Assignment of Ensemble Techniques (ML 9)

Theoretical

Q1. Can we use Bagging for regression problems?

Ans-> Yes, bagging can be effectively used for regression problems. Bagging (Bootstrap Aggregating) is an ensemble learning technique that can be applied to both classification and regression tasks. In regression, bagging involves training multiple regression models on different bootstrap samples of the training data and then averaging their predictions to produce a final prediction. This process helps to reduce variance and improve the overall accuracy and stability of the model.

Q2. What is the difference between multiple model training and single model training?

Ans-> Single model training uses one machine learning model to learn from data and make predictions, while multiple model training utilizes several models, often in an ensemble, to improve accuracy or handle diverse data patterns. Single model training is simpler to implement and interpret, while multiple model training can be more robust and adaptable to complex scenarios.

- Here's a more detailed breakdown:
- Single Model Training:
- Concept: A single machine learning model is trained on the entire dataset, learning a single set of parameters to represent the relationships within the data.
- Advantages:
- Simpler Implementation: Easier to set up, train, and deploy. Easier Interpretation: Understanding how the model makes predictions is generally more straightforward.
- Faster Training (Potentially): Training a single model can be quicker than training multiple models, especially with large datasets.
- Disadvantages:
- Limited Generalization: May struggle to generalize well to diverse or complex datasets with varying patterns.
- Potential Overfitting: Can be prone to overfitting, especially with complex models and limited data.
- Single Point of Failure: If the single model fails, the entire system can be impacted.
- Multiple Model Training:
- Concept: Multiple models are trained, either independently or collaboratively, to solve a problem. This can involve ensemble methods (like bagging or boosting) where individual model predictions are combined, or using separate models for different subsets or aspects of the data.
- Advantages:
- Improved Accuracy and Robustness: Ensembles of models can often achieve higher accuracy and better generalization than a single model.
- Handles Diverse Data: Can be more effective at capturing complex relationships and diverse patterns in the data.
- Fault Tolerance: If one model in an ensemble fails, others can still contribute to the prediction.
- Disadvantages: Increased Complexity: More complex to implement and manage than single model training.
- Higher Computational Cost: Training multiple models can be more resource-intensive.
- Potentially Slower Inference: Combining predictions from multiple models can take longer than using a single model.

Q3. Explain the concept of feature randomness in Random Forest

Ans-> In Random Forest, feature randomness, also known as feature bagging, refers to the process of selecting a random subset of features at each node of a decision tree during the tree construction process. This contrasts with traditional decision trees where all features are considered for splitting at each node. By randomly choosing a subset of features, Random Forest introduces diversity among the individual trees, making the overall ensemble more robust and less prone to overfitting.

Q4. What is OOB (Out of Bag) Score?

Ans-> The Out-of-Bag (OOB) score is a performance metric used in ensemble learning methods like Random Forests to estimate the generalization error of a model. It leverages the fact that, during the bootstrapping process in bagging, not all training data is used to train each individual tree. These unused samples, called out-of-bag samples, are then used to evaluate the performance of each tree, providing an unbiased estimate of how well the model will perform on unseen data.

Q5. How can you measure the importance of features in Random Forest Model?

Ans-> Feature importance in a Random Forest model can be measured using Mean Decrease in Impurity (MDI), also known as Gini importance, or by Permutation Feature Importance. MDI quantifies how much each feature reduces impurity (like Gini impurity) when used to split nodes in decision trees within the forest, according to GeeksforGeeks. Permutation importance, on the other hand, assesses the impact of shuffling a feature's values on the model's performance.

1. Mean Decrease in Impurity (MDI) / Gini Importance:

- How it works: For each feature, the algorithm calculates the average reduction in impurity (e.g., Gini impurity) when that feature is used to split nodes across all trees in the forest.
- Interpretation: Features that consistently lead to larger reductions in impurity are considered more important.
- Implementation: In scikit-learn, the feature_importances_ attribute of a trained RandomForestClassifier or RandomForestRegressor provides these importance scores.
- Example: If feature "A" is used in many splits and significantly reduces impurity in each case, its Gini importance score will be higher compared to a feature "B" that is used less frequently and has a smaller impact on impurity reduction.

2. Permutation Feature Importance:
    - How it works: For each feature, the algorithm randomly shuffles its values while keeping other features' values intact. Then, the model's performance (e.g., accuracy, F1-score) is evaluated on this modified dataset. Interpretation: A larger decrease in performance (e.g., lower accuracy) after shuffling a feature indicates that the feature is more important.
    - Example: If shuffling feature "X" significantly reduces the model's accuracy, it suggests that "X" is crucial for making accurate predictions.
    - Implementation: You can manually implement permutation importance by shuffling features and evaluating model performance. Libraries like sklearn.inspection.permutation_importance can also be used.
    - Advantages: This method is model-agnostic, meaning it can be applied to any model, not just Random Forests.
    - In essence, both methods offer valuable insights into feature importance:
    - MDI (Gini Importance): Useful for understanding how features contribute to the internal workings of the Random Forest model. Permutation Importance:
    - A more robust approach that assesses the overall impact of features on the model's predictive power.

Q6. Explain the working principle of a Bagging Classifier.

Ans-> **Bagging** stands for **Bootstrap Aggregating**. It's an ensemble technique that improves model stability and accuracy by reducing variance (which helps combat overfitting).

Here's a step-by-step:

1. **Bootstrap Sampling**:
    - From the original training dataset, multiple subsets are created by *random sampling with replacement*.
    - So, some observations may be repeated, and some might be left out in any given subset.
2. **Base Learner Training**:
    - A **base classifier** (e.g. a decision tree) is trained independently on each of these bootstrap samples.
    - All models are typically of the same type but trained on different data.
3. **Prediction Aggregation**:
    - For **classification**, each base model votes for a class label.
    - The final prediction is made using **majority voting** (i.e., the most common prediction wins).
4. **Result**:
    - Averaging out the "opinion" of all classifiers smooths the decision boundary.
    - This reduces the chance of overfitting to noise in the training data.

Q7. How do you evaluate a Bagging Classifier's performance?

Ans-> A Bagging Classifier's performance can be evaluated using various metrics like accuracy, precision, recall, F1-score, and AUC (Area Under the ROC Curve), depending on the specific task and data characteristics. Additionally, analyzing the individual base estimators (e.g., decision trees) within the Bagging ensemble can provide insights into the model's behavior.

- Detailed Explanation:
    1. Accuracy: This is a common metric that measures the overall correctness of the classifier's predictions. It's calculated as the ratio of correctly predicted instances to the total number of instances.
    2. Precision, Recall, and F1-score: These metrics are particularly useful when dealing with imbalanced datasets, where one class has significantly more instances than others. Precision: Measures the proportion of correctly predicted positive instances out of all instances predicted as positive. Recall: Measures the proportion of correctly predicted positive instances out of all actual positive instances. F1-score: A harmonic mean of precision and recall, providing a balanced measure of the classifier's performance.
    3. AUC (Area Under the ROC Curve): The ROC curve plots the true positive rate against the false positive rate at various threshold settings. AUC quantifies the classifier's ability to distinguish between different classes.
    4. Analyzing Base Estimators: Examining the individual estimators (e.g., decision trees) in the Bagging ensemble can provide further insights: Feature importance: Analyzing which features are most used by the base estimators can reveal important insights into the data and the model's decision-making process. Out-of-bag (OOB) error: If the bagging classifier is built with replacement (which is common), the OOB error can be used to estimate the generalization error of the model without needing a separate validation set.

5. Cross-validation: Techniques like k-fold cross-validation can be used to get a more robust estimate of the model's performance by training and evaluating it on different subsets of the data.

Q8. How does a Bagging Regressor work?

Ans-> **Bagging Classifier**, shifting gears to the **Bagging Regressor** should feel pretty intuitive—it uses the same core logic but is tailored for **regression tasks**. Let's break it down:

🔄 Working Principle of a Bagging Regressor

1. **Bootstrap Sampling**:
   - Just like in classification, it creates several training subsets by randomly sampling (with replacement) from the original dataset.
   - Each subset might have duplicates and may leave out some original samples.
2. **Train Base Regressors**:
   - A **base regressor** (commonly a decision tree regressor) is trained on each bootstrap sample independently.
3. **Aggregate Predictions**:
   - Instead of voting (as in classification), predictions from all regressors are **averaged** to get the final output.
     - Example: If 5 models predict 4.1, 4.3, 3.9, 4.0, and 4.2 → final prediction = mean = **4.1**

✨ Why It Works Well

- **Reduces Variance**: Averaging helps smooth out noisy predictions from individual overfit regressors.
- **Enhances Stability**: It makes the model more robust, especially with algorithms prone to high variance (like decision trees).
- **Maintains Interpretability**: Each model can still be interpretable on its own (e.g., individual trees in the ensemble).

Q9. What is the main advantage of ensemble techniques?

Ans-> The primary advantage of ensemble techniques is improved predictive performance and generalization by leveraging the strengths of multiple models and reducing overfitting. Ensemble methods combine the predictions of several models to produce a more accurate and reliable prediction than any single model could achieve alone.

- Here's a more detailed breakdown:
- Enhanced Accuracy: By combining the predictions of multiple models, ensemble techniques often achieve higher accuracy than individual models.
- Reduced Overfitting: Ensemble methods can help mitigate overfitting, where a model performs well on training data but poorly on unseen data.
- Increased Robustness: Ensemble models are less susceptible to noise and outliers in the data because the predictions are averaged or aggregated from multiple models.
- Improved Generalization: By combining different models, ensembles can generalize better to unseen data, leading to more reliable predictions in real-world scenarios.
- Flexibility: Ensemble techniques can be applied to various machine learning tasks and combined with different model types, offering flexibility in model selection.
- Scalability: Many ensemble methods, like XGBoost and LightGBM, are designed to handle large datasets and complex tasks efficiently.

Q10. What is the main challenge of ensemble methods?

Ans-> The primary challenge of ensemble methods is increased computational cost and complexity due to the need for training and managing multiple models. Additionally, ensembles can be difficult to interpret and may be prone to overfitting if not carefully managed.

- Here's a more detailed breakdown:
- Computational Cost: Ensemble methods, by their nature, involve training multiple base models. This can significantly increase the time and resources needed for both training and prediction.
- Complexity: Combining multiple models introduces complexity in terms of understanding how each model contributes to the final prediction and how to best tune and optimize the ensemble.
- Interpretability: Ensemble models can act like "black boxes," making it difficult to understand the reasoning behind individual predictions. This lack of transparency can be a concern in applications where explainability is crucial.
- Overfitting: If the base models are too similar or the ensemble method is not appropriately chosen, ensembles can still overfit the training data, leading to poor generalization on unseen data.
- Data and Model Quality: Ensemble methods are sensitive to the quality and diversity of the data and the individual models within the ensemble. Ensuring high-quality data and diverse base models is crucial for effective ensemble performance.
- Finding Optimal Configurations: Determining the best hyperparameters for each individual model within the ensemble can be a complex task, requiring significant tuning and potentially leading to exponential increases in the search space.

Q11. Explain the key idea behind ensemble techniques

Ans-> The core idea behind ensemble techniques in machine learning is to combine the predictions of multiple individual models (often called "base learners") to create a more robust and accurate prediction system. Instead of relying on a single model, which might be prone to errors or overfitting, an ensemble leverages the strengths of various models to reduce overall error and improve generalization performance on unseen data.

- Here's a more detailed explanation:
    1. Combining Strengths, Reducing Weaknesses: Individual models can have biases or weaknesses in certain areas of the data. For example, one model might be good at predicting one type of data point, while another model excels at a different type. By combining these diverse models, the ensemble can compensate for individual model limitations and make more accurate predictions across a wider range of data.
    2. Addressing Overfitting: Ensemble methods can help reduce overfitting, which is when a model learns the training data too well and performs poorly on new, unseen data. Combining multiple models, especially when they are trained on different subsets of the data or with slightly different parameters, can help to generalize better and avoid overfitting.
    3. Types of Ensemble Techniques:

- Bagging: Creates multiple models by training them on different random subsets of the training data (with replacement). Random Forest is a popular example.
- Boosting: Sequentially trains models, with each model learning from the mistakes of its predecessors. Boosting algorithms like AdaBoost and Gradient Boosting are commonly used.
- Stacking: Uses the predictions of multiple base models as input to a meta-model, which then learns to combine those predictions.

    4. Why Ensemble Methods are Powerful:

- Improved Accuracy: By combining diverse models, ensembles can often achieve higher accuracy than any single model alone.
- Increased Robustness: Ensembles are less sensitive to variations in the training data and are more stable in their predictions.
- Better Generalization: Ensembles can generalize better to unseen data, reducing the risk of overfitting.
- Reduced Variance: Ensembles can help reduce the variance of the predictions, leading to more stable and reliable results.

Q12. What is a Random Forest Classifier?

Ans-> A random forest classifier is a machine learning algorithm that uses an ensemble of decision trees to classify data. It works by building multiple decision trees, each trained on a different random subset of the data and features, and then combining their predictions to make a final classification. This approach helps to reduce overfitting and improve accuracy compared to using a single decision tree.

Q13. What are the main types of ensemble techniques?

Ans-> Ensemble methods in machine learning combine multiple individual models to create a more powerful and accurate predictive model. The main types of ensemble methods are bagging, boosting, and stacking. These methods differ in how they combine individual models and the types of models they use. Here's a breakdown of the key ensemble methods:

1. Bagging (Bootstrap Aggregating):
    - How it works: Bagging trains multiple models independently on different random subsets of the training data (created by bootstrapping, which involves sampling with replacement).
    - Example: Random Forests, where multiple decision trees are trained on different data subsets.
    - Goal: Reduce variance and improve model stability by averaging or taking a majority vote of the individual models' predictions.
2. Boosting:
    - How it works: Boosting trains models sequentially, with each model learning from the mistakes of its predecessors.
    - Example: AdaBoost, Gradient Boosting, XGBoost.
    - Goal: Reduce bias and improve model accuracy by focusing on difficult-to-classify instances.
3. Stacking (Stacked Generalization):
    - How it works: Stacking combines the predictions of multiple diverse models (base learners) using a meta-learner (or meta-model).
    - Example: Using the predictions of multiple regression models as input for a final linear regression model.
    - Goal: Improve prediction accuracy by leveraging the strengths of different models and their interactions.
    - In addition to these main types, other ensemble methods include:
    - Voting: Simple averaging or majority voting of predictions from multiple models.
    - Blending: Similar to stacking but uses a separate validation set to train the meta-learner.
    - Cascading: Similar to boosting, but the models are arranged in a cascade, where each model has a specific role.

Q14. What is ensemble learning in Machine Learning?

Ans-> Ensemble learning in machine learning combines the predictions of multiple individual models (often called "weak learners") to produce a more accurate and reliable prediction than any single model could achieve. This approach leverages the diversity of different models to mitigate their individual weaknesses and improve overall performance.

Q15. When should we avoid using ensemble methods?

Ans-> Ensemble methods, while powerful, may not always be the best choice. Avoid them when computational resources are limited, interpretability is crucial, or when the base models are highly correlated or the dataset is small and lacks diversity.

- Here's a more detailed breakdown:

  1. Limited Computational Resources: Ensemble methods, by definition, involve training and combining multiple models. This can be significantly more computationally expensive than using a single model. If you're dealing with limited computing power or time constraints, a simpler, single model might be preferable.
  2. Interpretability is Paramount: Ensemble models, especially complex ones like Random Forests or stacked models, can be difficult to interpret. If you need to understand why a model is making a particular prediction, and how each individual model contributes, a simpler, single model might be more appropriate. For example, in healthcare or finance, understanding the reasoning behind a decision is often crucial.
  3. Highly Correlated Models or Insufficient Data: Ensemble methods thrive when the individual models are diverse and make different types of errors. If your base models are highly correlated (meaning they tend to make similar mistakes), the ensemble might not provide much benefit over a single model. Similarly, if your dataset is small and lacks diversity, the ensemble might overfit to the training data or not generalize well.
  4. Simple Problems: For very simple problems, where a single, well-tuned model can achieve high accuracy, the complexity and computational cost of an ensemble might not be justified.
  5. Tight Latency Requirements: If your application requires very fast predictions (e.g., real-time processing), the added time to combine predictions from multiple models in an ensemble might be detrimental.

Q16. How does Bagging help in reducing overfitting?

Ans-> Bagging helps reduce overfitting by averaging out the variance in model predictions. It achieves this by training multiple models on different subsets of the data, which helps to capture different aspects of the underlying data distribution and create a more robust model. By averaging the predictions of these diverse models, bagging minimizes the impact of noisy or outlier data points, leading to a more generalized model that performs better on unseen data.

- Here's a more detailed explanation:
- Reduces Variance: Bagging works by creating multiple models, each trained on a slightly different subset of the data (obtained through bootstrapping). This introduces diversity among the models and reduces the variance of the overall ensemble prediction.
- Averaging Predictions: When a new data point is encountered, each model in the ensemble makes a prediction, and these predictions are then combined (typically by averaging for regression or voting for classification).
- Handles Outliers and Noise: By averaging predictions, bagging helps to smooth out the impact of individual data points, including outliers or noisy data. This makes the overall prediction less sensitive to individual data points and leads to more stable predictions.
- Improved Generalization: The diversity among the models in the ensemble, combined with the averaging of their predictions, results in a model that is less likely to overfit the training data and therefore generalizes better to new, unseen data.

Q17. Why is Random Forest better than a Single Decision Tree?

Ans-> Random Forest is generally a better choice than a single decision tree because it reduces overfitting, handles noisy data better, and provides more robust and accurate predictions. While individual decision trees are simpler to understand, they are prone to overfitting the training data and can be unstable with noisy or missing data.

- Here's a more detailed breakdown:

  1. Reduced Overfitting:

  - Single Decision Trees: Can easily overfit the training data, meaning they learn the training set too well and perform poorly on new, unseen data.
  - Random Forest: Uses an ensemble of multiple trees, each trained on a random subset of the data and features. This averaging effect helps to reduce overfitting and improve generalization to new data.

  2. Robustness to Noisy Data:

  - Single Decision Trees: Sensitive to noisy data (outliers, irrelevant features) and can create complex branches to fit the noise, leading to poor generalization.
  - Random Forest: The ensemble approach helps to average out the impact of noisy data points, making the model more robust to outliers and irrelevant features.

  3. Improved Accuracy:

  - Single Decision Trees: Due to overfitting and sensitivity to noise, they often have lower accuracy on unseen data.
  - Random Forest: The combination of multiple trees reduces variance and bias, leading to higher overall accuracy, especially on complex datasets.

  4. Feature Importance:

- Random Forest: Provides a measure of feature importance, indicating which features are most influential in making predictions. This can be helpful for understanding the data and selecting important features for other models.

5. Handling Large Datasets:

- Random Forest: Can handle large datasets with many features and observations more effectively than a single decision tree.

Q18. What is the role of bootstrap sampling in Bagging?

Ans-> In Bagging (Bootstrap Aggregating), bootstrap sampling plays a crucial role in creating diverse training datasets for individual models. It involves randomly sampling with replacement from the original dataset to generate multiple subsets, which are then used to train separate models. This process introduces variability among the models, leading to a more robust and accurate final prediction.

- Bagging in Machine Learning
- Here's a more detailed breakdown:
- Creating Diversity: Bagging relies on the principle of creating different training datasets for each base model. Bootstrapping achieves this by randomly selecting data points from the original dataset with replacement. This means some data points might appear multiple times in a sample, while others might be left out entirely.
- Reduces Variance: By training models on these diverse subsets, bagging reduces the variance of the overall model. Individual models might overfit to specific parts of the data, but when their predictions are aggregated, the errors tend to cancel out, leading to a more stable prediction.
- Parallel Training: The bootstrap samples can be processed independently and in parallel, making the bagging process efficient.
- Aggregation: Finally, the predictions of all the individual models are aggregated, typically through averaging for regression tasks or majority voting for classification tasks, to produce the final prediction.

Q19. What are some real-world applications of ensemble techniques?

Ans-> Ensemble techniques, which combine multiple machine learning models to improve accuracy and robustness, have broad real-world applications. These techniques are used in various industries to enhance predictions and decision-making processes. Here are some examples:

1. Healthcare:

- Disease Diagnosis: Ensemble methods, like random forests, are used to diagnose diseases by combining predictions from multiple models trained on different features such as patient history, test results, and medical images according to SoluLab. This can lead to more accurate diagnoses and better patient outcomes.
- Personalized Treatment: Ensemble models can help tailor treatments by combining data from various sources, like genetics, medical imaging, and electronic health records.
- Risk Assessment: Ensemble methods are applied to patient risk assessment, predicting the likelihood of certain health events or complications.

2. Finance:

- Credit Scoring: Ensemble models are used to assess the creditworthiness of individuals by combining predictions from multiple algorithms.
- Fraud Detection: They help identify fraudulent transactions by analyzing patterns across multiple models, reducing false positives and negatives.
- Stock Market Prediction: Ensemble methods are employed to forecast stock prices by aggregating predictions from various models.

3. E-commerce and Marketing:

- Customer Segmentation: Ensemble models are used to segment customers based on their behavior and preferences, allowing for more effective targeting and personalized recommendations.
- Churn Prediction: They help predict customer churn (when customers stop using a service) by analyzing their interactions and behavior.
- Recommendation Systems: Ensemble methods are used to improve the accuracy of product recommendations by combining predictions from different models.

4. Other Applications:

- Remote Sensing and Land Cover Mapping: Ensemble methods are used to analyze satellite imagery and classify land cover types.
- Computer Security: They are used in intrusion detection, malware detection, and other security applications.
- Weather Forecasting: Ensemble methods are employed to improve the accuracy of weather predictions by combining multiple forecasting models.
- Autonomous Driving: Ensemble methods are used to improve the reliability and safety of self-driving cars.
- Spam Detection: They are used to classify emails as spam or not spam, by combining predictions from different spam filters.

Q20. What is the difference between Bagging and Boosting?

Ans-> Bagging and boosting are both ensemble learning methods that combine multiple machine learning models to improve performance. The key difference lies in how they build these ensembles: bagging trains models independently and in parallel, while boosting trains models sequentially, with each model learning from the errors of its predecessors.

- Here's a more detailed breakdown:
- Bagging (Bootstrap Aggregating):
    - Parallel Training: Bagging creates multiple subsets of the training data by sampling with replacement (bootstrapping). Each subset is used to train a separate model independently.
    - Variance Reduction: By averaging the predictions of these independent models, bagging reduces variance and helps prevent overfitting.
    - Example: Random Forest is a popular bagging algorithm that uses decision trees as base learners.
- Boosting:
    - Sequential Training: Boosting trains models sequentially, with each new model focusing on correcting the errors made by the previous ones. The models are not independent; they build upon each other.
    - Bias Reduction: Boosting aims to reduce bias by focusing on the difficult-to-classify instances. It gives more weight to misclassified samples, so subsequent models pay more attention to them.
    - Example: Gradient Boosting Machines (GBM), XGBoost, and AdaBoost are popular boosting algorithms.

Practical

Q21. Train a Bagging Classifier using Decision Trees on a sample dataset and print model accuracy.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = BaggingClassifier(DecisionTreeClassifier(), n_estimators=50, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Bagging Classifier Accuracy:", accuracy_score(y_test, y_pred))
```

```
Bagging Classifier Accuracy: 0.956140350877193
```

Q22. Train a Bagging Regressor using Decision Trees and evaluate using Mean Squared Error (MSE)

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = BaggingRegressor(DecisionTreeRegressor(), n_estimators=50, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Bagging Regressor MSE:", mean_squared_error(y_test, y_pred))
```

```
Bagging Regressor MSE: 3056.494602247191
```

Q23. Train a Forest Classfier on the Breast Cancer dataset and print feature importance scores

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import pandas as pd

data = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
```

```
model.fit(X_train, y_train)

feature_importances = pd.Series(model.feature_importances_, index=data.feature_names)
print("Top 5 Features:\n", feature_importances.sort_values(ascending=False).head())
```

```
Top 5 Features:
 worst area              0.153892
worst concave points    0.144663
mean concave points     0.106210
worst radius            0.077987
mean concavity          0.068001
dtype: float64
```

Q24. Train a Random Forest Regressor and compare its performance with a single Decision Tree

```
from sklearn.datasets import load_diabetes
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

dt = DecisionTreeRegressor(random_state=42)
rf = RandomForestRegressor(random_state=42)

dt.fit(X_train, y_train)
rf.fit(X_train, y_train)

print("Decision Tree MSE:", mean_squared_error(y_test, dt.predict(X_test)))
print("Random Forest MSE:", mean_squared_error(y_test, rf.predict(X_test)))
```

```
Decision Tree MSE: 4976.797752808989
Random Forest MSE: 2952.0105887640448
```

Q25. Compute the Out-of-Bag (OOB) Score for a Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

X, y = load_breast_cancer(return_X_y=True)
X_train, _, y_train, _ = train_test_split(X, y, test_size=0.2, random_state=42)

rf = RandomForestClassifier(oob_score=True, random_state=42)
rf.fit(X_train, y_train)

print("Out-of-Bag (OOB) Score:", rf.oob_score_)
```

```
Out-of-Bag (OOB) Score: 0.9560439560439561
```

Q26. Train a Bagging Classifier using SVM as a base estimator and print accuracy

```
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = BaggingClassifier(estimator=SVC(), n_estimators=10, random_state=42)
model.fit(X_train, y_train)

print("Bagging with SVM Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

```
Bagging with SVM Accuracy: 0.9473684210526315
```

Q27. Train a Random Forest Classifier with different numbers of trees and compare accuracy

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

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for n in [10, 50, 100]:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    print(f"Random Forest (n={n}) Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

```
Random Forest (n=10) Accuracy: 0.956140350877193
Random Forest (n=50) Accuracy: 0.9649122807017544
Random Forest (n=100) Accuracy: 0.9649122807017544
```

Q28. Train a Bagging Classifier using Logistic Regression as a base estimator and print AUC score

```
# prompt: Train a Bagging Classifier using Logistic Regression as a base estimator and print AUC score

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = BaggingClassifier(estimator=LogisticRegression(solver='liblinear'), n_estimators=10, random_state=42)
model.fit(X_train, y_train)

# Get probability predictions for AUC
y_pred_proba = model.predict_proba(X_test)[:, 1] # Probability of the positive class

print("Bagging with Logistic Regression AUC:", roc_auc_score(y_test, y_pred_proba))
```

```
Bagging with Logistic Regression AUC: 0.99737962659679
```

Q29. Train a Random Forest Regressor and analyze feature importance scores

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

X, y = load_diabetes(return_X_y=True)
X_train, _, y_train, _ = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

print("Feature Importances:", model.feature_importances_)
```

```
Feature Importances: [0.05864167 0.00963304 0.35546898 0.08840759 0.05278353 0.05722749
 0.05133862 0.02421276 0.23095698 0.07132935]
```

Q30. Train an ensemble model using both Bagging and Random Forest and compare accuracy

```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

bagging = BaggingClassifier(DecisionTreeClassifier(), random_state=42)
random_forest = RandomForestClassifier(random_state=42)

for model, name in zip([bagging, random_forest], ["Bagging", "Random Forest"]):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name} Accuracy:", accuracy_score(y_test, y_pred))
```

```
Bagging Accuracy: 0.956140350877193
Random Forest Accuracy: 0.9649122807017544
```

Q31. Train a Random Forest Classifier and tune hyperparameters using GridSearchCV

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV

X, y = load_breast_cancer(return_X_y=True)
X_train, _, y_train, _ = train_test_split(X, y, test_size=0.2, random_state=42)

params = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5, None],
}

gs = GridSearchCV(RandomForestClassifier(random_state=42), params, cv=3)
gs.fit(X_train, y_train)

print("Best Parameters:", gs.best_params_)
print("Best Score:", gs.best_score_)
```

```
Best Parameters: {'max_depth': None, 'n_estimators': 50}
Best Score: 0.956009643313582
```

Q32. Train a Bagging Regressor with different numbers of base estimators and compare performance

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for n in [5, 10, 20]:
    model = BaggingRegressor(estimator=DecisionTreeRegressor(), n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"n_estimators={n} → MSE: {mean_squared_error(y_test, y_pred)}")
```

```
n_estimators=5 → MSE: 3367.1020224719105
n_estimators=10 → MSE: 3256.961797752809
n_estimators=20 → MSE: 3202.4416853932585
```

Q33. Train a Random Forest Classifier and analyze misclassified samples

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import numpy as np

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

misclassified_indices = np.where(y_pred != y_test)[0]
print("Misclassified sample indices:", misclassified_indices)
print("Number of misclassified samples:", len(misclassified_indices))
```

```
Misclassified sample indices: [ 8 20 77 82]
Number of misclassified samples: 4
```

Q34. Train a Bagging Classifier and compare its performance with a single Decision Tree classifier

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

single_tree = DecisionTreeClassifier(random_state=42)
bagging_model = BaggingClassifier(estimator=DecisionTreeClassifier(), random_state=42)

single_tree.fit(X_train, y_train)
bagging_model.fit(X_train, y_train)

print("Single Decision Tree Accuracy:", accuracy_score(y_test, single_tree.predict(X_test)))
print("Bagging Classifier Accuracy:", accuracy_score(y_test, bagging_model.predict(X_test)))
```

```
Single Decision Tree Accuracy: 0.9473684210526315
Bagging Classifier Accuracy: 0.956140350877193
```

Q35. Train a Random Forest Classifier and Visualize the confusion matrix

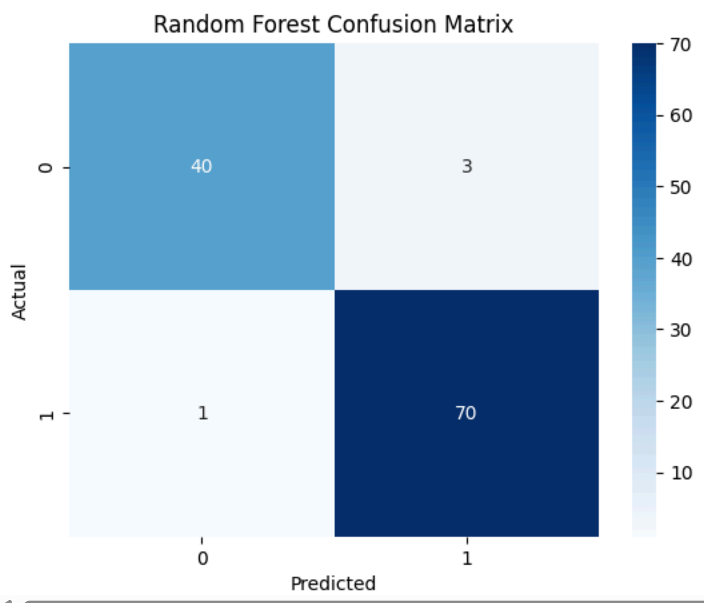
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Q36. Train a Stacking classifier using Decision Trees, SVM and Logistic Regression, and compare accuracy

```
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
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 accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

estimators = [
    ('dt', DecisionTreeClassifier()),
    ('svm', SVC(probability=True)),
    ('lr', LogisticRegression())
]

model = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
model.fit(X_train, y_train)

print("Stacking Classifier Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

```
Stacking Classifier Accuracy: 0.9649122807017544
```

Q37. Train a Random Forest Classifier and print the top 5 most important features

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import numpy as np

data = load_breast_cancer()
X_train, _, y_train, _ = train_test_split(data.data, data.target, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

importances = model.feature_importances_
top_indices = np.argsort(importances)[-5:]

print("Top 5 Features:")
for i in top_indices[::-1]:
    print(f"{data.feature_names[i]}: {importances[i]:.4f}")
```

```
Top 5 Features:
worst area: 0.1539
worst concave points: 0.1447
mean concave points: 0.1062
worst radius: 0.0780
mean concavity: 0.0680
```

Q38. Train a Bagging Classifier and evaluate performance using Precision, Recall, and F1-score

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = BaggingClassifier(DecisionTreeClassifier(), random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Precision: 0.9583333333333334
Recall: 0.971830985915493
F1 Score: 0.965034965034965
```

Q39. Train a Random Forest Classifier and analyze the effect of max_depth on accuracy

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

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for depth in [3, 5, None]:
    model = RandomForestClassifier(max_depth=depth, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"Max Depth = {depth}: Accuracy = {accuracy_score(y_test, y_pred):.4f}")
```

```
Max Depth = 3: Accuracy = 0.9649
Max Depth = 5: Accuracy = 0.9649
Max Depth = None: Accuracy = 0.9649
```

Q40. Train a Bagging Regressor using different base estimators (Decision Tree and KNeighbours) and compare performance

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for estimator in [DecisionTreeRegressor(), KNeighborsRegressor()]:
    model = BaggingRegressor(estimator=estimator, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{estimator.__class__.__name__} → MSE: {mean_squared_error(y_test, y_pred):.4f}")
```

```
DecisionTreeRegressor → MSE: 3256.9618
KNeighborsRegressor → MSE: 2990.6536
```

Q41. Train a Random Forest Classifier and evaluate its performance using ROC-AUC Score

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Get probability predictions for AUC
y_pred_proba = model.predict_proba(X_test)[:, 1] # Probability of the positive class

print("Random Forest Classifier ROC-AUC Score:", roc_auc_score(y_test, y_pred_proba))
```

```
Random Forest Classifier ROC-AUC Score: 0.9952505732066819
```

Q42. Train a Bagging Classifier and evaluate its performance using cross-validation

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score

X, y = load_breast_cancer(return_X_y=True)
model = BaggingClassifier(estimator=DecisionTreeClassifier(), random_state=42)

scores = cross_val_score(model, X, y, cv=5)
print("Cross-Validation Scores:", scores)
print("Mean Accuracy:", scores.mean())
```

```
Cross-Validation Scores: [0.92982456 0.93859649 0.97368421 0.93859649 0.97345133]
Mean Accuracy: 0.9508306163639186
```

Q43. Train a Random Forest Classifier and plot the Precision-Recall curve

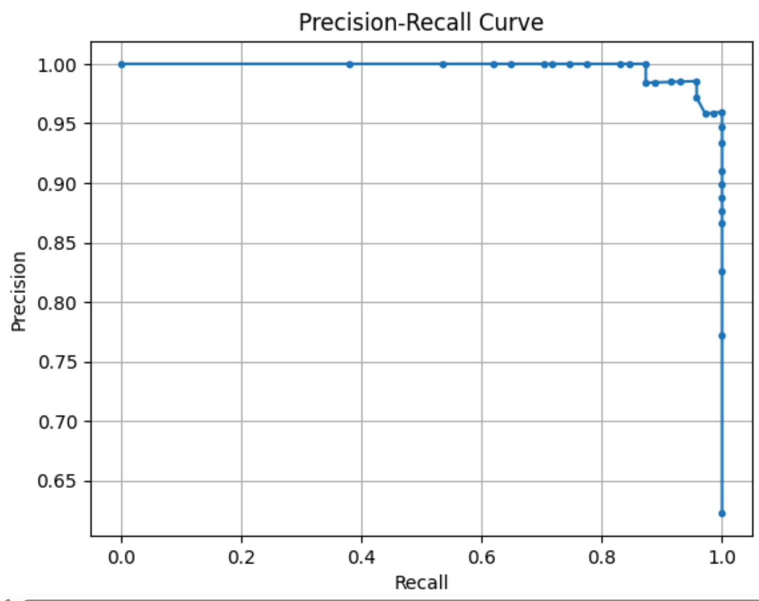
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_proba = model.predict_proba(X_test)[:, 1]

precision, recall, _ = precision_recall_curve(y_test, y_proba)

plt.plot(recall, precision, marker='.')
plt.title("Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
plt.show()
```



Q44. Train a Stacking Classifier with Random Forest and Logistic Regression and compare accuracy

```
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

estimators = [
    ('rf', RandomForestClassifier()),
    ('lr_base', LogisticRegression())
]

stack = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
stack.fit(X_train, y_train)

print("Stacking Classifier Accuracy:", accuracy_score(y_test, stack.predict(X_test)))
```

```
Stacking Classifier Accuracy: 0.9736842105263158
```

Q45. Train a Bagging Regressor with different levels of bootstrap samples and compare performance

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for sample_fraction in [0.5, 0.75, 1.0]:
    model = BaggingRegressor(estimator=DecisionTreeRegressor(), max_samples=sample_fraction, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"Bootstrap Fraction = {sample_fraction:.2f} → MSE: {mean_squared_error(y_test, y_pred):.4f}")
```

```
Bootstrap Fraction = 0.50 → MSE: 3235.3548
Bootstrap Fraction = 0.75 → MSE: 3230.5035
Bootstrap Fraction = 1.00 → MSE: 3256.9618
```