```python
from model_base import ModelBase
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, roc_auc_score, precision_recall_curve,confusion_matrix
import numpy as np

class AdaBoostModel(ModelBase):
    def __init__(self):
        super().__init__()
        # Default weak learner
        base_estimator = DecisionTreeClassifier(max_depth=1)
        self.model = AdaBoostClassifier(estimator=base_estimator, random_state=42)

    def train(self, X_train, y_train):
        print("Training AdaBoost Classifier...")
        self.model.fit(X_train, y_train)

    def tune_hyperparameters(self, X_train, y_train):
        print("Tuning hyperparameters for AdaBoost...")
        param_grid = {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1, 1],
            'estimator': [
                DecisionTreeClassifier(max_depth=1),
                DecisionTreeClassifier(max_depth=2)
            ]
        }

        grid = GridSearchCV(
            AdaBoostClassifier(random_state=42),
            param_grid,
            cv=5,
            n_jobs=-1
        )
        grid.fit(X_train, y_train)
        print("Best Params:", grid.best_params_)
        self.model = grid.best_estimator_

    def evaluate_with_threshold(self, X_test, y_test, threshold=0.5):
        y_probs = self.model.predict_proba(X_test)[:, 1]
        y_pred = (y_probs >= threshold).astype(int)

        print(f"Evaluation with Threshold = {threshold}")
        print(classification_report(y_test, y_pred))
        print("ROC-AUC Score:", roc_auc_score(y_test, y_probs))
        cm = confusion_matrix(y_test, y_pred)
        print("Confusion Matrix:\n", cm)
        return y_pred, y_probs

    def find_best_threshold(self, X_val, y_val):
        y_probs = self.model.predict_proba(X_val)[:, 1]
        precision, recall, thresholds = precision_recall_curve(y_val, y_probs)

        f1_scores = 2 * (precision * recall) / (precision + recall + 1e-8)
        best_index = np.argmax(f1_scores)
        best_threshold = thresholds[best_index]
        print(f"Best threshold based on F1-score: {best_threshold:.2f}")
        return best_threshold
```