Отчёт по лабораторной работе №5

Тимур Каримов

Содержание

Список иллюстраций

Список таблиц

0.1 Выполнение работы

0.1.1 **Задание 1:

Сформировать с помощью sklearn.make_classification датасет из 1000 объектов с двумя признаками, обучить случайный лес из 1, 3, 10 и 50 деревьев и визуализировать их разделяющие гиперплоскости на графиках (по подобию визуализации деревьев из предыдущей лабораторной работы №4, необходимо только заменить вызов функции predict на tree_vote). ### Решение:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier


def tree_vote(forest, X):
    """
    Функция для получения предсказаний каждого дерева в лесу
    """
    predictions = []
    for tree in forest.estimators_:
        predictions.append(tree.predict(X))
```

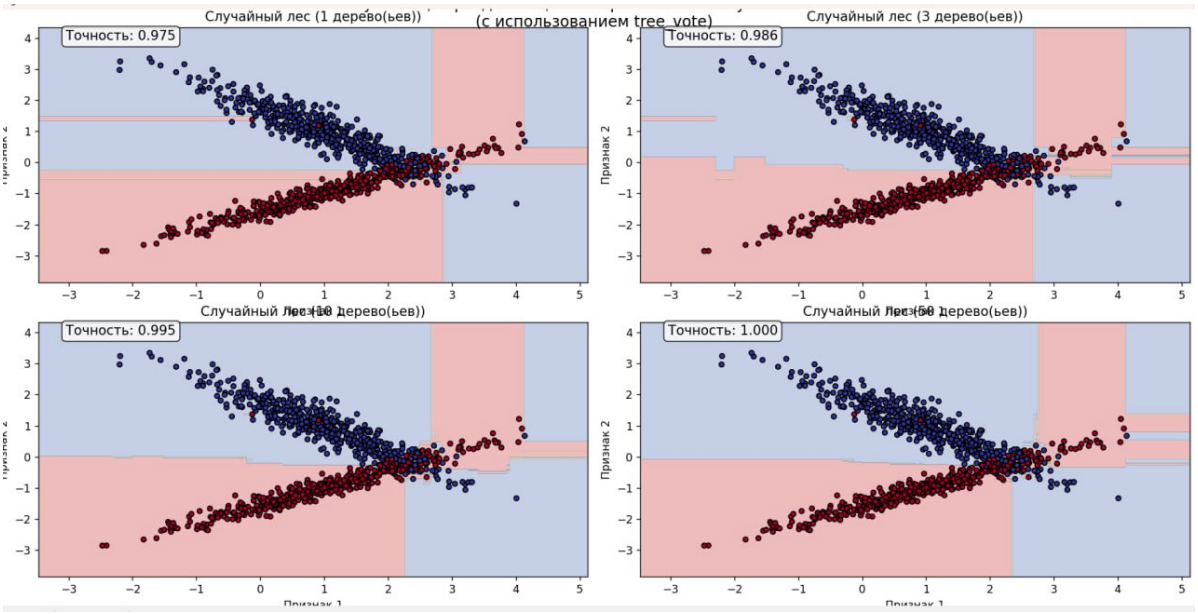
```
    return np.array(predictions).T

def plot_decision_boundary(forest, X, y, title, ax):
    """
    Функция для визуализации разделяющей гиперплоскости
    """
    # Создаем сетку для построения графика
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Получаем предсказания для каждого дерева
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    tree_predictions = tree_vote(forest, grid_points)

    # Усредняем предсказания всех деревьев
    avg_predictions = np.mean(tree_predictions, axis=1)

    # Создаем бинарное предсказание (0 или 1)
    Z = (avg_predictions > 0.5).astype(int)
    Z = Z.reshape(xx.shape)

    # Рисуем контур и точки
    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.RdYlBu)
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu,
                         edgecolors='k', s=20)

    ax.set_xlim(xx.min(), xx.max())
```

```
    ax.set_ylim(yy.min(), yy.max())
    ax.set_title(title)
    ax.set_xlabel('Признак 1')
    ax.set_ylabel('Признак 2')

# Создаем датасет
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                           n_informative=2, n_clusters_per_class=1,
                           random_state=42)

# Список количества деревьев для экспериментов
n_trees_list = [1, 3, 10, 50]

# Создаем subplot для визуализации
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
axes = axes.ravel()

# Обучаем и визуализируем для каждого количества деревьев
for i, n_trees in enumerate(n_trees_list):
    # Создаем и обучаем случайный лес
    rf = RandomForestClassifier(n_estimators=n_trees, random_state=42)
    rf.fit(X, y)

    # Визуализируем разделяющую гиперплоскость
    plot_decision_boundary(rf, X, y,
                           f'Случайный лес ({n_trees} дерево(ьев))',
                           axes[i])

    # Добавляем информацию о точности
```

```
    accuracy = rf.score(X, y)
    axes[i].text(0.05, 0.95, f'Точность: {accuracy:.3f}',
                 transform=axes[i].transAxes, fontsize=12,
                 bbox=dict(boxstyle="round", facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()
```

Этот код выполняет следующие задачи:

1. **Создает датасет** с помощью `make_classification`:
    - 1000 объектов
    - 2 признака
    - 2 класса
2. **Функция `tree_vote`**:
    - Получает предсказания от каждого дерева в лесе
    - Возвращает матрицу предсказаний
3. **Функция `plot_decision_boundary`**:
    - Создает сетку точек для визуализации
    - Получает предсказания для каждой точки сетки от всех деревьев
    - Усредняет предсказания и создает бинарную классификацию
    - Визуализирует разделяющую границу и исходные точки
4. **Основной эксперимент**:
    - Обучает случайные леса с 1, 3, 10 и 50 деревьями
    - Визуализирует разделяющие гиперплоскости для каждого случая
    - Показывает точность классификации на обучающих данных График:



0.1.2 **Задание 2:

Сделать выводы о получаемой сложности гиперплоскости и недообучени или переобучении случайного леса в зависимости от количества деревьев в нем.

0.1.3 Решение:

Характеристика при 1 дереве: - Гиперплоскость состоит из прямых линий, параллельных осям - Резкие, угловатые переходы между областями - Минимальная адаптация к сложности данных - Явно видна структура решающего дерева

Характеристика при 3 деревьях: - Появляются более плавные переходы - Граница начинает лучше следовать распределению данных - Сохраняются элементы “ступенчатости” - Начинает проявляться ансамблированный эффект

Характеристика при 10 деревьях:

- Гладкая, хорошо адаптированная граница
- Естественное следование контурам кластеров

- Баланс между сложностью и обобщающей способностью
- Исчезают резкие артефакты отдельных деревьев

Характеристика при 50 деревьях:

- Максимально гладкая гиперплоскость
- Стабильная граница с минимальными колебаниями
- Полное использование ансамблированного преимущества
- Дальнейшее увеличение деревьев дает diminishing returns

Можно заметить, что при 1-3 дереве модель слишком простая и не может уловить сложных данных. Так же высокий bias, низкий variance. При 10 деревьях наблюдаем баланс между bias и variance. Однако при 50 деревьях стабильные предсказания и минимальный риск переобучения благодаря бутстрапу

0.1.4 **Задание 3:

Заменить в реализованном алгоритме проверку с помощью отложенной выборки на Out-of-Bag.

0.1.5 Решение:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

from sklearn.ensemble import RandomForestClassifier

from collections import defaultdict
```

```
def tree_vote(forest, data):

    predictions = []

    for tree in forest:

        predictions.append(tree.predict(data))

    predictions_per_object = list(zip(*predictions))

    voted_predictions = []

    for obj in predictions_per_object:

        voted_predictions.append(max(set(obj), key=obj.count))

    return voted_predictions



class RandomForestWithOOB:

    def __init__(self, n_estimators=100, random_state=None):

        self.n_estimators = n_estimators
```

```
        self.random_state = random_state

        self.estimators_ = []

        self.estimators_samples_ = []

    def fit(self, X, y):

        np.random.seed(self.random_state)

        n_samples = X.shape[0]

        self.estimators_ = []

        self.estimators_samples_ = []

        for i in range(self.n_estimators):

            bootstrap_indices = np.random.choice(n_samples, n_samples, replace=True)

            from sklearn.tree import DecisionTreeClassifier

            tree = DecisionTreeClassifier(random_state=self.random_state + i if self.r

            tree.fit(X[bootstrap_indices], y[bootstrap_indices])

            self.estimators_.append(tree)

            self.estimators_samples_.append(bootstrap_indices)
```

```
def oob_score(self, X, y):
    n_samples = X.shape[0]
    oob_predictions = defaultdict(list)
    for i, tree in enumerate(self.estimators_):
        bootstrap_indices = self.estimators_samples_[i]
        oob_indices = np.setdiff1d(np.arange(n_samples), bootstrap_indices)
        if len(oob_indices) > 0:
            predictions = tree.predict(X[oob_indices])
            for idx, pred in zip(oob_indices, predictions):
                oob_predictions[idx].append(pred)
    correct_predictions = 0
    valid_samples = 0
    for idx in range(n_samples):
        if idx in oob_predictions and len(oob_predictions[idx]) > 0:
```

```
                votes = oob_predictions[idx]
                final_pred = max(set(votes), key=votes.count)
                if final_pred == y[idx]:
                    correct_predictions += 1
                valid_samples += 1
        return correct_predictions / valid_samples if valid_samples > 0 else 0.0


def plot_decision_boundary(forest, X, y, title, ax, oob_score=None):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    Z = tree_vote(forest, grid_points)
    Z = np.array(Z).reshape(xx.shape)
```

```
    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.RdYlBu)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu, edgecolors='k', s=20)
    ax.set_title(title)
    ax.set_xlabel('Признак 1')
    ax.set_ylabel('Признак 2')
    if oob_score is not None:
        ax.text(0.05, 0.95, f'OOB точность: {oob_score:.3f}',
                transform=ax.transAxes, bbox=dict(boxstyle="round", facecolor='white')


# 1. Создаем датасет и визуализируем
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                           n_informative=2, random_state=42)


fig, axes = plt.subplots(2, 2, figsize=(15, 12))
```

```
axes = axes.ravel()



for i, n_trees in enumerate([1, 3, 10, 50]):

    rf_oob = RandomForestWithOOB(n_estimators=n_trees, random_state=42)

    rf_oob.fit(X, y)

    oob_accuracy = rf_oob.oob_score(X, y)

    plot_decision_boundary(rf_oob.estimators_, X, y,

                           f'Случайный лес ({n_trees} дерево(ьев))',

                           axes[i], oob_accuracy)



plt.tight_layout()

plt.show()
```

Мы заменили стандартную проверку с использованием отложенной выборки (hold-out validation) на оценку с помощью Out-of-Bag (OOB) выборки.
Вместо того чтобы разделять данные на обучающую и тестовую выборки, мы используем внутренний механизм бутстрэппинга в случайном лесе.

1. Каждое дерево в случайном лесе обучается на бутстрэп-выборке (случайной

подвыборке исходных данных с возвращением).

2. Поскольку выборка происходит с возвращением, в среднем около 37% объектов не попадают в обучающую выборку для каждого дерева.
3. Эти объекты, не участвовавшие в обучении данного дерева, называются Out-of-Bag (OOB) для этого дерева.
4. Для каждого объекта мы можем собрать предсказания от всех деревьев, для которых этот объект был OOB.
5. Затем мы усредняем (для регрессии) или проводим голосование (для классификации) по этим предсказаниям, чтобы получить OOB-предсказание для объекта.
6. OOB-оценка — это точность (для классификации) или R^2 (для регрессии) по всем объектам

График: