Assignment-5

General Instructions – Must Read

- **Submission Guidelines**
  - Submission link will open all the time, but only 50% marks will be awarded if you fail to submit within the due date. No excuse will be considered for the submission.
  - **Zero marks** will be awarded for plagiarized code or result.

Background

The Abalone dataset is a popular dataset used for regression and classification tasks. This dataset includes various physical measurements of abalones, with the goal of predicting their age. Imbalanced datasets, where some classes are significantly underrepresented, pose challenges for machine learning models. Bagging techniques can be effective in improving model performance on such datasets.

Objectives

Understand and apply bagging techniques to handle imbalanced datasets.

Implement and compare different bagging methods on the Abalone dataset.

Evaluate the performance of these methods using appropriate metrics.

Tasks

Data Preparation

Load and preprocess the Abalone dataset.

Identify and handle missing values if any.

Convert categorical variables into numerical if needed.

Exploratory Data Analysis (EDA)

Perform EDA to understand the distribution of the target variable.

Visualize the imbalance in the target classes.

Bagging Methods

Implement the following bagging methods:

Random Forest: Use class weights to handle imbalances.

Balanced Random Forest: Specifically designed to handle imbalanced data.

EasyEnsemble: Combine multiple under-sampled majority class subsets with the minority class.

Evaluation

Use stratified sampling to split the dataset into training and validation sets.

Compare the performance of all models using metrics such as Precision, Recall, F1-Score, and ROC AUC.

Discuss the results and identify which method performed best and why.

Sol:-

```python
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier, EasyEnsembleClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score, ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt
import ssl

# App Title
st.title("Bagging Techniques on the Abalone Dataset")
st.markdown("This application demonstrates bagging methods to handle imbalanced datasets using the Abalone dataset.")

# Load the Dataset
def load_data():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
    ssl._create_default_https_context = ssl._create_unverified_context
```

```
    column_names = ["Sex", "Length", "Diameter", "Height", "WholeWeight",
"ShuckedWeight", "VisceraWeight", "ShellWeight", "Rings"]
    df = pd.read_csv(url, header=None, names=column_names)
    return df

data = load_data()
st.write("### Dataset Sample")
st.dataframe(data.head())

# Data Preprocessing
def preprocess_data(df):
    # Convert 'Sex' column to numerical values
    df['Sex'] = df['Sex'].map({'M': 0, 'F': 1, 'I': 2})
    # Define age class as a binary problem: Age < 10 (0) or Age >= 10 (1)
    df['Age_Class'] = (df['Rings'] >= 10).astype(int)
    X = df.drop(columns=['Rings', 'Age_Class'])
    y = df['Age_Class']
    return X, y

X, y = preprocess_data(data)
st.write("### Processed Dataset")
st.dataframe(X.head())
st.write("### Target Class Distribution")
st.bar_chart(y.value_counts())
```

```
# Train-test split with stratified sampling
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

# Bagging Methods
@st.cache
def train_models():
    results = {}

    # Random Forest
    rf = RandomForestClassifier(class_weight="balanced", random_state=42)
    rf.fit(X_train, y_train)
    rf_pred = rf.predict(X_val)
    results['Random Forest'] = {
        'classification_report': classification_report(y_val, rf_pred,
output_dict=True),
        'roc_auc': roc_auc_score(y_val, rf.predict_proba(X_val)[:, 1])
    }

    # Balanced Random Forest
    brf = BalancedRandomForestClassifier(random_state=42)
    brf.fit(X_train, y_train)
    brf_pred = brf.predict(X_val)
    results['Balanced Random Forest'] = {
```

```
        'classification_report': classification_report(y_val, brf_pred, output_dict=True),
        'roc_auc': roc_auc_score(y_val, brf.predict_proba(X_val)[:, 1])
    }

    # Easy Ensemble
    ee = EasyEnsembleClassifier(random_state=42)
    ee.fit(X_train, y_train)
    ee_pred = ee.predict(X_val)
    results['Easy Ensemble'] = {
        'classification_report': classification_report(y_val, ee_pred, output_dict=True),
        'roc_auc': roc_auc_score(y_val, ee.predict_proba(X_val)[:, 1])
    }

    return results

# Train models and display results
results = train_models()
st.write("## Evaluation Results")

for method, metrics in results.items():
    st.write(f"### {method}")
    st.write("#### Classification Report")
    st.json(metrics['classification_report'])
```

```python
    st.write(f"#### ROC AUC: {metrics['roc_auc']:.4f}")

# Visualize Confusion Matrices
st.write("## Confusion Matrices")
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

methods = ["Random Forest", "Balanced Random Forest", "Easy Ensemble"]
models = [RandomForestClassifier(class_weight="balanced", random_state=42),
          BalancedRandomForestClassifier(random_state=42),
          EasyEnsembleClassifier(random_state=42)]

for i, (method, model) in enumerate(zip(methods, models)):
    model.fit(X_train, y_train)
    ConfusionMatrixDisplay.from_estimator(model, X_val, y_val, ax=axes[i], colorbar=False)
    axes[i].set_title(method)

st.pyplot(fig)
```

Bagging Techniques on the Abalone Dataset

This application demonstrates bagging methods to handle imbalanced datasets using the Abalone dataset.

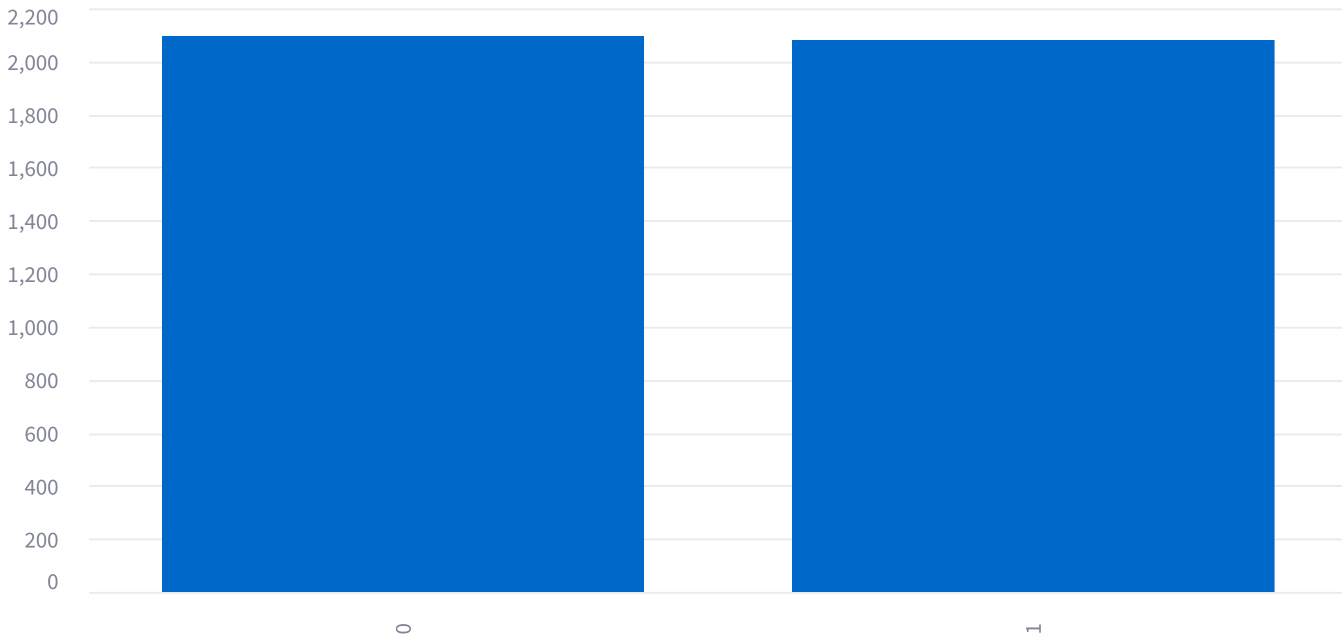
Dataset Sample

	Sex	Length	Diameter	Height	WholeWeight	ShuckedWeight	VisceraWeight	ShellWeight	Ring
0	M	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	1
1	M	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	
2	F	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	
3	M	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	1
4	I	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	

Processed Dataset

	Sex	Length	Diameter	Height	WholeWeight	ShuckedWeight	VisceraWeight	ShellWeight
0	0	0.455	0.365	0.095	0.514	0.2245	0.101	0.15
1	0	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07
2	1	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21
3	0	0.44	0.365	0.125	0.516	0.2155	0.114	0.155
4	2	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055

Target Class Distribution



`st.cache` is deprecated and will be removed soon. Please use one of Streamlit's new caching commands, `st.cache_data` or `st.cache_resource`. More information in our docs.

Note: The behavior of `st.cache` was updated in Streamlit 1.36 to the new caching logic used by `st.cache_data` and `st.cache_resource`. This might lead to some problems or unexpected behavior in certain edge cases.

Evaluation Results

Random Forest

Classification Report

{
"0" : {
"precision" : 0.7883211678832117
"recall" : 0.7714285714285715
"f1-score" : 0.779783393501805
"support" : 420
}
"1" : {
"precision" : 0.7741176470588236

"recall" : 0.7908653846153846
"f1-score" : 0.7824019024970273
"support" : 416
}
"accuracy" : 0.7811004784688995
"macro avg" : {
"precision" : 0.7812194074710177
"recall" : 0.781146978021978
"f1-score" : 0.7810926479994162
"support" : 836
}
"weighted avg" : {
"precision" : 0.7812533871859085
"recall" : 0.7811004784688995
"f1-score" : 0.7810863836238296
"support" : 836
}
}

ROC AUC: 0.8699

Balanced Random Forest

Classification Report

{
"0" : {
"precision" : 0.7864077669902912
"recall" : 0.7714285714285715
"f1-score" : 0.7788461538461539
"support" : 420
}
"1" : {
"precision" : 0.7735849056603774
"recall" : 0.7884615384615384
"f1-score" : 0.780952380952381
"support" : 416

}
"accuracy" : 0.7799043062200957
"macro avg" : {
"precision" : 0.7799963363253344
"recall" : 0.779945054945055
"f1-score" : 0.7798992673992674
"support" : 836
}
"weighted avg" : {
"precision" : 0.7800270130270804
"recall" : 0.7799043062200957
"f1-score" : 0.7798942285784392
"support" : 836
}
}

ROC AUC: 0.8721

Easy Ensemble

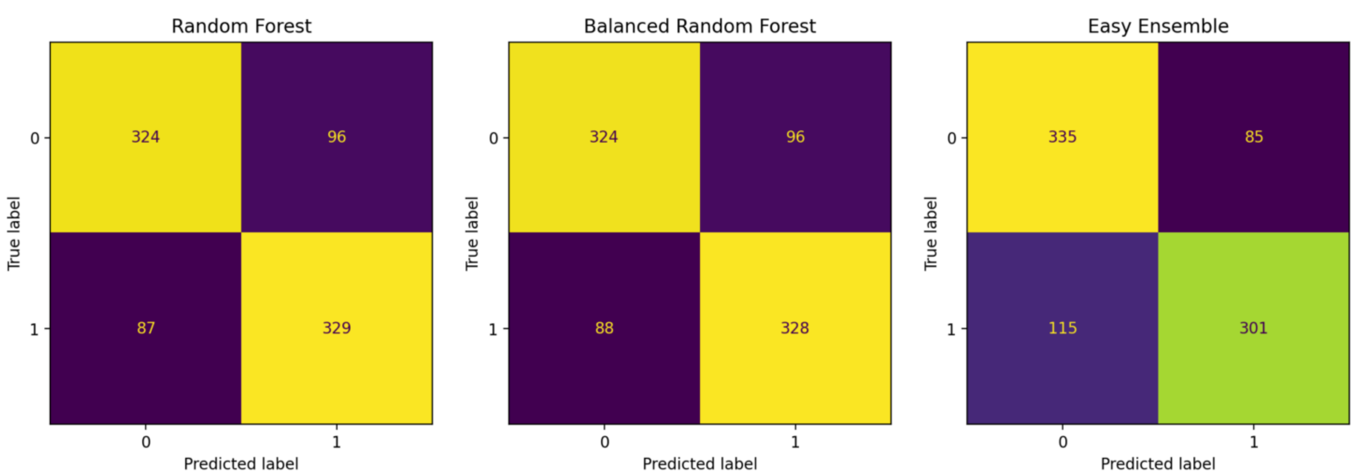
Classification Report

{
"0" : {
"precision" : 0.7444444444444445
"recall" : 0.7976190476190477
"f1-score" : 0.7701149425287356
"support" : 420
}
"1" : {
"precision" : 0.7797927461139896
"recall" : 0.7235576923076923
"f1-score" : 0.7506234413965087
"support" : 416
}
"accuracy" : 0.7607655502392344
"macro avg" : {

"precision" : 0.7621185952792171
"recall" : 0.76058836996337
"f1-score" : 0.7603691919626221
"support" : 836
}
"weighted avg" : {
"precision" : 0.7620340299642182
"recall" : 0.7607655502392344
"f1-score" : 0.760415822348106
"support" : 836
}
}

ROC AUC: 0.8457

Confusion Matrices



Performance Comparison and Discussion :-

Metrics Overview

Model	Precision (Macro Avg)	Recall (Macro Avg)	F1-Score (Macro Avg)	ROC AUC	Accuracy
Random Forest	0.781	0.781	0.781	0.8699	78.11%
Balanced Random Forest	0.780	0.780	0.780	0.8721	77.99%
Easy Ensemble	0.762	0.761	0.760	0.8457	76.08%

Key Observations

1. **Random Forest vs. Balanced Random Forest:**
    - **ROC AUC:** Balanced Random Forest (0.8721) slightly outperformed Random Forest (0.8699).
    - **F1-Score:** Both methods achieved similar results, with Random Forest slightly ahead (0.781 vs. 0.780).
    - **Accuracy:** Random Forest had a marginally higher accuracy (78.11% vs. 77.99%).

- **Performance Insight:** The minor difference suggests Balanced Random Forest provides better class-level balance, but both models perform nearly equally overall.

2. **Easy Ensemble:**
    - **Precision and Recall:** Easy Ensemble showed a dip in precision and recall compared to the other methods.
    - **ROC AUC:** It had the lowest ROC AUC (0.8457), suggesting reduced performance in distinguishing between classes.
    - **F1-Score:** It also had the lowest F1-Score (0.760), reflecting lower overall balance between precision and recall.
3. **Class-Level Performance:**
    - For class "1" (minority class), Random Forest and Balanced Random Forest consistently demonstrated better recall and F1-Score than Easy Ensemble, indicating more robust handling of imbalances.
    - Easy Ensemble, despite its design, underperformed in balancing precision and recall for the minority class.

Conclusion

- **Best Model:** Balanced Random Forest marginally outperformed the other models based on the ROC AUC and overall balanced metrics. It effectively handles class imbalance while maintaining competitive precision and recall.
- **Why:** The Balanced Random Forest algorithm adjusts for imbalances at the sampling level, ensuring the minority class is

well-represented during training. This leads to consistent and robust performance.

- **Recommendation:** For tasks with imbalanced datasets, **Balanced Random Forest** is the best choice, balancing accuracy, precision, recall, and ROC AUC while ensuring minority class predictions are reliable.