Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**Отчёт по лабораторной работе №5**

по дисциплине

«Системы искусственного интеллекта»

*Выполнил:*

Студент группы P3333

Анисимов Максим Дмитриевич

*Преподаватель:*

Авдюшина Анна Евгеньевна



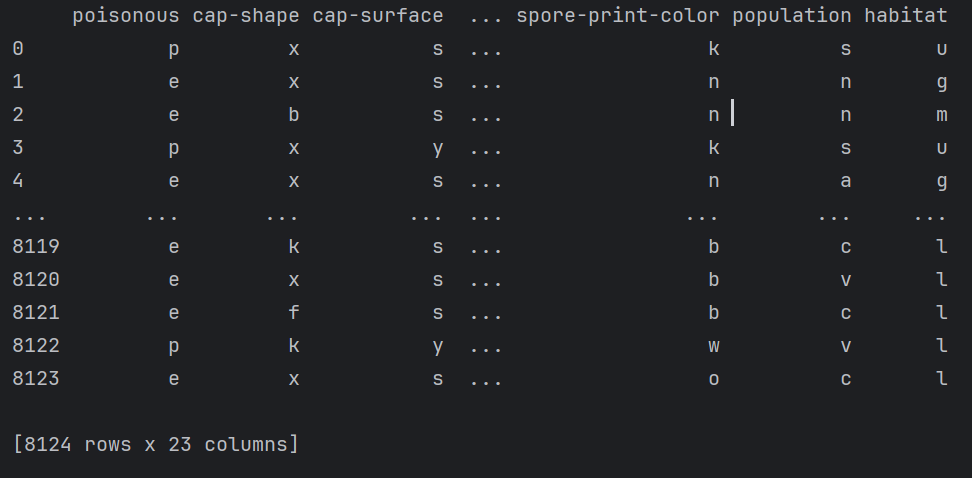
Санкт-Петербург, 2024

Для студентов с четным порядковым номером в группе – датасет с классификацией грибов

Дерево решений - способ представления правил в иерархической последовательности, где каждому объекту соответствующий узел, дающий решение

* Узел - признак
* Ребро - значение признака
* Лист - метка класса

from random import shuffle  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
import matplotlib.pyplot as plt  
  
data = pd.read\_csv("mushroom.csv", delimiter=",")  
print(data)

Вывод:

Отобрать случайным образом sqrt(n) признаков

y\_label = "poisonous"  
y\_classes = set(data[y\_label].to\_list())  
y\_classes\_count = len(y\_classes)  
  
X\_labels\_count = int(np.round(np.sqrt(len(data.columns) - 1))) # Определяем кол-во призков на основе квад. к. из кол-ва столбцов (первый столбец-метка)  
X\_labels = data.columns[1:].to\_list()  
shuffle(X\_labels) # перемешивает  
X\_labels = X\_labels[:X\_labels\_count]  
  
X\_attributes = dict()  
for label in X\_labels:  
 X\_i\_set = set(data[label].to\_list())  
 X\_attributes[label] = X\_i\_set  
  
print(X\_labels\_count, X\_labels)

Вывод:

5 ['ring-type', 'stalk-color-above-ring', 'odor', 'gill-color', 'stalk-shape']

# Обрабатываем полученные значения в данных.  
# Для каждого признака, содержащего пропущенные значения (обозначенные как "?"), она находит моду этого признака и заменяет все пропущенные значения этим значением  
def mode\_nulls(data, X\_attributes):  
 for key in X\_attributes:  
 if "?" in X\_attributes[key]:  
 nulls\_count = 0  
 X\_attributes[key] = X\_attributes[key] - set("?")  
  
 mode\_value = ""  
 mode\_count = 0  
 for value in X\_attributes[key]:  
 count = data.loc[data[key] == value].shape[0]  
 if mode\_count < count:  
 mode\_value = value  
 mode\_count = count  
  
 for i in range(data.shape[0]):  
 if data[key][i] == "?":  
 data[key][i] = mode\_value  
 nulls\_count += 1  
  
 print("{} null values in {} changed to {}".format(nulls\_count, key, mode\_value))  
 return data  
  
  
data = mode\_nulls(data, X\_attributes)  
data\_n = data[X\_labels + [y\_label]]

Реализовать без использования сторонних библиотек построение дерева решений (дерево не бинарное, numpy и pandas использовать можно, использовать список списков для реализации дерева - нельзя)

X - любой атрибут

T - исходное множество

Ti - разбиение множества T по значениям A1,A2,...,An атрибута X

Cj - j-й класс Y

freq(Cj,T) - количество элементов из T относящихся к классу Cj

Оценка среднего количества информации, необходимого для определения класса примера из множества



Оценка среднего количества информации, необходимого для определения класса примера из множества  после разбиения множества

Оценка потенциальной информации, получаемой при разбиении множества T на n подмножеств. Необходим для учета атрибутов с уникальными значениями

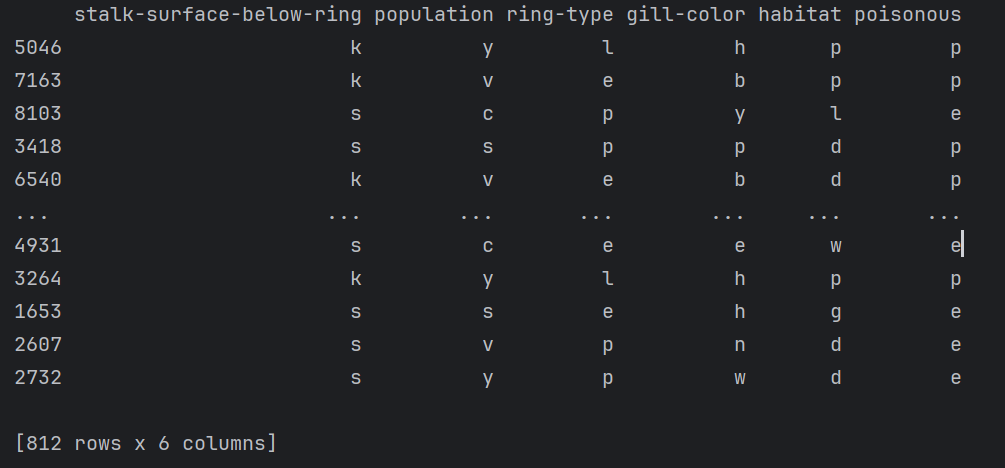
Нормированный прирост информации



Критерий Gain\_ratio считается для всех атрибутов. Выбирается атрибут с максимальным Gain\_ratio. Этот атрибут будет являться проверкой в текущем узле дерева, а затем по этому атрибуту производится дальнейшее построение дерева.

Такие же рассуждения можно применить к полученным подмножествам T\_1,T\_2,…,T\_n и продолжить рекурсивно процесс построения дерева, до тех пор, пока в узле не окажутся примеры из одного класса.

# Класс, представляющий узел дерева решений.  
class DecisionTreeNode:  
 def \_\_init\_\_(self, parent\_attribute=None, parent\_attribute\_value=None, attribute=None, entropy=0.0,  
 samples\_count=0):  
 self.parent\_attribute = parent\_attribute  
 self.parent\_attribute\_value = parent\_attribute\_value  
 self.attribute = attribute  
 self.entropy = entropy  
 self.samples\_count = samples\_count  
 self.samples = dict()  
 self.probability = dict()  
 self.prediction = None  
 self.children = list()  
  
 # Метод для предсказания класса для входного примера X  
 def predict(self, X):  
 for child in self.children:  
 if X[self.attribute] == child.parent\_attribute\_value:  
 return child.predict(X)  
  
 return self.prediction  
  
 # Метод для предсказания вероятности классов для входного примера X  
 def predict\_proba(self, X):  
 for child in self.children:  
 if X[self.attribute] == child.parent\_attribute\_value:  
 return child.predict\_proba(X)  
  
 return self.probability  
  
 def \_\_str\_\_(self):  
 key\_row, count\_row, prob\_row = "", "", ""  
 for key in sorted(self.probability):  
 count\_str = "{}".format(self.samples[key])  
 prob\_str = "{:.5f}".format(self.probability[key])  
 max\_len = max([len(count\_str), len(prob\_str), len(key)])  
 key\_row += " {}{} |".format(" " \* (max\_len - len(key)), key)  
 count\_row += " {}{} |".format(" " \* (max\_len - len(count\_str)), count\_str)  
 prob\_row += " {}{} |".format(" " \* (max\_len - len(prob\_str)), prob\_str)  
  
 parent\_attribute = str(self.parent\_attribute)  
 parent\_attribute\_value = str(self.parent\_attribute\_value)  
  
 row\_len = len(key\_row) - 1  
  
 hline = "+" + "-" \* row\_len + "+"  
 attr\_row = "{}{}".format(parent\_attribute, " " \* (row\_len - len(parent\_attribute)))  
 attr\_val\_row = "{}{}".format(parent\_attribute\_value, " " \* (row\_len - len(parent\_attribute\_value)))  
 lines = [  
 hline,  
 "|" + attr\_row + "|",  
 "|" + attr\_val\_row + "|",  
 "|" + "Count &" + " " \* (row\_len - 7) + "|",  
 "|" + "Probability" + " " \* (row\_len - 11) + "|",  
 hline,  
 "|" + key\_row,  
 hline,  
 "|" + count\_row,  
 hline,  
 "|" + prob\_row,  
 hline  
 ]  
  
 return "\n".join(lines)  
  
  
# Класс для вычисления энтропии и связанных показателей  
class InformationEntropy:  
 def \_\_init\_\_(self, df: pd.DataFrame, y\_label: str):  
 self.y\_label = y\_label  
 self.y\_classes = set(df[y\_label].to\_list())  
 self.X\_values = dict()  
 for label in df.columns:  
 if label != y\_label:  
 self.X\_values[label] = set(df[label].to\_list())  
  
 # Метод для подсчета частоты появления класса в данных  
 def freq(self, df, C\_j):  
 return df.loc[df[y\_label] == C\_j].shape[0]  
  
 # Метод для вычисления энтропии для всего набора данных  
 def info(self, df):  
 if df.shape[0] == 0:  
 return 0  
  
 result = 0  
 for y\_class in self.y\_classes:  
 freq\_c\_div\_df = self.freq(df, y\_class) / df.shape[0]  
 if freq\_c\_div\_df == 0:  
 continue  
 result -= freq\_c\_div\_df \* np.log2(freq\_c\_div\_df)  
 return result  
  
 # Метод для вычисления взвешенной энтропии для конкретного признака  
 def info\_X(self, df, X\_label):  
 if df.shape[0] == 0:  
 return 0  
  
 result = 0  
 for attr in self.X\_values[X\_label]:  
 df\_i = df.loc[df[X\_label] == attr]  
 if df\_i.shape[0] == 0:  
 continue  
 result += df\_i.shape[0] \* self.info(df\_i)  
 result /= df.shape[0]  
 return result  
  
 # Метод для вычисления "split information" для конкретного признака  
 def split\_info\_X(self, df, X\_label):  
 result = 1e-9  
 for attr in self.X\_values[X\_label]:  
 df\_i = df.loc[df[X\_label] == attr]  
 if df\_i.shape[0] == 0:  
 continue  
 df\_i\_div\_df = df\_i.shape[0] / df.shape[0]  
 result -= df\_i\_div\_df \* np.log2(df\_i\_div\_df)  
 return result  
  
 # Метод для вычисления относительного выигрыша признака  
 def gain\_ratio\_X(self, df, X\_label):  
 return (self.info(df) - self.info\_X(df, X\_label)) / self.split\_info\_X(df, X\_label)  
  
  
# Класс для построения и использования дерева решений  
class DecisionTree:  
 def \_\_init\_\_(self, max\_leaf\_entropy=0.0, max\_leaf\_samples=1):  
 assert (max\_leaf\_entropy > 0) or (  
 max\_leaf\_samples > 0), "Entropy ratio and samples count to define leaf can't be 0 at once"  
  
 self.decision\_tree\_node = None  
 self.max\_leaf\_entropy = max\_leaf\_entropy  
 self.max\_leaf\_samples = max\_leaf\_samples  
 self.info\_entropy = None  
  
 # Рекурсивный метод для построения дерева решений  
 def build\_tree(self, df: pd.DataFrame, TreeNode: DecisionTreeNode):  
 if df.shape[0] == 0:  
 return  
  
 best\_attr = None  
 best\_ratio = 0  
 for attr in self.info\_entropy.X\_values:  
 ratio = self.info\_entropy.gain\_ratio\_X(df, attr)  
 if best\_ratio < ratio:  
 best\_attr = attr  
 best\_ratio = ratio  
  
 TreeNode.attribute = best\_attr  
 TreeNode.entropy = best\_ratio  
 max\_samples\_count = 0  
 for y\_class in self.info\_entropy.y\_classes:  
 TreeNode.samples[y\_class] = df.loc[df[self.info\_entropy.y\_label] == y\_class].shape[0]  
 TreeNode.probability[y\_class] = TreeNode.samples[y\_class] / df.shape[0]  
 if max\_samples\_count < TreeNode.samples[y\_class]:  
 max\_samples\_count = TreeNode.samples[y\_class]  
 TreeNode.prediction = y\_class  
 TreeNode.samples\_count = df.shape[0]  
  
 if (TreeNode.entropy > self.max\_leaf\_entropy) and (TreeNode.samples\_count > self.max\_leaf\_samples):  
 for attr in self.info\_entropy.X\_values[best\_attr]:  
 df\_loc = df.loc[df[best\_attr] == attr]  
 if df\_loc.shape[0] > 0:  
 child = DecisionTreeNode()  
 child.parent\_attribute = best\_attr  
 child.parent\_attribute\_value = attr  
 TreeNode.children.append(child)  
 self.build\_tree(df\_loc, TreeNode.children[-1])  
  
 # Метод для обучения дерева решений на данных  
 def fit(self, df: pd.DataFrame, y\_label: str):  
 self.info\_entropy = InformationEntropy(df, y\_label)  
 self.decision\_tree\_node = DecisionTreeNode()  
 self.build\_tree(df, self.decision\_tree\_node)  
 return self  
  
 # Метод для предсказания классов для входных данных  
 def predict(self, X\_test: pd.DataFrame):  
 y\_test = []  
 for i in range(X\_test.shape[0]):  
 y\_test.append(self.decision\_tree\_node.predict(X\_test.iloc[i]))  
 return y\_test  
  
 # Метод для предсказания вероятностей классов для входных данных  
 def predict\_proba(self, X\_test: pd.DataFrame):  
 y\_test = []  
 for i in range(X\_test.shape[0]):  
 y\_test.append(self.decision\_tree\_node.predict\_proba(X\_test.iloc[i]))  
 return y\_test  
  
 def \_\_str\_\_(self):  
 result = self.decision\_tree\_node.\_\_str\_\_()  
  
 return result  
  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_n.drop(columns=[y\_label]), data\_n[y\_label], test\_size=0.9,  
 random\_state=0)  
  
df\_train = X\_train.join(y\_train)  
y\_test = y\_test.to\_list()  
df\_train

Вывод:

После выполнения у нас есть список predictions, который содержит предсказанные метки классов для тестовой выборки с использованием обученного дерева решений

[812 rows x 6 columns]  
└──+-------------------+  
│ |None |  
│ |None |  
│ |Count & |  
│ |Probability |  
│ +-------------------+  
│ | e | p |  
│ +-------------------+  
│ | 407 | 405 |  
│ +-------------------+  
│ | 0.50123 | 0.49877 |  
│ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |l |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 40 | 0 |  
│ │ +-------------------+  
│ │ | 1.00000 | 0.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |p |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 31 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |n |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 328 | 15 |  
│ │ +-------------------+  
│ │ | 0.95627 | 0.04373 |  
│ │ +-------------------+  
│ │ ├──+-------------------+  
│ │ │ |stalk-shape |  
│ │ │ |t |  
│ │ │ |Count & |  
│ │ │ |Probability |  
│ │ │ +-------------------+  
│ │ │ | e | p |  
│ │ │ +-------------------+  
│ │ │ | 236 | 0 |  
│ │ │ +-------------------+  
│ │ │ | 1.00000 | 0.00000 |  
│ │ │ +-------------------+  
│ │ └──+-------------------+  
│ │ │ |stalk-shape |  
│ │ │ |e |  
│ │ │ |Count & |  
│ │ │ |Probability |  
│ │ │ +-------------------+  
│ │ │ | e | p |  
│ │ │ +-------------------+  
│ │ │ | 92 | 15 |  
│ │ │ +-------------------+  
│ │ │ | 0.85981 | 0.14019 |  
│ │ │ +-------------------+  
│ │ │ ├──+-------------------+  
│ │ │ │ |stalk-color-below-ring|  
│ │ │ │ |w |  
│ │ │ │ |Count & |  
│ │ │ │ |Probability |  
│ │ │ │ +-------------------+  
│ │ │ │ | e | p |  
│ │ │ │ +-------------------+  
│ │ │ │ | 58 | 11 |  
│ │ │ │ +-------------------+  
│ │ │ │ | 0.84058 | 0.15942 |  
│ │ │ │ +-------------------+  
│ │ │ │ ├──+-------------------+  
│ │ │ │ │ |population |  
│ │ │ │ │ |c |  
│ │ │ │ │ |Count & |  
│ │ │ │ │ |Probability |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | e | p |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 10 | 2 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 0.83333 | 0.16667 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ ├──+-------------------+  
│ │ │ │ │ |population |  
│ │ │ │ │ |n |  
│ │ │ │ │ |Count & |  
│ │ │ │ │ |Probability |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | e | p |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 14 | 0 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ ├──+-------------------+  
│ │ │ │ │ |population |  
│ │ │ │ │ |v |  
│ │ │ │ │ |Count & |  
│ │ │ │ │ |Probability |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | e | p |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 8 | 9 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 0.47059 | 0.52941 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ ├──+-------------------+  
│ │ │ │ │ |population |  
│ │ │ │ │ |s |  
│ │ │ │ │ |Count & |  
│ │ │ │ │ |Probability |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | e | p |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 13 | 0 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ └──+-------------------+  
│ │ │ │ │ |population |  
│ │ │ │ │ |y |  
│ │ │ │ │ |Count & |  
│ │ │ │ │ |Probability |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | e | p |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 13 | 0 |  
│ │ │ │ │ +-------------------+  
│ │ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ │ +-------------------+  
│ │ │ ├──+-------------------+  
│ │ │ │ |stalk-color-below-ring|  
│ │ │ │ |n |  
│ │ │ │ |Count & |  
│ │ │ │ |Probability |  
│ │ │ │ +-------------------+  
│ │ │ │ | e | p |  
│ │ │ │ +-------------------+  
│ │ │ │ | 5 | 0 |  
│ │ │ │ +-------------------+  
│ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ +-------------------+  
│ │ │ ├──+-------------------+  
│ │ │ │ |stalk-color-below-ring|  
│ │ │ │ |o |  
│ │ │ │ |Count & |  
│ │ │ │ |Probability |  
│ │ │ │ +-------------------+  
│ │ │ │ | e | p |  
│ │ │ │ +-------------------+  
│ │ │ │ | 20 | 0 |  
│ │ │ │ +-------------------+  
│ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ +-------------------+  
│ │ │ ├──+-------------------+  
│ │ │ │ |stalk-color-below-ring|  
│ │ │ │ |y |  
│ │ │ │ |Count & |  
│ │ │ │ |Probability |  
│ │ │ │ +-------------------+  
│ │ │ │ | e | p |  
│ │ │ │ +-------------------+  
│ │ │ │ | 0 | 4 |  
│ │ │ │ +-------------------+  
│ │ │ │ | 0.00000 | 1.00000 |  
│ │ │ │ +-------------------+  
│ │ │ └──+-------------------+  
│ │ │ │ |stalk-color-below-ring|  
│ │ │ │ |e |  
│ │ │ │ |Count & |  
│ │ │ │ |Probability |  
│ │ │ │ +-------------------+  
│ │ │ │ | e | p |  
│ │ │ │ +-------------------+  
│ │ │ │ | 9 | 0 |  
│ │ │ │ +-------------------+  
│ │ │ │ | 1.00000 | 0.00000 |  
│ │ │ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |c |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 25 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |a |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 39 | 0 |  
│ │ +-------------------+  
│ │ | 1.00000 | 0.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |s |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 59 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |f |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 210 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
│ ├──+-------------------+  
│ │ |odor |  
│ │ |y |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 62 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
│ └──+-------------------+  
│ │ |odor |  
│ │ |m |  
│ │ |Count & |  
│ │ |Probability |  
│ │ +-------------------+  
│ │ | e | p |  
│ │ +-------------------+  
│ │ | 0 | 3 |  
│ │ +-------------------+  
│ │ | 0.00000 | 1.00000 |  
│ │ +-------------------+  
None

Провести оценку реализованного алгоритма с использованием Accuracy, precision и recall

Accuracy (точность) - это метрика, которая измеряет общую правильность классификации модели.

Accuracy = True Positives+True Negatives/ Total Samples

Precision (Точность) - это метрика, которая измеряет долю объектов, классифицированных как положительные и действительно являющихся положительными.

Precision = True Positives/True Positives+False Positives

Recall (Полнота) - это метрика, которая измеряет, как много объектов положительного класса было правильно классифицировано моделью.

Recall = True Positives/(True Positives+False Negatives)

# функция вычисляет элементы матрицы ошибок  
def confusion(y\_true, y\_pred, y\_positive="e"):  
 TP, FP, FN, TN = 0, 0, 0, 0  
 for i in range(len(y\_true)):  
 if y\_pred[i] == y\_positive:  
 if y\_true[i] == y\_positive:  
 TP += 1  
 else:  
 FP += 1  
 else:  
 if y\_true[i] == y\_positive:  
 FN += 1  
 else:  
 TN += 1  
 return TP, FP, FN, TN  
  
  
# функция строит ROC-кривую  
def TPR\_by\_FPR(y\_true, y\_probs, y\_positive="e", y\_negative="p", lines\_count=0):  
 use\_probs\_for\_line = lines\_count <= 0  
 y\_probs\_sorted = sorted([v[y\_positive] for v in y\_probs], reverse=True)  
  
 # Получение следующее значение прямой классификации  
 def get\_classification\_line\_value(i):  
 if use\_probs\_for\_line:  
 return y\_probs\_sorted[i]  
 return 1 - i / lines\_count  
  
 # Количество точек в ROC  
 points\_count = lines\_count + 1  
 if use\_probs\_for\_line:  
 points\_count = len(y\_probs\_sorted)  
 # Списки значений  
 FPR\_values, TPR\_values = [0], [0]  
 last\_line\_value = -123  
 for i in range(points\_count):  
 classification\_line\_value = get\_classification\_line\_value(i)  
 if abs(last\_line\_value - classification\_line\_value) < 1e-3:  
 continue  
 last\_line\_value = classification\_line\_value  
 # Ставим порог и отсекаем позитивные и негативные  
 y\_pred = []  
 for j in range(len(y\_probs)):  
 y\_pred.append(y\_positive if y\_probs[j][y\_positive] >= classification\_line\_value else y\_negative)  
 TP, FP, FN, TN = confusion(y\_true, y\_pred, y\_positive)  
  
 try:  
 FPR = FP / (TN + FP)  
 TPR = TP / (TP + FN)  
  
 FPR\_values.append(FPR)  
 TPR\_values.append(TPR)  
 except:  
 pass  
  
 return FPR\_values, TPR\_values  
  
  
# функция строит PR кривую  
def Precision\_by\_Recall(y\_true, y\_probs, y\_positive="e", y\_negative="p", lines\_count=0):  
 use\_probs\_for\_line = lines\_count <= 0  
 y\_probs\_sorted = sorted([v[y\_positive] for v in y\_probs], reverse=True)  
  
 def get\_classification\_line\_value(i):  
 if use\_probs\_for\_line:  
 return y\_probs\_sorted[i]  
 return 1 - i / lines\_count  
  
 points\_count = lines\_count + 1  
 if use\_probs\_for\_line:  
 points\_count = len(y\_probs\_sorted)  
  
 Recall\_values, Precision\_values = [0], [1]  
 last\_line\_value = -123  
 for i in range(points\_count):  
 classification\_line\_value = get\_classification\_line\_value(i)  
 if abs(last\_line\_value - classification\_line\_value) < 1e-3:  
 continue  
 last\_line\_value = classification\_line\_value  
  
 y\_pred = []  
 for j in range(len(y\_probs)):  
 y\_pred.append(y\_positive if y\_probs[j][y\_positive] >= classification\_line\_value else y\_negative)  
 TP, FP, FN, TN = confusion(y\_true, y\_pred, y\_positive)  
  
 try:  
 Recall = TP / (TP + FN)  
 Precision = TP / (TP + FP)  
  
 Recall\_values.append(Recall)  
 Precision\_values.append(Precision)  
 except:  
 pass  
  
 return Recall\_values, Precision\_values  
  
  
probs = dt.predict\_proba(X\_test)  
  
# вычисление элементов матрицы ошибок и выводит метрики производительности классификации  
TP, FP, FN, TN = confusion(y\_test, predictions)  
print("TP, FP, FN, TN = {}, {}, {}, {}".format(TP, FP, FN, TN))  
print("Precision = {}".format(TP / (TP + FP)))  
print("Recall = {}".format(TP / (TP + FN)))  
print("Accuracy = {}".format((TP + TN) / (TP + FP + FN + TN)))  
print(X\_test.columns.to\_list())

TP, FP, FN, TN = 3539, 476, 262, 3035

Precision = 0.8814445828144458

Recall = 0.9310707708497764

Accuracy = 0.899070021881838

['cap-shape', 'cap-surface', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'population']

roc\_x, roc\_y = TPR\_by\_FPR(y\_test, probs, "e", "p", 0) # 0 указывает, что используется полное распределение вероятностей, а не фиксированное количество линий  
pr\_x, pr\_y = Precision\_by\_Recall(y\_test, probs, "e", "p", 0)

Построить кривые AUC-ROC и AUC-PR

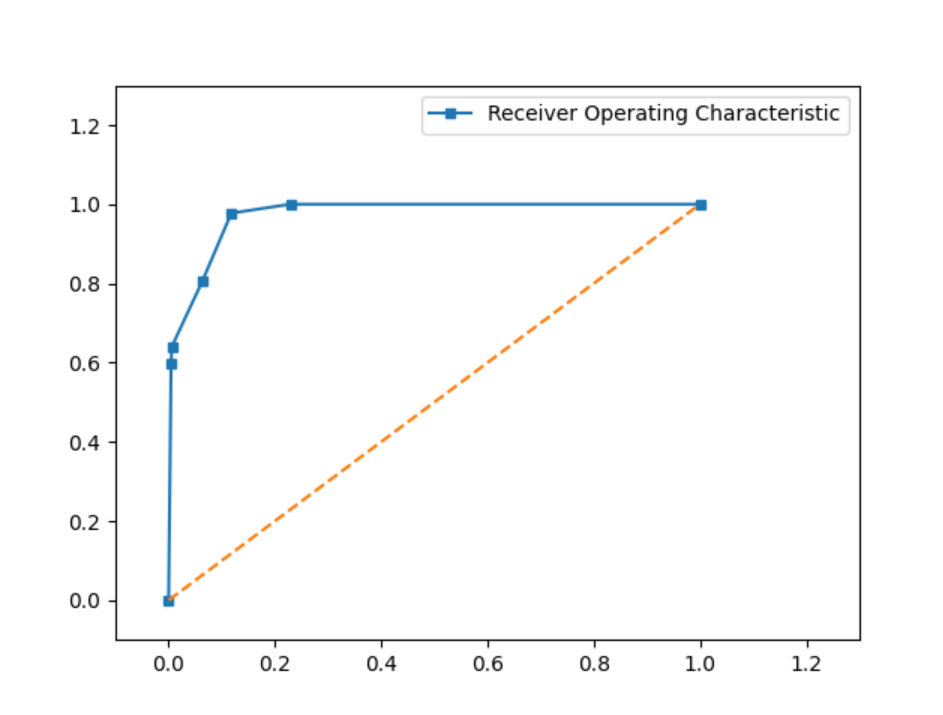
AUC-ROC - это метрика, используемая для оценки качества бинарной классификационной модели

График зависимости True Positive Rate (чувствительности) от False Positive Rate при изменении порога классификации

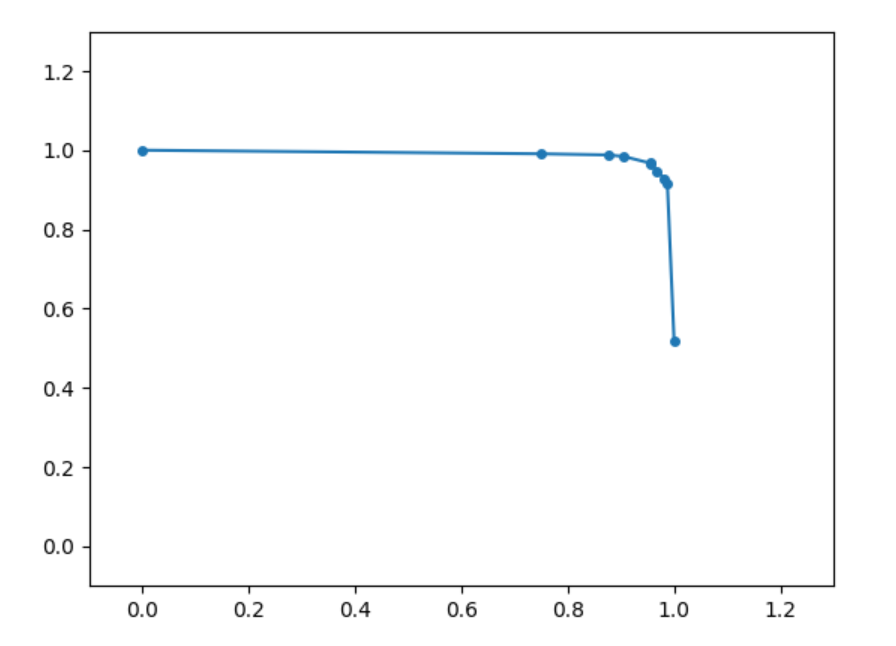
AUC-PR - это метрика, также используемая для оценки качества бинарной классификационной модели, особенно в случаях, когда классы несбалансированы.

График зависимости Precision (точности) от Recall (полноты) при изменении порога классификации.

# roc  
plt.plot(roc\_x, roc\_y, 's-', markersize = 4, label = 'Receiver Operating Characteristic')  
  
# y=x line  
plt.plot([0, 1], [0, 1], '--')  
plt.xlim(-0.1, 1.3)  
plt.ylim(-0.1, 1.3)  
plt.legend(loc='upper right')  
  
plt.show()



# precision-recall  
plt.plot(pr\_x, pr\_y, 'o-', markersize = 4, label = 'Precision Recall')  
plt.xlim(-0.1, 1.3)  
plt.ylim(-0.1, 1.3)  
  
plt.show()



def integrate\_traps(x\_values, y\_values):  
 return sum(  
 [(y\_values[i] + y\_values[i + 1]) \* (x\_values[i + 1] - x\_values[i]) / 2 for i in range(len(x\_values) - 1)])  
  
  
print("Area under curve ROC = {}".format(integrate\_traps(roc\_x, roc\_y)))  
print("Area under curve PR = {}".format(integrate\_traps(pr\_x, pr\_y)))

Area under curve ROC = 0.8359373565741556

Area under curve PR = 0.8305969765970881