Лабораторная работа 5. Деревья решений

1. Цели

Приобрести навыки построения деревьев решений

2. Задачи

1. Построить модель дерева решений
2. Интерпретировать работу построенной модели

3. Теоретические сведения

Методические указания для решения поставленного задания

3.1. Деревья решений

Классификация – задача, при которой множество объектов нужно различать и группировать в классы по имеющимся признакам. Задача классификации сводится к задаче распознавания объектов, принадлежащих к множеству классов. Решение подобной задачи производится с помощью решающего правила, то есть для определения принадлежности объектов к классу, нужно, чтобы эти объекты выполняли некоторые условия. Решающее дерево может использоваться как для классификации, так и для регрессии. Такая модель пригодна как для предсказания стоимости домов, так и для определения типов звёздных объектов на основе их характеристик. В данной работе модель будет определять спектральный класс звёзд

Наиболее эффективное решающее дерево – бинарное, так как его обход осуществляется за сублинейное время, а выбор происходит по одной переменной за раз. В наилучшем варианте каждый узел сокращает множество возможных ответов вдвое. Дерево принятие решений можно прочесть как булеву функцию

На рёбрах дерева решений записываются признаки, от которых зависит целевая функция, на листьях значение целевой переменной, а в остальных узлах – промежуточные значения, представленные в виде объектов

Идея построения дерева решений будет рассматриваться на примере алгоритма Квинлана. Для реализации алгоритма необходимо множество объектов, каждый из которых характеризуется набором атрибутов, возможно задающих его принадлежность к классу

Алгоритм представляет собой итеративный дихотомайзер и создает дерево, имеющее множество путей, находя жадным образом для каждого узла признак, который даёт наибольший прирост информации для заданного множества элементов. Понятие прироста информации выражается через понятие энтропии информации

Важными ограничениями, накладываемыми на алгоритм, являются:

1. Алгоритм использует жадный подход, выбирая лучший признак, чтобы разделить набор данных на каждой итерации. Нововведение, которое может быть сделано в алгоритме, должно использовать возврат во время поиска оптимального дерева решений
2. Результатом работы алгоритма ID3 не всегда является оптимальное дерево – всё зависит от его конкретной реализации
3. ID3 более трудно использовать на непрерывных данных. Если значения признаков непрерывны, то в связи с этим существует множество способов, с помощью которых можно разделить данные по этим значениям

Предположим, что множество A элементов, некоторые из которых обладают свойством S, классифицировано посредством множества атрибута Q, имеющего q возможных значений. Тогда прирост информации ($gain$) определяется как

$$gain(A, Q) = entropy(A, S) - \sum_{i=1}^{q} \frac{|A_i|}{|A|} entropy(A_i, S)$$

Где энтропия множества объектов A по отношению к множеству свойств S мощностью s определяется как

$$entropy(A, S) = -\sum_{i=1}^{S} \frac{m_i}{n} \log_2 \frac{m_i}{n}$$

Где A_i – множество элементов A, на которые имеется значение атрибута q_i из множества Q

На каждой итерации алгоритм должен выбирать тот атрибут, для которого прирост информации максимален

Алгоритм ID3 основан на следующей рекурсивной процедуре
1. Выбирается признак для корневого узла дерева и формируются ветви для каждого из возможных значений того признака
2. Если все примеры на некотором листе принадлежат одному классу, то этот лист помечается именем этого класса

Алгоритм ID3(A, S, Q):
1. Создать корень дерева
2. Если s выполнима на всех элементах – поставить в корень 1 и выйти
3. Если s не выполнима ни на одном элементе – поставить в корень 0 и выйти
4. Если $Q = \emptyset$
   4.1. Если s выполнима на половине или большей части A поставить в корень метку 1 и выйти
   4.2. Если s не выполнима на большей части A поставить в корень метку 0 и выйти
5. Выбрать $q_i \in Q$, для которого прирост информации максимален
6. Поставить в корень метку q_i
7. Для каждого значения атрибута $q_i \in Q$
   7.1. Добавить нового потомка корня и поместить соответствующее исходящее ребро меткой q_i
   7.2. Если $g(A, q_i) = \emptyset$, то пометить этого потомка в зависимости от того, на какой части A выполняется s
   7.3. Иначе запустить ID3$\left(A_q, S, Q \setminus \{q_i\}\right)$ и добавить его результат как поддерево с корнем в этом потомке

Алгоритм может создавать узлы и листья, содержащие малое количество примеров. Чтобы избежать такой ситуации используют эмпирическое правило – при разбиении множества, по крайней мере два подмножества должны иметь не меньше заданного минимального количества объектов. Если данное правило не выполняется, то дальнейшее разбиение этого множества прекращается и соответствующий узел отмечается как лист. В листьях могут присутствовать объекты разных классов. Тогда в качестве решения листа выбирается класс, наиболее часто встречающийся в узле. Если в листе присутствует равное число примеров

нескольких классов, то решение дает класс, наиболее часто встречающийся у непосредственного предка данного листа.

3.2. Построение модели с помощью `scikit-learn`

Пример построения модели будет приведён ниже, демонстрация будет производится на наборе Star dataset to predict star types (рус. Набор звездных данных для прогнозирования типов звезд)

3.2.1. Подключение библиотек

Подключение библиотек

```
import sklearn
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import tree
from sklearn.model_selection import train_test_split
```

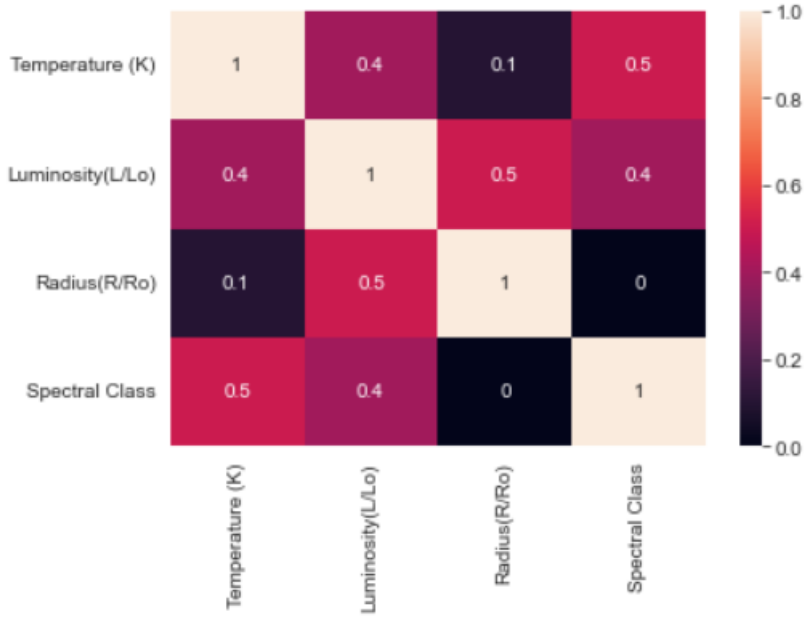
3.2.2. Работа с набором данных

Из набора будут удалены столбцы `Absolute magnitude(Mv)`, `Star type` и `Star color`, так как они мультиколлинеарны. Будут факторизованы столбец `Spectral Class`.

```
PATH = "stars.csv"
COLUMNS_FOR_FACTORISATION = ["Spectral Class"]

dataset = pd.read_csv(PATH)
factorization_table = {}

dataset.drop(
    [
        "Absolute magnitude(Mv)",
        "Star type",
        "Star color",
    ],
    axis=1,
    inplace=True,
)

for column in dataset.columns:
    if column in COLUMNS_FOR_FACTORISATION:
        dataset[column], table = pd.factorize(dataset[column])
        factorization_table[column] = pd.DataFrame(
            columns=[column],
            data=table
        )

dataset
```

Также будет проведена проверка на значимость признаков и их корреляцию

```
sns.heatmap(
    round(
        abs(dataset.corr()),
        1,
    ),
    annot=True
)
```



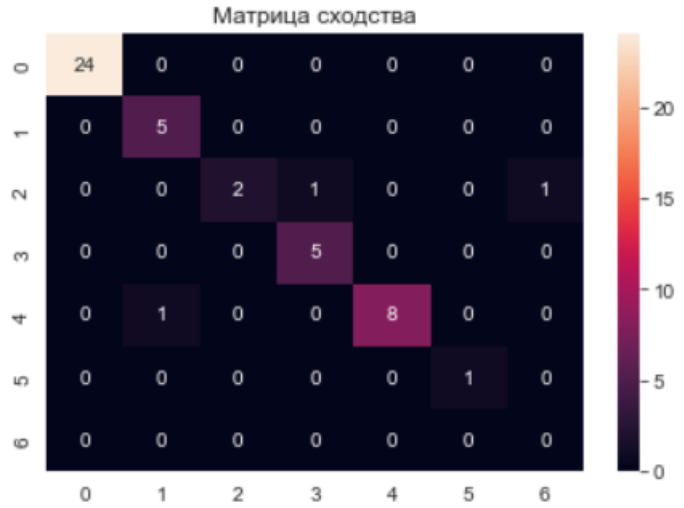
3.2.3. Построение модели

Перед построением модели данные будут поделены на тренировочные и тестовые

```
train_input, test_input, train_output, test_output = train_test_split(
    dataset.drop("Spectral Class", axis=1),
    dataset["Spectral Class"],
    test_size=0.2
)
```

Код ниже строит модель дерева решения, а также выводит на экран матрицу сходства предсказанных значений тестового набора и эталонных значений

```
model = tree.DecisionTreeClassifier()
model.fit(train_input, train_output)

predictions = model.predict(test_input)
confusion_matrix = sklearn.metrics.confusion_matrix(predictions, test_output)

sns.heatmap(
    confusion_matrix,
    annot=True
)
plt.title('Матрица сходства')
```



Для получения самого дерева решения используется следующий код

```
DPI = 450
plt.figure(dpi=DPI)
tree.plot_tree(model)
plt.title("Дерево решений")
```

4. Задание

Выбрать с сайта kaggle.com набор данных в формате .csv, пригодный для построения дерева решений, загрузить и подготовить его к дальнейшей обработке. Наборы данных не должны повторяться внутри группы. Задание индивидуальное. Требования:

1. Построить модель дерева решений
2. Оценить точность модели построив матрицу сходства
3. Получить интерпретацию построенной модели
4. Указать какие знания можно получить из набора
5. Сохранить IPython Notebook

4.1. Продвинутое задание

Построить вторую модель, без использования средств scikit-learn