```python
import numpy as np
from metrics import sum_squared_errors, sum_squared_residuals


class LinearRegression: # HML p 106
    def __init__(self):
        self.best_theta = None
        self.intercept_ = None
        self.coef_ = None
        self.is_fitted = False

    def fit(self, features: np.array, targets: np.array) -> None:
        X = np.copy(features)
        y = np.copy(targets)
        X = np.c_[np.ones((len(X), 1)), X] # add x0 = 1 to each instance
        self.best_theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
        self.intercept_ = self.best_theta[0]
        self.coef_ = self.best_theta[1:]
        self.is_fitted = True

    def predict(self, features: np.array) -> np.array:
        predictions = np.c_[np.ones((len(features), 1)), features] # add x0 = 1 to each instance
        return predictions.dot(self.best_theta)

    def score(self, features:np.array, targets: np.array) -> float:
        if not self.is_fitted:
            raise ValueError('Linear regression model must first be fitted!')
        predicted_targets = self.predict(features)
        return (1 - (sum_squared_residuals(targets,predicted_targets) /
        sum_squared_errors(targets,predicted_targets)))

def accuracy_score(
        true_labels,
        predicted_labels,
        normalize: bool = True,
    ) -> float:
    n_samples_true, n_samples_predicted = len(true_labels), len(predicted_labels)
    if n_samples_true != n_samples_predicted:
        raise ValueError()
    n_correct_predictions = 0
    for i in range(len(true_labels)):
        if  true_labels[i] == predicted_labels[i]:
            n_correct_predictions += 1
    if normalize:
        return n_correct_predictions / n_samples_true
    else:
        return n_correct_predictions
```