

**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 хранения данных в узлах дерева опишу структуру, 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.

```
1 class Graph:
2     def __init__(self):
3         self.data = []
4         self.info = []
5         self.size = 0
6         self.add()
7
8     def can_go(self, u, c):
9         return c in self.data[u]
10
11    def go(self, u, c):
12        return self.data[u][c]
13
14    def set_go(self, u, c, v):
15        self.data[u][c] = v
16
17    def get_tree_data(self, u):
18        return self.info[u]
19
20    def set_tree_data(self, u, tree_data):
21        self.info[u] = tree_data
22
23    def add(self):
24        self.data.append(dict())
25        self.info.append(TreeData())
26        self.size += 1
27        return self.size - 1
28
29    def is_leaf(self, u):
30        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.

```
1 class DecisionTree:
2     def __init__(self, max_depth, min_count, rnd_split):
3         self.tree = Graph()
4         self.max_depth = max_depth
5         self.min_count = min_count
6         self.rnd_split = rnd_split
7         self.data = []
8
9     def get_count(self, arr):
10        res = [0 for _ in range(2)]
11        for elem in arr:
12            res[elem] += 1
13        return res
14
15    def ans(self, cnt0, cnt1):
16        if (cnt0 + cnt1 == 0):
17            return None
18        else:
19            return (cnt1) / (cnt0 + cnt1)
20
21    def ans_class(self, p):
22        return 1 if (p > 0.5) else 0
23
24    def decide(self, X):
25        return self.decide_rec(0, X)
26
27    def decide_rec(self, u, X):
28        u_data = self.tree.get_tree_data(u)
29        if (self.tree.is_leaf(u)):
30            return u_data.value
31        else:
32            dec = u_data.decide(X)
33            return self.decide_rec(self.tree.go(u, dec), X)
34
35    def make_leaf(self, u, value):
36        u_data = TreeData(value = value, leaf = True)
37        self.tree.set_tree_data(u, u_data)
38
39    def rnd_ids(self, n):
40        res = []
41        while (len(res) * len(res) < n):
42            rnd_num = randint(0, n - 1)
43            while (rnd_num in res):
44                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

```

```

94         res = (i, tmp[j][0])
95     if (res == (-1, -1)):
96         self.make_leaf(u, u_ans)
97         return
98     u_data = TreeData(value = res[1], ind = res[0], leaf = False)
99     self.tree.set_tree_data(u, u_data)
100    l, r = [], []
101    for (j, elem) in enumerate(X):
102        if (u_data.decide(elem)):
103            l.append(j)
104        else:
105            r.append(j)
106    l, r = np.array(l), np.array(r)
107    if (len(l) == 0 or len(r) == 0):
108        self.make_leaf(u, u_ans)
109        return
110    ul = self.tree.add()
111    ur = self.tree.add()
112    self.tree.set_go(u, True, ul)
113    self.tree.set_go(u, False, ur)
114    self.build_rec(ul, l, h + 1)
115    self.build_rec(ur, r, h + 1)

```

Сам классификатор содержит дерево и использует его методы.

```
1 class MyDecisionTreeClassifier(ClassifierMixin, BaseEstimator):
2     def __init__(self, max_depth = 10, min_count = 50, rnd_split = False):
3         self.max_depth = max_depth
4         self.min_count = min_count
5         self.rnd_split = rnd_split
6         self.tree = DecisionTree(max_depth, min_count, rnd_split)
7
8     def fit(self, X, y):
9         # Check that X and y have correct shape
10        X, y = check_X_y(X, y)
11        # Store the classes seen during fit
12        self.classes_ = unique_labels(y)
13
14        self.X_ = X
15        self.y_ = y
16        self.tree.build(X, y)
17        # Return the classifier
18        return self
19
20    def predict(self, X):
21        # Check is fit had been called
22        check_is_fitted(self, ['X_', 'y_'])
23
24        # Input validation
25        X = check_array(X)
26
27        p = []
28        for elem in X:
29            z = self.tree.decide(elem)
30            p.append(1 if z > 0.5 else 0)
31        return np.array(p)
```


2 Случайный лес

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

```
1 class MyRandomForestClassifier(ClassifierMixin, BaseEstimator):
2     def __init__(self, n_trees = 10, max_depth = 2, min_count = 5):
3         self.n_trees = n_trees
4         self.max_depth = max_depth
5         self.min_count = min_count
6         self.trees = [DecisionTree(max_depth, min_count, rnd_split = True) for _ in
7                         range(self.n_trees)]
8
9     def fit(self, X, y):
10         # Check that X and y have correct shape
11         X, y = check_X_y(X, y)
12         # Store the classes seen during fit
13         self.classes_ = unique_labels(y)
14
15         self.X_ = X
16         self.y_ = y
17         for (i, tree) in enumerate(self.trees):
18             rnd_ids = np.random.choice(len(X), (2 * len(X)) // self.n_trees)
19             X_sub = X[rnd_ids]
20             y_sub = y[rnd_ids]
21             tree.build(X_sub, y_sub)
22         # Return the classifier
23         return self
24
25     def predict(self, X):
26         # Check is fit had been called
27         check_is_fitted(self, ['X_', 'y_'])
28
29         # Input validation
30         X = check_array(X)
31
32         p = []
33         for elem in X:
34             z = 0
35             for tree in self.trees:
36                 z += tree.decide(elem)
37             z /= self.n_trees
38             p.append(1 if z > 0.5 else 0)
39         return np.array(p)
```

3 Мягкое голосование

Мягкое голосование принимает список классификаторов, выдающий вероятность класса. По их предсказаниям производится голосование.

```
1 class SVEnsemble(ClassifierMixin, BaseEstimator):
2     def __init__(self, models):
3         self.models = models
4
5     def fit(self, X, y):
6         # Check that X and y have correct shape
7         X, y = check_X_y(X, y)
8         # Store the classes seen during fit
9         self.classes_ = unique_labels(y)
10
11         self.X_ = X
12         self.y_ = y
13         for model in self.models:
14             model.fit(X, y)
15         # Return the classifier
16         return self
17
18     def predict(self, X):
19         # Check is fit had been called
20         check_is_fitted(self, ['X_', 'y_'])
21
22         # Input validation
23         X = check_array(X)
24
25         y, p = np.ndarray((X.shape[0],)), []
26         for model in self.models:
27             p.append(model.predict(X))
28         p = np.array(p).T
29         for i in range(len(X)):
30             p_of_labels = np.array([0 for _ in range(2)])
31             for j in range(len(self.models)):
32                 p1 = p[i][j]
33                 p0 = 1 - p1
34                 p_of_labels[0] += p0
35                 p_of_labels[1] += p1
36             y[i] = p_of_labels.argmax()
37         return y
```

4 Жёсткое голосование

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

```
1 class HVEnsemble(ClassifierMixin, BaseEstimator):
2     def __init__(self, models):
3         self.models = models
4
5     def fit(self, X, y):
6         # Check that X and y have correct shape
7         X, y = check_X_y(X, y)
8         # Store the classes seen during fit
9         self.classes_ = unique_labels(y)
10
11        self.X_ = X
12        self.y_ = y
13        for model in self.models:
14            model.fit(X, y)
15        # Return the classifier
16        return self
17
18    def predict(self, X):
19        # Check is fit had been called
20        check_is_fitted(self, ['X_', 'y_'])
21
22        # Input validation
23        X = check_array(X)
24
25        y, p = np.ndarray((X.shape[0],)), []
26        for model in self.models:
27            p.append(model.predict(X))
28        p = np.array(p).T
29        for i in range(len(X)):
30            cnt = np.array([0 for _ in range(2)])
31            for j in range(len(self.models)):
32                label = (1 if p[i][j] > 0.5 else 0)
33                cnt[label] += 1
34            y[i] = cnt.argmax()
35        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

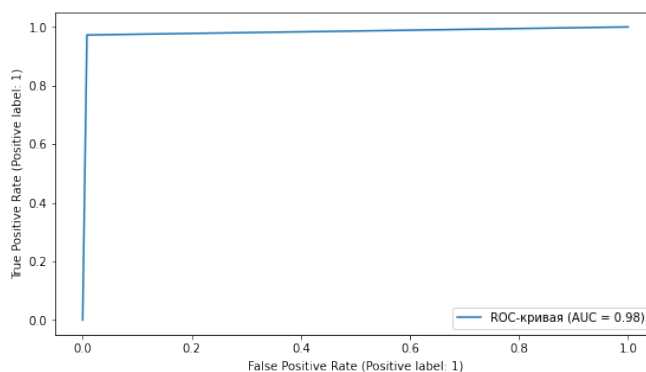
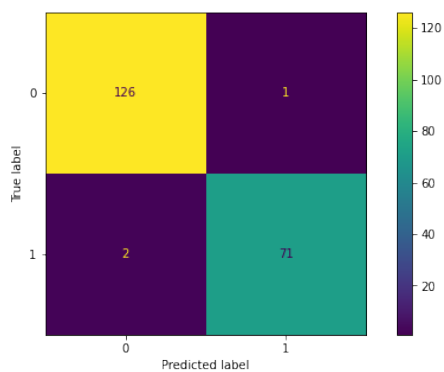
Лучшие гиперпараметры модели: `'dtc__max_depth': 1, 'dtc__min_count': 1`

Лучший счёт модели: 0.9875

Accuracy: 0.985

Recall: 0.9726027397260274

Precision: 0.9861111111111112



1.2 sklearn.tree.DecisionTreeClassifier

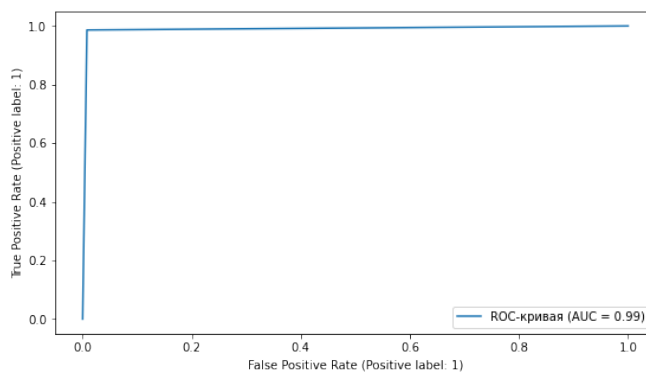
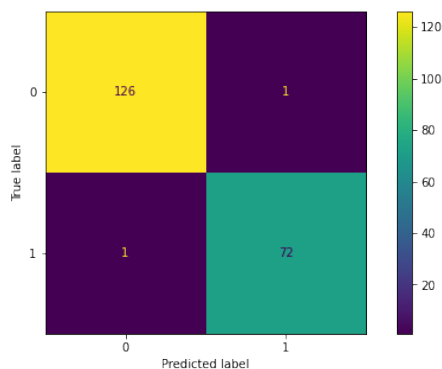
Лучшие гиперпараметры модели: 'dct__criterion': 'gini', 'dct__max_depth': 2, 'dct__splitter': 'best'

Лучший счёт модели: 0.99

Accuracy: 0.99

Recall: 0.9863013698630136

Precision: 0.9863013698630136



2 Случайный лес

2.1 Моя реализация

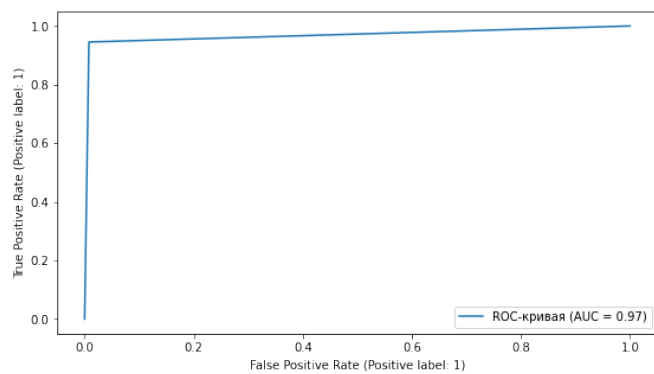
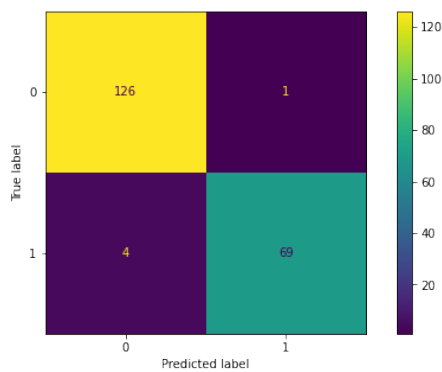
Лучшие гиперпараметры модели: 'rfc__max_depth': 1, 'rfc__min_count': 10, 'rfc__n_trees': 25

Лучший счёт модели: 0.5912499999999999

Accuracy: 0.975

Recall: 0.9452054794520548

Precision: 0.9857142857142858



2.2 sklearn.ensemble.RandomForestClassifier

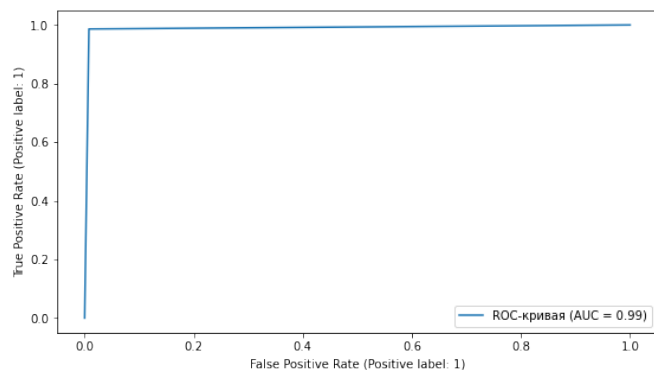
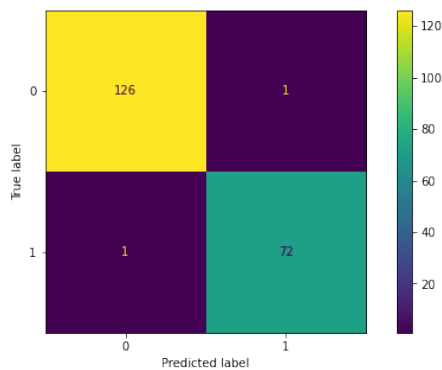
Лучшие гиперпараметры модели: 'dct__criterion': 'log_loss', 'dct__max_depth': 4, 'dct__n_estimators': 25

Лучший счёт модели: 0.9950000000000001

Accuracy: 0.99

Recall: 0.9863013698630136

Precision: 0.9863013698630136



3 Градиентный бустинг

3.1 sklearn.ensemble.GradientBoostingClassifier

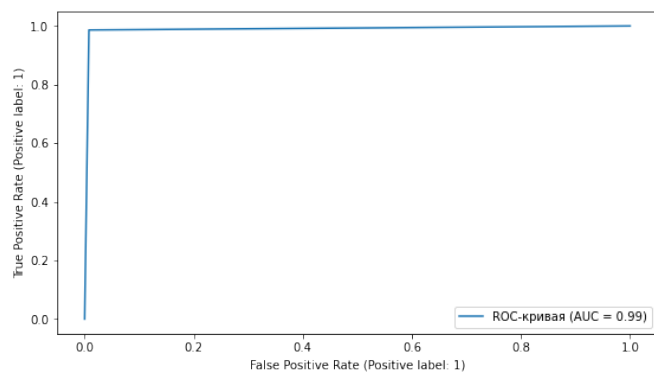
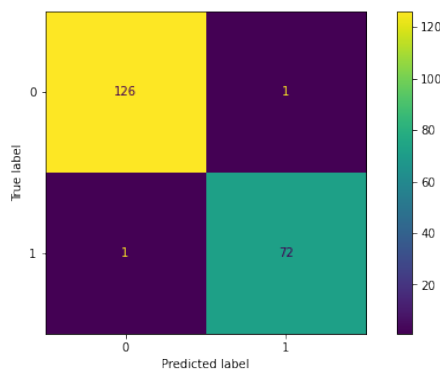
Лучшие гиперпараметры модели: 'gbc__learning_rate': 0.01, 'gbc__n_estimators': 50

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:

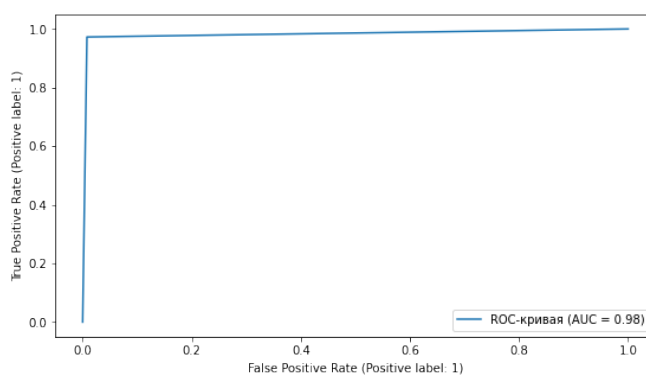
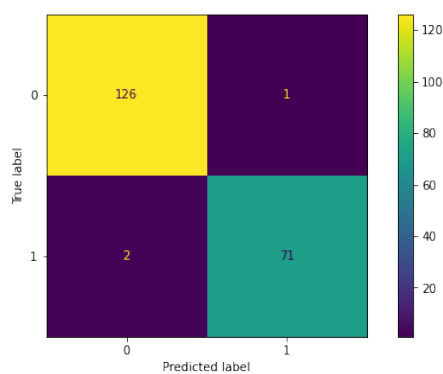
```
1 models = []
2 models.append(kNN_p(3))
3 models.append(LogisticRegression_p(SGD_step = 0.05, batch_size = 5, epoches = 2))
4 models.append(SVM_p(SGD_step = 0.01, alpha = 1.0, batch_size = 5, epoches = 4))
5 models.append(NaiveBayes_p())
6 models.append(MyDecisionTreeClassifier_p(max_depth = 2, min_count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

4.1 Мягкое голосование

Accuracy: 0.985

Recall: 0.9726027397260274

Precision: 0.9861111111111112

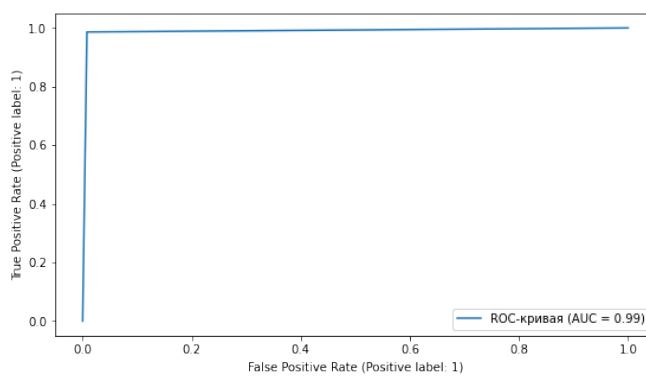
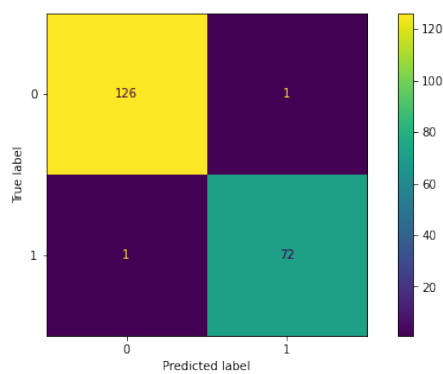


4.2 Жёсткое голосование

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.DecisionTreeClassifier.html>
(дата обращения: 02.06.2022).
- [4] *Как работает случайный лес? — Medium*
URL: <https://medium.com/nuances-of-programming/как-работает-случайный-лес-56209a70c0e0>
(дата обращения: 03.06.2022).
- [5] *sklearn.ensemble.RandomForestClassifier*
URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
(дата обращения: 03.06.2022).
- [6] *sklearn.tree.GradientBoostingClassifier*
URL: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>
(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).