Министерство науки и высшего образования
Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования «Рыбинский государственный
авиационный технический университет имени П. А. Соловьева»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ УПРАВЛЕНИЯ

Математического и программного обеспечения электронных
вычислительных средств

ОТЧЁТ

по дисциплине:
«Методы и алгоритмы анализа данных»
на тему:
«Решающие деревья»

Выполнил: студент группы ИВМ-24 Морозов А. А.
Руководитель: ассистент Вязниковцев Д. А.

Рыбинск 2024

Содержание

Цель работы

Целями данной лабораторной работы являются:

- изучение теории о решающих деревьях;
- первичная обработка данных;
- построение решающих деревьев.

1 Первичная обработка данных

В начале необходимо загрузить датасет о физических упражнениях и их эффективности *linnerud* из библиотеки *sklearn*:

```
def preparation():
    linnerud = load_linnerud()
    print(linnerud.DESCR)
    print(f"Ключи датасета: {linnerud.keys()}")
    print(f"D.keys() -> a set-like object providing a view on D's keys")
    print(f"Признаки датасета: {linnerud.feature_names, linnerud.target_names}")
    # X: pd.DataFrame = linnerud.data
    X = pd.DataFrame(data=linnerud.data, columns=linnerud.feature_names)
    print(X)
    print(f"Последние 5 значений X:\n{X[-5:]}")
    print(f"Размер X: {len(X)}")
    y = linnerud.target
    print(f"Последние 5 значений y:\n{y[-5:]}")
    print(f"Размер y: {len(y)}")
    return linnerud, X, y
```

После выполнения кода были созданы *dataframe* и *ndarray* и в консоли отобразилась некоторая базовая информация о датасете.

Выведем график распределения целевой переменной (рисунок 1):

```
def raspredelenie(y):
    plt.title('... distribution')
```

```
plt.xlabel('...')
plt.ylabel('# samples')
plt.hist(y, bins=20)
plt.show()
```

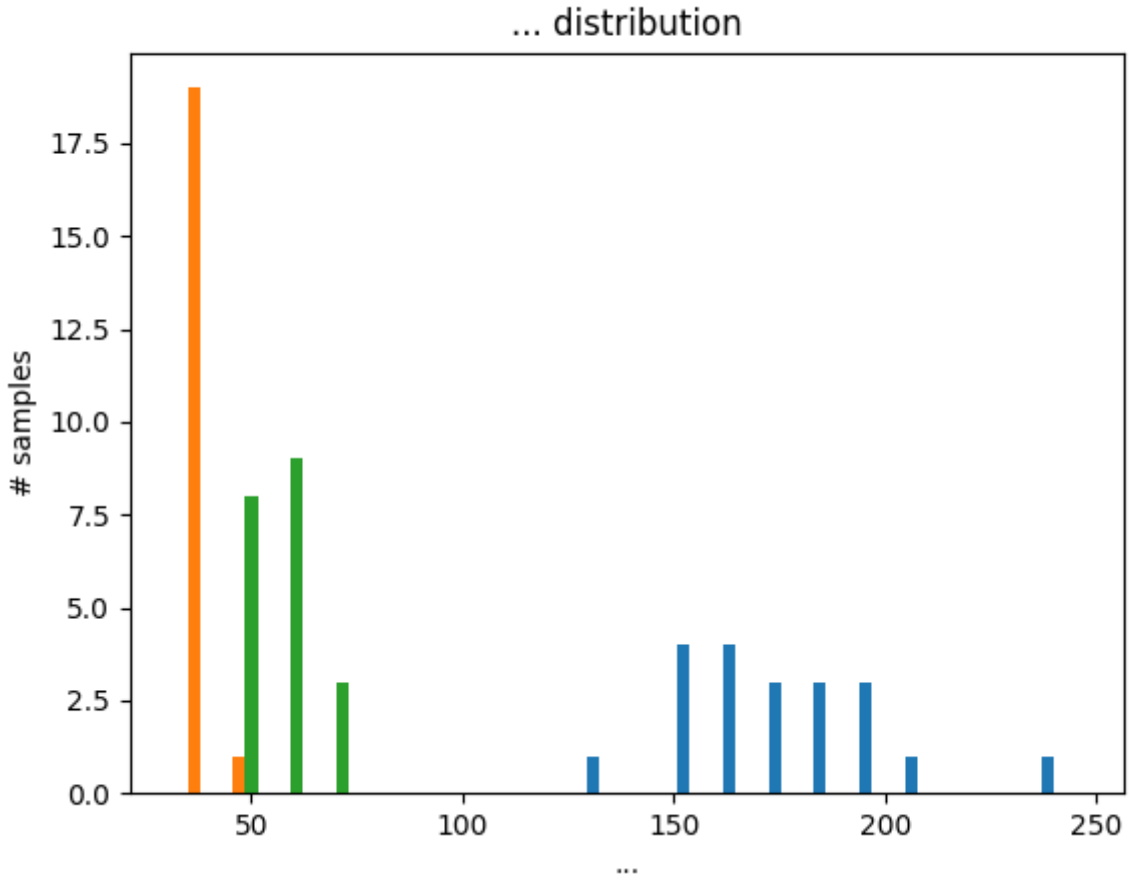

Рисунок 1.1 – График распределения целевой переменной

Теперь разобьём данные на тестовую и обучающую выборки в соотношении 1 к 4:

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.25, random_state=13)

В данных выборках у X и y по 20 записей в каждом. Каждая запись содержит 3 числа.

Теперь можно приступать к построению решающего дерева.

2 Критерий ошибки

Для построения оптимального решающего дерева необходимо исходить из того, что критерий ошибки должен стремится к 0:

$$Q(R_m, j, t) = \frac{|R_l|}{|R_m|} H(R_l) + \frac{|R_r|}{|R_m|} H(R_r) \to \min_{j,t} \quad (2.1)$$

Для расчёта критерия ошибки воспользуемся кодом:

```
def H(R, y):
    return y[R.index].var(ddof=0)


def split_node(R_m, feature, t):
    left = R_m[R_m[feature] <= t]
    right = R_m[R_m[feature] > t]
    return left, right


def q_error(R_m, feature, t, y):
    left, right = split_node(R_m, feature, t)
    return (len(left) / len(R_m) * H(left,y)+len(right)/len(R_m)*
H(right, y))
```

С помощью данных функций получим критерий ошибки для какого-либо признака. Используя его, можно найти оптимальное разбиение вершины по заданному признаку:

```
def zadanie_2_3(X_train, y):
```

```
    feature = 'Chins'
    feature_values = np.unique(X_train[feature])
    print(feature_values, len(feature_values))
    Q_array = list(map(lambda x: q_error(X_train, feature, x, y), feature_values))
    print(Q_array)
    nan_value = feature_values[np.where(np.isnan(Q_array))]
    plt.figure(figsize=(10, 6))
    plt.plot(feature_values, Q_array, marker='o', linestyle='-')
    plt.xlabel('Порог')
    plt.ylabel('Значение ошибки')
    plt.title(f'Feature {feature}')
    plt.grid(True)
    plt.show()
    return feature, nan_value
```

После выполнения кода получим график критерия ошибки в зависимости от порога признака (рисунок 2).

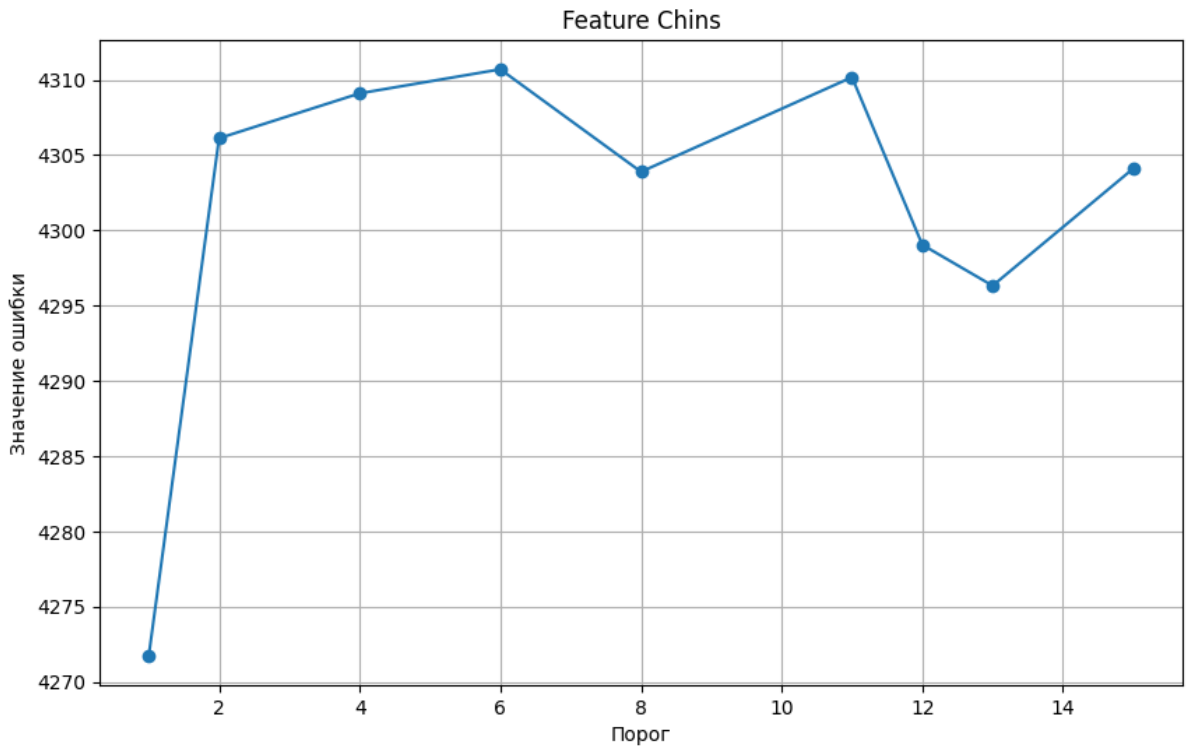

Рисунок 2.1 – График критерия ошибки признака *Chins*

Теперь найдём признак с наименьшим значением ошибки:

```
def zadanie_2_4(X_train):
    results = []
    for f in X_train.columns:
        t, Q_array = get_optimal_split(X_train, f, y)
        # print(t, Q_array, Q_array[np.argmin(Q_array)])
        results.append((f, t, Q_array[np.argmin(Q_array)]))
    results = sorted(results, key=lambda x: x[2])
    print(f"Результаты 2.4: {results}")
    results_df = pd.DataFrame(results, columns=['feature', 'optimal
t', 'min Q error'])
    optimal_feature, optimal_t, optimal_error = results[0]
    print(results_df)
```

```
    _, optimal_Q_array = get_optimal_split(X_train, optimal_feature, y)
    plt.figure(figsize=(10, 6))
    # print(X_train[optimal_feature], optimal_Q_array)
    # print(np.unique(X_train[optimal_feature]))
    # print(nan_value)
    plt.plot(np.delete(np.unique(X_train[optimal_feature]),
                       np.where(np.unique(X_train[optimal_feature]) == nan_value)[0][0]), optimal_Q_array, marker='o',
             linestyle='-')
    plt.xlabel('Порог')
    plt.ylabel('Значение ошибки')
    plt.title(f'Feature {feature}')
    plt.grid(True)
    plt.show()
    return optimal_feature, optimal_t, optimal_error, X_train
```

В результате выполнения кода выше в консоли выведутся строчки изображённые на рисунке 3.

```
Результаты 2.4: [('Chins', 1.0, 4271.754497354497), ('Situps', 50.0, 4271.754497354497), ('Jumps', 120.0, 4306.886772486772)]
  feature  optimal t  min Q error
0   Chins        1.0  4271.754497
1  Situps       50.0  4271.754497
2   Jumps      120.0  4306.886772
```

Рисунок 2.2 – Вывод в консоль

Из вывод в консоль следует то, что признак *Chins* со значением разбиения 1 имеет наименьшую ошибку. График критерия ошибки для признака *Chins* изображён на рисунке 2.

Изобразим разбиение визуально (рисунок 4). Для этого построим диаграмму рассеяния целевой переменной в зависимости от значения найденного признака. Далее изобразим вертикальную линию, соответствующую порогу разбиения.

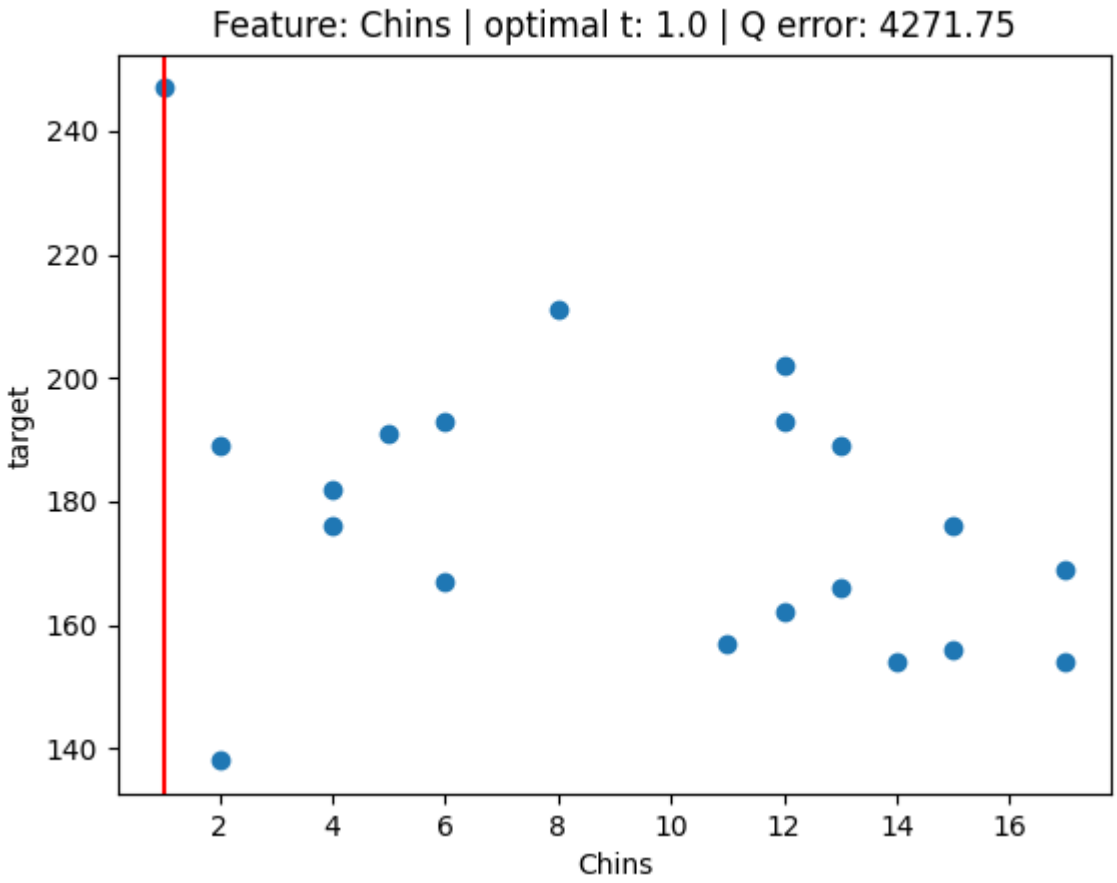

Рисунок 2.3 – График разбиения

3 Решающее дерево

Создадим решающее дерево и выведем (рисунок 5) его с помощью кода:

```
def zadanie_3_1(X_train, X_test, y_train, y_test):
    from sklearn.tree import DecisionTreeRegressor
    dt = DecisionTreeRegressor(max_depth=3, random_state=13)
    dt.fit(X_train, y_train)

    from sklearn.tree import plot_tree
    plot_tree(dt, feature_names=X.columns, filled=True, rounded=True)
    plt.show()
    return dt
```

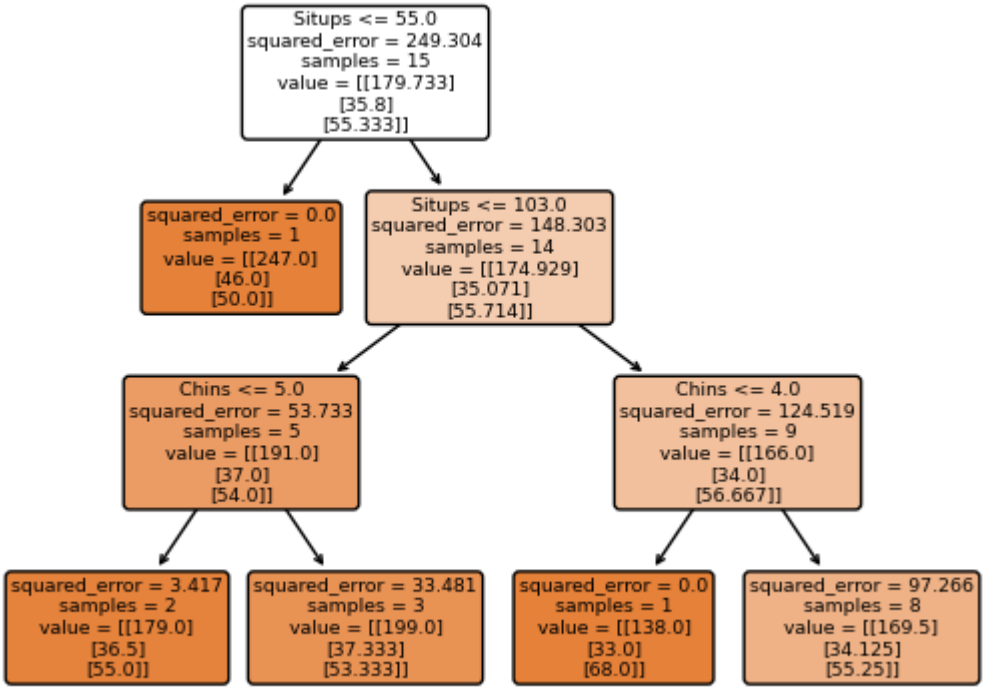

Рисунок 3.1 – Решающее дерево

При выполнении кода с рисунка 6 будут выводиться графики (рисунок 7) с зависимостью значения ошибки от количества слоёв дерева и минимального количества примеров в листе.

```python
def zadanie_3_2(X_train, X_test, y_train, y_test, dt):
    mean_squared_error(y_test, dt.predict(X_test))

    max_depth_array = range(2, 20)
    mse_array = []
    for max_depth in max_depth_array:
        dt = DecisionTreeRegressor(max_depth=max_depth, random_state=13)
        dt.fit(X_train, y_train)
        mse_array.append(mean_squared_error(y_test, dt.predict(X_test)))
    plt.plot( *args: max_depth_array, mse_array)
    plt.title('Dependence of MSE on max depth')
    plt.xlabel('max depth')
    plt.ylabel('MSE')
    plt.show()

    pd.DataFrame({
        'max_depth': max_depth_array,
        'MSE': mse_array
    }).sort_values(by='MSE').reset_index(drop=True)

    min_samples_leaf_array = range(1, 20)
    mse_array = []
    for min_samples_leaf in min_samples_leaf_array:
        dt = DecisionTreeRegressor(max_depth=6, min_samples_leaf=min_samples_leaf, random_state=13)
        dt.fit(X_train, y_train)
        mse_array.append(mean_squared_error(y_test, dt.predict(X_test)))
    plt.plot( *args: min_samples_leaf_array, mse_array)
    plt.title('Dependence of MSE on min samples leaf')
    plt.xlabel('min samples leaf')
    plt.ylabel('MSE')
    plt.show()

    min_samples_split_array = range(2, 20)
    mse_array = []
    for min_samples_split in min_samples_split_array:
        dt = DecisionTreeRegressor(max_depth=6, min_samples_split=min_samples_split, random_state=13)
        dt.fit(X_train, y_train)
        mse_array.append(mean_squared_error(y_test, dt.predict(X_test)))
    plt.plot( *args: min_samples_split_array, mse_array)
    plt.title('Dependence of MSE on min samples split')
    plt.xlabel('min samples split')
    plt.ylabel('MSE')
    plt.show()

    from sklearn.tree import plot_tree
    dt = DecisionTreeRegressor(max_depth=6, random_state=13)
    dt.fit(X_train, y_train)
    plot_tree(dt, feature_names=X.columns, filled=True, rounded=True)
    plt.show()

    mean_squared_error(y_test, dt.predict(X_test))

    print(f"dt.feature_importances_: {dt.feature_importances_}")

    pd.DataFrame({
        'feature': X.columns,
        'importance': dt.feature_importances_
    }).sort_values(by='importance', ascending=False).reset_index(drop=True)
```

Рисунок 3.2 – Код для оценки качества обучения

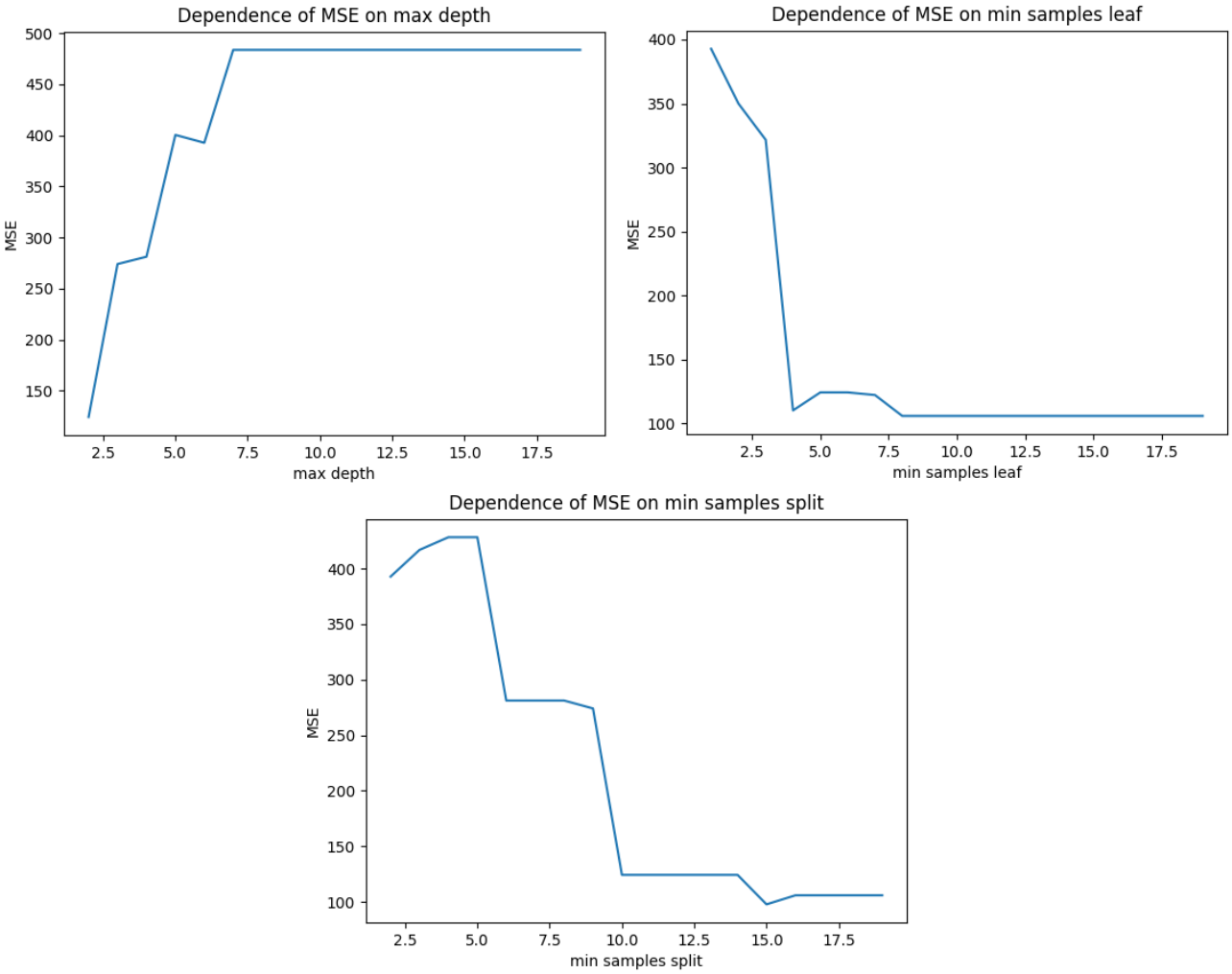

Рисунок 3.3 – Графики оценки качества обучения

Вывод

Таким образом, построение решающего дерева – задача нахождения минимального критерия ошибки и оптимизации множества других параметров.

Решающие деревья являются мощным инструментом для решения задач классификации и регрессии. Однако для повышения их эффективности в реальных задачах рекомендуется использовать ансамблевые методы, такие как случайные леса (*Random Forest*) или градиентный бустинг (*Gradient Boosting*), которые уменьшают недостатки одиночных деревьев.

Эта работа продемонстрировала важность грамотной настройки гиперпараметров и визуального анализа модели для достижения высокого качества предсказаний.