```
# ---------------------------------------------
#                   Expt - 8.a
# ---------------------------------------------



import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
from sklearn.tree import DecisionTreeClassifier

# Reproducible
RNG = np.random.default_rng(42)

# Step 1: Create dataset
df = pd.DataFrame({
    'X1': [1, 2, 3, 4, 5, 6, 6, 7, 9, 9],
    'X2': [5, 3, 6, 8, 1, 9, 5, 8, 9, 2],
    'label': [1, 1, 0, 1, 0, 1, 0, 1, 0, 0]
})

# Ensure label dtype is int
df['label'] = df['label'].astype(int)

# Visualize
sns.scatterplot(x=df['X1'], y=df['X2'], hue=df['label'])
plt.title("Initial Data Distribution")
plt.show()

# Initialize uniform weights
df['weights'] = 1.0 / len(df)

# Helper: safe alpha calculation
def calculate_model_weight(error):
    eps = 1e-10
    # clip to (eps, 1-eps) to avoid division by zero or log of zero
    error = float(np.clip(error, eps, 1 - eps))
    return 0.5 * np.log((1 - error) / error)

# Helper: update sample weight for a single row
def update_row_weights_vec(weights, labels, preds, alpha):
    # vectorized: new_w = w * exp(-alpha * y * h) where y in {+1,-1}, h in {+1,-1}
    y_signed = np.where(labels == 1, 1.0, -1.0)
    h_signed = np.where(preds == 1, 1.0, -1.0)
    return weights * np.exp(-alpha * y_signed * h_signed)

# Helper: weighted resampling using probabilities p (length n), returns sampled indices (with replacement)
def weighted_resample_indices(n, p, rng=np.random.default_rng()):
    # np.random.choice with p must sum to 1
    p = np.array(p, dtype=float)
    p = p / p.sum()
    return rng.choice(len(p), size=n, replace=True, p=p)

# ----- First weak learner -----
X = df[['X1', 'X2']].values
y = df['label'].values  # integer dtype

dt1 = DecisionTreeClassifier(max_depth=1, random_state=42)
dt1.fit(X, y)

# plot (pass integer y)
plot_decision_regions(X, y.astype(int), clf=dt1, legend=2)
plt.title("Decision Region - Tree 1")
plt.show()

# predictions and error
df['y_pred'] = dt1.predict(X)
error1 = np.sum(df['weights'] * (df['label'] != df['y_pred']))
alpha1 = calculate_model_weight(error1)

# update weights (vectorized)
df['updated_weights'] = update_row_weights_vec(df['weights'].values, df['label'].values, df['y_pred'].values,
alpha1)
df['normalized_weights'] = df['updated_weights'] / df['updated_weights'].sum()

# ----- Second weak learner: resample based on normalized_weights -----
indices2 = weighted_resample_indices(len(df), df['normalized_weights'].values, rng=RNG)
second_df = df.iloc[indices2].reset_index(drop=True).copy()
# assign weights column for sampled dataset as the normalized weights from original rows (in sampling order)
second_df['weights'] = df['normalized_weights'].values[indices2]

# ensure label dtype integer for plotting
second_df['label'] = second_df['label'].astype(int)

X2 = second_df[['X1', 'X2']].values
y2 = second_df['label'].values

dt2 = DecisionTreeClassifier(max_depth=1, random_state=42)
dt2.fit(X2, y2)
```

```
plot_decision_regions(X2, y2.astype(int), clf=dt2, legend=2)
plt.title("Decision Region - Tree 2")
plt.show()

second_df['y_pred'] = dt2.predict(X2)
error2 = np.sum(second_df['weights'] * (second_df['label'] != second_df['y_pred']))
alpha2 = calculate_model_weight(error2)

# update weights on second_df
second_df['updated_weights'] = update_row_weights_vec(second_df['weights'].values, second_df['label'].values,
second_df['y_pred'].values, alpha2)
# normalized for next sampling
second_df['normalized_weights'] = second_df['updated_weights'] / second_df['updated_weights'].sum()

# ----- Third weak learner: resample from second_df -----
indices3 = weighted_resample_indices(len(second_df), second_df['normalized_weights'].values, rng=RNG)
third_df = second_df.iloc[indices3].reset_index(drop=True).copy()
third_df['weights'] = second_df['normalized_weights'].values[indices3]
third_df['label'] = third_df['label'].astype(int)

X3 = third_df[['X1', 'X2']].values
y3 = third_df['label'].values

# safety check
if X3.shape[0] == 0:
    raise ValueError("Third resampled dataset is empty. Check sampling logic.")

dt3 = DecisionTreeClassifier(max_depth=1, random_state=42)
dt3.fit(X3, y3)

plot_decision_regions(X3, y3.astype(int), clf=dt3, legend=2)
plt.title("Decision Region - Tree 3")
plt.show()

third_df['y_pred'] = dt3.predict(X3)
error3 = np.sum(third_df['weights'] * (third_df['label'] != third_df['y_pred']))
alpha3 = calculate_model_weight(error3)

# Final model weights
print("Model weights (alphas):", alpha1, alpha2, alpha3)

# Ensemble predict (weighted vote)
def ensemble_predict(query):
    # query: numpy array shape (1,2) or (n,2)
    p1 = dt1.predict(query)
    p2 = dt2.predict(query)
    p3 = dt3.predict(query)

    # convert 0/1 -> -1/+1
    def to_signed(arr):
        return np.where(np.array(arr) == 1, 1.0, -1.0)

    s1 = to_signed(p1)
    s2 = to_signed(p2)
    s3 = to_signed(p3)

    # weighted sum
    final_score = alpha1 * s1 + alpha2 * s2 + alpha3 * s3
    # output 1 if positive, 0 otherwise
    return np.where(final_score > 0, 1, 0)

# Tests
print("Prediction for [1,5]:", int(ensemble_predict(np.array([[1, 5]]))[0]))
print("Prediction for [9,9]:", int(ensemble_predict(np.array([[9, 9]]))[0]))
# -------------------------------------------------------
#                 EXPT - 8.b
# -------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeRegressor, plot_tree

# Step 1: Generate synthetic data
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3 * X[:, 0]**2 + 0.05 * np.random.randn(100)

# Step 2: Visualize the data
df = pd.DataFrame()
df['X'] = X.reshape(100)
df['y'] = y

plt.scatter(df['X'], df['y'])
plt.title('X vs y')
plt.xlabel('X')
plt.ylabel('y')
plt.show()

# Step 3: First prediction - mean of y
```

```
df['pred1'] = df['y'].mean()
df['res1'] = df['y'] - df['pred1']

# Step 4: Fit first tree on residuals
tree1 = DecisionTreeRegressor(max_leaf_nodes=8)
tree1.fit(df[['X']], df['res1'])

# Visualize the tree
plot_tree(tree1)
plt.title("First Tree on Residuals")
plt.show()

# Step 5: Update prediction
df['pred2'] = df['pred1'] + tree1.predict(df[['X']])
df['res2'] = df['y'] - df['pred2']

# Step 6: Manual Gradient Boosting Function
def gradient_boost(X, y, number, lr, count=1, regs=None, foo=None):
    if regs is None:
        regs = []
    if number == 0:
        return
    else:
        if count > 1:
            y = y - lr * regs[-1].predict(X)
        else:
            foo = y.copy()

        tree_reg = DecisionTreeRegressor(max_depth=5, random_state=42)
        tree_reg.fit(X, y)
        regs.append(tree_reg)

        # Plot current prediction
        x1 = np.linspace(-0.5, 0.5, 500).reshape(-1, 1)
        y_pred = sum(lr * regressor.predict(x1) for regressor in regs)

        print(f"Iteration {count}")
        plt.figure(figsize=(8, 4))
        plt.plot(x1, y_pred, label='Boosted Prediction', linewidth=2)
        plt.scatter(X[:, 0], foo, color='red', label='True y', alpha=0.6)
        plt.title(f"Boosting Round {count}")
        plt.xlabel("X")
        plt.ylabel("y")
        plt.legend()
        plt.show()
    # Recursive call
    gradient_boost(X, y, number - 1, lr, count + 1, regs, foo=foo)

# Step 7: Run Gradient Boosting
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3 * X[:, 0]**2 + 0.05 * np.random.randn(100)

gradient_boost(X, y, number=5, lr=1)
```