Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Искусственный интеллект»

Студент: К. М. Воронов
Преподаватель: Самир Ахмед
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

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. Требования к отчету сохраняются такими же.

1 Описание

Дерево принятия решений выглядит следующим образом:

```
class DecisionTree(BaseEstimator, ClassifierMixin):

    def entropy(self, x):
        return np.sum(-x * np.log2(x + (np.abs(x) < 1e-12)))

    def __init__(self, leaf_size = 1, depth = 4, features = None):
        self.leaf_size = leaf_size
        self.depth = depth
        self.features = features

    class Node:
        def __init__(self):
            self.left = None
            self.right = None
            self.value = None
            self.number = None

    def fit(self, data, labels):
        self.root = self.Node()
        self.grow_tree(self.root, data, labels, np.arange(len(labels)), 0)
        return self

    def ans(self, y):
        k = 0

        for i in range(len(y)):
            if y[i] == 1:
                k += 1
        return k / len(y)


    def grow_tree(self, node, data, labels, idx, depth):

        x = data[idx]
        y = labels[idx]

        if (not self.depth is None) and (self.depth < depth):
            node.value = self.ans(y)
            return

        if self.leaf_size >= len(x):
            node.value = self.ans(y)
            return

        if (len(np.unique(y)) == 1):
            node.value = self.ans(y)
```

```
47              return
48
49          maximum = -1
50          split_id = 0
51          leftidx = np.array([])
52          rightidx = np.array([])
53          index = 0
54
55          for i in (self.features if self.features is not None else range(data.shape[1]))
                :
56              left = 1
57              sortid = x[:, i].argsort()
58              left_classes = np.zeros(2)
59              left_classes[y[sortid[0]]] = 1
60
61              classes = np.zeros(2)
62              for j in range(len(sortid)):
63                  classes[y[sortid[j]]] += 1
64
65              xm = self.entropy(classes / len(x))
66
67              lenxm = len(sortid)
68
69              right_classes = np.zeros(2)
70
71              for j in range(left, len(sortid)):
72                  right_classes[y[sortid[j]]] += 1
73
74              while left < len(x):
75
76                  while left < len(x) and abs(x[sortid[left-1]][i] - x[sortid[left-2]][i])
                         < 1e-6:
77                      left += 1
78                      left_classes[y[sortid[left - 1]]] += 1
79                      right_classes[y[sortid[left - 1]]] -= 1
80
81                  if left == len(x):
82                      break
83
84                  p = left_classes / left
85                  xl = self.entropy(p)
86                  p = right_classes / (len(x) - left)
87                  xr = self.entropy(p)
88
89                  lenxl = left
90                  lenxr = len(x) - left
91
92
93                  if maximum < xm * lenxm - lenxl * xl - lenxr * xr:
```

```
                maximum = xm * lenxm - lenxl * xl - lenxr * xr
                split_id = left - 1
                leftidx = sortid[0:left]
                rightidx = sortid[left:]
                index = i

            left += 1

            left_classes[y[sortid[left - 1]]] += 1
            right_classes[y[sortid[left - 1]]] -= 1


    if len(leftidx) == 0 or len(rightidx) == 0:
        node.value = self.ans(y)
        return

    node.number = index
    node.value = x[sortid[split_id]][index]
    node.left = self.Node()
    node.right = self.Node()

    self.grow_tree(node.left, x, y, leftidx , depth + 1)
    self.grow_tree(node.right, x, y, rightidx , depth + 1)


def print_tree(node, d = 0):
    if node is None:
        return
    print(' '*4*d, node.number, node.value)
    print_tree(node.left, d+1)
    print_tree(node.right, d+1)

def predict_proba(self, data):
    res = np.ndarray((data.shape[0], 2))
    for i, st in enumerate(data):
        node = self.root
        while not node.number is None:
            if st[node.number] > node.value:
                node = node.right
            else:
                node = node.left
        res[i][1] = node.value
        res[i][0] = 1 - node.value
    return res

def predict(self, data):
    return np.argmax(self.predict_proba(data), axis=1)
```

Для хранения вершин я использовал класс Node, который включает в себя левую

правую вершину, номер признака, по которому надо смотреть и значение для сравнения. В случае, когда вершина является листом, номер признака принимает значение None, а значение value ответом. Метод ans считает вероятность единички в выборке. Метод def grow_tree непосредственно строит дерево. Метод predict_proba ситает вероятности (нужно для случайного леса), а predict непосредственно ответ. В качестве параметров выступают минимальное число элементов в листе и глубина дерева.

Случайный лес тоже построен в качестве класса. Выглядит он следующим образом:

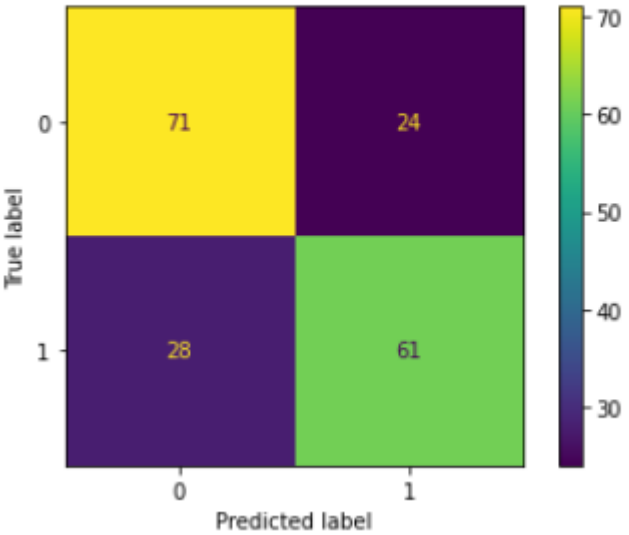
```
class RandomForest(BaseEstimator, ClassifierMixin):

    def __init__(self, features = 1, leaf_size = 1, depth = None, n_estimators = 10):
        self.leaf_size = leaf_size
        self.depth = depth
        self.features = features
        self.n_estimators = n_estimators

    def fit(self, data, labels):
        features = np.arange(data.shape[1])
        self.estimators = []
        indexes = np.arange(len(data))
        for i in range(self.n_estimators):
            np.random.shuffle(features)
            self.estimators.append(DecisionTree(leaf_size = self.leaf_size, depth =
                self.depth, features = features[:self.features]))
            idx = np.random.choice(indexes, (len(data),))
            self.estimators[-1].fit(data[idx], labels[idx])

    def predict_proba(self, data):
        pred = np.stack([est.predict_proba(data) for est in self.estimators], axis=1)
        return pred.mean(axis=1)

    def predict(self, data):
        return self.predict_proba(data).argmax(axis=1)
```

Здесь используется мягкое голосование. В качестве параметров выступают количество деревьев, количество фич, а также минимальное число элементов в листе и глубина деревьев.

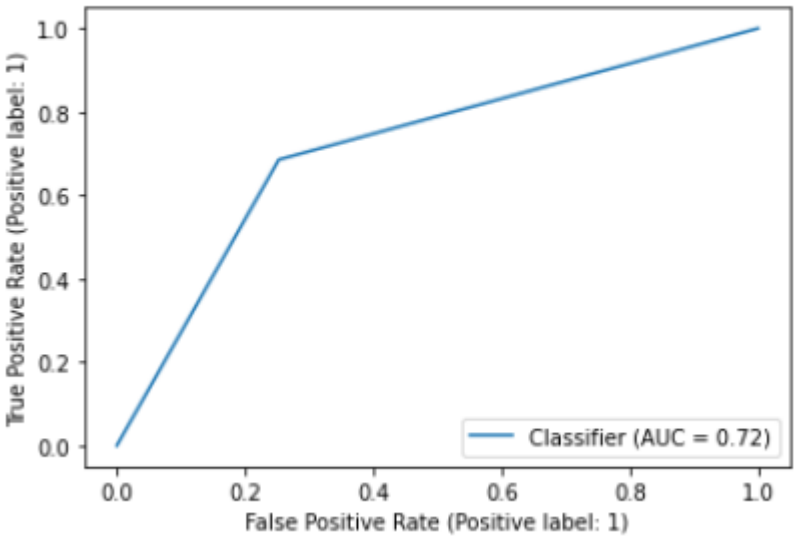
2 Результаты

Мои реализации отработали почему-то хуже, чем реализации sklearn. Для всех моделей я использовал кросс-валидацию. Мое дерево решений

```
{'depth': 10, 'leaf_size': 8}
Accuracy train: 0.6104650079209766
Accuracy tests: 0.717391304347826
```

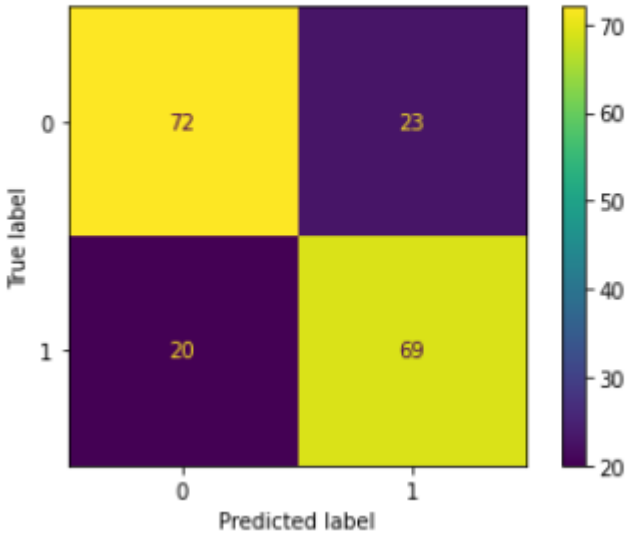



```
Precision tests: 0.7176470588235294
Recall tests: 0.6853932584269663
```

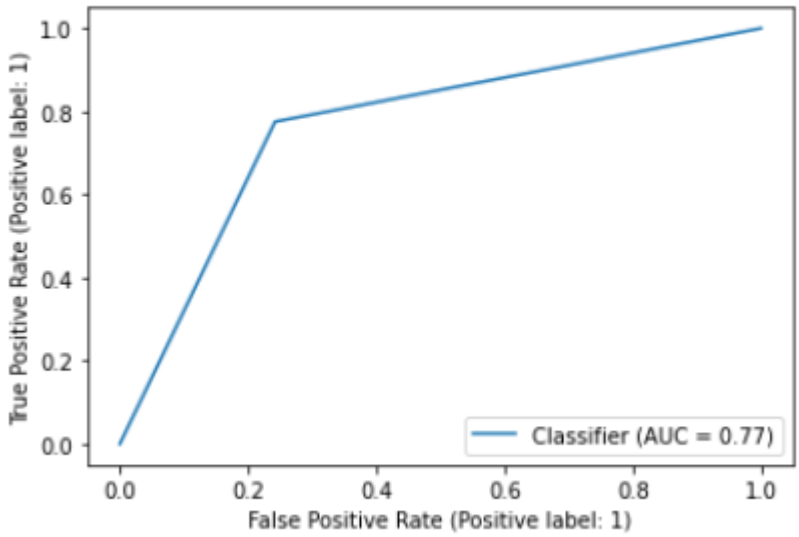


Дерево решений sklearn

```
{'max_depth': None, 'min_samples_leaf': 9}
Accuracy train: 0.8432858074736744
Accuracy tests: 0.7663043478260869
```

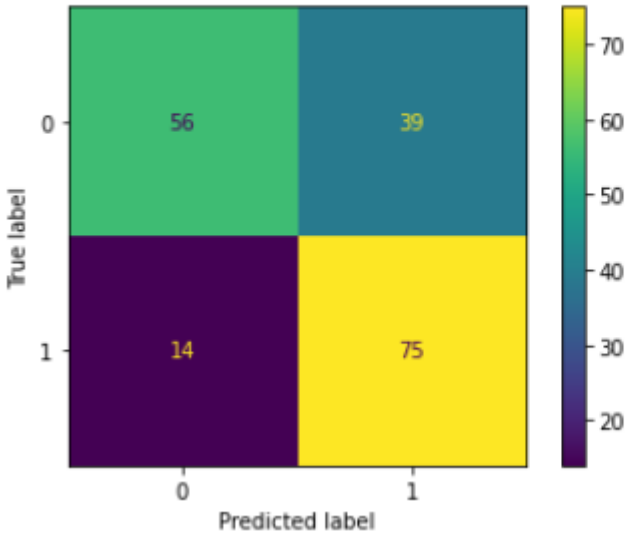



```
Precision tests: 0.75
Recall tests: 0.7752808988764045
```

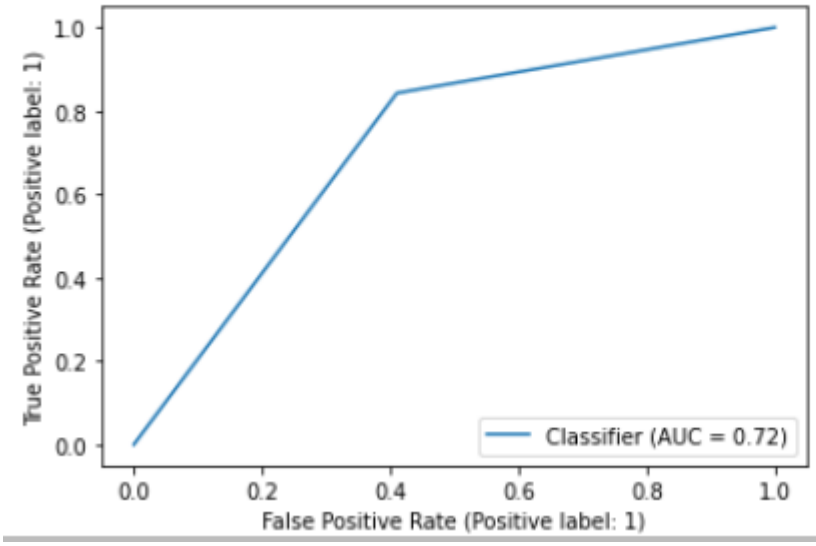


Мой случайный лес

```
{'depth': None, 'features': 3, 'leaf_size': 9, 'n_estimators': 30}
Accuracy train: 0.7737675892274718
Accuracy tests: 0.7119565217391305
```

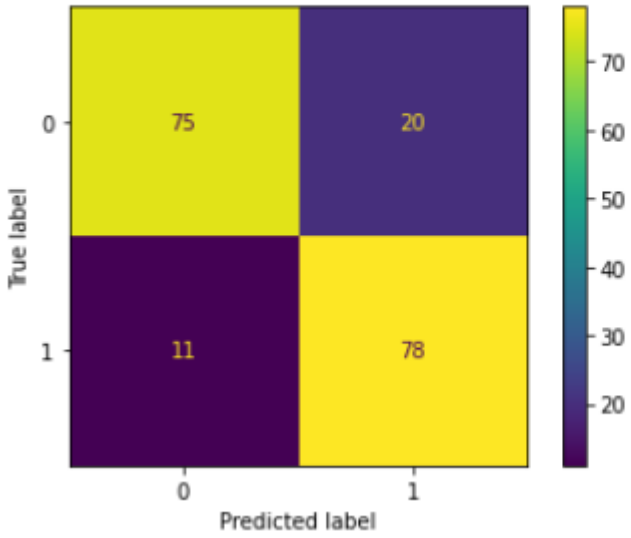



```
Precision tests: 0.6578947368421053
Recall tests: 0.8426966292134831
```

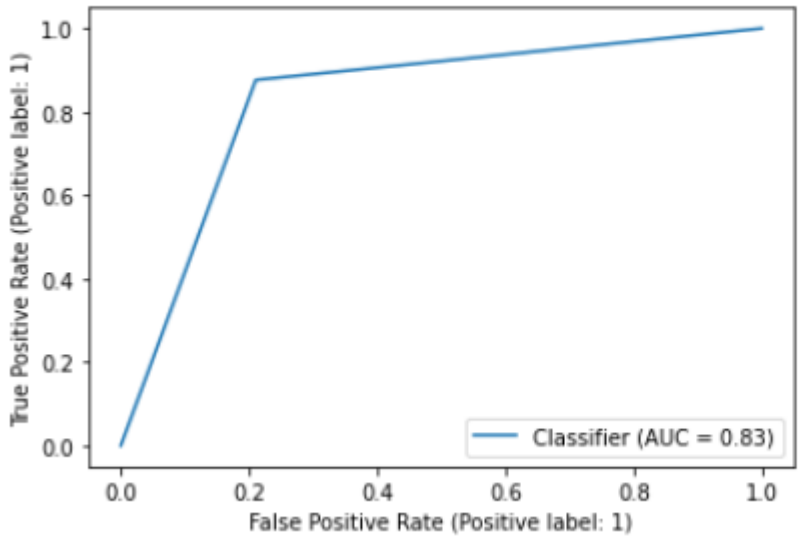


Случайный лес sklearn

```
{'max_features': None, 'min_samples_leaf': 3, 'n_estimators': 30}
Accuracy train: 0.8678501537601342
Accuracy tests: 0.8315217391304348
```

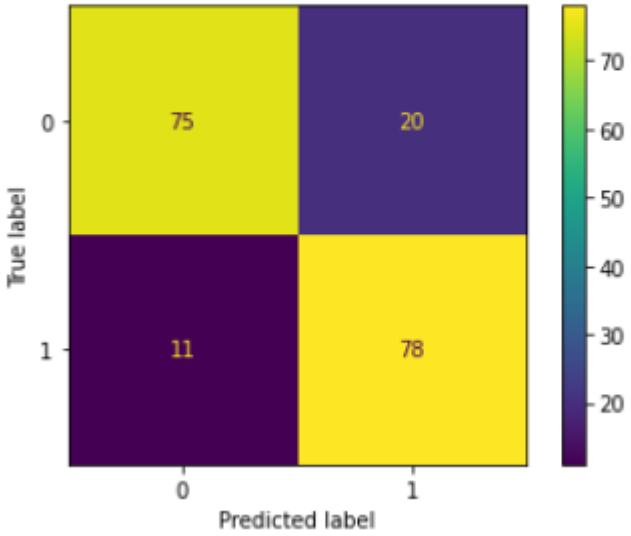



```
Precision tests: 0.7959183673469388
Recall tests: 0.8764044943820225
```

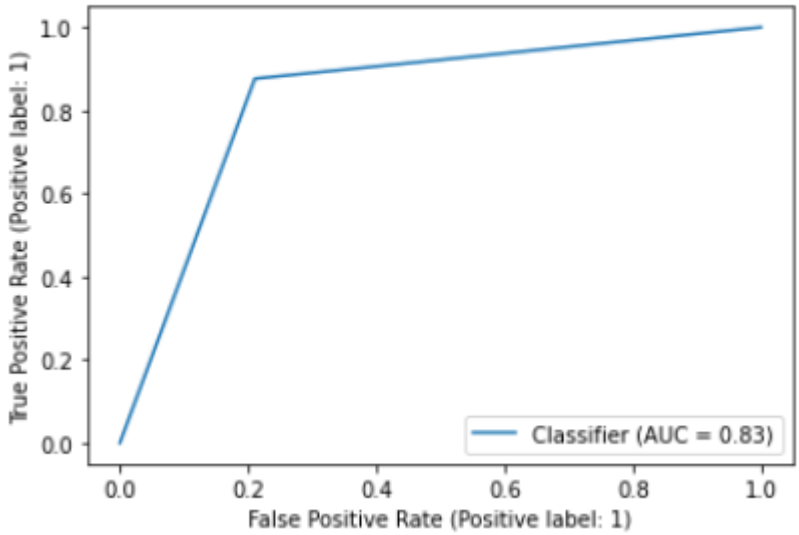


Также я воспользовался бустингом от sklearn. Он дал очень хорошие результаты, близкие к методу Байеса из прошлой лабораторной.

```
{'learning_rate': 0.1, 'n_estimators': 50}
Accuracy train: 0.882816140154692
Accuracy tests: 0.8315217391304348
```




```
Precision tests: 0.7959183673469388
Recall tests: 0.8764044943820225
```



3 Выводы

Данная лабораторная работа была непростой. Дерево решений вызывало у меня трудности в написании. Я сталкивался с разными проеблемами, например, деление категориальных признаков (они находятся в одной точке). Однако результаты вышли не совсем плохие. Точность в районе 70 % выглядит не так плохо, с учётом, что у меня почти всегда больше Recall, что для моей области это важнее, так как лучше определить здорового человека больным, чем наоборот. Бустинг, как уже говорил, дал хорошие результаты в 83 процента правильных ответов на тестовой выборке. В целом, если бы времени было бы больше, я бы попробовал большее количество моделей.