

Ensemble Learning | Assignment

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

Answer:

Ensemble Learning is a machine learning technique where multiple models (called base learners) are combined to solve a single problem and improve overall performance.

The key idea behind ensemble learning is the "Wisdom of the Crowd." By aggregating the predictions of several models, the ensemble can:

- Reduce Variance: Minimize the impact of overfitting (common in Bagging).
- Reduce Bias: Improve the model's ability to learn complex patterns (common in Boosting).
- Improve Robustness: Reduce the likelihood of making a significant error by averaging out the idiosyncratic errors of individual models.

Why it Works:

- Reduces variance (e.g., Bagging)
- Reduces bias (e.g., Boosting)
- Minimizes overfitting
- Improves generalization on unseen data

Common Ensemble Methods:

- Bagging (Random Forest)
- Boosting (AdaBoost, Gradient Boosting)
- Stacking

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

Answer:

Bagging: Bagging (Bootstrap Aggregating) is an ensemble learning technique where multiple models are trained independently on different bootstrap samples of the dataset, and their predictions are combined (usually by averaging or majority voting).

Training Process

- All models are trained in parallel
- Each model is unaware of the others
- Reduces randomness through averaging

Data Sampling Method

- Uses random sampling with replacement
- Each model gets a different subset of data

Handling Bias and Variance

- Mainly reduces variance
- Suitable for overfitting models

Sensitivity to Noise

- Less sensitive to noisy data
- Treats all samples equally

Common Algorithms

- Bagging Classifier
- Random Forest

Boosting: Boosting is an ensemble learning technique where models are trained sequentially, and each new model focuses more on the data points that were misclassified by previous models.

Training Process

- Models are trained one after another
- Each model learns from the errors of the previous one
- Improves learning on difficult samples

Data Sampling Method

- Uses the same dataset
- Assigns higher weights to misclassified observations

Handling Bias and Variance

- Reduces bias and variance
- Suitable for underfitting models

Sensitivity to Noise

- More sensitive to noise
- Can over-focus on outliers

Common Algorithms

- AdaBoost
- Gradient Boosting
- XGBoost

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

Answer:

Bootstrap sampling is a statistical technique that involves creating multiple random subsets of the original dataset by sampling with replacement. This means a single observation can appear multiple times in one subset and not at all in another.

Role in Bagging/Random Forest:

- **Diversity:** It ensures that each individual tree in the forest is trained on a slightly different version of the data, which leads to a diverse set of models.
- **Independence:** Since each tree "sees" different data points, their errors are less likely to be correlated. When these trees are averaged, the collective error (variance) is significantly reduced.
- Each tree is trained on a different bootstrap sample
- Creates diversity among trees
- Reduces correlation between models
- Leads to lower variance and better accuracy

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 data points from the original training set that were *not* included in a specific bootstrap sample during the training of a particular base learner (like a tree in a Random Forest). Typically, about 1/3 of the data remains "out-of-bag" for each tree.

Role of OOB Score:

- **Validation:** The OOB score is used as a built-in cross-validation mechanism.
- **Evaluation:** For each data point, the ensemble predicts the outcome using only the trees that did *not* have that point in their training set. The accuracy of these predictions (the OOB score) provides an unbiased estimate of the model's performance on unseen data without needing a separate validation set.

Advantages:

- No need for separate validation data
- Efficient and unbiased performance estimate

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

Evaluating which features contribute most to predictions differs in stability and reliability between these two models:

- **Single Decision Tree:**
 - **High Variance:** Importance is determined by how much a feature reduces impurity (like Gini or Entropy) at specific nodes.
 - **Instability:** A small change in the data can lead to a completely different tree structure, making the feature importance rankings highly volatile.
- **Random Forest:**
 - **Stability:** It calculates importance by averaging the impurity reduction across hundreds or thousands of trees.
 - **Reliability:** Because it looks at many different subsets of features and data, it provides a much more robust and "global" view of which variables are truly influential, reducing the risk of being misled by noise in the data.

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:

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier

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

# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Get feature importance
importances = pd.Series(rf.feature_importances_, index=data.feature_names)
top_5 = importances.sort_values(ascending=False).head(5)

print("Top 5 Most Important Features:")
print(top_5)
```

Output:

Top 5 Most Important Features:

worst concave points	0.143205
worst area	0.124505
mean concave points	0.108257
worst perimeter	0.095456
worst radius	0.076829

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:

```
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

# Load data
X, y = load_iris(return_X_y=True)

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

# Single Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
dt_acc = accuracy_score(y_test, dt.predict(X_test))

# Bagging Classifier
bag = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=50,
random_state=42)
bag.fit(X_train, y_train)
bag_acc = accuracy_score(y_test, bag.predict(X_test))

print("Decision Tree Accuracy:", dt_acc)
print("Bagging Accuracy:", bag_acc)
```

Output:

Decision Tree Accuracy: 0.93
Bagging Accuracy: 0.97

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:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
```

```
X, y = load_iris(return_X_y=True)
```

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

```
rf = RandomForestClassifier(random_state=42)
```

```
grid = GridSearchCV(rf, param_grid, cv=5)
grid.fit(X, y)
```

```
print("Best Parameters:", grid.best_params_)
print("Best Accuracy:", grid.best_score_)
```

Output:

```
Best Parameters: {'max_depth': 5, 'n_estimators': 100}
Best Accuracy: 0.97
```

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:

```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
X, y = fetch_california_housing(return_X_y=True)
```

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

```
bag = BaggingRegressor(random_state=42)
rf = RandomForestRegressor(random_state=42)

bag.fit(X_train, y_train)
rf.fit(X_train, y_train)

print("Bagging MSE:", mean_squared_error(y_test, bag.predict(X_test)))
print("Random Forest MSE:", mean_squared_error(y_test, rf.predict(X_test)))
```

Output:

Bagging MSE: 0.29
Random Forest MSE: 0.24

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:

As a data scientist working in a financial institution, predicting loan default is a **high-risk classification problem** where accuracy, stability, and interpretability are crucial. Ensemble learning helps improve predictive performance and reduce financial risk.

1. Choosing Between Bagging and Boosting

- **Bagging** is preferred when:
 - The model suffers from **high variance**
 - Data is relatively clean
 - Goal is model stability
- **Boosting** is preferred when:
 - The problem is complex
 - Data contains **hard-to-classify defaulters**
 - Higher accuracy is required

Chosen Method: **Boosting**

- Boosting algorithms (e.g., Gradient Boosting, XGBoost) focus more on **customers likely to default**
- Helps capture subtle patterns in transaction history

2. Handling Overfitting

To avoid overfitting while using ensemble models:

- Limit tree depth (max_depth)
- Use regularization techniques
- Apply early stopping (in boosting)
- Remove irrelevant or highly correlated features
- Use cross-validation to ensure generalization

3. Selecting Base Models

The following base learners are selected:

- **Decision Trees**
 - Handle non-linear relationships
 - Easy to interpret
- **Logistic Regression**
 - Used as a baseline model
 - Provides explainability to stakeholders

These models are well-suited for structured financial data.

4. Evaluating Performance Using Cross-Validation

- Use **K-Fold Cross-Validation (K=5 or 10)**
- Ensures model stability across different data splits
- Key evaluation metrics:
 - Accuracy
 - Precision & Recall
 - ROC-AUC (important for imbalance)

- Average the scores across folds to assess real-world performance

5. Business Justification: Why Ensemble Learning Improves Decisions

- Reduces false negatives (missed defaulters)
- Improves risk assessment accuracy
- Provides consistent predictions across customer segments
- Minimizes financial loss due to loan defaults
- Enables better credit approval strategies

Real-World Impact:

Ensemble learning allows the institution to **identify risky borrowers early**, improve loan portfolio quality, and make **data-driven, reliable lending decisions**.

Code:

```
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score

# Step 1: Create a synthetic loan default dataset
X, y = make_classification(
    n_samples=2000,
    n_features=20,
    n_informative=12,
    n_redundant=4,
    weights=[0.7, 0.3], # class imbalance (more non-defaulters)
    random_state=42
)

# Step 2: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Step 3: Train Boosting model
gb_model = GradientBoostingClassifier(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    random_state=42
)

gb_model.fit(X_train, y_train)
```

```
# Step 4: Predictions
y_pred = gb_model.predict(X_test)
y_prob = gb_model.predict_proba(X_test)[:, 1]

# Step 5: Evaluation
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)

# Step 6: Cross-validation
cv_scores = cross_val_score(gb_model, X, y, cv=5, scoring='roc_auc')

print("Test Accuracy:", accuracy)
print("Test ROC-AUC:", roc_auc)
print("Cross-Validation ROC-AUC Scores:", cv_scores)
print("Mean CV ROC-AUC:", cv_scores.mean())
```

Output:

Test Accuracy: 0.89

Test ROC-AUC: 0.93

Cross-Validation ROC-AUC Scores: [0.91 0.92 0.94 0.93 0.92]

Mean CV ROC-AUC: 0.924