematurely. The records were obtained during regular check-ups either around the 22nd week of gestation or around the 32nd week of gestation.

In this study, the raw EHG data were further processed to create "transformed" features from each channel: Median frequency, Shannon energy, Log energy, and Lyapunov exponent. We have already pre-computed these features.

Let's read in the data now -

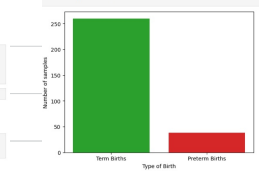
```
features = pd.read_csv("raw_features.csv")        # read in as data frame
y = pd.read_csv("target.csv").squeeze("columns")  # read in as data series
```

We know that there is a class imbalance in this data - there are many more term births (label 1) than pre-term births (label 0). Let's confirm:

```
y.value_counts()
```

```
0    208
1.0    38
0.0    38
Name: count, dtype: int64
```

```
plt.bar(["Term Births","Preterm Births"],height=[y.value_counts()[1],y.value_counts()[0]],color=['tab:green','tab:red']);
plt.ylabel("Number of samples");
plt.xlabel("Type of Birth");
```



Our "features" data file includes many engineered features computed from the raw EHG symbols, we will focus specifically on the features used by Khan et al. in the paper we are trying to reproduce:

```
khan_features = [
    'Features!ager_fmed_ch1', 'Features!ager_max_lyap_ch1',
    'Features!ager_sampen_ch1', 'Features!ager_fmed_ch2',
    'Features!ager_max_lyap_ch2', 'Features!ager_sampen_ch2',
    'Features!ager_fmed_ch3', 'Features!ager_max_lyap_ch3',
    'Features!ager_sampen_ch3',
]
generic_features= [ c for c in features.columns if 'Features!acharya' in c and 'SampleEntropy' in c ]
X = features[khan_features + generic_features]

X.head()
```

	Features!ager_fmed_ch1	Features!ager_max_lyap_ch1	Features!ager_sampen_ch1	Features!ager_fmed_ch2	Features!ager_max_lyap_ch2	Features!ager_sampen_ch2	Features!ager_fmed_ch3	Features
0	0.154949	0.094890	0.676284	0.133788	0.095959	0.519813	0.145392	
1	0.157153	0.097758	0.705688	0.242667	0.108000	0.668954	0.146667	
2	0.205479	0.101883	0.967597	0.149515	0.100426	0.647371	0.164366	
3	1.611604	0.077030	0.768602	0.148123	0.118442	0.678978	1.529010	
4	0.146667	0.107116	0.815574	0.220667	0.116404	0.839187	0.168000	

rows × 405 columns

1. Train an SVM classifier - baseline

Because of the small number of samples, and especially the small number of samples in the minority class, it is not practical to divide this data into a single training set and held-out test set. The results would be highly dependent on the specific division of samples.

Instead, we will *evaluate* our model using a K-fold CV - we will consider the *average* accuracy across folds as our test set accuracy. This is different from *selecting* a model using a validation set inside K-fold CV. We will have a single loop over folds, and in each iteration, we will train the *same* model (on a different training subset) and then evaluate that model on a test subset.

Inside the following cell:

- Set up a `kfold` CV loop with `nfold = 3` splits. Since we have *not* previously shuffled the data, make sure to shuffle it here, and use the random state defined in the `random_state` variable.
- Inside the `kfold` CV loop, use the training and test indices provided by the `kfold` to select the appropriate rows of `X` and `y`. Since they are a `pandas` data frame/series and not a `numpy` array, use `.iloc` to select rows by integer index. Save these subsets of the data as `Xtr_kfold`, `Xts_kfold`, `ytr_kfold`, and `yts_kfold`.
- Then, train an SVM model on the training subset. Use the `rbf` kernel, the random state defined in the `random_state` variable, `gamma = 0.1`, and `C = 10`.
- Get the predictions of the model on the test subset in this fold. Save these in `yhat_ts`.
- We will record four classifier metrics for each fold, but code is already provided for you to use to record:
  - accuracy (saved in `acc_ts_base`)
  - balanced accuracy (saved in `bal_ts_base`)
  - sensitivity (saved in `sen_ts_base`)
  - specificity (saved in `spe_ts_base`)
  - and we will also save the data to generate the confusion matrix (saved in `cm_ts_base`)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
nfold = 3
acc_ts_base = np.zeros(nfold)
bal_ts_base = np.zeros(nfold)
sen_ts_base = np.zeros(nfold)
spe_ts_base = np.zeros(nfold)
cm_ts_base  = np.zeros(shape = (2, 2, nfold))
kf = KFold(n_splits=nfold, shuffle=True, random_state=random_state)
for isplit, (train_index, test_index) in enumerate(kf.split(X, y)):
    Xtr_kfold = X.iloc[train_index]
    Xts_kfold = X.iloc[test_index]
    ytr_kfold = y.iloc[train_index]
    yts_kfold = y.iloc[test_index]
    clf = SVC(kernel='rbf', random_state=random_state, gamma=0.1, C=10)
    clf.fit(Xtr_kfold, ytr_kfold)
    yhat_ts = clf.predict(Xts_kfold)
    acc_ts_base[isplit] = accuracy_score(yts_kfold, yhat_ts)
    bal_ts_base[isplit] = balanced_accuracy_score(yts_kfold, yhat_ts)
    sen_ts_base[isplit] = recall_score(yts_kfold, yhat_ts)
    spe_ts_base[isplit] = recall_score(yts_kfold, yhat_ts, pos_label=0)
    cm_ts_base[:, :, isplit] = confusion_matrix(yts_kfold, yhat_ts, labels=[0, 1])
```

Let's see the average performance:

```
print("Accuracy:     ", acc_ts_base.mean())
```

```
Accuracy:     0.8691245791245791
```

However, since there is a class imbalance, a high accuracy alone does not mean that the classifier is effective.

```
print("Bal Accuracy: ", bal_ts_base.mean())
```

```
Bal Accuracy:  0.5101010101010101
```

We can see some hints about the problem if we look at the model's sensitivity (what proportion of the true positive - term birth - samples are correctly predicted as positive?) and specificity (what proportion of the true negative - pre-term birth - samples are correctly predicted as negative?):

```
print("Sensitivity: ", sen_ts_base.mean())
print("Specificity: ", spe_ts_base.mean())
```

```
Sensitivity:  0.9924242424242425
Specificity:  0.02777777777777776
```

In our K-fold CV loop, we also saved the confusion matrix in each fold. Let's look at those now:

```
f, axes = plt.subplots(1, nfold, figsize = (nfold*5, 5))
for i in range(nfold):
    disp = ConfusionMatrixDisplay(confusion_matrix = cm_ts_base[:, :, i],
                                  display_labels = [0, 1])
    disp.plot(ax=axes[i])
    disp.ax_.set_title("Fold %d" % i)
    disp.im_.colorbar.remove()
```

2. Train an SVM classifier - stratified split

The second problem is easy to fix. To make sure that the ratio of the class labels is similar across folds, we can use a `StratifiedKFold` instead of the `KFold`.

Copy your code from Part 1, but replace the `KFold` with a `StratifiedKFold` to ensure that the distribution of `y` is consistent across folds. Otherwise, keep everything else the same.

You will save the same metrics as before, in:

- accuracy (`acc_ts_strat`)
- balanced accuracy (`bal_ts_strat`)
- sensitivity (`sen_ts_strat`)
- specificity (`spe_ts_strat`)
- and the data to generate the confusion matrix (`cm_ts_strat`)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
nfold = 3
acc_ts_strat = np.zeros(nfold)
bal_ts_strat = np.zeros(nfold)
sen_ts_strat = np.zeros(nfold)
spe_ts_strat = np.zeros(nfold)
cm_ts_strat  = np.zeros(shape = (2, 2, nfold))
skf = StratifiedKFold(n_splits=nfold, shuffle=True, random_state=random_state)
for isplit, (train_index, test_index) in enumerate(skf.split(X, y)):
    Xtr_kfold = X.iloc[train_index]
    Xts_kfold = X.iloc[test_index]
    ytr_kfold = y.iloc[train_index]
    yts_kfold = y.iloc[test_index]
    clf = SVC(kernel='rbf', random_state=random_state, gamma=0.1, C=10)
    clf.fit(Xtr_kfold, ytr_kfold)
    yhat_ts = clf.predict(Xts_kfold)
    acc_ts_strat[isplit] = accuracy_score(yts_kfold, yhat_ts)
    bal_ts_strat[isplit] = balanced_accuracy_score(yts_kfold, yhat_ts)
    sen_ts_strat[isplit] = recall_score(yts_kfold, yhat_ts)
    spe_ts_strat[isplit] = recall_score(yts_kfold, yhat_ts, pos_label=0)
    cm_ts_strat[:, :, isplit] = confusion_matrix(yts_kfold, yhat_ts, labels=[0, 1])
```

```
print("Accuracy:     ", acc_ts_strat.mean())
print("Bal Accuracy: ", bal_ts_strat.mean())
print("Sensitivity:  ", sen_ts_strat.mean())
print("Specificity:  ", spe_ts_strat.mean())
```

```
Accuracy:      0.8658585858585859
Bal Accuracy:  0.5081417624521073
Sensitivity:   0.9885057471264368
Specificity:   0.027777777777777776
```

```
f, axes = plt.subplots(1, nfold, figsize = (nfold*5, 5))
for i in range(nfold):
    disp = ConfusionMatrixDisplay(confusion_matrix = cm_ts_strat[:, :, i],
                                  display_labels = [0, 1])
    disp.plot(ax=axes[i])
    disp.ax_.set_title("Fold %d" % i)
    disp.im_.colorbar.remove()
```

3. Train an SVM classifier - oversampling

The root cause of the first problem - that the SVM classifier does "prediction by mode" - is the class imbalance. To "force" the classifier to focus on both classes, and not only the majority class, we can consider resampling the data so that the class balance is fair. We will mention two resampling techniques -

- **Oversampling**: Generate more samples from the minority class, so that it matches the count of the majority class.
- **Undersampling**: Remove some samples from the majority class, so that it matches the count of the minority class.

In this example, since we have a very small data set, understampling is not practical. We will try oversampling the minority class.

Khan et al. used ADASYN oversampling, so we will too!

Adaptive Synthetic Sampling (ADASYN) is an oversampling technique in which we generate synthetic samples of the minority class, and focus especially on generating synthetic samples for the minority samples which are *harder to learn*.

ADASYN quantifies the learning difficulty of a minority class sample by calculating the proportion of majority class samples among the sample's K nearest neighbors. More majority neighbors imply a *harder to learn* minority sample.

Here is the ADASYN algorithm in detail:

- **Calculate the degree of class imbalance**:

$$d = m_s/m_l$$

where m_s is the number of minority class examples and m_l is the number of majority class examples

- If $d < d_{th}$ (where d_{th} is a preset threshold for the maximum tolerated degree of class imbalance ratio):
  - **Calculate the number of synthetic data examples that need to be generated for the minority class:**

$$G = (m_l - m_s) \times \beta$$

  where β is a parameter used to specify the desired balance level after generation of the synthetic data. $\beta = 1$ means a fully balanced data set is created after generalization process.

  - For each example $x_i \in \text{minority class}$, **find K nearest neighbors based on the Euclidean distance** in n dimensional space, and **calculate the ratio** r_i defined as:

$$r_i = \Delta_i/K,$$

$$i = 1, \dots, m_s$$

  where Δ_i is the number of examples in the K nearest neighbors of x_i that belong to the majority class, therefore $r_i \in [0, 1]$.

  - **Normalize** r_i according to

$$\hat{r}_i = r_i / \sum_{i=1}^{m_s} r_i$$

  - **Calculate the number of synthetic data examples that need to be generated for each minority example** x_i:

$$g_i = \hat{r}_i \times G$$

  - **For each minority class data example** x_i**, generate** g_i **synthetic data examples** according to the following steps:
    - Loop from 1 to g_i :
      - Randomly choose one minority example from the K nearest neighbors for data x_i
      - Generate the synthetic data example:

      $s_i = x_i + (x_{zi} - x_i) \times \lambda$

      where λ is a random number: $\lambda \in [0, 1]$

You may refer to the documentation for the ADASYN oversampling implementation.

It is similar to `sklearn` "transformers" that we have worked with before, like `StandardScaler` or `CountVectorizer`; where those transformers had a `fit_transform` method, the `ADASYN` implementation has `fit_resample`.

In the following cell,

- Create an `ADASYN` overampler. Specify the random state from the `random_state` variable defined above, and `n_neighbors = 5`.
- Use it to generate `X_oversamp` and `y_oversamp`, the oversampled feature data and label data respectively.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
adasyn = ADASYN(random_state=random_state, n_neighbors=5)
X_oversamp, y_oversamp = adasyn.fit_resample(X, y)
```

```
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
```

Then, copy your code from Part 2, but

- split on the oversampled data, not the original data
- when defining `Xtr_fold`, `Xts_kfold`, `ytr_kfold`, `yts_kfold`, select rows from the oversampled data, not the original data.

Otherwise, keep everything else the same.

You will save the same metrics as before, in:

- accuracy (`acc_ts_over`)
- balanced accuracy (`bal_ts_over`)
- sensitivity (`sen_ts_over`)
- specificity (`spe_ts_over`)
- and the data to generate the confusion matrix (`cm_ts_over`)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
nfold = 3
acc_ts_over = np.zeros(nfold)
bal_ts_over = np.zeros(nfold)
sen_ts_over = np.zeros(nfold)
spe_ts_over = np.zeros(nfold)
cm_ts_over  = np.zeros(shape = (2, 2, nfold))
skf = StratifiedKFold(n_splits=nfold, shuffle=True, random_state=random_state)
for isplit, (train_index, test_index) in enumerate(skf.split(X_oversamp, y_oversamp)):
    Xtr_kfold = X_oversamp.iloc[train_index]
    Xts_kfold = X_oversamp.iloc[test_index]
    ytr_kfold = y_oversamp.iloc[train_index]
    yts_kfold = y_oversamp.iloc[test_index]
    clf = SVC(kernel='rbf', random_state=random_state, gamma=0.1, C=10)
    clf.fit(Xtr_kfold, ytr_kfold)
    yhat_ts = clf.predict(Xts_kfold)
    acc_ts_over[isplit] = accuracy_score(yts_kfold, yhat_ts)
    bal_ts_over[isplit] = balanced_accuracy_score(yts_kfold, yhat_ts)
    sen_ts_over[isplit] = recall_score(yts_kfold, yhat_ts)
    spe_ts_over[isplit] = recall_score(yts_kfold, yhat_ts, pos_label=0)
    cm_ts_over[:, :, isplit] = confusion_matrix(yts_kfold, yhat_ts, labels=[0, 1])
```

```
print("Accuracy:     ", acc_ts_over.mean())
print("Bal Accuracy: ", bal_ts_over.mean())
print("Sensitivity:  ", sen_ts_over.mean())
print("Specificity:  ", spe_ts_over.mean())
```

```
Accuracy:      0.9715909090909092
Bal Accuracy:  0.9712628660845954
Sensitivity:   0.9462710505212509
Specificity:   0.99625468164794
```

```
f, axes = plt.subplots(1, nfold, figsize = (nfold*5, 5))
for i in range(nfold):
    disp = ConfusionMatrixDisplay(confusion_matrix = cm_ts_over[:, :, i],
                                  display_labels = [0, 1])
    disp.plot(ax=axes[i])
    disp.ax_.set_title("Fold %d" % i)
    disp.im_.colorbar.remove()
```

4. Train an SVM classifier - oversampling with independent test set

The model trained on an oversampled minority class *seems* to be very effective - but in fact, it only appears so due to a data leakage error.

When we oversample the data, new samples are derived from the existing samples. Since we oversample using the *entire* data set all mixed together:

- *test* samples inside the KFold loop will include "new" synthetic samples derived from samples in the *training* set
- and *training* samples inside the KFold loop include "new" synthetic samples derived from samples in the *test* set

Thus, our evaluation will be overly optimistic - we training a model on samples derived from the "test" data, and evaluating it on samples derived from the training data. This is a much easier "task" than training on a training set, and then evaluating on a totally independent test set.

In the next cell, copy the code from the "oversampling" section, but move the oversampling step inside the KFold loop.

- You will split on the original data, not the oversampled data.
- Create an oversampled training set `Xtr_kfold`, `ytr_kfold` in each fold by oversampling the training data for that fold,
- and create an oversampled test set `Xts_kfold`, `yts_kfold` in each fold by oversampling the test data for that fold.

Keep everything else the same.

You will save the same metrics as before, in:

- accuracy (`acc_ts_ind`)
- balanced accuracy (`bal_ts_ind`)
- sensitivity (`sen_ts_ind`)
- specificity (`spe_ts_ind`)
- and the data to generate the confusion matrix (`cm_ts_ind`)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
nfold = 3
acc_ts_ind = np.zeros(nfold)
bal_ts_ind = np.zeros(nfold)
sen_ts_ind = np.zeros(nfold)
spe_ts_ind = np.zeros(nfold)
cm_ts_ind  = np.zeros(shape = (2, 2, nfold))
skf = StratifiedKFold(n_splits=nfold, shuffle=True, random_state=random_state)
for isplit, (train_index, test_index) in enumerate(skf.split(X, y)):
    Xtr_original = X.iloc[train_index]
    Xts_original = X.iloc[test_index]
    ytr_original = y.iloc[train_index]
    yts_original = y.iloc[test_index]
    adasyn_train = ADASYN(random_state=random_state, n_neighbors=5)
    Xtr_kfold, ytr_kfold = adasyn_train.fit_resample(Xtr_original, ytr_original)
    adasyn_test = ADASYN(random_state=random_state, n_neighbors=5)
    Xts_kfold, yts_kfold = adasyn_test.fit_resample(Xts_original, yts_original)
    clf = SVC(kernel='rbf', random_state=random_state, gamma=0.1, C=10)
    clf.fit(Xtr_kfold, ytr_kfold)
    yhat_ts = clf.predict(Xts_kfold)
    acc_ts_ind[isplit] = accuracy_score(yts_kfold, yhat_ts)
    bal_ts_ind[isplit] = balanced_accuracy_score(yts_kfold, yhat_ts)
    sen_ts_ind[isplit] = recall_score(yts_kfold, yhat_ts)
    spe_ts_ind[isplit] = recall_score(yts_kfold, yhat_ts, pos_label=0)
    cm_ts_ind[:, :, isplit] = confusion_matrix(yts_kfold, yhat_ts, labels=[0, 1])
```

```
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.vali
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
```

```
print("Accuracy:     ", acc_ts_ind.mean())
print("Bal Accuracy: ", bal_ts_ind.mean())
print("Sensitivity:  ", sen_ts_ind.mean())
print("Specificity:  ", spe_ts_ind.mean())
```

```
Accuracy:      0.5401450753711889
Bal Accuracy:  0.5344135161334994
Sensitivity:   0.9808429118773946
Specificity:   0.08798412038960439
```

```
f, axes = plt.subplots(1, nfold, figsize = (nfold*5, 5))
for i in range(nfold):
    disp = ConfusionMatrixDisplay(confusion_matrix = cm_ts_ind[:, :, i],
                                  display_labels = [0, 1])
    disp.plot(ax=axes[i])
    disp.ax_.set_title("Fold %d" % i)
    disp.im_.colorbar.remove()
```


5. Train an SVM classifier - test set with imbalance

In the example above, we evaluated our classifier on an oversampled test set, so that we can directly compare it to the classifier with data leakage.

However, since in real use the model will make predictions on imbalanced (real-life) data, our model evaluation should be similar.

In the following cell, copy the code from the previous section, but this time,

- Create an oversampled training set `Xtr_kfold`, `ytr_kfold` in each fold by oversampling the training data for that fold,
- and create the test set `Xts_kfold`, `yts_kfold` in each fold by selecting rows from the original data for that fold.

Keep everything else the same.

You will save the same metrics as before, in:

- accuracy (`acc_ts_imb`)
- balanced accuracy (`bal_ts_imb`)
- sensitivity (`sen_ts_imb`)
- specificity (`spe_ts_imb`)
- and the data to generate the confusion matrix (`cm_ts_imb`)

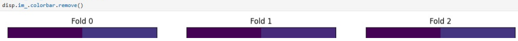
```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
nfold = 3
acc_ts_imb = np.zeros(nfold)
bal_ts_imb = np.zeros(nfold)
sen_ts_imb = np.zeros(nfold)
spe_ts_imb = np.zeros(nfold)
cm_ts_imb  = np.zeros(shape = (2, 2, nfold))
skf = StratifiedKFold(n_splits=nfold, shuffle=True, random_state=random_state)
for isplit, (train_index, test_index) in enumerate(skf.split(X, y)):
    # 选取训练集和测试集的原始数据
    Xtr_original = X.iloc[train_index]
    Xts_original = X.iloc[test_index]
    ytr_original = y.iloc[train_index]
    yts_original = y.iloc[test_index]
    adasyn = ADASYN(random_state=random_state, n_neighbors=5)
    Xtr_kfold, ytr_kfold = adasyn.fit_resample(Xtr_original, ytr_original)
    Xts_kfold = Xts_original
    yts_kfold = yts_original
    clf = SVC(kernel='rbf', random_state=random_state, gamma=0.1, C=10)
    clf.fit(Xtr_kfold, ytr_kfold)
    yhat_ts = clf.predict(Xts_kfold)
    acc_ts_imb[isplit] = accuracy_score(yts_kfold, yhat_ts)
    bal_ts_imb[isplit] = balanced_accuracy_score(yts_kfold, yhat_ts)
    sen_ts_imb[isplit] = recall_score(yts_kfold, yhat_ts)
    spe_ts_imb[isplit] = recall_score(yts_kfold, yhat_ts, pos_label=0)
    cm_ts_imb[:, :, isplit] = confusion_matrix(yts_kfold, yhat_ts, labels=[0, 1])
```

```
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.valida
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.valida
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
/opt/conda/lib/python3.12/site-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.valida
te_data` instead. This function becomes public and is part of the scikit-learn developer API.
  warnings.warn(
```

```
print("Accuracy:     ", acc_ts_imb.mean())
print("Bal Accuracy: ", bal_ts_imb.mean())
print("Sensitivity:  ", sen_ts_imb.mean())
print("Specificity:  ", spe_ts_imb.mean())
```

```
Accuracy:      0.8591582491582491
Bal Accuracy:  0.5043103448275862
Sensitivity:   0.9808429118773946
Specificity:   0.027777777777777776
```

```
f, axes = plt.subplots(1, nfold, figsize = (nfold*5, 5))
for i in range(nfold):
    disp = ConfusionMatrixDisplay(confusion_matrix = cm_ts_imb[:, :, i],
                                  display_labels = [0, 1])
    disp.plot(ax=axes[i])
    disp.ax_.set_title("Fold %d" % i)
    disp.im_.colorbar.remove()
```



Autodiff

```
import numpy as np
from IPython.display import Image, display, SVG
# Utility functions I wrote for visualization
from vis import vis_forward_pass, vis_backward_pass
```

We saw how the backpropagation algorithm can be used to find gradients for any neural network, as long as each step in its computational graph is differentiable. Working from the end of the computational graph (output) toward the beginning (input), we can use the chain rule to get the derivate of the output with respect to each weight.

However, outside of toy examples, we don't train modern neural networks by working out these derivatives and then implementing them in code. Modern neural networks are enabled by frameworks for **automatic differentiation** (like TensorFlow and Pytorch), which build the computational graph and apply the backpropagation algorithm to it without us having to compute derivatives by hand.

In this exercise, we will build our own very basic framework for automatic differentiation, to better understand how it works.

Build a Variable class and some operations

In our framework, a `Variable` will represent a node in the computational graph and holds these essential pieces of information:

- `value` - the value of the variable.
- `local_gradients` - the local derivative, i.e. the derivative of the operation that this node performs with respect to its arguments.
- `accumulated_gradient` - the accumulated gradient we will compute in the backwards pass along the computational graph.

(We'll also add a `name` attribute so that we can visualize our computational graph.)

Here is our `Variable` class:

```
class Variable:
    def __init__(self, value, local_gradients=(), name=None):
        self.value = value
        self.local_gradients = local_gradients
        self.accumulated_gradient = 0
        self.name = name or f"Var_{id(self)}"
```

Of course, we want to support some operations on our variables. Let's implement addition (`add`) and multiplication (`mul`).

For each operation, we should return a new instance of a `Variable` with:

- the `value` that is the output of this operation for the forward pass on the computational graph,
- and the `local_gradients` of this operation, that we can use for the backward pass on the computational graph.

```
def add(a, b):
    "Create the variable that results from adding two variables."
    value = a.value + b.value
    local_gradients = (
        (a, 1),  # the local derivative with respect to a is 1
        (b, 1)  # the local derivative with respect to b is 1
    )
    return Variable(value, local_gradients, name=f"(a.name)+(b.name)")

def mul(a, b):
    "Create the variable that results from multiplying two variables."
    value = a.value * b.value
    local_gradients = (
        (a, b.value),  # the local derivative with respect to a is b.value
        (b, a.value)  # the local derivative with respect to b is a.value
    )
    return Variable(value, local_gradients, name=f"(a.name)*(b.name)")
```

Now, we can use this basic framework to do a forward pass on a graph. We'll define two variables, `a` and `b`, and give them values:

```
a = Variable(4, name='a')
b = Variable(5, name='b')
```

Note that both of these have "empty" `local_gradients` and zero `accumulated_gradient`, to start:

```
a.local_gradients, b.local_gradients
```

```
((), ())
```

```
a.accumulated_gradient, b.accumulated_gradient
```

```
(0, 0)
```

If we `add` the variables:

```
c = add(a, b)
```

we get a new `Variable` whose `value` is the output of the "forward pass":

```
c.value
```

```
9
```

and also has `local_gradients` with respect to the variables `a` and `b`:

```
for grad in c.local_gradients:
    print('Gradient with respect to', grad[0].name, '=', grad[1])
```

```
Gradient with respect to a = 1
Gradient with respect to b = 1
```

```
vis_forward_pass(c)
```

```
<graphviz.graphs.Digraph at 0x7f08195e7070>
```

Note: each node in this graph is its own `Variable`.

Build the backward pass

We will need a `get_gradients` function. It will operate as follows, using recursion:

- First, it will start at the final output node of the computational graph. We will call `compute_gradients` on that output `Variable`. We will pass a `path_value` of 1 to `compute_gradients` - this is the gradient of that output `Variable` with respect to itself. As we traverse the graph, we will be accumulating the gradient of this output variable with respect to each node in the graph.
- Inside the `compute_gradients` function, we get the `local_gradients` of the node and iterate over them - these are the child nodes of this node when traversing the graph for the backward pass. For each child node and the gradient with respect to that child node:
  - we multiply the values along the edges: the `path_value` up until this node, times the `local_gradient` of this node with respect to this child.
  - and then we add this to the `accumulated_gradient` of the child node - i.e. we will sum the path values for all paths to this child.
  - finally, we call `compute_gradients` again on this child node, passing the `path_value` we have computed, to continue recursing on the graph.
- When we reach the input nodes, they do not have any `local_gradients` so the recursion will end.

```
def get_gradients(output_variable):
    """Compute the first derivatives of `variable`
    with respect to child variables.
    """
    def compute_gradients(variable, path_value):
        for child_variable, local_gradient in variable.local_gradients:
            # "Multiply the edges of a path":
            value_of_path_to_child = path_value * local_gradient
            # "Add together the different paths":
            child_variable.accumulated_gradient += value_of_path_to_child
            # recurse through graph:
            compute_gradients(child_variable, value_of_path_to_child)

    # start here
    compute_gradients(output_variable, path_value=1)
    # (path_value=1 is from `variable` differentiated w.r.t. itself)
    output_variable.accumulated_gradient = 1
```

Let's try it now. We will use the example from Figure 2D in the lecture handout.

```
a = Variable(2, name='a')
b = Variable(1, name='b')
c = add(a, b)
d = add(b, Variable(1, name='1'))
e = mul(c, d)
```

```
e.value
```

```
6
```

```
vis_forward_pass(e)
```

```
<graphviz.graphs.Digraph at 0x7f08195e5d80>
```

```
get_gradients(e)
```

```
[e.accumulated_gradient, b.accumulated_gradient]
```

```
(2, 5)
```

```
vis_backward_pass(e)
```

```
<graphviz.graphs.Digraph at 0x7f08195e56b0>
```

(you can confirm that this matches Figure 2D in the lecture handout.)

We have correctly computed the value in the forward pass and the backward pass.

Extend the framework

Using the framework we have developed so far, implement these additional operations, similar to the way `add` and `mul` are implemented above:

- `sub` (subtract)
- `square` (square)
- `sigmoid`

Each function should return a new `Variable` with a correct `value` and tuple of `local_gradients`, as well as a `name`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
def sub(a, b):
    "Create the variable that results from subtracting a-b."
    value = a.value - b.value
    local_gradients = (
        (a, 1),  # the local derivative with respect to a is 1
        (b, -1)  # the local derivative with respect to b is -1
    )
    return Variable(value, local_gradients, name=f"(a.name)-(b.name)")
```

Note: for a function that takes a single input, `local_gradients` will have one element but should still be defined as a tuple. When you define `local_gradients`, you will use e.g.

```
local_gradients = (
    (7, 7),
)
```

with the comma being strictly required in order for it to be a tuple and not just a value.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
def square(a):
    "Create the variable that results from squaring the variable a, i.e. a^2."
    value = a.value ** 2
    local_gradients = (
        (a, 2 * a.value),  # the local derivative with respect to a is 2*a
    )
    return Variable(value, local_gradients, name=f"(a.name)^2")
```

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
def sigmoid(a):
    "Create the variable that results from applying a sigmoid function to a."
    value = 1 / (1 + np.exp(-a.value))
    local_gradients = (
        (a, value * (1 - value)),  # the local derivative of sigmoid
    )
    return Variable(value, local_gradients, name=f"sigmoid(a.name)")
```

Build a neural network

With these new operations, we can construct the computational graph shown in Figure 1S in the lecture handout, where:

data sample -

- let the value of `x` be 4
- let the value of `y` be 1

initial weights -

- let the value of `wh` be -2
- let the value of `wo` be 3

activation and loss functions -

- let the activation function at the hidden unit be sigmoid
- let the activation function at the output unit be linear
- use the mean squared error loss function: $(y - \hat{y})^2$

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

# Input Variables
x = Variable(4, name='x')
y = Variable(1, name='y')
wh = Variable(-2, name='wh')
wo = Variable(3, name='wo')

# Variables created by operations on those input Variables
zh = mul(x, wh)
uh = sigmoid(zh)
zo = mul(uh, wo)
uo = zo
L = square(sub(uo, y))
```

Visualize the forward pass:

```
vis_forward_pass(L)
```

```
<graphviz.graphs.Digraph at 0x7f08199e6f00>
```

Then, do a backward pass to get the gradient of the loss function with respect to each of the weights.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
get_gradients(L)
```

```
vis_backward_pass(L)
```

```
<graphviz.graphs.Digraph at 0x7f08199e61b0>
```

Construct and compile model

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation, Input
from tensorflow.keras import optimizers
import tensorflow.keras.backend as K
import tensorflow as tf
```

Construct a neural network following the problem parameters described in the question.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

model = Sequential()
input_dim = 9
model.add(Dense(10, input_dim=input_dim, activation='relu', name='hidden'))
model.add(Dense(1, activation='linear', name='output'))
```

Create an optimizer and compile the model. Select the appropriate loss function for this type of problem, and use an appropriate metric.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

opt = optimizers.Adam()
model.compile(optimizer=opt, loss='mean_squared_error', metrics=['mse'])
```

PCA for dimensionality reduction before classification

```
import numpy as np
import random

from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Upload the `data.csv` file to this workspace, then read in the data to a `numpy` array in `X` and the labels to `y`. If you want to, you can add code to the following cell to explore `X` (for example, see its shape).

```
dat = np.genfromtxt('data.csv',delimiter=',')
X = dat[:, 1:]
y = dat[:, -1]
```

Use `train_test_split` to split the data into training and test sets. Reserve 30% of the data for the test set.

Make sure to shuffle the data, and pass `random_state = 42` so that your random split will match the auto-grader's.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True, random_state=42)
```

You will use the training data to fit a support vector classifier. However, instead of fitting the training data directly, you will first transform it using PCA. Then, you will use only a subset of features - the first `n_comp` principal components - as input to your classifier.

You will use K-fold cross validation to find the optimal value of `n_comp`. You should consider every possible value of `n_comp`, from 1 component (simplest possible model) to all of the components (most flexible model).

In the next cell,

- Use the `sklearn` implementation of `KFold` to iterate over candidate models. In your `KFold`, use 5 splits, and don't shuffle the data (you already shuffled it when dividing into training and test.)
- Use the `sklearn` implementation of `PCA` to transform the data. Pass `random_state = 42` to `PCA` so that your result will match the auto-grader's.
- Use the `sklearn` implementation of `SVC` to classify the data using the first `n_comp` principal components. Pass `random_state = 42` to `SVC` so that your result will match the auto-grader's.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
n_components = np.arange(1, X_train.shape[1] + 1)
acc_mean = np.zeros(len(n_components))
acc_se = np.zeros(len(n_components))
kf = KFold(n_splits=5, shuffle=False)

# Initialize accuracy lists for each n_components
accs = [[] for _ in n_components]

for train_index, val_index in kf.split(X_train):
    X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    # Fit PCA with max components ONCE per fold
    pca_full = PCA(n_components=X_train.shape[1], random_state=42)
    X_train_pca_full = pca_full.fit_transform(X_train_fold)
    X_val_pca_full = pca_full.transform(X_val_fold)

    for j, n_comp in enumerate(n_components):
        # Slice the PCA-transformed data for the current n_components
        X_train_pca = X_train_pca_full[:, :n_comp]
        X_val_pca = X_val_pca_full[:, :n_comp]

        svc = SVC(random_state=42)
        svc.fit(X_train_pca, y_train_fold)
        y_pred = svc.predict(X_val_pca)
        accuracy = accuracy_score(y_val_fold, y_pred)

        accs[j].append(accuracy)
```

Compute the mean validation accuracy and the standard error of the mean validation accuracy across the folds. Save the results in `acc_mean` and `acc_se`, respectively.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
acc_mean = np.array([np.mean(acc) for acc in accs])
acc_se = np.array([np.std(acc, ddof=1) / np.sqrt(len(acc)) for acc in accs])
```

Then, compute the optimal value of `n_comp`, and save this in `n_pca_opt`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
n_pca_opt = n_components[np.argmax(acc_mean)]
```

Finally, compute the optimal `n_comp` according to the one-SE rule, and save this in `n_pca_one_se`.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
best_acc = np.max(acc_mean)
best_acc_se = acc_se[np.argmax(acc_mean)]
threshold = best_acc - best_acc_se
n_pca_one_se = n_components[np.where(acc_mean >= threshold)[0][0]]
```

K means clustering for portfolio diversity

```
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
```

Read in the `stock_movement.csv` dataset. Each row in this data corresponds to one stock, and each column corresponds to one day of trading. The data is indexed with its "ticker", or stock symbol.

```
df = pd.read_csv("stock_movement.csv", index_col=0)
df.head()
```

	2020-01-02	2020-01-03	2020-01-06	2020-01-07	2020-01-08	2020-01-09	2020-01-10	2020-01-13	2020-01-14	2020-01-15	...	2022-12-28	2022-12-29	2022-12-30	2023-01-03	2023-01-04	2023-01-05	2023-01-06	2023-01-09
AAPL	0.007886	0.000537	0.011531	-0.002782	0.011570	0.004586	-0.000518	0.010207	-0.007713	-0.000979	...	-0.027860	0.012433	0.011666	-0.039986	-0.004068	-0.016194	0.027706	-0.002451
ADBE	0.009704	0.005783	0.011873	-0.001665	0.008894	0.000679	-0.004907	0.008960	-0.005214	-0.007623	...	-0.014743	0.014962	0.007733	-0.007098	-0.003242	-0.019913	0.001030	0.006747
ADI	0.002177	0.002109	0.004558	0.004627	0.002381	-0.006600	-0.015172	0.001905	0.003742	-0.012927	...	-0.008301	0.009525	0.011362	-0.021228	0.003606	-0.032522	0.025582	-0.001975
ADP	-0.008449	0.009789	0.005719	-0.005976	0.004740	0.004276	-0.004688	0.004328	-0.004585	0.003812	...	-0.018032	0.012571	-0.004328	-0.016126	-0.007161	-0.019938	0.021535	-0.008295
ADSK	0.015114	0.001920	0.013945	0.005720	0.007223	0.000960	-0.003340	0.010229	-0.004760	-0.000626	...	-0.010813	0.019414	0.008726	-0.022837	0.000292	-0.016617	0.012358	0.008147

5 rows × 763 columns

```
X = df.values
```

You will fit clusters across a range of different "number of clusters", so that the finance experts can decide which value for "number of clusters" is most useful for them. This is the list of values you will consider:

```
n_cluster_list = [3, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

For each value in `n_cluster_list`:

- Use the `sklearn` implementation of K means clustering to assign each of the samples in `X` to a cluster, specifying the number of clusters to fit.
- Also specify `random_state=42` and `n_init=1`, but leave other arguments at their default values.
- Save the cluster indices assigned by the model to the corresponding column in `c_idx_def`
- Then, find the size of each of the clusters. (Hint: you can use `np.unique`.)
- For each cluster that has fewer than the required number of samples (specified on the question page), re-assign that sample to the closest cluster that *did* have enough samples assigned by the default model. ("Closest" according to "minimum Euclidean distance to the cluster center". Hint: use `np.linalg.norm`.)
- Save your new cluster assignments (for all samples, not just the ones you modified) to the corresponding column in `c_idx_mod`.

For full credit, you should use no more than three `for` loops.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

# initialize graded arrays
c_idx_def = np.zeros((X.shape[0], len(n_cluster_list)))
c_idx_mod = np.zeros((X.shape[0], len(n_cluster_list)))

# fill in the rest of your code here...
for i, n_clusters in enumerate(n_cluster_list):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=1)
    labels = kmeans.fit_predict(X)
    c_idx_def[:, i] = labels

    cluster_sizes = np.bincount(labels)
    valid_clusters = np.where(cluster_sizes >= 3)[0]
    invalid_clusters = np.where(cluster_sizes < 3)[0]

    new_labels = labels.copy()
    for invalid_cluster in invalid_clusters:
        invalid_samples = np.where(labels == invalid_cluster)[0]
        distances = np.linalg.norm(X[invalid_samples][:, np.newaxis] - kmeans.cluster_centers_[valid_clusters], axis=2)
        closest_clusters = valid_clusters[np.argmin(distances, axis=1)]
        new_labels[invalid_samples] = closest_clusters

    c_idx_mod[:, i] = new_labels
```

Multi-level regression using K means clustering

```
import random
import numpy as np
import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

Read in the dataset:

```
df = pd.read_csv('data-volunteer.csv')
```

The data has the following features, some of which you will use for clustering and some of which you will use for the regression model:

- `GEODIV` (geographical region of the US where the respondent lives, ordinal-encoded. You will use this for clustering only.)
- `GTMETSTA` (whether or not the respondent lives in a metropolitan area. You will use this for clustering only.)
- `GTCBSASZ` (size of the metro area where the respondent lives. You will use this for clustering only.)
- `PESEX` (sex of the respondent. You will use this for the regression only.)
- `PRTAGE` (age of the respondent, ordinal encoded. You will use this for the regression only.)
- `PEEDUCA` (education level of the respondent, ordinal encoded. You will use this for the regression only.)
- `PEMLR` (whether the respondent worked in the last week (1), did not work in the last week (2), or is retired (3). You will use this for the regression only.)
- `PTS1OT` (number of hours spent volunteering in the last 12 months. You will use this as the target variable for the regression.)

Split the data into training and test sets. Use 2,500 samples for the test set and the remaining samples for the training set. Use `random_state = 42`.

- `ytr` and `yts` should each be a 1d `numpy` array with only the target variable.
- `Xtr` and `Xts` should be `pandas` data frames with all of the remaining variables (excluding the target variable.)

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
regression_features = ['PESEX', 'PRTAGE', 'PEEDUCA', 'PEMLR']
clustering_features = ['GEDIV', 'GTMETSTA', 'GTCBSASZ']
target = 'PTS1OT'

X = df[regression_features + clustering_features]
y = df[target]
Xtr, Xts, ytr, yts = train_test_split(X, y, test_size=2500, random_state=42)
ytr = ytr.to_numpy()
yts = yts.to_numpy()
```

In the next cells, you will use `sklearn` to perform K-means clustering using `Xtr`. First, set `n_cluster` as specified on the question page.

```
n_cluster = 5
```

Then, assign cluster labels to each data point, using only the geographical features that were specified as "You will use this for clustering only".

(Use the `random_state = 42` as shown below so that your clustering will match the auto-grader's.) Save the assigned class labels in `Xtr_cid` and `Xts_cid` for the training and test data, respectively.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
kmeans = KMeans(n_clusters=n_cluster, random_state=42)
Xtr_cid = kmeans.fit_predict(Xtr[clustering_features])
Xts_cid = kmeans.predict(Xts[clustering_features])
```

Finally, fit regression coefficients using the training data in each cluster, and then use the fitted regression models to create `yhat_ts`, the predicted values on the test set.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)

# this just generates an array that's the correct shape - yhat_ts shouldn't really be all zeros
yhat_ts = np.zeros(yts.shape)

for i in range(n_cluster):
    train_indices = np.where(Xtr_cid == i)[0]
    test_indices = np.where(Xts_cid == i)[0]
    if len(train_indices) > 0:
        reg = LinearRegression()
        reg.fit(Xtr.iloc[train_indices][regression_features], ytr[train_indices])
        yhat_ts[test_indices] = reg.predict(Xts.iloc[test_indices][regression_features])
```

Then, compute the mean squared error of your model on the test data.

```
#grade (write your code in this cell and DO NOT DELETE THIS LINE)
mse_ts = mean_squared_error(yts, yhat_ts)
```

Husky vs wolf classifier ¶

In this question, you will create a convolutional neural network using Tensorflow with Keras, then train the model to distinguish between "husky" and "wolf".

```
import keras
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation, Input, Conv2D, MaxPooling2D, Flatten, Rescaling
from tensorflow.keras import optimizers
```

First, we will use the `keras` utility function to read in data from the `data` directory, using `image_dataset_from_directory`.

```
image_size = (256,256)
batch_size = 4

train_ds = keras.utils.image_dataset_from_directory(
    'data/train',
    label_mode="binary",
    class_names = ["husky", "wolf"],
    image_size=image_size,
    batch_size=batch_size,
    seed=27,
    validation_split=0.125,
    subset='training'
)
val_ds = keras.utils.image_dataset_from_directory(
    'data/train',
    label_mode="binary",
    class_names = ["husky", "wolf"],
    image_size=image_size,
    batch_size=batch_size,
    seed=27,
    validation_split=0.125,
    subset='validation'
)
test_ds = keras.utils.image_dataset_from_directory(
    'data/test',
    label_mode="binary",
    class_names = ["husky", "wolf"],
    image_size=image_size,
    batch_size=batch_size,
    seed=27,
    shuffle=False
)
```

Next, you will create a model and compile it. While the details are up to you, your model must:

- be a `Sequential` type model
- accept as input images that are `(256, 256, 3)`
- have a `Rescaling` layer immediately after the input
- have one or more convolutional layers with non-linear activation
- have one or more max pooling layers following a convolution + activation
- end with a dense layer
- use appropriate activation and loss function at the output
- have fewer than 250,000 parameters
- use the `accuracy` metric (when you compile it)

```
model = Sequential([
    Input(shape=(256, 256, 3)),
    Rescaling(1./255),

    Conv2D(4, (3, 3), activation='relu'),
    MaxPooling2D(),

    Conv2D(8, (3, 3), activation='relu'),
    MaxPooling2D(),

    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D(),

    Flatten(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

```
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Fit your model for no more than 20 epochs, using `train_ds` and/or `val_ds`.

```
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20
)
```

```
acc_ts = model.evaluate(test_ds)[1]
```

```
5/5 [==============================] - 0s 99ms/step - loss: 0.1499 - accuracy: 0.9500
```

Your model should achieve at least 80% accuracy on the test data.

Save your model as `model.keras`. The grader will evaluate your saved model file.

```
if model.count_params() <= 250000:
    model.save("model.keras")
else:
    print("Your model is required to have fewer than 250,000 parameters.")
```

Explain your model

To better understand your model performance, you will use GradCam, a technique that highlights the areas of an image that the model "focuses" on the most to make its prediction.

The following cell will sample random images from the test set, apply GradCam, and show the GradCam heatmap overlaid on top of the original image. Areas highlighted in red or yellow are especially relevant, according to the model.

You may run this more than once to sample different images.

```
import matplotlib.pyplot as plt
from matplotlib import cm
from tf_keras_vis.gradcam import Gradcam
import numpy as np
import os

# Define image titles for visualization
image_titles = ['husky', 'wolf']

# Create lists of file paths for husky and wolf images
husky_files = np.array(['data/test/husky/'+x for x in os.listdir('data/test/husky')])
wolf_files = np.array(['data/test/wolf/'+x for x in os.listdir('data/test/wolf')])

# Load random images for each class and convert them to a numpy array
husky = keras.utils.load_img(np.random.choice(husky_files), target_size=image_size)
wolf = keras.utils.load_img(np.random.choice(wolf_files), target_size=image_size)
images = np.asarray([np.array(husky), np.array(wolf)])
X = np.array([keras.utils.img_to_array(img) for img in images])

# Render the original images
f, ax = plt.subplots(nrows=1, ncols=2, figsize=(6, 6))
for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].axis('off')
plt.tight_layout()
plt.show()

# Define a function to modify the model for GradCAM
def model_modifier_function(cloned_model):
    cloned_model.layers[-1].activation = keras.activations.linear

# Define a score function for GradCAM
def score_function(output):
    if output.shape[1] == 1:
        return (output[0], output[1])
    elif output.shape[1] == 2:
        return (output[0,0], output[1,1])

# Create GradCam object
gradcam = Gradcam(model, model_modifier=model_modifier_function, clone=True)

# Generate heatmap with GradCAM
cam = gradcam(score_function, X)

# Render the images with GradCAM heatmaps overlaid
f, ax = plt.subplots(nrows=1, ncols=2, figsize=(6, 6))
for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :3] * 255)
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].imshow(heatmap, cmap='jet', alpha=0.5)
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```