Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

Системы искусственного интеллекта

Лабораторная работа №5

Деревья решений

Группа: P3324

Выполнил: Маликов Глеб Игоревич

Преподаватель:

Королёва Юлия Александровна

Санкт-Петербург

2024г.

**Оглавление**

[Введение 3](#_Toc178010343)

[Описание метода 4](#_Toc178010344)

[Реализация метода 5](#_Toc178010345)

[Результаты выполнения 6](#_Toc178010346)

[Примеры использования метода 7](#_Toc178010347)

# Введение

В данной лабораторной работе реализуется алгоритм построения деревьев решений без использования сторонних библиотек. Для решения задачи бинарной классификации использовался датасет с данными об успеваемости студентов инженерного и педагогического факультетов. Основной задачей было классифицировать студентов на успешных и неуспешных на основе их оценок (OUTPUT Grade). В рамках работы были реализованы методы оценки модели, включая Accuracy, Precision, Recall, а также построены кривые AUC-ROC и AUC-PR.

# Описание метода

Деревья решений — это модель, которая принимает решения на основе последовательных вопросов о признаках данных. Алгоритм C4.5, разработанный Россом Куином, является одним из наиболее известных методов построения деревьев решений. Он использует отношение прироста информации (Information Gain Ratio) для выбора оптимальных признаков на каждом шаге разбиения.

Основные этапы работы алгоритма C4.5:

1. **Выбор наилучшего признака для разбиения**: используется отношение прироста информации, которое учитывает как прирост информации от разбиения, так и "разделительную информацию" (split information), чтобы избежать предпочтения признаков с большим количеством уникальных значений.
2. **Разбиение данных**: Данные разделяются на подмножества на основе выбранного признака.
3. **Рекурсия**: Процесс повторяется для каждого подмножества до тех пор, пока все экземпляры в подмножестве принадлежат к одному классу или пока не исчерпаются признаки.
4. **Остановка**: когда выполнение условий для остановки достигнуто, создаётся листовой узел с предсказанием класса.

**Критерий разбиения**

Оценка среднего количества информации, необходимого для определения класса примера из множества (энтропия):

Оценка среднего количества информации, необходимого для определения класса примера из множества после разбиения множества по (условная энтропия):

# Реализация метода

В ходе лабораторной работы была реализована модель дерева решений на языке Python. Ниже представлен код реализации классов DecisionTreeNode и DecisionTree, а также основной логики обучения и оценки модели.

class DecisionTreeNode:  
 def \_\_init\_\_(self, attribute=None, branches=None, is\_leaf=False, prediction=None, class\_counts=None):  
 self.attribute = attribute # Атрибут для разбиения  
 self.branches = branches or {} # Дочерние узлы  
 self.is\_leaf = is\_leaf # Является ли узел листом  
 self.prediction = prediction # Предсказание класса (для листа)  
 self.class\_counts = class\_counts # Количество классов (для вероятностей)  
  
 def predict\_instance(self, instance):  
 if self.is\_leaf:  
 return self.prediction  
 attribute\_value = instance.get(self.attribute)  
 if attribute\_value in self.branches:  
 return self.branches[attribute\_value].predict\_instance(instance)  
 else:  
 # Если значение атрибута не встречалось при обучении, возвращаем наиболее частый класс  
 return self.prediction  
  
 def predict\_proba\_instance(self, instance):  
 if self.is\_leaf:  
 total = sum(self.class\_counts.values())  
 proba = self.class\_counts.get(1, 0) / (total + 1e-9)  
 return proba  
 attribute\_value = instance.get(self.attribute)  
 if attribute\_value in self.branches:  
 return self.branches[attribute\_value].predict\_proba\_instance(instance)  
 else:  
 # Если значение атрибута не встречалось при обучении, возвращаем вероятности наиболее частого класса  
 total = sum(self.class\_counts.values())  
 proba = self.class\_counts.get(1, 0) / (total + 1e-9)  
 return proba  
  
class DecisionTree:  
 def \_\_init\_\_(self):  
 self.root = None  
  
 def entropy(self, y):  
 class\_counts = y.value\_counts()  
 probabilities = class\_counts / len(y)  
 return -sum(probabilities \* np.log2(probabilities + 1e-9)) # Добавляем 1e-9 для избежания log(0)  
  
 def information\_gain\_ratio(self, X, y, attribute):  
 # Энтропия до разбиения  
 entropy\_before = self.entropy(y)  
  
 # Группировка по значению атрибута  
 values, counts = np.unique(X[attribute], return\_counts=True)  
 weighted\_entropy = 0  
 split\_info = 0  
 for v, count in zip(values, counts):  
 subset\_y = y[X[attribute] == v]  
 weighted\_entropy += (count / len(y)) \* self.entropy(subset\_y)  
 split\_info -= (count / len(y)) \* np.log2((count / len(y)) + 1e-9)  
  
 info\_gain = entropy\_before - weighted\_entropy  
 gain\_ratio = info\_gain / (split\_info + 1e-9)  
 return gain\_ratio  
  
 def build\_tree\_recursive(self, X, y, attributes, default\_class=None):  
 if len(y) == 0:  
 return DecisionTreeNode(is\_leaf=True, prediction=default\_class)  
 elif len(y.unique()) == 1:  
 return DecisionTreeNode(is\_leaf=True, prediction=y.iloc[0], class\_counts=y.value\_counts().to\_dict())  
 elif len(attributes) == 0:  
 majority\_class = y.mode()[0]  
 class\_counts = y.value\_counts().to\_dict()  
 return DecisionTreeNode(is\_leaf=True, prediction=majority\_class, class\_counts=class\_counts)  
 else:  
 default\_class = y.mode()[0]  
 # Выбор атрибута с максимальным Gain Ratio  
 gain\_ratios = {attr: self.information\_gain\_ratio(X, y, attr) for attr in attributes}  
 best\_attr = max(gain\_ratios, key=gain\_ratios.get)  
  
 # Создание узла  
 class\_counts = y.value\_counts().to\_dict()  
 tree = DecisionTreeNode(attribute=best\_attr, prediction=default\_class, class\_counts=class\_counts)  
  
 # Разбиение по значениям лучшего атрибута  
 for attr\_val in np.unique(X[best\_attr]):  
 subset\_X = X[X[best\_attr] == attr\_val].drop(columns=[best\_attr])  
 subset\_y = y[X[best\_attr] == attr\_val]  
 subtree = self.build\_tree\_recursive(  
 subset\_X,  
 subset\_y,  
 [attr for attr in attributes if attr != best\_attr],  
 default\_class=default\_class  
 )  
 tree.branches[attr\_val] = subtree  
 return tree  
  
 def build\_tree(self, X, y):  
 attributes = list(X.columns)  
 self.root = self.build\_tree\_recursive(X, y, attributes)  
 print("Дерево решений построено.")  
  
 def predict\_instance(self, instance):  
 return self.root.predict\_instance(instance)  
  
 def predict\_proba\_instance(self, instance):  
 return self.root.predict\_proba\_instance(instance)  
  
 def predict(self, X):  
 return X.apply(self.predict\_instance, axis=1).values  
  
 def predict\_proba(self, X):  
 return X.apply(self.predict\_proba\_instance, axis=1).values  
  
 def compute\_metrics(self, y\_true, y\_pred):  
 TP = np.sum((y\_true == 1) & (y\_pred == 1))  
 TN = np.sum((y\_true == 0) & (y\_pred == 0))  
 FP = np.sum((y\_true == 0) & (y\_pred == 1))  
 FN = np.sum((y\_true == 1) & (y\_pred == 0))  
  
 confusion\_matrix = np.array([[TN, FP], [FN, TP]])  
 accuracy = (TP + TN) / (TP + TN + FP + FN + 1e-9)  
 precision = TP / (TP + FP + 1e-9)  
 recall = TP / (TP + FN + 1e-9)  
  
 return confusion\_matrix, accuracy, precision, recall  
  
 def calculate\_auc(self, y\_true, y\_scores, curve\_type='ROC'):  
 # Сортируем по убыванию вероятностей  
 desc\_score\_indices = np.argsort(-y\_scores)  
 y\_true = y\_true[desc\_score\_indices]  
 y\_scores = y\_scores[desc\_score\_indices]  
  
 # Уникальные значения  
 distinct\_value\_indices = np.where(np.diff(y\_scores))[0]  
 threshold\_idxs = np.r\_[distinct\_value\_indices, y\_true.size - 1]  
  
 tps = np.cumsum(y\_true)[threshold\_idxs]  
 fps = 1 + threshold\_idxs - tps  
  
 tps = np.r\_[0, tps]  
 fps = np.r\_[0, fps]  
  
 if curve\_type == 'ROC':  
 fpr = fps / fps[-1] if fps[-1] != 0 else fps  
 tpr = tps / tps[-1] if tps[-1] != 0 else tps  
 auc = np.trapezoid(tpr, fpr)  
 return fpr, tpr, auc  
 elif curve\_type == 'PR':  
 precision = tps / (tps + fps + 1e-9)  
 recall = tps / tps[-1] if tps[-1] != 0 else tps  
 auc = np.trapezoid(precision, recall)  
 return recall, precision, auc  
 else:  
 raise ValueError("curve\_type должен быть 'ROC' или 'PR'")  
  
 def calculate\_auc\_metrics(self, y\_true, y\_scores):  
 # Расчет AUC-ROC  
 fpr, tpr, auc\_roc = self.calculate\_auc(y\_true, y\_scores, curve\_type='ROC')  
 # Расчет AUC-PR  
 recall, precision, auc\_pr = self.calculate\_auc(y\_true, y\_scores, curve\_type='PR')  
 return (fpr, tpr, auc\_roc), (recall, precision, auc\_pr)

# Результаты выполнения

Для задачи бинарной классификации необходимо преобразовать много классовую метку «OUTPUT Grade» в бинарную: успешный/неуспешный студент.

def preprocess\_data(data, success\_threshold):  
 output\_label, id\_label = "OUTPUT Grade", "Student ID"  
 success\_label = "Success"  
   
 # Бинаризация метки  
 output\_column = data[output\_label]  
 success\_column = output\_column.apply(lambda x: 1 if x >= success\_threshold else 0)  
 data[success\_label] = success\_column  
   
 # Удаление ненужных колонок  
 data = data.drop(columns=[output\_label, id\_label])  
   
 return data, success\_label

Значение threshold варьирует от 1 до 6 включительно для сравнения результатов.

Случайно выбираются признаков для модели. Ниже представлены получившиеся признаки:

Selected columns: 5 features

['Taking notes in classes' "Father's occupation" 'Course ID'

'Transportation to the university' 'Student Age']

Для каждого порового значения была проведена тренировка и оценены результаты согласно коду, приведённому ниже:

for t in thresholds:  
 print(f"\nОбработка порога: {t}")  
 # Бинаризация меток с текущим порогом  
 binarized\_data, \_ = preprocess\_data(data, success\_threshold=t)  
 binarized\_data = binarized\_data[selected\_features.tolist() + [success\_label]]  
   
 # Разделение данных  
 X = binarized\_data.drop(columns=[success\_label])  
 y = binarized\_data[success\_label]  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split\_custom(X, y, test\_size=0.2, random\_seed=RANDOM\_SEED)  
   
 print(f"Размер обучающей выборки: {X\_train.shape[0]}")  
 print(f"Размер тестовой выборки: {X\_test.shape[0]}")  
   
 # Обучение дерева  
 tree = DecisionTree()  
 tree.build\_tree(X\_train, y\_train)  
   
 # Предсказание  
 y\_pred = tree.predict(X\_test)  
 y\_proba = tree.predict\_proba(X\_test)  
   
 # Вычисление метрик  
 confusion\_matrix, accuracy, precision, recall = tree.compute\_metrics(y\_test.values, y\_pred)  
   
 # Вычисление AUC-ROC и AUC-PR  
 auc\_roc\_metrics, auc\_pr\_metrics = tree.calculate\_auc\_metrics(y\_test.values, y\_proba)  
 fpr, tpr, auc\_roc = auc\_roc\_metrics  
 recall\_vals, precision\_vals, auc\_pr = auc\_pr\_metrics

Получившиеся результаты:

Обработка порога: 1

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[ 0 2]

[ 1 26]]

Accuracy: 0.8966

Precision: 0.9286

Recall: 0.9630

AUC-ROC: 0.8519

AUC-PR: 0.8407

Обработка порога: 2

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[ 4 6]

[ 1 18]]

Accuracy: 0.7586

Precision: 0.7500

Recall: 0.9474

AUC-ROC: 0.8132

AUC-PR: 0.8787

Обработка порога: 3

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[12 2]

[ 4 11]]

Accuracy: 0.7931

Precision: 0.8462

Recall: 0.7333

AUC-ROC: 0.8595

AUC-PR: 0.8756

Обработка порога: 4

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[17 2]

[ 1 9]]

Accuracy: 0.8966

Precision: 0.8182

Recall: 0.9000

AUC-ROC: 0.9632

AUC-PR: 0.8390

Обработка порога: 5

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[20 1]

[ 0 8]]

Accuracy: 0.9655

Precision: 0.8889

Recall: 1.0000

AUC-ROC: 0.9792

AUC-PR: 0.8165

Обработка порога: 6

Размер обучающей выборки: 116

Размер тестовой выборки: 29

Дерево решений построено.

Confusion Matrix:

[[21 1]

[ 1 6]]

Accuracy: 0.9310

Precision: 0.8571

Recall: 0.8571

AUC-ROC: 0.9026

AUC-PR: 0.7873

Были построены ROC и Precision-Recall кривые для каждого порога бинаризации, что позволило визуально оценить качество моделей.

A graph of different thresholds

Description automatically generated

Рисунок 1 – Кривая AUC-ROC

A graph of different types of curves

Description automatically generated

Рисунок 2 - Кривая AUC-PR

Метрики для различных порогов:

Threshold Accuracy Precision Recall AUC-ROC AUC-PR

0 1 0.896552 0.928571 0.962963 0.851852 0.840741

1 2 0.758621 0.750000 0.947368 0.813158 0.878650

2 3 0.793103 0.846154 0.733333 0.859524 0.875553

3 4 0.896552 0.818182 0.900000 0.963158 0.839017

4 5 0.965517 0.888889 1.000000 0.979167 0.816468

5 6 0.931034 0.857143 0.857143 0.902597 0.787309

Точность (Accuracy). Наиболее высокая точность (0.9655) наблюдается при пороге 5, что указывает на высокую степень правильных предсказаний. Снижение порога приводит к уменьшению точности, как видно на пороге 2 (0.7586).

Точность предсказаний (Precision). Порог 1 обеспечивает наибольшую точность (0.9286), но на пороге 5 и 6 значение Precision остается на достаточно высоком уровне (0.8889 и 0.8571 соответственно).

Полнота (Recall). Наибольшая полнота (1.000) достигается на пороге 5, что означает, что модель не пропускает положительные примеры. В то же время, на низких порогах (2 и 1) полнота также высока, но это сопровождается падением других метрик, таких как точность.

AUC-ROC и AUC-PR. Максимальные значения AUC-ROC (0.9792) и AUC-PR (0.8787) также соответствуют порогу 5, что подтверждает, что этот порог является оптимальным по большинству ключевых метрик.

# Примеры использования метода

Данный метод наилучшим образом подходит для задач классификации. Например, в диагностике заболеваний, где важно минимизировать пропущенные случаи болезни, стоит выбрать порог с высоким показателем Recall (например, порог 5 с Recall = 1.000). Высокая полнота гарантирует, что все положительные случаи будут определены, даже если часть предсказаний окажется ложной.

Также этот метод отлично подходит для задач классификаций новых объектов прежде не проходивших классификации.