Front matter

title: "Отчёт по лабораторной работе №4" author: "Тимур Каримов"

Generic options

lang: ru-RU toc-title: "Содержание"

Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true toc-depth: 2 lof: true lot: true fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt

Polyglossia

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

Babel

babel-lang: russian babel-otherlangs: english

Fonts

mainfont: IBM Plex Serif romanfont: IBM Plex Serif sansfont: IBM Plex Sans monofont: IBM Plex Mono mathfont: STIX Two Math mainfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 romanfontoptions: Ligatures=Common,Ligatures=TeX,Scale=0.94 sansfontoptions: Ligatures=Common,Ligatures=TeX,Scale=MatchLowercase,Scale=0.94 monofontoptions: Scale=MatchLowercase,Scale=0.94,FakeStretch=0.9

Biblatex

biblatex: true biblio-style: "gost-numeric" biblatexoptions:

- parentracker=true
- backend=biber
- hyperref=auto
- language=auto
- autolang=other*
- citestyle=gost-numeric

Pandoc-crossref

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц" lolTitle: "Листинги"

Misc

indent: true header-includes:

- \usepackage{indentfirst}
- \usepackage{float}
- \floatplacement{figure}{H}

Цель работы

Изучить метод обучения: деревья решения, научиться строить деревья решений, работать с деревьями в случае пропущенных значений, работать с деревьями с категориальными признаками.

Выполнение лабораторной работы

Задание 1

Формулировка: Реализуйте дерево для задачи регрессии. Возьмите за основу дерево, реализованное в файле "trees", заменив механизм предсказания в листе на взятие среднего значения по выборке, и критерий Джини на дисперсию значений.

Решение:

Изменим таким образом, что

1. **Leaf класс**:
   - `predict()` теперь возвращает среднее значение целевой переменной в листе
   - Используется `np.mean()` для расчета среднего В классификации мы использовали моду (наиболее частый класс), потому что минимизировали критерий Джини (меру неопределенности классов). В регрессии мы минимизируем дисперсию, поэтому используем среднее.

Задача	Критерий разделения	Предсказание в листе
Классификация	Джини	Мода (наиболее частый класс)
Регрессия	Дисперсия	Среднее значение

2. **Критерий variance**:
   - Заменяет критерий Джини
   - Рассчитывает дисперсию целевой переменной
   - Формула: $\frac{1}{n}\sum_{i=1}^n(y_i - \bar{y})^2$ Когда мы используем среднее значение в листе как предсказание, MSE становится равной дисперсии:
3. **Функция quality**:

- Рассчитывает уменьшение дисперсии после разделения
- Использует взвешенную дисперсию левого и правого подмножеств

Чем больше уменьшение дисперсии, тем лучше разделение

Код

```
import matplotlib.pyplot as plt

import random



from matplotlib.colors import ListedColormap

from sklearn import datasets



from sklearn import model_selection



import numpy as np



classification_data, classification_labels =
datasets.make_classification(n_features = 2, n_informative = 2,

                                                    n_classes = 2,
n_redundant=0,

                                                    n_clusters_per_class=1,
random_state=5)



class Node:



    def __init__(self, index, t, true_branch, false_branch):

        self.index = index  # индекс признака, по которому ведется сравнение с
порогом в этом узле

        self.t = t  # значение порога

        self.true_branch = true_branch  # поддерево, удовлетворяющее условию в
узле

        self.false_branch = false_branch  # поддерево, не удовлетворяющее условию
в узле
```

```
def predict(data, tree):


    classes = []
    for obj in data:
        prediction = classify_object(obj, tree)
        classes.append(prediction)
    return classes


def gini(labels):
    #  подсчет количества объектов разных классов
    classes = {}
    for label in labels:
        if label not in classes:
            classes[label] = 0
        classes[label] += 1


    #  расчет критерия
    impurity = 1
    for label in classes:
        p = classes[label] / len(labels)
        impurity -= p ** 2


    return impurity


def split(data, labels, index, t):


    left = np.where(data[:, index] <= t)
```

```
    right = np.where(data[:, index] > t)

    true_data = data[left]
    false_data = data[right]
    true_labels = labels[left]
    false_labels = labels[right]

    return true_data, false_data, true_labels, false_labels

def build_tree(data, labels):

    quality, t, index = find_best_split(data, labels)

    #  Базовый случай - прекращаем рекурсию, когда нет прироста в качества
    if quality == 0:
        return Leaf(data, labels)

    true_data, false_data, true_labels, false_labels = split(data, labels, index,
t)

    # Рекурсивно строим два поддерева
    true_branch = build_tree(true_data, true_labels)
    false_branch = build_tree(false_data, false_labels)

    # Возвращаем класс узла со всеми поддеревьями, то есть целого дерева
    return Node(index, t, true_branch, false_branch)
```

```
def classify_object(obj, node):


    #  Останавливаем рекурсию, если достигли листа
    if isinstance(node, Leaf):
        answer = node.prediction
        return answer


    if obj[node.index] <= node.t:
        return classify_object(obj, node.true_branch)
    else:
        return classify_object(obj, node.false_branch)


class Leaf:


    def __init__(self, data, labels):
        self.data = data
        self.labels = labels
        self.prediction = self.predict()


    def predict(self):
        # Предсказание в листе - среднее значение целевой переменной
        return np.mean(self.labels)


def variance(labels):
    # Расчет дисперсии целевой переменной
    if len(labels) == 0:
        return 0
```

```
    mean = np.mean(labels)

    return np.mean((labels - mean) ** 2)


def quality(left_labels, right_labels, current_variance):

    # Расчет качества разбиения через уменьшение дисперсии

    n = len(left_labels) + len(right_labels)

    p_left = len(left_labels) / n

    p_right = len(right_labels) / n

    return current_variance - (p_left * variance(left_labels) + p_right *
variance(right_labels))


def find_best_split(data, labels):

    min_leaf = 5

    current_variance = variance(labels)

    best_quality = 0

    best_t = None

    best_index = None


    n_features = data.shape[1]


    for index in range(n_features):

        t_values = np.unique(data[:, index])

        for t in t_values:

            true_data, false_data, true_labels, false_labels = split(data, labels,
index, t)

            if len(true_data) < min_leaf or len(false_data) < min_leaf:

                continue

            current_quality = quality(true_labels, false_labels, current_variance)
```

```
            if current_quality > best_quality:
                best_quality, best_t, best_index = current_quality, t, index

    return best_quality, best_t, best_index

regression_data, regression_labels = datasets.make_regression(n_features=2,
random_state=1)

train_data, test_data, train_labels, test_labels =
model_selection.train_test_split(
    regression_data, regression_labels, test_size=0.3, random_state=1)

# Построение дерева
tree = build_tree(train_data, train_labels)

# Прогнозирование
predictions = predict(test_data, tree)
print(predictions)

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

# Создадим цветовые карты для регрессии
cmap = plt.cm.viridis
light_cmap = plt.cm.viridis_r

def get_meshgrid(data, step=.05, border=1.2):
```

```
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max,
step))


# Предсказания на тренировочной и тестовой выборках
train_predictions = predict(train_data, tree)
test_predictions = predict(test_data, tree)


# Расчет метрик
train_mse = mean_squared_error(train_labels, train_predictions)
test_mse = mean_squared_error(test_labels, test_predictions)
train_r2 = r2_score(train_labels, train_predictions)
test_r2 = r2_score(test_labels, test_predictions)


plt.figure(figsize = (16, 7))


# график обучающей выборки
plt.subplot(1,2,1)
xx, yy = get_meshgrid(train_data)
mesh_predictions = np.array(predict(np.c_[xx.ravel(), yy.ravel()], tree))
mesh_predictions = mesh_predictions.reshape(xx.shape)


# Визуализация предсказаний на сетке
contour = plt.contourf(xx, yy, mesh_predictions, cmap=light_cmap, alpha=0.8)
plt.colorbar(contour, label='Predicted value')


# Визуализация реальных точек
```

```
scatter = plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels,
cmap=cmap, edgecolors='black', s=50)

plt.colorbar(scatter, label='True value')

plt.title(f'Train MSE={train_mse:.2f}, R²={train_r2:.2f}')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')



# график тестовой выборки

plt.subplot(1,2,2)

contour = plt.contourf(xx, yy, mesh_predictions, cmap=light_cmap, alpha=0.8)

plt.colorbar(contour, label='Predicted value')



scatter = plt.scatter(test_data[:, 0], test_data[:, 1], c=test_labels, cmap=cmap,
edgecolors='black', s=50)

plt.colorbar(scatter, label='True value')

plt.title(f'Test MSE={test_mse:.2f}, R²={test_r2:.2f}')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')



plt.tight_layout()

plt.show()
```

Выводы

Изучили метод обучения: деревья решения. В данной работе мы научились строить деревья решений, работать с деревьями в случае пропущенных значений, работать с деревьями с категориальными признаками.

Список литературы{.unnumbered}