Assignment Code: DA-AG-014

Ensemble Learning | **Assignment**

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 200

Question 1: What is Ensemble Learning in machine learning? Explain the key idea behind it.

Answer:

Ensemble learning is a machine-learning paradigm where multiple models are combined to produce a better overall model. The main idea is that by aggregating diverse models you can reduce variance, reduce bias, and improve predictive stability compared to any single model.

Key ideas / motivations:

- **Reduce variance**: Averaging predictions stabilizes models that are high-variance.
- **Reduce bias**: Sequential methods like boosting try to correct errors of prior learners and can reduce bias.
- **Exploit model diversity**: Different models make different errors; combining them can cancel out individual mistakes.
- **Robustness & improved generalization**: Ensembles are less sensitive to noise or particular training splits.

Question 2: What is the difference between Bagging and Boosting?

Answer:

Aspect	**Bagging (Bootstrap Aggregation)**	**Boosting**
Objective	Reduce variance by averaging many independent models	Reduce bias (and sometimes variance) by sequentially correcting errors
Training style	Train base learners in parallel on bootstrap samples	Train base learners sequentially; each focuses on previous errors
Data sampling	Uses bootstrap samples	Often uses full dataset with sample weights updated, or reweighting of examples
Example methods	Random Forest	AdaBoost, Gradient Boosting, XGBoost, LightGBM
Susceptibility	Better for high-variance models	Can over fit if boosting too long; powerful on structured data
Final prediction	Simple averaging or majority vote	Weighted sum or voting of learners

Question 3: What is bootstrap sampling and what role does it play in Bagging methods like Random Forest?

Answer:

Bootstrap sampling means sampling with replacement from the training set to create multiple training sets for training base learners.

Role in Bagging / Random Forest:

- Provides data diversity among base learners: each model sees a different subset of training data.
- Because of sampling with replacement, each bootstrap sample typically contains about 63.2% unique examples from the original dataset, leaving roughly 36.8% as out-of-bag (OOB) for that tree.
- In Random Forest, bootstrap sampling plus random feature selection helps decor relate trees, producing stronger variance reduction when their predictions are averaged.

Question 4: What are Out-of-Bag (OOB) samples and how is OOB score used to evaluate ensemble models?

Answer:

Out-of-Bag (OOB) samples are the training examples not included in a particular bootstrap sample used to train a specific base model. For each base learner, roughly 36.8% of original samples are OOB.

OOB score / use:

- OOB samples can be used as a built-in validation set to estimate generalization error without a separate holdout set or cross-validation.
- For each training example, aggregate predictions from all base learners for which the example was OOB; compare aggregated prediction to true label to compute OOB accuracy (classification) or OOB MSE (regression).
- OOB is especially handy in RandomForest: `RandomForestClassifier(oob_score=True)` provides `oob_score_`.

Benefits & caveats:

- Efficient: reuses training data; no need for separate validation.
- Slight optimistic/biased vs. an independent test set if tuning hyper parameters on OOB repeatedly, but typically a good quick estimate.

Question 5: Compare feature importance analysis in a single Decision Tree vs. a Random Forest.

Answer:

Single Decision Tree:

- Feature importance is usually computed based on impurity reduction achieved when splitting on that feature across the tree.
- Pros: easy to compute and interpret for that tree.
- Cons: Highly sensitive to small changes in training data; may overestimate importance of features with many categories or many unique values.

Random Forest:

- Aggregates feature importance across many trees, typically by averaging impurity-based importance or by using permutation importance.

- Pros: More stable and reliable than a single tree; reduces variance of importance estimates.
- Cons: Impurity-based importance can still be biased toward high-cardinality features; permutation importance is more robust but costs more to compute.

Question 6: Write a Python program to: ● Load the

Breast Cancer dataset using

`sklearn.datasets.load_breast_cancer()`

- Train a Random Forest Classifier
- Print the top 5 most important features based on feature importance scores.

(*Include your Python code and output in the code box below.*)

Answer:

Code:

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

data = load_breast_cancer()
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_state=42, stratify=y
)
rf = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
print("Top 5 features (feature : importance)")
for i in indices[:5]:
    print(f"{feature_names[i]} : {importances[i]:.4f}")
print("Test accuracy:", accuracy_score(y_test, rf.predict(X_test)))
```

Output:
```
Top 5 features (feature : importance)
worst perimeter : 0.1331
worst area : 0.1281
worst concave points : 0.1081
mean concave points : 0.0944
worst radius : 0.0906
Test accuracy: 0.956140350877193
```

Question 7: Write a Python program to:

- Train a Bagging Classifier using Decision Trees on the Iris dataset
- Evaluate its accuracy and compare with a single Decision Tree

(*Include your Python code and output in the code box below.*)
Answer:

Code:
```
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

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

# Single Decision Tree model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_acc = accuracy_score(y_test, dt.predict(X_test))
print(f"Single Decision Tree Accuracy: {dt_acc:.4f}")

# Bagging Classifier with DecisionTree base estimator
bag = BaggingClassifier(
    estimator=DecisionTreeClassifier(random_state=42)
    n_estimators=50,
    random_state=42,
    n_jobs=-1
)
bag.fit(X_train, y_train)
bag_acc = accuracy_score(y_test, bag.predict(X_test))
print(f"Bagging Classifier Accuracy: {bag_acc:.4f}")
```

Output:

```
Single Decision Tree Accuracy: 0.8947
Bagging Classifier Accuracy: 0.9211
```

Question 8: Write a Python program to:

- Train a Random Forest Classifier
- Tune hyperparameters max_depth and n_estimators using GridSearchCV
- Print the best parameters and final accuracy

(*Include your Python code and output in the code box below.*)

Answer:

Code:

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

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_state=42, stratify=y
)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20]
}

rf = RandomForestClassifier(random_state=42)
```

```
grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)

print("Best parameters:", grid.best_params_)
print("Best CV accuracy:", grid.best_score_)
best_model = grid.best_estimator_
print("Test accuracy (best model):", accuracy_score(y_test, best_model.predict(X_test)))
```

Output:

```
Best parameters: {'max_depth': None, 'n_estimators': 200}
Best CV accuracy: 0.9604395604395606
Test accuracy (best model): 0.956140350877193
```

Question 9: Write a Python program to:

- Train a Bagging Regressor and a Random Forest Regressor on the California Housing dataset
- Compare their Mean Squared Errors (MSE)

(*Include your Python code and output in the code box below.*)

Answer:

Code:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.metrics import mean_squared_error

# --- Load dataset ---
housing = fetch_california_housing()
X, y = housing.data, housing.target

# --- Split into train and test sets ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# --- Scale features (optional for tree models, but keeps consistency) ---
```

```
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# --- Bagging Regressor with DecisionTree base estimator ---
bag_reg = BaggingRegressor(
    estimator=DecisionTreeRegressor(random_state=42),
    n_estimators=50,
    random_state=42,
    n_jobs=-1
)
bag_reg.fit(X_train_s, y_train)
y_pred_bag = bag_reg.predict(X_test_s)

# --- Random Forest Regressor ---
rf_reg = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
    n_jobs=-1
)
rf_reg.fit(X_train_s, y_train)
y_pred_rf = rf_reg.predict(X_test_s)

# --- Compare Mean Squared Errors ---
mse_bag = mean_squared_error(y_test, y_pred_bag)
mse_rf = mean_squared_error(y_test, y_pred_rf)

print(f"Bagging Regressor MSE: {mse_bag:.4f}")
print(f"Random Forest Regressor MSE: {mse_rf:.4f}")
```

Output:

```
Bagging Regressor MSE: 0.2570
Random Forest Regressor MSE: 0.2552
```

Question 10: You are working as a data scientist at a financial institution to predict loan default. You have access to customer demographic and transaction history data.

You decide to use ensemble techniques to increase model performance.

Explain your step-by-step approach to:

- Choose between Bagging or Boosting
- Handle overfitting
- Select base models
- Evaluate performance using cross-validation
- Justify how ensemble learning improves decision-making in this real-world context.

(*Include your Python code and output in the code box below.*)
Answer:

Step 1 — Understanding the problem

We need to build a loan-default prediction model.
The goal is to predict whether a customer will default, using demographic and transaction data.
Since financial data can be noisy, unbalanced, and non-linear, ensemble learning can help produce a more accurate and stable model.

Step 2 — Choosing between Bagging and Boosting

Technique	When to choose	Reason
Bagging (e.g., Random Forest)	If the base learner (Decision Tree) is high variance and data has noise.	Bagging reduces variance and gives stable, interpretable predictions.
Boosting (e.g., XGBoost, AdaBoost, Gradient Boosting)	If the model underfits or we want higher predictive power on complex relationships.	Boosting reduces bias by focusing on hard-to-predict cases.

Step 3 — Handling overfitting

To prevent the ensemble from memorizing the training data:

- Limit tree **depth** (`max_depth`), set `min_samples_leaf`, or use smaller `learning_rate` in boosting.
- Use **cross-validation** and **early stopping** (for boosting).
- Apply **regularization** (e.g., `subsample`, `colsample_bytree` in XGBoost).
- Perform **feature selection** or drop highly correlated features.
- Use **OOB score** (for bagging) as a quick overfitting indicator.

Step 4 — Selecting base models

- Base learners: **Decision Tree**, **Logistic Regression**, or **LightGBM** trees.
- Ensemble choice:
  - **Random Forest:** many trees trained on random feature subsets (reduces correlation).
  - **Gradient Boosting / XGBoost:** sequentially builds small trees that fix previous errors.
- If computational resources permit, try **Stacking**: combine Random Forest, XGBoost, and Logistic Regression with a meta-learner.

Step 5 — Evaluating performance using Cross-Validation

- Use **Stratified K-Fold Cross-Validation (k = 5 or 10)** to preserve class ratios.
- Primary metrics: **AUC-ROC**, **Precision**, **Recall**, and **F1-score**.

- Compare base models and ensembles; choose the one with the best mean CV score and lowest standard deviation.
- Optionally use a **hold-out test set** or **temporal validation** (train on older data, test on recent loans).

Step 6 — How Ensemble Learning improves real-world decision-making

- **Higher accuracy = lower financial risk:** Fewer misclassified defaulters → fewer bad loans.
- **Reduced variance:** Model more stable across time and datasets.
- **Better generalization:** Combines diverse patterns in customer behaviour.
- **Explain ability:** Tree-based ensembles allow feature importance and SHAP analysis for regulatory transparency.
- **Trust & compliance:** Banks can justify decisions using interpretable feature impact charts.