Загрузка и изучение датасета

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor,
AdaBoostRegressor, GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

# Загрузка данных
data = load_breast_cancer()
X = data.data
y = data.target

# Преобразование в DataFrame для удобства работы
df = pd.DataFrame(data=X, columns=data.feature_names)
df['target'] = y

# Удаление или заполнение пропусков
imp = SimpleImputer(strategy="mean")
X_imputed = imp.fit_transform(X)

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y,
test_size=0.3, random_state=42)

# Масштабирование
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Проверка на наличие пропусков
missing_data = df.isnull().sum()
print(missing_data)
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
```

```
radius error                0
texture error               0
perimeter error             0
area error                  0
smoothness error            0
compactness error           0
concavity error             0
concave points error        0
symmetry error              0
fractal dimension error     0
worst radius                0
worst texture               0
worst perimeter             0
worst area                  0
worst smoothness            0
worst compactness           0
worst concavity             0
worst concave points        0
worst symmetry              0
worst fractal dimension     0
target                      0
dtype: int64
```

```
# Проверка на наличие категориальных признаков
categorical_features = df.select_dtypes(include=['object']).columns
print(categorical_features)
```

```
Index([], dtype='object')
```

Обучение моделей

```
bagging_params = {
    'n_estimators':[5, 10, 50, 100]
}

tree_params = {
    'n_estimators':[50, 100, 150, 200]
}

adaboost_params = {
    'n_estimators':[30, 50, 100, 150]
}

gradient_params = {
    'n_estimators':[50, 100, 150, 200]
}

# Бэггинг

RANDOM_STATE=123
```

```
grid_search =
GridSearchCV(estimator=BaggingRegressor(random_state=RANDOM_STATE),
param_grid=bagging_params, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f"Лучшие параметры: {grid_search.best_params_}")
bagging = grid_search.best_estimator_
```

```
Лучшие параметры: {'n_estimators': 50}
```

```
grid_search =
GridSearchCV(estimator=RandomForestRegressor(random_state=RANDOM_STATE
), param_grid=tree_params, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f"Лучшие параметры: {grid_search.best_params_}")
random_forest = grid_search.best_estimator_
```

```
Лучшие параметры: {'n_estimators': 200}
```

```
grid_search =
GridSearchCV(estimator=AdaBoostRegressor(estimator=DecisionTreeRegress
or(), random_state=RANDOM_STATE), param_grid=adaboost_params, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f"Лучшие параметры: {grid_search.best_params_}")
adaboost = grid_search.best_estimator_
```

```
Лучшие параметры: {'n_estimators': 50}
```

```
grid_search =
GridSearchCV(estimator=AdaBoostRegressor(random_state=RANDOM_STATE),
param_grid=adaboost_params, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f"Лучшие параметры: {grid_search.best_params_}")
adaboost_limited_tree_depth = grid_search.best_estimator_
```

```
Лучшие параметры: {'n_estimators': 50}
```

```
grid_search =
GridSearchCV(estimator=GradientBoostingRegressor(random_state=RANDOM_S
TATE), param_grid=gradient_params, cv=3)
grid_search.fit(X_train_scaled, y_train)

print(f"Лучшие параметры: {grid_search.best_params_}")
gradient_boosting = grid_search.best_estimator_
```

```
Лучшие параметры: {'n_estimators': 100}
```

Оценка моделей

```
y_pred_bagging = bagging.predict(X_test_scaled)
y_pred_rf = random_forest.predict(X_test_scaled)
y_pred_adaboost = adaboost.predict(X_test_scaled)
y_pred_adaboost_limited =
adaboost_limited_tree_depth.predict(X_test_scaled)
y_pred_gb = gradient_boosting.predict(X_test_scaled)

# MAE
print(f"Bagging: {mean_absolute_error(y_test, y_pred_bagging):.4f}")
print(f"Random Forest: {mean_absolute_error(y_test, y_pred_rf):.4f}")
print(f"AdaBoost: {mean_absolute_error(y_test, y_pred_adaboost):.4f}")
print(f"AdaBoost (tree depth = 3): {mean_absolute_error(y_test,
y_pred_adaboost_limited):.4f}")
print(f"Gradient Boosting: {mean_absolute_error(y_test,
y_pred_gb):.4f}")
```

```
Bagging: 0.0751
Random Forest: 0.0760
AdaBoost: 0.0409
AdaBoost (tree depth = 3): 0.0820
Gradient Boosting: 0.0789
```

```
# R^2
print(f"Bagging: {r2_score(y_test, y_pred_bagging):.4f}")
print(f"Random Forest: {r2_score(y_test, y_pred_rf):.4f}")
print(f"AdaBoost: {r2_score(y_test, y_pred_adaboost):.4f}")
print(f"AdaBoost (tree depth = 3): {r2_score(y_test,
y_pred_adaboost_limited):.4f}")
print(f"Gradient Boosting: {r2_score(y_test, y_pred_gb):.4f}")
```

```
Bagging: 0.8498
Random Forest: 0.8592
AdaBoost: 0.8241
AdaBoost (tree depth = 3): 0.8946
Gradient Boosting: 0.8678
```

Вывод

AdaBoost без ограничения глубины деревьев имеет наименьшее значение MAE, но наименьшее значение R^2, что может указывать на хорошую точность предсказаний, но плохое объяснение общей дисперсии данных.

AdaBoost с ограниченной глубиной деревьев (tree depth = 3) показывает наибольшее значение R^2 и наибольшее значение MAE. Это говорит о том, что модель хорошо объясняет вариации в данных, но делает больше ошибок в среднем. Возможно, ограничение глубины деревьев уменьшило переобучение, что улучшило обобщающую способность модели.

Остальные модели показали примерно одинаковый результат.