Случайные леса

В этом задании вам предстоит реализовать ансамбль деревьев решений, известный как случайный лес, применить его к публичным данным пользователей социальной сети Вконтакте, и сравнить его эффективность с бустингом, предоставляемым библиотекой `CatBoost`.

В результате мы сможем определить, какие подписки пользователей больше всего влияют на определение возраста и пола человека.

In [3]:
```
import inspect
import random
from collections import Counter
from dataclasses import dataclass
from itertools import product
from typing import Callable, List, Tuple, Union

import numpy as np
import numpy.typing as npt
import pandas
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
```

In [4]:
```
def set_seed(seed=42):
    np.random.seed(seed)
    random.seed(seed)


# Этой функцией будут помечены все места, которые необходимо дозаполнить
# Это могут быть как целые функции, так и отдельные части внутри них
# Всегда можно воспользоваться интроспекцией и найти места использования
def todo():
    stack = inspect.stack()
    caller_frame = stack[1]
    function_name = caller_frame.function
    line_number = caller_frame.lineno
    raise NotImplementedError(f"TODO at {function_name}, line {line_numbe


SEED = 0xC0FFEE
set_seed(SEED)
```

In [5]:
```
def mode(data):
    counts = Counter(data)
    return counts.most_common(n=1)[0][0]
```

Задание 1 (2 балла)

Random Forest состоит из деревьев решений. Каждое такое дерево строится на одной из выборок, полученных при помощи bootstrap. Элементы, которые не вошли в новую обучающую выборку, образуют **out-of-bag** выборку. Кроме того, в

каждом узле дерева мы случайным образом выбираем набор из `max_features` и ищем признак для предиката разбиения только в этом наборе.

Сегодня мы будем работать только с бинарными признаками, поэтому нет необходимости выбирать значение признака для разбиения.

Методы

`predict(X)` - возвращает предсказанные метки для элементов выборки `X`

Параметры конструктора

`X, y` - обучающая выборка и соответствующие ей метки классов. Из нее нужно получить выборку для построения дерева при помощи bootstrap. Out-of-bag выборку нужно запомнить, она понадобится потом.

`criterion="gini"` - задает критерий, который будет использоваться при построении дерева. Возможные значения: `"gini"`, `"entropy"`.

`max_depth=None` - ограничение глубины дерева. Если `None` - глубина не ограничена

`min_samples_leaf=1` - минимальное количество элементов в каждом листе дерева.

`max_features="auto"` - количество признаков, которые могут использоваться в узле. Если `"auto"` - равно `sqrt(X.shape[1])`

In [6]:
```python
# Для начала реализуем сами критерии


def gini(x: npt.ArrayLike) -> float:
    """
    Calculate the Gini impurity of a list or array of class labels.

    Args:
        x (ArrayLike): Array-like object containing class labels.

    Returns:
        float: Gini impurity value.
    """
    if len(x) == 0:
        return 0.0

    x = np.asarray(x)

    _, count = np.unique(x, return_counts=True)
    probabilities = count / len(x)

    return np.sum(probabilities * (1 - probabilities))


def entropy(x: npt.ArrayLike) -> float:
    """
    Calculate the entropy of a list or array of class labels.
```

```
    Args:
        x (ArrayLike): Array-like object containing class labels.

    Returns:
        float: Entropy value.
    """
    if len(x) == 0:
        return 0.0

    x = np.asarray(x)

    _, count = np.unique(x, return_counts=True)
    probabilities = count / len(x)

    return -np.sum(probabilities * np.log2(probabilities, where=(probabil


def gain(left_y: npt.ArrayLike, right_y: npt.ArrayLike, criterion: Callab
    """
    Calculate the information gain of a split using a specified criterion

    Args:
        left_y (ArrayLike): Class labels for the left split.
        right_y (ArrayLike): Class labels for the right split.
        criterion (Callable): Function to calculate impurity (e.g., gini

    Returns:
        float: Information gain from the split.
    """
    left_y, right_y = np.asarray(left_y), np.asarray(right_y)

    y = np.concatenate([left_y, right_y])
    R = len(y)
    R_l, R_r = len(left_y), len(right_y)

    return criterion(y) - (R_l / R) * criterion(left_y) - (R_r / R) * cri
```

In [7]:
```
@dataclass
class DecisionTreeLeaf:
    classes: np.ndarray

    def __post_init__(self):
        self.max_class = mode(self.classes)


@dataclass
class DecisionTreeInternalNode:
    split_dim: int
    left: Union["DecisionTreeInternalNode", DecisionTreeLeaf]
    right: Union["DecisionTreeInternalNode", DecisionTreeLeaf]


DecisionTreeNode = Union[DecisionTreeInternalNode, DecisionTreeLeaf]
```

In [8]:
```
class DecisionTree:
    def __init__(self, X, y, criterion="gini", max_depth=None, min_sample
        self.criterion: Callable = gini if criterion == "gini" else entro
        self.max_depth: int | None = max_depth
```

```
        self.min_samples: int = min_samples_leaf
        self.max_features: int = int(np.sqrt(X.shape[1])) if max_features

        n_samples = X.shape[0]
        bootstrap_indices = np.random.choice(n_samples, size=n_samples, r
        oob_mask = np.ones(n_samples, dtype=bool)
        oob_mask[bootstrap_indices] = False
        self._out_of_bag_X = X[oob_mask]
        self._out_of_bag_y = y[oob_mask]
        X = X[bootstrap_indices]
        y = y[bootstrap_indices]

        self.root = self._build_node(X, y, depth=0)

    @property
    def out_of_bag(self) -> Tuple[np.ndarray, np.ndarray]:
        return self._out_of_bag_X, self._out_of_bag_y

    def _build_node(self, points: np.ndarray, classes: np.ndarray, depth:
        if len(points) <= self.min_samples or len(np.unique(classes)) ==
            return DecisionTreeLeaf(classes)

        ind, mask = self._find_best_split(points, classes)
        if mask is None:
            return DecisionTreeLeaf(classes)

        X_left, X_right = points[mask], points[~mask]
        y_left, y_right = classes[mask], classes[~mask]

        if len(y_right) <= self.min_samples or len(y_left) <= self.min_sa
            return DecisionTreeLeaf(classes)

        return DecisionTreeInternalNode(ind, self._build_node(X_left, y_l

    def _find_best_split(self, X: np.ndarray, y: np.ndarray):
        max_gain = -1
        best_mask = None
        best_ind = None

        feature_inds = np.random.choice(np.arange(0, X.shape[1]), size=se
        for ind in feature_inds:
            mask = X[:, ind] < 0.5
            y_left = y[mask]
            y_right = y[~mask]

            if len(y_left) >= self.min_samples and len(y_right) >= self.m
                gain_ind = gain(y_left, y_right, self.criterion)
                if gain_ind > max_gain:
                    max_gain = gain_ind
                    best_mask = mask
                    best_ind = ind

        return best_ind, best_mask

    def _predict(self, point: np.ndarray, node: DecisionTreeNode) -> int:
        if isinstance(node, DecisionTreeLeaf):
            return node.max_class

        if point[node.split_dim] < 0.5:
            return self._predict(point, node.left)
```

```
        return self._predict(point, node.right)

    def predict(self, points: np.ndarray) -> np.ndarray:
        classes_pred = []
        for point in points:
            classes_pred.append(self._predict(point, self.root))

        return np.array(classes_pred)
```

Задание 2 (2 балла)

Теперь реализуем сам Random Forest. Идея очень простая: строим `n` деревьев, а затем берем модальное предсказание.

Параметры конструктора

`n_estimators` - количество используемых для предсказания деревьев.

Остальное - параметры деревьев.

Методы

`fit(X, y)` - строит `n_estimators` деревьев по выборке `X`.

`predict(X)` - для каждого элемента выборки `X` возвращает самый частый класс, который предсказывают для него деревья.

In [9]:
```
class RandomForestClassifier:

    _n_features: int = None

    def __init__(self, criterion="gini", max_depth=None, min_samples_leaf
        self._criterion = criterion
        self._max_depth = max_depth
        self._min_samples_leaf = min_samples_leaf
        self._max_features = max_features
        self._n_estimators = n_estimators
        self._estimators: list[DecisionTree | None] = []

    @property
    def estimators(self) -> List[DecisionTree]:
        return self._estimators

    @property
    def n_features(self) -> int:
        if self._n_features is None:
            raise RuntimeError("Fit random forest before accessing to num
        return self._n_features

    def fit(self, X, y):
        self._n_features = X.shape[1]

        for _ in range(self._n_estimators):
            tree = DecisionTree(X, y, self._criterion, self._max_depth, s
            self.estimators.append(tree)
```

```
    def predict(self, X) -> np.ndarray:
        all_preds = []
        for estimator in self.estimators:
            all_preds.append(estimator.predict(X))

        y_pred = np.stack(all_preds, axis=1)
        preds = []
        for pred in y_pred:
            preds.append(mode(pred))

        return np.array(preds)

    def get_params(self, deep=True) -> dict:
        return {
            "criterion": self._criterion,
            "max_depth": self._max_depth,
            "min_samples_leaf": self._min_samples_leaf,
            "max_features": self._max_features,
            "n_estimators": self._n_estimators
        }

    def set_params(self, **params) -> "RandomForestClassifier":
        for key, value in params.items():
            setattr(self, key, value)
        return self
```

Задание 3 (2 балла)

Часто хочется понимать, насколько большую роль играет тот или иной признак для предсказания класса объекта. Есть различные способы посчитать его важность. Один из простых способов сделать это для Random Forest - посчитать out-of-bag ошибку предсказания `err_oob`, а затем перемешать значения признака `j` и посчитать ее (`err_oob_j`) еще раз. Оценкой важности признака `j` для одного дерева будет разность `err_oob_j - err_oob`, важность для всего леса считается как среднее значение важности по деревьям.

Реализуйте функцию `feature_importance`, которая принимает на вход Random Forest и возвращает массив, в котором содержится важность для каждого признака.

In [10]:
```
def accuracy_score(y_true: np.ndarray, y_pred: np.ndarray) -> float:
    y_true = y_true.reshape(-1)
    y_pred = y_pred.reshape(-1)
    return np.mean(y_true == y_pred)


def feature_importance(rfc: RandomForestClassifier):
    matrix_importance = np.zeros(rfc.n_features)

    for estimator in rfc.estimators:
        X, y = estimator.out_of_bag
        if len(X) == 0:
            continue

        y_pred = estimator.predict(X)
        err_obb = accuracy_score(y, y_pred)
```

```
        for j in range(rfc.n_features):
            X_shuffle = X.copy()
            np.random.shuffle(X_shuffle[:, j])
            y_pred_shuffled = estimator.predict(X_shuffle)
            err_obb_j = accuracy_score(y, y_pred_shuffled)
            matrix_importance[j] += err_obb - err_obb_j

    valid_estimators = sum(1 for est in rfc.estimators if len(est.out_of_
    if valid_estimators == 0:
        return matrix_importance
    return matrix_importance / valid_estimators


def most_important_features(importance, names, k=20):
    # Выводит названия k самых важных признаков
    indices = np.argsort(importance)[::-1][:k]
    return np.array(names)[indices]
```

Наконец, пришло время протестировать наше дерево на простом синтетическом наборе данных. В результате точность должна быть примерно равна `1.0`, наибольшее значение важности должно быть у признака с индексом `4`, признаки с индексами `2` и `3` должны быть одинаково важны, а остальные признаки - не важны совсем.

In [11]:
```
def synthetic_dataset(size):
    X = [
        (np.random.randint(0, 2), np.random.randint(0, 2), i % 6 == 3, i
        for i in range(size)
    ]
    y = [i % 3 for i in range(size)]
    return np.array(X), np.array(y)


X, y = synthetic_dataset(1000)
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X, y)
print("Accuracy:", np.mean(rfc.predict(X) == y))
print("Importance:", feature_importance(rfc))
```

```
Accuracy: 1.0
Importance: [-1.24005774e-03 -1.08250356e-03  1.70921794e-01  1.65304361e-
01
  3.35875415e-01  1.68137126e-04]
```

Задание 4 (1 балл)

Теперь поработаем с реальными данными.

Выборка состоит из публичных анонимизированных данных пользователей социальной сети Вконтакте. Первые два столбца отражают возрастную группу (`zoomer`, `doomer` и `boomer`) и пол (`female`, `male`). Все остальные столбцы являются бинарными признаками, каждый из них определяет, подписан ли пользователь на определенную группу/публичную страницу или нет.

Необходимо обучить два классификатора, один из которых определяет

возрастную группу, а второй - пол.

Эксперименты с множеством используемых признаков и подбор гиперпараметров приветствуются. Лес должен строиться за какое-то разумное время.

In [12]:
```
def read_dataset(path):
    dataframe = pandas.read_csv(path, header=0)
    dataset = dataframe.values.tolist()
    random.shuffle(dataset)
    y_age = [row[0] for row in dataset]
    y_sex = [row[1] for row in dataset]
    X = [row[2:] for row in dataset]

    return np.array(X), np.array(y_age), np.array(y_sex), list(dataframe.
```

In [13]:
```
X, y_age, y_sex, features = read_dataset("src/homeworks/homework6/vk.csv"
X_train, X_test, y_age_train, y_age_test, y_sex_train, y_sex_test = train
```

In [12]:
```
from sklearn.model_selection import GridSearchCV
```

Возраст

In [13]:
```
rfc = RandomForestClassifier(n_estimators=10)

params = {
    "_criterion": ["gini", "entropy"],
    "_max_depth" : [None, 10, 20],
    "_min_samples_leaf": [1, 3, 15],
    "_max_features": ["auto", np.log2(149)],
    "_n_estimators": [10, 40]}

grid = GridSearchCV(
    estimator=rfc,
    param_grid=params,
    scoring='accuracy',
    cv=5,
    n_jobs=-1
)

grid.fit(X_train, y_age_train)
print("Лучшие параметры для rfc:", grid.best_params_)
```

```
Лучшие параметры для rfc: {'_criterion': 'entropy', '_max_depth': None, '_
max_features': 'auto', '_min_samples_leaf': 1, '_n_estimators': 40}
```

In [14]:
```
rfc = RandomForestClassifier("entropy", None, 1, "auto", 40)

rfc.fit(X_train, y_age_train)
print("Accuracy:", np.mean(rfc.predict(X_test) == y_age_test))
print("Most important features:")
for i, name in enumerate(most_important_features(feature_importance(rfc),
    print(str(i + 1) + ".", name)
```

```
Accuracy: 0.7238335435056746
Most important features:
1. ovsyanochan
2. 4ch
3. styd.pozor
4. mudakoff
5. rhymes
6. dayvinchik
7. rapnewrap
8. pravdashowtop
9. pixel_stickers
10. tumblr_vacuum
11. reflexia_our_feelings
12. bot_maxim
13. iwantyou
14. leprum
15. bestad
16. i_d_t
17. xfilm
18. ohhluul
19. ne1party
20. bog_memes
```

Пол

In [ ]:
```python
rfc = RandomForestClassifier(n_estimators=10)

params = {
    "_criterion": ["gini", "entropy"],
    "_max_depth" : [None, 10, 20],
    "_min_samples_leaf": [1, 3, 15],
    "_max_features": ["auto", np.log2(149)],
    "_n_estimators": [10, 40]}

grid = GridSearchCV(
    estimator=rfc,
    param_grid=params,
    scoring='accuracy',
    cv=5,
    n_jobs=-1
)

grid.fit(X_train, y_sex_train)
print("Лучшие параметры для rfc:", grid.best_params_)
```

In [ ]:
```python
rfc = RandomForestClassifier("entropy", None, 1, "auto", 40)

rfc.fit(X_train, y_sex_train)
print("Accuracy:", np.mean(rfc.predict(X_test) == y_sex_test))
print("Most important features:")
for i, name in enumerate(most_important_features(feature_importance(rfc),
    print(str(i + 1) + ".", name)
```

```
Accuracy: 0.849936948297604
Most important features:
1. 40kg
2. girlmeme
3. modnailru
4. zerofat
5. 9o_6o_9o
6. mudakoff
7. be.beauty
8. i_d_t
9. woman.blog
10. 4ch
11. reflexia_our_feelings
12. igm
13. cook_good
14. beauty
15. femalemem
16. recipes40kg
17. thesmolny
18. sh.cook
19. be.women
20. rapnewrap
```

CatBoost

В качестве аьтернативы попробуем CatBoost.

Устаниовить его можно просто с помощью `pip install catboost`. Туториалы можно найти, например, здесь и здесь. Главное - не забудьте использовать `loss_function='MultiClass'`.

Сначала протестируйте CatBoost на синтетических данных. Выведите точность и важность признаков.

In [14]:
```
X, y = synthetic_dataset(1000)

cb_model = CatBoostClassifier(iterations=10, learning_rate=0.01, depth=10
cb_model.fit(X, y)
y_pred = cb_model.predict(X)

print("Accuracy:", accuracy_score(y_pred, y))
print("Importance:", cb_model.feature_importances_)
```

```
Accuracy: 1.0
Importance: [6.91453034e-04 4.02295744e-04 2.80921406e+01 2.82148745e+01
 4.36910485e+01 8.42592869e-04]
```

Задание 5 (3 балла)

Попробуем применить один из используемых на практике алгоритмов. В этом нам поможет CatBoost. Также, как и реализованный ними RandomForest, применим его для определения пола и возраста пользователей сети Вконтакте, выведите названия наиболее важных признаков так же, как в задании 3.

Эксперименты с множеством используемых признаков и подбор гиперпараметров приветствуются.

In [15]:
```
X, y_age, y_sex, features = read_dataset("src/homeworks/homework6/vk.csv"
X_train, X_test, y_age_train, y_age_test, y_sex_train, y_sex_test = train
X_train, X_eval, y_age_train, y_age_eval, y_sex_train, y_sex_eval = train
    X_train, y_age_train, y_sex_train, train_size=0.8
)
```

In [16]:
```
max_depth = range(1, 10, 3)
min_samples_leaf = range(1, 10, 3)
learning_rate = np.linspace(0.001, 1.0, 5)


def get_best_params(y_train, y_eval):
    best_score, best_params = None, None
    for lr, md, msl in tqdm(list(product(learning_rate, max_depth, min_sa
        cb_model = CatBoostClassifier(iterations=100, learning_rate=lr, d
        cb_model.fit(X_train, y_train)
        y_pred = cb_model.predict(X_eval)
        score = accuracy_score(y_eval, y_pred)

        if best_score is None or score > best_score:
            best_score = score
            best_params = {
                'learning_rate': lr,
                'depth': md,
                'min_data_in_leaf': msl
            }
    return best_params, best_score
```

Возраст

In [17]:
```
best_params, best_score = get_best_params(y_age_train, y_age_eval)
best_params, best_score
```

```
 0%|          | 0/45 [00:00<?, ?it/s]
```

Out[17]:
```
({'learning_rate': np.float64(0.25075), 'depth': 7, 'min_data_in_leaf':
1},
 np.float64(0.7512263489838823))
```

In [18]:
```
cb_model = CatBoostClassifier(iterations=100, loss_function='MultiClass',
cb_model.fit(X_train, y_age_train)
y_pred = cb_model.predict(X_test)

print("Accuracy:", accuracy_score(y_age_test, y_pred))
print("Most important features:")
for i, name in enumerate(most_important_features(cb_model.feature_importa
    print(str(i + 1) + ".", name)
```

```
Accuracy: 0.7074401008827238
Most important features:
1. ovsyanochan
2. 4ch
3. styd.pozor
4. mudakoff
5. leprum
6. xfilm
7. dayvinchik
8. i_des
9. rhymes
10. tumblr_vacuum
```

Пол

In [19]:
```
best_params, best_score = get_best_params(y_sex_train, y_sex_eval)
best_params, best_score
```

```
 0%|          | 0/45 [00:00<?, ?it/s]
```

Out[19]:
```
({'learning_rate': np.float64(0.5005), 'depth': 4, 'min_data_in_leaf':
1},
 np.float64(0.8647512263489839))
```

In [20]:
```
cb_model = CatBoostClassifier(iterations=100, loss_function='MultiClass',
cb_model.fit(X_train, y_sex_train)
y_pred = cb_model.predict(X_test)

print("Accuracy:", accuracy_score(y_sex_test, y_pred))
print("Most important features:")
for i, name in enumerate(most_important_features(cb_model.feature_importa
    print(str(i + 1) + ".", name)
```

```
Accuracy: 0.8776796973518285
Most important features:
1. 40kg
2. mudakoff
3. girlmeme
4. 9o_6o_9o
5. modnailru
6. thesmolny
7. be.beauty
8. i_d_t
9. zerofat
10. femalemem
```