Лабораторная работа №4. Логические методы классификации

Цель лабораторной работы: изучение принципов построения информационных систем с использованием логических методов классификации.

Основные задачи:

- освоение технологии внедрения алгоритмов на основе решающих списков в приложения;
- освоение технологии внедрения алгоритмов на основе решающих деревьев в приложения;
- изучение параметров логической классификации;
- освоение модификаций логических методов классификации.

Ход выполнения индивидуального задания:

1. Построение модели классификации на основе дерева классификации

1.1 Построение логического классификатора с заданием max_depth (максимальной глубины) и max_features (максимального количества признаков) пользователем и визуализация дерева решений

Первым делом импортируем все необходимые библиотеки для выполнения лабораторной работы, а также загрузим наш набор данных с помощью **pandas**:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import tree


data_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/gla
```

```
columns = [
    "Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type"
]

data = pd.read_csv(data_path, names=columns)
data.drop(columns=["Id"], inplace=True)
```

Признак **Id** удаляем из набора данных из-за ненадобности.

Построим классификатор, используя библиотеку **scikit,** а также оценим его точность при помощи метода **hold-out**:

```
X_train, X_holdout, y_train, y_holdout = train_test_split(data[['RI', 'Na'
                                                                data['Type'],
                                                                test_size=0.3,
                                                                random_state=12,
                                                                stratify=data['T

tree_model = DecisionTreeClassifier(max_depth=5,
                                    random_state=21,
                                    max_features=2)

tree_model.fit(X_train, y_train)

tree_pred = tree_model.predict(X_holdout)
accuracy = accuracy_score(y_holdout, tree_pred)
print("Оценка точности классификатора методом hold-out:", accuracy)
```

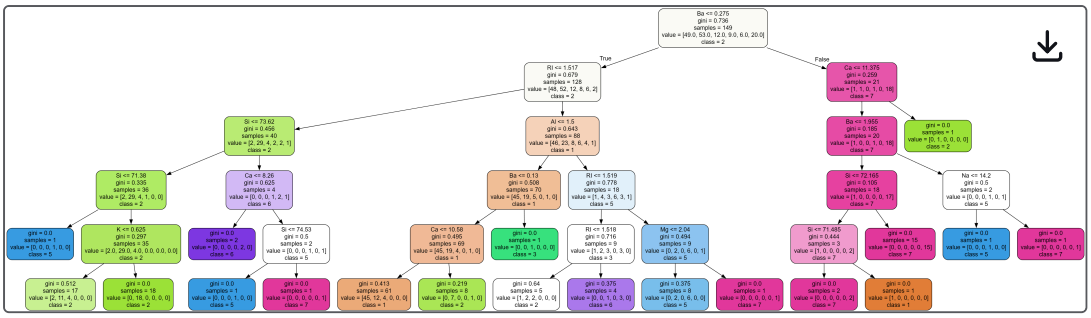
Первым делом мы разбили наш набор данных на **тестовую** и **обучающую** выборки при помощи функции `train_test_split`. Далее мы создаём классификатор, основанный на дереве классификации, с параметрами: **max_depth = 5** и **max_features = 2**. Выбор данных значений **не обосновывается**. После чего происходит обучение модели и вычисление точности методом **hold-out.**

```
Оценка точности классификатора методом hold-out: 0.6461538461538463
```

Для того чтобы визуализировать дерево решений для заданных параметров, воспользуемся следующим кодом:

```
tree.export_graphviz(tree_model,
                     feature_names=data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K'
                     class_names=[str(c) for c in data['Type'].unique()],
                     out_file='glass_tree_1.dot',
                     filled=True, rounded=True)
```

Дерево экспортируется в формат **.dot**. Чтобы можно было посмотреть на изображение в формате **.png**, воспользуемся сторонним сервисом **Graphviz Online**. Как результат получаем следующее изображение:

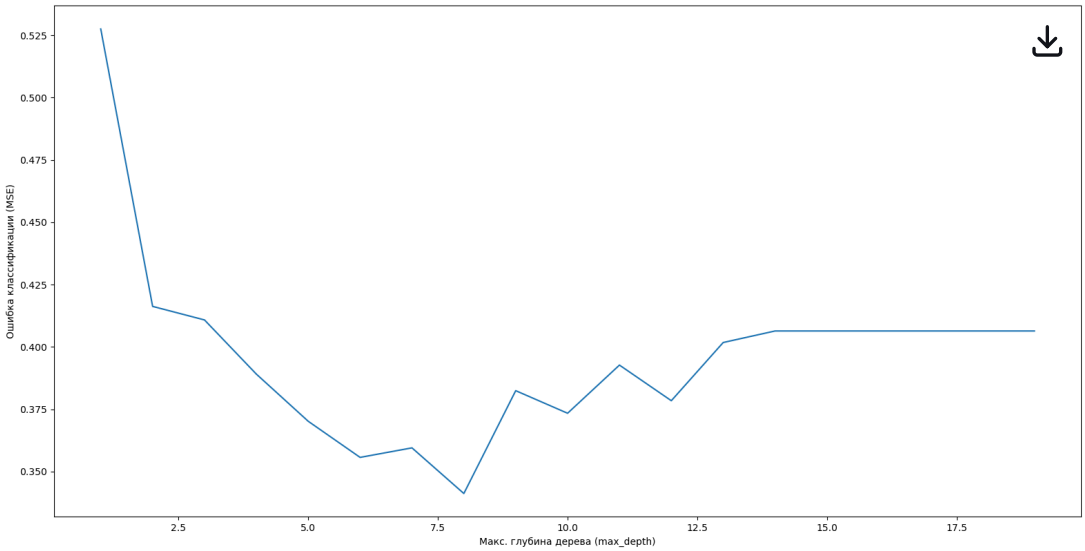


1.2 Вычисление оценки cross validation (MSE) для различных значений max_depth и построение графика зависимости

Реализация вычисления оценки cross validation (MSE) для различных значений max_depth в коде:

```
d_list = list(range(1,20))
cv_scores = []

for d in d_list:
    tree_model = DecisionTreeClassifier(max_depth=d,
                                        random_state=21,
                                        max_features=2)
    scores = cross_val_score(tree_model,
                             data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca'
                             data['Type'],
                             cv=9,
                             scoring='accuracy')
    cv_scores.append(scores.mean())

MSE = [1-x for x in cv_scores]

plt.figure(1)
plt.plot(d_list, MSE)
plt.xlabel('Макс. глубина дерева (max_depth)');
plt.ylabel('Ошибка классификации (MSE)')

d_min = min(MSE)
all_d_min = []
for i in range(len(MSE)):
    if MSE[i] <= d_min:
        all_d_min.append(d_list[i])

print('Оптимальные значения max_depth: ', all_d_min)
```

Значение **max_depth** будет варьироваться в пределах **от 1 до 19**, при этом значение **max_features** будет оставаться фиксированным, а именно **2**. Как результат получаем график зависимости **ошибки классификации** от максимальной глубины дерева и **оптимальное значение** max_depth:



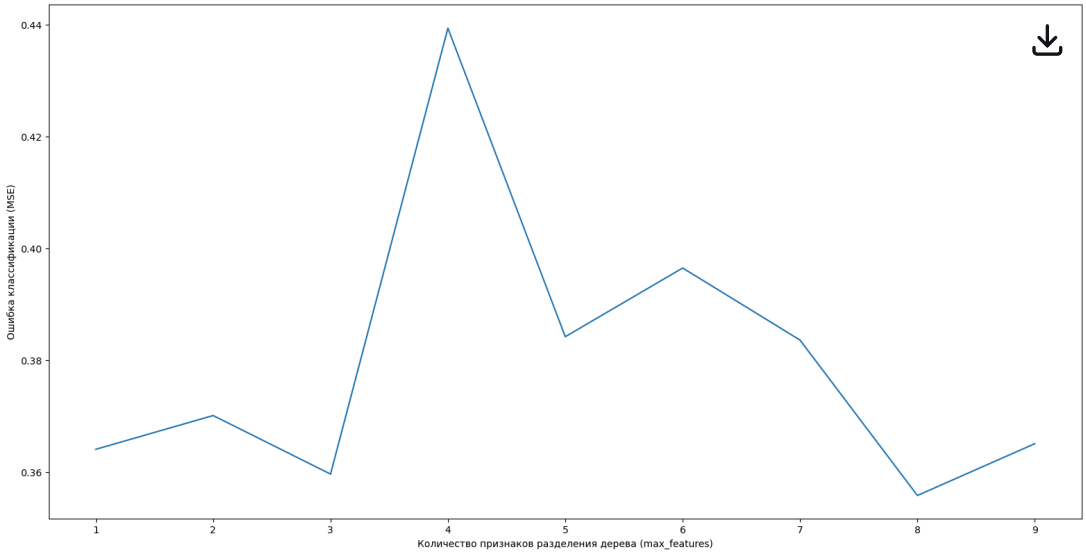

```
Оптимальные значения max_depth:  [8]
```

1.3 Вычисление оценки cross validation (MSE) для различных значений max_features и построение графика зависимости

Для вычисления оценки cross validation (MSE) для различных значений max_features используем следующую часть программы:

```
f_list = list(range(1, 10))
cv_scores_features = []

for f in f_list:
    tree_model = DecisionTreeClassifier(max_depth=5,
                                        random_state=21,
                                        max_features=f)
    scores = cross_val_score(tree_model,
                             data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca'
                             data['Type'],
                             cv=9,
                             scoring='accuracy')
    cv_scores_features.append(scores.mean())

MSE_features = [1-x for x in cv_scores_features]

plt.figure(2)
plt.plot(f_list, MSE_features)
plt.xlabel('Количество признаков разделения дерева (max_features)');
plt.ylabel('Ошибка классификации (MSE)')

f_min = min(MSE_features)
all_f_min = []
for i in range(len(MSE_features)):
    if MSE_features[i] <= f_min:
        all_f_min.append(f_list[i])

print('Оптимальные значения max_features: ', all_f_min)
```

Значение **max_features** уже будет изменяться в пределах **от 1 до 9**, а переменная **max_depth** останется неизменной и будет равна **5**. По итогу также получаем график ошибки классификации и **оптимальное значение** max_features:




```
Оптимальные значения max_features: [8]
```

1.4 Вычисление оптимального значения max_depth и max_features. Графическое представление лучшего дерева

Чтобы вычислить оптимальные значения для параметров **max_depth** и **max_features** воспользуемся следующим кодом:

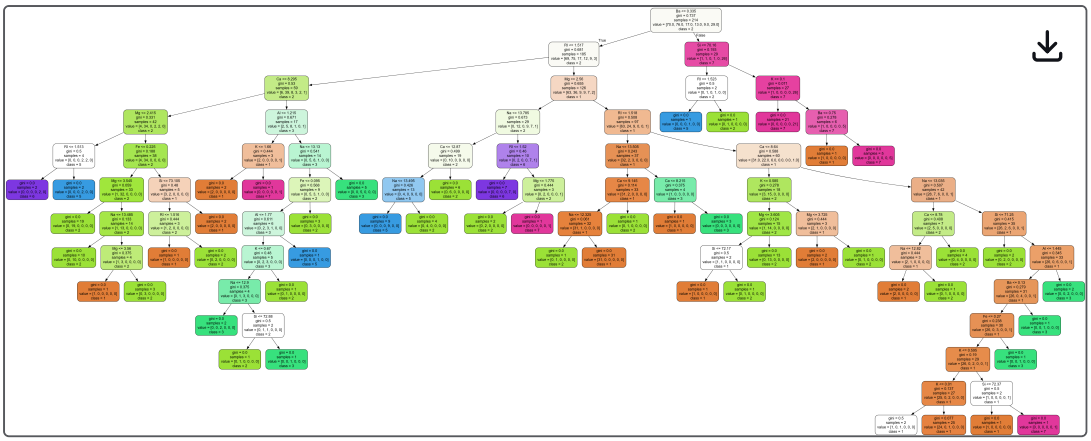
```
dtc = DecisionTreeClassifier(max_depth=10, random_state=21, max_features=2
tree_params = { 'max_depth': range(1,20), 'max_features': range(1,10) }
tree_grid = GridSearchCV(dtc, tree_params, cv=9, verbose=True, n_jobs=-1)
tree_grid.fit(data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],
print('Лучшее сочетание параметров: ', tree_grid.best_params_)
print('Лучшие баллы cross validation: ', tree_grid.best_score_)
```

С помощью строчки `tree_grid.best_params_` мы определим **лучшее** сочетание для двух этих параметров. Как результат мы получаем следующий вывод:

```
Лучшее сочетание параметров:  {'max_depth': 12, 'max_features': 3}
Лучшие баллы cross validation:  0.6865942028985508
```

Также построим лучшее дерево, используя `tree_grid.best_estimator_` в функции `tree.export_graphviz`:

```
tree.export_graphviz(tree_grid.best_estimator_,
                                feature_names=data[['RI', 'Na', 'Mg', 'Al'
                                class_names=[str(c) for c in data['Type'].
                                out_file='glass_tree_2.dot',
                                filled=True, rounded=True)
```

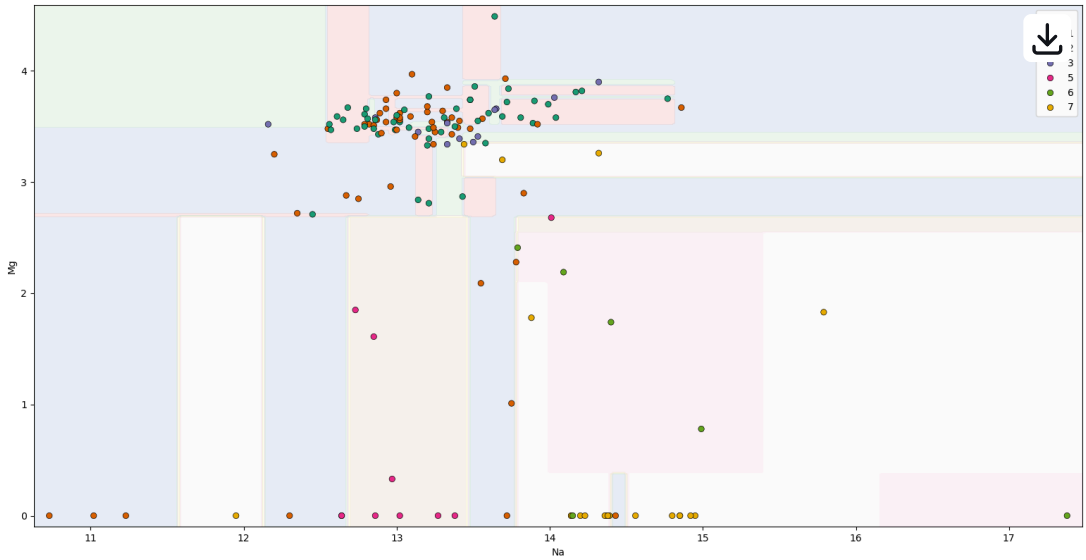


1.5 Демонстрация полученного классификатора при оптимальных значениях max_depth и max_features

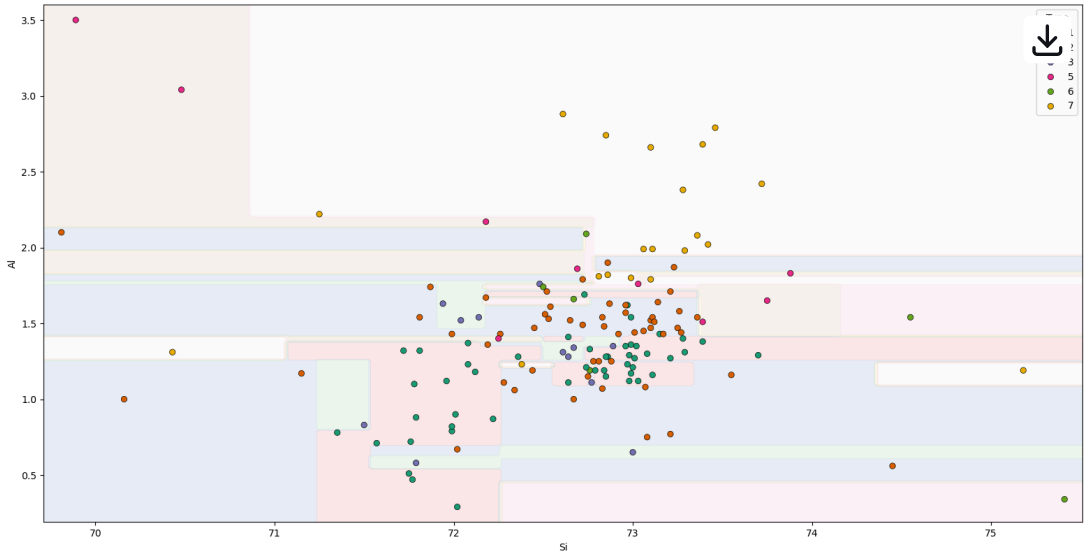
В качестве оптимального значения **max_depth** выберем **12**, а для **max_features** будет равен **3**. Для визуализации **решающих границ и распределения классов** был использован следующий код:

```
X = data[['Ca', 'Na']]
y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
tree_model = DecisionTreeClassifier(max_depth=12, random_state=21, max_fea
tree_model.fit(X_train, y_train)

x_min, x_max = X.iloc[:, 0].min() - 0.1, X.iloc[:, 0].max() + 0.1
y_min, y_max = X.iloc[:, 1].min() - 0.1, X.iloc[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_

Z = tree_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(3)
plt.contourf(xx, yy, Z, alpha=0.3, cmap='Pastel1')
sns.scatterplot(x=X_train.iloc[:, 0], y=X_train.iloc[:, 1], hue=y_train, p
plt.xlabel('Ca')
plt.ylabel('Na')
```

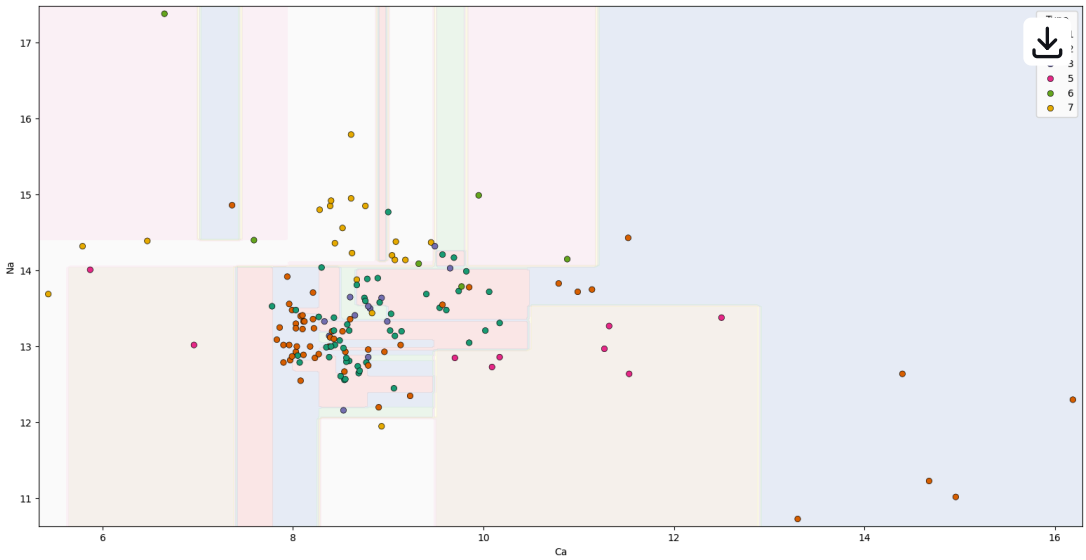
Классификационные границы для **натрия** и **магния**:



Классификационные границы для **кремния** и **алюминия**:



Классификационные границы для **кальция** и **натрия**:



Контрольные вопросы

1. Дерево решений строится путем рекурсивного разбиения пространства признаков на основе критериев информативности. Каждый узел представляет собой проверку условия (например, сравнение значения признака с порогом), а ветви указывают возможные исходы. Разбиение продолжается до тех пор, пока не будет достигнут терминальный критерий (например, минимальный размер листа или отсутствие значительного прироста информации).
2. Информативность признака оценивается на основе статистических методов, например, используя критерий χ^2 (хи-квадрат), дисперсионный анализ (ANOVA) или коэффициент взаимной информации. Эти методы позволяют количественно оценить, насколько сильна связь между признаком и целевой переменной.
3. В энтропийном подходе информативность измеряется уменьшением неопределенности (энтропии) при разбиении данных по определенному признаку. Например, используется прирост информации (Information Gain) или коэффициент Джини
4. Это мера, оценивающая информативность признака при наличии более чем двух классов. Она применяется в задачах многоклассовой классификации, например, с использованием обобщенного критерия Джини или кросс-энтропии.
5. Бинаризация заключается в разбиении непрерывного признака на две категории (например, больше или меньше порога). Алгоритм:
   - Определение возможных порогов (например, медиана, квантили или методом оптимального разбиения).
   - Разделение значений на две группы (меньше/больше порога).
   - Выбор оптимального порога на основе критерия информативности.
6. Порядок поиска:
   - Определение базовых условий (например, значения признаков или их диапазоны).
   - Комбинирование условий в логические выражения.
   - Оценка качества закономерностей по заданному критерию (например, по точности предсказаний или приросту информации).
   - Фильтрация нерелевантных закономерностей.