# Классификация, деревья решений и метод ближайших соседей

В первых двух темах (1, 2) мы попрактиковались в первичном анализе данных с Pandas и в построении картинок, позволяющих делать выводы по данным. Сегодня наконец перейдем к машинному обучению. Поговорим о задачах машинного обучения и рассмотрим 2 простых подхода – деревья решений и метод ближайших соседей. Также обсудим, как с помощью кросс-валидации выбирать модель для конкретных данных.

5. Примеры применения

Деревья решений и метод ближайших соседей в задаче прогнозирования оттока клиентов телеком-оператора

Сложный случай для деревьев решений

Деревья решений и метод ближайших соседей в задаче распознавания рукописных цифр MNIST

Сложный случай для метода ближайших соседей

1. Плюсы и минусы деревьев решений и метода ближайших соседей
2. Домашнее задание №3

Оглавление

[Классификация, деревья решений и метод ближайших соседей 1](#_Toc152092225)

[Введение 1](#_Toc152092226)

[1. Дерево решений 3](#_Toc152092227)

[Как строится дерево решений 4](#_Toc152092228)

[Как дерево решений работает с количественными признаками 11](#_Toc152092229)

[Основные параметры дерева 16](#_Toc152092230)

[Класс DecisionTreeClassifier в Scikit-learn 16](#_Toc152092231)

[Дерево решений в задаче регрессии 17](#_Toc152092232)

[2. Метод ближайших соседей 18](#_Toc152092233)

[Метод ближайших соседей в реальных задачах 19](#_Toc152092234)

[Класс KNeighborsClassifier в Scikit-learn 19](#_Toc152092235)

[3. Выбор параметров модели и кросс-валидация 20](#_Toc152092236)

[Примеры применения 21](#_Toc152092237)

[Деревья решений и метод ближайших соседей в задаче прогнозирования оттока клиентов телеком-оператора. 21](#_Toc152092238)

# Введение

Наверное, хочется сразу рвануть в бой, но сначала поговорим про то, какую именно задачу будем решать и каково ее место в области машинного обучения.

Классическое, общее (и не больно-то строгое) определение машинного обучения звучит так (T. Mitchell "Machine learning", 1997): г*оворят, что компьютерная программа обучается при решении какой-то задачи из класса T, если ее производительность, согласно метрике P, улучшается при накоплении опыта E.*

Далее в разных сценариях под T, P, и E подразумеваются совершенно разные вещи. Среди самых популярных задач T в машинном обучении:

* классификация – отнесение объекта к одной из категорий на основании его признаков;
* регрессия – прогнозирование количественного признака объекта на основании прочих его признаков;
* кластеризация – разбиение множества объектов на группы на основании признаков этих объектов так, чтобы внутри групп объекты были похожи между собой, а вне одной группы – менее похожи;
* детекция аномалий – поиск объектов, "сильно непохожих" на все остальные в выборке либо на какую-то группу объектов;
* и много других, более специфичных. Хороший обзор дан в главе "Machine Learning basics" книги "Deep Learning" (Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016)

Под опытом E понимаются данные, и в зависимости от этого алгоритмы машинного обучения могут быть поделены на те, что обучаются с учителем и без учителя (supervised & unsupervised learning). В задачах обучения без учителя имеется выборка, состоящая из объектов, описываемых набором признаков. В задачах обучения с учителем вдобавок к этому для каждого объекта некоторой выборки, называемой обучающей, известен целевой признак – по сути это то, что хотелось бы прогнозировать для прочих объектов, не из обучающей выборки.

***Пример***

Задачи классификации и регрессии – это задачи обучения с учителем. В качестве примера будем представлять задачу кредитного скоринга: на основе накопленных кредитной организацией данных о своих клиентах хочется прогнозировать невозврат кредита. Здесь для алгоритма опыт E – это имеющаяся обучающая выборка: набор объектов (людей), каждый из которых характеризуется набором признаков (таких как возраст, зарплата, тип кредита, невозвраты в прошлом и т.д.), а также целевым признаком. Если этот целевой признак – просто факт невозврата кредита (1 или 0, т.е. банк знает о своих клиентах, кто вернул кредит, а кто – нет), то это задача (бинарной) классификации. Если известно, на сколько по времени клиент затянул с возвратом кредита и хочется то же самое прогнозировать для новых клиентов, то это будет задачей регрессии.

Наконец, третья абстракция в определении машинного обучения – это метрика оценки производительности алгоритма P. Такие метрики различаются для разных задач и алгоритмов, и про них мы будим говорить по мере изучения алгоритмов. Пока скажем, что *самая простая метрика качества алгоритма, решающего задачу классификации – это доля правильных ответов* (accuracy, не называйте ее точностью, этот перевод зарезервирован под другую метрику, precision) – то есть попросту доля верных прогнозов алгоритма на тестовой выборке.

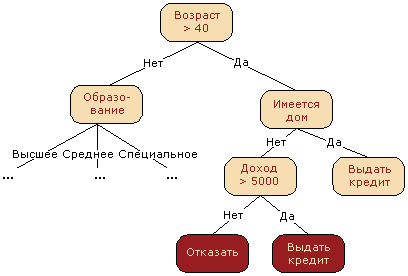
Далее будем говорить о двух задачах обучения с учителем: о классификации и регресcии.

# Дерево решений

Начнем обзор методов классификации и регрессии с одного из самых популярных – с дерева решений. Деревья решений используются в повседневной жизни в самых разных областях человеческой деятельности, порой и очень далеких от машинного обучения. Деревом решений можно назвать наглядную инструкцию, что делать в какой ситуации. Приведем пример из области консультирования научных сотрудников института. Финуниверситет выпускает инфо-схемы, облегчающие жизнь своим сотрудникам. Вот фрагмент инструкции по публикации научной статьи на портале института.

В терминах машинного обучения можно сказать, что это элементарный классификатор, который определяет форму публикации на портале (книга, статья, глава книги, препринт, публикация в "Финуниверситете и СМИ") по нескольким признакам: типу публикации (монография, брошюра, статья и т.д.), типу издания, где опубликована статья (научный журнал, сборник трудов и т.д.) и остальным.



Зачастую дерево решений служит обобщением опыта экспертов, средством передачи знаний будущим сотрудникам или моделью бизнес-процесса компании. Например, до внедрения масштабируемых алгоритмов машинного обучения в банковской сфере задача кредитного скоринга решалась экспертами. Решение о выдаче кредита заемщику принималось на основе некоторых интуитивно (или по опыту) выведенных правил, которые можно представить в виде дерева решений.

В этом случае можно сказать, что решается задача бинарной классификации (целевой класс имеет два значения: "Выдать кредит" и "Отказать") по признакам "Возраст", "Наличие дома", "Доход" и "Образование".

Дерево решений как алгоритм машинного обучения – по сути то же самое: объединение логических правил вида "Значение признака *a* меньше *x* И Значение признака *b* меньше *y*… => Класс 1" в структуру данных "Дерево". Огромное преимущество деревьев решений в том, что они легко интерпретируемы, понятны человеку. Например, по схеме на рисунке выше можно объяснить заемщику, почему ему было отказано в кредите. Скажем, потому, что у него нет дома и доход меньше 5000. Как мы увидим дальше, многие другие, хоть и более точные, модели не обладают этим свойством и могут рассматриваться скорее, как "черный ящик", в который загрузили данные и получили ответ. В связи с этой "понятностью" деревьев решений и их сходством с моделью принятия решений человеком (можно легко объяснять боссу свою модель), деревья решений получили огромную популярность, а один из представителей этой группы методов классификации, “Дерево решений” С4.5, рассматривается первым в списке 10 лучших алгоритмов интеллектуального анализа данных ("Top 10 algorithms in data mining", Knowledge and Information Systems, 2008. . [PDF](https://raw.githubusercontent.com/csuldw/MachineLearning/master/doc/10Algorithms-08.pdf)).

### Как строится дерево решений

В примере с кредитным скорингом мы видели, что решение о выдаче кредита принималось на основе возраста, наличия недвижимости, дохода и других. Но какой признак выбрать первым? Для этого рассмотрим пример попроще, где все признаки бинарные.

Здесь можно вспомнить игру "20 вопросов", которая часто упоминается во введении в деревья решений. Наверняка каждый в нее играл. Один человек загадывает знаменитость, а второй пытается отгадать, задавая только вопросы, на которые можно ответить "Да" или "Нет" (опустим варианты "не знаю" и "не могу сказать"). Какой вопрос отгадывающий задаст первым делом? Конечно, такой, который сильнее всего уменьшит количество оставшихся вариантов. К примеру, вопрос "Это Анджелина Джоли?" в случае отрицательного ответа оставит более 7 миллиардов вариантов для дальнейшего перебора (конечно, поменьше, не каждый человек – знаменитость, но все равно немало), а вот вопрос "Это женщина?" отсечет уже около половины знаменитостей. То есть, признак "пол" намного лучше разделяет выборку людей, чем признак "это Анджелина Джоли", "национальность-испанец" или "любит футбол". Это интуитивно соответствует понятию прироста информации, основанного на энтропии.

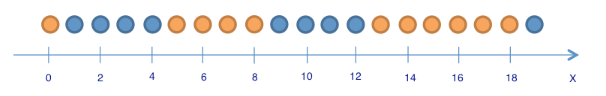
***Энтропия***

Энтропия Шеннона определяется для системы с  возможными состояниями следующим образом:

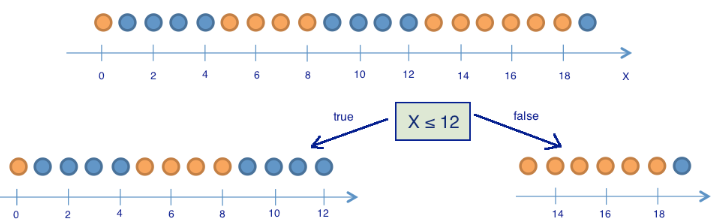


где  – вероятности нахождения системы в -ом состоянии. Это очень важное понятие, используемое в физике, теории информации и других областях. Опуская предпосылки введения (комбинаторные и теоретико-информационные) этого понятия, отметим, что, интуитивно, энтропия соответствует степени хаоса в системе. Чем выше энтропия, тем менее упорядочена система и наоборот. Это поможет нам формализовать "эффективное разделение выборки", про которое мы говорили в контексте игры "20 вопросов".

***Пример***

Для иллюстрации того, как энтропия поможет определить хорошие признаки для построения дерева, приведем тот же игрушечный пример, что в материале "Энтропия и деревья принятия решений". Будем предсказывать цвет шарика по его координате. Конечно, ничего общего с жизнью это не имеет, но позволяет показать, как энтропия используется для построения дерева решений.

Здесь 9 синих шариков и 11 желтых. Если мы наудачу вытащили шарик, то он с вероятностью  будет синим и с вероятностью  – желтым. Значит, энтропия состояния . Само это значение пока ни о чем нам не говорит. Теперь посмотрим, как изменится энтропия, если разбить шарики на две группы – с координатой меньше либо равной 12 и больше 12.



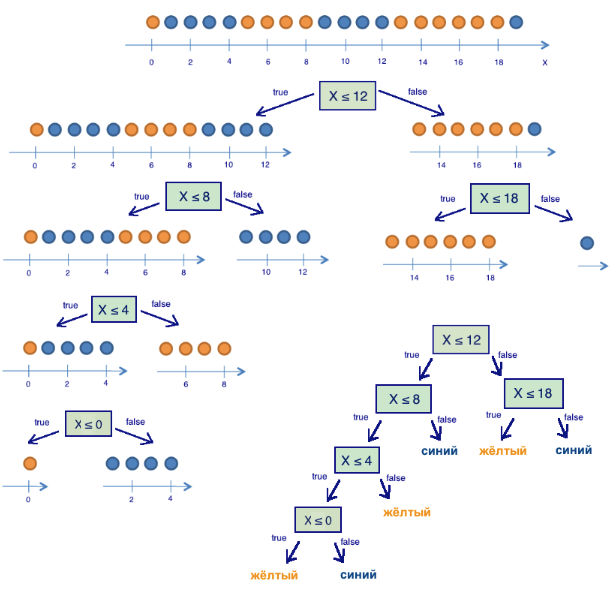
В левой группе оказалось 13 шаров, из которых 8 синих и 5 желтых. Энтропия этой группы равна . В правой группе оказалось 7 шаров, из которых 1 синий и 6 желтых. Энтропия правой группы равна  Как видим, энтропия уменьшилась в обеих группах по сравнению с начальным состоянием, хоть в левой и не сильно. Поскольку энтропия – по сути степень хаоса (или неопределенности) в системе, уменьшение энтропии называют приростом информации. Формально прирост информации (information gain, IG) при разбиении выборки по признаку  (в нашем примере это признак "") определяется как



где  – число групп после разбиения,  – число элементов выборки, у которых признак  имеет -ое значение. В нашем случае после разделения получилось две группы () – одна из 13 элементов (), вторая – из 7 (). Прирост информации получился



Получается, разделив шарики на две группы по признаку "координата меньше либо равна 12", мы уже получили более упорядоченную систему, чем в начале. Продолжим деление шариков на группы до тех пор, пока в каждой группе шарики не будут одного цвета.



Для правой группы потребовалось всего одно дополнительное разбиение по признаку "координата меньше либо равна 18", для левой – еще три. Очевидно, энтропия группы с шариками одного цвета равна 0 (), что соответствует представлению, что группа шариков одного цвета – упорядоченная.

В итоге мы построили дерево решений, предсказывающее цвет шарика по его координате. Отметим, что такое дерево решений может плохо работать для новых объектов (определения цвета новых шариков), поскольку оно идеально подстроилось под обучающую выборку (изначальные 20 шариков). Для классификации новых шариков лучше подойдет дерево с меньшим числом "вопросов", или разделений, пусть даже оно и не идеально разбивает по цветам обучающую выборку. Эту проблему, переобучение, мы еще рассмотрим далее.

***Алгоритм построения дерева***

Можно убедиться в том, что построенное в предыдущем примере дерево является в некотором смысле оптимальным – потребовалось только 5 "вопросов" (условий на признак ), чтобы "подогнать" дерево решений под обучающую выборку, то есть чтобы дерево правильно классифицировало любой обучающий объект. При других условиях разделения выборки дерево получится глубже.

В основе популярных алгоритмов построения дерева решений, таких как ID3 и C4.5, лежит принцип жадной максимизации прироста информации – на каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Дальше процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).

В разных алгоритмах применяются разные эвристики для "ранней остановки" или "отсечения", чтобы избежать построения переобученного дерева.

def build(L):

create node t

if the stopping criterion is True:

assign a predictive model to t

else:

Find the best binary split L = L\_left + L\_right

t.left = build(L\_left)

t.right = build(L\_right)

return t

***Другие критерии качества разбиения в задаче классификации***

Мы разобрались в том, как понятие энтропии позволяет формализовать представление о качестве разбиения в дереве. Но это всего лишь эвристика, существуют и другие:

* Неопределенность Джини (Gini impurity): . Максимизацию этого критерия можно интерпретировать как максимизацию числа пар объектов одного класса, оказавшихся в одном поддереве. Подробнее об этом (как и обо многом другом) можно узнать из репозитория (https://github.com/esokolov/ml-course-hse) Евгения Соколова. Не путать с индексом Джини!
* Ошибка классификации (misclassification error): 

На практике ошибка классификации почти не используется, а неопределенность Джини и прирост информации работают почти одинаково.

В случае задачи бинарной классификации ( – вероятность объекта иметь метку +) энтропия и неопределенность Джини примут следующий вид:





Когда мы построим графики этих двух функций от аргумента , то увидим, что график энтропии очень близок к графику удвоенной неопределенности Джини, и поэтому на практике эти два критерия "работают" почти одинаково.

***Импорт библиотек***

from \_\_future\_\_ import division, print\_function

# отключим всякие предупреждения Anaconda

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

%matplotlib inline

import seaborn as sns

from matplotlib import pyplot as plt

***Отрисовка картинки***

plt.rcParams['figure.figsize'] = (6,4)

xx = np.linspace(0,1,50)

plt.plot(xx, [2 \* x \* (1-x) for x in xx], label='gini')

plt.plot(xx, [4 \* x \* (1-x) for x in xx], label='2\*gini')

plt.plot(xx, [-x \* np.log2(x) - (1-x) \* np.log2(1 - x) for x in xx], label='entropy')

plt.plot(xx, [1 - max(x, 1-x) for x in xx], label='missclass')

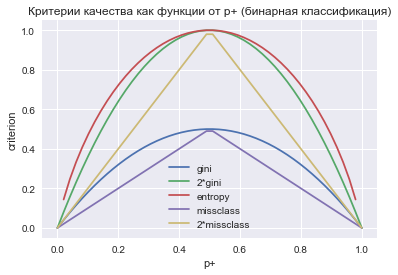
plt.plot(xx, [2 - 2 \* max(x, 1-x) for x in xx], label='2\*missclass')

plt.xlabel('p+')

plt.ylabel('criterion')

plt.title('Критерии качества как функции от p+ (бинарная классификация)')

plt.legend();



***Пример***

Рассмотрим пример применения дерева решений из библиотеки Scikit-learn для синтетических данных. Два класса будут сгенерированы из двух нормальных распределений с разными средними.

***Код для генерации данных***

# первый класс

np.random.seed(7)

train\_data = np.random.normal(size=(100, 2))

train\_labels = np.zeros(100)

# добавляем второй класс

train\_data = np.r\_[train\_data, np.random.normal(size=(100, 2), loc=2)]

train\_labels = np.r\_[train\_labels, np.ones(100)]

Напишем вспомогательную функцию, которая будет возвращать решетку для дальнейшей красивой визуализации.

def get\_grid(data, eps=0.01):

x\_min, x\_max = data[:, 0].min() - 1, data[:, 0].max() + 1

y\_min, y\_max = data[:, 1].min() - 1, data[:, 1].max() + 1

return np.meshgrid(np.arange(x\_min, x\_max, eps), np.arange(y\_min, y\_max, eps))

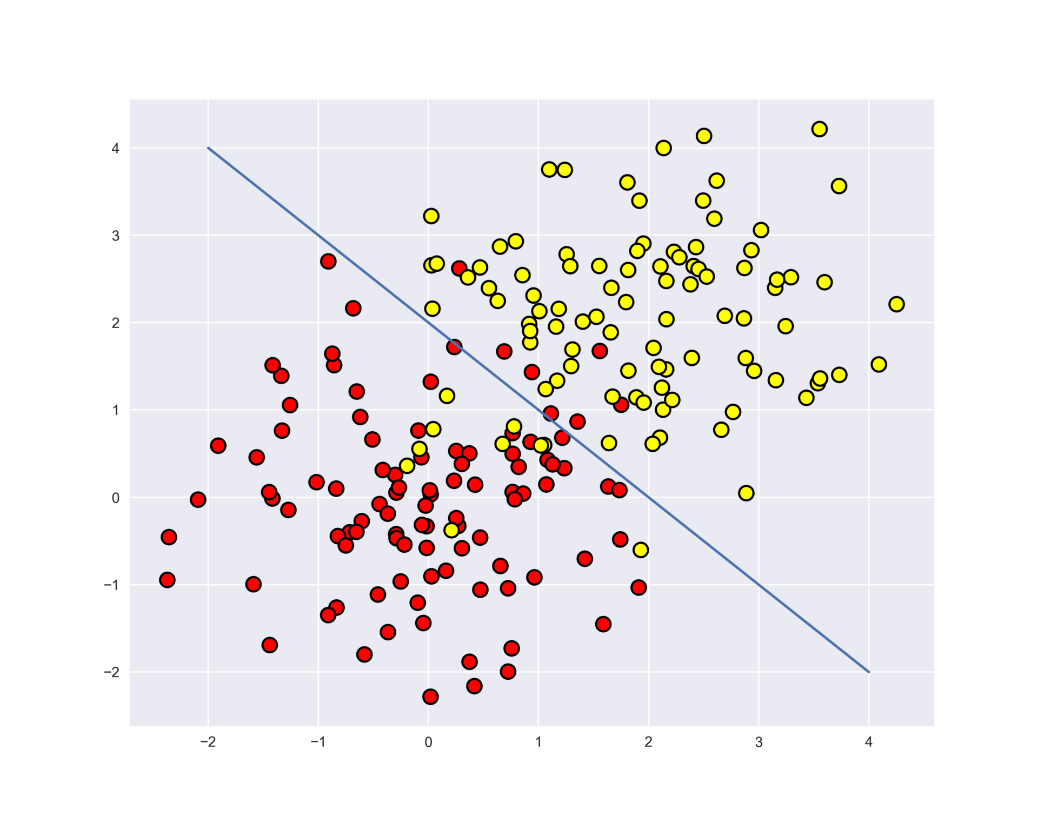
Отобразим данные. Неформально, задача классификации в этом случае – построить какую-то "хорошую" границу, разделяющую 2 класса (красные точки от желтых). Если утрировать, то машинное обучение в этом случае сводится к тому, как выбрать хорошую разделяющую границу. Возможно, прямая будет слишком простой границей, а какая-то сложная кривая, огибающая каждую красную точку – будет слишком сложной и будем много ошибаться на новых примерах из того же распределения, из которого пришла обучающая выборка. Интуиция подсказывает, что хорошо на новых данных будет работать какая-то гладкая граница, разделяющая 2 класса, или хотя бы просто прямая (в -мерном случае – гиперплоскость).

***Отрисовка картинки***

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

plt.scatter(train\_data[:, 0], train\_data[:, 1], c=train\_labels, s=100,

cmap='autumn', edgecolors='black', linewidth=1.5);

plt.plot(range(-2,5), range(4,-3,-1));

Попробуем разделить эти два класса, обучив дерево решений. В дереве будем использовать параметр max\_depth, ограничивающий глубину дерева. Визуализируем полученную границу разделения классов.

***Код для обучения дерева и отрисовки его разделяющей границы***

from sklearn.tree import DecisionTreeClassifier

# Напишем вспомогательную функцию, которая будет возвращать решетку для дальнейшей визуализации.

def get\_grid(data):

x\_min, x\_max = data[:, 0].min() - 1, data[:, 0].max() + 1

y\_min, y\_max = data[:, 1].min() - 1, data[:, 1].max() + 1

return np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

# параметр min\_samples\_leaf указывает, при каком минимальном количестве

# элементов в узле он будет дальше разделяться

clf\_tree = DecisionTreeClassifier(criterion='entropy', max\_depth=3, random\_state=17)

# обучаем дерево

clf\_tree.fit(train\_data, train\_labels)

# немного кода для отображения разделяющей поверхности

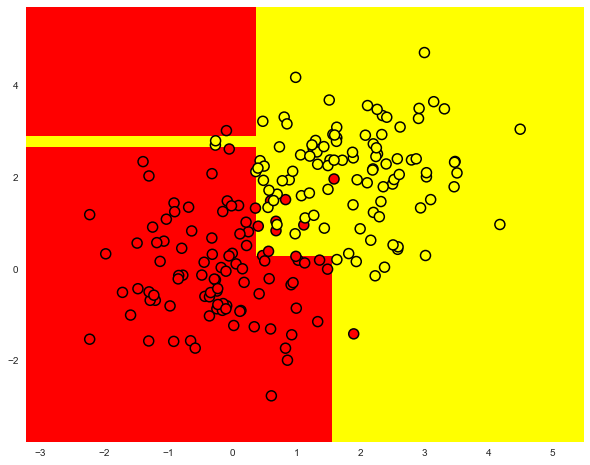
xx, yy = get\_grid(train\_data)

predicted = clf\_tree.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.pcolormesh(xx, yy, predicted, cmap='autumn')

plt.scatter(train\_data[:, 0], train\_data[:, 1], c=train\_labels, s=100,

cmap='autumn', edgecolors='black', linewidth=1.5);

А как выглядит само построенное дерево? Видим, что дерево "нарезает" пространство на 7 прямоугольников (в дереве 7 листьев). В каждом таком прямоугольнике прогноз дерева будет константным, по превалированию объектов того или иного класса.

***Код для отображения дерева***

from io import StringIO

import pydotplus

# используем .dot формат для визуализации дерева

from ipywidgets import Image

from sklearn.tree import export\_graphviz

dot\_data = StringIO()

export\_graphviz(clf\_tree, feature\_names=["x1", "x2"], out\_file=dot\_data, filled=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Image(value=graph.create\_png())

**conda install python- graphviz**

# используем .dot формат для визуализации дерева

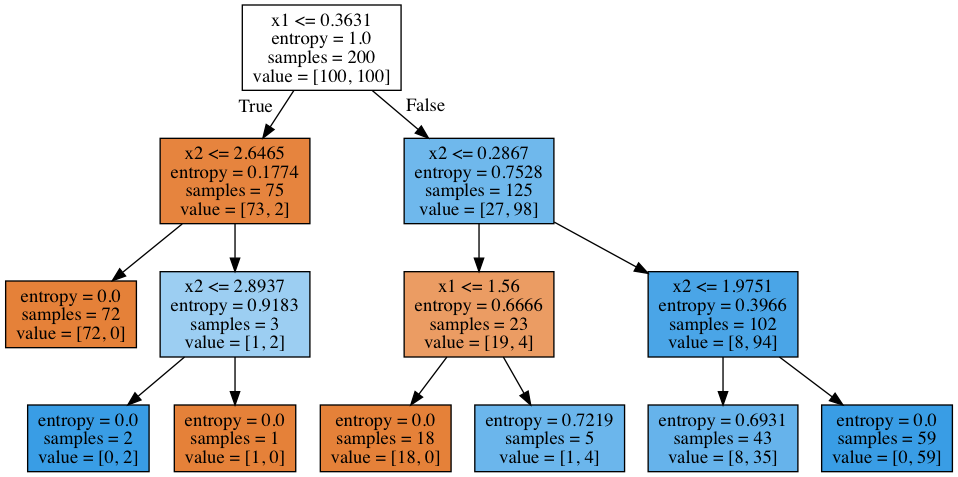
from sklearn.tree import export\_graphviz

export\_graphviz(clf\_tree, feature\_names=['x1', 'x2'],

out\_file='../../img/small\_tree.dot', filled=True)

# для этого понадобится библиотека pydot (pip install pydot)

!dot -Tpng '../../img/small\_tree.dot' -o '../../img/small\_tree.png'



Как "читается" такое дерево?

В начале было 200 объектов, 100 — одного класса и 100 – другого. Энтропия начального состояния была максимальной – 1. Затем было сделано разбиение объектов на 2 группы в зависимости от сравнения признака  со значением (найдите этот участок границы на рисунке выше, до дерева). При этом энтропия и в левой, и в правой группе объектов уменьшилась. И так далее, дерево строится до глубины 3. При такой визуализации чем больше объектов одного класса, тем цвет вершины ближе к темно-оранжевому и, наоборот, чем больше объектов второго класса, тем ближе цвет к темно-синему. В начале объектов одного класса поровну, поэтому корневая вершина дерева – белого цвета.

## Как дерево решений работает с количественными признаками

Допустим, в выборке имеется количественный признак "Возраст", имеющий много уникальных значений. Дерево решений будет искать лучшее (по критерию типа прироста информации) разбиение выборки, проверяя бинарные признаки типа "Возраст < 17", "Возраст < 22.87" и т.д. Но что если таких "нарезаний" возраста слишком много? А что если есть еще количественный признак "Зарплата", и зарплату тоже можно "нарезать" большим числом способов? Получается слишком много бинарных признаков для выбора лучшего на каждом шаге построения дерева. Для решения этой проблемы применяют эвристики для ограничения числа порогов, с которыми мы сравниваем количественный признак.

Рассмотрим это на игрушечном примере. Пусть имеется следующая выборка:

data = pd.DataFrame(

{

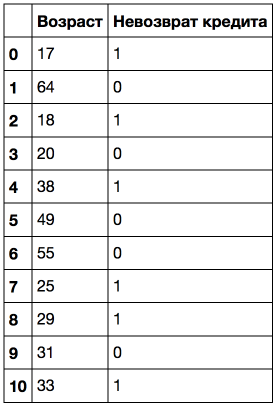
"Возраст": [17, 64, 18, 20, 38, 49, 55, 25, 29, 31, 33],

"Невозврат кредита": [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1],

}

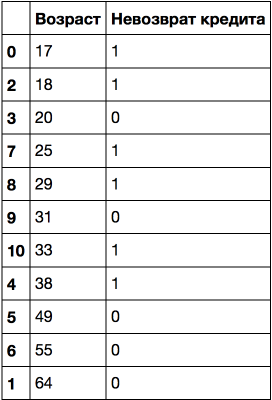
)

data



Отсортируем ее по возрастанию возраста.

data.sort\_values("Возраст")



Обучим на этих данных дерево решений (без ограничения глубины) и посмотрим на него.

age\_tree = DecisionTreeClassifier(random\_state=17)

age\_tree.fit(data["Возраст"].values.reshape(-1, 1), data["Невозврат кредита"].values)

dot\_data = StringIO()

export\_graphviz(age\_tree, feature\_names=["Возраст"], out\_file=dot\_data, filled=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Image(value=graph.create\_png())

***Код для обучения и отрисовки дерева***

age\_tree = DecisionTreeClassifier(random\_state=17)

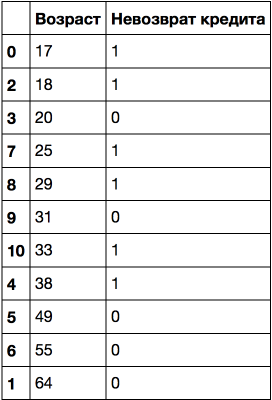
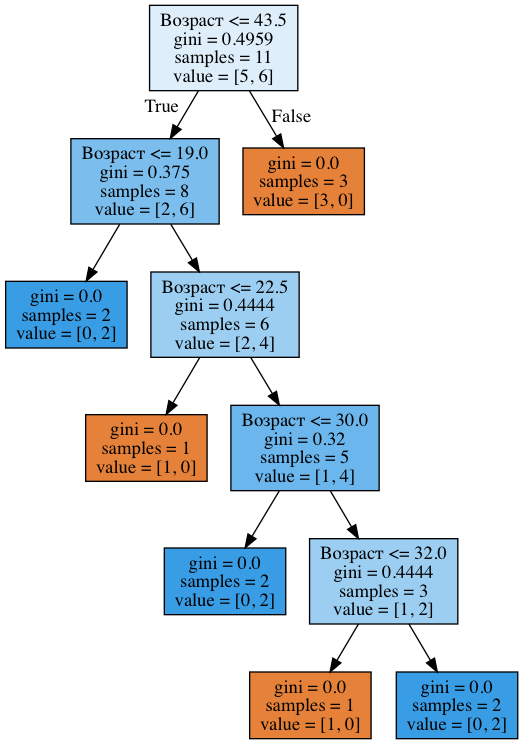
age\_tree.fit(data['Возраст'].values.reshape(-1, 1), data['Невозврат кредита'].values)

export\_graphviz(age\_tree, feature\_names=['Возраст'],

out\_file='../../img/age\_tree.dot', filled=True)

!dot -Tpng '../../img/age\_tree.dot' -o '../../img/age\_tree.png'

На картинке ниже видим, что дерево задействовало 5 значений, с которыми сравнивается возраст: 43.5, 19, 22.5, 30 и 32 года. Если приглядеться, то это аккурат средние значения между возрастами, при которых целевой класс "меняется" с 1 на 0 или наоборот. Сложная фраза, поэтому пример: 43.5 – это среднее между 38 и 49 годами, клиент, которому 38 лет не вернул кредит, а тот, которому 49 – вернул. Аналогично, 19 лет – среднее между 18 и 20 годами. То есть в качестве порогов для "нарезания" количественного признака, дерево "смотрит" на те значения, при которых целевой класс меняет свое значение.

Подумайте, почему не имеет смысла в данном случае рассматривать признак "Возраст < 17.5".

Рассмотрим пример посложнее: добавим признак "Зарплата" (тыс. рублей/месяц).

data2 = pd.DataFrame(

{

"Возраст": [17, 64, 18, 20, 38, 49, 55, 25, 29, 31, 33],

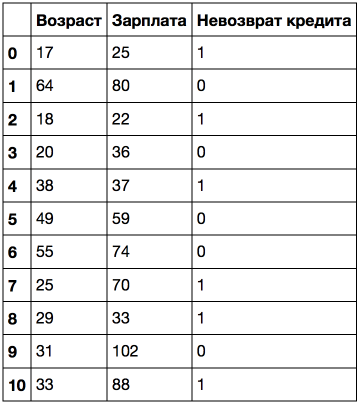
"Зарплата": [25, 80, 22, 36, 37, 59, 74, 70, 33, 102, 88],

"Невозврат кредита": [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1],

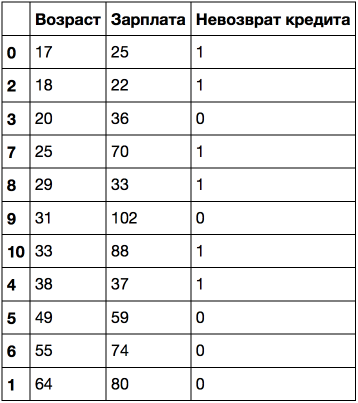
}

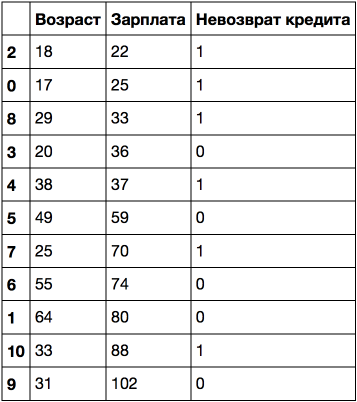
)

data2



Если отсортировать по возрасту, то целевой класс ("Невозврат кредита") меняется (с 1 на 0 или наоборот) 5 раз. А если отсортировать по зарплате – то 7 раз. Как теперь дерево будет выбирать признаки? Посмотрим.

 data2.sort\_values("Возраст")

data2.sort\_values("Зарплата")

age\_sal\_tree = DecisionTreeClassifier(random\_state=17)

age\_sal\_tree.fit(

data2[["Возраст", "Зарплата"]].values, data2["Невозврат кредита"].values

);

dot\_data = StringIO()

export\_graphviz(

age\_sal\_tree, feature\_names=["Возраст", "Зарплата"], out\_file=dot\_data, filled=True

)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Image(value=graph.create\_png())

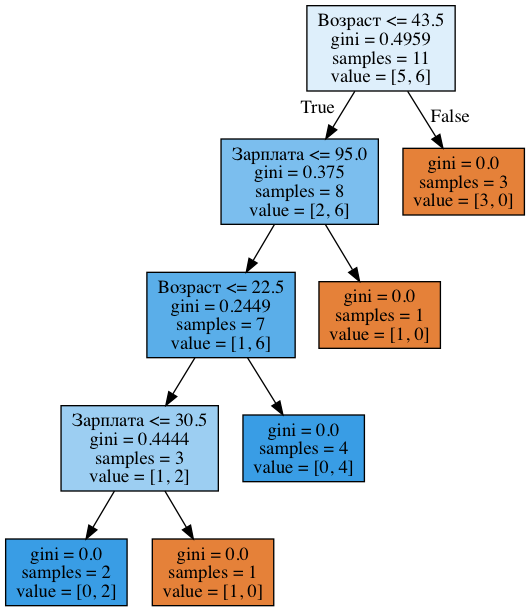
***Код для обучения и отрисовки дерева***

age\_sal\_tree = DecisionTreeClassifier(random\_state=17)

age\_sal\_tree.fit(data2[['Возраст', 'Зарплата']].values, data2['Невозврат кредита'].values);

export\_graphviz(age\_sal\_tree, feature\_names=['Возраст', 'Зарплата'],

out\_file='../../img/age\_sal\_tree.dot', filled=True)

!dot -Tpng '../../img/age\_sal\_tree.dot' -o '../../img/age\_sal\_tree.png'

Видим, что в дереве задействованы как разбиения по возрасту, так и по зарплате. Причем пороги, с которыми сравниваются признаки: 43.5 и 22.5 года – для возраста и 95 и 30.5 тыс. руб/мес – для зарплаты. И опять можно заметить, что 95 тыс. – это среднее между 88 и 102, при этом человек с зарплатой 88 оказался "плохим", а с 102 – "хорошим". То же самое для 30.5 тыс. То есть перебирались сравнения зарплаты и возраста не со всеми возможными значениями, а только с несколькими. А почему в дереве оказались именно эти признаки? Потому что по ним разбиения оказались лучше (по критерию неопределенности Джини).

**Вывод:** самая простая эвристика для обработки количественных признаков в дереве решений: количественный признак сортируется по возрастанию, и в дереве проверяются только те пороги, при которых целевой признак меняет значение. Звучит не очень строго, но надеюсь, я донес смысл с помощью игрушечных примеров.

Дополнительно, когда в данных много количественных признаков, и у каждого много уникальных значений, могут отбираться не все пороги, описанные выше, а только топ-*N*, дающих максимальный прирост все того же критерия. То есть, по сути, для каждого порога строится дерево глубины 1, считается насколько снизилась энтропия (или неопределенность Джини) и выбираются только лучшие пороги, с которыми стоит сравнивать количественный признак.

Для иллюстрации: при разбиении по признаку "Зарплата ≤ 34.5" в левой подгруппе энтропия 0 (все клиенты "плохие"), а в правой – 0.954 (3 "плохих" и 5 "хороших", можете проверить, 1 часть домашнего задания будет как раз на то, чтоб разобраться досконально с построением деревьев). Прирост информации получается примерно 0.3.  
А при разбиении по признаку "Зарплата ≤ 95" в левой подгруппе энтропия 0.97 (6 "плохих" и 4 "хороших"), а в правой – 0 (всего один объект). Прирост информации получается примерно 0.11.

Посчитав таким образом прирост информации для каждого разбиения, можно предварительно, до построения большого дерева (по всем признакам) отобрать пороги, с которыми будет сравниваться каждый количественный признак.

### Основные параметры дерева

В принципе дерево решений можно построить до такой глубины, чтоб в каждом листе был ровно один объект. Но на практике это не делается (если строится только одно дерево) из-за того, что такое дерево будет *переобученным* – оно слишком настроится на обучающую выборку и будет плохо работать на прогноз на новых данных. Где-то внизу дерева, на большой глубине будут появляться разбиения по менее важным признакам (например, приехал ли клиент из Саратова или Костромы). Если утрировать, может оказаться так, что из всех 4 клиентов, пришедших в банк за кредитом в зеленых штанах, никто не вернул кредит. Но мы не хотим, чтобы наша модель классификации порождала такие специфичные правила.

Есть два исключения, ситуации, когда деревья строятся до максимальной глубины:

* Случайный лес (композиция многих деревьев) усредняет ответы деревьев, построенных до максимальной глубины (почему стоит делать именно так, разберемся позже)
* Стрижка дерева (*pruning*). При таком подходе дерево сначала строится до максимальной глубины, потом постепенно, снизу вверх, некоторые вершины дерева убираются за счет сравнения по качеству дерева с данным разбиением и без него (сравнение проводится с помощью *кросс-валидации*, о которой чуть ниже). Подробнее можно почитать в материалах [репозитория](https://github.com/esokolov/ml-course-hse) Евгения Соколова.

### Класс DecisionTreeClassifier в Scikit-learn

Основные параметры класса sklearn.tree.DecisionTreeClassifier:

max\_depth – максимальная глубина дерева

max\_features - максимальное число признаков, по которым ищется лучшее разбиение в дереве (это нужно потому, что при большом количестве признаков будет "дорого" искать лучшее (по критерию типа прироста информации) разбиение среди всех признаков)

min\_samples\_leaf – минимальное число объектов в листе. У этого параметра есть понятная интерпретация: скажем, если он равен 5, то дерево будет порождать только те классифицирующие правила, которые верны как мимимум для 5 объектов

Параметры дерева надо настраивать в зависимости от входных данных, и делается это обычно с помощью кросс-валидации, про нее чуть ниже.

### Дерево решений в задаче регрессии

При прогнозировании количественного признака идея построения дерева остается та же, но меняется критерий качества:

Дисперсия вокруг среднего:



где *ℓ* – число объектов в листе, 𝑦𝑖 – значения целевого признака. Попросту говоря, минимизируя дисперсию вокруг среднего, мы ищем признаки, разбивающие выборку таким образом, что значения целевого признака в каждом листе примерно равны.

***Пример***

Сгенерируем данные, распределенные вокруг функции  c некоторым шумом, обучим на них дерево решений и изобразим, какие прогнозы делает дерево.

n\_train = 150

n\_test = 1000

noise = 0.1

def f(x):

x = x.ravel()

return np.exp(-x \*\* 2) + 1.5 \* np.exp(-(x - 2) \*\* 2)

def generate(n\_samples, noise):

X = np.random.rand(n\_samples) \* 10 - 5

X = np.sort(X).ravel()

y = np.exp(-X \*\* 2) + 1.5 \* np.exp(-(X - 2) \*\* 2) + \

np.random.normal(0.0, noise, n\_samples)

X = X.reshape((n\_samples, 1))

return X, y

X\_train, y\_train = generate(n\_samples=n\_train, noise=noise)

X\_test, y\_test = generate(n\_samples=n\_test, noise=noise)

from sklearn.tree import DecisionTreeRegressor

reg\_tree = DecisionTreeRegressor(max\_depth=5, random\_state=17)

reg\_tree.fit(X\_train, y\_train)

reg\_tree\_pred = reg\_tree.predict(X\_test)

plt.figure(figsize=(10, 6))

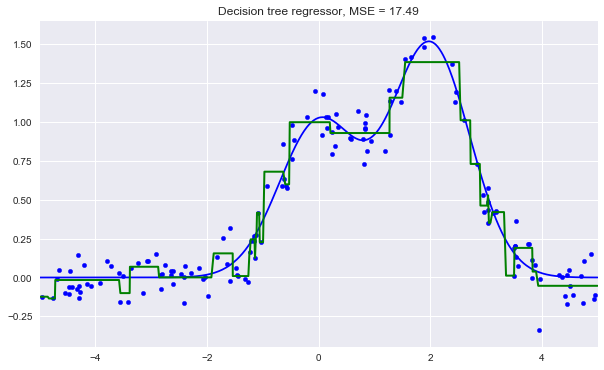
plt.plot(X\_test, f(X\_test), "b")

plt.scatter(X\_train, y\_train, c="b", s=20)

plt.plot(X\_test, reg\_tree\_pred, "g", lw=2)

plt.xlim([-5, 5])

plt.title("Decision tree regressor, MSE = %.2f" % np.sum((y\_test - reg\_tree\_pred) \*\* 2))

plt.show()

# Метод ближайших соседей

Метод ближайших соседей (k Nearest Neighbors, или kNN) — тоже очень популярный метод классификации, также иногда используемый в задачах регрессии. Это, наравне с деревом решений, один из самых понятных подходов к классификации. На уровне интуиции суть метода такова: посмотри на соседей, какие преобладают, таков и ты. Формально основой метода является гипотеза компактности: если метрика расстояния между примерами введена достаточно удачно, то схожие примеры гораздо чаще лежат в одном классе, чем в разных.

Согласно методу ближайших соседей, тестовый пример (зеленый шарик) будет отнесен к классу "синие", а не "красные".

Например, если не знаешь, какой тип товара указать в объявлении для Bluetooth-гарнитуры, можешь найти 5 похожих гарнитур, и если 4 из них отнесены к категории "Аксессуары", и только один - к категории "Техника", то здравый смысл подскажет для своего объявления тоже указать категорию "Аксессуары".

Для классификации каждого из объектов тестовой выборки необходимо последовательно выполнить следующие операции:

* Вычислить расстояние до каждого из объектов обучающей выборки
* Отобрать 𝑘 объектов обучающей выборки, расстояние до которых минимально
* Класс классифицируемого объекта — это класс, наиболее часто встречающийся среди 𝑘 ближайших соседей

Примечательное свойство такого подхода – его ленивость. Это значит, что вычисления начинаются только в момент классификации тестового примера, а заранее, только при наличии обучающих примеров, никакая модель не строится. В этом отличие, например, от ранее рассмотренного дерева решений, где сначала на основе обучающей выборки строится дерево, а потом относительно быстро происходит классификация тестовых примеров.

Стоит отметить, что метод ближайших соседей – хорошо изученный подход (в машинном обучении, эконометрике и статистике больше известно наверно только про линейную регрессию). Для метода ближайших соседей существует немало важных теорем, утверждающих, что на "бесконечных" выборках это оптимальный метод классификации. Авторы классической книги "The Elements of Statistical Learning" считают kNN теоретически идеальным алгоритмом, применимость которого просто ограничена вычислительными возможностями и проклятием размерностей.

### Метод ближайших соседей в реальных задачах

* В чистом виде kNN может послужить хорошим стартом (baseline) в решении какой-либо задачи;
* В соревнованиях Kaggle kNN часто используется для построения мета-признаков (прогноз kNN подается на вход прочим моделям) или в стекинге/блендинге;
* Идея ближайшего соседа расширяется и на другие задачи, например, в рекомендательных системах простым начальным решением может быть рекомендация какого-то товара (или услуги), популярного среди ближайших соседей человека, которому хотим сделать рекомендацию;
* На практике для больших выборок часто пользуются приближенными методами поиска ближайших соседей. Вот лекция Артема Бабенко (https://www.youtube.com/watch?v=UUm4MOyVTnE) про эффективные алгоритмы поиска ближайших соседей среди миллиардов объектов в пространствах высокой размерности (поиск по картинкам). Также известны открытые библиотеки, в которых реализованы такие алгоритмы, спасибо компании Spotify за ее библиотеку Annoy.

Качество классификации методом ближайших соседей зависит от нескольких параметров:

* число соседей
* метрика расстояния между объектами (часто используются метрика Хэмминга, евклидово расстояние, косинусное расстояние и расстояние Минковского). Отметим, что при использовании большинства метрик значения признаков надо масштабировать. Условно говоря, чтобы признак "Зарплата" с диапазоном значений до 100 тысяч не вносил больший вклад в расстояние, чем "Возраст" со значениями до 100.
* веса соседей (соседи тестового примера могут входить с разными весами, например, чем дальше пример, тем с меньшим коэффициентом учитывается его "голос").

### Класс KNeighborsClassifier в Scikit-learn

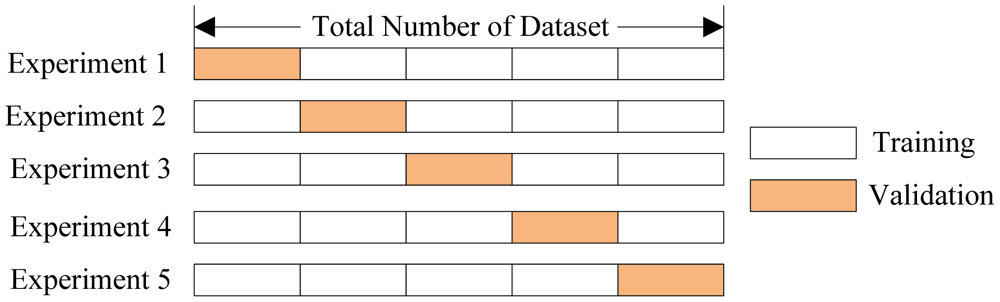
Основные параметры класса sklearn.neighbors.KNeighborsClassifier:

* weights: "uniform" (все веса равны), "distance" (вес обратно пропорционален расстоянию до тестового примера) или другая определенная пользователем функция
* algorithm (опционально): "brute", "ball\_tree", "KD\_tree", или "auto". В первом случае ближайшие соседи для каждого тестового примера считаются перебором обучающей выборки. Во втором и третьем - расстояние между примерами хранятся в дереве, что ускоряет нахождение ближайших соседей. В случае указания параметра "auto" подходящий способ нахождения соседей будет выбран автоматически на основе обучающей выборки.
* leaf\_size (опционально): порог переключения на полный перебор в случае выбора BallTree или KDTree для нахождения соседей
* metric: "minkowski", "manhattan", "euclidean", "chebyshev" и другие

# Выбор параметров модели и кросс-валидация

Главная задача обучаемых алгоритмов – их способность обобщаться, то есть хорошо работать на новых данных. Поскольку на новых данных мы сразу не можем проверить качество построенной модели (нам ведь надо для них сделать прогноз, то есть истинных значений целевого признака мы для них не знаем), то надо пожертвовать небольшой порцией данных, чтоб на ней проверить качество модели.

Чаще всего это делается одним из 2 способов:

* отложенная выборка (held-out/hold-out set). При таком подходе мы оставляем какую-то долю обучающей выборки (как правило от 20% до 40%), обучаем модель на остальных данных (60-80% исходной выборки) и считаем некоторую метрику качества модели (например, самое простое – долю правильных ответов в задаче классификации) на отложенной выборке.
* кросс-валидация (cross-validation, на русский еще переводят как скользящий или перекрестный контроль). Тут самый частый случай – K-fold кросс-валидация.

Тут модель обучается K раз на разных (K-1) подвыборках исходной выборки (белый цвет), а проверяется на одной подвыборке (каждый раз на разной, оранжевый цвет). Получаются 𝐾 оценок качества модели, которые обычно усредняются, выдавая среднюю оценку качества классификации/регресии на кросс-валидации.

Кросс-валидация дает лучшую по сравнению отложенной выборкой оценку качества модели на новых данных. Но кросс-валидация вычислительно дорогостоящая, если данных много.

Кросс-валидация – очень важная техника в машинном обучении (применяемая также в статистике и эконометрике), с ее помощью выбираются гиперпараметры моделей, сравниваются модели между собой, оценивается полезность новых признаков в задаче и т.д. Более подробно можно почитать, например, тут <https://sebastianraschka.com/blog/2016/model-evaluation-selection-part1.html> у Sebastian Raschka или в любом классическом учебнике по машинному (статистическому) обучению.

# Примеры применения

## Деревья решений и метод ближайших соседей в задаче прогнозирования оттока клиентов телеком-оператора.

Считаем данные в DataFrame и проведем предобработку. Штаты пока сохраним в отдельный объект Series, но удалим из датафрейма. Первую модель будем обучать без штатов, потом посмотрим, помогают ли они.

df = pd.read\_csv('../../data/telecom\_churn.csv')

df['International plan'] = pd.factorize(df['International plan'])[0]

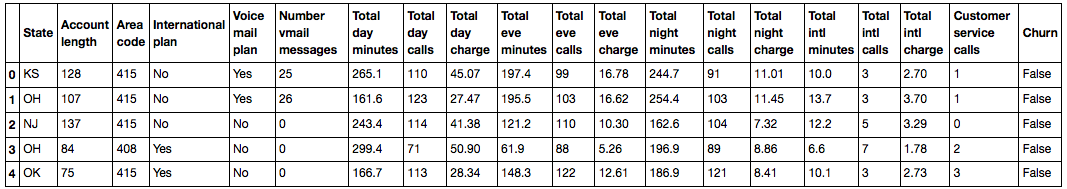
df['Voice mail plan'] = pd.factorize(df['Voice mail plan'])[0]

df['Churn'] = df['Churn'].astype('int')

states = df['State']

y = df['Churn']

df.drop(['State', 'Churn'], axis=1, inplace=True)



from sklearn.model\_selection import StratifiedKFold, train\_test\_split

Выделим 70% выборки (X\_train, y\_train) под обучение и 30% будут отложенной выборкой (X\_holdout, y\_holdout). отложенная выборка никак не будет участвовать в настройке параметров моделей, на ней мы в конце, после этой настройки, оценим качество полученной модели.

X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split(

df.values, y, test\_size=0.3, random\_state=17

)

Обучим 2 модели – дерево решений и kNN, пока не знаем, какие параметры хороши, поэтому наугад: глубину дерева берем 5, число ближайших соседей – 10.

tree = DecisionTreeClassifier(max\_depth=5, random\_state=17)

knn = KNeighborsClassifier(n\_neighbors=10)

tree.fit(X\_train, y\_train)

knn.fit(X\_train, y\_train)

Качество прогнозов будем проверять с помощью простой метрики – доли правильных ответов.

from sklearn.metrics import accuracy\_score

Сделаем прогнозы для отложенной выборки. Видим, что метод ближайших соседей справился намного лучше. Но это мы пока выбирали параметры наугад.

tree\_pred = tree.predict(X\_holdout)

accuracy\_score(y\_holdout, tree\_pred) # 0.94

knn\_pred = knn.predict(X\_holdout)

accuracy\_score(y\_holdout, knn\_pred) # 0.88

Теперь настроим параметры дерева на кросс-валидации. Настраивать будем максимальную глубину и максимальное используемое на каждом разбиении число признаков. Суть того, как работает GridSearchCV: для каждой уникальной пары значений параметров max\_depth и max\_features будет проведена 5-кратная кросс-валидация и выберется лучшее сочетание параметров.

from sklearn.model\_selection import GridSearchCV, cross\_val\_score

tree\_params = {'max\_depth': range(1,11),

'max\_features': range(4,19)}

tree\_grid = GridSearchCV(tree, tree\_params,

cv=5, n\_jobs=-1,

verbose=True)

tree\_grid.fit(X\_train, y\_train)

Лучшее сочетание параметров и соответствующая средняя доля правильных ответов на кросс-валидации:

tree\_grid.best\_params\_

{'max\_depth': 6, 'max\_features': 17}

tree\_grid.best\_score\_

0.94256322331761677

accuracy\_score(y\_holdout, tree\_grid.predict(X\_holdout))

0.94599999999999995

Теперь попробуем настроить число соседей в алгоритме kNN.

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

knn\_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n\_jobs=-1))])

knn\_params = {'knn\_\_n\_neighbors': range(1, 10)}

knn\_grid = GridSearchCV(knn\_pipe, knn\_params,

cv=5, n\_jobs=-1,

verbose=True)

knn\_grid.fit(X\_train, y\_train)

knn\_grid.best\_params\_, knn\_grid.best\_score\_

({'knn\_\_n\_neighbors': 7}, 0.88598371195885128)

accuracy\_score(y\_holdout, knn\_grid.predict(X\_holdout))

0.89000000000000001

В этом примере дерево показало себя лучше, чем метод ближайших соседей: 94.2% правильных ответов на кросс-валидации и 94.6% на отложенной выборке против 88.6% / 89% для kNN. Более того, в данной задаче дерево проявляет себя очень хорошо, и даже случайный лес (который пока представляем просто как кучу деревьев, которые вместе работают почему-то намного лучше, чем одно дерево) в этом примере показывает долю правильных ответов не намного выше (95.1% на кросс-валидации и 95.3% –на отложенной выборке), а обучается намного дольше.

from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n\_estimators=100, n\_jobs=-1, random\_state=17)

print(np.mean(cross\_val\_score(forest, X\_train, y\_train, cv=5))) # 0.949

forest\_params = {'max\_depth': range(1,11),

'max\_features': range(4,19)}

forest\_grid = GridSearchCV(forest, forest\_params,

cv=5, n\_jobs=-1,

verbose=True)

forest\_grid.fit(X\_train, y\_train)

forest\_grid.best\_params\_, forest\_grid.best\_score\_ # ({'max\_depth': 9, 'max\_features': 6}, 0.951)

accuracy\_score(y\_holdout, forest\_grid.predict(X\_holdout)) # 0.953

Нарисуем получившееся дерево. Из-за того, что оно не совсем игрушечное (максимальная глубина – 6), картинка получается уже не маленькой, но по дереву можно "прогуляться", если отдельно открыть рисунок.

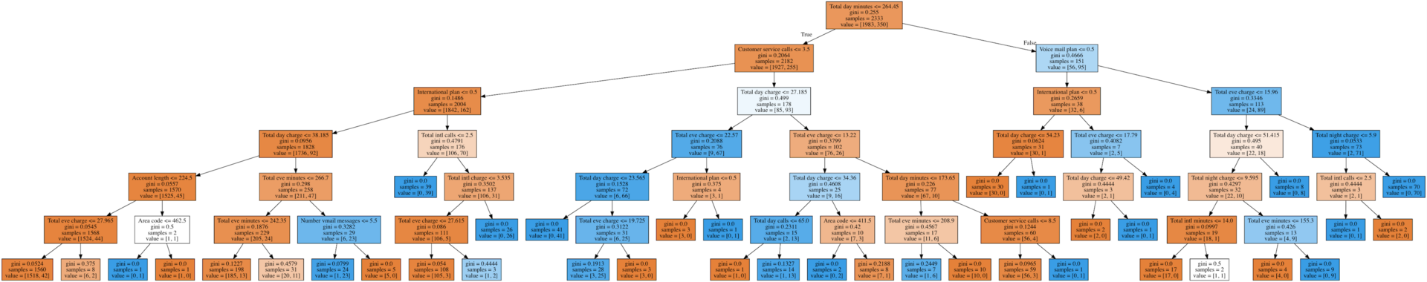
dot\_data = StringIO()

export\_graphviz(

tree\_grid.best\_estimator\_, feature\_names=df.columns, out\_file=dot\_data, filled=True

)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Image(value=graph.create\_png())

**Сложный случай для деревьев решений**

В продолжение обсуждения плюсов и минусов обсуждаемых методов приведем очень простой пример задачи классификации, с которым дерево справляется, но делает все как-то "сложнее", чем хотелось бы. Создадим множество точек на плоскости (2 признака), каждая точка будет относиться к одному из классов (+1, красные, или -1 – желтые). Если смотреть на это как на задачу классификации, то вроде все очень просто – классы разделяются прямой.

**Код для генерации данных и картинки**

def form\_linearly\_separable\_data(n=500, x1\_min=0, x1\_max=30, x2\_min=0, x2\_max=30):

data, target = [], []

for i in range(n):

x1, x2 = np.random.randint(x1\_min, x1\_max), np.random.randint(x2\_min, x2\_max)

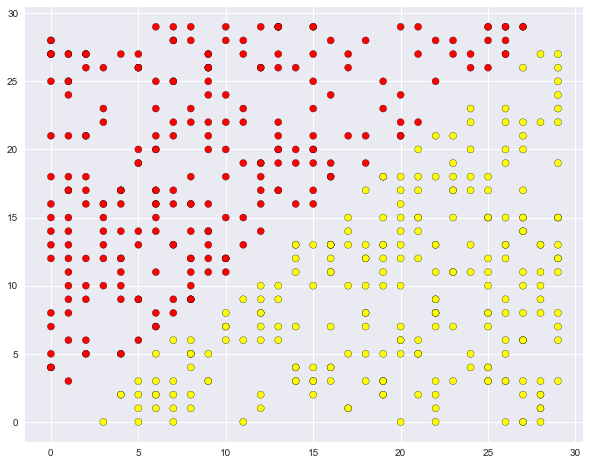
if np.abs(x1 - x2) > 0.5:

data.append([x1, x2])

target.append(np.sign(x1 - x2))

return np.array(data), np.array(target)

X, y = form\_linearly\_separable\_data()

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='autumn', edgecolors='black');

Однако дерево решений строит уж больно сложную границу и само по себе оказывается глубоким. Кроме того, представьте, как плохо дерево будет обобщаться на пространство вне представленного квадрата , обрамляющего обучающую выборку.

**Код для отрисовки разделяющей поверхности, которую строит дерево**

tree = DecisionTreeClassifier(random\_state=17).fit(X, y)

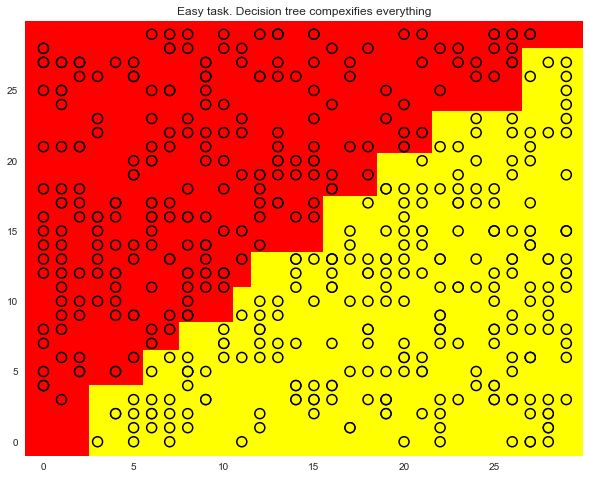
xx, yy = get\_grid(X, eps=.05)

predicted = tree.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.pcolormesh(xx, yy, predicted, cmap='autumn')

plt.scatter(X[:, 0], X[:, 1], c=y, s=100,

cmap='autumn', edgecolors='black', linewidth=1.5)

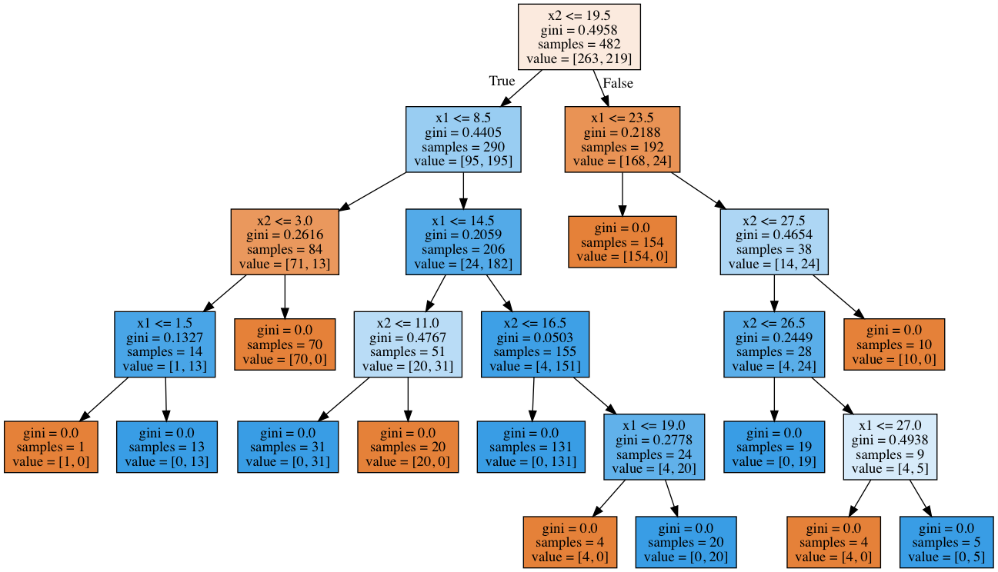
plt.title('Easy task. Decision tree compexifies everything');

Вот такая сложная конструкция, хотя решение (хорошая разделяющая поверхность) – это всего лишь прямая .

**Код для отрисовки дерева**

export\_graphviz(tree, feature\_names=['x1', 'x2'],

out\_file='../../img/deep\_toy\_tree.dot', filled=True)

!dot -Tpng '../../img/deep\_toy\_tree.dot' -o '../../img/deep\_toy\_tree.png'

Метод одного ближайшего соседа здесь справляется вроде лучше дерева, но все же не так хорошо, как линейный классификатор (это следующая тема).

**Код для отрисовки разделяющей поверхности, которую строит kNN**

knn = KNeighborsClassifier(n\_neighbors=1).fit(X, y)

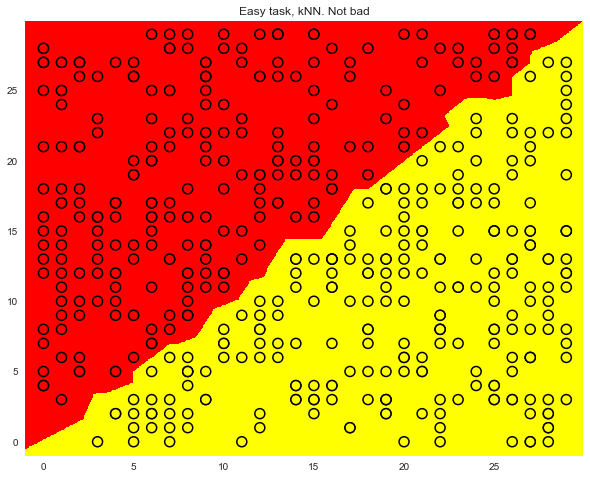
xx, yy = get\_grid(X, eps=.05)

predicted = knn.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.pcolormesh(xx, yy, predicted, cmap='autumn')

plt.scatter(X[:, 0], X[:, 1], c=y, s=100,

cmap='autumn', edgecolors='black', linewidth=1.5);

plt.title('Easy task, kNN. Not bad');

# Деревья решений и метод ближайших соседей в задаче распознавания рукописных цифр MNIST

Теперь посмотрим на описанные 2 алгоритма в реальной задаче. Используем "встроенные" в sklearn данные по рукописным цифрам. Эта задача будет примером, когда метод ближайших соседей работает на удивление хорошо.

Картинки здесь представляются матрицей 8 x 8 (интенсивности белого цвета для каждого пикселя). Далее эта матрица "разворачивается" в вектор длины 64, получается признаковое описание объекта.

Нарисуем несколько рукописных цифр, видим, что они угадываются.

**Загрузка данных и отрисовка нескольких цифр**

from sklearn.datasets import load\_digits

data = load\_digits()

X, y = data.data, data.target

Картинки здесь представляются матрицей 8 x 8 (интенсивности белого цвета для каждого пикселя). Далее эта матрица "разворачивается" в вектор длины 64, получается признаковое описание объекта.

X[0,:].reshape([8,8])

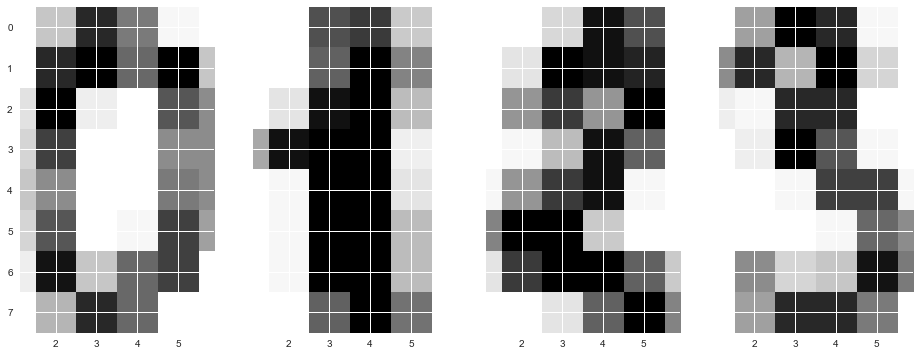
array([[ 0., 0., 5., 13., 9., 1., 0., 0.],  
[ 0., 0., 13., 15., 10., 15., 5., 0.],  
[ 0., 3., 15., 2., 0., 11., 8., 0.],  
[ 0., 4., 12., 0., 0., 8., 8., 0.],  
[ 0., 5., 8., 0., 0., 9., 8., 0.],  
[ 0., 4., 11., 0., 1., 12., 7., 0.],  
[ 0., 2., 14., 5., 10., 12., 0., 0.],  
[ 0., 0., 6., 13., 10., 0., 0., 0.]])

Нарисуем несколько рукописных цифр, видим, что они угадываются.

f, axes = plt.subplots(1, 4, sharey=True, figsize=(16,6))

for i in range(4):

axes[i].imshow(X[i,:].reshape([8,8]));

axes[i].imshow(X[i, :].reshape([8, 8]), cmap="Greys");

Посмотрим на соотношение классов в выборке, видим, что примерно поровну нулей, единиц, ..., девяток.

np.bincount(y)

out: array([178, 182, 177, 183, 181, 182, 181, 179, 174, 180], dtype=int64)

**Настройка DT и kNN на данных MNIST**

Выделим 70% выборки (X\_train, y\_train) под обучение и 30% будут отложенной выборкой (X\_holdout, y\_holdout). отложенная выборка никак не будет участвовать в настройке параметров моделей, на ней мы в конце, после этой настройки, оценим качество полученной модели.

X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split(X, y, test\_size=0.3,

random\_state=17)

Обучим дерево решений и kNN, опять параметры пока наугад берем.

tree = DecisionTreeClassifier(max\_depth=5, random\_state=17)

knn = KNeighborsClassifier(n\_neighbors=10)

tree.fit(X\_train, y\_train)

knn.fit(X\_train, y\_train)

Сделаем прогнозы для отложенной выборки. Видим, что метод ближайших соседей справился намного лучше. Но это мы пока выбирали параметры наугад.

tree\_pred = tree.predict(X\_holdout)

knn\_pred = knn.predict(X\_holdout)

accuracy\_score(y\_holdout, knn\_pred), accuracy\_score(y\_holdout, tree\_pred) # (0.97, 0.666)

Теперь так же, как раньше настроим параметры моделей на кросс-валидации, только учтем, что признаков сейчас больше, чем в прошлой задаче — 64.

tree\_params = {'max\_depth': [1, 2, 3, 5, 10, 20, 25, 30, 40, 50, 64],

'max\_features': [1, 2, 3, 5, 10, 20 ,30, 50, 64]}

tree\_grid = GridSearchCV(tree, tree\_params,

cv=5, n\_jobs=-1,

verbose=True)

tree\_grid.fit(X\_train, y\_train)

Лучшее сочетание параметров и соответствующая средняя доля правильных ответов на кросс-валидации:

tree\_grid.best\_params\_, tree\_grid.best\_score\_

#({'max\_depth': 10, 'max\_features': 50}, 0.8568203376968316)

accuracy\_score(y\_holdout, tree\_grid.predict(X\_holdout))

# 0.8425925925925926

Это уже не 66%, но и не 97%. Метод ближайших соседей на этом наборе данных работает лучше. В случае одного ближайшего соседа на кросс-валидации достигается почти 99% угадываний.

np.mean(cross\_val\_score(KNeighborsClassifier(n\_neighbors=1), X\_train, y\_train, cv=5))

# 0.9864858028204642

knn = KNeighborsClassifier(n\_neighbors=1).fit(X\_train, y\_train)

accuracy\_score(y\_holdout, knn.predict(X\_holdout))

# 0.9833333333333333

Обучим на этих же данных случайный лес, он на большинстве выборок работает лучше, чем метод ближайших соседей. Но сейчас у нас исключение.

np.mean(

cross\_val\_score(RandomForestClassifier(random\_state=17), X\_train, y\_train, cv=5))

# 0.9753462341111744

rf = RandomForestClassifier(random\_state=17, n\_jobs=-1).fit(X\_train, y\_train)

accuracy\_score(y\_holdout, rf.predict(X\_holdout))

# 0.9666666666666667

Вы будете правы, если возразите, что мы тут не настраивали параметры RandomForestClassifier, но даже с настройкой доля правильных ответов не достигает 98%, как для у метода одного ближайшего соседа.

Результаты эксперимента

*(Обозначения: CV и Holdout– средние доли правильных ответов модели на кросс-валидации и отложенной выборке соот-но. DT – дерево решений, kNN – метод ближайших соседей, RF – случайный лес)*

| **algo\eval** | **CV** | **Holdout** |
| --- | --- | --- |
| **DT** | 0.844 | 0.838 |
| **kNN** | 0.987 | 0.983 |
| **RF** | 0.935 | 0.941 |

**Вывод** по этому эксперименту (и общий совет): в начале проверяйте на своих данных простые модели – дерево решений и метод ближайших соседей (а в следующий раз сюда добавится логистическая регрессия), может оказаться, что уже они работают достаточно хорошо.

# Плюсы и минусы деревьев решений

**Плюсы:**

* Порождение четких правил классификации, понятных человеку, например, "если возраст < 25 и интерес к мотоциклам, то отказать в кредите". Это свойство называют интерпретируемостью модели;
* Деревья решений могут легко визуализироваться, то есть может "интерпретироваться" (строгого определения я не видел) как сама модель (дерево), так и прогноз для отдельного взятого тестового объекта (путь в дереве);
* Быстрые процессы обучения и прогнозирования;
* Малое число параметров модели;
* Поддержка и числовых, и категориальных признаков.

**Минусы:**

* У порождения четких правил классификации есть и другая сторона: деревья очень чувствительны к шумам во входных данных, вся модель может кардинально измениться, если немного изменится обучающая выборка (например, если убрать один из признаков или добавить несколько объектов), поэтому и правила классификации могут сильно изменяться, что ухудшает интерпретируемость модели;
* Разделяющая граница, построенная деревом решений, имеет свои ограничения (состоит из гиперплоскостей, перпендикулярных какой-то из координатной оси), и на практике дерево решений по качеству классификации уступает некоторым другим методам;
* Необходимость отсекать ветви дерева (pruning) или устанавливать минимальное число элементов в листьях дерева или максимальную глубину дерева для борьбы с переобучением. Впрочем, переобучение - проблема всех методов машинного обучения;
* Нестабильность. Небольшие изменения в данных могут существенно изменять построенное дерево решений. С этой проблемой борются с помощью ансамблей деревьев решений (рассмотрим далее);
* Проблема поиска оптимального дерева решений (минимального по размеру и способного без ошибок классифицировать выборку) NP-полна, поэтому на практике используются эвристики типа жадного поиска признака с максимальным приростом информации, которые не гарантируют нахождения глобально оптимального дерева;
* Сложно поддерживаются пропуски в данных. Friedman оценил, что на поддержку пропусков в данных ушло около 50% кода CART (классический алгоритм построения деревьев классификации и регрессии – Classification And Regression Trees, в sklearn реализована улучшенная версия именно этого алгоритма);
* Модель умеет только интерполировать, но не экстраполировать (это же верно и для леса и бустинга на деревьях). То есть дерево решений делает константный прогноз для объектов, находящихся в признаковом пространстве вне параллелепипеда, охватывающего все объекты обучающей выборки. В нашем примере с желтыми и синими шариками это значит, что модель дает одинаковый прогноз для всех шариков с координатой > 19 или < 0.

**Плюсы и минусы метода ближайших соседей**

Плюсы:

* Простая реализация
* Можно адаптировать под нужную задачу выбором метрики или ядра (в двух словах: ядро может задавать операцию сходства для сложных объектов типа графов, а сам подход kNN остается тем же);
* Неплохая интерпретация, можно объяснить, почему тестовый пример был классифицирован именно так. Хотя этот аргумент можно атаковать: если число соседей большое, то интерпретация ухудшается (условно: "мы не дали ему кредит, потому что он похож на 350 клиентов, из которых 70 – плохие, что на 12% больше, чем в среднем по выборке").

Минусы:

* Метод считается быстрым в сравнении, например, с композициями алгоритмов, но в реальных задачах, как правило, число соседей, используемых для классификации, будет большим (100-150), и в таком случае алгоритм будет работать не так быстро, как дерево решений;
* Если в наборе данных много признаков, то трудно подобрать подходящие веса и определить, какие признаки не важны для классификации/регрессии;
* Зависимость от выбранной метрики расстояния между примерами. Выбор по умолчанию евклидового расстояния чаще всего ничем не обоснован. Можно отыскать хорошее решение перебором параметров, но для большого набора данных это отнимает много времени;
* Нет теоретических оснований выбора определенного числа соседей - только перебор (впрочем, чаще всего это верно для всех гиперпараметров всех моделей). В случае малого числа соседей метод чувствителен к выбросам, то есть склонен переобучаться;
* Как правило, плохо работает, когда признаков много, из-за "прояклятия размерности". Про это хорошо рассказывает известный в ML-сообществе профессор Pedro Domingos – [тут](https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf) в популярной статье "A Few Useful Things to Know about Machine Learning", также "the curse of dimensionality" описывается в книге Deep Learning в [главе](http://www.deeplearningbook.org/contents/ml.html) "Machine Learning basics".

**Сложный случай для метода ближайших соседей**

Теперь рассмотрим еще один простой пример. В задаче классификации один из признаков будет просто пропорционален вектору ответов, но методу ближайших соседей это не поможет.

**Код для генерации шумных данных с паттерном**

def form\_noisy\_data(n\_obj=1000, n\_feat=100, random\_seed=17):

np.random.seed(random\_seed)

y = np.random.choice([-1, 1], size=n\_obj)

# первый признак пропорционален целевому

x1 = 0.3 \* y

# остальные признаки – шум

x\_other = np.random.random(size=[n\_obj, n\_feat - 1])

return np.hstack([x1.reshape([n\_obj, 1]), x\_other]), y

X, y = form\_noisy\_data()

Как обычно, будем смотреть на долю правильных ответов на кросс-валидации и на отложенной выборке. Построим кривые, отражающие зависимость этих величин от параметра n\_neighbors в методе ближайших соседей. Такие кривые называются кривыми валидации.

X\_train, X\_holdout, y\_train, y\_holdout = train\_test\_split(

X, y, test\_size=0.3, random\_state=17)

Видим, что метод ближайших соседей с евклидовой метрикой не справляется с задачей, даже если варьировать число ближайших соседей в широком диапазоне. Напротив, дерево решений легко "обнаруживает" скрытую зависимость в данных при любом ограничении на максимальную глубину.

**Построение кривых валидации для kNN**

from sklearn.model\_selection import cross\_val\_score

cv\_scores, holdout\_scores = [], []

n\_neighb = [1, 2, 3, 5] + list(range(50, 550, 50))

for k in n\_neighb:

knn = KNeighborsClassifier(n\_neighbors=k)

cv\_scores.append(np.mean(cross\_val\_score(knn, X\_train, y\_train, cv=5)))

knn.fit(X\_train, y\_train)

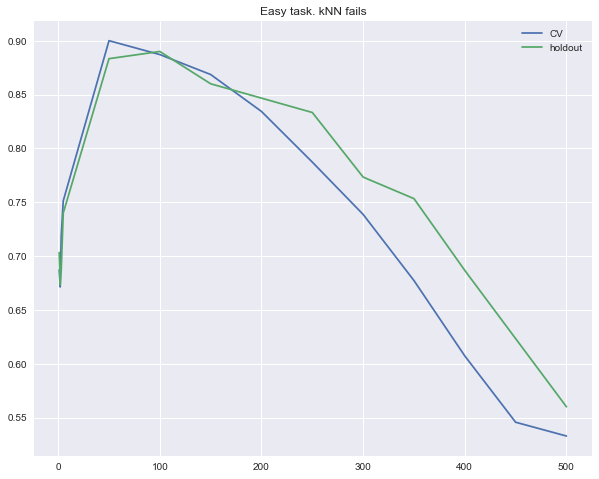
holdout\_scores.append(accuracy\_score(y\_holdout, knn.predict(X\_holdout)))

plt.plot(n\_neighb, cv\_scores, label='CV')

plt.plot(n\_neighb, holdout\_scores, label='holdout')

plt.title('Easy task. kNN fails')

plt.legend();



tree = DecisionTreeClassifier(random\_state=17, max\_depth=1)

tree\_cv\_score = np.mean(cross\_val\_score(tree, X\_train, y\_train, cv=5))

tree.fit(X\_train, y\_train)

tree\_holdout\_score = accuracy\_score(y\_holdout, tree.predict(X\_holdout))

print("Decision tree. CV: {}, holdout: {}".format(tree\_cv\_score, tree\_holdout\_score))

# Decision tree. CV: 1.0, holdout: 1.0

Итак, во втором примере дерево справилось с задачей идеально, а метод ближайших соседей испытал трудности. Впрочем, это минус скорее не метода, а используемой евклидовой метрики: в данном случае она не позволила выявить, что один признак намного лучше остальных.

# Домашнее задание № 3

В качестве закрепления материала предлагаем выполнить задание – разобраться с тем, как работает дерево решений, на игрушечном примере, затем обучить и настроить деревья в задаче классификации данных Adult репозитория UCI.