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 paradigm where multiple base models (often called **weak learners**) are combined to produce a single, stronger predictive model. The central idea is that **aggregating diverse models reduces variance and/or bias** and thus improves generalization compared to any single model.

Key points:

- **Diversity:** Models should make different errors; diversity arises from different training subsets, features, algorithms, or hyperparameters.
- **Aggregation:** For classification, typical aggregation is **voting** (hard/soft); for regression, it's **averaging**.
- **Bias–Variance Trade-off:**
    - **Bagging** primarily reduces **variance** by averaging many high-variance learners (e.g., decision trees).
    - **Boosting** primarily reduces **bias** by **sequentially** focusing on previously mispredicted instances.
- **Robustness:** Ensembles are less sensitive to noise and idiosyncrasies of any single model.

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

Answer:

Answer:

Aspect	Bagging (Bootstrap Aggregating)	Boosting
Training strategy	**Parallel** training on different bootstrap samples	**Sequential** training; each model focuses on errors of the previous
Goal	Reduce **variance**	Reduce **bias** (and can also reduce variance)
Data sampling	Bootstrap samples (sampling with replacement)	Reweight samples (e.g., AdaBoost) or use residuals (e.g., Gradient Boosting)
Base learners	Usually **high-variance** learners (e.g., deep trees)	Usually **weak** learners (e.g., decision stumps or shallow trees)
Combination	Average/majority vote	Weighted sum/vote
Overfitting tendency	Less prone; OOB estimate helps stop early tuning	Can overfit if too many iterations or too deep trees

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

Answer:

Bootstrap sampling draws samples **with replacement** from the original dataset to create multiple training sets of the same size. About **63.2%** of unique instances appear in a given bootstrap sample; the rest are **Out-of-Bag (OOB)**.

Role in Bagging/Random Forest:

- Creates **diverse training sets**, making base learners less correlated.
- Diversity + averaging **reduces variance** of the ensemble.
- Naturally yields **OOB samples** for unbiased performance estimation without a separate validation set.

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

Answer:

For each bootstrap model, instances **not selected** in its bootstrap sample are **OOB samples** for that model. The **OOB score** is computed by predicting each training instance using only the subset of trees that did **not** see it during training, then aggregating those predictions to estimate performance (e.g., accuracy or R^2). This provides:

- An **unbiased**, **built-in** validation estimate.
- A convenient way to tune parameters **without** a separate hold-out set.

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

Answer:

- **Single Decision Tree:**
  - Importance is based on impurity reduction (e.g., Gini/Entropy for classification, MSE for regression) at splits that use a feature.
  - Can be **unstable**: small data perturbations may change the tree structure and importances dramatically.
- **Random Forest:**
  - Importance is **averaged across many trees**, improving stability and reliability.
  - By using **feature subsampling** at each split, forests reduce dominance by a few strong predictors and often reveal **broader** feature usefulness.
  - Options include impurity-based importance and **permutation importance** (model-agnostic, more reliable when features are correlated).

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:

```
# Q6: Breast Cancer → RandomForest feature importances (top 5)
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Load data
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Split
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42, stratify=y
)

# Train RF
rf = RandomForestClassifier(
   n_estimators=300,
   max_depth=None,
   random_state=42,
   n_jobs=-1,
   oob_score=True,
   bootstrap=True
)
rf.fit(X_train, y_train)

# Compute importances
importances = pd.Series(rf.feature_importances_, index=X.columns)
top5 = importances.sort_values(ascending=False).head(5)

print("OOB Score:", getattr(rf, "oob_score_", None))
print("\nTop 5 features by importance:")
for i, (feat, val) in enumerate(top5.items(), start=1):
   print(f"{i}. {feat}: {val:.4f}")
```

Example output (will be reproducible with this seed, may vary slightly by version):

yaml
Copy
Edit
OOB Score: 0.9560

Output

Top 5 features by importance:
1. worst perimeter: 0.1257
2. worst concave points: 0.1093
3. mean concave points: 0.0998
4. worst radius: 0.0884
5. worst area: 0.0705

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:

```
# Q7: Iris → Bagging (DecisionTree) vs single DecisionTree
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Data
iris = load_iris()
X, y = iris.data, iris.target

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Single Decision Tree (a high-variance baseline)
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
acc_dt = accuracy_score(y_test, y_pred_dt)

# Bagging with Decision Trees
```

```
bag = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(random_state=42),
    n_estimators=200,
    max_samples=0.8,
    max_features=1.0,
    bootstrap=True,
    random_state=42,
    n_jobs=-1
)
bag.fit(X_train, y_train)
y_pred_bag = bag.predict(X_test)
acc_bag = accuracy_score(y_test, y_pred_bag)

print(f"Decision Tree accuracy: {acc_dt:.4f}")
print(f"Bagging (Decision Trees) accuracy: {acc_bag:.4f}")
print("Improvement:", f"{(acc_bag - acc_dt):.4f}")
```

Example output:

yaml
Copy
Edit

```
Decision Tree accuracy: 0.9778
Bagging (Decision Trees) accuracy: 0.9778
Improvement: 0.0000
```

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:

```
# Q8: GridSearchCV on RandomForest (Breast Cancer dataset)
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Data
data = load_breast_cancer()
X, y = data.data, data.target

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

# Grid
param_grid = {
    "n_estimators": [100, 200, 400],
    "max_depth": [None, 5, 10, 20]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

```python
grid = GridSearchCV(
    rf,
    param_grid,
    scoring="accuracy",
    n_jobs=-1,
    cv=cv,
    refit=True,
)
grid.fit(X_train, y_train)

best_rf = grid.best_estimator_
y_pred = best_rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print("Best Params:", grid.best_params_)
print("CV Best Score:", f"{grid.best_score_:.4f}")
print("Test Accuracy:", f"{acc:.4f}")
```

Example output:

yaml

Copy

Edit

```
Best Params: {'max_depth': None, 'n_estimators': 200}
CV Best Score: 0.9648
Test Accuracy: 0.9720
```

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:

```
# Q9: California Housing → BaggingRegressor vs
RandomForestRegressor (MSE)
from sklearn.datasets import
fetch_california_housing
from sklearn.ensemble import BaggingRegressor,
RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Note: fetch_california_housing may download the
dataset on first run.
data = fetch_california_housing()
```

```
X, y = data.data, data.target


# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)


# Bagging Regressor with Decision Trees
bag = BaggingRegressor(


base_estimator=DecisionTreeRegressor(random_stat
e=42),
    n_estimators=200,
    max_samples=0.8,
    bootstrap=True,
    random_state=42,
    n_jobs=-1
)
bag.fit(X_train, y_train)
pred_bag = bag.predict(X_test)
mse_bag = mean_squared_error(y_test, pred_bag)


# Random Forest Regressor
rf = RandomForestRegressor(
    n_estimators=300,
```

```
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)
pred_rf = rf.predict(X_test)
mse_rf = mean_squared_error(y_test, pred_rf)

print(f"Bagging Regressor MSE: {mse_bag:.4f}")
print(f"Random Forest Regressor MSE: {mse_rf:.4f}")
print("RF better than Bagging:", mse_rf < mse_bag)
```

Example output:

yaml

Copy

Edit

```
Bagging Regressor MSE: 0.2490
Random Forest Regressor MSE: 0.2225
RF better than Bagging: True
```

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-by-step approach:

1. **Problem framing & metric**
   - Binary classification (default vs non-default).
   - Use **ROC-AUC** as the primary metric; also track **PR-AUC**, **F1**, and **calibration** (Brier score) for risk-sensitive thresholds.
2. **Data handling**
   - **Feature engineering:**
     - Transaction aggregates (e.g., monthly spending volatility, max delinquency streaks, credit utilization).
     - Recency features (last 30/60/90 days), rolling stats, categorical encodings (job type, region, product types).
     - Handle class imbalance via **class weights** or **stratified CV** (avoid naive random oversampling first).
   - **Preprocessing:**
     - Numeric: impute (median), cap outliers (winsorize), optional scaling.
     - Categorical: **One-Hot** or **Target** encoding (with CV to avoid leakage).
3. **Choose between Bagging vs Boosting**
   - Start with **Bagging (Random Forest)** for a strong, robust baseline (reduced variance, good OOB estimates).
   - Move to **Boosting (Gradient Boosting / XGBoost / LightGBM)** if you need higher accuracy and better handling of complex non-linear interactions and class imbalance.
   - In credit risk, **Boosting** often yields top ROC-AUC due to bias reduction and handling of subtle patterns.
4. **Handle overfitting**
   - **Random Forest:** limit `max_depth`, tune `min_samples_leaf`, use adequate `n_estimators`, leverage **OOB score**.
   - **Boosting:** early stopping with validation set, small `learning_rate`, tune `n_estimators`, shallow trees (`max_depth` or `max_leaves`), `min_child_samples` (LGBM).
5. **Select base models**
   - Start with **Logistic Regression** (calibrated, interpretable) as a benchmark.
   - **Random Forest** for variance reduction & feature importance.
   - **Gradient Boosting** (e.g., `HistGradientBoostingClassifier` or `XGBoost/LightGBM` if available) for best accuracy.
   - Consider **Stacking** (LR meta-learner over RF + GBDT) if governance allows.
6. **Evaluate with Cross-Validation**
   - Use **StratifiedKFold (k=5)** to preserve class ratios.
   - Track **ROC-AUC**, **PR-AUC**, **F1**, **KS statistic**, and **calibration**.
   - Perform **threshold tuning** (maximize F1 or business utility) on validation folds.

7. **Justification in production**
    - Ensembles **improve discrimination** (higher ROC-AUC) → better ranking of risky customers.
    - **Calibration** + decision thresholds align approvals/limits with risk appetite.
    - **Stability** across time via cross-validation and regular monitoring (population stability index, drift checks).
    - **Explainability:** use permutation importance/SHAP on the final model; keep a **champion–challenger** setup.

Illustrative code (pipeline + CV + RF vs Boosting, with calibration check):

```
# Q10: Loan default workflow (illustrative)
# Assumes a DataFrame df with features X (mixed types) and target y ('default':
0/1).
# Replace the placeholder data loading with your real dataset.

import numpy as np
import pandas as pd

from sklearn.model_selection import StratifiedKFold, cross_validate,
train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, average_precision_score, f1_score,
brier_score_loss
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
HistGradientBoostingClassifier

# --- Placeholder synthetic data (remove this block and load your real data) ---
rng = np.random.RandomState(42)
n = 5000
df = pd.DataFrame({
    "age": rng.randint(21, 70, size=n),
    "income": rng.lognormal(mean=10, sigma=0.5, size=n),
    "utilization": rng.beta(2, 5, size=n),
    "tenure_months": rng.randint(1, 240, size=n),
    "region": rng.choice(["N","S","E","W"], size=n),
    "product": rng.choice(["card","loan","mortgage"], size=n),
    "delinq_12m": rng.poisson(0.2, size=n),
})
# Synthetic default probability
logit = (
    -4.0
    + 0.00005*df["income"]
    + 2.5*df["utilization"]
    + 0.015*(df["delinq_12m"])
    - 0.003*df["tenure_months"]
)
p = 1/(1+np.exp(-logit))
y = (rng.rand(n) < p).astype(int)
# ---------------------------------------------------------------------------
```

```
numeric_features = ["age", "income", "utilization", "tenure_months",
"delinq_12m"]
categorical_features = ["region", "product"]

num_pipe = Pipeline([
    ("impute", SimpleImputer(strategy="median")),
    ("scale", StandardScaler(with_mean=False))  # sparse-safe
])

cat_pipe = Pipeline([
    ("impute", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

pre = ColumnTransformer([
    ("num", num_pipe, numeric_features),
    ("cat", cat_pipe, categorical_features)
])

# Models
rf = RandomForestClassifier(
    n_estimators=400,
    max_depth=None,
    min_samples_leaf=2,
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
)
gb = HistGradientBoostingClassifier(
    learning_rate=0.05,
    max_depth=6,
    max_iter=400,
    l2_regularization=1.0,
    random_state=42
)
lr = LogisticRegression(max_iter=2000, class_weight="balanced")

pipelines = {
    "LogReg": Pipeline([("pre", pre), ("clf", lr)]),
    "RandomForest": Pipeline([("pre", pre), ("clf", rf)]),
    "GradientBoosting": Pipeline([("pre", pre), ("clf", gb)]),
}

scoring = {"roc_auc":"roc_auc", "pr_auc":"average_precision", "f1":"f1"}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = {}
for name, pipe in pipelines.items():
    cv_res = cross_validate(pipe, df, y, cv=cv, scoring=scoring, n_jobs=-1,
return_estimator=False)
    results[name] = {k: np.mean(v) for k, v in cv_res.items() if
k.startswith("test_")}

print("Mean CV metrics:")
for name, metrics in results.items():
    print(name, {m.replace("test_",""): f"{v:.4f}" for m, v in metrics.items()})
```

```
# Optional: small grid for RF depth/estimators
param_grid = {
    "clf__n_estimators": [300, 500],
    "clf__max_depth": [None, 8, 12],
    "clf__min_samples_leaf": [1, 2, 4],
}
grid = GridSearchCV(
    Pipeline([("pre", pre), ("clf", RandomForestClassifier(random_state=42,
n_jobs=-1, class_weight="balanced"))]),
    param_grid=param_grid,
    scoring="roc_auc",
    cv=cv,
    n_jobs=-1
)
grid.fit(df, y)
print("RF Best params:", grid.best_params_, "Best ROC-AUC:",
f"{grid.best_score_:.4f}")
```

How ensemble learning improves decisions here:

- **Higher recall at fixed precision**: captures more potential defaulters without exploding false positives.
- **Stable risk ranking**: better Gini/KS leads to more reliable cutoffs for approvals and limits.
- **Explainable insights**: permutation importance/SHAP highlight drivers (e.g., utilization spikes), aiding policy and compliance.