Assignment Code: DA-AG-015

Boosting Techniques | **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 Boosting in Machine Learning? Explain how it improves weak learners.

Answer:

Answer (20 marks):
Boosting is an ensemble method that builds a **strong predictor by adding many weak learners sequentially**. Each new learner focuses on the **mistakes** (residuals or hard examples) left by the current ensemble. After TTT rounds, the final model is an **additive model**:

FT(x)=∑t=1Tν·ft(x)F_T(x)=\sum_{t=1}^T \nu\cdot f_t(x)FT(x)=t=1∑Tν·ft(x)

where ftf_tft is a weak learner (often a shallow tree) and ν∈(0,1]\nu\in(0,1]ν∈(0,1] is the **learning rate**.

How it improves weak learners

- **Focus on hard cases:** At round ttt, the algorithm reweights samples (AdaBoost) or fits residuals/gradients (Gradient Boosting) so the new learner concentrates on what's still wrong.

- **Stage-wise optimization of a loss:** Gradient Boosting fits ftf_tft to the **negative gradient** of the loss, giving a principled direction of improvement.
- **Bias reduction:** Shallow trees have high bias; adding them stage-wise reduces bias while keeping variance controlled via shrinkage, subsampling, and early stopping.
- **Margins and robustness:** By increasing classification margins (esp. AdaBoost), ensembles generalize better even when training error is tiny.

Key ingredients: **sequential training**, **weak learners**, **loss-guided focus**, **shrinkage (learning rate)**, **regularization** (depth, subsampling, L1/L2), and **early stopping**.

Question 2: What is the difference between AdaBoost and Gradient Boosting in terms of how models are trained?

Answer:

Answer (20 marks):
Boosting is an ensemble method that builds a **strong predictor by adding many weak learners sequentially**. Each new learner focuses on the **mistakes** (residuals or hard examples) left by the current ensemble. After TTT rounds, the final model is an **additive model**:

FT(x)=∑t=1Tν·ft(x)F_T(x)=\sum_{t=1}^T \nu\cdot f_t(x)FT(x)=t=1∑Tν·ft(x) where ftf_tft is a weak learner (often a shallow tree) and ν∈(0,1]\nu\in(0,1]ν∈(0,1] is the **learning rate**.

How it improves weak learners

- **Focus on hard cases:** At round ttt, the algorithm reweights samples (AdaBoost) or fits residuals/gradients (Gradient Boosting) so the new learner concentrates on what’s still wrong.
- **Stage-wise optimization of a loss:** Gradient Boosting fits ftf_tft to the **negative gradient** of the loss, giving a principled direction of improvement.
- **Bias reduction:** Shallow trees have high bias; adding them stage-wise reduces bias while keeping variance controlled via shrinkage, subsampling, and early stopping.
- **Margins and robustness:** By increasing classification margins (esp. AdaBoost), ensembles generalize better even when training error is tiny.

Key ingredients: **sequential training**, **weak learners**, **loss-guided focus**, **shrinkage (learning rate)**, **regularization** (depth, subsampling, L1/L2), and **early stopping**.

Question 3: How does regularization help in XGBoost?

Answer:

XGBoost regularizes **tree complexity** and **weights** to prevent overfitting and improve generalization:

1. **Explicit L1/L2 penalty on leaf weights**
   Objective:

   obj=∑il(yi,y^i)+∑t=1T(Ω(ft)),Ω(f)=γ·#leaves+λ2∑wj2+α∑|wj|\text{obj} = \sum_i l(y_i,\hat{y}_i) + \sum_{t=1}^T \Big(\Omega(f_t)\Big),\quad \Omega(f)=\gamma\cdot \#\text{leaves} + \tfrac{\lambda}{2}\sum w_j^2 + \alpha \sum |w_j|obj=i∑l(yi,y^i)+t=1∑T(Ω(ft)),Ω(f)=γ·#leaves+2λ∑wj2+α∑|wj|

   - λ\lambdaλ: L2, smooths weights; α\alphaα: L1, drives sparsity; γ\gammaγ: min loss reduction to make a split.
2. **Tree-structure constraints**
   - `max_depth`, `max_leaves`, `min_child_weight` (min Hessian/weight in a leaf), `gamma` (min split gain) limit complexity.
3. **Stochastic regularization**
   - `subsample` (rows) and `colsample_bytree/bylevel/by_node` (features) reduce correlation and variance.
4. **Learning rate (shrinkage) + n_estimators**
   - Small `eta` with more trees improves generalization (stage-wise small steps).
5. **Early stopping**
   - Stop when validation metric doesn't improve—prevents overfit.
6. **Monotonic constraints / interaction constraints** (when used)
   - Encourage plausible shapes, reduce spurious fits.

Together, these keep trees **simple**, **diverse**, and **well-calibrated**.

Question 4: Why is CatBoost considered efficient for handling categorical data?

Answer:

CatBoost is designed for **categorical-rich tabular data**. It avoids heavy one-hot encoding and reduces target leakage:

1. **Ordered target statistics (ordered CTRs)**
   - Converts categories to numeric statistics (e.g., target mean) using **permutation-driven, out-of-fold** estimates to **avoid leakage** and **prediction shift**.
2. **Ordered Boosting**
   - Builds trees in an order that uses only past information, improving generalization with categoricals.
3. **High cardinality support**
   - Efficient CTR schemes and hash-based handling make millions of categories feasible.
4. **Built-in handling of missing values** and categorical interactions (automatic feature combinations).
5. **Symmetric/oblivious trees**
   - Balanced structure, fast scoring, good regularization on tabular data.

Result: **less preprocessing**, **strong accuracy**, and **stable training** on mixed numeric/categorical datasets.

Question 5: What are some real-world applications where boosting techniques are preferred over bagging methods?

Answer:

Boosting (GBDTs like XGBoost/LightGBM/CatBoost) often outperforms bagging on **structured/tabular** problems with complex patterns, mild–moderate noise, or imbalance:

- **Credit risk & default prediction**: finer separation near decision boundary; better ROC-AUC/KS.
- **Fraud/anomaly detection** in payments/insurance: focuses on hard/rare cases with class-weighting.
- **Click-through rate (CTR) & ranking**: web ads/recs; gradient boosting dominates classic leaderboard tasks.
- **Churn propensity** in telecom/SaaS: captures nonlinear effects and interactions.
- **Medical diagnostics** (tabular labs/claims): robust bias reduction with calibrated probabilities.
- **Demand/price modeling** in retail: handles mixed features + interactions better than bagging.

Why: **bias reduction**, **loss-driven learning**, **rich regularization**, **feature interaction discovery**, and **strong performance with limited preprocessing**.

Datasets:

- Use `sklearn.datasets.load_breast_cancer()` for classification tasks.
- Use `sklearn.datasets.fetch_california_housing()` for regression tasks.

Question 6: Write a Python program to:

- Train an AdaBoost Classifier on the Breast Cancer dataset
- Print the model accuracy

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

Answer:

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

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

# Model
clf = AdaBoostClassifier(
    n_estimators=200,          # more weak learners
    learning_rate=0.5,         # shrinkage
    random_state=42
)
clf.fit(X_train, y_train)
```

```
# Evaluation
y_pred = clf.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
```

Accuracy typically **0.95–0.98** with these settings.

Question 7: Write a Python program to:

- Train a Gradient Boosting Regressor on the California Housing dataset
- Evaluate performance using R-squared score

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

Answer:

```
from sklearn.datasets import fetch_california_housing

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model_selection import train_test_split
```

```python
from sklearn.metrics import r2_score


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


# Model (tuned conservative)
gbr = GradientBoostingRegressor(
    n_estimators=400,
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    random_state=42
)
gbr.fit(X_train, y_train)
```

```
# Evaluation

y_pred = gbr.predict(X_test)

print("R²:", r2_score(y_test, y_pred))
```

R^2 commonly **0.78–0.83** depending on version/hardware.

Question 8: Write a Python program to:

- Train an XGBoost Classifier on the Breast Cancer dataset
- Tune the learning rate using GridSearchCV
- Print the best parameters and accuracy

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

Answer:

```
# If xgboost is not installed in your environment, install it first:
# pip install xgboost

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

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

# Base model
xgb = XGBClassifier(
    n_estimators=400,
    max_depth=4,
    subsample=0.9,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    reg_alpha=0.0,
    objective="binary:logistic",
    eval_metric="logloss",
    random_state=42,
    n_jobs=-1
)

# Tune learning rate
param_grid = {"learning_rate": [0.01, 0.05, 0.1, 0.2]}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

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

best = grid.best_estimator_
y_pred = best.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred)

print("Best Params:", grid.best_params_)
print("Test Accuracy:", acc)
```

Best `learning_rate` often **0.05–0.1**, accuracy ~**0.97–0.99**.

Question 9: Write a Python program to:

- Train a CatBoost Classifier
- Plot the confusion matrix using `seaborn`

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

Answer:

```
# If catboost is not installed: pip install catboost

# seaborn for plotting



from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix,

classification_report

from catboost import CatBoostClassifier

import seaborn as sns
```

```
import matplotlib.pyplot as plt


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


# Model (CatBoost handles missing/categorical; here all
numeric but still fine)
model = CatBoostClassifier(
    iterations=500,
    depth=6,
    learning_rate=0.05,
    loss_function="Logloss",
    verbose=0,
    random_seed=42
)
model.fit(X_train, y_train, eval_set=(X_test, y_test),
verbose=0)


# Predictions
y_pred = model.predict(X_test)
```

```
# Confusion Matrix

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt="d")

plt.title("Confusion Matrix - CatBoost (Breast Cancer)")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()


print(classification_report(y_test, y_pred))
```

Question 10: You're working for a FinTech company trying to predict loan default using customer demographics and transaction behavior.
The dataset is imbalanced, contains missing values, and has both numeric and categorical features.

Describe your step-by-step data science pipeline using boosting techniques:

- Data preprocessing & handling missing/categorical values
- Choice between AdaBoost, XGBoost, or CatBoost
- Hyperparameter tuning strategy
- Evaluation metrics you'd choose and why
- How the business would benefit from your model

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

Answer:

Problem: Imbalanced binary classification (default vs. non-default) with missing values and mixed feature types.

1) Data preprocessing

- **Target & leakage check:** Remove post-outcome features (e.g., collections flags after default).
- **Train/validation/test split** with **stratification** to preserve default rate.
- **Imbalance:** Prefer **class weights** or **balanced sampling** (avoid naive oversample; try SMOTE only on train).
- **Missing values:**
  - For **XGBoost/AdaBoost**: impute numeric (median), encode categoricals (One-Hot/Target encoding).
  - For **CatBoost**: pass categorical column indices; CatBoost handles missing + encoding via **ordered statistics** (less leakage).
- **Feature engineering:** rolling transaction stats (spend volatility, delinquency streaks), utilization, recency flags, ratios (debt/income), bureau summaries.
- **Outliers:** cap/winsorize heavy-tailed monetary features.

2) Algorithm choice

- **Start with CatBoost** if many categoricals and missing values → minimal prep, strong baseline.
- **XGBoost** for large-scale performance, rich regularization and speed (GPU optional).
- **AdaBoost** when wanting a simple baseline on clean features; less typical for large, messy tabular.

3) Hyperparameter tuning

- **Search space:**
  - CatBoost: `depth (4-10)`, `iterations (500-2000)`, `learning_rate (0.02-0.2)`, `l2_leaf_reg`, `subsample`.
  - XGBoost: `learning_rate`, `max_depth`/`max_leaves`, `min_child_weight`, `subsample`, `colsample_bytree`, `reg_lambda`, `reg_alpha`, `n_estimators`, `gamma`.
- **Strategy: StratifiedKFold(5)** + **early stopping** on a validation set; start with **RandomizedSearch** then refine with **GridSearch** around the best region.

4) Evaluation metrics (why)

- Primary: **ROC-AUC** (ranking quality).
- For imbalanced defaults: **PR-AUC**, **Recall@fixed-Precision** (e.g., 90%), **KS statistic**, **F1** at business threshold.
- **Calibration** (Brier score, reliability curve) for probability-based decisioning (limits, pricing).
- **Cost-sensitive** evaluation when available (expected loss).

5) Business benefits

- **Lower default losses** via better recall at reasonable precision.
- **Approve more good loans** (profit uplift) with calibrated scores and policy cutoffs.
- **Explainability** (SHAP/permutation importance) supports governance and adverse-action reasons.
- **Stability monitoring** (drift/PSI) keeps the model healthy over time.

Illustrative code (CatBoost end-to-end skeleton):

```
# pip install catboost shap

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import roc_auc_score, average_precision_score,
brier_score_loss
from catboost import CatBoostClassifier, Pool

# Suppose df is your dataset with target column 'default' (0/1)
# and mixed feature types; identify categorical columns:
# df = pd.read_csv("loans.csv")
# Example:
# categorical_cols = ["state", "segment", "employment_type", "product"]
# numeric_cols = [c for c in df.columns if c not in categorical_cols +
["default"]]

# --- demo placeholder (replace with real df) ---
rng = np.random.RandomState(42)
n = 8000
df = pd.DataFrame({
    "income": rng.lognormal(10, 0.6, n),
```

```
    "utilization": rng.beta(2,5, n),
    "tenure_m": rng.randint(1, 240, n),
    "state": rng.choice(list("ABCDE"), n),
    "segment": rng.choice(["retail","salaried","self_emp"], n),
    "product": rng.choice(["cc","pl","gold"], n),
})
logit = -4.2 + 0.00004*df["income"] + 2.6*df["utilization"] +
0.012*df["tenure_m"]
p = 1/(1+np.exp(-logit))
df["default"] = (rng.rand(n) < p).astype(int)
categorical_cols = ["state","segment","product"]
numeric_cols = ["income","utilization","tenure_m"]
# -----------------------------------------------

X = df[categorical_cols + numeric_cols]
y = df["default"]

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

cat_idx = [X.columns.get_loc(c) for c in categorical_cols]

train_pool = Pool(X_train, y_train, cat_features=cat_idx)
test_pool  = Pool(X_test,  y_test,  cat_features=cat_idx)

model = CatBoostClassifier(
    iterations=2000,
    learning_rate=0.05,
    depth=8,
    l2_leaf_reg=3.0,
    loss_function="Logloss",
    eval_metric="AUC",
    random_seed=42,
    subsample=0.8,
    od_type="Iter",             # early stopping
    od_wait=100,
    verbose=False
)

model.fit(train_pool, eval_set=test_pool, use_best_model=True)

# Metrics
proba = model.predict_proba(test_pool)[:,1]
roc = roc_auc_score(y_test, proba)
pr  = average_precision_score(y_test, proba)
brier = brier_score_loss(y_test, proba)

print({"ROC_AUC": round(roc,4), "PR_AUC": round(pr,4), "Brier": round(brier,4)})

# Thresholding example (pick threshold by business need)
threshold = 0.3
pred = (proba >= threshold).astype(int)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred, digits=4))
```

This template includes **categorical handling**, **early stopping**, and **calibrated probability evaluation**. Replace the placeholder `df` with your real data and set `categorical_cols` accordingly.

•