MLClass. "Прикладной анализ данных".

Курс "Инструментарий Data Science".

Преподаватель ФКН НИУ ВШЭ Кашницкий Юрий

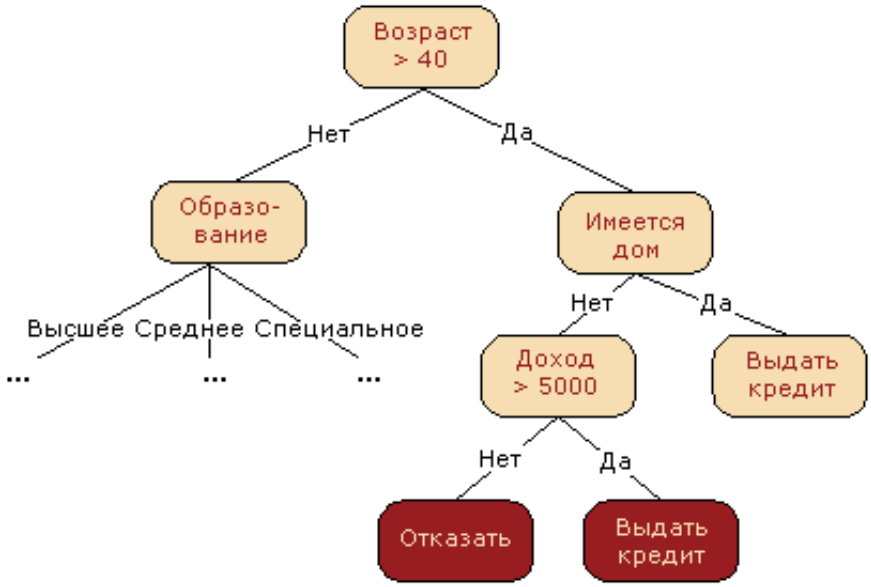
Дерево решений в задаче классификации

Что такое дерево решений

Начнем обзор методов классификации, конечно же, с самого популярного - 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. Высшая Школа Экономики 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 "НИУ ВШЭ 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 2 можно объяснить заемщику, почему ему было отказано в кредите. Например, 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, C4.5, рассматривается первым в списке 10 лучших алгоритмов интеллектуального анализа данных (Top 10 algorithms in data mining, 2007).

Как строить 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 6 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

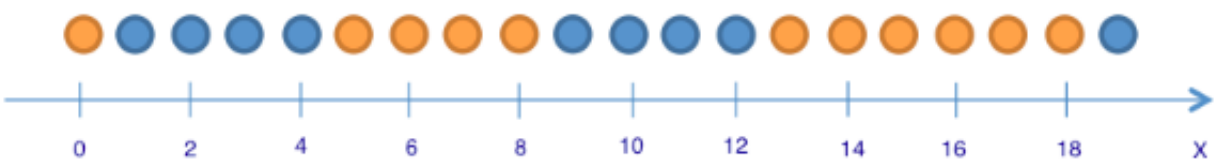
Энтропия Шеннона определяется как

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

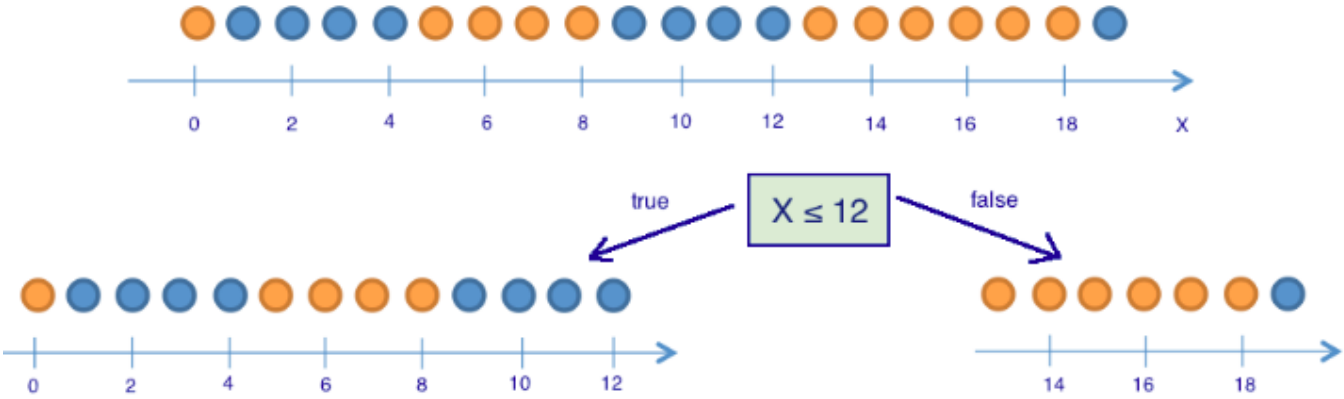
где N - число возможный реализаций, p_i - вероятности этих реализаций. Это очень важное понятие, используемое в физике, теории информации и других областях. Опуская предпосылки введения (комбинаторные и теоретико-информационные) этого понятия, отметим, что, интуитивно, энтропия соответствует степени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

Пример

Рассмотрим игрушечный пример, в котором 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 $p_1 = \frac{9}{20}$ будет синим и с вероятностью $p_2 = \frac{11}{20}$ - желтым. Значит, энтропия состояния $S_0 = -\frac{9}{20} log_2 \frac{9}{20} - \frac{11}{20} log_2 \frac{11}{20} \approx 1$. 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 .



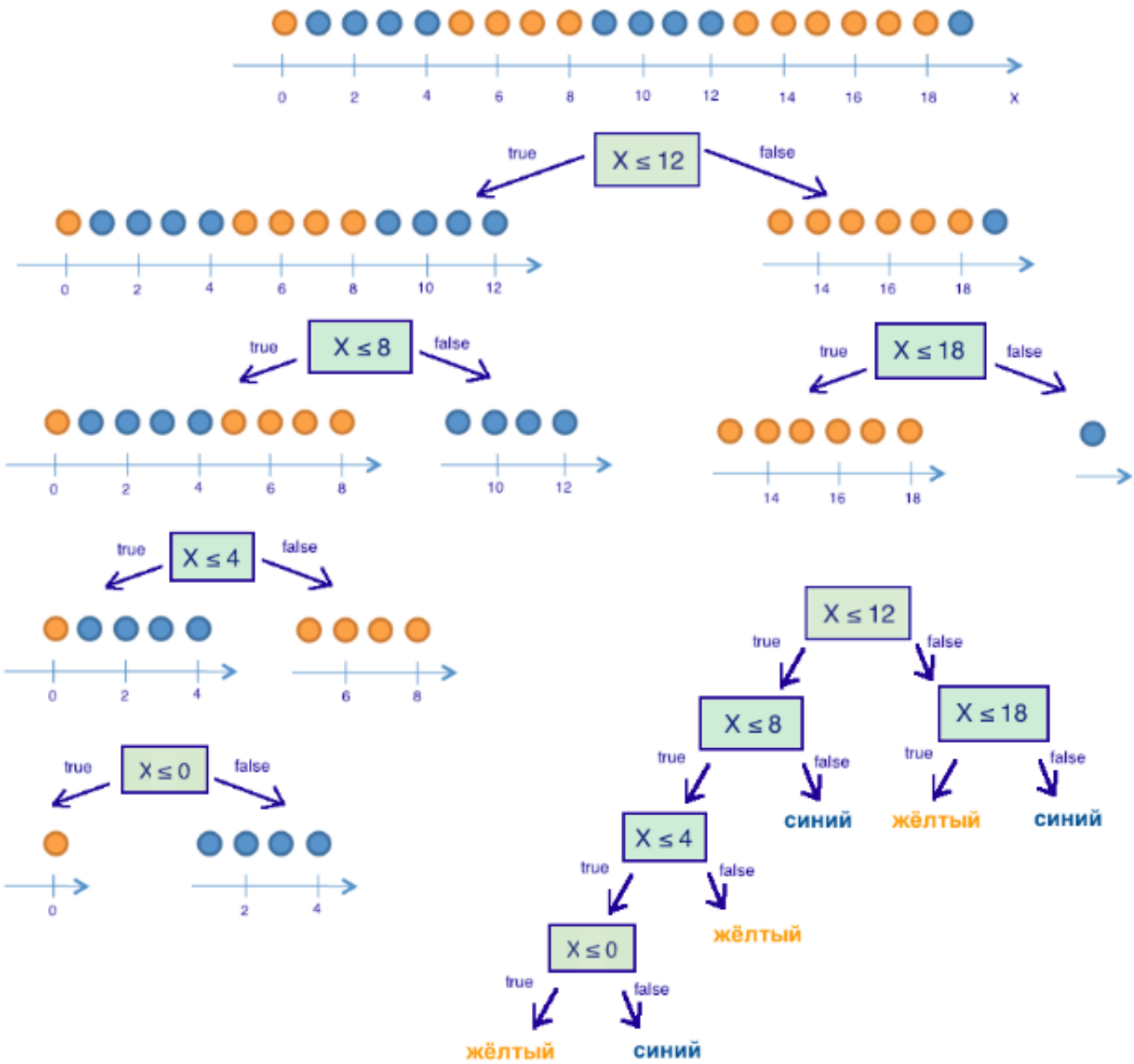
В левой группе оказалось 13 шаров, из которых 8 синих и 5 желтых. Энтропия этой группы равна $S_1 = -\frac{5}{13}log_2\frac{5}{13} - \frac{8}{13}log_2\frac{8}{13} \approx 0.96$. В правой группе оказалось 7 шаров, из которых 1 синий и 6 желтых. Энтропия правой группы равна $S_2 = -\frac{1}{7}log_2\frac{1}{7} - \frac{6}{7}log_2\frac{6}{7} \approx 0.6$. 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 разделении выборки по признаку Q (в нашем примере это признак "$x \leq 12$") определяется как

$$IG(Q) = S_O - \sum_{i=1}^{q}\frac{|N_i|}{N}S_i,$$

где q - число групп после разбиения, N_i - число элементов выборки, у которых признак Q имеет i-ое значение. В нашем случае после разделения получилось две группы ($q = 2$) - одна из 13 элементов ($N_1 = 13$), вторая - из 7 ($N_2 = 7$). Прирост информации получился

$$IG(" \; x \leq 12 \; ") = S_0 - \frac{13}{20}S_1 - \frac{7}{20}S_2 \approx 0.16.$$

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" с приростом информации, для левой - еще три. Очевидно, энтропия группы с шариками одного цвета равно 0 ($log_2 1 = 0$), что соответствует представлению, что группа шариков одного цвета - упорядоченная. В итоге 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 x), 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.

Пример

Рассмотрим пример на применение дерева решений из библиотеки Scikit-learn для синтетических данных.

In [1]:

```
%matplotlib inline
import numpy as np
import pylab as plt
plt.rcParams['figure.figsize'] = (10.0, 8.0) # чтоб рисунки побольше были
```

Сгенерируем данные. Два класса будут сгенерированы из двух нормальных распределений с разными средними.

In [27]:

```
# первый класс
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)]
```

In [1]:

```
print "Hello"
```

```
Hello
```

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

In [28]:

```
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))
```

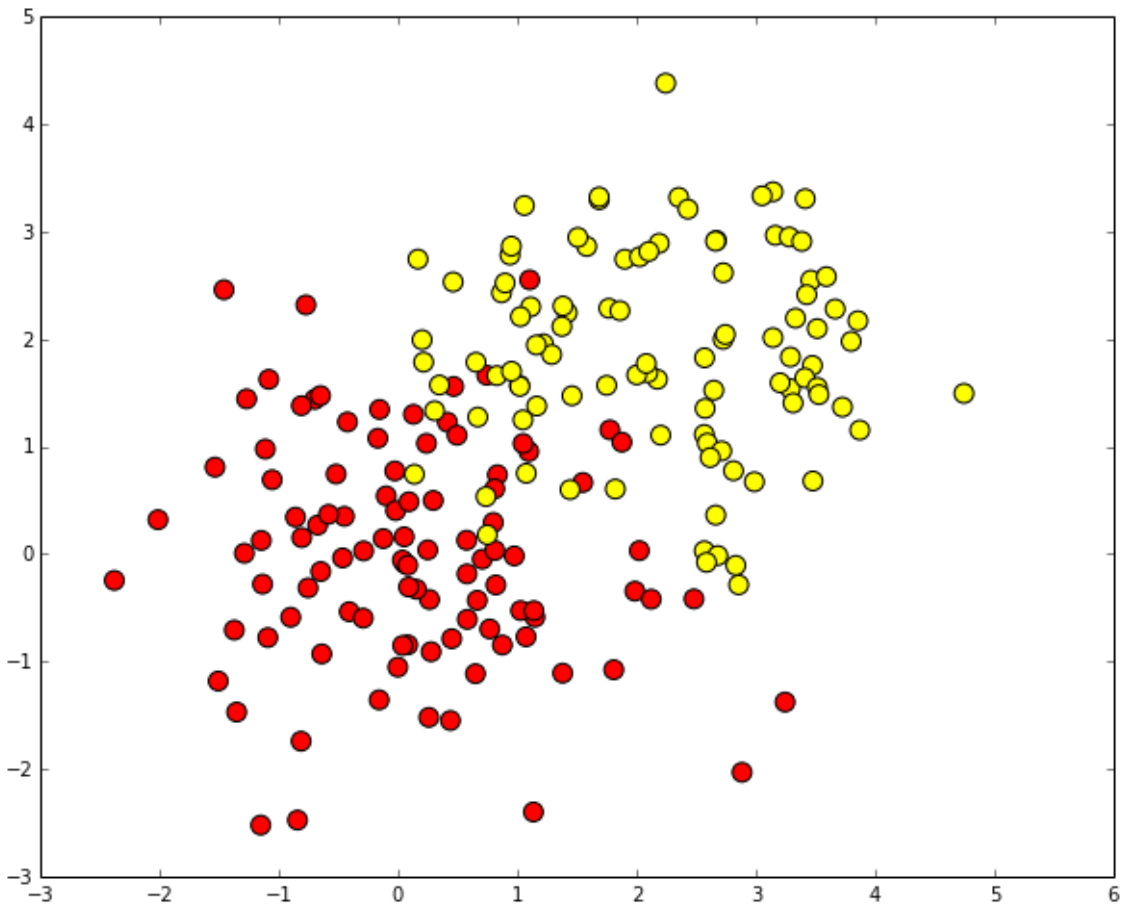
Отобразим данные

In [29]:

```
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, s=100, cma
p='autumn')
```

Out[29]:

```
<matplotlib.collections.PathCollection at 0x7fb22f3b5590>
```



Попробуем разделить эти два класса, обучив дерево решений. Визуализируем полученную границу разделения класссов.

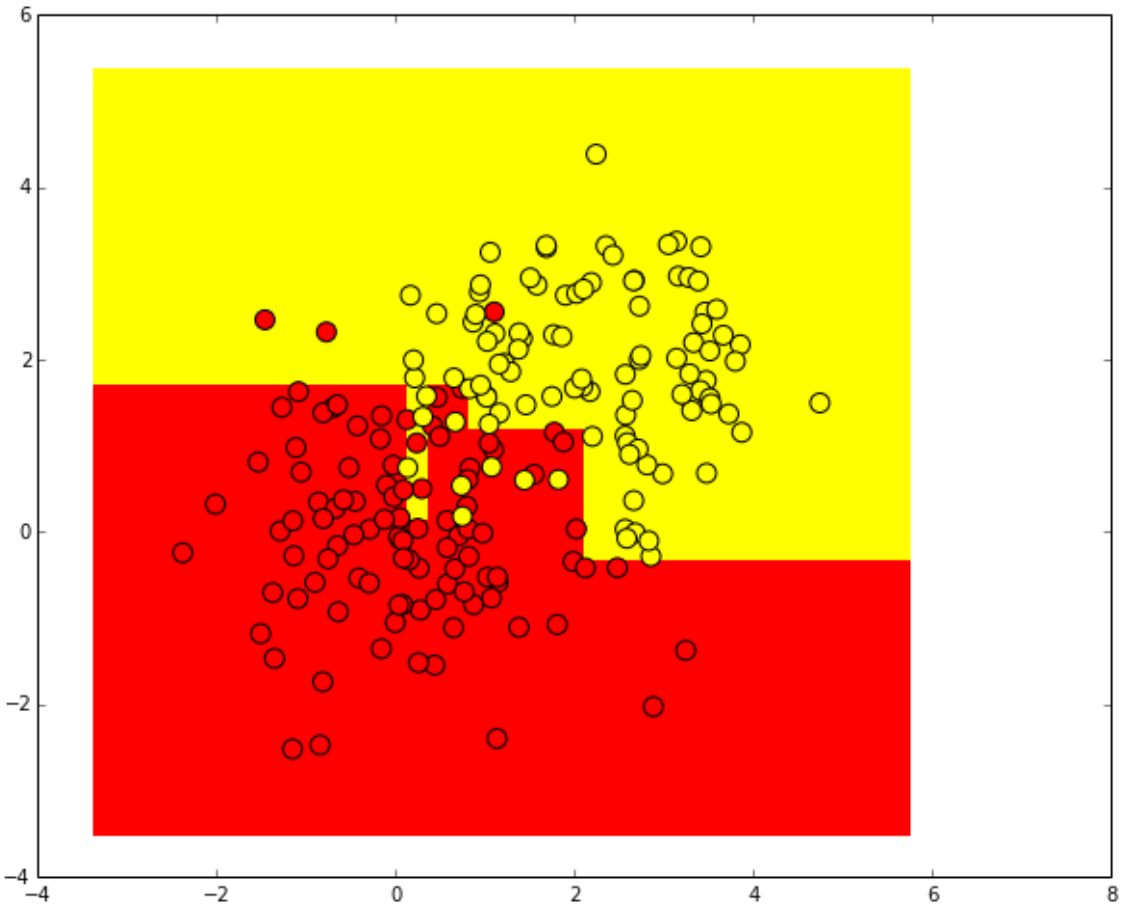
In [30]:

```
from sklearn.tree import DecisionTreeClassifier
# параметр min_samples_leaf указывает, при каком минимальном количестве
# элементов в узле он будет дальше разделяться
clf = DecisionTreeClassifier(min_samples_leaf=5)
clf.fit(train_data, train_labels)

xx, yy = get_grid(train_data)
predicted = clf.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, cma
p='autumn')
```

Out[30]:

```
<matplotlib.collections.PathCollection at 0x7fb22f285110>
```



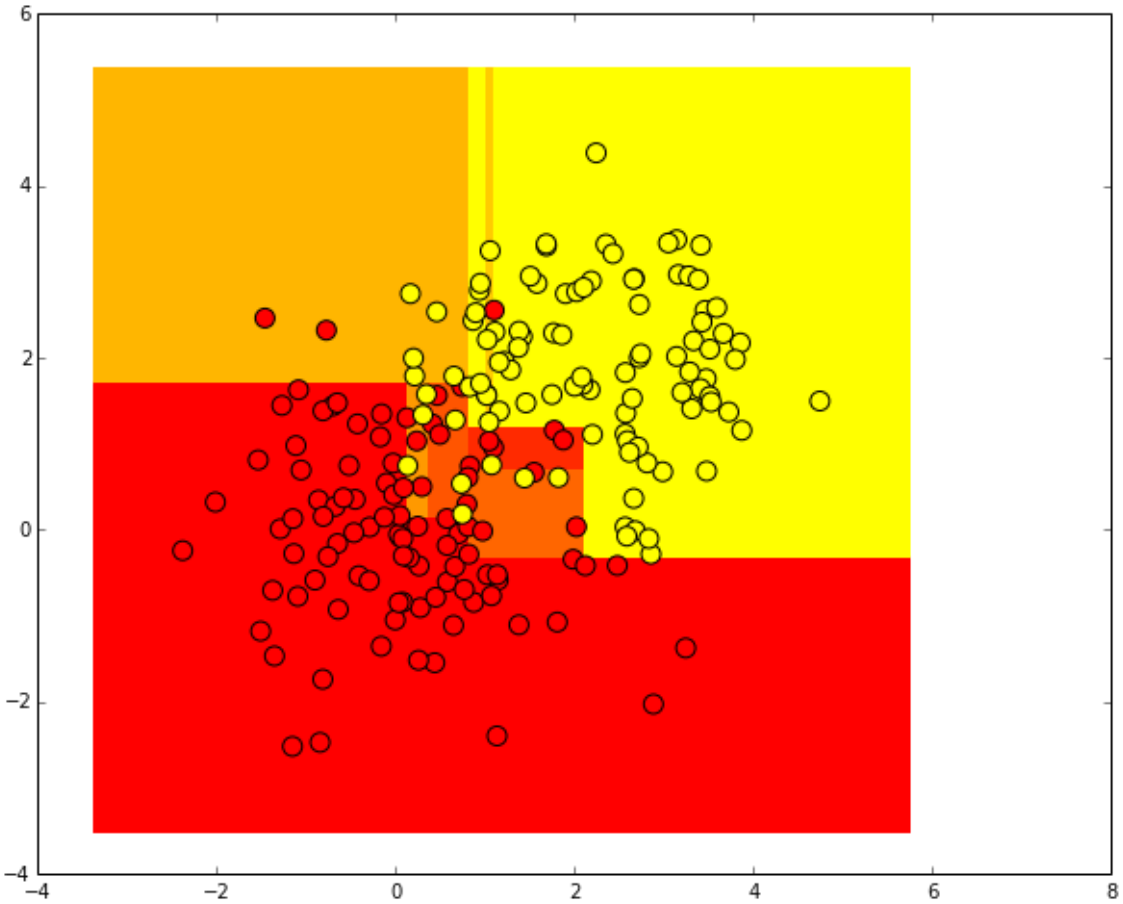
Проведем ту же процедуру, но теперь предскажем вещественные вероятности принадлежности первому классу. Сторого говоря, это не вероятности, а нормированные числа объектов разных классов из одного листа дерева. Например, если в каком-то листе оказалось 5 положительных объектов (с меткой 1) и 2 отрицательных (с меткой 0), то вместо того, чтобы предсказывать метку 1 для всех объектов, которые попадут на этот лист, будет предсказываться величина $\frac{5}{7}$.

In [31]:

```
predicted_proba = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1].re
shape(xx.shape)
plt.pcolormesh(xx, yy, predicted_proba, cmap='autumn')
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, s=100, cma
p='autumn')
```

Out[31]:

```
<matplotlib.collections.PathCollection at 0x7fb22f22ae10>
```



Сгенерируем случайно распределенные тестовые данные

In [32]:

```
test_data = np.random.normal(size=(100, 2), loc = 1)
predicted = clf.predict(test_data)
```

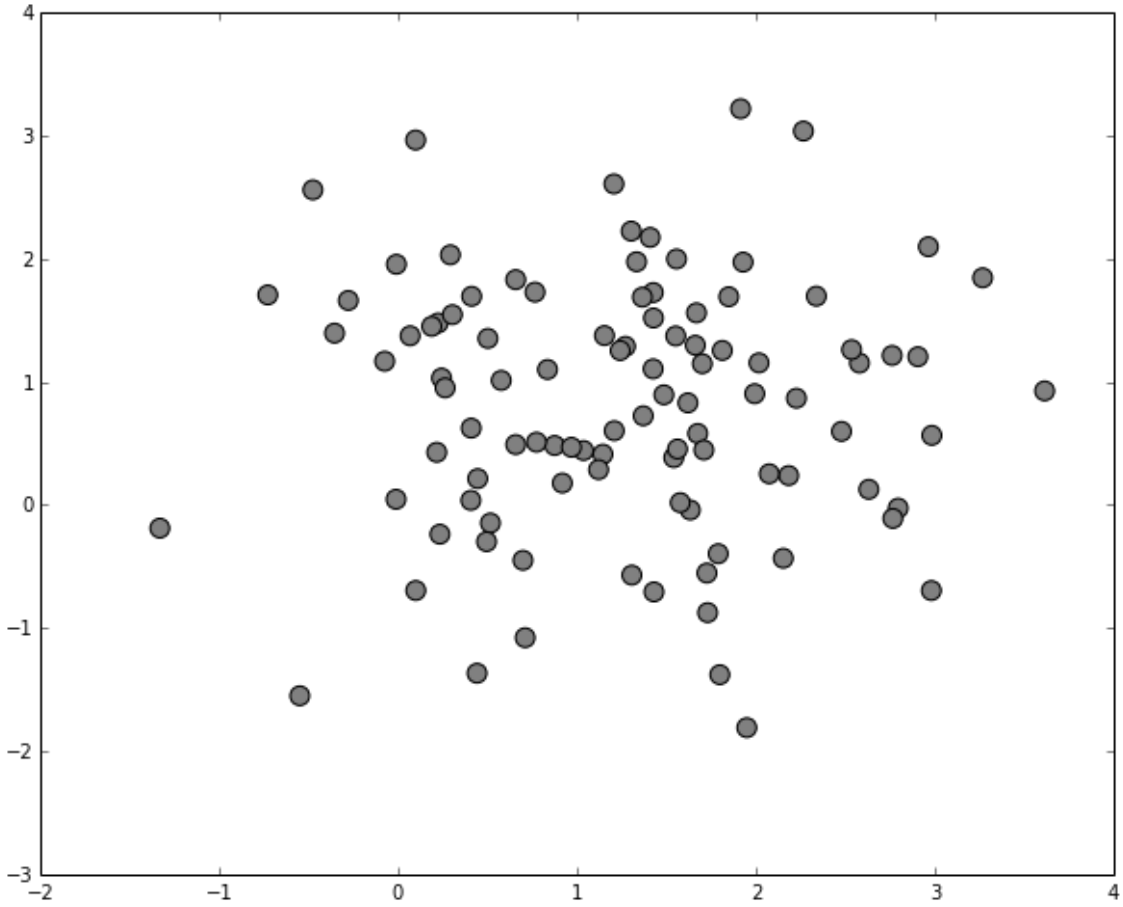
Отобразим их. Поскольку метки тестовых объектов неизвестны, нарисуем их серыми.

In [33]:

```
plt.scatter(test_data[:, 0], test_data[:, 1], c="gray", s=100)
```

Out[33]:

```
<matplotlib.collections.PathCollection at 0x7fb22f15d190>
```



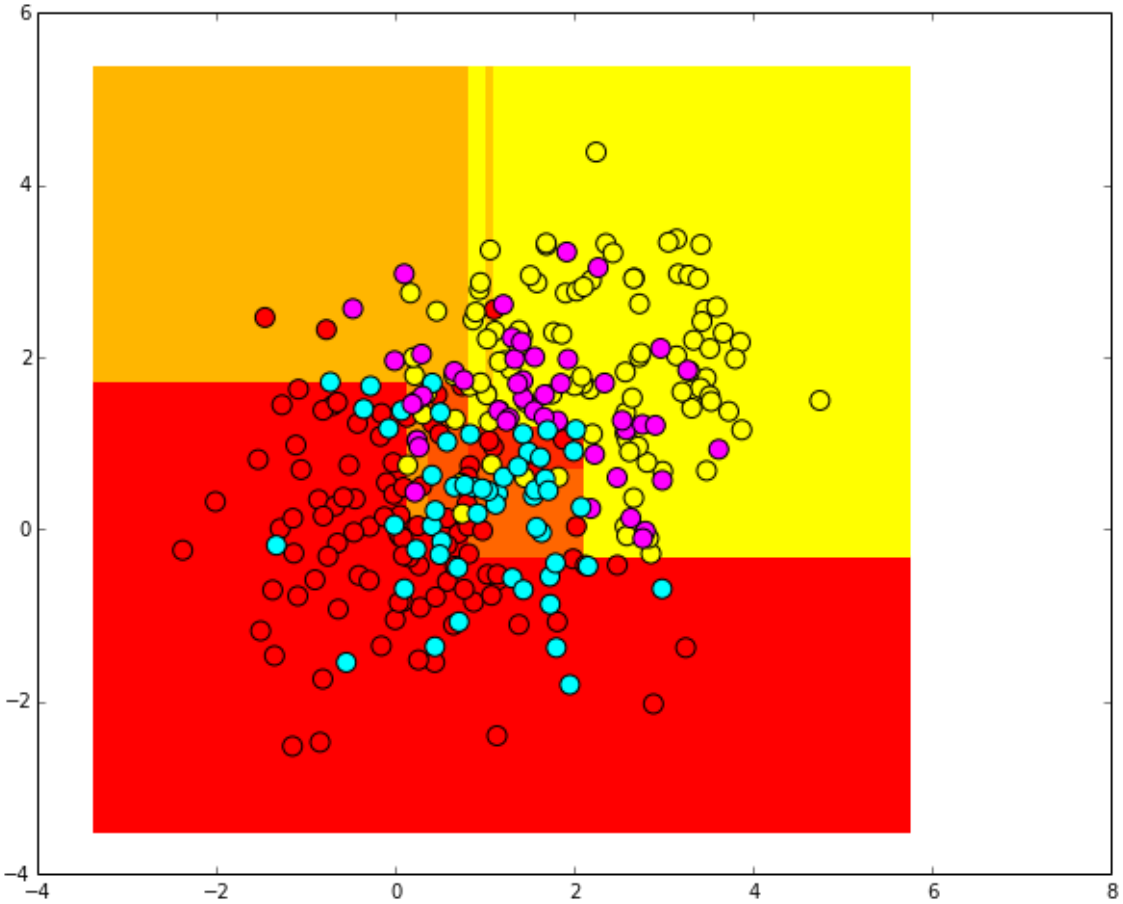
Посмотрим, как дерево решений классифицировало тестовые примеры. Для контраста используем другую цветовую гамму.

In [34]:

```
plt.pcolormesh(xx, yy, predicted_proba, cmap='autumn')
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, s=100, cma
p='autumn')
plt.scatter(test_data[:, 0], test_data[:, 1], c=predicted, s=100, cmap='coo
l')
```

Out[34]:

```
<matplotlib.collections.PathCollection at 0x7fb22f090a90>
```



Плюсы и минусы подхода

В этом разделе мы познакомились с самым простым и интуитивно понятным методом классификации, деревом решений, и посмотрели, как он используется в библиотеке Scikit-learn в задаче классификации. Пока мы не обсуждали, как оценивать качество классификации и как бороться с переобучением деревьев решений. Отметим плюсы и минусы данного подхода.

Плюсы:

- Порождение четких правил классификации, понятных человеку, например, "если возраст < 25 и интерес к мотоциклам, отказать в кредите"
- Деревья решений могут легко визуализироваться
- Относительно быстрые процессы обучения и классификации
- Малое число параметров модели
- Поддержка и числовых, и категориальных признаков

Минусы:

- Разделяющая граница, построенная деревом решений, имеет свои ограничения (состоит из гиперкубов), 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 NP-полна, поэтому на практике используются эвристики типа жадного поиска признака с максимальным приростом информации, которые не гарантируют нахождения глобально оптимального дерева
- Не поддерживаются пропуски в данных

In [5]:

```
print("Hello, world!")
import math
math.factorial(7)

help(math.factorial)

a = []

dir(a)
```

```
Hello, world!
Help on built-in function factorial in module math:

factorial(...)
    factorial(x) -> Integral

    Find x!. Raise a ValueError if x is negative or non-integra
l.
```

Out[5]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [ ]: