**Домашнее задание № 3.**

**Деревья решений в игрушечной задаче и на данных Adult**

**В задании Вам предлагается разобраться с тем, как работает дерево решений, на игрушечном примере, затем обучить и настроить деревья и (при желании) случайный лес в задаче классификации на данных Adult.**

Подключаем необходимые библиотеки

%matplotlib inline

from matplotlib import pyplot as plt

plt.rcParams['figure.figsize'] = (10, 8)

import collections

import numpy as np

import pandas as pd

import seaborn as sns

from sklearn import preprocessing

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import GridSearchCV

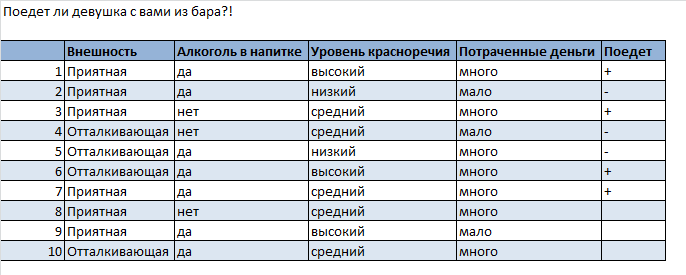
from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

## Часть 1. Игрушечный набор данных "девушка в баре"

**Цель – "на пальцах", с помощью игрушечной задачи классификации разобраться в том, как работают деревья решений. Само по себе дерево решений – довольно слабый алгоритм, но основанные на нем алгоритмы случайного леса и градиентного бустинга - пожалуй, лучшее, что есть на сегодняшний день (в задачах, где можно обойтись без нейронных сетей). Поэтому разобраться в том, как работает дерево решений, полезно.**

**Рассмотрим игрушечную задачу бинарной классификации: поедет ли с Вами девушка из бара? Это будет зависеть от Вашей внешности и красноречия, крепости предлагаемых напитков и, как это ни меркантильно, от количества потраченных в баре денег.**



### Создание набора данных

# Создание датафрейма с dummy variables

def create\_df(dic, feature\_list):

out = pd.DataFrame(dic)

out = pd.concat([out, pd.get\_dummies(out[feature\_list])], axis = 1)

out.drop(feature\_list, axis = 1, inplace = True)

return out

# Некоторые значения признаков есть в тесте, но нет в трейне и наоборот

def intersect\_features(train, test):

common\_feat = list( set(train.keys()) & set(test.keys()))

return train[common\_feat], test[common\_feat]

features = ['Внешность', 'Алкоголь\_в\_напитке',

'Уровень\_красноречия', 'Потраченные\_деньги']

**Обучающая выборка**

df\_train = {}

df\_train['Внешность'] = ['приятная', 'приятная', 'приятная', 'отталкивающая',

'отталкивающая', 'отталкивающая', 'приятная']

df\_train['Алкоголь\_в\_напитке'] = ['да', 'да', 'нет', 'нет', 'да', 'да', 'да']

df\_train['Уровень\_красноречия'] = ['высокий', 'низкий', 'средний', 'средний', 'низкий',

'высокий', 'средний']

df\_train['Потраченные\_деньги'] = ['много', 'мало', 'много', 'мало', 'много',

'много', 'много']

df\_train['Поедет'] = LabelEncoder().fit\_transform(['+', '-', '+', '-', '-', '+', '+'])

df\_train = create\_df(df\_train, features)

df\_train

**Тестовая выборка**

df\_test = {}

df\_test['Внешность'] = ['приятная', 'приятная', 'отталкивающая']

df\_test['Алкоголь\_в\_напитке'] = ['нет', 'да', 'да']

df\_test['Уровень\_красноречия'] = ['средний', 'высокий', 'средний']

df\_test['Потраченные\_деньги'] = ['много', 'мало', 'много']

df\_test = create\_df(df\_test, features)

df\_test

# Некоторые значения признаков есть в тесте, но нет в трейне и наоборот

y = df\_train['Поедет']

df\_train, df\_test = intersect\_features(train=df\_train, test=df\_test)

df\_train

df\_test

**Постройте от руки (или в графическом редакторе) дерево решений для этого набора данных. Дополнительно (для желающих) – можете сделать отрисовку дерева и написать код для построения всего дерева.**

Вопрос 1. Какова энтропия начальной системы (S0)? Под состояниями системы понимаем значения признака "Поедет" – 0 или 1 (то есть всего 2 состояния).

Вопрос 2. Рассмотрим разбиение обучающей выборки по признаку "Внешность\_приятная". Какова энтропия S1 левой группы, тех, у кого внешность приятная, и правой группы – S2? Каков прирост информации при данном разбиении (IG)?

# Ваш код здесь

**Постройте с помощью** sklearn **дерево решений, обучив его на обучающей выборке. Глубину можно не ограничивать.**

# Ваш код здесь

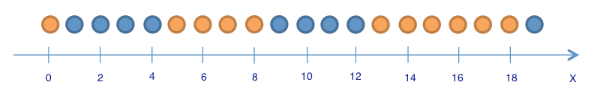
**Дополнительно: отобразите дерево с помощью graphviz. Можно использовать** pydot **или, например,** [онлайн-сервис](https://www.coolutils.com/ru/online/DOT-to-PNG) **dot2png.**

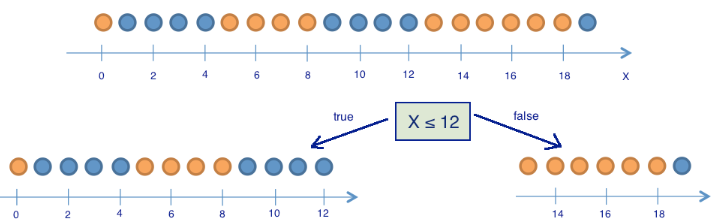
# Ваш код здесь

## Часть 2. Функции для расчета энтропии и прироста информации

Пример для проверки: 9 синих шариков и 11 желтых. Пусть шарик находится в состоянии "1", если он синий и "0" – если он желтый.

balls = [1 for i in range(9)] + [0 for i in range(11)]



Дальше пусть шарики разбиваются на 2 группы 

# две группы

balls\_left = [1 for i in range(8)] + [0 for i in range(5)] # 8 синих и 5 желтых

balls\_right = [1 for i in range(1)] + [0 for i in range(6)] # 1 синий и 6 желтых

**Реализуйте функцию для расчета энтропии Шеннона.**

def entropy(a\_list):

# Ваш код здесь

pass

Проверка

print(entropy(balls)) # 9 синих и 11 желтых

print(entropy(balls\_left)) # 8 синих и 5 желтых

print(entropy(balls\_right)) # 1 синий и 6 желтых

print(entropy([1,2,3,4,5,6])) # энтропия игральной кости с несмещенным центром тяжести

Вопрос 3. Чему равна энтропия состояния, заданного списком balls\_left?

Вопрос 4. Чему равна энтропия игральной кости с несмещенным центром тяжести?

# расчет прироста информации

def information\_gain(root, left, right):

''' root - изначальный набор данных, left и right два разбиения изначального набора'''

# Ваш код здесь

pass

Вопрос 5. Каков прирост информации при разделении выборки на balls\_left и balls\_right?

# Ваш код здесь

def best\_feature\_to\_split(X, y):

''' Выводит прирост информации при разбиении по каждому признаку'''

# Ваш код здесь

pass

**Опционально:**

* реализуйте алгоритм построения дерева за счет рекурсивного вызова функции best\_feature\_to\_split
* нарисуйте полученное дерево

## Часть 3. Набор данных "Adult"

**Описание набора:**

[**Набор**](http://archive.ics.uci.edu/ml/machine-learning-databases/adult) **данных UCI Adult**: классификация людей с помощью демографических данных для прогнозирования, зарабатывает ли человек более $ 50 000 в год.

Описание признаков:

**Age** – возраст, количественный признак  
**Workclass** – тип работодателя, количественный признак  
**fnlwgt** – итоговый вес обьекта, количественный признак  
**Education** – уровень образования, качественный признак  
**Education\_Num** – количество лет обучения, количественный признак  
**Martial\_Status** – семейное положение, категориальный признак  
**Occupation** – профессия, категориальный признак  
**Relationship** – тип семейных отношений, категориальный признак  
**Race** – раса, категориальный признак  
**Sex** – пол, качественный признак  
**Capital\_Gain** – прирост капитала, количественный признак  
**Capital\_Loss** – потери капитала, количественный признак  
**Hours\_per\_week** – количество часов работы в неделю, количественный признак  
**Country** – страна, категориальный признак

Целевая переменная: **Target** – уровень заработка, категориальный (бинарный) признак

**Считываем обучающую и тестовую выборки.**

data\_train = pd.read\_csv('../../data/adult\_train.csv', sep=';')

data\_train.tail()

data\_test = pd.read\_csv('../../data/adult\_test.csv', sep=';')

data\_test.tail()

# необходимо убрать строки с неправильными метками в тестовой выборке

data\_test = data\_test[(data\_test['Target'] == ' >50K.')

| (data\_test['Target']==' <=50K.')]

# перекодируем target в числовое поле

data\_train.at[data\_train['Target'] == ' <=50K', 'Target'] = 0

data\_train.at[data\_train['Target'] == ' >50K', 'Target'] = 1

data\_test.at[data\_test['Target'] == ' <=50K.', 'Target'] = 0

data\_test.at[data\_test['Target'] == ' >50K.', 'Target'] = 1

**Первичный анализ данных.**

data\_test.describe(include='all').T

data\_train['Target'].value\_counts()

fig = plt.figure(figsize=(25, 15))

cols = 5

rows = np.ceil(float(data\_train.shape[1]) / cols)

for i, column in enumerate(data\_train.columns):

ax = fig.add\_subplot(rows, cols, i + 1)

ax.set\_title(column)

if data\_train.dtypes[column] == np.object:

data\_train[column].value\_counts().plot(kind="bar", axes=ax)

else:

data\_train[column].hist(axes=ax)

plt.xticks(rotation="vertical")

plt.subplots\_adjust(hspace=0.7, wspace=0.2)

**Проверяем типы данных**

data\_train.dtypes

data\_test.dtypes

Выяснилось, что в тесте возраст отнесен к типу object, необходимо это исправить.

data\_test['Age'] = data\_test['Age'].astype(int)

Также приведем показатели типа float в int для соответствия train и test выборок.

data\_test['fnlwgt'] = data\_test['fnlwgt'].astype(int)

data\_test['Education\_Num'] = data\_test['Education\_Num'].astype(int)

data\_test['Capital\_Gain'] = data\_test['Capital\_Gain'].astype(int)

data\_test['Capital\_Loss'] = data\_test['Capital\_Loss'].astype(int)

data\_test['Hours\_per\_week'] = data\_test['Hours\_per\_week'].astype(int)

**Заполним пропуски в количественных полях медианными значениями, а в категориальных – наиболее часто встречающимся значением**

# выделим в выборках категориальные и числовые поля

categorical\_columns\_train = [c for c in data\_train.columns

if data\_train[c].dtype.name == 'object']

numerical\_columns\_train = [c for c in data\_train.columns

if data\_train[c].dtype.name != 'object']

categorical\_columns\_test = [c for c in data\_test.columns

if data\_test[c].dtype.name == 'object']

numerical\_columns\_test = [c for c in data\_test.columns

if data\_test[c].dtype.name != 'object']

print('categorical\_columns\_test:', categorical\_columns\_test)

print('categorical\_columns\_train:', categorical\_columns\_train)

print('numerical\_columns\_test:', numerical\_columns\_test)

print('numerical\_columns\_train:', numerical\_columns\_train)

# заполним пропуски

for c in categorical\_columns\_train:

data\_train[c] = data\_train[c].fillna(data\_train[c].mode())

for c in categorical\_columns\_test:

data\_test[c] = data\_test[c].fillna(data\_train[c].mode())

for c in numerical\_columns\_train:

data\_train[c] = data\_train[c].fillna(data\_train[c].median())

for c in numerical\_columns\_test:

data\_test[c] = data\_test[c].fillna(data\_train[c].median())

**Кодируем категориальные признаки 'Workclass', 'Education', 'Martial\_Status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Country'. Это можно сделать с помощью метода** pandas get\_dummies**.**

data\_train = pd.concat([data\_train, pd.get\_dummies(data\_train['Workclass'],

prefix="Workclass"),

pd.get\_dummies(data\_train['Education'], prefix="Education"),

pd.get\_dummies(data\_train['Martial\_Status'], prefix="Martial\_Status"),

pd.get\_dummies(data\_train['Occupation'], prefix="Occupation"),

pd.get\_dummies(data\_train['Relationship'], prefix="Relationship"),

pd.get\_dummies(data\_train['Race'], prefix="Race"),

pd.get\_dummies(data\_train['Sex'], prefix="Sex"),

pd.get\_dummies(data\_train['Country'], prefix="Country")],

axis=1)

data\_test = pd.concat([data\_test, pd.get\_dummies(data\_test['Workclass'], prefix="Workclass"),

pd.get\_dummies(data\_test['Education'], prefix="Education"),

pd.get\_dummies(data\_test['Martial\_Status'], prefix="Martial\_Status"),

pd.get\_dummies(data\_test['Occupation'], prefix="Occupation"),

pd.get\_dummies(data\_test['Relationship'], prefix="Relationship"),

pd.get\_dummies(data\_test['Race'], prefix="Race"),

pd.get\_dummies(data\_test['Sex'], prefix="Sex"),

pd.get\_dummies(data\_test['Country'], prefix="Country")],

axis=1)

data\_train.drop(['Workclass', 'Education', 'Martial\_Status',

'Occupation', 'Relationship', 'Race', 'Sex', 'Country'],

axis=1, inplace=True)

data\_test.drop(['Workclass', 'Education', 'Martial\_Status', 'Occupation',

'Relationship', 'Race', 'Sex', 'Country'],

axis=1, inplace=True)

data\_test.describe(include='all').T

set(data\_train.columns) - set(data\_test.columns)

data\_train.shape, data\_test.shape

**В тестовой выборке не оказалось Голландии. Заведем необходимый признак из нулей.**

data\_test['Country\_ Holand-Netherlands'] = np.zeros([data\_test.shape[0], 1])

set(data\_train.columns) - set(data\_test.columns)

data\_train.head(2)

data\_test.head(2)

X\_train=data\_train.drop(['Target'], axis=1)

y\_train = data\_train['Target']

X\_test=data\_test.drop(['Target'], axis=1)

y\_test = data\_test['Target']

## 3.1. Дерево решений без настройки параметров

**Обучите на имеющейся выборке дерево решений (**DecisionTreeClassifier**) максимальной глубины 3 и получите качество на тесте. Используйте параметр** random\_state **= 17 для воспроизводимости результатов.**

tree = # Ваш код здесь

tree.fit # Ваш код здесь

**Сделайте с помощью полученной модели прогноз для тестовой выборки.**

tree\_predictions = tree.predict # Ваш код здесь

accuracy\_score # Ваш код здесь

Вопрос 6. Какова доля правильных ответов дерева решений на тестовой выборке при максимальной глубине дерева = 3 и random\_state = 17?

## 3.2. Дерево решений с настройкой параметров

**Обучите на имеющейся выборке дерево решений (**DecisionTreeClassifier**, опять** random\_state **= 17 ). Максимальную глубину настройте на кросс-валидации с помощью** GridSearchCV**. Проведите 5-кратную кросс-валидацию**

tree\_params = {'max\_depth': range(2,11)}

locally\_best\_tree = GridSearchCV # Ваш код здесь

locally\_best\_tree.fit # Ваш код здесь

print("Best params:", locally\_best\_tree.best\_params\_)

print("Best cross validaton score", locally\_best\_tree.best\_score\_)

**Обучите на имеющейся выборке дерево решений максимальной глубины 9 (это лучшее значение** max\_depth **в моем случае) и оцените долю правильных ответов на тесте. Используйте параметр random\_state = 17 для воспроизводимости результатов.**

tuned\_tree = # Ваш код здесь

tuned\_tree.fit # Ваш код здесь

tuned\_tree\_predictions = tuned\_tree.predict # Ваш код здесь

accuracy\_score # Ваш код здесь

Вопрос 7. Какова доля правильных ответов дерева решений на тестовой выборке при максимальной глубине дерева = 9 и random\_state = 17?

## 3.3. Случайный лес без настройки параметров (опционально)¶

**Немного забежим вперед и попробуем в нашей задаче случайный лес. Пока можно его себе представлять, как куча деревьев решений, обученных на немного разных подвыборках исходной обучающей выборки, причем эта куча деревьев обычно работает существенно лучше, чем отдельные деревья.**

**Обучите на имеющейся выборке случайный лес (**RandomForestClassifier**), число деревьев сделайте равным ста, а** random\_state **= 17.**

rf = # Ваш код здесь

rf.fit # Ваш код здесь

**Сделайте с помощью полученной модели прогноз для тестовой выборки.**

forest\_predictions = rf.predict # Ваш код здесь

accuracy\_score # Ваш код здесь

## 3.4. Случайный лес с настройкой параметров (опционально)

**Обучите на имеющейся выборке случайный лес (**RandomForestClassifier**). Максимальную глубину и максимальное число признаков для каждого дерева настройте с помощью GridSearchCV.**

forest\_params = {'max\_depth': range(10, 21),

'max\_features': range(5, 105, 10)}

locally\_best\_forest = GridSearchCV # Ваш код здесь

locally\_best\_forest.fit # Ваш код здесь

print("Best params:", locally\_best\_forest.best\_params\_)

print("Best cross validaton score", locally\_best\_forest.best\_score\_)

**Сделайте с помощью полученной модели прогноз для тестовой выборки.**

tuned\_forest\_predictions = locally\_best\_forest.predict # Ваш код здесь

accuracy\_score # Ваш код здесь