

Bagging & Boosting, KNN & Stacking — Assignment Solutions

Prepared for: DA-AG-010-ASSIGNMENT ■ Prepared by: ChatGPT ■ Date: 2-Nov-2025 ■ ■

Question 1: Fundamental idea behind ensemble techniques; Bagging vs Boosting

Fundamental idea:

Ensemble methods combine multiple models (learners) to produce a single stronger model. The combination reduces variance, bias, or improves predictions by leveraging diversity among learners.

Bagging (Bootstrap Aggregating):

- Approach: Train many base learners independently on different bootstrap samples (random sampling with replacement) of the training data; aggregate predictions (majority vote for classification, average for regression).
- Objective: Reduce variance and overfitting of high-variance models (e.g., decision trees) by averaging many trees.
- Key property: Parallel training, each model gets equal weight.

Boosting:

- Approach: Train learners sequentially; each new learner focuses on mistakes made by previous ones (re-weighting samples or fitting residuals).
- Objective: Reduce bias by combining weak learners into a stronger one; often produces low-bias, lower-variance ensembles.
- Key property: Sequential dependency, learners are weighted based on performance.

Summary: Bagging reduces variance via parallel independent models; boosting reduces bias by sequentially correcting errors.

Question 2: How Random Forest reduces overfitting compared to a single tree

Random Forest reduces overfitting by introducing randomness in two ways:

- 1) Bootstrap aggregation: each tree is trained on a different bootstrap sample, so trees see different data.
- 2) Feature randomness: at each split, a random subset of features is considered, which decorrelates trees.

Two key hyperparameters:

- n_estimators: number of trees. More trees reduce variance (to a point) and stabilize predictions.
- max_features: number of features considered at each split. Smaller values increase diversity among trees and reduce correlation, lowering overfitting.

Additional useful hyperparameters: max_depth (limits tree size) and min_samples_leaf (prevents tiny leaves).

Question 3: What is Stacking and how it differs from bagging/boosting

Stacking (stacked generalization):

- Approach: Train multiple base-level models (diverse learners). Then train a meta-learner on the base models' predictions (usually on out-of-fold predictions) to produce final predictions.
- Difference: Bagging averages many similar models; boosting sequentially corrects errors. Stacking learns how to best combine differing model predictions via a meta-model.
- Example use case: Combine Logistic Regression, RandomForest, and SVM as base models; train a small neural network or LR as a meta-learner on their validation predictions to improve overall accuracy.

Question 4: OOB Score in Random Forest

OOB (Out-Of-Bag) score:

- Definition: For each training sample, the trees that did not include that sample in their bootstrap sample can be used to predict that sample. Aggregating those predictions gives an OOB estimate.
- Why useful: OOB provides an unbiased estimate of generalization performance without a separate validation set or cross-validation.
- How it helps: Saves data (no need for hold-out), quick model evaluation while training, and useful for hyperparameter tuning when data is limited.

Question 5: Compare AdaBoost and Gradient Boosting

How they handle errors:

- AdaBoost: Focuses on misclassified samples by increasing their weights so next weak learner focuses more on them.
- Gradient Boosting: Fits each new learner to the negative gradient (residuals) of the loss function; it's an optimization approach.

Weight adjustment:

- AdaBoost: Maintains sample weights; updates weights multiplicatively based on correctness and learner's error; final learners get weights based on performance.
- Gradient Boosting: No explicit sample-weight vector for boosting strategy (some variants use sample weights); instead optimizes loss via gradient descent in function space; final model is sum of scaled learners (learning rate controls contribution).

Typical use cases:

- AdaBoost: Good for clean datasets, works well with simple base learners (decision stumps); can be sensitive to noisy labels/outliers.
- Gradient Boosting (e.g., XGBoost, LightGBM, CatBoost): Powerful for structured/tabular data, handles complex patterns, provides many enhancements (regularization, tree pruning, categorical handling).

Question 6: Why CatBoost handles categorical features well

CatBoost strengths on categorical data:

- CatBoost uses ordered target statistics (also called target encoding with special ordering) to convert categorical features to numeric representations while avoiding target leakage.
- It applies permutations/ordering to compute aggregated statistics using only prior data for each row, which reduces overfitting.
- It also supports combinations of categorical features and integrates them into split finding efficiently.

Result: Minimal preprocessing (no one-hot encoding required), better handling of high-cardinality categories, and reduced target leakage compared to naive target encoding.

Practical Code & Explanations (Q7–Q10)

Below are step-by-step Python code blocks for Questions 7–10. Copy these into a Jupyter notebook or a .py file to run. Explanations and expected outputs are provided below each code block.

Notes:

- Requires: Python 3.8+, scikit-learn, pandas, numpy, matplotlib, seaborn (optional), scikit-plot (optional).
- For Q10 (Pima dataset), code attempts to download from the provided URL. If running offline, place

'diabetes.csv' next to the script or change the path.

Q7 — Code (Wine dataset)

```
# Question 7: KNN Classifier — Wine Dataset Analysis with Optimization

import numpy as np
import pandas as pd
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 accuracy_score, classification_report

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

# 2. Split 70/30
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42,
stratify=y)

# 3. Train KNN default K=5 without scaling
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Unscaled K=5 Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# 4. Apply StandardScaler, retrain KNN
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

knn_s = KNeighborsClassifier(n_neighbors=5)
knn_s.fit(X_train_s, y_train)
y_pred_s = knn_s.predict(X_test_s)
print("Scaled K=5 Accuracy:", accuracy_score(y_test, y_pred_s))
print(classification_report(y_test, y_pred_s))

# 5. GridSearchCV for best K and distance metric
param_grid = {
    'n_neighbors': list(range(1,21)),
    'metric': ['euclidean', 'manhattan']
}
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

```

grid.fit(X_train_s, y_train) # use scaled data for grid search
print("Best params:", grid.best_params_, "Best CV score:", grid.best_score_)

# 6. Train optimized KNN and compare
best_knn = grid.best_estimator_
y_pred_best = best_knn.predict(X_test_s)
print("Optimized KNN Accuracy on test:", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))

```

Q8 — Code (Breast Cancer + PCA + KNN)

```
# Question 8: PCA + KNN with Variance Analysis and Visualization
```

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

```

```
# 1. Load dataset
```

```

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

```

```
# 2. Standardize then apply PCA and scree plot
```

```

scaler = StandardScaler()
X_s = scaler.fit_transform(X)

pca = PCA()
X_pca = pca.fit_transform(X_s)

explained = pca.explained_variance_ratio_
# Scree plot code (uncomment to run in a notebook)
# plt.plot(np.cumsum(explained), marker='o')
# plt.xlabel('Number of components')
# plt.ylabel('Cumulative explained variance')
# plt.grid(True)

```

```
# 3. Retain 95% variance
```

```

pca95 = PCA(n_components=0.95)
X_pca95 = pca95.fit_transform(X_s)
n_components_95 = X_pca95.shape[1]
print("Number of components to retain 95% variance:", n_components_95)

```

```
# 4. Train KNN on original and PCA-transformed data
```

```

X_train, X_test, y_train, y_test = train_test_split(X_s, y, test_size=0.30, random_state=42,
stratify=y)

knn_orig = KNeighborsClassifier(n_neighbors=5)

knn_orig.fit(X_train, y_train)

acc_orig = accuracy_score(y_test, knn_orig.predict(X_test))

# PCA transform train/test with pca95 fitted on full or train set

pca95_train = PCA(n_components=0.95).fit(X_train)

X_train_p = pca95_train.transform(X_train)

X_test_p = pca95_train.transform(X_test)

knn_pca = KNeighborsClassifier(n_neighbors=5)

knn_pca.fit(X_train_p, y_train)

acc_pca = accuracy_score(y_test, knn_pca.predict(X_test_p))

print("KNN accuracy on original data:", acc_orig)

print("KNN accuracy on PCA (95%) data:", acc_pca)

# 5. Visualize first two principal components (scatter)

pca2 = PCA(n_components=2)

X2 = pca2.fit_transform(X_s)

# plt.figure(figsize=(8,6))

# plt.scatter(X2[:,0], X2[:,1], c=y, cmap='viridis', alpha=0.7)

# plt.xlabel('PC1'); plt.ylabel('PC2'); plt.title('First two principal components')

```

Q9 — Code (KNN Regressor)

```

# Question 9: KNN Regressor with distance metrics and K-value analysis

import numpy as np

from sklearn.datasets import make_regression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt

# 1. Generate synthetic regression data

X, y = make_regression(n_samples=500, n_features=10, noise=5.0, random_state=42)

# 2. KNN regressor with Euclidean and Manhattan (K=5)

knn_euc = KNeighborsRegressor(n_neighbors=5, metric='euclidean')

knn_man = KNeighborsRegressor(n_neighbors=5, metric='manhattan')

knn_euc.fit(X, y)

knn_man.fit(X, y)

pred_euc = knn_euc.predict(X)

pred_man = knn_man.predict(X)

mse_euc = mean_squared_error(y, pred_euc)

mse_man = mean_squared_error(y, pred_man)

```

```

print("MSE Euclidean (K=5) on training data:", mse_euc)
print("MSE Manhattan (K=5) on training data:", mse_man)

# 3. Test multiple K values and plot K vs MSE (use simple train/test split)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
ks = [1,5,10,20,50]
mse_list = []
for k in ks:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred = knn.predict(X_test)
    mse_list.append(mean_squared_error(y_test, pred))

# Plot K vs MSE (uncomment to show)
# plt.plot(ks, mse_list, marker='o')
# plt.xlabel('K'); plt.ylabel('MSE'); plt.title('K vs MSE')
print("K values:", ks)
print("MSEs:", mse_list)

```

Q10 — Code (Pima Diabetes + KNN Imputation + KD-Tree/Ball Tree)

```

# Question 10: KNN with KD-Tree/Ball Tree, Imputation, and Real-World Data (Pima Indians Diabetes)

import numpy as np
import pandas as pd
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import time

# Dataset URL (provided)
url = 'https://raw.githubusercontent.com/MasteriNeuron/datasets/refs/heads/main/diabetes.csv'

# Try to load (if offline, replace with local path)
try:
    df = pd.read_csv(url)
except Exception as e:
    print("Could not download dataset. Please download 'diabetes.csv' and place it locally.")
    # Example fallback: create a small synthetic dataset (not ideal)
    from sklearn.datasets import make_classification
    X_synth, y_synth = make_classification(n_samples=768, n_features=8, n_informative=5,
                                           random_state=42)
    df = pd.DataFrame(X_synth, columns=[f'feat{i}' for i in range(X_synth.shape[1])])

```

```

df['Outcome'] = y_synth

# Identify missing values marker (in some Pima datasets 0 means missing for some columns)
# Replace zeros with NaN for selected columns (Glucose, BloodPressure, SkinThickness, Insulin, BMI)
cols_with_zero_missing = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for c in cols_with_zero_missing:
    if c in df.columns:
        df[c] = df[c].replace(0, np.nan)

# 2. KNN imputation
imputer = KNNImputer(n_neighbors=5)
X = df.drop(columns=['Outcome'])
y = df['Outcome'] if 'Outcome' in df.columns else df.iloc[:, -1]
X_imp = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# 3. Train KNN using different algorithms and compare time & accuracy
X_train, X_test, y_train, y_test = train_test_split(X_imp, y, test_size=0.3, random_state=42,
stratify=y)

methods = ['brute', 'kd_tree', 'ball_tree']
results = {}
for method in methods:
    clf = KNeighborsClassifier(n_neighbors=5, algorithm=method)
    t0 = time.time()
    clf.fit(X_train, y_train)
    train_time = time.time() - t0
    acc = accuracy_score(y_test, clf.predict(X_test))
    results[method] = {'time': train_time, 'accuracy': acc}

print(results)

# 5. For decision boundary plotting: select top 2 features by importance (here we use feature
variance as proxy)
variances = X_imp.var().sort_values(ascending=False)
top2 = variances.index[:2].tolist()
print("Top 2 features:", top2)

# Train best method on 2 features and plot decision boundary (requires matplotlib)

```