



Машинное обучение

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. Д.Ю. КУПРИЯНОВ

ЛЕКЦИЯ 9 (СЕМЕСТР 2 – 2)

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/>

Часть 1

КЛАССИФИКАЦИЯ С ВЕСАМИ НАБЛЮДЕНИЙ

Невзвешенный набор данных

Во всех предыдущих примерах мы рассматривали наблюдения, как эквивалентные. Например, при принятии решения, к какому классу отнести лист дерева решений (классификации) мы пользовались следующим правилом:

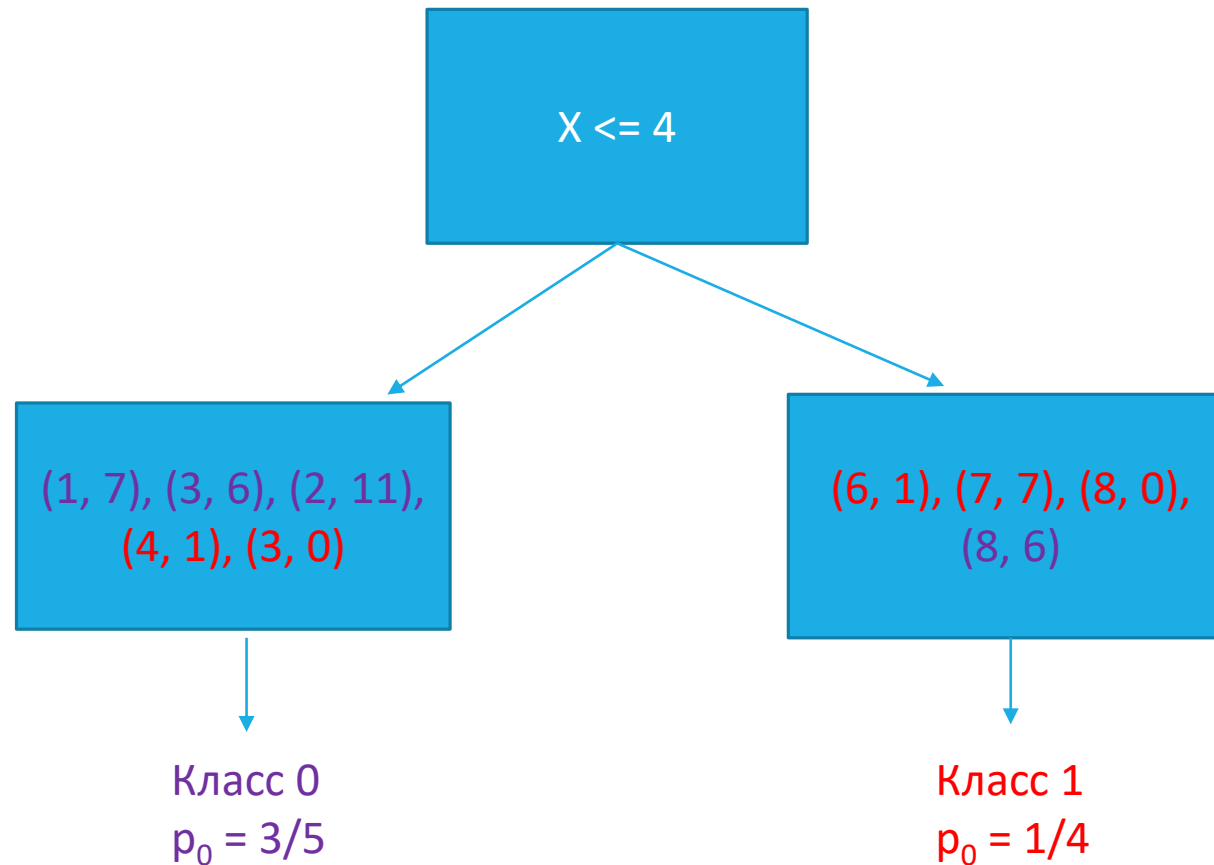
$$p_0 = N_0/N,$$

$$p_1 = N_1/N,$$

где p_0 – вероятность отнесения листа к классу, обратному целевой функции (при бинарной классификации), p_1 – вероятность отнесения листа к классу целевой функции, N_0 – число наблюдений, относящихся к классу, обратному целевой функции, в листе, N_1 – число наблюдений, относящихся к классу целевой функции, в листе, N – общее число наблюдений в листе.

Если порог $p = 0,5$, то если $p_0 > 0,5$, то все наблюдения, попавшие в лист, считаются относящимися к классу, обратному целевой функции, иначе все наблюдения, попавшие в лист, считаются относящимися к классу целевой функции.

Невзвешенный набор данных



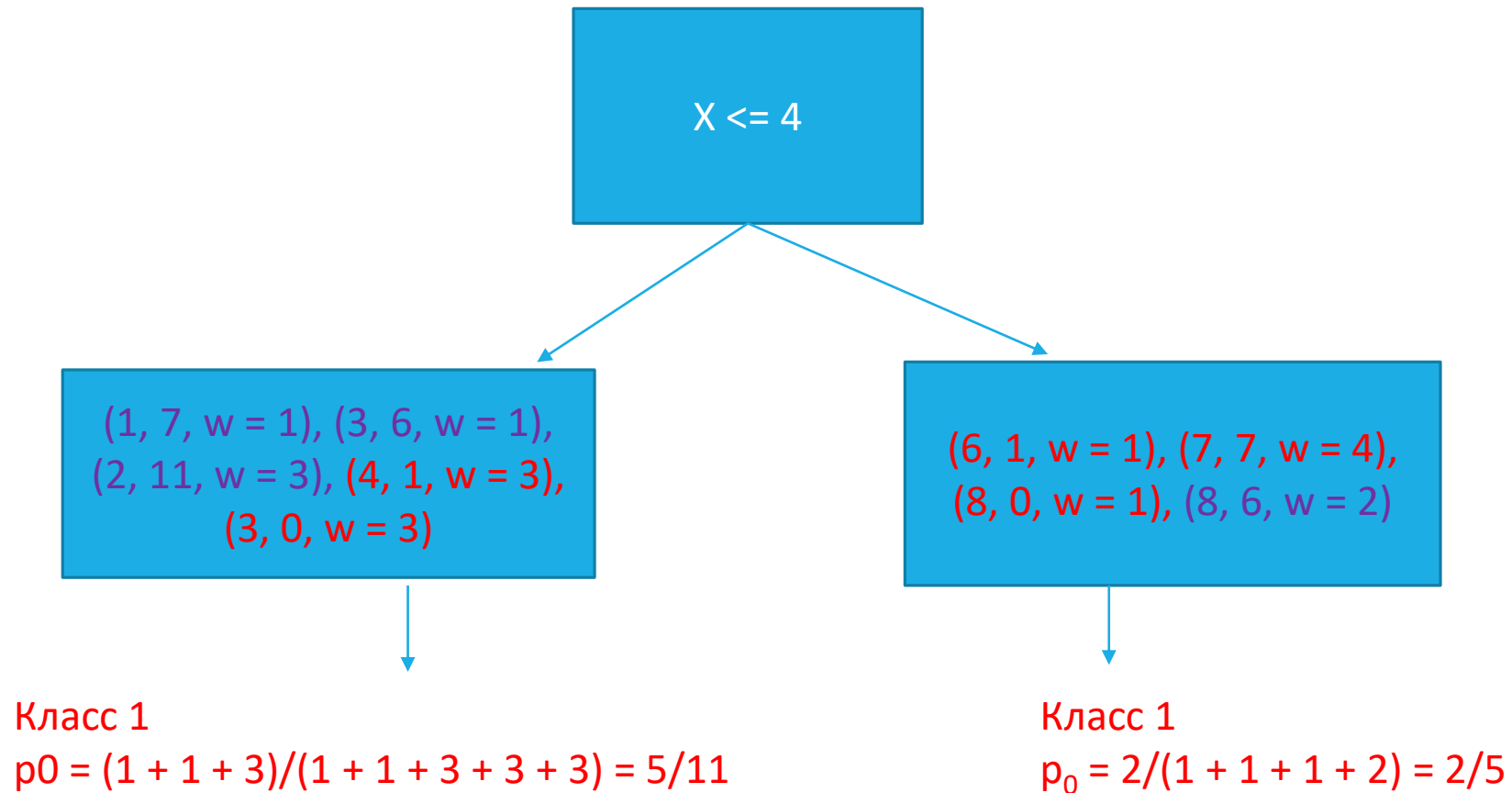
Взвешенный набор данных

Набор данных может иметь дополнительную разметку – вес наблюдений (например, вес может характеризовать степень доверия). Тогда расчет вероятности отнесения листа к определенному классу будет вычисляться как:

$$p_0 = \frac{\sum_{i=1}^N k_0(i) w_i}{\sum_{i=1}^N w_i},$$
$$p_1 = \frac{\sum_{i=1}^N k_1(i) w_i}{\sum_{i=1}^N w_i},$$

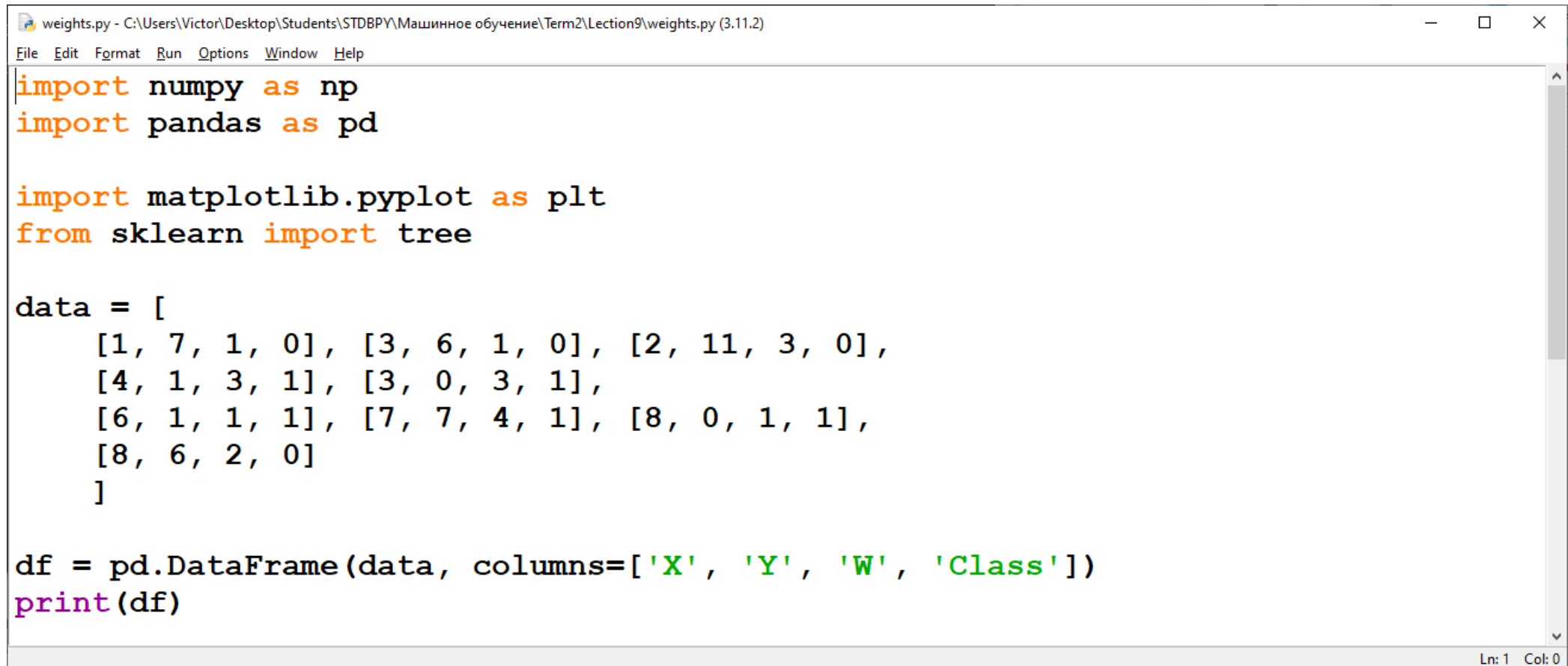
где w_i – веса наблюдений, $k_0(i)$ – функция, возвращающая 1, если i -е наблюдение не принадлежит целевой функции и 0 иначе, $k_1(i)$ – функция, возвращающая 0, если i -е наблюдение не принадлежит целевой функции и 1 иначе.

Взвешенный набор данных



Как это работает в реальности?

Загрузка данных



```
weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\weights.py (3.11.2)
File Edit Format Run Options Window Help

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn import tree

data = [
    [1, 7, 1, 0], [3, 6, 1, 0], [2, 11, 3, 0],
    [4, 1, 3, 1], [3, 0, 3, 1],
    [6, 1, 1, 1], [7, 7, 4, 1], [8, 0, 1, 1],
    [8, 6, 2, 0]
]

df = pd.DataFrame(data, columns=['X', 'Y', 'W', 'Class'])
print(df)
```

Ln: 1 Col: 0

Как это работает в реальности?

Загрузка данных

```
import numpy as np
import pandas as pd

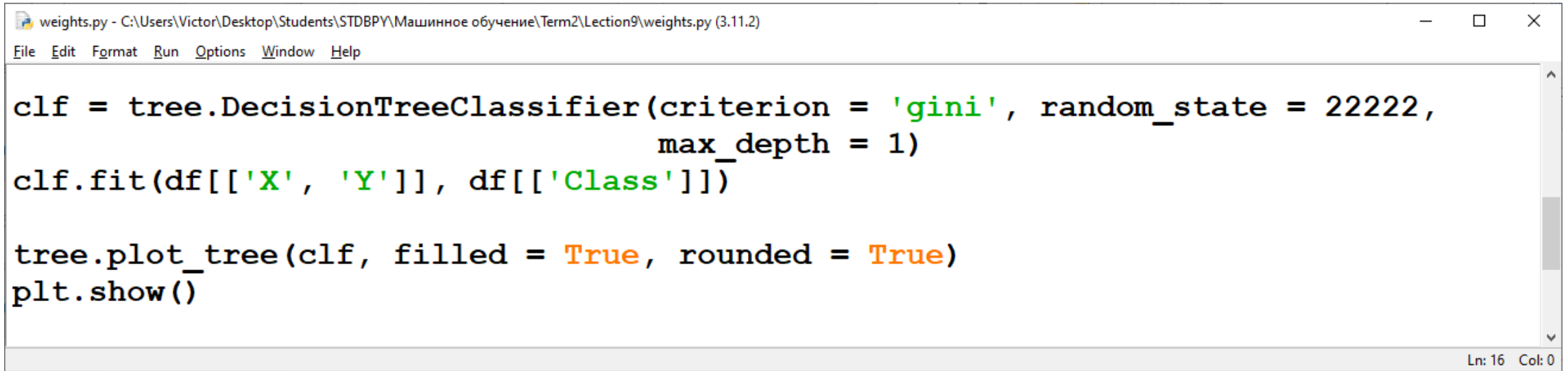
import matplotlib.pyplot as plt
from sklearn import tree

data = [
    [1, 7, 1, 0], [3, 6, 1, 0], [2, 11, 3, 0],
    [4, 1, 3, 1], [3, 0, 3, 1],
    [6, 1, 1, 1], [7, 7, 4, 1], [8, 0, 1, 1],
    [8, 6, 2, 0]
]

df = pd.DataFrame(data, columns=['X', 'Y', 'W', 'Class'])
print(df)
```

Как это работает в реальности?

Дерево без весов

A screenshot of a Python IDE window titled 'weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\weights.py (3.11.2)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor contains the following Python code:

```
clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222,
                                max_depth = 1)
clf.fit(df[['X', 'Y']], df[['Class']])

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

The status bar at the bottom right indicates 'Ln: 16 Col: 0'.

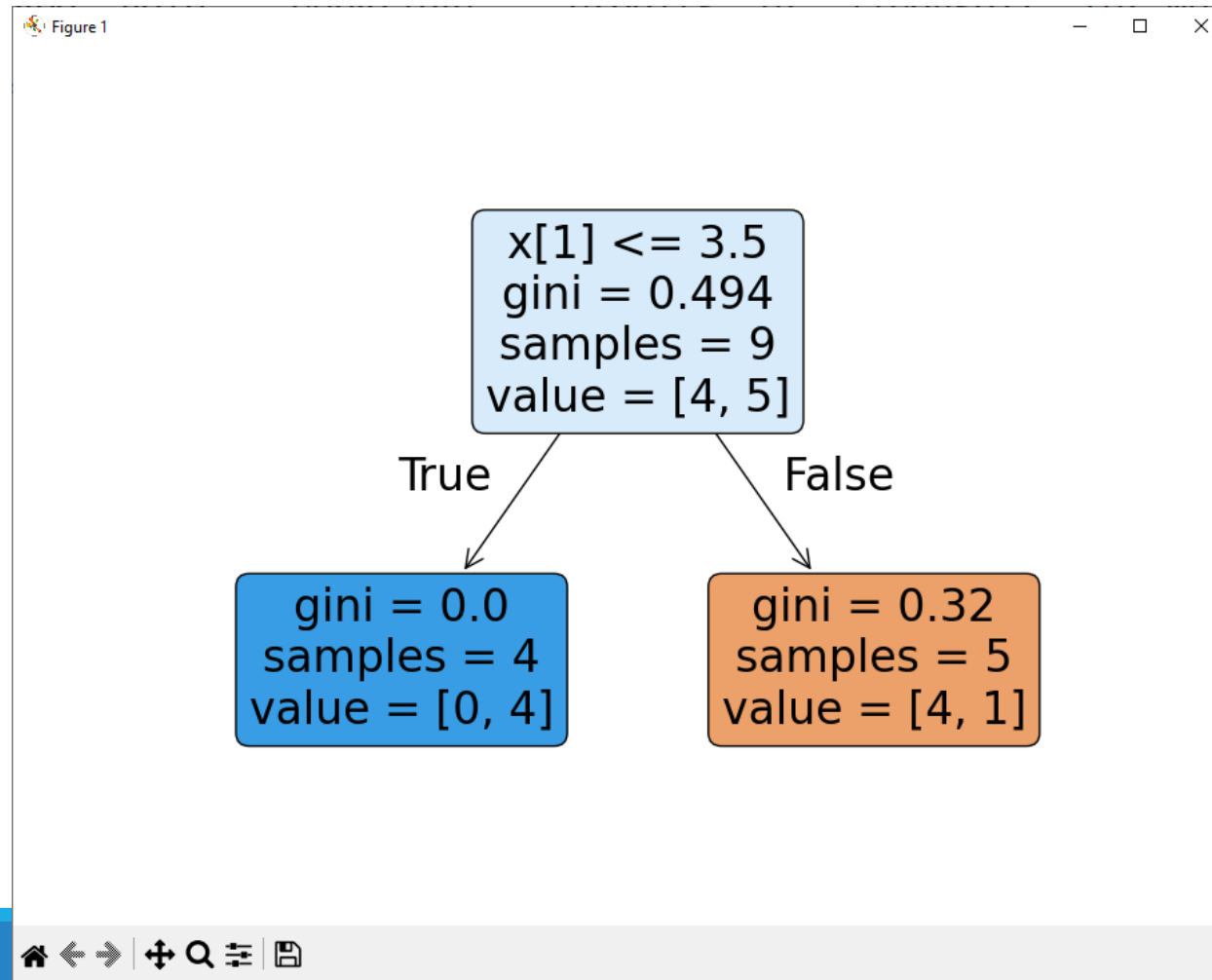
Как это работает в реальности?

Дерево без весов

```
clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222,  
                                max_depth = 1)  
clf.fit(df[['X', 'Y']], df[['Class']])  
  
tree.plot_tree(clf, filled = True, rounded = True)  
plt.show()
```

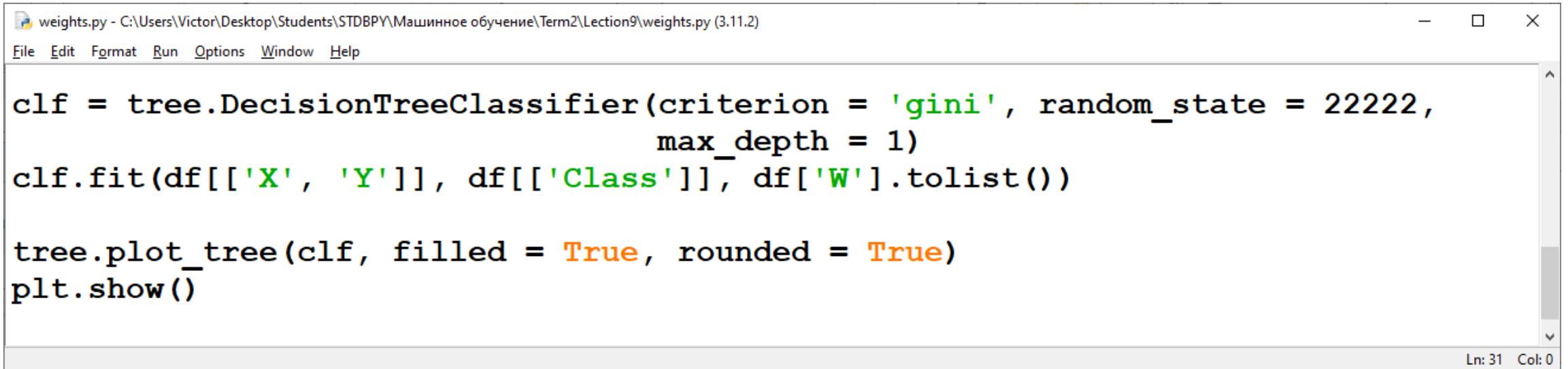
Как это работает в реальности?

Дерево без весов



Как это работает в реальности?

Дерево с весами



```
weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lection9\weights.py (3.11.2)
File Edit Format Run Options Window Help

clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222,
                                max_depth = 1)
clf.fit(df[['X', 'Y']], df[['Class']], df['W'].tolist())

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

Ln: 31 Col: 0

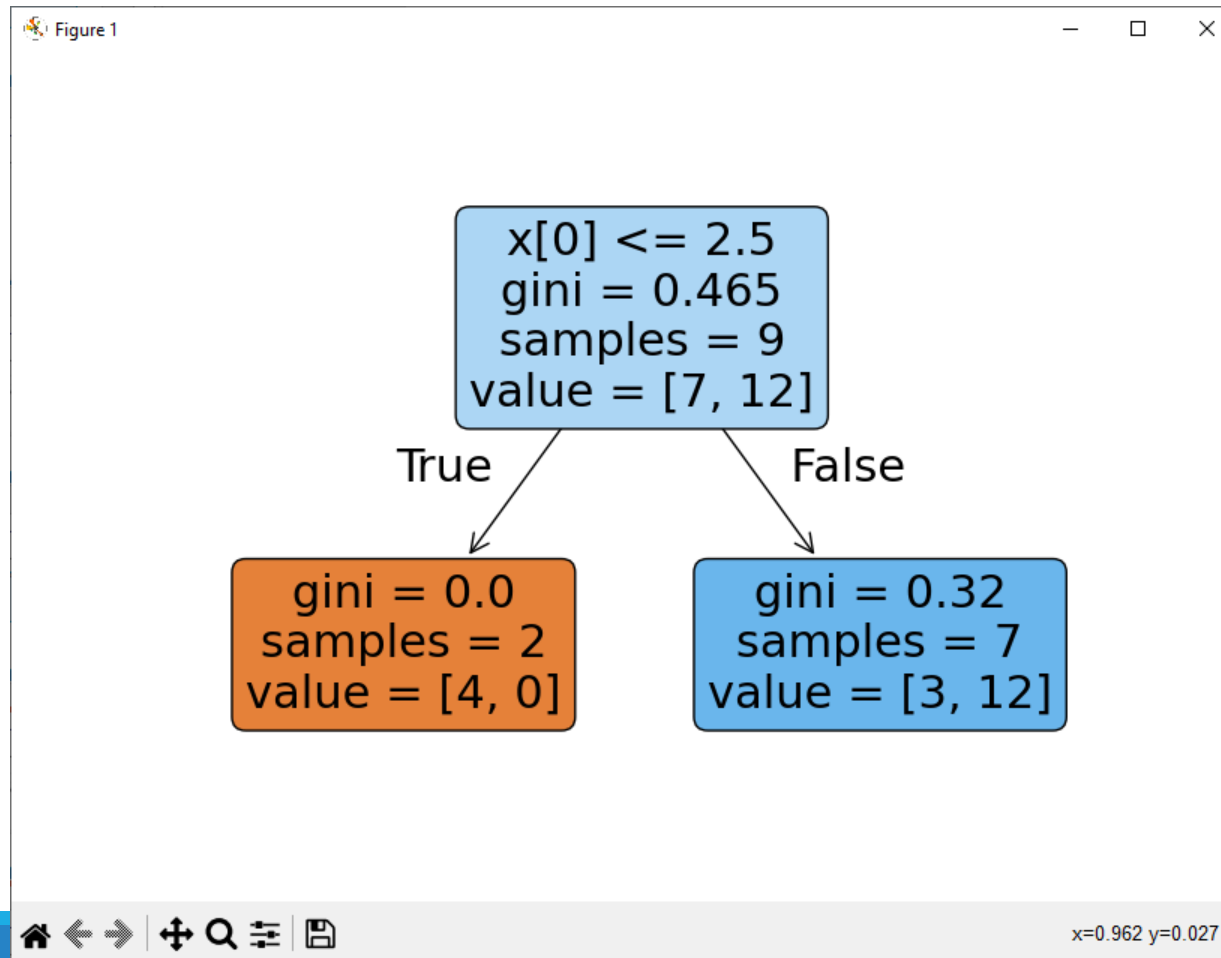
Как это работает в реальности?

Дерево с весами

```
clf = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222,  
                                max_depth = 1)  
clf.fit(df[['X', 'Y']], df[['Class']], df['W'].tolist())  
  
tree.plot_tree(clf, filled = True, rounded = True)  
plt.show()
```

Как это работает в реальности?

Дерево с весами



БУСТИНГ

Бустинг (boosting) – это подход в машинном обучении, когда функция прогноза (в нашем случае, классификатор) строится в виде ансамбля (композиции) базовых (простых) функций (например, деревьев решений):

$$f_0(x), f_1(x), f_2(x), \dots, f_k(x),$$

где x – наблюдение (вектор) для которого строится прогноз (в нашем случае, для которого вычисляется класс), $f_i(x)$ – функция прогноза простой модели (классификатора) для i -й итерации.

Агрегирующая модель представляет собой линейную комбинацию базовых моделей с настраиваемыми весами:

$$G_k(x) = f_0(x) + c_1 f_1(x) + c_2 f_2(x) + \dots + c_k f_k(x),$$

где c_k – вычисляемый коэффициент i -й итерации.

Все базовые модели, кроме начального приближения $f_0(x)$, берутся из одного семейства. Как правило, начальное приближение выбирается тождественным нолём или настраиваемой константой, а остальные модели – решающими деревьями небольшой глубины.

AdaBoost

AdaBoost – Адаптивный бустинг (сокращение от англ. adaptive boosting) — алгоритм машинного обучения, предложенный Йоавом Фройндом и Робертом Шапире. Исторически данный подход был первым алгоритмом бустинга.

Алгоритм AdaBoost

- ❑ Рассматривается задача бинарной классификации $y \in \{1, -1\}$, решаемая с помощью экспоненциальной функции потерь $e^{-G(x)y}$.
- ❑ Начальное приближение полагается равным нулю: $f_0(x) \equiv 0$.
- ❑ В качестве базовых алгоритмов $f_0(x), f_1(x), f_2(x), \dots, f_k(x)$ выбирают любые алгоритмы, способные обучаться на взвешенной обучающей выборке.
- ❑ Результатом работы AdaBoost является относительный рейтинг положительного класса $G_k(x)$.
- ❑ $G_k(x) = c_1 f_1(x) + c_2 f_2(x) + \dots + c_k f_k(x)$, а прогноз осуществляется извлечением знака от $G_k(x)$:
- ❑ $y(x) = \text{sign}(G_k(x)) = \text{sign}(c_1 f_1(x) + c_2 f_2(x) + \dots + c_k f_k(x))$.

Алгоритм AdaBoost

1. Инициализируем веса объектов $w_i^0 = 1 / N$, $i = 1, 2, \dots N$.
2. Цикл по всем $j = 1, 2, \dots k$:
 - i. на исходной выборке с весами w_i^0 строим классификатор $f_1(x)$;
 - ii. вычисляем взвешенную частоту ошибок: $E_j = \frac{\sum_{t=1}^N w_t^{j-1} [f_j(x_t) \neq y_t]}{\sum_{t=1}^N w_t^{j-1}}$;
 - iii. если $E_j = 0$ или $E_j \geq 0,5$, то конец цикла;
 - iv. вычисляем $c_j = \frac{1}{2} \ln((1 - E_j)/E_j)$;
 - v. вычисляем новые веса: $w_i^j = w_i^{j-1} e^{-y_i c_j f_j(x_i)}$;
 - vi. нормируем веса: $w_i^j = \frac{w_i^j}{\sum_{t=1}^N w_t^j}$;
 - vii. в начало цикла 2.i.
3. Получаем результирующий классификатор: $y(x) = \text{sign}(\sum_{j=1}^k c_j f_j(x))$.

Вспомним несбалансированный пример

Все предыдущие примеры работали с довольно сбалансированным набором данных. В каждом классе было приблизительно одинаковое число наблюдений. Но что будет, если мы возьмём несбалансированный пример? Например, такой как на сайте kaggle: <https://www.kaggle.com/mlg-ulb/creditcardfraud>.

Чтобы пример не был слишком вычислительно сложным, используем только первые 20 000 наблюдений.

Пример 5. Подготовка данных

```
*example2-1bin.py - E:\Works\Victor\Students\STDB\Term2\example2-1bin.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 svm

table = pd.read_csv("creditcard.csv").head(20000)

from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()

x = scaler_std.fit_transform(table[['V1', 'V2', 'V3', 'V4', 'V5', 'V6',
                                   'V7', 'V8', 'V9', 'V10', 'V11', 'V12',
                                   'V13', 'V14', 'V15', 'V16', 'V17', 'V18',
                                   'V19', 'V20', 'V21', 'V22', 'Amount']])
```

Ln: 9 Col: 0

Пример 5. Подготовка данных

```
import numpy as np
import pandas as pd

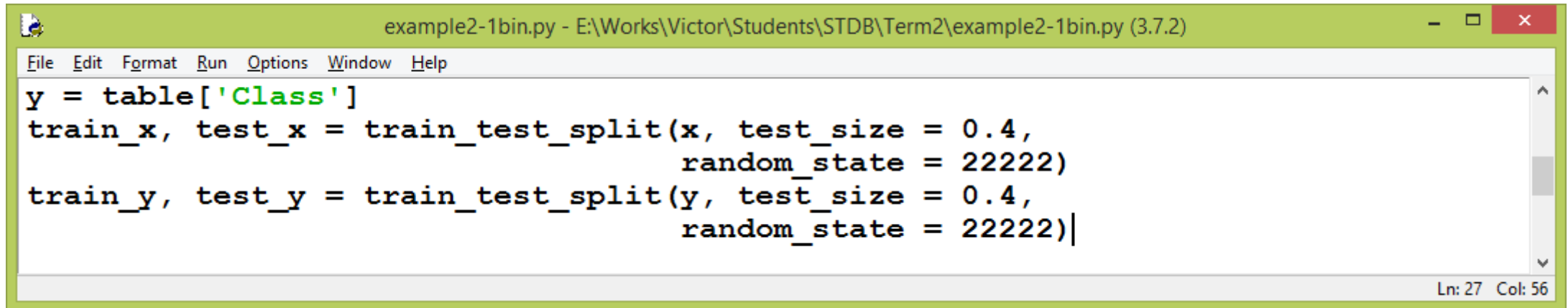
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import svm

table = pd.read_csv("creditcard.csv").head(20000)
```

```
from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()

x = scaler_std.fit_transform(table[[
    'V1', 'V2', 'V3', 'V4', 'V5', 'V6',
    'V7', 'V8', 'V9', 'V10', 'V11', 'V12',
    'V13', 'V14', 'V15', 'V16', 'V17', 'V18',
    'V19', 'V20', 'V21', 'V22', 'Amount']])
```

Пример 5. Подготовка данных

A screenshot of a Python IDE window titled "example2-1bin.py - E:\Works\Victor\Students\STDB\Term2\example2-1bin.py (3.7.2)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor contains the following Python code:

```
y = table['Class']
train_x, test_x = train_test_split(x, test_size = 0.4,
                                   random_state = 22222)
train_y, test_y = train_test_split(y, test_size = 0.4,
                                   random_state = 22222)|
```

The status bar at the bottom right indicates "Ln: 27 Col: 56".

```
example2-1bin.py - E:\Works\Victor\Students\STDB\Term2\example2-1bin.py (3.7.2)
File Edit Format Run Options Window Help
y = table['Class']
train_x, test_x = train_test_split(x, test_size = 0.4,
                                   random_state = 22222)
train_y, test_y = train_test_split(y, test_size = 0.4,
                                   random_state = 22222)|
Ln: 27 Col: 56
```


Решим обычным деревом

```
weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\weights.py (3.11.2)
File Edit Format Run Options Window Help

from sklearn import tree

clf1 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf1.fit(train_x, train_y)
res1 = clf1.predict(test_x)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(test_y, res1))

from sklearn.metrics import classification_report
print(classification_report(test_y, res1))

Ln: 65 Col: 12
```

Решим обычным деревом

```
from sklearn import tree
```

```
clf1 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
```

```
clf1.fit(train_x, train_y)
```

```
res1 = clf1.predict(test_x)
```

```
from sklearn.metrics import confusion_matrix
```

```
print(confusion_matrix(test_y, res1))
```

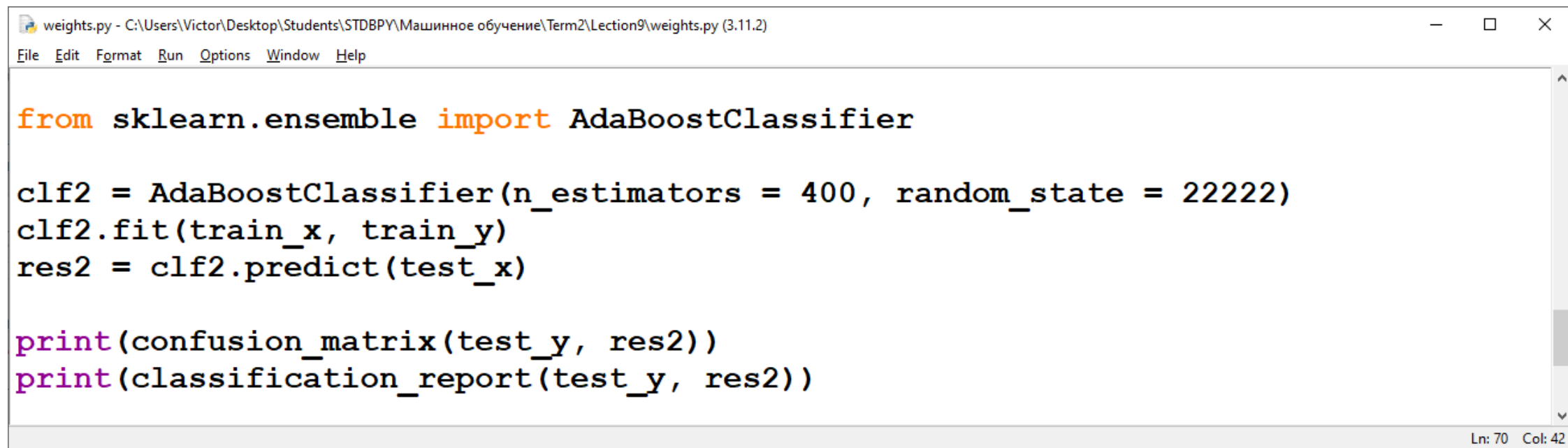
```
from sklearn.metrics import classification_report
```

```
print(classification_report(test_y, res1))
```

Решим обычным деревом

			precision	recall	f1-score	support
[[7953 12] [2 33]]	0		1.00	1.00	1.00	7965
	1		0.73	0.94	0.82	35
		accuracy			1.00	8000
		macro avg	0.87	0.97	0.91	8000
		weighted avg	1.00	1.00	1.00	8000

Решим с помощью AdaBoost

A screenshot of a Python IDE window titled 'weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\weights.py (3.11.2)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains the following Python code:

```
from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))
```

The status bar at the bottom right indicates 'Ln: 70 Col: 42'.

Решим с помощью AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))
```

Решим с помощью AdaBoost

		precision	recall	f1-score	support
[[7965 0] [3 32]]	0	1.00	1.00	1.00	7965
	1	1.00	0.91	0.96	35
accuracy				1.00	8000
macro avg		1.00	0.96	0.98	8000
weighted avg		1.00	1.00	1.00	8000

Решим с помощью RandomForest

```
weights.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\weights.py (3.11.2)
File Edit Format Run Options Window Help
from sklearn.ensemble import RandomForestClassifier

clf3 = RandomForestClassifier(n_estimators = 400, criterion = 'gini',
                             random_state = 22222)

clf3.fit(train_x, train_y)
res3 = clf3.predict(test_x)

print(confusion_matrix(test_y, res3))
print(classification_report(test_y, res3))
```

Ln: 67 Col: 37

Решим с помощью RandomForest

```
from sklearn.ensemble import RandomForestClassifier

clf3 = RandomForestClassifier(n_estimators = 400, criterion = 'gini',
                             random_state = 22222)
clf3.fit(train_x, train_y)
res3 = clf3.predict(test_x)

print(confusion_matrix(test_y, res3))
print(classification_report(test_y, res3))
```


Решим с помощью RandomForest

			precision	recall	f1-score	support
[[7965	0]	0	1.00	1.00	1.00	7965
[5	30]]	1	1.00	0.86	0.92	35
		accuracy			1.00	8000
		macro avg	1.00	0.93	0.96	8000
		weighted avg	1.00	1.00	1.00	8000

Сравним

[[7953 12]
[2 33]]

DT

[[7965 0]
[3 32]]

AB

[[7965 0]
[5 30]]

RF

А что, если классов много?

Мы рассмотрели AdaBoost с двумя классами. А что делать, если классов много. Будет ли работать?

Возьмем с сайта kaggle.com набор данных из лекции 8 для классификации стекол:
<https://www.kaggle.com/uciml/glass>.

Набор данных «Стекло»

```
many_classes.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\many_classes.py (3.11.2)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn import tree

table = pd.read_csv("glass.csv", sep = ',')

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
scaler_std = preprocessing.StandardScaler()

x = scaler_std.fit_transform(
    table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']])
table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']] = x
y = table['Type']
train_x, test_x = train_test_split(x, test_size = 0.4,
                                   random_state = 22222)
train_y, test_y = train_test_split(y, test_size = 0.4,
                                   random_state = 22222)
```

Ln: 48 Col: 0

Набор данных «Стекло»

```
import numpy as np
import pandas as pd

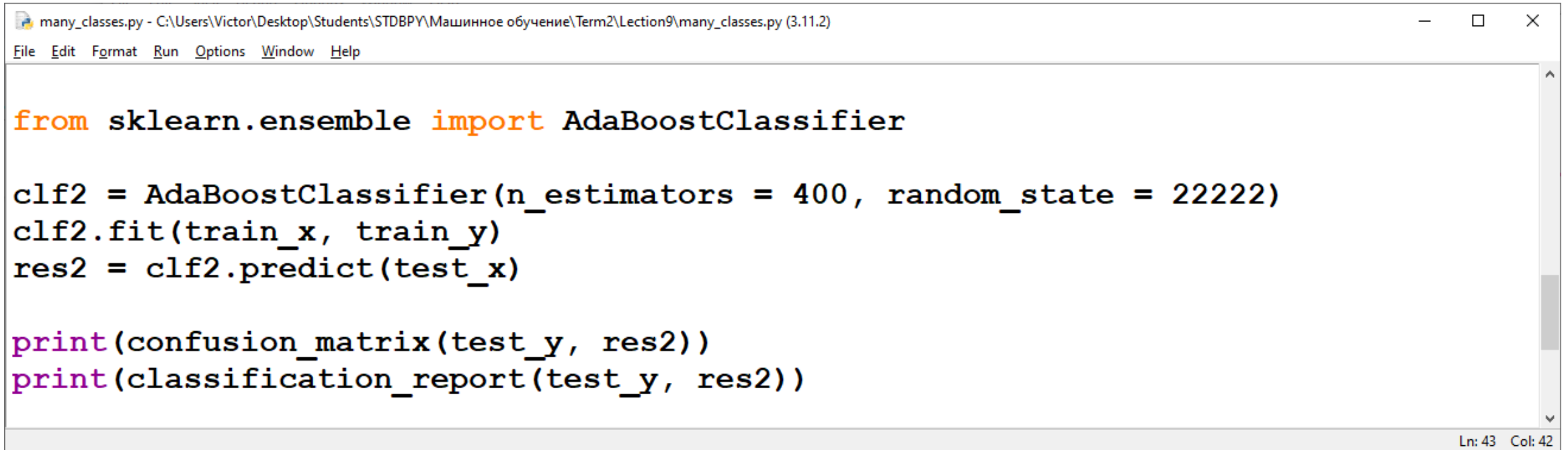
import matplotlib.pyplot as plt
from sklearn import tree

table = pd.read_csv("glass.csv", sep = ',')

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
scaler_std = preprocessing.StandardScaler()

x = scaler_std.fit_transform(table[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']])
table[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']] = x
y = table['Type']
train_x, test_x = train_test_split(x, test_size = 0.4, random_state = 22222)
train_y, test_y = train_test_split(y, test_size = 0.4, random_state = 22222)
```

Решим с помощью AdaBoost по умолчанию

A screenshot of a Python IDE window. The title bar shows the file path: many_classes.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\many_classes.py (3.11.2). The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))
```

The status bar at the bottom right indicates the cursor position: Ln: 43 Col: 42.

Решим с помощью AdaBoost по умолчанию

```
from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))
```

Результат не радует

	precision	recall	f1-score	support
1	0.44	0.55	0.49	22
2	0.56	0.63	0.59	35
3	0.00	0.00	0.00	4
5	0.33	0.14	0.20	7
6	1.00	1.00	1.00	3
7	0.93	0.87	0.90	15
accuracy			0.59	86
macro avg	0.55	0.53	0.53	86
weighted avg	0.57	0.59	0.57	86

Сравним с DT и RF

Model	Precision	Recall	F1-score	accuracy
AB	0,55	0,53	0,53	0,59
DT	0,53	0,56	0,53	0,71
RF	0,73	0,70	0,71	0,74

Почему так плохо? Дело в том, что мы использовали дерево по умолчанию. Оно слишком обучено для коррекции ошибки. Давайте возьмем дерево, ограниченное по размеру.

Решим с помощью AdaBoost с настроенным деревом

```
many_classes.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture9\many_classes.py (3.11.2)
File Edit Format Run Options Window Help

weak_learner = tree.DecisionTreeClassifier(max_leaf_nodes = 11)

from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(estimator = weak_learner,
                          n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))|
```

Ln: 44 Col: 42

Решим с помощью AdaBoost с настроенным деревом

```
weak_learner = tree.DecisionTreeClassifier(max_leaf_nodes = 11)

from sklearn.ensemble import AdaBoostClassifier

clf2 = AdaBoostClassifier(estimator = weak_learner,
                          n_estimators = 400, random_state = 22222)
clf2.fit(train_x, train_y)
res2 = clf2.predict(test_x)

print(confusion_matrix(test_y, res2))
print(classification_report(test_y, res2))
```

Результат совершенно другой!

	precision	recall	f1-score	support
1	0.59	0.77	0.67	22
2	0.80	0.69	0.74	35
3	0.50	0.25	0.33	4
5	0.75	0.86	0.80	7
6	1.00	1.00	1.00	3
7	0.93	0.87	0.90	15
accuracy			0.74	86
macro avg	0.76	0.74	0.74	86
weighted avg	0.76	0.74	0.74	86

Сравним с DT и RF

Model	Precision	Recall	F1-score	accuracy
AB	0,76	0,74	0,74	0,74
DT	0,53	0,56	0,53	0,71
RF	0,73	0,70	0,71	0,74

Отлично! На уровне RandomForest, а по отдельным характеристикам даже лучше.

Градиентный бустинг

Градиентный бустинг (gradient boosting) — алгоритм машинного обучения, реализующий за счет ансамбля простых моделей (например, деревьев решений) идею градиентного спуска относительно задачи минимизации квадратичной функции потерь.

Градиентный бустинг считается одним из сильнейших алгоритмов машинного обучения.

Идея алгоритма

Классификацию будет снова выполнять композиция функций:

$$G_k(x) = f_0(x) + f_1(x) + f_2(x) + \dots + f_k(x)$$

В качестве $f_0(x)$ возьмем какой-то простой классификатор. Например, дерево небольшой глубины с оптимальными параметрами, подобранными на обучающей выборке X и метках класса y .

Последующие функции $f_i(x)$ будем обучать уже не на метках класса y , а на метках, полученных разницей между требуемым y и результатом $f_0(x) + \dots + f_{i-1}(x)$. Например, $f_1(x)$ будет обучаться на $s_1 = y - f_0(x)$, $f_2(x)$ будет обучаться на $s_2 = y - f_0(x) - f_1(x)$ и т.д.

Решим с помощью градиентного бустинга

```
many_classes.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lection9\many_classes.py (3.11.2)
File Edit Format Run Options Window Help

from sklearn.ensemble import GradientBoostingClassifier

clf4 = GradientBoostingClassifier(n_estimators = 400, max_leaf_nodes = 4,
                                  random_state = 22222)

clf4.fit(train_x, train_y)
res4 = clf4.predict(test_x)

print(confusion_matrix(test_y, res4))
print(classification_report(test_y, res4))|
```

Ln: 64 Col: 42

Решим с помощью градиентного бустинга

```
from sklearn.ensemble import GradientBoostingClassifier

clf4 = GradientBoostingClassifier(n_estimators = 400, max_leaf_nodes = 4,
                                random_state = 22222)
clf4.fit(train_x, train_y)
res4 = clf4.predict(test_x)

print(confusion_matrix(test_y, res4))
print(classification_report(test_y, res4))
```

Решим с помощью градиентного бустинга

	precision	recall	f1-score	support
1	0.71	0.77	0.74	22
2	0.79	0.77	0.78	35
3	0.40	0.50	0.44	4
5	0.67	0.57	0.62	7
6	1.00	1.00	1.00	3
7	0.93	0.87	0.90	15
accuracy			0.77	86
macro avg	0.75	0.75	0.75	86
weighted avg	0.77	0.77	0.77	86

Сравним с DT и RF

Model	Precision	Recall	F1-score	accuracy
AB	0,76	0,74	0,74	0,74
DT	0,53	0,56	0,53	0,71
RF	0,73	0,70	0,71	0,74
GB	0,75	0,75	0,75	0,77

Интернет ресурсы и литература

1. <https://deepmachinelearning.ru/docs/Machine-learning/Boosting/AdaBoost>
2. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier>
3. https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_multiclass.html#sphx-glr-auto-examples-ensemble-plot-adaboost-multiclass-py
4. <https://education.yandex.ru/handbook/ml/article/gradientnyj-busting>
5. <https://deepmachinelearning.ru/docs/Machine-learning/Boosting/GB-idea>
6. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>