AdaBoost Algorithm Explained with a Mathematical Example

February 2, 2025

AdaBoost (Adaptive Boosting) is an ensemble method that combines weak learners (e.g., decision stumps) sequentially, focusing on misclassified samples by adjusting their weights. Here's a step-by-step explanation using your dataset:

Step 1: Dataset

Features: ‘Age‘, ‘Income‘
Target: ‘Purchase‘ (Yes/No)

Sample	Age	Income	Purchase
1	25	High	Yes
2	30	Low	No
3	35	Medium	Yes
4	40	High	Yes
5	45	Low	No

Step 2: Initialize Sample Weights

Start with equal weights for all samples:

$$w_i = \frac{1}{5} = 0.2 \quad \text{(for all samples)}$$

Step 3: Train Weak Learners Sequentially

We iteratively train weak learners (e.g., decision stumps) and update weights. Let's run **3 iterations**:

Iteration 1

1. **Find the best weak learner**:
    - Test possible splits (e.g., ‘Age 30‘, ‘Income = High‘, etc.).
    - **Best stump**: ‘Income = High¿
    - Predict **Yes** if ‘Income = High‘, else **No**.
2. **Calculate weighted error**:

Sample	Prediction	Actual	Correct?	Weight
1	Yes	Yes		0.2
2	No	No		0.2
3	No	Yes		0.2
4	Yes	Yes		0.2
5	No	No		0.2

$$\text{Error} = \sum(\text{Weights of misclassified}) = 0.2 \quad \text{(only Sample 3)}$$

3. **Compute learner weight (α)**:

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\text{Error}}{\text{Error}}\right) = \frac{1}{2}\ln\left(\frac{0.8}{0.2}\right) \approx 0.693$$

4. **Update sample weights**:

- Increase weights for misclassified samples (Sample 3):

$$w_3 = 0.2 \times e^{\alpha_1} = 0.2 \times 2 = 0.4$$

- Decrease weights for correctly classified samples:

$$w_{\text{correct}} = 0.2 \times e^{-\alpha_1} = 0.2 \times 0.5 = 0.1$$

- **Normalize weights** (divide by total weight):

$$\text{Total weight} = 0.4 + 4(0.1) = 0.8 \quad \Rightarrow \quad \text{New weights} = \frac{w_i}{0.8}$$

Sample	New Weight
1	0.125
2	0.125
3	**0.5**
4	0.125
5	0.125

Iteration 2

1. **Find the best weak learner** (using updated weights):

- **Best stump**: ‘Age 35¿
- Predict **Yes** if ‘Age 35‘, else **No**.

2. **Calculate weighted error**:

Sample	Prediction	Actual	Correct?	Weight
1	Yes	Yes		0.125
2	Yes	No		0.125
3	Yes	Yes		0.5
4	No	Yes		0.125
5	No	No		0.125

$$\text{Error} = 0.125 + 0.125 = 0.25 \quad \text{(Samples 2 and 4)}$$

3. **Compute learner weight (α)**:

$$\alpha_2 = \frac{1}{2} \ln \left(\frac{1 - 0.25}{0.25} \right) \approx 0.549$$

4. **Update sample weights**:
   - Increase weights for misclassified samples (2 and 4):

$$w_2 = 0.125 \times e^{\alpha_2} \approx 0.125 \times 1.73 \approx 0.216$$

$$w_4 = 0.125 \times e^{\alpha_2} \approx 0.216$$

   - Decrease weights for correctly classified samples:

$$w_{\text{correct}} = 0.125 \times e^{-\alpha_2} \approx 0.125 \times 0.58 \approx 0.072$$

   - **Normalize weights** (total 0.216 + 0.216 + 0.072 + 0.072 + 0.072 = 0.648):

Sample	New Weight
1	0.111
2	**0.333**
3	0.111
4	**0.333**
5	0.111

Iteration 3

1. **Find the best weak learner**:
   - **Best stump**: ‘Income = Low¿
   - Predict **No** if ‘Income = Low‘, else **Yes**.

2. **Calculate weighted error**:

Sample	Prediction	Actual	Correct?	Weight
1	Yes	Yes		0.111
2	No	No		0.333
3	Yes	Yes		0.111
4	Yes	Yes		0.333
5	No	No		0.111

$$\text{Error} = 0 \quad \text{(All correct)}$$

Since the error is 0, we stop here.

Step 4: Combine Weak Learners

Final model = Weighted sum of predictions from all weak learners:

$$\text{Final Prediction} = \text{sign}\left(\alpha_1 \cdot \text{Prediction}_1 + \alpha_2 \cdot \text{Prediction}_2\right)$$

Step 5: Test the Model

For a test sample **Age = 32, Income = Medium**:

1. **Stump 1** (‘Income = High¿):
   - Income = Medium → Predict **No** ($\alpha = 0.693$).
2. **Stump 2** (‘Age 35¿):
   - Age = 32 35 → Predict **Yes** ($\alpha = 0.549$).

$$\text{Final Score} = (0.693 \times -1) + (0.549 \times +1) = -0.144$$

$$\text{Prediction} = \text{sign}(-0.144) = \textbf{No}$$

Key Takeaways

1. **Adaptive Weighting**: Misclassified samples get higher weights in subsequent iterations.
2. **Weak Learners**: Simple rules (stumps) are combined to form a strong classifier.
3. **Weighted Voting**: Final prediction depends on the weighted votes of all learners.

AdaBoost focuses on correcting errors sequentially, making it powerful for complex datasets.

AdaBoost Algorithm Explained with a Mathematical Example

February 2, 2025

0.1 AdaBoost in Machine learning

[5]:
```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,f1_score,recall_score
```

[27]:
```
data=load_breast_cancer()
```

[27]:
```
data=load_breast_cancer()
```

[28]:
```
df = pd.DataFrame(data=data.data, columns=data.feature_names)
```

[29]:
```
df["target"]=data.target
```

[30]:
```
df["target"].value_counts
```

[30]:
```
<bound method IndexOpsMixin.value_counts of 0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int64>
```

[31]:
```
df
```

[31]:
```
     mean radius  mean texture  ...  worst fractal dimension  target
0          17.99         10.38  ...                  0.11890       0
1          20.57         17.77  ...                  0.08902       0
2          19.69         21.25  ...                  0.08758       0
3          11.42         20.38  ...                  0.17300       0
```

```
4          20.29        14.34  ...                 0.07678       0
..           ...          ...  ...                     ...     ...
564        21.56        22.39  ...                 0.07115       0
565        20.13        28.25  ...                 0.06637       0
566        16.60        28.08  ...                 0.07820       0
567        20.60        29.33  ...                 0.12400       0
568         7.76        24.54  ...                 0.07039       1

[569 rows x 31 columns]
```

[32]:
```
X = df.drop(["target"], axis = 1)
y = df["target"]
```

[33]:
```
X
```

[33]:
```
     mean radius  mean texture  ...  worst symmetry  worst fractal dimension
0          17.99         10.38  ...          0.4601                  0.11890
1          20.57         17.77  ...          0.2750                  0.08902
2          19.69         21.25  ...          0.3613                  0.08758
3          11.42         20.38  ...          0.6638                  0.17300
4          20.29         14.34  ...          0.2364                  0.07678
..           ...           ...  ...             ...                      ...
564        21.56         22.39  ...          0.2060                  0.07115
565        20.13         28.25  ...          0.2572                  0.06637
566        16.60         28.08  ...          0.2218                  0.07820
567        20.60         29.33  ...          0.4087                  0.12400
568         7.76         24.54  ...          0.2871                  0.07039

[569 rows x 30 columns]
```

[42]:
```
X_train, X_test, y_train, y_test = train_test_split(
                            X, y,
                            # To split in balanced, let me set
                            # stratify=y,
                            # For reproducible output
                            random_state=0)
```

[43]:
```
X_train.shape
```

[43]:
```
(426, 30)
```

[44]:
```
X_test.shape
```

[44]:
```
(143, 30)
```

[45]:
```
print(y_train.shape)
print(y_test.shape)
```

```
(426,)
(143,)
```

[50]:
```
model = AdaBoostClassifier(
    n_estimators=100,
    learning_rate=0.5,
    random_state=42
)
```

[51]:
```
model.fit(X_train,y_train)
```

```
/home/mohsin/Documents/notebookvirtual/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

[51]:
```
AdaBoostClassifier(learning_rate=0.5, n_estimators=100, random_state=42)
```

[55]:
```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

[62]:
```
ada_train = accuracy_score(y_train, y_train_pred)
ada_test = accuracy_score(y_test, y_test_pred)
```

[71]:
```
print(f"test data accuracy {ada_train: .2f}")
print(f"test data accuracy {ada_test: .2f}")
```

```
test data accuracy  1.00
test data accuracy  0.94
```

[ ]: