



Машинное обучение

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. Д.Ю. КУПРИЯНОВ

ЛЕКЦИЯ 5

Библиотеки

В данной лекции будут рассмотрены примеры с использованием следующих библиотек:

- NumPy – <https://numpy.org/>
- Pandas – <https://pandas.pydata.org/>
- scikit-learn – <https://scikit-learn.org>
- Matplotlib – <https://matplotlib.org/>
- Graphviz – <https://graphviz.gitlab.io/>

Классификация и кластеризация

Две наиболее востребованных задачи машинного обучения – это классификация и кластеризация данных. В каком-то смысле эти задачи схожи так как позволяют поделить множество наблюдений на набор групп, объединённых по одинаковым признакам. Тем не менее, эти задачи очень отличаются.

Прежде всего, задача классификации чаще всего подразумевает заранее известное конечное множество групп, каждая из которых соответствует одной метке класса (class label). Сами группы в задачах классификации называют классами. В задачах кластеризации число групп может быть неизвестно заранее и даже может не быть конечным. Группы в задачах кластеризации принято называть кластерами. Сам факт выявления числа кластеров – это уже отдельная востребованная задача.

Во вторых, классификация данных – это задача, ориентированная на обучение с учителем, а кластеризация данных – это задача, решаемая чаще всего без стадии обучения.

Примеры задач

Наиболее классическим примером классификации на сегодняшний день можно считать задачу классификации всех входящих писем по принципу спам/не спам.

Другим примером классификации может быть задача определения пола клиента по его действиям на сайте с целью предложения контекстной рекламы.

Примером задачи классификации с числом классов, отличным от двух, может выступать распознавание символов номеров машин по фотографии. Фактически у нас есть для каждого символа 22 класса, соответствующих либо букве, либо цифре из перечня: У, К, Е, Н, Х, В, А, Р, О, С, М, Т, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Примером задач кластеризации может быть задача выявления по результатам маркетинговых исследований характерных группы потребителей, дифференцированных по сфере их интересов.

Другим примером кластеризации может служить задачи разделения множества новостей на группы по содержанию.

Часть 1

МЕТОДЫ КЛАССИФИКАЦИИ ДАННЫХ

Постановка задачи

Дано:

Множество уже классифицированных пар U . Причём каждое $u \in U$ – это пара вида: (\vec{v}, k) , где $\vec{v} \in V$ – это вектор признаков, а $k \in K$ – это метка, соответствующая конкретному классу. При этом множество K – конечно. Грубо говоря, \vec{v} – это результаты наблюдений, а k – это решения учителя, о том какому классу данные наблюдения относятся.

Нужно найти:

Для нового набора наблюдений $\vec{v}_n \in V$ (число наблюдений может не быть конечным) найти алгоритм (программу), сопоставляющий каждому новому наблюдению метку класса k из множества K с минимальным числом ошибок.

Наиболее известные алгоритмы

- ☐ Деревья принятия решений
- ☐ Наивный Байесовский классификатор
- ☐ Метод опорных векторов (SVM)
- ☐ Нейронные сети
- ☐ Другие алгоритмы

Деревья принятия решений

В общем случае, дерево принятия решений – это некоторая иерархическая конструкция, которая пошагово даёт инструкцию для решения какой-либо задачи. В том числе, в качестве задачи может быть вопрос разделения данных на группы.

Для простоты далее мы будем иногда называть деревья принятия решений просто деревьями решений.

Пример

Рассмотрим пример классификации решения о выдаче кредита на основе данных о возрасте, наличии образования, наличия квартиры и доходе клиента.

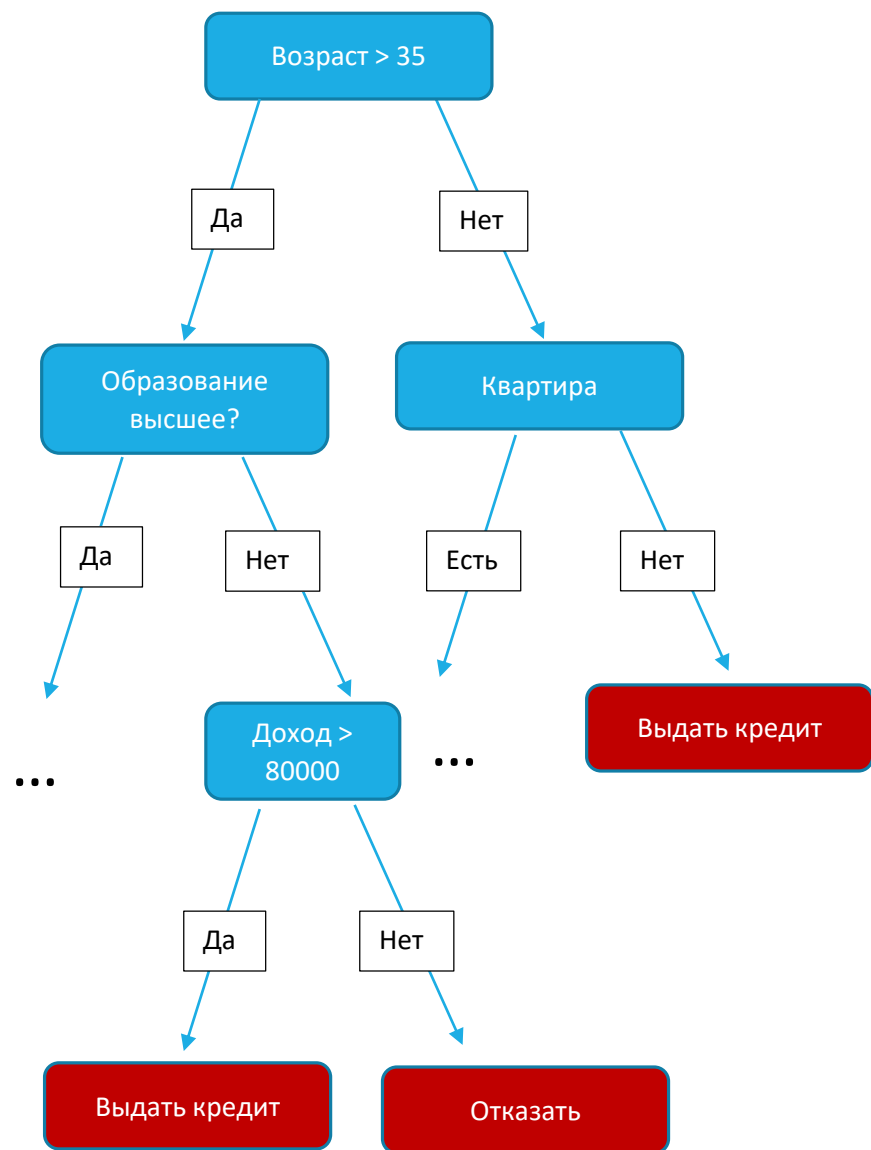
Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
53	да		130000	выдать
44	нет		60000	отказать
28	да		90000	выдать
51	да	да	80000	выдать
30	да		80000	отказать
34	нет		140000	выдать
28	да	да	50000	отказать

Построение дерева решений

Дерево решений можно построить как по мнению экспертов, так и на основе реальных данных.

Во втором случае дерево решений строится на основе множества уже классифицированных пар U , с учетом того, что мы уже знаем правильный класс для каждого наблюдения.

В каждой ветке дерева могут быть элементы разных классов.



На рисунке показан пример части дерева принятий решения, построенный на основе экспертного мнения.

Приведенное дерево принятия решений, позволяет получить ответ, следует ли выдать клиенту кредит или отказать клиенту в кредите.

Как видно из данного примера, основная идея деревьев решений – это разделение на каждом шаге множества наблюдений на два или более простых подмножества.

Построение дерева решений

Основная проблема при построении дерева заключается в том, как правильно выбрать разделяющий параметр и его значение, чтобы эффективнее всего получить в итоге поддеревья с одним классом? Это не всегда очевидно!

Существует две наиболее используемые (известные) оценки, предназначенные для принятия решения о разделении множества в узлах дерева: энтропия Шенона и неопределённость Джини.

Оценивая, как изменяются значения функции оценки для разделения по различным параметрам, можно выбрать наиболее эффективное построение (ветвление) дерева.

Энтропия Шенона

Энтропия Шенона [2] рассчитывается следующим образом:

$$S = - \sum_{i=1}^K p_i \log_2 p_i,$$

где $p_i = \frac{N_i}{N}$, N – общее число элементов поддерева, N_i число элементов поддерева относящихся к классу i (если пронумеровать все значения множества меток класса K).

В библиотеке `sklearn` для языка Python данный вариант носит название `entropy` [1].

Неопределённость Джини

Неопределённость Джини [1] рассчитывается следующим образом:

$$G = 1 - \sum_{i=1}^K p_i^2 \quad ,$$

где $p_i = \frac{N_i}{N}$, N – общее число элементов поддерева, N_i число элементов поддерева относящихся к классу i (если пронумеровать все значения множества меток класса K).

В библиотеке `sklearn` для языка Python данный вариант носит название `gini` [1].

Критерий отбора (Прирост информации)

Получив значение функции оценки состояния до разбиения и для каждого из полученных в результате разбиения поддеревьев мы можем дать оценку уменьшению энтропии при помощи следующего критерия прироста информации (Information Gain):

$$IG = H_{l-1} - \sum_{j=1}^R \frac{N_{l,j}}{N_{l-1}} H_{l,j},$$

где R – число поддеревьев, возникающих в результате разбиения, j – номер поддерева, возникшего в результате разбиения, H_{l-1} и N_{l-1} – значение функции оценки энтропии и число элементов до разбиения, $H_{l,j}$ и $N_{l,j}$ – значение функции оценки энтропии j -го поддерева и число элементов в нём.

Пример

В качестве примера рассмотрим минимум данных для принятия решения о выдаче клиенту кредитов.

Вычислим характеристики до разделения:

$$N = 8$$

$$i = 1, 2$$

$$H = -(5/8 \times \log_2(5/8) + 3/8 \times \log_2(3/8)) = 0,954$$

Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
53	да		130000	выдать
44	нет		60000	отказать
28	да		90000	выдать
51	да	да	80000	выдать
30	да		80000	отказать
34	нет		140000	выдать
28	да	да	50000	отказать

Варианты разделения

В нашем примере есть возможность разделения по 4 параметрам. Рассмотрим следующие варианты:

- ☐ Образование = “да” – Образование = “нет”;
- ☐ Квартира = “да” – Квартира = “нет”;
- ☐ Возраст ≥ 40 – Возраст < 40 ;
- ☐ Доход ≥ 95000 – Доход < 95000 .

Какой из вариантов разделения лучше? Давайте оценим!

Разделение по образованию

Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
53	да		130000	выдать
28	да		90000	выдать
51	да	да	80000	выдать
30	да		80000	отказать
28	да	да	50000	отказать

$$N = 6$$

$$i = 1, 2$$

$$H = -(4/6 \times \log_2(4/6) + 2/6 \times \log_2(2/6)) = 0,918$$

$$IG = 0,954 - (6/8 \times 0,918 + 2/8 \times 1) = 0,016$$

Возраст	Образование	Квартира	Доход	Класс
44	нет		60000	отказать
34	нет		140000	выдать

$$N = 2$$

$$i = 1, 2$$

$$H = -(1/2 \times \log_2(1/2) + 1/2 \times \log_2(1/2)) = 1$$

Разделение по квартире

Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
51	да	да	80000	выдать
28	да	да	50000	отказать

$$N = 3$$

$$i = 1, 2$$

$$H = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) = 0,918$$

$$IG = 0,954 - (3/8 \times 0,918 + 5/8 \times 0,971) = 0,003$$

Возраст	Образование	Квартира	Доход	Класс
53	да		130000	выдать
44	нет		60000	отказать
28	да		90000	выдать
30	да		80000	отказать
34	нет		140000	выдать

$$N = 5$$

$$i = 1, 2$$

$$H = -(3/5 \times \log_2(3/5) + 2/5 \times \log_2(2/5)) = 0,971$$

Разделение по возрасту

Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
53	да		130000	выдать
44	нет		60000	отказать
51	да	да	80000	выдать

$$N = 4$$

$$i = 1, 2$$

$$H = -(3/4 \times \log_2(3/4) + 1/4 \times \log_2(1/4)) = 0,811$$

$$IG = 0,954 - (4/8 \times 0,811 + 4/8 \times 1) = 0,049$$

Возраст	Образование	Квартира	Доход	Класс
28	да		90000	выдать
30	да		80000	отказать
34	нет		140000	выдать
28	да	да	50000	отказать

$$N = 4$$

$$i = 1, 2$$

$$H = -(2/4 \times \log_2(2/4) + 2/4 \times \log_2(2/4)) = 1$$

Разделение по доходу

Возраст	Образование	Квартира	Доход	Класс
42	да	да	140000	выдать
53	да		130000	выдать
34	нет		140000	выдать

$$N = 3$$

$$i = 1$$

$$H = -(3/3 \times \log_2(3/3)) = 0$$

$$IG = 0,954 - (3/8 \times 0 + 5/8 \times 0,971) = 0,348$$

Возраст	Образование	Квартира	Доход	Класс
44	нет		60000	отказать
28	да		90000	выдать
51	да	да	80000	выдать
30	да		80000	отказать
28	да	да	50000	отказать

$$N = 5$$

$$i = 1, 2$$

$$H = -(2/5 \times \log_2(2/5) + 3/5 \times \log_2(3/5)) = 0,971$$

ИТОГОВЫЙ ВЫБОР

Значения параметра IG получились следующими:

- ☐ Образование = “да” – Образование = “нет”, $IG = 0,016$;
- ☐ Квартира = “да” – Квартира = “нет”, $IG = 0,003$;
- ☐ Возраст ≥ 40 – Возраст < 40 , $IG = 0,049$;
- ☐ Доход ≥ 95000 – Доход < 95000 , $IG = 0,348$.

Выбираем вариант 4 – разделение по доходу.

Выбор для неопределённости Джини

Значения параметра IG получились следующими:

- ☐ Образование = “да” – Образование = “нет”, $IG = 0,001$;
- ☐ Квартира = “да” – Квартира = “нет”, $IG = 0,002$;
- ☐ Возраст ≥ 40 – Возраст < 40 , $IG = 0,031$;
- ☐ Доход ≥ 95000 – Доход < 95000 , $IG = 0,169$.

По-прежнему выбираем вариант 4 – разделение по доходу.

Непрерывные переменные

В нашем примере мы не обсуждали вопрос того, как было выбрано значение Доход ≥ 95000 . На самом деле, в случае непрерывной шкалы измерения параметров дерево классификации превращается в дерево регрессии и выбор нужно делать немного другим способом.

Прежде всего, все значения выстраиваются по порядку. В качестве разделяющих рассматриваются средние арифметические значения между двумя соседними. В качестве эффективности оценки разбиения используется дисперсия:

$$IG = H_{l-1} - \sum_{j=1}^R \frac{1}{N_{l,j}} \sum_{i=1}^{N_{l,j}} (y_{j,i} - \bar{Y}_j)^2,$$

где R – число поддеревьев, возникающих в результате разбиения, j – номер поддерева, возникшего в результате разбиения, $N_{l,j}$ – число элементов в поддереве с номером j , $y_{j,i}$ – i -й элемент поддерева с номером j , \bar{Y}_j – среднее арифметическое элементов поддерева с номером j .

Что делать дальше?

На основе рассмотренной идеи разделения ветвей можно реализовать ряд алгоритмов построения деревьев принятия решений, в том числе:

- ❑ CART (Classification and Regression Tree),
- ❑ C4.5 (приемник ID3),
- ❑ C5.0 (приемник C4.5).

Похожие идеи будут и у алгоритмов:

- ❑ ID3 (Iterative Dichotomiser 3)
- ❑ CHAID (CHi-squared Automatic Interaction Detector),
- ❑ MARS (расширяет деревья решений для лучшей обработки числовых данных).

Переобучение

Используя критерий Джини или любой другой похожий критерий можно построить дерево, доведя количество наблюдений в листах до единицы или доведя до ситуации, когда в каждом листе элементы только одного класса. Будет ли такое дерево обеспечивать «хорошее» принятие решений? Обычно нет!

Подобное дерево решений будет слишком ориентированным на частные факторы, не имеющие общего значения в рамках исходной задачи. В том числе, будут учтены все выбросы, что тоже внесёт дополнительные ошибки. При этом подобное дерево решений будет иметь высокую алгоритмическую сложность из-за большой глубины. Такая ситуация называется переобучением!

Отсечение ветвей

Есть два варианта исключения данной проблемы: не достраивать дерево до максимальной глубины, введя критерий остановки, либо построить дерево, а затем убрать «лишние» ветки, выполнив, так называемое, отсечение ветвей (pruning).

Алгоритм ID3

Алгоритм ID3 предложен Россом Куинланом ещё в 1986 году. Он довольно прост и ограничен следующим образом:

1. Алгоритм применяется только к номинальным данным.
2. Узел всегда разделяется по всем имеющимся в нём значениям выбранного в качестве разделителя атрибута. Дерево не бинарно и может иметь много ветвей. Не может быть правил разделения вида $x > c$. Каждое поддереву соответствует одному возможному значению разделяющего атрибута.
3. В качестве критерия разбиения используется Энтропия Шенона и прирост информации.
4. Алгоритм останавливается, если в узле все объекты одного класса или, если энтропия Шеннона становится достаточно мала.
5. Может применяться обрезание ветвей.

Алгоритм CART

1. Алгоритм CART строит бинарное дерево принятия решений. Таким образом, у каждой вершины дерева только два потомка. На каждом шаге нужно выбрать только одно разделяющее значение.
2. Для выбора параметра, по которому выполняется разделение, используется критерий отбора на основе неопределённости Джини (или дисперсии для непрерывных шкал измерения).
3. Правила разделения имеют вид $x > c$ или $x \in C$.
4. Строится полное дерево (если не задан критерий остановки), а затем выполняется отсечение ветвей по алгоритму *minimal cost-complexity tree pruning*, который позволяет получить оценку в узлах дерева, эффективности соединения, порожденных ими поддеревьев [3].

Как выбрать оптимальное дерево?

При использовании алгоритмов отсечения ветвей становится актуальной задача, как при этом выбрать оптимальное дерево.

Обычно используется один из подходов оценки качества дерева, основанных на разных способах выделения обучающей и валидационной выборках:

1. Подход, предлагающий оставить из исходной обучающей выборки некоторую часть (не используя её при построении дерева) и в дальнейшем оценить различные варианты деревьев с её помощью на основе $(1 - SE)$ -правила.
2. Подход, предлагающий использовать механизм перекрёстной проверки (cross validation).

(1-SE)-правило

Пусть $R(t)_{min}$ – это минимальная ошибка классификации среди всех рассматриваемых деревьев.

При этом, стандартная ошибка (оценка реальной ошибки) определяется как:

$$SE(R(t)) = \sqrt{\frac{R(t)(1 - R(t))}{N_{tst}}},$$

где N_{tst} – число наблюдений тестовой выборки, с помощью которой оцениваются различные варианты деревьев.

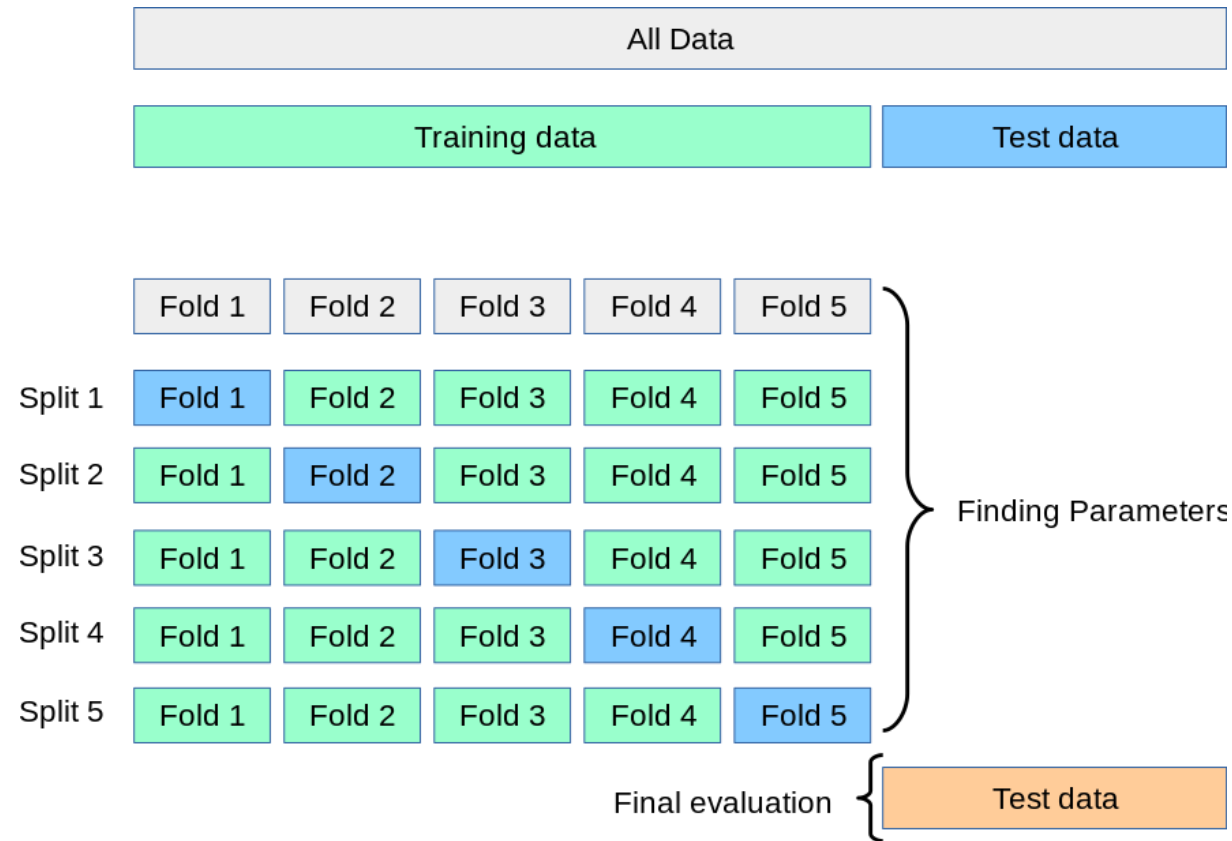
Тогда согласно (1-SE)-правилу лучшим будем считать дерево минимально размера, для которого выполняется условие:

$$R(t) \in [R(t)_{min}, R(t)_{min} + SE(R(t))].$$

Перекры́стная проверка

Перекры́стная проверка (cross-validation) используется, если обучающий набор слишком мал, или входящие в него наблюдения слишком уникальны, чтобы их можно было разделить на основную и тестовую части.

Перекрёстная проверка



Вычисляется средняя оценка по всем разбиениям. Например, среднее значение SE.

Часть 2

ПРОГРАММНОЕ РЕШЕНИЕ

Ирисы Фишера

Решим классическую задачу классификации. Классификацию ирисов Фишера [7]. Ирисы Фишера – это набор данных, собранных американским ботаником Эдгаром Андерсоном. Каждая запись данного набора состоит из длины наружной доли околоцветника или чашелистника (англ. sepal length), ширины наружной доли околоцветника или чашелистника (англ. sepal width), длины внутренней доли околоцветника или лепестка (англ. petal length), ширина внутренней доли околоцветника или лепестка (англ. petal width) и указания вида ириса (класса). Всего рассмотрено три вида ирисов: Ирис щетинистый (setosa), Ирис разноцветный (versicolor), и Ирис виргинский (virginica). Первый из них линейно отделим от других.

На данной задаче часто проверяют качество методов кластеризации, сравнивая полученные результаты с реальным делением на классы (по видам ирисов).

Ирисы Фишера на Википедии

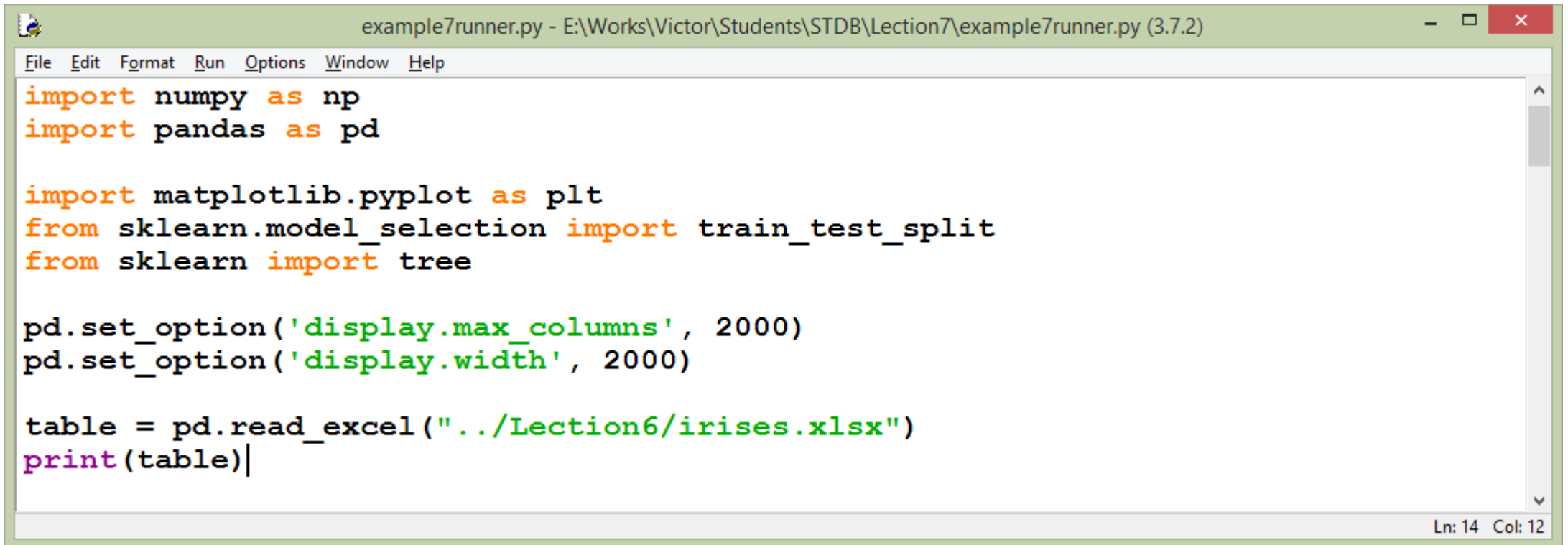
Ирисы Фишера

Длина чашелистика ↕	Ширина чашелистика ↕	Длина лепестка ↕	Ширина лепестка ↕	Вид ириса ↕
5.1	3.5	1.4	0.2	<i>setosa</i>
4.9	3.0	1.4	0.2	<i>setosa</i>
4.7	3.2	1.3	0.2	<i>setosa</i>
4.6	3.1	1.5	0.2	<i>setosa</i>
5.0	3.6	1.4	0.2	<i>setosa</i>
5.4	3.9	1.7	0.4	<i>setosa</i>
4.6	3.4	1.4	0.3	<i>setosa</i>
5.0	3.4	1.5	0.2	<i>setosa</i>
4.4	2.9	1.4	0.2	<i>setosa</i>
4.9	3.1	1.5	0.1	<i>setosa</i>
5.4	3.7	1.5	0.2	<i>setosa</i>
4.8	3.4	1.6	0.2	<i>setosa</i>
4.8	3.0	1.4	0.1	<i>setosa</i>



Пример построения дерева решения для задачи с Ирисами

Импортируем данные



```
example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import tree

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)

table = pd.read_excel("../Lecture6/irises.xlsx")
print(table)|
```

Ln: 14 Col: 12

Данные

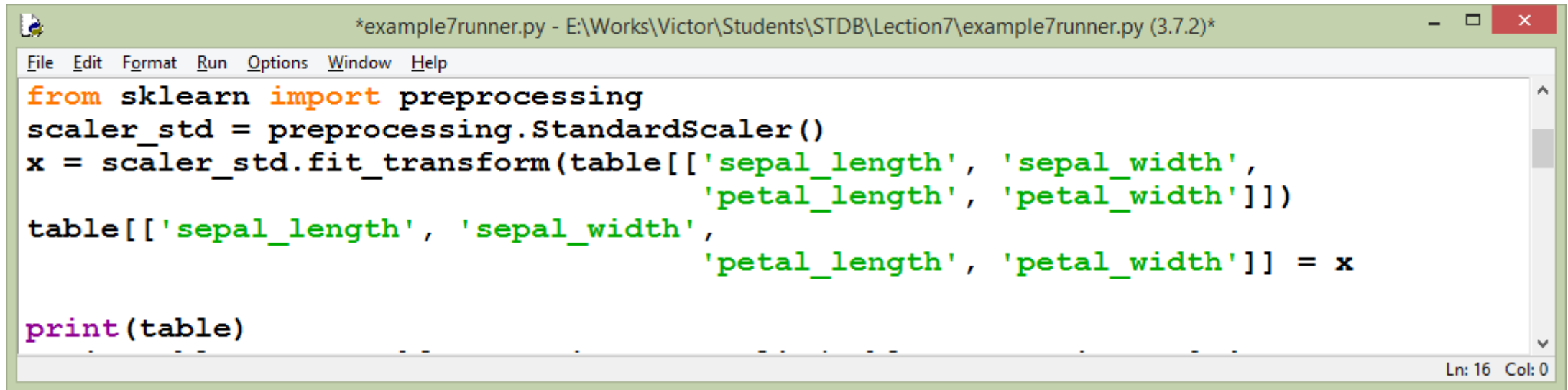
	sepal_length	sepal_width	petal_length	petal_width	class_label
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
--	--	--	--	--	

Стандартизация

В общем случае, для построения дерева-решений выполнять стандартизацию не нужно. Но мы рассмотрим несколько видов деревьев решений: на основе исходных данных и на основе главных компонент.

Метод главных компонент требует стандартизации. Поэтому, для однозначности исследуемых данных выполним стандартизацию до начала построения какого-либо дерева решений.

Стандартизация

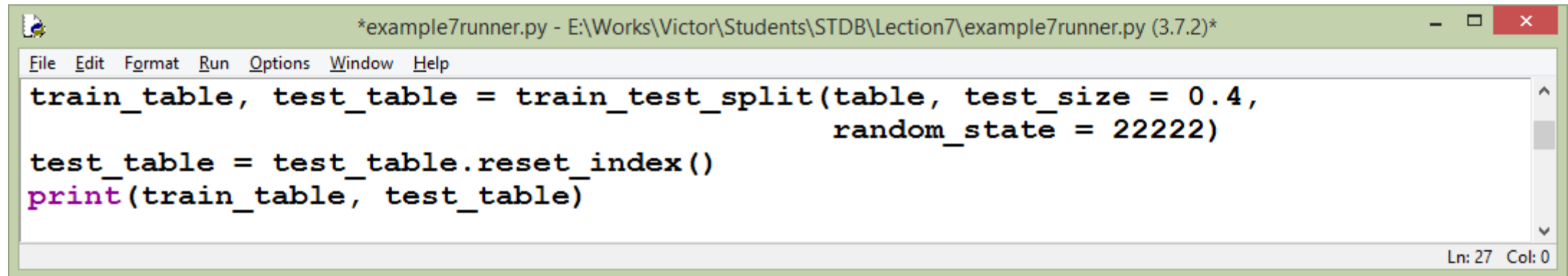


```
*example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)*
File Edit Format Run Options Window Help
from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width',
                                   'petal_length', 'petal_width']])
table[['sepal_length', 'sepal_width',
        'petal_length', 'petal_width']] = x

print(table)
```

Ln: 16 Col: 0

Выделим обучающую и тестовую выборки

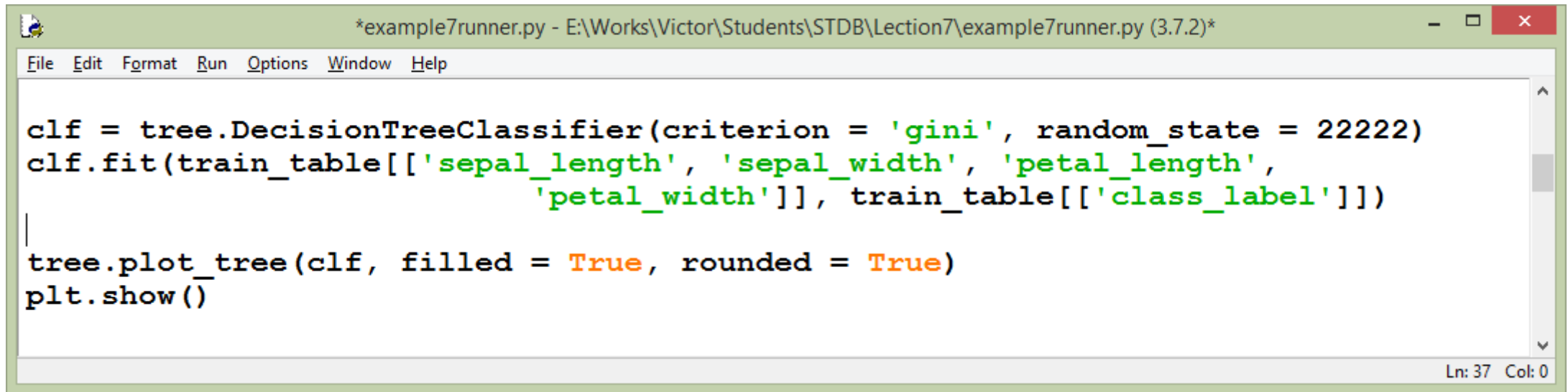


The screenshot shows a Python IDE window titled '*example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains the following Python code:

```
train_table, test_table = train_test_split(table, test_size = 0.4,  
                                           random_state = 22222)  
  
test_table = test_table.reset_index()  
print(train_table, test_table)
```

The status bar at the bottom right indicates 'Ln: 27 Col: 0'.

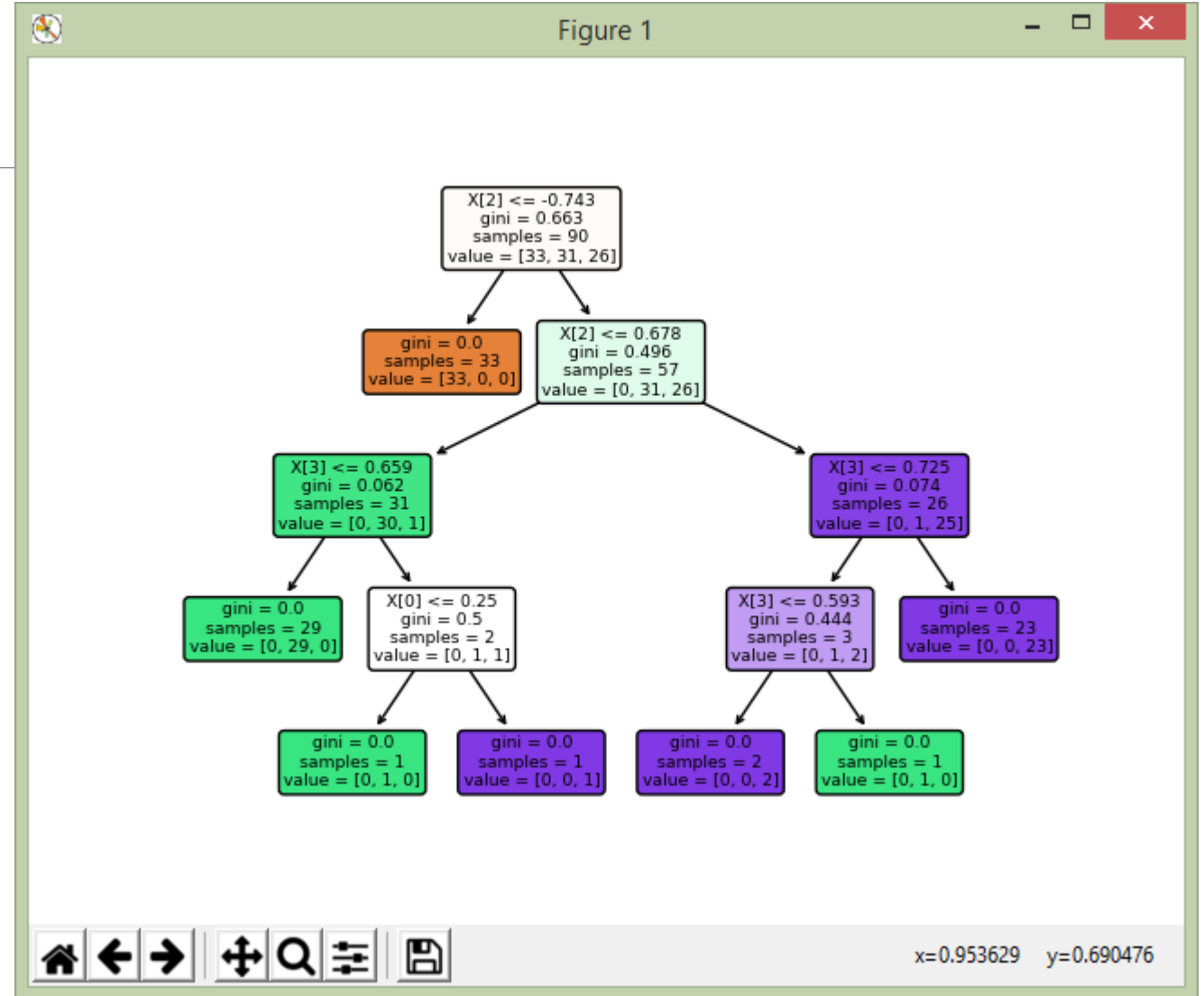
Построение дерева и визуализация



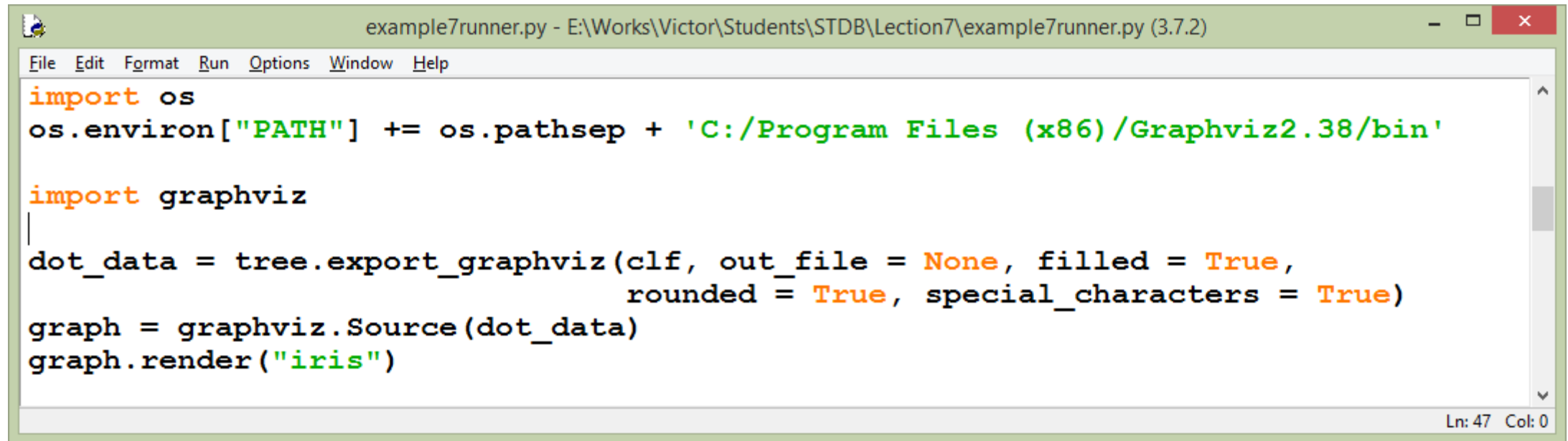
```
*example7runner.py - E:\Works\Victor\Students\STDB\Lecion7\example7runner.py (3.7.2)*  
File Edit Format Run Options Window Help  
  
clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)  
clf.fit(train_table[['sepal_length', 'sepal_width', 'petal_length',  
                    'petal_width']], train_table[['class_label']])  
  
tree.plot_tree(clf, filled = True, rounded = True)  
plt.show()  
  
Ln: 37 Col: 0
```

Результат

Видно довольно плохо. Поэтому используем другую библиотеку для визуализации [4].



Библиотека graphviz



The screenshot shows a text editor window with the title bar 'example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)'. The menu bar includes 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code in the editor is as follows:

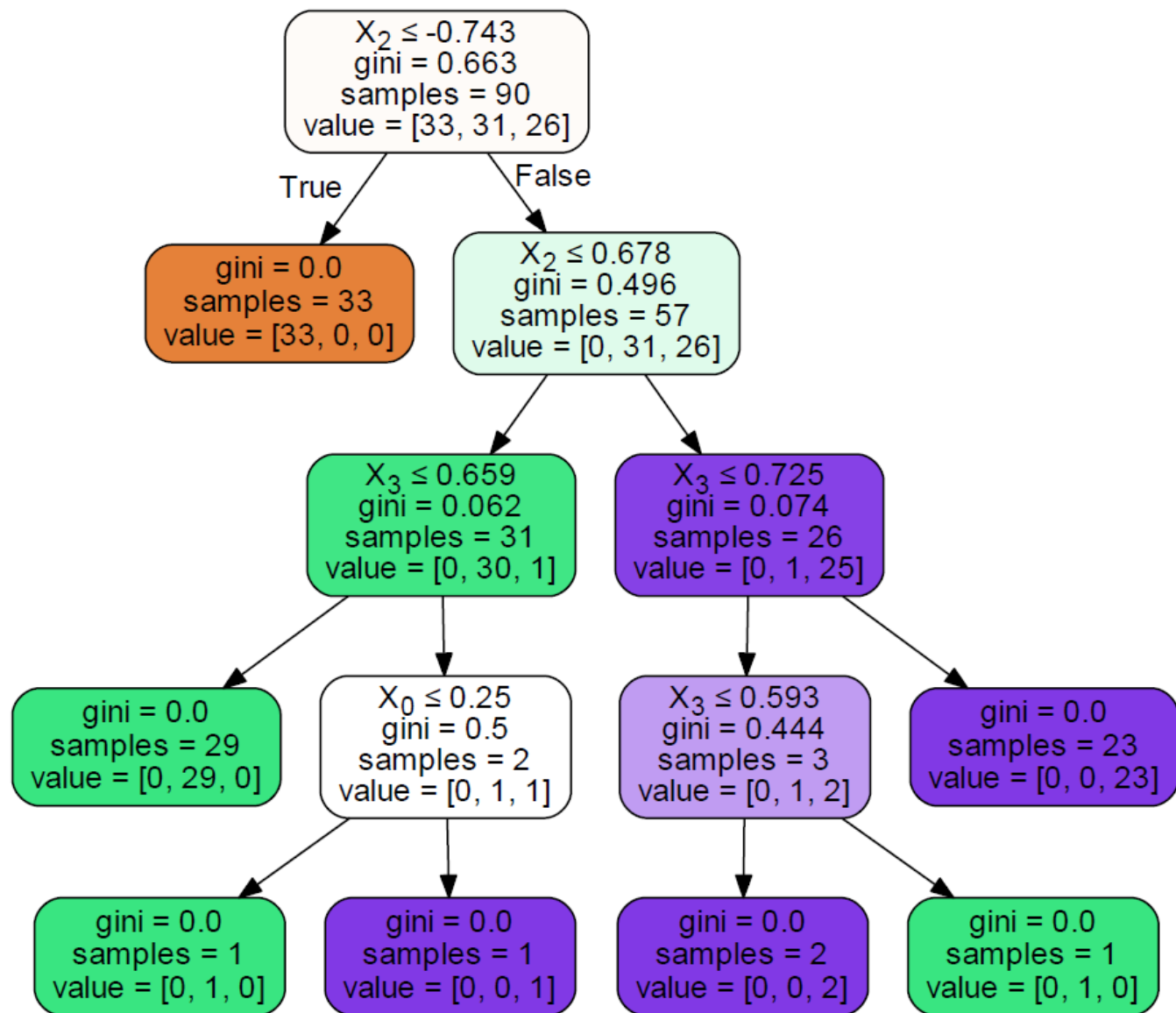
```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin'

import graphviz

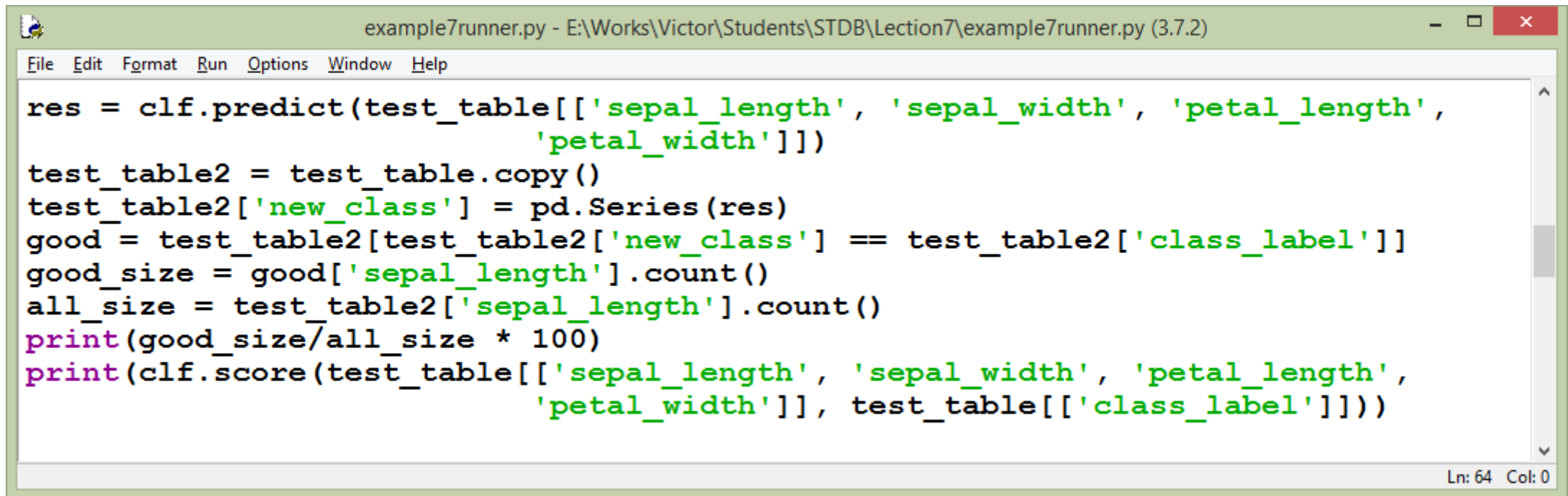
dot_data = tree.export_graphviz(clf, out_file = None, filled = True,
                                rounded = True, special_characters = True)
graph = graphviz.Source(dot_data)
graph.render("iris")
```

The status bar at the bottom right indicates 'Ln: 47 Col: 0'.

Результат



Классифицируем тестовую выборку



```
example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)
File Edit Format Run Options Window Help

res = clf.predict(test_table[['sepal_length', 'sepal_width', 'petal_length',
                              'petal_width']])
test_table2 = test_table.copy()
test_table2['new_class'] = pd.Series(res)
good = test_table2[test_table2['new_class'] == test_table2['class_label']]
good_size = good['sepal_length'].count()
all_size = test_table2['sepal_length'].count()
print(good_size/all_size * 100)
print(clf.score(test_table[['sepal_length', 'sepal_width', 'petal_length',
                              'petal_width']], test_table[['class_label']]))
```

Ln: 64 Col: 0

Результат

```
93.33333333333333  
0.9333333333333333  
>>> |
```

```
from sklearn.decomposition import PCA
pca = PCA()
table_pca = pca.fit_transform(table[['sepal_length', 'sepal_width',
                                     'petal_length', 'petal_width']])

explained_variance_ratio = pca.explained_variance_ratio_
print(explained_variance_ratio)
explained_variance = pca.explained_variance_
print(explained_variance)

pca = PCA(n_components = 2)
pca.fit_transform(table[['sepal_length', 'sepal_width',
                          'petal_length', 'petal_width']])

table3_p = pca.transform(table[['sepal_length', 'sepal_width',
                                'petal_length', 'petal_width']])
table3 = pd.DataFrame(table3_p, columns = ['pc1', 'pc2'])
table3['class_label'] = table['class_label']
print(table3)

pca = PCA(n_components = 1)
pca.fit_transform(table[['sepal_length', 'sepal_width',
                          'petal_length', 'petal_width']])

table4_p = pca.transform(table[['sepal_length', 'sepal_width',
                                'petal_length', 'petal_width']])
table4 = pd.DataFrame(table4_p, columns = ['pc1'])
table4['class_label'] = table['class_label']
```


Результат метода PCA

```
[0.72962445 0.22850762 0.03668922 0.00517871]  
[2.93808505 0.9201649 0.14774182 0.02085386]
```

95% - информативности – 2 компоненты.
Метод Казера – 1 компонента.

```
*example7runner.py - E:\Works\Victor\Students\STDB\Lecture7\example7runner.py (3.7.2)*
File Edit Format Run Options Window Help

train_table3, test_table3 = train_test_split(table3, test_size = 0.4,
                                             random_state = 22222)
train_table4, test_table4 = train_test_split(table4, test_size = 0.4,
                                             random_state = 22222)

clf3 = tree.DecisionTreeClassifier(criterion = 'entropy', random_state = 22222,
                                   ccp_alpha = 0.1)
clf3.fit(train_table3[['pc1', 'pc2']], train_table3[['class_label']])

print(clf3.score(test_table3[['pc1', 'pc2']], test_table3[['class_label']]))

clf4 = tree.DecisionTreeClassifier(criterion = 'entropy', random_state = 22222,
                                   ccp_alpha = 0.1)
clf4.fit(train_table4[['pc1']], train_table4[['class_label']])

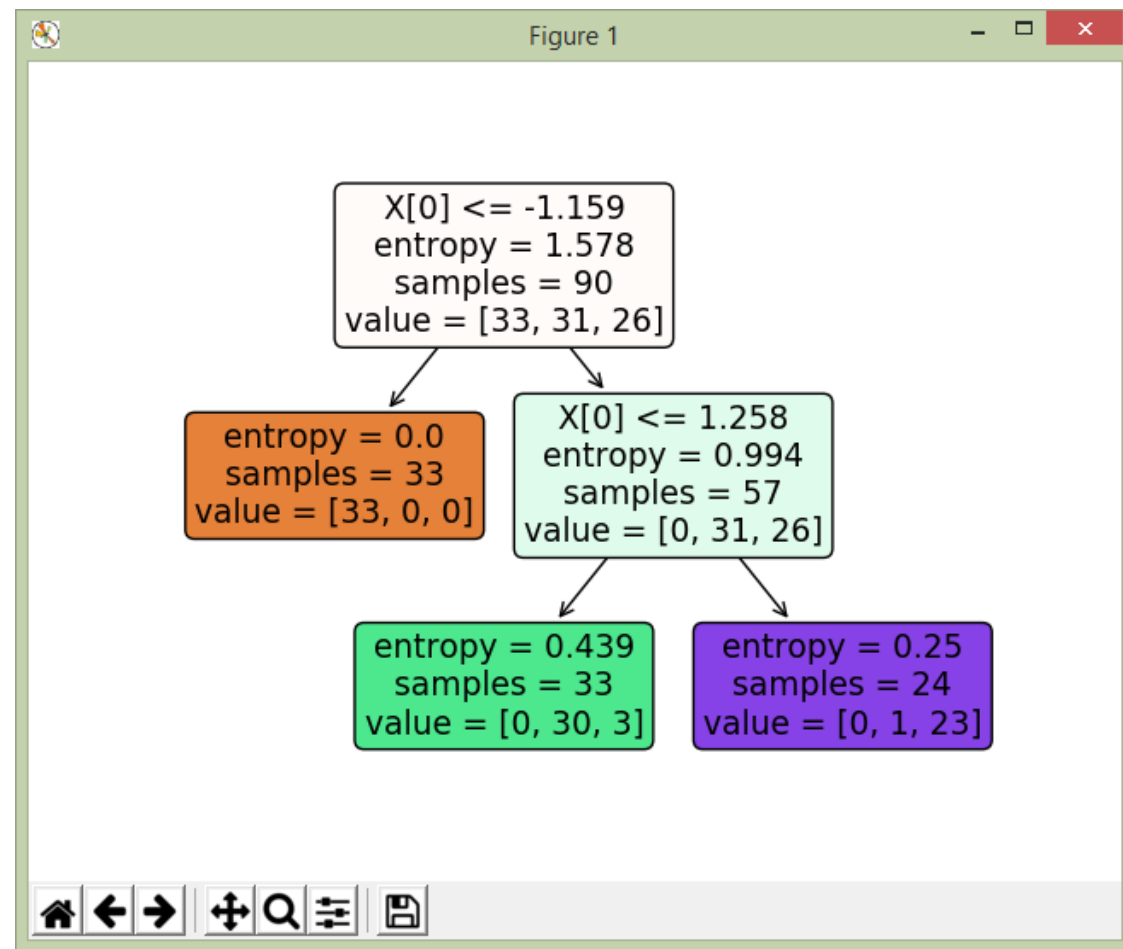
print(clf4.score(test_table4[['pc1']], test_table4[['class_label']]))

tree.plot_tree(clf3, filled = True, rounded = True)
plt.show()
tree.plot_tree(clf4, filled = True, rounded = True)
plt.show()
```

Ln: 113 Col: 49

Результат

```
0.8833333333333333  
0.8833333333333333  
>>> |
```



Интернет ресурсы и литература

1. <https://scikit-learn.org/stable/modules/tree.html>
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman *The Elements of Statistical Learning. Data Mining, Inference, and Prediction. Second Edition.* – Springer, 2009. – 739 p.
3. И.М. Андреев *Описание алгоритма CART* // Методы. Алгоритмы. Программы. – № 3-4 (7-8), 2004. – pp. 48-53.
4. https://graphviz.gitlab.io/pages/Download/Download_windows.html