Assignment Code: DA-AG-0010

Bagging & Boosting KNN & Stacking

Assignment

Total Marks: 200

Question 1 : What is the fundamental idea behind ensemble techniques? How does bagging differ from boosting in terms of approach and objective?

Answer:

Fundamental Idea:
Ensemble learning combines multiple weak or base models to produce a stronger, more robust model. The idea is that a group of diverse models can collectively reduce errors due to bias, variance, or noise.

Bagging (Bootstrap Aggregating):

- Builds multiple independent models (e.g., Decision Trees) on *different random subsets* of the training data (sampled with replacement).
- Each model votes (classification) or averages (regression) its prediction.
- Objective: **Reduce variance** and prevent overfitting.
- Example: **Random Forest**.

Boosting:

- Builds models sequentially. Each new model focuses on correcting the mistakes made by the previous ones.
- Objective: **Reduce bias** by combining weak learners to create a strong one.
- Example: **AdaBoost, Gradient Boosting, XGBoost**.

Question 2: Explain how the Random Forest Classifier reduces overfitting compared to a single decision tree. Mention the role of two key hyperparameters in this process.

Answer:

Random Forest reduces overfitting through:

1. **Bootstrap Sampling:** Each tree is trained on a random subset of data, ensuring diversity among trees.
2. **Feature Randomness:** At each split, a random subset of features is considered, making trees less correlated.

Key Hyperparameters:

- n_estimators: Number of trees in the forest. More trees → stable, less variance.
- max_features: Number of features considered at each split. Controls model diversity.

Together, these mechanisms prevent the model from memorizing the training data.

Question 3: What is Stacking in ensemble learning? How does it differ from traditional bagging/boosting methods? Provide a simple example use case.

Answer:

Stacking (Stacked Generalization):

Stacking combines multiple base models (level-0 learners) and uses another model (meta-learner) to learn how to best combine their predictions.

Difference:

- Bagging/Boosting → combine same type of models (homogeneous ensemble).
- Stacking → combines *different* types (heterogeneous ensemble), e.g., SVM + Decision Tree + KNN.

Example Use Case:

In a medical diagnosis system:

- Base models: Logistic Regression, Random Forest, SVM.
- Meta-model: XGBoost combining predictions for final decision.

Question 4:What is the OOB Score in Random Forest, and why is it useful? How does it help in model evaluation without a separate validation set?

Answer:

Out-of-Bag (OOB) Score:
During training, each tree uses a random bootstrap sample. About 1/3 of samples are left out (OOB samples).
These are used as a *validation set* for that tree.
Benefits:

- Provides an unbiased estimate of model accuracy without needing a separate validation set.
- Saves computation and prevents data wastage.

Question 5: Compare AdaBoost and Gradient Boosting in terms of:

- How they handle errors from weak learners
- Weight adjustment mechanism
- Typical use cases **Answer:**

Feature	AdaBoost	Gradient Boosting
Error Handling	Focuses on misclassified samples	Fits new learners to minimize loss function's gradient
Weight Adjustment	Increases weights of misclassified samples	Reduces residual errors iteratively
Weak Learner	Usually Decision Stumps	Any differentiable model
Use Cases	Binary classification, Spam detection	Regression, complex datasets
Loss Function	Exponential	Any differentiable loss (e.g., MSE, log-loss)

Question 6:Why does CatBoost perform well on categorical features without requiring extensive preprocessing? Briefly explain its handling of categorical variables.

Answer:

CatBoost uses **ordered target statistics** to handle categorical features efficiently.
Instead of label encoding or one-hot encoding, it:

1. Converts categorical variables into numeric values using target mean encoding.
2. Prevents target leakage by computing encodings in an *ordered* way (only using previous data points).
3. Reduces overfitting with Bayesian averaging.

Hence, CatBoost requires minimal preprocessing and is ideal for mixed data types.

Question 7: KNN Classifier Assignment: Wine Dataset Analysis with Optimization

Task:

1. Load the Wine dataset (sklearn.datasets.load_wine()).
2. Split data into 70% train and 30% test.
3. Train a KNN classifier (default K=5) without scaling and evaluate using:
    a. Accuracy
    b. Precision, Recall, F1-Score (print classification report)
4. Apply StandardScaler, retrain KNN, and compare metrics.
5. Use GridSearchCV to find the best K (test K=1 to 20) and distance metric (Euclidean, Manhattan).
6. Train the optimized KNN and compare results with the unscaled/scaled versions.

Answer:

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score

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

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

# 1. Without Scaling
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Without Scaling Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# 2. With StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_scaled = KNeighborsClassifier(n_neighbors=5)
knn_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = knn_scaled.predict(X_test_scaled)
print("With Scaling Accuracy:", accuracy_score(y_test, y_pred_scaled))
print(classification_report(y_test, y_pred_scaled))

# 3. GridSearchCV Optimization
param_grid = {
    'n_neighbors': list(range(1, 21)),
    'metric': ['euclidean', 'manhattan']
}
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid.fit(X_train_scaled, y_train)

print("Best Params:", grid.best_params_)
best_knn = grid.best_estimator_
y_pred_best = best_knn.predict(X_test_scaled)
print("Optimized Accuracy:", accuracy_score(y_test, y_pred_best))
```

Observation:

- Scaling improves performance drastically.
- Optimal K often lies between 3–7 for this dataset.

Question 8 : PCA + KNN with Variance Analysis and Visualization

Task:

1. Load the Breast Cancer dataset (sklearn.datasets.load_breast_cancer()).
2. Apply PCA and plot the scree plot (explained variance ratio).
3. Retain 95% variance and transform the dataset.
4. Train KNN on the original data and PCA-transformed data, then compare accuracy.
5. Visualize the first two principal components using a scatter plot (color by class).

Answer:

```
 from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

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

# PCA
pca = PCA().fit(X)
plt.plot(range(1, len(pca.explained_variance_ratio_)+1), pca.explained_variance_ratio_,
marker='o')
plt.title("Scree Plot - Explained Variance")
plt.xlabel("Principal Component")
plt.ylabel("Variance Ratio")
plt.show()

# Retain 95% variance
pca_95 = PCA(0.95)
X_pca = pca_95.fit_transform(X)

# Compare KNN on original vs PCA
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
knn = KNeighborsClassifier(5).fit(X_train, y_train)
knn_pca = KNeighborsClassifier(5).fit(X_pca[:398], y[:398])

print("Original Accuracy:", accuracy_score(y_test, knn.predict(X_test)))
print("PCA Accuracy:", accuracy_score(y_test, knn_pca.predict(pca_95.transform(X_test))))
```

```
# Visualization
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='coolwarm')
plt.title("First Two Principal Components")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Question 9:KNN Regressor with Distance Metrics and K-Value Analysis

Task:

1. Generate a synthetic regression dataset (sklearn.datasets.make_regression(n_samples=500, n_features=10)).
2. Train a KNN regressor with:
    a. Euclidean distance (K=5)
    b. Manhattan distance (K=5)
    c. Compare Mean Squared Error (MSE) for both.
3. Test K=1, 5, 10, 20, 50 and plot K vs. MSE to analyze bias-variance tradeoff.

Answer:

```
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

X, y = make_regression(n_samples=500, n_features=10, noise=10,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

mse_results = {}

for metric in ['euclidean', 'manhattan']:
    knn = KNeighborsRegressor(n_neighbors=5, metric=metric)
    knn.fit(X_train, y_train)
    mse_results[metric] = mean_squared_error(y_test, knn.predict(X_test))
```

```
print("MSE Comparison:", mse_results)

# Bias-Variance analysis
mse_values = []
for k in [1, 5, 10, 20, 50]:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train, y_train)
    mse_values.append(mean_squared_error(y_test, knn.predict(X_test)))

plt.plot([1,5,10,20,50], mse_values, marker='o')
plt.title("K vs MSE (Bias-Variance Tradeoff)")
plt.xlabel("K")
plt.ylabel("MSE")
plt.show()
```

Question 10: KNN with KD-Tree/Ball Tree, Imputation, and Real-World Data

Task:

1. Load the Pima Indians Diabetes dataset (contains missing values).
2. Use KNN Imputation (sklearn.impute.KNNImputer) to fill missing values.
3. Train KNN using:
    a. Brute-force method
    b. KD-Tree
    c. Ball Tree
4. Compare their training time and accuracy.
5. Plot the decision boundary for the best-performing method (use 2 most important features).

Dataset: Pima Indians Diabetes

Answer:

```
 import pandas as pd
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import time
```

```
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
cols =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigree','Age','Outcome']
df = pd.read_csv(url, names=cols)

# Replace 0 with NaN for missing
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] =
df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0, np.nan)

# Impute missing values
imputer = KNNImputer(n_neighbors=5)
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=cols)

X = df_imputed.drop('Outcome', axis=1)
y = df_imputed['Outcome']

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

methods = ['brute', 'kd_tree', 'ball_tree']
for algo in methods:
    start = time.time()
    knn = KNeighborsClassifier(n_neighbors=5, algorithm=algo)
    knn.fit(X_train, y_train)
    acc = accuracy_score(y_test, knn.predict(X_test))
    print(f"{algo} -> Accuracy: {acc:.3f}, Time: {time.time()-start:.3f}s")
```