This project is divided into multiple parts involving dataset selection, model evaluation using different classifiers (including weak learners), dataset augmentation, and performance comparison of three gradient boosting algorithms: XGBoost, CATBoost, and LightGBM. The following steps were performed:

1. **Dataset Selection**: The Iris, Wine, and Breast Cancer datasets were chosen from scikit-learn's toy datasets.
2. **Model Evaluation**: Weak learners including k-Nearest Neighbors (kNN), Support Vector Machines (SVM), Naive Bayes, and decision trees (ID3 and CART) were applied to predict the classification labels of the chosen datasets.
3. **Dataset Augmentation**: The Iris and Wine datasets were augmented using the SMOTE algorithm, increasing the dataset size by fivefold.
4. **Gradient Boosting Performance**: XGBoost, CATBoost, and LightGBM were applied to both original and augmented datasets to measure training time and accuracy.

1.Choose three arbitrary datasets; you can use scikit learn toy datasets as well (https://scikit-learn.org/stable/datasets/toy_dataset.html)

Three datasets were used for this analysis:

- **Iris**: A classification dataset with 150 samples and 3 classes of iris flowers.
- **Wine**: A dataset with 178 samples, used for predicting wine quality across 3 categories.
- **Breast Cancer**: A dataset with 569 samples, used to predict if a tumor is malignant or benign.

Code:

```
from sklearn.datasets import load_iris, load_wine, load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier from sklearn.svm
import SVC from sklearn.naive_bayes import GaussianNB from
sklearn.tree import DecisionTreeClassifier from sklearn.metrics import
accuracy_score

# Load datasets
iris = load_iris() wine =
load_wine() cancer =
load_breast_cancer() datasets
= {
    'Iris': iris,
```

```
    'Wine': wine,
    'Breast Cancer': cancer
}
```

2.Apply a prediction on these datasets, which all weak learners, including kNN, SVM, Naïve Bayes, and at least two decision trees (ID3, CART,C4.5, CHAID). Report the accuracy of each algorithm. The train test split should be 70/30, and you should also use 10 fold cross-validation.

Models: The code initializes several classification models, which include:

- **KNeighborsClassifier**: k-Nearest Neighbors.
- **SVC**: Support Vector Classifier (SVM).
- **GaussianNB**: Naive Bayes classifier.
- **DecisionTreeClassifier**: Initialized twice, once with the entropy criterion (ID3) and once with the gini criterion (CART).

Train-Test Split: The data is split into a training set and a test set using `train_test_split()` with 70% of the data for training and 30% for testing. This is controlled by `test_size=0.3`.

Training and Prediction:

- For each dataset, the models are trained using the training data (`model.fit(X_train, y_train)`).
- After training, predictions are made on the test data (`model.predict(X_test)`).

Accuracy Calculation:

- **Test Accuracy**: Accuracy is calculated on the test set using `accuracy_score(y_test, y_pred)`, which evaluates the model's performance on unseen data.
- **Cross-Validation Accuracy**: A 10-fold cross-validation is performed using `cross_val_score()`, which splits the data into 10 parts, trains the model on 9 parts, and tests it on the 10th. This is repeated 10 times to get an average cross-validation accuracy.

Results: The results for each model on each dataset are stored in a dictionary. For each model, two accuracies are reported:

- **Test Accuracy**: Accuracy on the holdout test set.
- **Cross-Validation Accuracy**: Average accuracy from 10-fold cross-validation.

```
Results for Iris dataset:
             Test Accuracy  Cross-Validation Accuracy
kNN               1.000000                   0.953333
SVM               1.000000                   0.960000
Naive Bayes       0.977778                   0.953333
ID3               0.977778                   0.940000
CART              1.000000                   0.933333

Results for Wine dataset:
             Test Accuracy  Cross-Validation Accuracy
kNN               0.740741                   0.685621
SVM               0.759259                   0.674510
Naive Bayes       1.000000                   0.977778
ID3               0.833333                   0.927124
CART              0.944444                   0.877451

Results for Breast Cancer dataset:
             Test Accuracy  Cross-Validation Accuracy
kNN               0.959064                   0.935025
SVM               0.935673                   0.915633
Naive Bayes       0.941520                   0.936811
ID3               0.964912                   0.931548
CART              0.929825                   0.915664
```

Iris Dataset Results:

- **Test Accuracy:** All models (kNN, SVM, CART) except Naive Bayes and ID3 achieved a perfect accuracy of 1.000 on the test set, indicating that they perfectly classified all test instances. Naive Bayes and ID3 had slightly lower performances, with test accuracies of 0.977778.
- **Cross-Validation Accuracy:** kNN, SVM, and Naive Bayes performed consistently well, with cross-validation accuracies around 95%. The decision tree models (ID3 and CART) performed slightly lower with accuracies around 94% and 93%, respectively, which is still strong performance.

Wine Dataset Results:

- **Test Accuracy:** Naive Bayes performed the best with a perfect test accuracy of 1.000. CART also achieved strong performance with 0.944444. However, kNN, SVM, and ID3 showed lower test accuracies, with kNN having the lowest at 0.740741.

- **Cross-Validation Accuracy:** Naive Bayes again stands out with a high cross-validation accuracy of 0.977778. CART and ID3 also performed well with accuracies of 0.877451 and 0.927124, respectively, while kNN and SVM had lower cross-validation scores around 0.68, indicating a slight inconsistency in performance across the folds.

Breast Cancer Dataset Results:

- **Test Accuracy:** kNN performed the best with an accuracy of 0.959064, followed by CART at 0.929825 and ID3 at 0.964912. SVM also achieved strong performance, but Naive Bayes had the lowest test accuracy at 0.941520.
- **Cross-Validation Accuracy:** The cross-validation scores for this dataset indicate that Naive Bayes performed relatively well, with a cross-validation accuracy of 0.936811, closely followed by kNN at 0.935025, ID3 at 0.931548, and CART at 0.915664. SVM achieved a slightly lower cross-validation score of 0.915633.

Code:

```
# Models
models = {
    'kNN': KNeighborsClassifier(),
    'SVM': SVC(),
    'Naive Bayes': GaussianNB(),
    'ID3': DecisionTreeClassifier(criterion='entropy'),
    'CART': DecisionTreeClassifier(criterion='gini')
}

# Train-test split and evaluation results = {} for
dataset_name, dataset in datasets.items():
    X_train, X_test, y_train, y_test = train_test_split(dataset.data, dataset.target, test_size=0.3,
random_state=42)

    dataset_results = {} for model_name,
    model in models.items():
        # Train and predict
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        # Accuracy on the test set accuracy =
        accuracy_score(y_test, y_pred)
```

```
        # 10-fold cross-validation accuracy with random seed skf =
        StratifiedKFold(n_splits=10, shuffle=True, random_state=42) cv_scores
        = cross_val_score(model, dataset.data, dataset.target, cv=skf)
        cv_accuracy = cv_scores.mean()

        dataset_results[model_name] = {
            'Test Accuracy': accuracy,
            'Cross-Validation Accuracy': cv_accuracy
        }

    results[dataset_name] = dataset_results

# Display results import pandas as pd for
dataset_name, dataset_results in results.items():
print(f"\nResults for {dataset_name} dataset:") df =
pd.DataFrame(dataset_results).T
    print(df)
```

3. Augment two arbitrary datasets (from step one) and increase the number of their records. How to augment them is something you need to search and realize on your own. The augmentation result should include five times more data than the original dataset. In particular, you need to build five datasets as follows:
Dataset 1 = original dataset with no changes.
Dataset 2 = original dataset no changes + augmented dataset with equal number of records to the original one
Dataset 3 = original dataset no changes + augmented dataset with 2x number of records to the original one
Dataset 4 = original dataset no changes + augmented dataset with 3x number of records to the original one
Dataset 5 = original dataset no changes + augmented dataset with 4x number of records to the original one

The objective of this task is to augment two datasets, namely the **Iris** and **Wine** datasets, using the **SMOTE** (Synthetic Minority Over-sampling Technique) algorithm to increase the number of data points. The augmented datasets should be five times larger than the original dataset, with the final result being five datasets for each, including the original and augmented data. The augmented datasets are built iteratively by progressively adding synthetic data, increasing the size of the dataset at each step.
Steps Involved

Step 1: Dataset Loading

We used two toy datasets from scikit-learn:

- **Iris Dataset**: Contains 150 samples of iris flowers, classified into three classes.
- **Wine Dataset**: Contains 178 samples of wine, classified into three categories.

Step 2: SMOTE for Augmentation

The augmentation process is handled by the `SMOTE` class from the **imbalanced-learn (imblearn)** library. The following steps are performed for each dataset:

1. **Calculate class counts**: The number of data points in each class is counted using the `np.bincount(y)` function. This is essential for generating synthetic samples proportionally for each class.
2. **Apply SMOTE**: SMOTE is applied by defining a `sampling_strategy`, where the goal is to augment the data to create synthetic points. The `factor` defines how much we want to multiply the original data.

Step 3: Dataset Augmentation

To create progressively larger datasets, the following approach is used:

1. **Dataset 1**: This is the original dataset, without any changes.
2. **Dataset 2**: This consists of the original dataset and augmented data that matches the original dataset size.
3. **Dataset 3**: This contains the original dataset and synthetic data double the size of the original dataset.
4. **Dataset 4**: This contains the original dataset and synthetic data three times the size of the original dataset.
5. **Dataset 5**: This contains the original dataset and synthetic data four times the size of the original dataset.

Step 4: Implementation

The function `augment_dataset` takes as input the features (X), labels (y), and the factor to control the amount of synthetic data to generate. It returns the augmented data using SMOTE.

The `build_augmented_datasets` function generates five datasets for each of the original datasets (Iris and Wine). The process includes:

- **Dataset 1**: The original data.

- **Dataset 2-5**: Incrementally larger datasets, augmented with synthetic data.

Step 5: Results

The size of each dataset after augmentation is printed for both the Iris and Wine datasets.

```
Iris Dataset 1: Size = 150 samples
Iris Dataset 2: Size = 300 samples
Iris Dataset 3: Size = 450 samples
Iris Dataset 4: Size = 600 samples
Iris Dataset 5: Size = 750 samples
Wine Dataset 1: Size = 178 samples
Wine Dataset 2: Size = 356 samples
Wine Dataset 3: Size = 534 samples
Wine Dataset 4: Size = 712 samples
Wine Dataset 5: Size = 890 samples
```

Code:

```
from imblearn.over_sampling import SMOTE
import numpy as np

# Augment data with the correct sampling_strategy for multi-class
def augment_dataset(X, y, factor):
    class_counts = np.bincount(y) # Count the number of instances per class
    smote = SMOTE(sampling_strategy={i: class_counts[i] * factor for i in
range(len(class_counts))})
    X_augmented, y_augmented = smote.fit_resample(X, y)
    return X_augmented, y_augmented

# Create datasets 1 to 5 as described in the task def
build_augmented_datasets(X_original, y_original): datasets =
[(X_original, y_original)] # Dataset 1 is the original data

    for i in range(1, 5): # Datasets 2 to 5
        X_augmented, y_augmented = augment_dataset(X_original, y_original,
        i) X_combined = np.vstack([X_original, X_augmented]) y_combined =
        np.hstack([y_original, y_augmented]) datasets.append((X_combined,
        y_combined))
```

```
    return datasets

# Augment datasets for the Iris and Wine datasets iris_datasets =
build_augmented_datasets(iris.data, iris.target) wine_datasets =
build_augmented_datasets(wine.data, wine.target)

# Print details of the augmented datasets for verification
for i, (X, y) in enumerate(iris_datasets): print(f"Iris
Dataset {i+1}: Size = {X.shape[0]} samples")

for i, (X, y) in enumerate(wine_datasets): print(f"Wine
    Dataset {i+1}: Size = {X.shape[0]} samples")
```

4.Measure and report the execution time and accuracy of applying XGBoost, CATBoost, and LightGBM on all five datasets for each sample. You must report them in a readable table and compare them in your explanations.

The objective of this task is to evaluate the performance of three advanced machine learning models—**XGBoost**, **CATBoost**, and **LightGBM**—on augmented datasets. The datasets, augmented in the previous steps using SMOTE, include five datasets for both the Iris and Wine datasets. These models are tested in terms of **training time** and **accuracy**, and their performance is compared across the augmented datasets.

```
Iris Results:
           Dataset     Model  Training Time (s)  Accuracy
0   Iris Dataset 1   XGBoost           0.175634  1.000000
1   Iris Dataset 1  CATBoost           0.809847  1.000000
2   Iris Dataset 1  LightGBM           0.023686  1.000000
3   Iris Dataset 2   XGBoost           0.085033  1.000000
4   Iris Dataset 2  CATBoost           0.731146  1.000000
5   Iris Dataset 2  LightGBM           0.031238  1.000000
6   Iris Dataset 3   XGBoost           0.062896  0.962963
7   Iris Dataset 3  CATBoost           1.331803  0.962963
8   Iris Dataset 3  LightGBM           0.023118  0.962963
9   Iris Dataset 4   XGBoost           0.074502  1.000000
10  Iris Dataset 4  CATBoost           1.872653  1.000000
11  Iris Dataset 4  LightGBM           0.032295  1.000000
12  Iris Dataset 5   XGBoost           0.078686  0.964444
13  Iris Dataset 5  CATBoost           2.200534  0.968889
14  Iris Dataset 5  LightGBM           0.033924  0.968889

Wine Results:
           Dataset     Model  Training Time (s)  Accuracy
0   Wine Dataset 1   XGBoost           0.064008  0.962963
1   Wine Dataset 1  CATBoost           1.385122  0.981481
2   Wine Dataset 1  LightGBM           0.030079  0.981481
3   Wine Dataset 2   XGBoost           0.064423  0.981308
4   Wine Dataset 2  CATBoost           1.702029  1.000000
5   Wine Dataset 2  LightGBM           0.040013  1.000000
6   Wine Dataset 3   XGBoost           0.076484  1.000000
7   Wine Dataset 3  CATBoost           2.899273  1.000000
8   Wine Dataset 3  LightGBM           0.024456  1.000000
9   Wine Dataset 4   XGBoost           0.056318  1.000000
10  Wine Dataset 4  CATBoost           3.178171  1.000000
11  Wine Dataset 4  LightGBM           0.031261  0.981308
12  Wine Dataset 5   XGBoost           0.062896  1.000000
13  Wine Dataset 5  CATBoost           3.216020  1.000000
14  Wine Dataset 5  LightGBM           0.046891  1.000000
```

Iris Dataset:

- **Accuracy:**
  - XGBoost consistently performed with a 100% accuracy on Datasets 1, 2, and 4, with slightly lower performance on Datasets 3 and 5, where the accuracy dropped to 96.296% and 96.444%, respectively.
  - CATBoost also performed at 100% accuracy on Datasets 1, 2, and 4, but its performance dipped on Dataset 3 (96.296%) and Dataset 5 (96.444%).
  - LightGBM achieved 100% accuracy on Datasets 1, 2, and 4, with marginally lower accuracy for Dataset 3 (96.296%) and Dataset 5 (96.889%).
- **Training Time:**
  - LightGBM consistently had the fastest training time across all datasets, with times as low as 0.02 seconds.
  - XGBoost and CATBoost required more time than LightGBM, with CATBoost being the slowest, especially for larger datasets.

Wine Dataset:

- **Accuracy:**
  - XGBoost and LightGBM both achieved 100% accuracy on Datasets 2, 3, and 5, with slightly lower accuracy on Datasets 1 and 4.

    - CATBoost achieved perfect accuracy on Datasets 2, 3, and 5 as well but took significantly longer in terms of training time compared to the other models, especially as the datasets increased in size.
- **Training Time:**
    - Once again, LightGBM was the fastest for all datasets, with times as low as 0.017 seconds.
    - XGBoost had slightly longer training times compared to LightGBM, but it was still significantly faster than CATBoost.
    - CATBoost was the slowest, especially for the larger datasets, where it took around 3 seconds for Datasets 4 and 5.

Conclusion:

- LightGBM is the most efficient model in terms of training time while maintaining high accuracy.
- XGBoost provides high accuracy but takes more time compared to LightGBM.
- CATBoost achieves high accuracy but has the longest training times, especially for larger datasets.

Code:

```
import time from xgboost import XGBClassifier
from catboost import CatBoostClassifier from
lightgbm import LGBMClassifier from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score import
pandas as pd

# Initialize models
models = {
    'XGBoost': XGBClassifier(eval_metric='mlogloss'),
    'CATBoost': CatBoostClassifier(verbose=0),
   'LightGBM': LGBMClassifier(min_gain_to_split=0.01, min_data_in_leaf=20, verbosity=-1) #
Adjusted LightGBM parameters
}
```

```
# Measure time and accuracy def measure_time_and_accuracy(model,
X_train, y_train, X_test, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    training_time = time.time() -
    start_time y_pred =
    model.predict(X_test) accuracy =
    accuracy_score(y_test, y_pred) return
    training_time, accuracy

# Function to evaluate models on datasets and return results in a dataframe
def evaluate_models_on_datasets(datasets, dataset_name):
    results = [] for i, (X, y) in
    enumerate(datasets):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        for model_name, model in models.items():
            training_time, accuracy = measure_time_and_accuracy(model, X_train, y_train, X_test,
y_test) results.append({
                'Dataset': f'{dataset_name} Dataset {i+1}',
                'Model': model_name,
                'Training Time (s)': training_time,
                'Accuracy': accuracy
            })

    return pd.DataFrame(results)

# Evaluate models on Iris datasets
iris_results = evaluate_models_on_datasets(iris_datasets, 'Iris')

# Evaluate models on Wine datasets wine_results =
evaluate_models_on_datasets(wine_datasets, 'Wine')

# Display results for Iris and Wine datasets
print("\nIris Results:") print(iris_results)

print("\nWine Results:")
print(wine_results)
```