**Question 1:**

Which of the following statement(s) is/are true about ensemble models? Select all that apply

a. By combining multiple machine learning models, ensembles produce models with enhanced performance.

b. The scikit-learn implementation for RandomForestClassifier() uses Gini as the default function to measure the quality of the split.

c. None of the above.

**Answer: b.**

a. By combining multiple machine learning models, ensembles produce models with enhanced performance.

Ensemble methods often enhance performance, but this is not guaranteed in all cases. The effectiveness of an ensemble depends on factors

such as:

The diversity and quality of the individual models.

The method used to combine the models.

The specific problem and dataset.

In some cases, an ensemble might not improve performance or could even degrade it if the individual models are not diverse enough or if the

ensemble method is not well-suited to the problem.

Therefore, while ensembles often improve performance, it is not an absolute certainty.

A couple of examples below

import numpy as np

import pandas as pd

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import accuracy\_score

# Generate a synthetic dataset for the example

X, y = make\_classification(n\_samples=1000, n\_features=20, random\_state=42)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Single Logistic Regression Model

single\_model = LogisticRegression(max\_iter=1000, random\_state=42)

single\_model.fit(X\_train, y\_train)

single\_y\_pred = single\_model.predict(X\_test)

single\_accuracy = accuracy\_score(y\_test, single\_y\_pred)

print(f'Single Logistic Regression Model Accuracy: {single\_accuracy:.2f}')

# Ensemble Model using Bagging with Logistic Regression

ensemble\_model = BaggingClassifier(estimator=LogisticRegression(max\_iter=1000, random\_state=42),

n\_estimators=10, random\_state=42)

ensemble\_model.fit(X\_train, y\_train)

ensemble\_y\_pred = ensemble\_model.predict(X\_test)

ensemble\_accuracy = accuracy\_score(y\_test, ensemble\_y\_pred)

print(f'Ensemble Model (Bagging with Logistic Regression) Accuracy: {ensemble\_accuracy:.2f}')

Single Logistic Regression Model Accuracy: 0.85

Ensemble Model (Bagging with Logistic Regression) Accuracy: 0.84

import numpy as np

import pandas as pd

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import accuracy\_score

# Generate a synthetic dataset for the example

X, y = make\_classification(n\_samples=1000, n\_features=20, random\_state=42)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Single Logistic Regression Model

single\_model = LogisticRegression(max\_iter=1000, random\_state=42)

single\_model.fit(X\_train, y\_train)

single\_y\_pred = single\_model.predict(X\_test)

single\_accuracy = accuracy\_score(y\_test, single\_y\_pred)

print(f'Single Logistic Regression Model Accuracy: {single\_accuracy:.2f}')

# Hard Voting Ensemble with Logistic Regression

ensemble\_model = VotingClassifier(estimators=[

('lr1', LogisticRegression(max\_iter=1000, random\_state=42)),

('lr2', LogisticRegression(max\_iter=1000, random\_state=43)),

('lr3', LogisticRegression(max\_iter=1000, random\_state=44))

], voting='hard')

ensemble\_model.fit(X\_train, y\_train)

ensemble\_y\_pred = ensemble\_model.predict(X\_test)

ensemble\_accuracy = accuracy\_score(y\_test, ensemble\_y\_pred)

print(f'Hard Voting Ensemble Model Accuracy: {ensemble\_accuracy:.2f}')

Single Logistic Regression Model Accuracy: 0.85

Hard Voting Ensemble Model Accuracy: 0.85

b. The scikit-learn implementation for RandomForestClassifier() uses gini as the default function to measure the quality of the split.

Check: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

In particular the argument criterion

**Question 2**

Using scikit-learn and our cancer dataset (the one we used in our laboratory to classify between malignant and benign tumors), we would like to

build a random forest classifier that performs hard voting with the individual predictions of each tree. To do this, we will use the program shown

below that utilises RandomForestClassifier() and specifically its fit() and predict() methods.

We want to train a model with 200 trees; a minimum of 5 samples to split an internal node; 2 as the minimum number of samples required to be

at a leaf node; and a random state of 42. To train each base estimator, we will bootstrap 80% of the original training set, that is, 80% is the

fraction of the training set to be used for each bootstrap sample.

Select all that apply:

##################################################

# Docstring:

# data: Bunch

# Dictionary-like object, with the following attributes.

# data{ndarray, dataframe} of shape (569, 30)

# The data matrix. If as\_frame=True, data will be a pandas DataFrame.

# target{ndarray, Series} of shape (569,)

# The classification target. If as\_frame=True, target will be a pandas Series.

# feature\_namesndarray of shape (30,)

# The names of the dataset columns.

# target\_namesndarray of shape (2,)

# The names of target classes.

# frameDataFrame of shape (569, 31)

# Only present when as\_frame=True. DataFrame with data and target.#

# For more information:

# scikit-learn.org/stable/modules/generated/sklearn.datasets.load\_breast\_cancer.html

# X: (...)

# y

# (...)

##################################################

# Import libraries from sklearn:

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report

# Load the dataset

data = load\_breast\_cancer()

X, y = data.data, data.target

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Select all the answers that apply from the options shown below (a,b,c or d).

################################################################

# a)

# Start of code for option a)

# Create RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=200, min\_samples\_split=2, min\_samples\_leaf=5, random\_state=42

#Train the RandomForestClassifier

rf.fit(X\_train, y\_train)

# Make predictions

y\_pred\_training = rf.predict (X\_train)

y\_pred\_test = rf.predict(X\_test)

# End of code for option a)

################################################################

################################################################

# b)

# Start of code for option b)

# Create RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=200, min\_samples\_split=5, min\_samples\_leaf=2, random\_state=42, max\_samples=0.8)

#Train the RandomForestClassifier

rf.fit(X\_train, y\_train)

# Make predictions

y\_pred\_training = rf.predict (X\_train)

y\_pred\_test = rf.predict(X\_test)

# End of code for option b)

################################################################

################################################################

# c)

# Start of code for option c)

# Create RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=200, min\_samples\_split=5, max\_leaf\_nodes=2, random\_state=42, max\_samples=0.8)

#Train the RandomForestClassifier

rf.fit(X\_train, y\_train)

# Make predictions

y\_pred\_training = rf.predict (X\_train)

y\_pred\_test = rf.predict(X\_test)

# End of code for option c)

################################################################

# There is a 4th of option:

# d) None of the above.

# Rewrite the whole program.

################################################################

# Evaluate the performance

# Print classification reports

print('Classification Report - Training Set:')

print(classification\_report(y\_train, y\_pred\_training))

print('Classification Report - Test Set:')

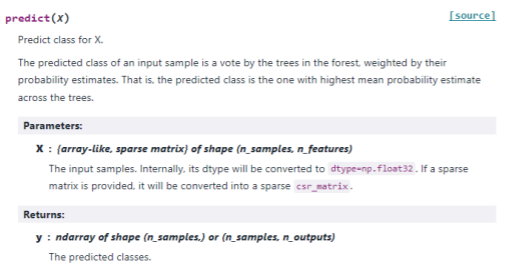
print(classification\_report(y\_test, y\_pred\_test))

**Answer: d**

Feedback:

The scikit-learn RandomForestClassifiers() does not perform hard voting. Let's check the definition of the method predict() to double

check that.



scikit-learn 中的 `RandomForestClassifier` 不会像 `VotingClassifier` 那样执行硬投票。相反，它使用不同的机制进行预测。

让我们分解一下要点：

### RandomForestClassifier 的预测方法

`RandomForestClassifier` 的 `predict` 方法汇总森林中每棵树的预测。每棵树为每个样本提供一个预测，最终预测基于多数投票（即在所有树中获得最多投票的类）。这是一种集成学习的形式，但与不同分类器之间的硬投票不同。

来自 scikit-learn 文档中 `RandomForestClassifier` 的 `predict` 方法（如您提供的图像所示）：

- 该方法获取输入样本 `X`，并通过汇总森林中所有树的投票返回预测的类。

### 硬投票

另一方面，硬投票通常是指将来自多个不同分类器（例如 RandomForest、SVM、LogisticRegression）的预测组合起来，并针对每个样本在这些分类器中进行多数投票。这是使用 scikit-learn 中的 `VotingClassifier` 完成的。

### 所提供选项的评估

- \*\*选项 a)\*\*：此选项不符合要求，因为它设置了 `min\_samples\_split=2` 和 `min\_samples\_leaf=5`，与所需参数不匹配。

- \*\*选项 b)\*\*：此选项设置 `n\_estimators=200`、`min\_samples\_split=5`、`min\_samples\_leaf=2`、`random\_state=42` 和 `max\_samples=0.8`。虽然它正确地设置了 `RandomForestClassifier`，但它没有在结合不同分类器的预测的背景下实现硬投票。

- \*\*选项 c)\*\*：此选项不正确，因为它设置了 `max\_leaf\_nodes=2`，而这在要求中并未指定。

### 硬投票的正确实现

要使用多个分类器实现硬投票，您可以使用 `VotingClassifier`，如下所示：

```python

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

# Load the dataset

data = load\_breast\_cancer()

X, y = data.data, data.target

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the individual classifiers

rf = RandomForestClassifier(n\_estimators=200, min\_samples\_split=5, min\_samples\_leaf=2, random\_state=42, max\_samples=0.8)

lr = LogisticRegression(random\_state=42)

svc = SVC(probability=True, random\_state=42)

# Create the VotingClassifier

voting\_clf = VotingClassifier(estimators=[('rf', rf), ('lr', lr), ('svc', svc)], voting='hard')

# Train the VotingClassifier

voting\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred\_training = voting\_clf.predict(X\_train)

y\_pred\_test = voting\_clf.predict(X\_test)

# Evaluate the performance

print('Classification Report - Training Set:')

print(classification\_report(y\_train, y\_pred\_training))

print('Classification Report - Test Set:')

print(classification\_report(y\_test, y\_pred\_test))

```

此方法通过结合 RandomForest、LogisticRegression 和 SVC 分类器的预测，正确实现了硬投票。每个分类器对类别进行投票，多数票决定最终预测。

### 总结

虽然选项 b 正确地设置了具有指定参数的 `RandomForestClassifier`，但它没有实现问题陈述所要求的硬投票。要实现硬投票，您需要使用上面演示的 `VotingClassifier`。

**Question 3**

Decision Trees:

a. are completely invariant to scaling of the data.

b. tend to overfit and provide poor generalization performance.

c. None of the above.

Select all that apply:

**Answer: a and b.**

Feedback:

From week 4's reading material (Müller, Andreas C and Sarah Guido, Introduction to Machine Learning with Python : A Guide for Data Scientists

(O’Reilly Media, Inc, First edition., 2017)), section 2.3.5. Decision Trees, subsection "Strengths, weaknesses, and parameters":

"As each feature is processed separately, and the possible splits of the data don’t depend on scaling, no preprocessing like normalization or

standardization of features is needed for decision tree algorithms. In particular, decision trees work well when you have features that are on

completely different scales, or a mix of binary and continuous features.

The main downside of decision trees is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.

Therefore, in most applications, the ensemble methods we discuss next are usually used in place of a single decision tree."

**Question 4:**

Select all that apply in the context of building a predictive model:

a. It's crucial for the training set to be a good representation of the new cases we aim to predict for achieving good generalization.

b. Instances that are obvious outliers could potentially distort our model's performance. It might be beneficial to either remove them or rectify the errors manually.

c. None of the above.

Select all that apply:

**Answer: a and b.**

Feedback:

a. In the context of building a predictive model, it is crucial that the training set accurately represents the new cases we aim to predict. This

representation is essential for the model to generalize well, meaning it can apply what it has learned to unseen data effectively. If the training

set is not representative, the model may learn patterns that are specific to the training data but do not hold true in real-world scenarios, leading

to poor performance on new data.

b. Moreover, instances that are obvious outliers can significantly distort the model's performance. Outliers are data points that deviate markedly

from the rest of the dataset. These anomalies can introduce noise and bias into the model, causing it to learn incorrect or irrelevant patterns.

This can result in overfitting, where the model performs well on the training data but poorly on new data.

To address this issue, it might be beneficial to either remove the outliers or manually rectify the errors they represent. Removing outliers can

help in creating a cleaner and more representative dataset, leading to a model that generalizes better. Alternatively, if the outliers contain

valuable information but are erroneous, correcting these errors manually can ensure that the dataset remains comprehensive and accurate,

ultimately enhancing the model’s performance.

**Question 5**

Which of the following statement(s) is/are true about ensemble models? Select all that apply

a. Prior to building each tree in the forest, the scikit-learn implementation of sklearn.ensemble.RandomForestClassifier() uses max\_features

once per tree. Once the features are selected for a specific tree (for example, 80% of the total number of features), those same features are

used to build the entire tree.

b. The learning rate is a hyper-parameter of the sklearn.ensemble.RandomForestClassifier() classifier that must be set up or tuned by the user.

c. The Random Forest has usually better interpretability than the single decision trees.

d. None of the above

**Answer: d**Feedback:

a. In a RandomForestClassifier, the number of features to consider when looking for the best split (often denoted as max\_features) is used each time a split is made in any of the decision trees within the forest.

This means that for each node in each tree:

1. A subset of features (of size max\_features) is randomly selected from the total set of features.

2. The best split is found among these randomly selected features. This process is repeated for every node in every tree, ensuring that each split is based on a different random subset of features.

3. By using max\_features this way, Random Forests introduce randomness at each split, which helps in reducing overfitting and improving generalization.

b. RandomForestClassifier() does not have a hyperparameter called learning rate.

c. From week 5's reading material: "It is basically impossible to interpret tens or hundreds of trees in detail, and trees in random forests tend to

be deeper than decision trees (because of the use of feature subsets). Therefore, if you need to summarize the prediction making in a visual

way to nonexperts, a single decision tree might be a better choice."

Source: Müller, Andreas C and Sarah Guido, Introduction to Machine Learning with Python : A Guide for Data Scientists (O’Reilly Media, Inc, First

edition., 2017)), section 2.3.6. 2.3.6 Ensembles of Decision Trees, Random Forests, subsection "Strengths, weaknesses, and parameters":

Question 6

Check the Python code below:

##################################################

# Docstring:

# data: Bunch

# Dictionary-like object, with the following attributes.

# data{ndarray, dataframe} of shape (569, 30)

# The data matrix. If as\_frame=True, data will be a pandas DataFrame.

# target{ndarray, Series} of shape (569,)

# The classification target. If as\_frame=True, target will be a pandas Series.

#

# feature\_namesndarray of shape (30,)

# The names of the dataset columns.

#

# target\_namesndarray of shape (2,)

# The names of target classes.

#

# frameDataFrame of shape (569, 31)

# Only present when as\_frame=True. DataFrame with data and target.#

#

# For more information: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\_breast\_cancer.html

##################################################

# Import libraries from sklearn:

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

# Load the dataset

data = load\_breast\_cancer()

X, y = data.data, data.target

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a pipeline with scaling and LogisticRegression

pipeline = Pipeline([

('scaler', StandardScaler()),

('logreg', LogisticRegression(penalty='l2', solver='liblinear', random\_state=42))

])

# Set up the parameter grid for GridSearchCV

param\_grid = {

'logreg\_\_C': [0.1, 1.0, 10.0, 100.0]

}

# Create GridSearchCV object

grid\_search = GridSearchCV(pipeline, param\_grid, cv=5, n\_jobs=-1, verbose=2)

# Train the model using GridSearchCV

grid\_search.fit(X\_train, y\_train)

# [\*\*\*\*\*\*\*\* Question 6 \*\*\*\*\*\*]

y\_pred\_training = grid\_search.predict(X\_train)

y\_pred\_test = grid\_search.predict(X\_test)

# Make predictions

#y\_pred\_training = [\*\*\*\*\*\*\*\* Question 6 \*\*\*\*\*\*].predict(X\_train)

#y\_pred\_test = [\*\*\*\*\*\*\*\* Question 6 \*\*\*\*\*\*].predict(X\_test)

# Evaluate the performance

# Print classification reports

print('Classification Report - Training Set:')

print(classification\_report(y\_train, y\_pred\_training))

print('Classification Report - Test Set:')

print(classification\_report(y\_test, y\_pred\_test))

**Answer: b. False**

Feedback:

The statement is partially true but contains a part that is false. Let's clarify it:

When executing grid\_search.fit(X\_train, y\_train), the GridSearchCV object does indeed search for the optimal parameters using cross-validation.

However, it does return a fitted model using the best parameters on the entire training set. There is no need to fit a new model manually after

finding the best parameters.

Here is the corrected version:

"When executing grid\_search.fit(X\_train, y\_train), the GridSearchCV object searches for the optimal parameters using cross-validation and

returns a fitted model using the best parameters on the entire training dataset."

So the original statement is false because it incorrectly states that you need to fit a new model manually after finding the best parameters.

**Question 7**

Select all that apply. An artificial neural network with multiple layers, where all layers (including the output or final layer) have linear activation

functions.

a. Will behave like a standard artificial neural network with typical activation functions such as ReLU or Sigmoid.

b. Will be equivalent to a logistic regression model.

c. Will be functionally equivalent to an artificial neural network with a single linear layer.

d. None of the above.

**Answer: c**

Feedback:

a. False. ReLU and Sigmoid are non linear functions.

b. False. Logistic regression requires a Sigmoid.

c. True. When all layers in a neural network have linear activation functions, the composition of these linear functions is itself a linear function.

Therefore, the network will be equivalent to a single-layer linear model, regardless of the number of layers.

Question 8

The following Keras code will allow the model to perform regression tasks, assuming the data (not shown here) is suitable for the ANN.

import tensorflow as tf

# Initialize the model

tf.random.set\_seed(42)

model = tf.keras.Sequential()

# Add layers to the model

model.add(tf.keras.layers.Dense(64, input\_dim=10, activation='relu')) # First hidden layer with 64 neurons

model.add(tf.keras.layers.Dense(32, activation='relu')) # Second hidden layer with 32 neurons

model.add(tf.keras.layers.Dense(1, activation='linear')) # Output layer with 1 neuron for regression

# Compile the model

model.compile(optimizer='adam', loss='mse', metrics=['mse']) #

# The model is now ready to be trained with model.fit()

True or False?

**Answer: True**

Explanation:

* The given Keras code initializes and sets up a neural network model suitable for regression tasks.
* The input layer expects 10 features (input\_dim=10).
* The network has two hidden layers with 64 and 32 neurons, respectively, both using the ReLU activation function.
* The output layer has one neuron with a linear activation function (activation='linear'), which is appropriate for regression tasks.
* The model is compiled with the Adam optimizer and mean squared error (MSE) as both the loss function and metric, which are standard choices for regression problems.
* Therefore, the code is correctly set up to perform regression tasks, making the statement true.

Question 9 Select all that apply:

Select all that apply. Assume we have 1,000,000 data points in our training set; and a mini-batch size=1,000. In the context of training an artificial neural network model, select the term/s that describe the process in which the model observes all training samples once:

a. One iteration.

b. One forward pass.

c. One forward and backward pass.

d. None of the above.

**Answer: d**

Feedback:

The question is giving the definition for epoch. An epoch is defined as one complete pass through the entire training dataset.

**Question 10**

The following Keras code will correctly compile a model designed for multi-class classification tasks. Assume the data (not shown here) is

suitable for the ANN. For the output, we know that we have one target probability per class for each instance. In this case, one-hot vectors, e.g.,

[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.] to represent class 3.

import tensorflow as tf

# Initialize the model

model = tf.keras.Sequential()

# Add layers to the model

model.add(tf.keras.layers.Dense(128, input\_dim=20, activation='relu')) # First hidden layer with 128 neurons

model.add(tf.keras.layers.Dense(64, activation='relu')) # Second hidden layer with 64 neurons

model.add(tf.keras.layers.Dense(10, activation='softmax')) # Output layer with 10 neurons for multi-class classification

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# The model is now ready to be trained with model.fit()

**Answer: a. True. The code is correct for the given specifications.**

Explanation:

The provided Keras code correctly compiles a model designed for multi-class classification tasks. Here's why:

1. Model Architecture:
   * First Hidden Layer: 128 neurons with ReLU activation function.
   * Second Hidden Layer: 64 neurons with ReLU activation function.
   * Output Layer: 10 neurons with softmax activation function, appropriate for multi-class classification where each output neuron corresponds to a class.
2. Compilation:
   * Optimizer: adam optimizer, a popular choice for training neural networks.
   * Loss Function: categorical\_crossentropy, which is appropriate for multi-class classification tasks with one-hot encoded target vectors.
   * Metrics: accuracy, which is a standard metric for classification tasks.

The use of the softmax activation function in the output layer ensures that the output of the network is a probability distribution over the 10 classes, which is necessary for multi-class classification. The categorical\_crossentropy loss function is designed to handle such probability distributions and compute the loss appropriately.

Therefore, the code correctly compiles a model designed for multi-class classification tasks.