Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2 по курсу «Искусственный интеллект»

Студент: М. А. Инютин
Преподаватели: Д. В. Сошников
С. Х. Ахмед
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №2

Задача: Вы построили базовые (слабые) модели машинного обучения под вашу задачу. Некоторые задачи показали себя не очень, некоторые показали себя хорошо. Как выяснилось, вашим инвесторам показалось этого мало и они хотят, чтобы вы построили модели посерьезней и поточнее. Вы вспомнили, что когда то вы проходили курс машинного обучения и слышали что есть способ улучшить результаты вашей задачи: ансамбли: беггинг, пастинг, бустинг и стекинг, а также классификация путем жесткого и мягкого голосования и вы решили это опробовать. Требования к написанным классам вы оставляете теми же, что и в предыдущей работе. Будьте аккуратны в оптимизации целевой метрики и учитывайте несбалансированность классов.

Ваша задача:

1. Используя модели которые вы реализовали в предыдущей лабораторной работе, реализовать два подхода для построения ансамблей: жесткое и мягкое голосование, однако учтите, некоторые модели не предусматривают оценку вероятностей, например SVM и потому вам необходимо будет оценивать вероятности;
2. Реализовать дерево решений;
3. Реализовать случайный лес;
4. Воспользоваться готовой коробочной реализацией градиентного бустинга для решения вашей задачи.

Для всех моделей провести fine-tuning.

1 Описание

Подготовка данных и совместимость с scikit-learn сделаны так же, как и в прошлой лабораторной работе. Приведу реализации моделей.

1 Дерево принятия решений

Для хранений данных в узлах дерева опишу струкуру, которая может принять решение или выдать вероятность класса. Если вершина является листом, то value содержит вероятность класса, иначе value содержит значение для предиката, а ind индекс значения.

```
1  class TreeData():
2      def __init__(self, value = None, ind = None, leaf = True):
3          self.value = value
4          self.ind = ind
5          self.leaf = leaf
6
7      def is_leaf(self):
8          return self.leaf
9
10     def decide(self, X):
11         if (self.is_leaf()):
12             return None
13         else:
14             return True if (X[self.ind] >= self.value) else False
15
16     def predict(self, X):
17         if (self.is_leaf()):
18             return value
19         else:
20             return None
```

Буду хранить дерево в виде списка смежности вершин и массива с данными для каждой вершины.

```
class Graph:
    def __init__(self):
        self.data = []
        self.info = []
        self.size = 0
        self.add()

    def can_go(self, u, c):
        return c in self.data[u]

    def go(self, u, c):
        return self.data[u][c]

    def set_go(self, u, c, v):
        self.data[u][c] = v

    def get_tree_data(self, u):
        return self.info[u]

    def set_tree_data(self, u, tree_data):
        self.info[u] = tree_data

    def add(self):
        self.data.append(dict())
        self.info.append(TreeData())
        self.size += 1
        return self.size - 1

    def is_leaf(self, u):
        return self.info[u].is_leaf()
```

Класс дерева принятия решений содержит вспомогательные функции: get_count для подсчёта количества классов, ans для вычисления вероятности, ans_class для определения номера класса по вероятности, функции для вычисления предсказания, создания терминальной вершины и построения дерева по набору данных.

```
class DecisionTree:
    def __init__(self, max_depth, min_count, rnd_split):
        self.tree = Graph()
        self.max_depth = max_depth
        self.min_count = min_count
        self.rnd_split = rnd_split
        self.data = []

    def get_count(self, arr):
        res = [0 for _ in range(2)]
        for elem in arr:
            res[elem] += 1
        return res

    def ans(self, cnt0, cnt1):
        if (cnt0 + cnt1 == 0):
            return None
        else:
            return (cnt1) / (cnt0 + cnt1)

    def ans_class(self, p):
        return 1 if (p > 0.5) else 0

    def decide(self, X):
        return self.decide_rec(0, X)

    def decide_rec(self, u, X):
        u_data = self.tree.get_tree_data(u)
        if (self.tree.is_leaf(u)):
            return u_data.value
        else:
            dec = u_data.decide(X)
            return self.decide_rec(self.tree.go(u, dec), X)

    def make_leaf(self, u, value):
        u_data = TreeData(value = value, leaf = True)
        self.tree.set_tree_data(u, u_data)

    def rnd_ids(self, n):
        res = []
        while (len(res) * len(res) < n):
            rnd_num = randint(0, n - 1)
            while (rnd_num in res):
                rnd_num = randint(0, n - 1)
```

```
45              res.append(rnd_num)
46          return res
47
48      def build(self, X, y):
49          self.X = X
50          self.y = y
51          self.n = X.shape[0]
52          self.d = X.shape[1]
53          ids = np.arange(self.n)
54          self.build_rec(0, ids, 1)
55
56      def build_rec(self, u, ids, h):
57          X = self.X[ids]
58          y = self.y[ids]
59          u_cnt = self.get_count(y)
60          u_ans = self.ans(u_cnt[0], u_cnt[1])
61          stop1 = (len(y) <= self.min_count)
62          stop2 = (h > self.max_depth)
63          stop3 = (u_ans == None)
64          if (stop1 or stop2 or stop3):
65              self.make_leaf(u, u_ans)
66              return
67          z = self.ans_class(u_ans)
68          n = len(ids)
69          min_loss = n - u_cnt[z]
70          res = (-1, -1)
71          u_data = self.tree.get_tree_data(u)
72          split_ids = (self.rnd_ids(self.d) if self.rnd_split else range(self.d))
73          for i in split_ids:
74              cnt_l, cnt_r = [0 for _ in range(2)], [elem for elem in u_cnt]
75              tmp = sorted([(elem[i], y[j]) for (j, elem) in enumerate(X)])
76              loss_l, loss_r = 0, n - u_cnt[z]
77              size_l, size_r = 0, n
78              for j in range(self.n):
79                  while (size_l < n and tmp[size_l][0] <= tmp[j][0]):
80                      y_l = tmp[size_l][1]
81                      cnt_l[y_l] += 1
82                      cnt_r[y_l] -= 1
83                      size_l += 1
84                      size_r -= 1
85                  if (size_l == 0 or size_r == 0):
86                      continue
87                  ans_l = self.ans_class(self.ans(cnt_l[0], cnt_l[1]))
88                  ans_r = self.ans_class(self.ans(cnt_r[0], cnt_r[1]))
89                  loss_l = size_l - cnt_l[ans_l]
90                  loss_r = size_r - cnt_r[ans_r]
91                  split_loss = loss_l + loss_r
92                  if (split_loss < min_loss):
93                      min_loss = split_loss
```

```
                res = (i, tmp[j][0])
        if (res == (-1, -1)):
            self.make_leaf(u, u_ans)
            return
        u_data = TreeData(value = res[1], ind = res[0], leaf = False)
        self.tree.set_tree_data(u, u_data)
        l, r = [], []
        for (j, elem) in enumerate(X):
            if (u_data.decide(elem)):
                l.append(j)
            else:
                r.append(j)
        l, r = np.array(l), np.array(r)
        if (len(l) == 0 or len(r) == 0):
            self.make_leaf(u, u_ans)
            return
        ul = self.tree.add()
        ur = self.tree.add()
        self.tree.set_go(u, True, ul)
        self.tree.set_go(u, False, ur)
        self.build_rec(ul, l, h + 1)
        self.build_rec(ur, r, h + 1)
```

Сам классификатор содержит дерево и использует его методы.

```
class MyDecisionTreeClassifier(ClassifierMixin, BaseEstimator):
    def __init__(self, max_depth = 10, min_count = 50, rnd_split = False):
        self.max_depth = max_depth
        self.min_count = min_count
        self.rnd_split = rnd_split
        self.tree = DecisionTree(max_depth, min_count, rnd_split)

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        self.tree.build(X, y)
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        p = []
        for elem in X:
            z = self.tree.decide(elem)
            p.append(1 if z > 0.5 else 0)
        return np.array(p)
```

2 Случайный лес

Классификатор случайного леса содержит список решающих деревьев со случайным выбором параметра для разделения.

```
class MyRandomForestClassifier(ClassifierMixin, BaseEstimator):
    def __init__(self, n_trees = 10, max_depth = 2, min_count = 5):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.min_count = min_count
        self.trees = [DecisionTree(max_depth, min_count, rnd_split = True) for _ in
            range(self.n_trees)]

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        for (i, tree) in enumerate(self.trees):
            rnd_ids = np.random.choice(len(X), (2 * len(X)) // self.n_trees)
            X_sub = X[rnd_ids]
            y_sub = y[rnd_ids]
            tree.build(X_sub, y_sub)
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        p = []
        for elem in X:
            z = 0
            for tree in self.trees:
                z += tree.decide(elem)
            z /= self.n_trees
            p.append(1 if z > 0.5 else 0)
        return np.array(p)
```

3 Мягкое голосование

Мягкое голосование принимает список классификаторов, выдающих вероятность класса. В случае с бинарной классификацией можно однозначно определить вероятность второго класса. Вычислим сумму по всем моделям и выделим класс с наибольшей суммой вероятностей.

```
class SVEnsemble(ClassifierMixin, BaseEstimator):
    def __init__(self, models):
        self.models = models

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        for model in self.models:
            model.fit(X, y)
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        y, p = np.ndarray((X.shape[0],)), []
        for model in models:
            p.append(model.predict(X))
        p = np.array(p).T
        for i in range(len(X)):
            p_of_labels = np.array([0 for _ in range(2)])
            for j in range(len(models)):
                p1 = p[i][j]
                p0 = 1 - p1
                p_of_labels[0] += p0
                p_of_labels[1] += p1
            y[i] = p_of_labels.argmax()
        return y
```

4 Жёсткое голосование

Как и мягкое голосование классификатор принимает список моделей, выдающих вероятности класса. На их подсчитываются голоса, затем производится голосование.

```
class HVEnsemble(ClassifierMixin, BaseEstimator):
    def __init__(self, models):
        self.models = models

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        for model in self.models:
            model.fit(X, y)
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])

        # Input validation
        X = check_array(X)

        y, p = np.ndarray((X.shape[0],)), []
        for model in models:
            p.append(model.predict(X))
        p = np.array(p).T
        for i in range(len(X)):
            cnt = np.array([0 for _ in range(2)])
            for j in range(len(models)):
                label = (1 if p[i][j] > 0.5 else 0)
                cnt[label] += 1
            y[i] = cnt.argmax()
        return y
```

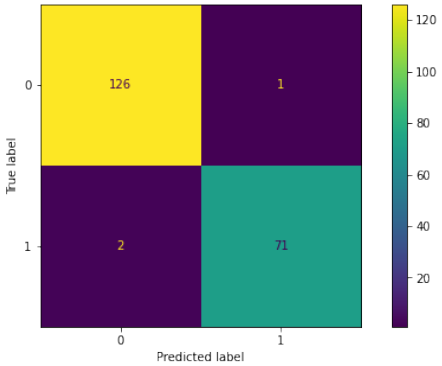
2 Результаты моделей

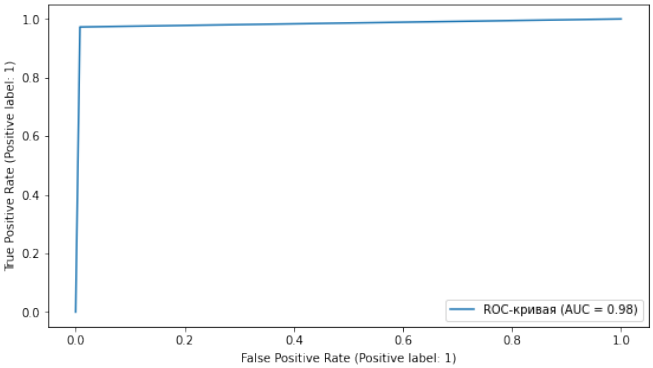
Ниже приведены результаты всех моделей. Для дерева принятия решений и случайного леса сравнивается моя реализация и из библиотеки. Мягкое и жёсткое голосование использует классификаторы из предыдущей лабораторной работы.

1 Дерево принятия решений

1.1 Моя реализация

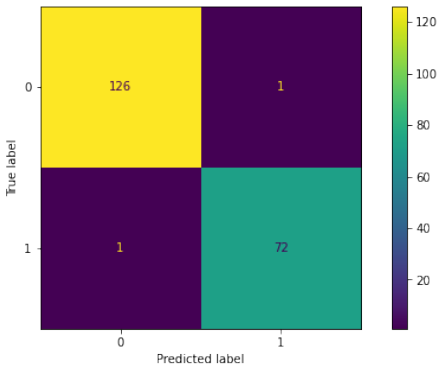
```
Лучшие гиперпараметры модели: 'dtc__max_depth': 1,'dtc__min_count': 1
Лучший счёт модели: 0.9875
Accuracy: 0.985
Recall: 0.9726027397260274
Precision: 0.9861111111111112
```

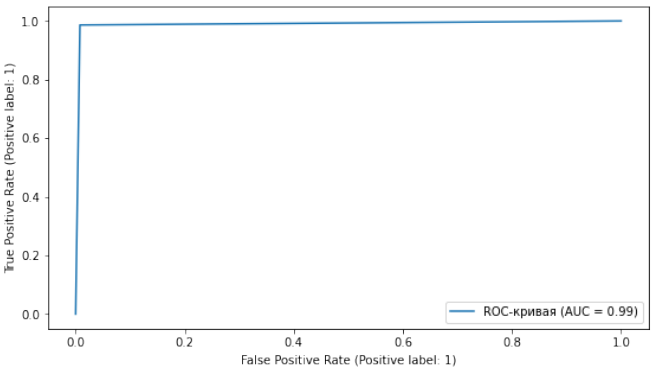




1.2 sklearn.tree.DecisionTreeClassifier

```
Лучшие гиперпараметры модели: 'dct__criterion': 'gini','dct__max_depth': 2,'dct__split
'best'
Лучший счёт модели: 0.99
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```

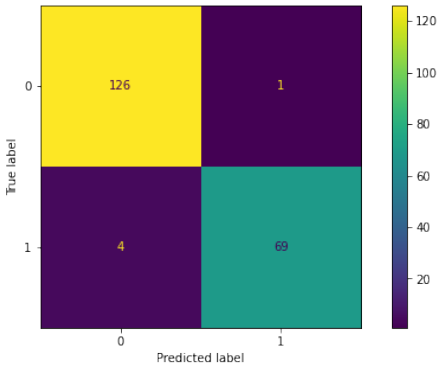


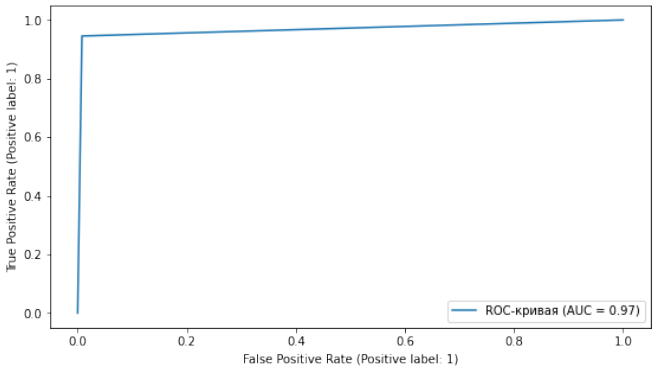


2 Случайный лес

2.1 Моя реализация

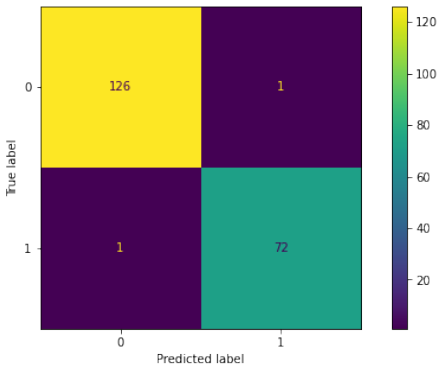
```
Лучшие гиперпараметры модели: 'rfc__max_depth': 1,'rfc__min_count': 10,'rfc__n_trees'
25
Лучший счёт модели: 0.5912499999999999
Accuracy: 0.975
Recall: 0.9452054794520548
Precision: 0.9857142857142858
```

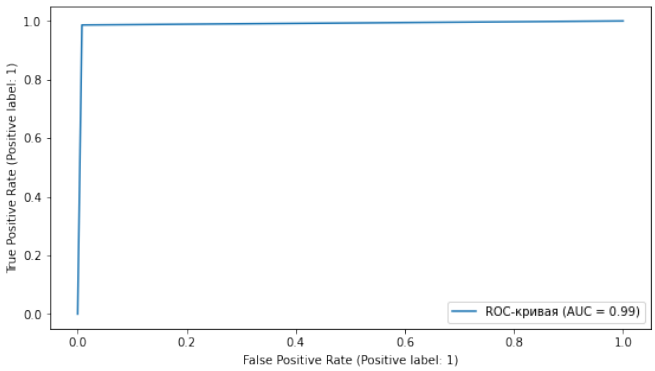




2.2 sklearn.ensemble.RandomForestClassifier

```
Лучшие гиперпараметры модели: 'dct__criterion': 'log_loss','dct__max_depth':
4,'dct__n_estimators': 25
Лучший счёт модели: 0.9950000000000001
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```

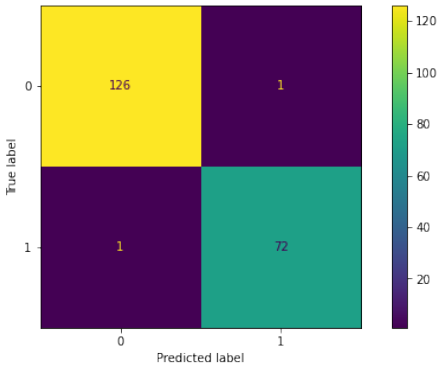


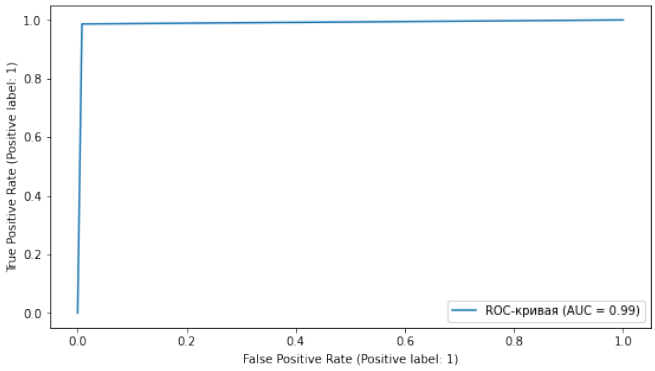


3 Градиентный бустинг

3.1 sklearn.ensemble.GradientBoostingClassifier

```
Лучшие гиперпараметры модели: 'gbc__learning_rate': 0.01,'gbc__n_estimators':
50
Лучший счёт модели: 0.99125
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```





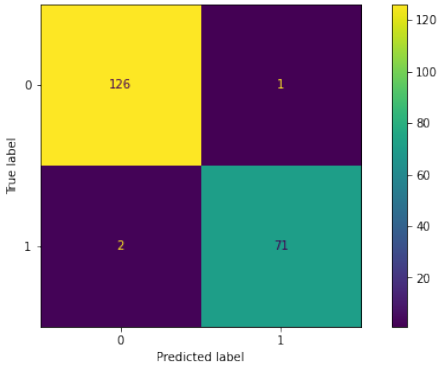
4 Мягкое и жёсткое голосование

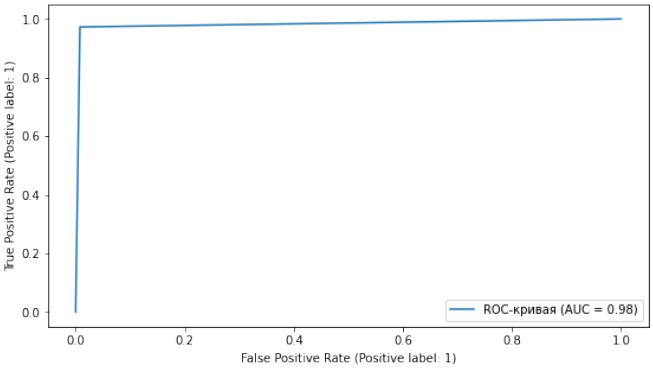
Для голосования используются одни и те же модели:

```
models = []
models.append(kNN_p(3))
models.append(LogisticRegression_p(SGD_step = 0.05, batch_size = 5, epoches = 2))
models.append(SVM_p(SGD_step = 0.01, alpha = 1.0, batch_size = 5, epoches = 4))
models.append(NaiveBayes_p())
models.append(MyDecisionTreeClassifier_p(max_depth = 2, min_count = 5))
```

4.1 Мягкое голосование

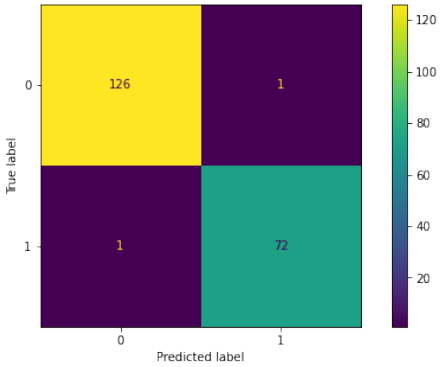
```
Accuracy: 0.985
Recall: 0.9726027397260274
Precision: 0.9861111111111112
```

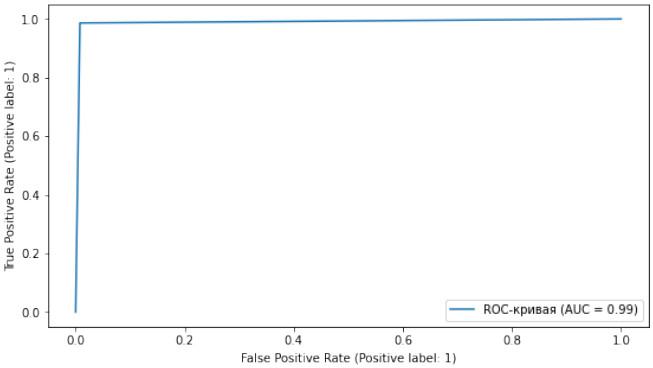




4.2 Жёсткое голосование

```
Accuracy: 0.99
Recall: 0.9863013698630136
Precision: 0.9863013698630136
```





3 Выводы

В ходе выполнения лабораторной работы я познакомился с деревом принятия решений, случайным лесом, мягким и жёстким голосованием. Из всех моделей мне больше всего понравилось реализовывать решающее дерево.

В результате набор данных Smoker Condition получилось разделить точностью 99%, как и линейными моделями. Они показали себя хорошо в прошлой работе, поэтому мягкое и жёсткое голосование на их основе показывает такой же высокий результат.

Список литературы

[1] *project-template/_template.py at master · scikit-learn-contrib*
URL: https://github.com/scikit-learn-contrib/project-template/blob/master/skltemplate/_template.py
(дата обращения: 02.06.2022).

[2] *Решающие деревья — Учебник по ML от ШАД*
URL: https://ml-handbook.ru/chapters/decision_tree/intro
(дата обращения: 02.06.2022).

[3] *sklearn.tree.DecisionTreeClassifier*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeCl
(дата обращения: 02.06.2022).

[4] *Как работает случайный лес? — Medium*
URL: https://medium.com/nuances-of-programming/как-работает-случайный-лес-56209a70
(дата обращения: 03.06.2022).

[5] *sklearn.ensemble.RandomForestClassifier*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFore
(дата обращения: 03.06.2022).

[6] *sklearn.tree.GradientBoostingClassifier*
URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBo
(дата обращения: 03.06.2022).

[7] *How to Develop Voting Ensembles With Python*
URL: https://machinelearningmastery.com/voting-ensembles-with-python/
(дата обращения: 03.06.2022).

[8] *Can you interpret probabilistically the output of a Support Vector Machine?*
URL: https://mmuratarat.github.io/2019-10-12/probabilistic-output-of-svm
(дата обращения: 03.06.2022).