Theoretical

In [ ]:
```
#Can we use Bagging for regression problems
'''
Yes, Bagging (Bootstrap Aggregating) can be used for regression problems, and it is
and improving model stability. It works by creating multiple subsets of the trainin
individual regression models (such as Decision Trees or Linear Regression) on these
to produce the final output. This averaging helps smooth predictions and mitigates
for high-variance models like Decision Trees. A common implementation is `BaggingRe
regressors work together to provide more stable and accurate predictions. Bagging i
datasets or when models are sensitive to noise, as it helps in creating a more robu
sequentially corrects errors, Bagging runs models in parallel and focuses on varian
'''
# What is the difference between multiple model training and single model training
'''
Single model training involves using a single algorithm to learn patterns from the
is simpler, requires fewer computational resources, and is easier to interpret, but
complex or underfitting if too simple. In contrast, multiple model training, often
Bagging, Boosting, and Stacking, involves training several models and combining the
stability, and generalization. This approach reduces variance and bias, making it m
it requires more computational power and can be harder to interpret compared to a s
efficient and straightforward, multiple model training is generally more effective
on complex datasets.
'''
# Explain the concept of feature randomness in Random Forest
'''
Feature randomness in Random Forest refers to the process of selecting a random sub
a decision tree, rather than considering all features. This technique helps introdu
reducing correlation between them and improving the overall model's generalization
node considers all available features to find the best split, which can lead to ove
However, in Random Forest, when constructing each tree, a random subset of features
different trees learn different patterns from the data. This randomness helps reduc
and enhances predictive accuracy. Typically, for classification problems, the numbe
√(total features), while for regression problems, it is (total features) / 3. This
accuracy, making Random Forest a powerful ensemble learning method.
'''
#What is OOB (Out-of-Bag) Score
'''
The Out-of-Bag (OOB) Score is a built-in validation technique used in Random Forest
needing a separate validation set. It works by leveraging the bootstrap sampling me
forest. In Random Forest, each tree is trained on a random subset of the dataset, c
approximately 63% of the data is used for training. The remaining 37% of the data,
particular tree, is called Out-of-Bag (OOB) data. The model uses this OOB data to e
and calculating an error metric (e.g., accuracy for classification or mean squared
provides an unbiased estimate of the model's generalization ability without requiri
when working with limited data. It also helps detect overfitting since it measures
'''
#How can you measure the importance of features in a Random Forest model
'''
Feature importance in a Random Forest model can be measured using Mean Decrease in
MDI, also known as Gini Importance, evaluates feature importance based on how much
impurity for classification or variance for regression) across all decision trees i
to impurity reduction are considered more important, and this information is availa
attribute. However, MDI tends to favor features with more unique values. In contras
```

```
importance by randomly shuffling each feature and observing the drop in model accur
indicates that the feature is highly important. This method provides a more reliabl
performance but is computationally expensive. Using both approaches together helps
most to the model's predictions.
'''
#Explain the working principle of a Bagging Classifier
'''
A Bagging Classifier works on the principle of Bootstrap Aggregating (Bagging) to i
The process involves training multiple instances of a base classifier on different
combining their predictions to make the final decision.

Working Principle of Bagging Classifier:
Bootstrap Sampling: Multiple subsets of the training data are created by randomly s
contains approximately 63% of the original data, with some samples appearing multip
Train Multiple Base Models: A classifier (e.g., Decision Tree, SVM) is trained on e
Aggregate Predictions: For classification tasks, predictions from all base models a
frequent class is selected).
Final Prediction: The class with the highest votes across all models is chosen as t

Advantages of Bagging Classifier:
Reduces Variance: Since multiple models are trained on different data subsets, Bagg
overfitting.
Enhances Stability: Works well with high-variance models like Decision Trees, makin
Handles Noisy Data: Since the training data varies across models, the classifier is
'''
#How do you evaluate a Bagging Classifier's performance
'''
To evaluate a Bagging Classifier, several metrics are used to assess its performanc
common approach is to split the dataset into training and testing sets, train the m
performance on the test set using metrics like accuracy, precision, recall, F1-scor
-validation can be used to obtain a more reliable performance estimate by averaging
Another key evaluation metric specific to bagging is the Out-of-Bag (OOB) Score, wh
evaluating each tree on the data points that were not used for training that partic
model generalization, reducing the need for a separate validation set. Overall, a B
ability to reduce variance, prevent overfitting, and maintain high predictive accur
'''
#How does a Bagging Regressor work
'''
A Bagging Regressor works based on the Bootstrap Aggregating (Bagging) technique, w
the stability of regression models. It is particularly useful for high-variance mod
accuracy by averaging multiple predictions.

Working Principle of Bagging Regressor:
Bootstrap Sampling: The training dataset is randomly sampled with replacement to cr
contains approximately 63% of the original training data, meaning some data points
out.
Train Multiple Base Regressors: Each subset is used to train an individual base reg
etc.). Since the data is randomly sampled, the individual models will be slightly d
Aggregate Predictions: Once all base models are trained, they make predictions on n
obtained by averaging the outputs of all models, which helps reduce variance and in
model.
Out-of-Bag (OOB) Score (Optional): Since each tree is trained on a different subset
used to evaluate the model's performance without requiring a separate validation se
'''
#What is the main advantage of ensemble techniques
'''
```

```
The main advantage of ensemble techniques is their ability to improve model perform
combining multiple models. Instead of relying on a single weak learner, ensemble me
models to reduce bias, variance, and overfitting, leading to better accuracy and ro
strategies such as Bagging (which reduces variance by averaging multiple models tra
(which reduces bias by sequentially improving weak models), ensembles can handle co
improve model resilience to noise and anomalies, making them more reliable in real-
the cost of increased computational complexity and reduced interpretability. Overal
between accuracy and robustness, making them highly effective in various machine le
'''
#What is the main challenge of ensemble methods
'''
The main challenge of ensemble methods is their increased computational complexity
single models. Since ensembles combine multiple models (e.g., in Bagging, Boosting,
require significantly more computation, especially with large datasets. This can ma
Another challenge is interpretability, as ensemble methods, particularly those usin
Gradient Boosting, make it difficult to understand how individual predictions are m
regression or decision trees, ensembles act as "black boxes," making them harder to
may lead to overfitting if not tuned properly, particularly in Boosting methods whe
potentially fitting noise in the data. Balancing performance, computational efficie
using ensemble methods in real-world applications.
'''
# Explain the key idea behind ensemble techniques
'''
The key idea behind ensemble techniques is to combine multiple models to improve ov
generalization. Instead of relying on a single model, ensemble methods aggregate pr
reduce bias, variance, and overfitting, leading to better accuracy and robustness.
that a group of diverse models can make better predictions than an individual model
Bagging (Bootstrap Aggregating) and Boosting. Bagging(e.g., Random Forest) trains m
bootstrap samples and averages their outputs to reduce variance. **Boosting** (e.g.
sequentially, where each new model corrects the errors of the previous one, reducin
another ensemble method that combines different models using a meta-learner to make
and aggregation, ensemble methods create **more accurate, stable, and generalizable
complex machine learning problems.
'''
#What is a Random Forest Classifier
'''
A random forest classifier is an ensemble learning algorithm that builds multiple d
to improve accuracy and reduce overfitting. It follows the bagging (bootstrap aggre
on a different randomly sampled subset of the training data, and at each split, a r
introduce diversity. For classification, the final prediction is made using majorit
enhances stability, generalization, and resistance to noise while preventing overfi
decision trees. Random forest also provides feature importance scores, making it us
used in applications requiring high accuracy, such as medical diagnosis, fraud dete
robustness and ability to handle large, high-dimensional datasets effectively.
'''
#What are the main types of ensemble techniques
'''
The main types of ensemble techniques are bagging, boosting, stacking, and voting.

Bagging (bootstrap aggregating) reduces variance by training multiple models on dif
averaging their predictions. Random Forest is a popular example of bagging.Boosting
learners sequentially, where each new model focuses on correcting the errors of the
Gradient Boosting, and XGBoost.Stacking (stacked generalization) combines multiple
how to best aggregate the predictions of base models. Unlike bagging and boosting,
variations of the same model.Voting combines multiple models where each model indep
prediction is based on majority voting (for classification) or averaging (for regre
```

```
are diverse.Each ensemble technique has its strengths, with bagging reducing overfi
focusing on difficult cases, stacking leveraging multiple model types, and voting e
classifiers.
'''
#What is ensemble learning in machine learning
'''
Ensemble learning in machine learning is a technique that combines multiple models
and generalization. Instead of relying on a single model, ensemble methods aggregat
reduce bias, variance, and overfitting. The main types of ensemble techniques inclu
on different subsets of data to reduce variance; boosting, which trains models sequ
stacking, which combines diverse models using a meta-learner; and voting, which agg
enhance robustness. Ensemble learning is widely used in real-world applications suc
recommendation systems, as it often outperforms individual models by leveraging the
'''
#When should we avoid using ensemble methods
'''
Ensemble methods should be avoided in situations where they add unnecessary complex
If a simple model (such as logistic regression or a single decision tree) performs
an ensemble may introduce excessive computational overhead without much benefit. En
interpretability is a priority, as they make it difficult to explain individual pre
Additionally, when working with small datasets, ensembles may overfit rather than i
models requires sufficient data diversity. If real-time predictions are needed, ens
computational cost, making them less suitable for applications with strict time con
'''
#How does Bagging help in reducing overfitting
'''
Bagging helps in reducing overfitting by training multiple models on different **ra
and then averaging their predictions. Since each base model (e.g., decision tree) i
they learn different patterns, reducing their dependency on specific noise or anoma
lower variance, as the combined model does not rely on a single overfitted hypothes
models. In classification, bagging uses **majority voting**, and in regression, it
smoothing out extreme values and preventing the model from memorizing the training
algorithm, further enhances this effect by introducing **feature randomness**, ensu
'''
#Why is Random Forest better than a single Decision Tree
'''
Random Forest is better than a single decision tree because it improves accuracy, r
While a single decision tree tends to overfit the training data, making it less eff
mitigates this issue by combining multiple decision trees trained on different boot
the predictions for regression or using majority voting for classification, Random
stable and robust. Additionally, it introduces feature randomness, ensuring that ea
features at each split, leading to more diverse models and preventing over-reliance
bagging, feature randomness, and ensemble averaging makes Random Forest more accura
of handling high-dimensional data better than a single decision tree.
'''
#What is the role of bootstrap sampling in Bagging
'''
Bootstrap sampling in bagging plays a crucial role in reducing variance and improvi
diverse training subsets from the original dataset. It involves randomly selecting
some samples may appear multiple times while others may be left out. Each base mode
bootstrapped subset, leading to slight variations in learning. This diversity among
individual models capture different aspects of the data rather than memorizing spec
regression or using majority voting in classification, bagging ensures a more gener
sensitive to noise and anomalies in the training data.
'''
#What are some real-world applications of ensemble techniques
```

```
'''
Ensemble techniques are widely used in various real-world applications due to their
and generalization. Some common applications include:

Fraud Detection – Financial institutions use ensemble methods like Random Forest an
transactions by analyzing patterns in banking and credit card data.

Medical Diagnosis – Ensemble learning helps in disease prediction, medical image an
combining multiple models for more accurate diagnoses.

Recommendation Systems – Platforms like Netflix, Amazon, and YouTube use ensemble t
by combining different models that analyze user preferences and behaviors.

Finance and Stock Market Prediction – Financial institutions use ensembles to predi
transactions, and assess credit risk by aggregating predictions from multiple model

Customer Segmentation and Marketing – Businesses use ensemble learning to analyze c
optimize marketing campaigns for better targeting and personalization.

Fraud Detection – Banks and e-commerce platforms use ensemble methods to identify f
fraud, and prevent financial crimes.

Natural Language Processing (NLP) – Sentiment analysis, spam detection, language tr
ensembles to increase accuracy and reliability.

Image and Speech Recognition – Face recognition, object detection, medical image an
ensemble learning for better accuracy and robustness against variations in data.

Medical Diagnosis – Machine learning ensembles help in disease prediction, medical
X-rays or MRIs), and drug discovery by combining multiple models for better predict

Recommendation Systems – Online platforms like Netflix, Amazon, and Spotify use ens
by analyzing user behavior and preferences.
'''
#What is the difference between Bagging and Boosting?
'''
Bagging and boosting are both ensemble learning techniques used to improve the accu
models, but they work in different ways.Bagging, short for bootstrap aggregating, i
different subsets of the data that are randomly sampled **with replacement**. Each
final prediction is made by averaging (for regression) or majority voting (for clas
and overfitting, making it useful for high-variance models like decision trees. A p
Forest algorithm.
On the other hand, boosting is a sequential technique where each model is trained t
Models are trained one after another, and each new model gives more importance to m
improving overall performance. Boosting focuses on reducing bias and is effective f
Examples include AdaBoost, Gradient Boosting, and XGBoost.
The main difference between bagging and boosting is how models are trained. Bagging
parallel and averages their predictions to reduce variance, whereas boosting trains
learning from the errors of its predecessor, reducing both bias and variance. Baggi
overfitting in high-variance models, while boosting is more effective for improving
prone to overfitting if not properly tuned.
```

Practical

In [1]:
```
#Train a Bagging Classifier using Decision Trees on a sample dataset and print mode
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_iris()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimato
bagging_clf.fit(X_train, y_train)

y_pred = bagging_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Classifier Accuracy: {accuracy:.2f}")
```

```
Bagging Classifier Accuracy: 1.00
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
```

In [2]:
```
#Train a Bagging Regressor using Decision Trees and evaluate using Mean Squared Err
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


data = load_diabetes()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


bagging_reg = BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_estimators
bagging_reg.fit(X_train, y_train)

y_pred = bagging_reg.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Bagging Regressor Mean Squared Error: {mse:.2f}")
```

```
Bagging Regressor Mean Squared Error: 3056.49
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
```

In [3]:
```
#2 Train a Random Forest Classifier on the Breast Cancer dataset and print feature
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, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

feature_importances = rf_clf.feature_importances_

for feature, importance in zip(data.feature_names, feature_importances):
    print(f"{feature}: {importance:.4f}")
```

```
mean radius: 0.0487
mean texture: 0.0136
mean perimeter: 0.0533
mean area: 0.0476
mean smoothness: 0.0073
mean compactness: 0.0139
mean concavity: 0.0680
mean concave points: 0.1062
mean symmetry: 0.0038
mean fractal dimension: 0.0039
radius error: 0.0201
texture error: 0.0047
perimeter error: 0.0113
area error: 0.0224
smoothness error: 0.0043
compactness error: 0.0053
concavity error: 0.0094
concave points error: 0.0035
symmetry error: 0.0040
fractal dimension error: 0.0053
worst radius: 0.0780
worst texture: 0.0217
worst perimeter: 0.0671
worst area: 0.1539
worst smoothness: 0.0106
worst compactness: 0.0203
worst concavity: 0.0318
worst concave points: 0.1447
worst symmetry: 0.0101
worst fractal dimension: 0.0052
```

In [6]:
```
#Train a Random Forest Regressor and compare its performance with a single Decision
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import load_diabetes  # Changed from fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
# Load dataset
data = load_diabetes()  # Replaced with a local dataset
X, y = data.data, data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Create and train a Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train, y_train)
y_pred_dt = dt_regressor.predict(X_test)
dt_mse = mean_squared_error(y_test, y_pred_dt)
print(f"Decision Tree Regressor Mean Squared Error: {dt_mse:.2f}")

# Create and train a Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)
y_pred_rf = rf_regressor.predict(X_test)
rf_mse = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Regressor Mean Squared Error: {rf_mse:.2f}")
```

```
Decision Tree Regressor Mean Squared Error: 4976.80
Random Forest Regressor Mean Squared Error: 2952.01
```

In [7]:
```
#2 Compute the Out-of-Bag (OOB) Score for a Random Forest Classifier2
from sklearn.ensemble import RandomForestRegressor
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


data = load_diabetes()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


rf_regressor = RandomForestRegressor(n_estimators=100, oob_score=True, random_state
rf_regressor.fit(X_train, y_train)


oob_score = rf_regressor.oob_score_
print(f"Out-of-Bag (OOB) Score: {oob_score:.2f}")


y_pred_rf = rf_regressor.predict(X_test)
rf_mse = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Regressor Mean Squared Error: {rf_mse:.2f}")
```

```
Out-of-Bag (OOB) Score: 0.45
Random Forest Regressor Mean Squared Error: 2952.01
```

In [8]:
```
#Train a Bagging Classifier using SVM as a base estimator and print accuracy2
```

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


data = load_digits()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


svm_base = SVC(kernel='rbf', probability=True)
bagging_clf = BaggingClassifier(base_estimator=svm_base, n_estimators=10, random_st


bagging_clf.fit(X_train, y_train)


y_pred = bagging_clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Classifier Accuracy with SVM: {accuracy:.2f}")
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
Bagging Classifier Accuracy with SVM: 0.99
```

In [9]:
```
#2 Train a Random Forest Classifier with different numbers of trees and compare acc
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_digits()
X, y = data.data, data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Different numbers of trees to compare
n_estimators_list = [10, 50, 100, 200]

# Train and evaluate models
for n_estimators in n_estimators_list:
    rf_clf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf_clf.fit(X_train, y_train)
    y_pred = rf_clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Random Forest with {n_estimators} trees - Accuracy: {accuracy:.4f}")
```

```
Random Forest with 10 trees - Accuracy: 0.9583
Random Forest with 50 trees - Accuracy: 0.9722
Random Forest with 100 trees - Accuracy: 0.9722
Random Forest with 200 trees - Accuracy: 0.9694
```

In [10]:
```python
# Train a Bagging Classifier using Logistic Regression as a base estimator and prin
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Create and train a Bagging Classifier with Logistic Regression as base estimator
bagging_clf = BaggingClassifier(base_estimator=LogisticRegression(), n_estimators=5
bagging_clf.fit(X_train, y_train)

# Predict probabilities for AUC computation
y_probs = bagging_clf.predict_proba(X_test)[:, 1]  # Get probability of positive cl

# Compute AUC score
auc_score = roc_auc_score(y_test, y_probs)
print(f"Bagging Classifier with Logistic Regression - AUC Score: {auc_score:.4f}")
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
Bagging Classifier with Logistic Regression - AUC Score: 0.9219
```

In [12]:
```python
#Train a Random Forest Regressor and analyze feature importance scores2
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
import numpy as np

data = load_diabetes()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

feature_importances = rf_regressor.feature_importances_

feature_names = data.feature_names
for name, importance in zip(feature_names, feature_importances):
    print(f"{name}: {importance:.4f}")

sorted_indices = np.argsort(feature_importances)[::-1]
```

```
print("\nFeature Importance Ranking:")
for i in sorted_indices:
    print(f"{feature_names[i]}: {feature_importances[i]:.4f}")
```

```
age: 0.0586
sex: 0.0096
bmi: 0.3555
bp: 0.0884
s1: 0.0528
s2: 0.0572
s3: 0.0513
s4: 0.0242
s5: 0.2310
s6: 0.0713

Feature Importance Ranking:
bmi: 0.3555
s5: 0.2310
bp: 0.0884
s6: 0.0713
age: 0.0586
s2: 0.0572
s1: 0.0528
s3: 0.0513
s4: 0.0242
sex: 0.0096
```

In [13]:
```
#Train an ensemble model using both Bagging and Random Forest and compare accuracy
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimato
bagging_clf.fit(X_train, y_train)
y_pred_bagging = bagging_clf.predict(X_test)
bagging_acc = accuracy_score(y_test, y_pred_bagging)

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
rf_acc = accuracy_score(y_test, y_pred_rf)

print(f"Bagging Classifier Accuracy: {bagging_acc:.4f}")
print(f"Random Forest Classifier Accuracy: {rf_acc:.4f}")
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
```

```
Bagging Classifier Accuracy: 0.9722
Random Forest Classifier Accuracy: 1.0000
```

In [16]:
```python
#Train a Random Forest Classifier and tune hyperparameters using GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=
grid_search.fit(X_train, y_train)

best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"Accuracy: {accuracy:.4f}")
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 100}
Accuracy: 1.0000
```

In [17]:
```python
#Train a Bagging Regressor with different numbers of base estimators and compare pe
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

data = load_diabetes()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

n_estimators_list = [10, 50, 100, 200]
mse_scores = {}

for n in n_estimators_list:
    bagging_regressor = BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_
    bagging_regressor.fit(X_train, y_train)
    y_pred = bagging_regressor.predict(X_test)
    mse_scores[n] = mean_squared_error(y_test, y_pred)
```

```
for n, mse in mse_scores.items():
    print(f"Bagging Regressor (n_estimators={n}) - MSE: {mse:.4f}")
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
Bagging Regressor (n_estimators=10) - MSE: 3256.9618
Bagging Regressor (n_estimators=50) - MSE: 3056.4946
Bagging Regressor (n_estimators=100) - MSE: 2970.8632
Bagging Regressor (n_estimators=200) - MSE: 2995.6186
```

In [18]:

```
#Train a Random Forest Classifier and analyze misclassified samples
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

misclassified_indices = np.where(y_test != y_pred)[0]
print(f"Number of misclassified samples: {len(misclassified_indices)}")

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

misclassified_samples = X_test[misclassified_indices]
actual_labels = y_test[misclassified_indices]
predicted_labels = y_pred[misclassified_indices]
```

```
for i in range(len(misclassified_indices)):
    print(f"Sample {i+1}: Actual={actual_labels[i]}, Predicted={predicted_labels[i]
```

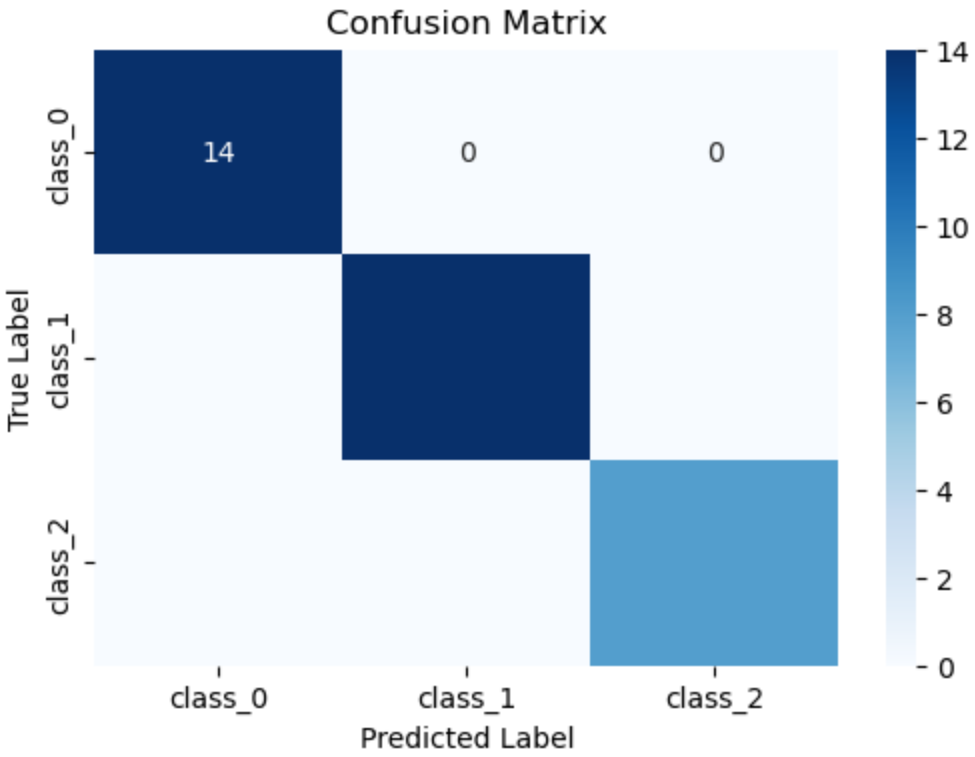
```
Accuracy: 1.0000
Number of misclassified samples: 0
Confusion Matrix:
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
```

In [19]:
```
#Train a Bagging Classifier and compare its performance with a single Decision Tree
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
y_pred_dt = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, y_pred_dt)

bagging_classifier = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(), n_estimators=50, random_state=42
)
bagging_classifier.fit(X_train, y_train)
y_pred_bagging = bagging_classifier.predict(X_test)
bagging_accuracy = accuracy_score(y_test, y_pred_bagging)

print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
print(f"Bagging Classifier Accuracy: {bagging_accuracy:.4f}")
```

```
Decision Tree Accuracy: 0.9444
Bagging Classifier Accuracy: 0.9722
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/sklearn/ensembl
e/_base.py:156: FutureWarning: `base_estimator` was renamed to `estimator` in versio
n 1.2 and will be removed in 1.4.
  warnings.warn(
```

In [20]:
```
#Train a Random Forest Classifier and visualize the confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=data.target_names, y
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



In [22]:
```
#Train a Stacking Classifier using Decision Trees, SVM, and Logistic Regression, an
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

base_learners = [
    ('decision_tree', DecisionTreeClassifier(random_state=42)),
```

```
    ('svm', make_pipeline(StandardScaler(), SVC(probability=True, random_state=42))
    ('logistic', make_pipeline(StandardScaler(), LogisticRegression(max_iter=5000,
]

stacking_classifier = StackingClassifier(estimators=base_learners, final_estimator=
stacking_classifier.fit(X_train, y_train)
y_pred = stacking_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Stacking Classifier Accuracy: {accuracy:.2f}")
```

```
Stacking Classifier Accuracy: 1.00
```

In [23]:
```
# Train a Random Forest Classifier and print the top 5 most important features
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import numpy as np

data = load_wine()
X, y = data.data, data.target
feature_names = data.feature_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

importances = rf_classifier.feature_importances_
sorted_indices = np.argsort(importances)[::-1]  # Sort in descending order

print("Top 5 most important features:")
for i in range(5):
    print(f"{feature_names[sorted_indices[i]]}: {importances[sorted_indices[i]]:.4f
```

```
Top 5 most important features:
flavanoids: 0.2023
color_intensity: 0.1712
proline: 0.1390
alcohol: 0.1124
od280/od315_of_diluted_wines: 0.1116
```

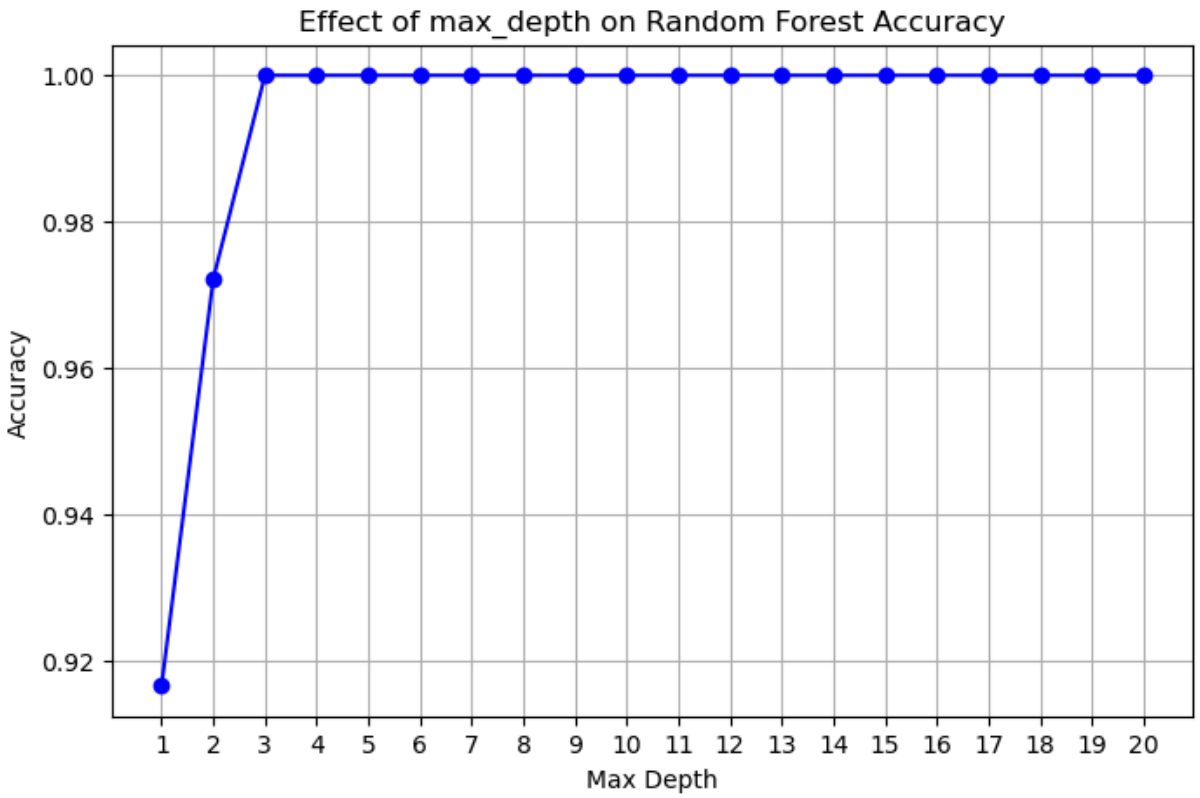
In [25]:
```
#= Train a Bagging Classifier and evaluate performance using Precision, Recall, and
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score

data = load_wine()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

bagging_classifier = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estima
bagging_classifier.fit(X_train, y_train)
```

```
y_pred = bagging_classifier.predict(X_test)

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
Precision: 0.97
Recall: 0.97
F1-score: 0.97
```

In [26]:

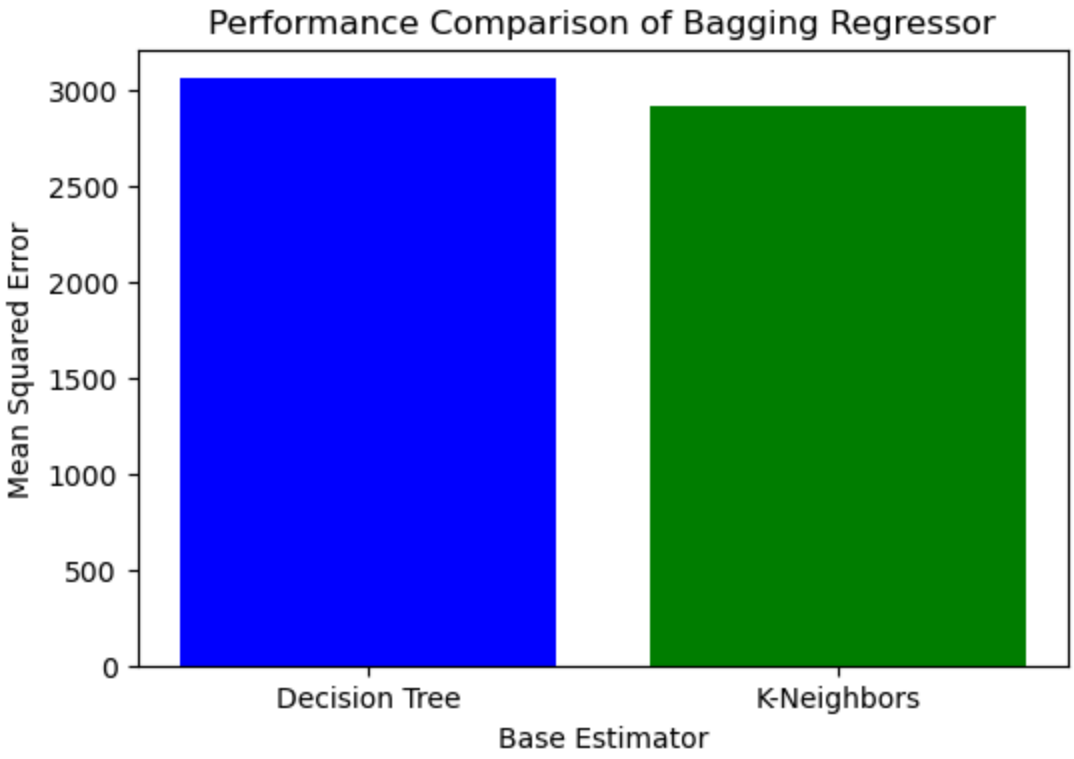
```
#Train a Random Forest Classifier and analyze the effect of max_depth on accuracy
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_wine()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Evaluate different max_depth values
max_depth_values = range(1, 21)
accuracies = []

for depth in max_depth_values:
    rf = RandomForestClassifier(n_estimators=100, max_depth=depth, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))

# Plot the effect of max_depth on accuracy
plt.figure(figsize=(8, 5))
plt.plot(max_depth_values, accuracies, marker='o', linestyle='-', color='b')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Effect of max_depth on Random Forest Accuracy')
plt.xticks(max_depth_values)
plt.grid()
plt.show()
```



In [28]:
```
#Train a Bagging Regressor using different base estimators (DecisionTree and KNeigh
import numpy as np
import matplotlib.pyplot as plt
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

# Load dataset
data = load_diabetes()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Define base estimators
estimators = {
    "Decision Tree": DecisionTreeRegressor(),
    "K-Neighbors": KNeighborsRegressor()
}

# Train and evaluate Bagging Regressor with different base estimators
results = {}

for name, estimator in estimators.items():
    model = BaggingRegressor(estimator=estimator, n_estimators=50, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
```

```
    results[name] = mse
    print(f"Bagging Regressor with {name}: Mean Squared Error = {mse:.4f}")

# Plot comparison
plt.figure(figsize=(6, 4))
plt.bar(results.keys(), results.values(), color=['blue', 'green'])
plt.xlabel("Base Estimator")
plt.ylabel("Mean Squared Error")
plt.title("Performance Comparison of Bagging Regressor")
plt.show()
```

```
Bagging Regressor with Decision Tree: Mean Squared Error = 3056.4946
Bagging Regressor with K-Neighbors: Mean Squared Error = 2918.7795
```

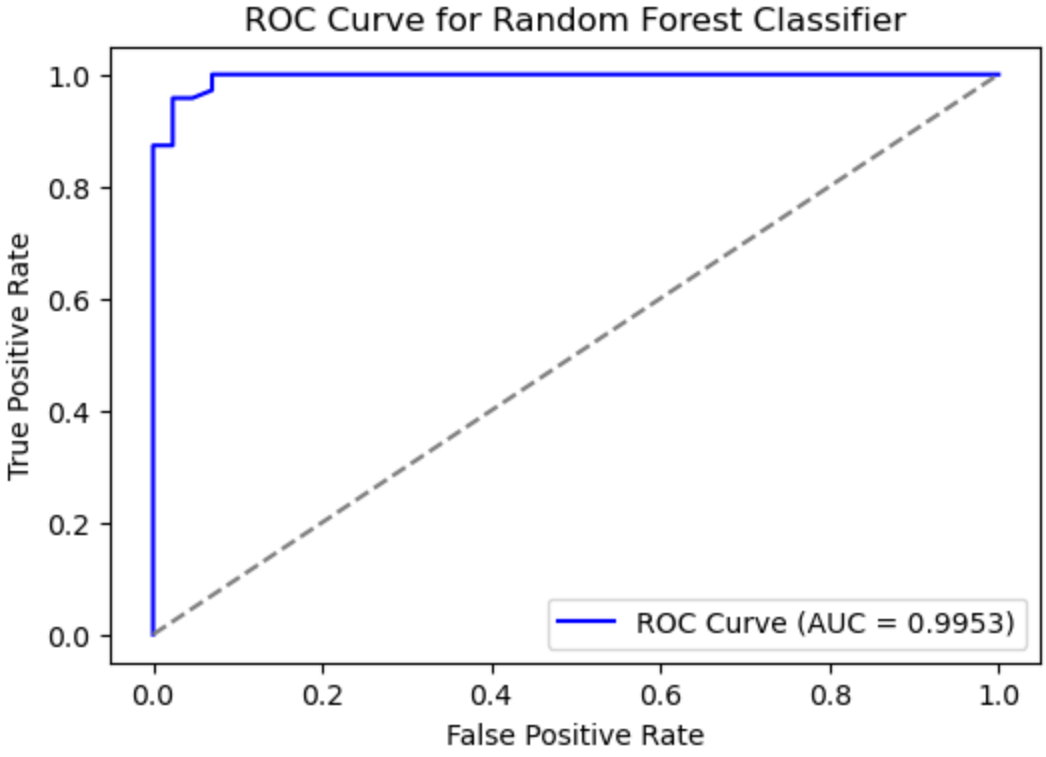


In [29]:

```
#= Train a Random Forest Classifier and evaluate its performance using ROC-AUC Scor
import numpy as np
import matplotlib.pyplot as plt
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, roc_curve

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Train Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
# Predict probabilities
y_prob = rf_classifier.predict_proba(X_test)[:, 1]

# Compute ROC-AUC Score
auc_score = roc_auc_score(y_test, y_prob)
print(f"Random Forest Classifier ROC-AUC Score: {auc_score:.4f}")

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc_score:.4f})", color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Random Forest Classifier")
plt.legend()
plt.show()
```

```
Random Forest Classifier ROC-AUC Score: 0.9953
```



In [31]:

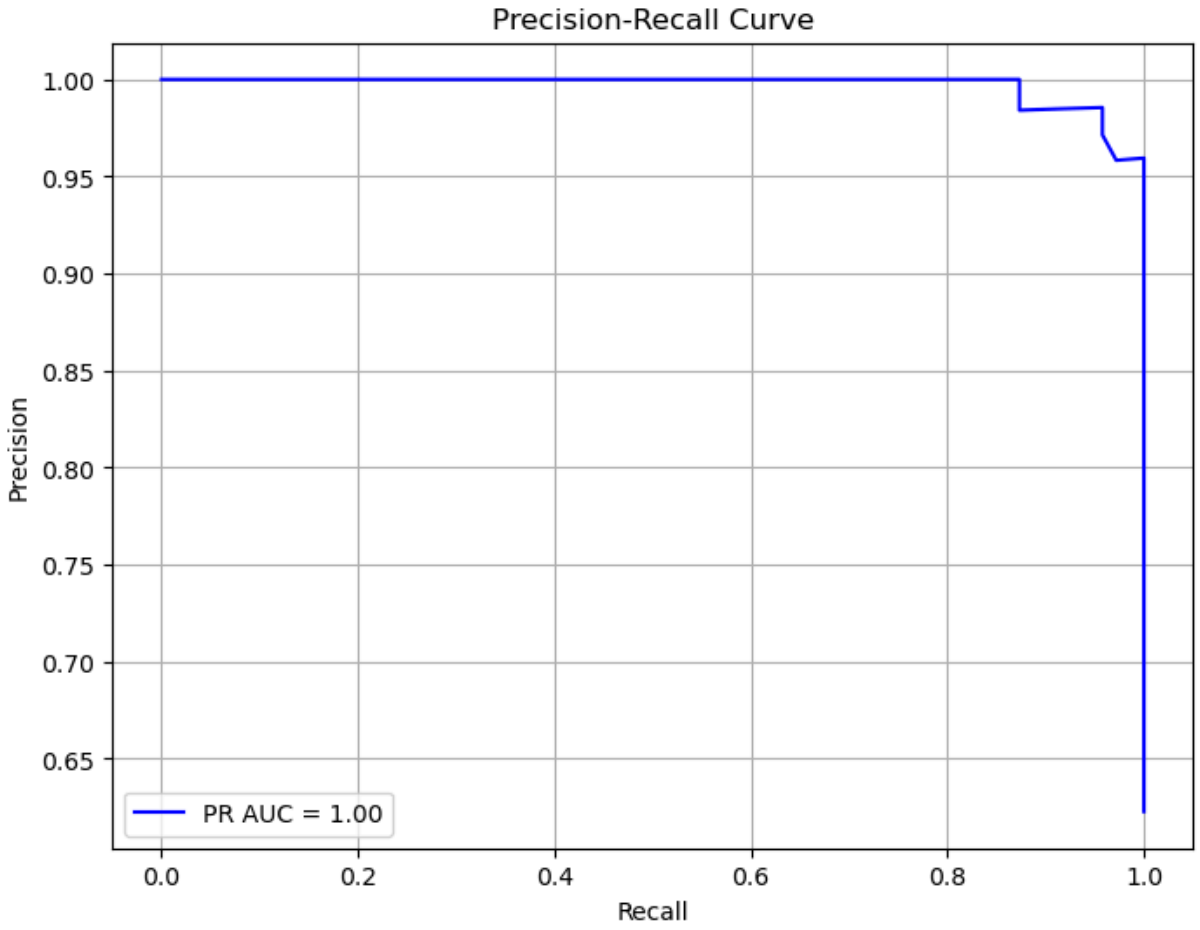
```
#= Train a Bagging Classifier and evaluate its performance using cross-validatio.
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
import numpy as np

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Define Bagging Classifier with Decision Tree as base estimator
bagging_clf = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=50
```

```
# Perform cross-validation
cv_scores = cross_val_score(bagging_clf, X, y, cv=5, scoring="accuracy")

# Print results
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean Accuracy: {np.mean(cv_scores):.4f}")
print(f"Standard Deviation: {np.std(cv_scores):.4f}")
```

```
Cross-Validation Scores: [0.9122807  0.92105263 0.98245614 0.95614035 1.        ]
Mean Accuracy: 0.9544
Standard Deviation: 0.0339
```

In [32]:

```
#Train a Random Forest Classifier and plot the Precision-Recall curv
import numpy as np
import matplotlib.pyplot as plt
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, auc

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Train a Random Forest Classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

# Get predicted probabilities
y_scores = rf_clf.predict_proba(X_test)[:, 1]

# Compute precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_scores)

# Compute AUC (Area Under Curve)
pr_auc = auc(recall, precision)

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f"PR AUC = {pr_auc:.2f}", color="blue")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.grid()
plt.show()
```



In [33]:
```
#Train a Stacking Classifier with Random Forest and Logistic Regression and compare
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

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Define base learners
base_learners = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42))
]

# Define Stacking Classifier with Logistic Regression as final estimator
stacking_clf = StackingClassifier(estimators=base_learners, final_estimator=Logisti

# Train Stacking Classifier
stacking_clf.fit(X_train, y_train)

# Predict and evaluate accuracy
y_pred = stacking_clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)

print(f"Stacking Classifier Accuracy: {accuracy:.2f}")
```

```
Stacking Classifier Accuracy: 0.96
```

In [35]:
```
# Train a Bagging Regressor with different levels of bootstrap samples and compare
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load synthetic dataset
X, y = make_regression(n_samples=1000, n_features=10, noise=0.1, random_state=42)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Evaluate Bagging Regressor with different bootstrap sample sizes
bootstrap_options = [True, False]

for bootstrap in bootstrap_options:
    bagging_regressor = BaggingRegressor(
        estimator=DecisionTreeRegressor(),  # Updated from base_estimator to estima
        n_estimators=100,
        bootstrap=bootstrap,
        random_state=42
    )
    bagging_regressor.fit(X_train, y_train)
    y_pred = bagging_regressor.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Bagging Regressor (bootstrap={bootstrap}) Mean Squared Error: {mse:.2f}
```

```
Bagging Regressor (bootstrap=True) Mean Squared Error: 2631.84
Bagging Regressor (bootstrap=False) Mean Squared Error: 6250.25
```

In [ ]:
```

```