PART A: THEORY QUESTIONS (Easy Language)

Q1. Can we use Bagging for regression problems?

Ans. Yes, Bagging can be used for regression problems. In regression, Bagging trains multiple regression models on different bootstrap samples and averages their predictions to reduce error.

Q2. Difference between multiple model training and single model training

Ans. single model training uses only one model to make predictions.

Multiple model training trains many models and combines their output to improve accuracy and stability.

Q3. Feature randomness in Random Forest

Ans. Random Forest selects a random subset of features for each split in a tree. This reduces correlation between trees and improves overall performance.

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

Ans. OOB score is an internal validation method where data not used in training a tree is used to test it. It helps estimate model accuracy without a separate test set.

Q5. Feature importance in Random Forest

Ans. Random Forest measures feature importance based on:

Reduction in impurity

Contribution of each feature to model accuracy

Q6. Working principle of Bagging Classifier

Ans. Create multiple bootstrap samples

Train classifiers on each sample

Combine predictions using majority voting

Q7. Evaluation of Bagging Classifier

Ans. Performance can be evaluated using:

Accuracy

Precision, Recall, F1-score

ROC-AUC

Q8. How does Bagging Regressor work?

Ans. Bagging Regressor trains multiple regressors and takes the average of predictions to reduce variance.

Q9. Main advantage of ensemble techniques

Ans. They increase accuracy and reduce overfitting compared to single models.

Q10. Main challenge of ensemble methods

Ans. High computational cost

Less interpretability

Q11. Key idea behind ensemble techniques

Ans. Combining multiple weak models to create a strong model.

Q12. What is a Random Forest Classifier?

Ans. A Random Forest Classifier is an ensemble of decision trees trained using bagging and feature randomness.

Q13. Types of ensemble techniques

Ans. Bagging

Boosting

Stacking

Q14. What is ensemble learning?

Ans. Ensemble learning combines predictions from multiple models to improve performance.

Q15. When should we avoid ensemble methods?

Ans. When data is small

When model interpretability is important

Limited computing power

Q16. How Bagging reduces overfitting

Ans. By averaging predictions from multiple models trained on different samples.

Q17. Why Random Forest is better than single Decision Tree

Ans. It reduces overfitting and improves accuracy.

Q18. Role of bootstrap sampling in Bagging

Ans. Bootstrap sampling creates different datasets by random sampling with replacement.

Q19. Real-world applications of ensemble techniques

Ans. Fraud detection

Medical diagnosis

Stock market prediction

Recommendation systems

Q20. Difference between Bagging and Boosting

Bagging Boosting

Models trained independently Models trained sequentially

Reduces variance Reduces bias

Part-B Practical Questions

```
#Ans21. Bagging Classifier with Decision Tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

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

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

```
Accuracy: 1.0
```

```
#Ans22. Bagging Regressor with MSE
from sklearn.datasets import load_diabetes
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)

model = BaggingRegressor(
    estimator=DecisionTreeRegressor(),
    n_estimators=50
)
model.fit(X, y)
pred = model.predict(X)

print("MSE:", mean_squared_error(y, pred))
```

```
MSE: 496.7668886877828
```

```
#Ans23. Random Forest Classifier – Feature Importance
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier

X, y = load_breast_cancer(return_X_y=True)

rf = RandomForestClassifier(n_estimators=100)
```

```
rf.fit(X, y)

print(rf.feature_importances_)
```

```
[0.04358491 0.01657074 0.0566336  0.05370734 0.00619668 0.00582527
 0.03030234 0.10803607 0.0031623  0.00296843 0.0091137  0.00314255
 0.00696574 0.05403568 0.00418359 0.00497882 0.00803527 0.00684839
 0.00526496 0.00518077 0.14325934 0.01936686 0.12675686 0.11401296
 0.01382592 0.01271619 0.02453337 0.09431863 0.00860227 0.00787042]
```

```
#Ans24.Random Forest vs Decision Tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

dt = DecisionTreeRegressor()
dt.fit(X, y)

print("DT MSE:", mean_squared_error(y, dt.predict(X)))
```

```
DT MSE: 0.0
```

```
#Ans25. OOB Score
rf = RandomForestClassifier(
    n_estimators=100,
    oob_score=True,
    bootstrap=True
)
rf.fit(X, y)

print("OOB Score:", rf.oob_score_)
```

```
OOB Score: 0.0
```

```
#Ans26. Random Forest with GridSearchCV
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators':[50,100],
    'max_depth':[5,10,None]
}

grid = GridSearchCV(RandomForestClassifier(), params)
grid.fit(X, y)

print(grid.best_params_)
```

```
{'max_depth': 10, 'n_estimators': 100}
```

```
#Ans27. StackingClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

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

stack = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression()
)

stack.fit(X_train, y_train)
print("Stacking Accuracy:", stack.score(X_test, y_test))
```

```
Stacking Accuracy: 0.9333333333333333
```

```
#Ans28. Confusion Matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[2 3 2]
 [6 4 6]
 [4 2 1]]
```

```
#Ans29. Train a Random Forest Classifier and tune hyperparameters using GridSearchCV
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 5, 10]
}

grid = GridSearchCV(RandomForestClassifier(), param_grid)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best Accuracy:", grid.best_score_)
```

```
Best Parameters: {'max_depth': 5, 'n_estimators': 100}
Best Accuracy: 0.9666666666666668
```

```
#Ana30.Train a Bagging Regressor with different numbers of estimators
from sklearn.datasets import load_diabetes
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)

for n in [10, 50, 100]:
    model = BaggingRegressor(
        estimator=DecisionTreeRegressor(),
        n_estimators=n
    )
    model.fit(X, y)
    pred = model.predict(X)
    print(f"Estimators: {n}, MSE:", mean_squared_error(y, pred))
```

```
Estimators: 10, MSE: 623.6270135746606
Estimators: 50, MSE: 478.4904660633484
Estimators: 100, MSE: 454.71061266968326
```

```
#Ans31. Train a Random Forest Classifier and analyze misclassified samples
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

pred = rf.predict(X_test)

misclassified = X_test[pred != y_test]
print("Number of Misclassified Samples:", len(misclassified))
```

```
Number of Misclassified Samples: 0
```

```
#Ans32. Compare Bagging Classifier with single Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

bag = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=50
)
bag.fit(X_train, y_train)

print("Decision Tree Accuracy:", dt.score(X_test, y_test))
print("Bagging Accuracy:", bag.score(X_test, y_test))
```

```
Decision Tree Accuracy: 1.0
Bagging Accuracy: 1.0
```

```
#Ans33. Random Forest Confusion Matrix
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
```

```
# Ensure X_train, y_train, X_test, y_test are from the Iris dataset
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Classifier and get predictions
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
pred = rf.predict(X_test)

cm = confusion_matrix(y_test, pred)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
#Ans34. Stacking Classifier (DT + SVM + Logistic Regression)
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

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

stack = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression()
)

stack.fit(X_train, y_train)
print("Stacking Accuracy:", stack.score(X_test, y_test))
```

```
Stacking Accuracy: 0.9333333333333333
```

```
#Ans35. Random Forest – Top 5 Important Features
import numpy as np

importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

print("Top 5 Features:")
for i in indices[:5]:
    print(importances[i])
```

```
Top 5 Features:
0.12328635140149158
0.12024825985108482
0.11542114456349123
0.11376376884389261
0.11284888922284506
```

```
#Ans36. Bagging Classifier – Precision, Recall, F1-score
from sklearn.metrics import precision_score, recall_score, f1_score

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

```
Precision: 0.2407407407407407
Recall: 0.2261904761904762
F1 Score: 0.21850877192982457
```

```
#Ans37.Effect of max_depth on Random Forest Accuracy
for depth in [3, 5, 10, None]:
    rf = RandomForestClassifier(max_depth=depth)
    rf.fit(X_train, y_train)
    print(f"Max Depth {depth} Accuracy:", rf.score(X_test, y_test))
```

```
Max Depth 3 Accuracy: 1.0
Max Depth 5 Accuracy: 1.0
Max Depth 10 Accuracy: 1.0
Max Depth None Accuracy: 1.0
```

```
#Ans38. Bagging Regressor with Decision Tree & KNN
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_diabetes

X, y = load_diabetes(return_X_y=True)

models = [
    DecisionTreeRegressor(),
    KNeighborsRegressor()
]

for model in models:
    bag = BaggingRegressor(estimator=model, n_estimators=50)
    bag.fit(X, y)
    pred = bag.predict(X)
    print(model.__class__.__name__, "MSE:", mean_squared_error(y, pred))
```

```
DecisionTreeRegressor MSE: 536.7758895927602
KNeighborsRegressor MSE: 2242.987424760181
```

```
#Ans39. Random Forest – ROC AUC Score
from sklearn.metrics import roc_auc_score

prob = rf.predict_proba(X_test)
print("ROC AUC:", roc_auc_score(y_test, prob, multi_class='ovr'))
```

```
ROC AUC: 1.0
```

```
#Ans40. Bagging Classifier – Cross Validation
from sklearn.model_selection import cross_val_score

scores = cross_val_score(bag, X, y, cv=5)
print("Cross Validation Scores:", scores)
```

```
Cross Validation Scores: [1.      0.95296 0.      0.7     0.     ]
```

```
#Ans41. Precision-Recall Curve
from sklearn.metrics import precision_recall_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Ensure X_train, y_train, X_test, y_test are from the Iris dataset and rf is fitted
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Calculate probabilities
prob = rf.predict_proba(X_test)

# For multi-class, precision_recall_curve works for one class vs rest.
# Let's consider class 1. Binarize y_test for class 1.
y_test_binary = (y_test == 1)

precision, recall, _ = precision_recall_curve(y_test_binary, prob[:,1])
print("Precision values:", precision[:5])
print("Recall values:", recall[:5])
```

```
Precision values: [0.3        0.5        0.5625     0.6        0.64285714]
Recall values: [1. 1. 1. 1. 1.]
```

```
#Ans42. Stacking (Random Forest + Logistic Regression)
estimators = [
    ('rf', RandomForestClassifier()),
]

stack = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression()
)

stack.fit(X_train, y_train)
print("Stacking Accuracy:", stack.score(X_test, y_test))
```

```
Stacking Accuracy: 1.0
```

```
#Ans43. Bagging Regressor – Different Bootstrap Levels
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_diabetes

# Ensure X and y are loaded for the BaggingRegressor example
X, y = load_diabetes(return_X_y=True)

for bootstrap in [True, False]:
    bag = BaggingRegressor(
        estimator=DecisionTreeRegressor(),
        n_estimators=50,
        bootstrap=bootstrap
    )
    bag.fit(X, y)
    pred = bag.predict(X)
    print("Bootstrap:", bootstrap, "MSE:", mean_squared_error(y, pred))
```

```
Bootstrap: True MSE: 485.6286606334842
Bootstrap: False MSE: 0.0
```