Практикум по программированию на языке Python

Занятие 7: Введение в инструменты для оптимизации и машинного обучения

Мурат Апишев (mel-lain@yandex.ru)

Москва, 2020

Инструменты специалиста по ML

Мы уже изучили

- основы языка Python
- введение в построение архитектуры кода
- базовые инструменты для манипуляции данными

Осталось понять, как

- использовать Python для математической оптимизации и статистики
- строить неглубокие модели для general ML с помощью готовых решений

Введение в библиотеку SciPy

- SciPy - это библиотека для научных вычислений, решения оптимизационных, численных и статистических задач
- Построена поверх NumPy
- Эффективно реализует многие методы линейной алгебры и оптимизации

Полезные ссылки:

- https://scipy-lectures.org/intro/scipy.html (https://scipy-lectures.org/intro/scipy.html)
- https://docs.scipy.org/doc/scipy/reference/ (https://docs.scipy.org/doc/scipy/reference/)

Состав SciPy

Что мы сегодня очень кратко рассмотрим:

- `linalg` - линейная алгебра
- `optimize` - методы оптимизации и решения уравнений
- `stats` - вероятностные распределения и статистика
- `sparse` - разреженные матрицы

Состав SciPy

Что есть ещё:

- `cluster` - алгоритмы кластеризации
- `constants` - физические и математические константы
- `fftpack` - инструменты для быстрого преобразования Фурье
- `integrate` - интегрирование и решение ОДУ
- `interpolate` - интерполяция и сплайны
- `io` - библиотека для работы с разными форматами входных и выходных данных
- `ndimage` - обработка N-мерных изображений
- `odr` - реализация Orthogonal distance regression
- `signal` - обработка сигналов
- `spatial` - пространственные структуры данных и алгоритмы
- `special` - набор специальных полезных функций
- `misc` - набор технических инструментов

Модуль scipy.linalg

In [47]:
```
import numpy as np
from scipy import linalg

A = np.array([[1,2], [3, 4]])
```

In [38]:
```
linalg.det(A)  # determinant
```

Out[38]:
```
-2.0
```

In [39]:
```
linalg.inv(A)  # inverted matrix
```

Out[39]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

In [76]:
```
linalg.norm(A)  # matrix and vector norms
```

Out[76]:
```
5.477225575051661
```

Модуль scipy.linalg

In [77]:
```
for e in linalg.eig(A):  # eigen values and vectors
    print(e)
```

```
[-0.37228132+0.j  5.37228132+0.j]
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

In [45]:
```
U, diags, V_T = linalg.svd(A)  # SVD
U @ np.diag(diags) @ V_T
```

Out[45]:
```
array([[1., 2.],
       [3., 4.]])
```

In [81]:
```
P, L, U = linalg.lu(A)  # pivoted LU decomposition
P @ L @ U
```

Out[81]:
```
array([[1., 2.],
       [3., 4.]])
```

Модуль scipy.optimize

Сгенерируем точки с помощью синусоиды с шумом и попробуем восстановить параметры функции

In [83]:
```
from scipy import optimize
```

In [82]:
```
x_data = np.linspace(-5, 5, num=50)
y_data = 2.9 * np.sin(1.5 * x_data) + np.random.normal(size=50)
```

In [84]:
```
def func_to_fit(x, a, b):
    return a * np.sin(b * x)
```

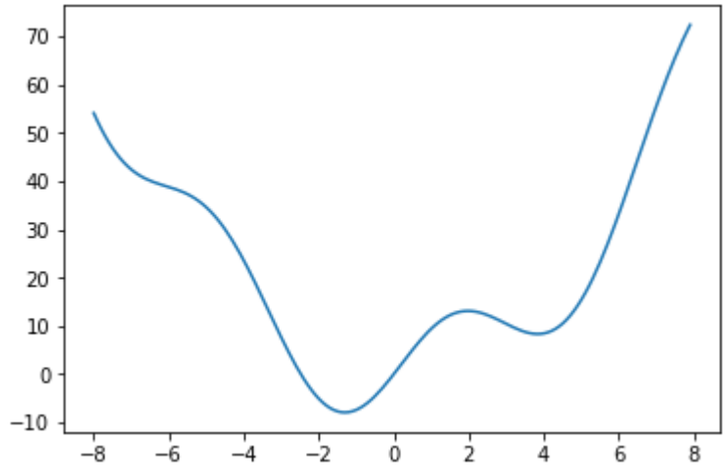
In [87]:
```
params, params_covariance = optimize.curve_fit(func_to_fit, x_data, y_data)
print(params)
```

```
[2.97536914 1.51777949]
```

Модуль scipy.optimize

Найдём минимум скалярной функции на отрезке (зависит от стартового приближения)

In [97]:
```
def f(x):
    return x**2 + 10 * np.sin(x)

x = np.arange(-8, 8, 0.1)
plt.plot(x, f(x))
plt.show()
```



In [107]:
```
result = optimize.minimize(f, x0=4, bounds=((-8, 8),))  # 'bounds' is optional
result.x
```

Out[107]:
```
array([3.83746783])
```

Модуль scipy.optimize

Найдём минимум скалярной функции на отрезке (зависит от стартового приближения)

In [108]:
```
result = optimize.minimize(f, x0=0, bounds=((-8, 8),))  # 'bounds' is optional
result.x
```

Out[108]:
```
array([-1.30643989])
```

In [123]:
```
result = optimize.basinhopping(f, x0=4, niter=1000)
result.x
```

Out[123]:
```
array([-1.30644001])
```

Функция `optimize.minimize` имеет параметр `method`, в котором можно определять конкрентный метод оптимизации, подходящий для решаемой задачи

Модуль scipy.optimize

Найдём нули скалярной функции

In [126]:
```
root = optimize.root(f, x0=1)
root.x
```

Out[126]:
```
array([0.])
```

In [127]:
```
root2 = optimize.root(f, x0=-2)
root2.x
```

Out[127]:
```
array([-2.47948183])
```

Находим только один корень, ближайший к стартовой точке, нужно идти по сетке для поиска нескольких

Модуль scipy.stats

По выборке восстанавливаем параметры известного распределения

In [131]:
```
from scipy import stats

samples = np.random.normal(size=1000, loc=9.8, scale=3.5)
loc, scale = stats.norm.fit(samples)

print(loc, scale)
```

```
9.741960556850568 3.4757631659935404
```

In [166]:
```
samples = np.random.beta(a=1, b=4.6, size=5000) + 10

a, b, loc, scale = stats.beta.fit(samples)
print(a, b, loc, scale)
```

```
1.0413361865370578 5.11430475356927 10.000013770776315 1.0415417093315682
```

Модуль scipy.stats

Посчитаем статистики выборки

In [169]:
```
samples = np.random.normal(size=1000, loc=0.0, scale=1.0)

np.mean(samples)
```

Out[169]: 0.02142806582351614

In [170]:
```
np.median(samples)
```

Out[170]: 0.04760864966009848

In [171]:
```
stats.scoreatpercentile(samples, 50)  # median
```

Out[171]: 0.04760864966009848

In [172]:
```
stats.scoreatpercentile(samples, 90)
```

Out[172]: 1.2622387134592612

Идея статистических тестов на пальцах

- Пусть у нас одна или несколько выборок
- Пусть эти выборки удовлетворяют определённым условиям (например, получены из нормального распределения)
- Тогда можно выдвинуть некоторые гипотезы относительно этих выборок и проверить их с помощью какого-то критерия
- Применение критерия подразумевает вычисление некоторой функции от выборки(-ок) и формулирование вывода на основании величины этого значения

Т-критерий

- Т-критерий (тест Стьдента) используется для проверки гипотез о
    - равенстве матожиданий двух выборок
    - равенстве матожидания одной выборки заданному значению

- В случае двух выборок, которые должны быть из распределения, близкого к нормальному, считается величина

$$t = \frac{m_1 - m_2}{\sqrt{\frac{d_1}{n_1} + \frac{d_2}{n_2}}},$$

где

- m_i - выборочные средние
- d_i - выборочные дисперсии
- n_i - размеры выборок

T-критерий

- Распределение значений t при выполнении условия нормальности будет стремиться к распределению Стьюдента

- Считаем от него значение p функции распределения для распределения Стьюдента $F_2(t) = P(x < t)$

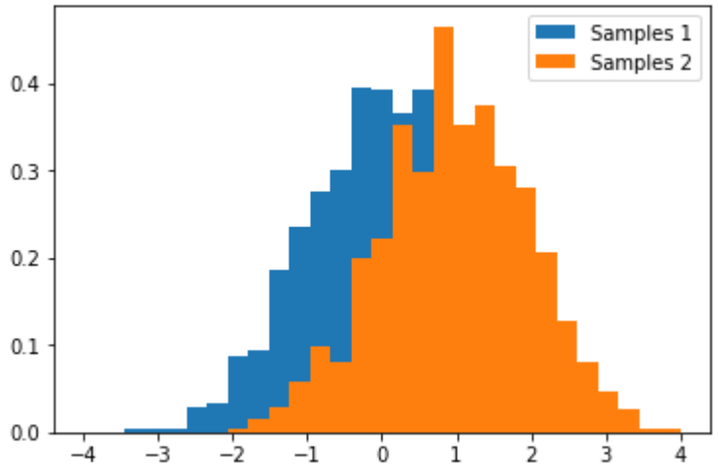
- Задаём *уровень значимости* α — это (небольшое) значение вероятности события, при котором событие уже можно считать неслучайным.

- Если окажется, что вероятность $1 - p$ получить значение T-критерия большее или равное t меньше α, то гипотеза равенства неверна

Модуль scipy.stats

Посчитаем статистический T-тест для случая выборок из нормального распределения

In [174]:
```
samples_1, samples_2 = gen_normal_samples()
```



In [178]:
```
t_score, p_value = stats.ttest_ind(samples_1, samples_2)
t_score, p_value
```

Out[178]:
```
(-22.557083985507102, 1.470706940823474e-100)
```

- Большое по модулю значений `t_score` означает большую разность между двумя порождающими случайными процессами
- Значение `p_value` характеризует вероятность того, что обе выборки были порождены процессами с одинаковым матожиданием

Модуль `scipy.sparse`

- Включает в себя набор различных типов разреженных матриц и инструментов для работы с ними и с другими разреженными структурами (например, графами)
- Мы рассмотрим основные форматы разреженных матриц, которые могут быть полезны в приложениях

- **Coordinate Format (COO)**
  - представляет собой 3 массива `row`, `col`, `data`
  - `row` содержит номера строк ненулевых элементов
  - `col` содержит номера столбцов ненулевых элементов
  - `data` содержит сами ненулевые элементы

In [183]:
```
from scipy.sparse import coo_matrix

S = np.array([[1, 0, 0], [4, 0, 6], [0, 8, 0]])

cm = coo_matrix(S)
cm.row, cm.col, cm.data
```

Out[183]:
```
(array([0, 1, 1, 2], dtype=int32),
 array([0, 0, 2, 1], dtype=int32),
 array([1, 4, 6, 8]))
```

Модуль scipy.sparse

- **Compressed Sparse Row Format (CSR)**
  - Представляет собой три массива: `indices`, `indptr`, `data`
  - `indices` содержит индексы столбцов ненулевых элементов
  - `data` содержит значения ненулевых элементов
  - `indptr` содержит старты строк в `indices` и `data` по следующим правилам:
    - длина `indptr` равна числу строк + 1, последний элемент равен числу ненулевых элементов
    - ненулевые значения i-й строки лежат в `data[indptr[i]: indptr[i + 1]]`
    - их индексы столбцов - в `indices[indptr[i]: indptr[i + 1]]`
    - элемент (i, j) доступ в `data[indptr[i]+k]`, где k - это позиция j в `indices[indptr[i]: indptr[i + 1]]`

In [191]:
```
from scipy.sparse import csr_matrix

S = np.array([[1, 0, 0], [4, 0, 6], [0, 8, 0]])

csrm = csr_matrix(S)
csrm.indices, csrm.indptr, csrm.data
```

Out[191]:
```
(array([0, 0, 2, 1], dtype=int32),
 array([0, 1, 3, 4], dtype=int32),
 array([1, 4, 6, 8], dtype=int64))
```

Модуль `scipy.sparse`

- **Compressed Sparse Column Format (CSC)**
  - Представляет собой три массива: `indices`, `indptr`, `data`
  - `indices` содержит индексы строк ненулевых элементов
  - `data` содержит значения ненулевых элементов
  - `indptr` содержит старты столбцов в `indices` и `data` по следующим правилам:
    - длина `indptr` равна числу столбцов + 1, последний элемент равен числу ненулевых элементов
    - ненулевые значения i-го столбца лежат в `data[indptr[i]: indptr[i + 1]]`
    - их индексы строк - в `indices[indptr[i]: indptr[i + 1]]`
    - элемент (i, j) доступ в `data[indptr[j]+k]`, где k - это позиция `i` в `indices[indptr[j]: indptr[j + 1]]`

In [189]:
```
from scipy.sparse import csc_matrix

S = np.array([[1, 0, 0], [4, 0, 6], [0, 8, 0]])

cscm = csc_matrix(S)
cscm.indices, cscm.indptr, cscm.data
```

Out[189]:
```
(array([0, 1, 2, 1], dtype=int32),
 array([0, 2, 3, 4], dtype=int32),
 array([1, 4, 8, 6], dtype=int64))
```

Прежде, чем перейти к обучению ML-моделей, вспомним основы

- Какие есть постановки задач машинного обучения?
- Что такое обучение с учителем?
- Что такое обучение без учителя?
- Что такое минимизация эмпирического риска?
- Что такое переобучение, как с ним бороться?
- Что такое кроссвалидация? Какие бывают виды?

Библиотека sklearn

- Стандартная и основная библиотека для general ML в Python
- Реализует многие используемые алгоритмы и модели, иногда в качестве обёртки
- Содержит множество полезных утилит для работы с данными, подбора параметров модели и оценивания качества
- Имеет понятный унифицированный интерфейс
- Использует NumPy и SciPy

Напоминание: линейные модели

- Модель имеет вид $a(x, w) = \langle x, w \rangle$
- В задаче классификации ответ y принадлежит дискретному множеству
- В задаче регрессии $y \in \mathbb{R}$
- Вид модели определяет дифференцируемая функция потерь $\mathcal{L}(y, a(x, w))$
- Линейная регрессия: $\mathcal{L} = (y - a(x, w))^2$
- В классификации используются различные аппроксимации пороговой функции потерь:
  - Логарифмическая (лог-регрессия): $\mathcal{L} = \log(1 + \exp(-a\langle x, w \rangle y))$
  - Кусочно-линейная (линейный SVM): $\mathcal{L} = (1 - a\langle x, w \rangle y)_+$
  - Ещё несколько других

Напоминание: метрики качества

- Функционал качества ≠ метрика качества
- Функционал удобно дифференцировать, его вид определяет свойства модели
- Метрика - внешний объективный критерий качества решения задачи, от модели обычно не зависит
- Популярные метрики для задачи классификации:
    - accuracy
    - precision/recall
    - F1-measure
    - ROC AUC
    - AUC PR
- Популярные метрики для задачи регрессии:
    - MAE
    - MSE
    - RMSE

Пример запуска лог-регрессии из sklearn

In [1]:
```
import seaborn
from sklearn.linear_model import LogisticRegression

df = seaborn.load_dataset('iris')

X = df[df.columns[: -1]].values
y = df['species'].values

model = LogisticRegression(solver='liblinear', multi_class='ovr')
model.fit(X, y)

preds = model.predict(df[df.columns[: -1]].values)

print('Train accuracy: {}'.format(sum([i == j for i, j in zip(preds, y)]) / float(len(preds))))
```

```
Train accuracy: 0.96
```

Можем выбрать другую модель

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Lasso
from sklearn.linear_model import Lars
from sklearn.linear_model import SGDRegressor
from sklearn.svm import LinearSVR
```

Можем выбрать другие параметры модели

In [3]:
```
model = LogisticRegression(penalty='l2',       # or 'l1'
                           C=1.0,              # or any float > 0.0 (less == stronger penalty)
                           class_weight=None,  # or dict, or 'balanced'
                           solver='liblinear', # or 'lbfgs', 'sag', 'saga'
                           multi_class='ovr'   # or 'multinomial'
                           )
```

Можем разбить данные на обучение и тест

In [4]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

model = LogisticRegression(solver='liblinear', multi_class='ovr')
model.fit(X_train, y_train)
```

```
(135, 4)
(135,)
(15, 4)
(15,)
```

Out[4]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

Можем померять качество на тесте встроенными средствами

In [5]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

preds = model.predict(X_test)
print('Test Accuracy: {}'.format(accuracy_score(y_test, preds)))
print('Test f1-score: {}\n'.format(f1_score(y_test, preds, average='macro')))

probs = model.predict_proba(X_test)
print(probs[: 5])
```

```
Test Accuracy: 0.8666666666666667
Test f1-score: 0.7857142857142857

[[8.48574952e-01 1.51407793e-01 1.72549282e-05]
 [9.80086647e-04 3.93724396e-01 6.05295518e-01]
 [6.33415596e-03 2.51172974e-01 7.42492870e-01]
 [6.97600507e-04 1.16864628e-01 8.82437771e-01]
 [2.08252171e-02 6.58094588e-01 3.21080195e-01]]
```

В `sklearn.metrics` есть десятки разнообразных метрик для различных задач.

Можем подобрать гиперпараметры по сетке

In [10]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV

model = GridSearchCV(LogisticRegression(solver='liblinear', multi_class='ovr'),
                     cv=5,
                     param_grid={"C": [0.5, 1.0, 10.0, 100.0]},
                     scoring=make_scorer(f1_score, **{'average':'macro'}))

model.fit(X_train, y_train)
f1_score(y_test, model.predict(X_test), average='macro')
```

Out[10]:
```
0.9076923076923077
```

Можем воспользоваться кроссвалидацией

In [11]:
```
from sklearn.model_selection import cross_validate

cv_results = cross_validate(LogisticRegression(solver='liblinear', multi_class='ovr'),
                            X_train,
                            y_train,
                            cv=3,
                            return_train_score=True,
                            scoring=make_scorer(f1_score, **{'average':'macro'}))

print('Train scores: {}'.format(cv_results['train_score']))
print('Test scores: {}'.format(cv_results['test_score']))
```

```
Train scores: [0.9666574  0.95555556 0.96773354]
Test scores: [0.97840474 0.93521421 0.95396825]
```

Можем посмотреть на веса признаков

- Интерпретируемость доступная благодаря тому, что мы используем линейную модель
- Важно, чтобы признаки имели один масштаб значений

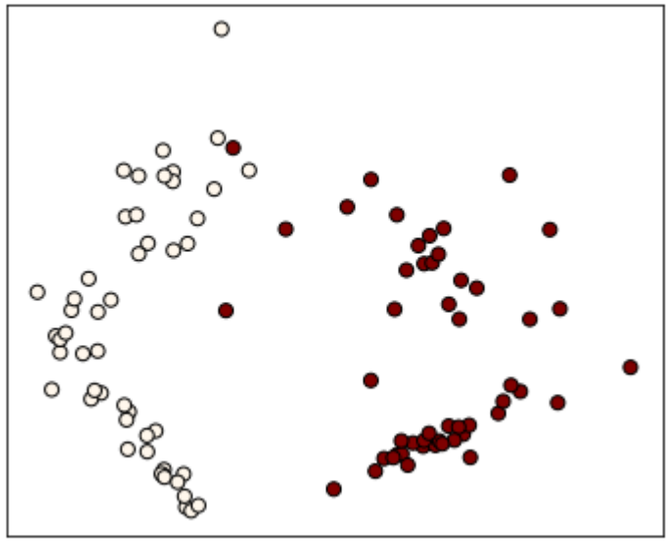
In [13]:
```
import pandas as pd
df = pd.DataFrame(data=model.best_estimator_.coef_,
                  columns=df.columns[: -1],
                  index=model.classes_)

df
```

Out[13]:

	sepal_length	sepal_width	petal_length	petal_width
setosa	0.936643	3.054329	-4.849467	-2.419512
versicolor	0.213950	-3.088029	1.144720	-2.913226
virginica	-3.836725	-3.116696	6.009686	9.748844

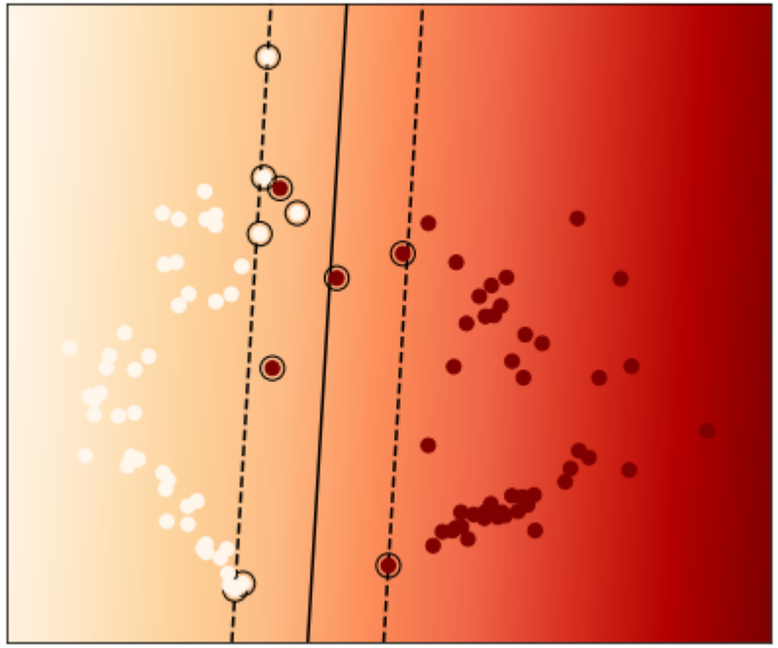
Можем сгенерировать данные

In [87]:
```
from sklearn.datasets import make_classification

X, y = make_classification(n_features=2, n_classes=2, n_samples=100, n_redundant=0, random_state=1)

plt.figure(1, figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.OrRd, s=50, edgecolors='k')
plt.xticks(())
plt.yticks(())
plt.show()
```



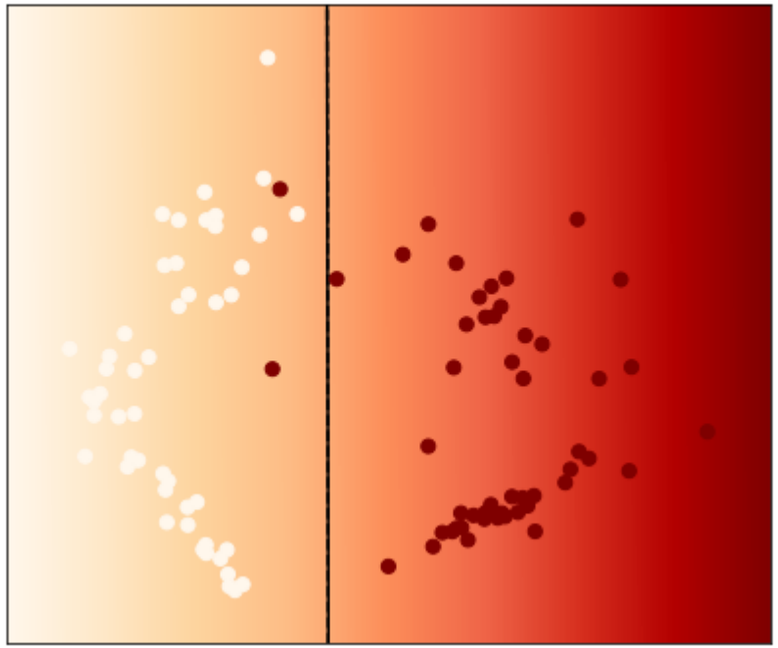
Обучим линейный SVM

In [81]:
```
import sklearn.svm as svm

model = svm.SVC(kernel='linear', C=1.0)
model.fit(X, y)
plot_results(model, X, y)
```



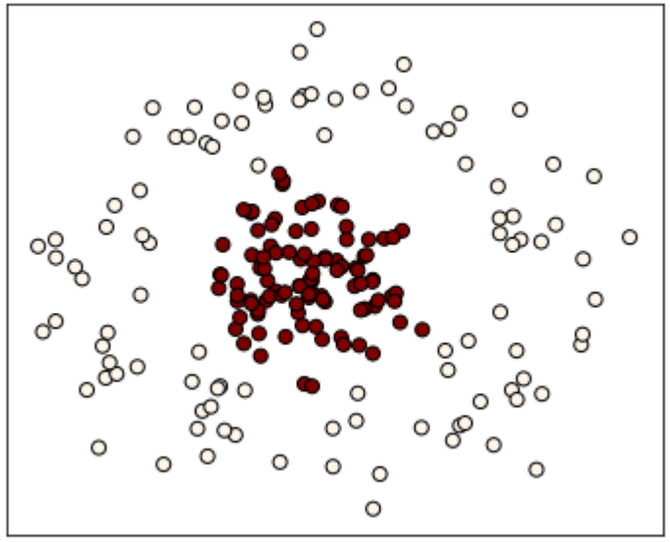
Лог-регрессия разделяет так

In [82]:
```
model = LogisticRegression(solver='liblinear')
model.fit(X, y)
plot_results(model, X, y, plot_logreg=True)
```



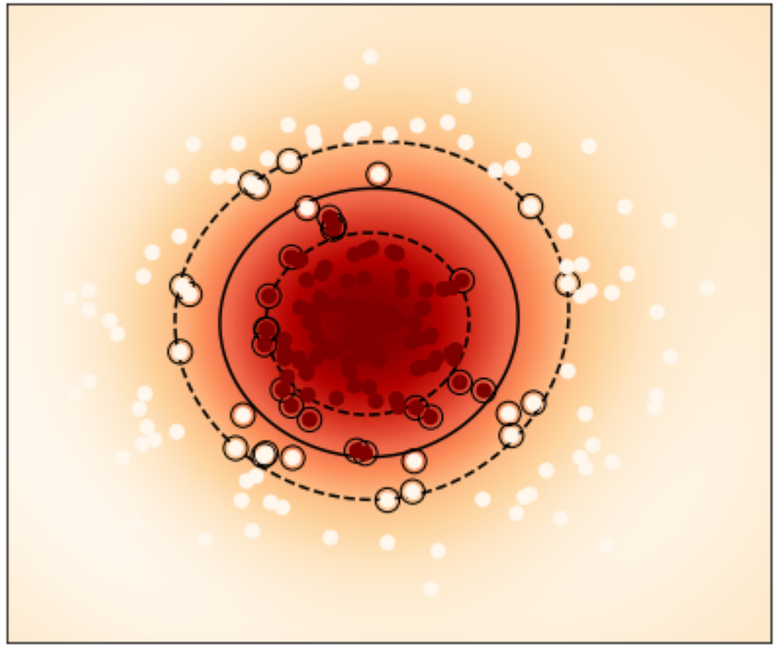
А что делать с такими данными?

In [88]:
```
from sklearn.datasets import make_circles

X, y = make_circles(noise=0.2, factor=0.2, random_state=1, n_samples=200)

plt.figure(1, figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.OrRd, s=50, edgecolors='k')
plt.xticks(())
plt.yticks(())
plt.show()
```



Вспомним про ядровой SVM

In [84]:
```
model = svm.SVC(kernel='rbf', C=1.0, gamma=1.0)
model.fit(X, y)
plot_results(model, X, y, level=0.98)
```



Данные для линейных моделей

- **Влияние выбросов**
  - на линейную регрессию - сильное
  - на логистическую регрессию - умеренное
  - на линейный SVM - сильное, если выброс стал опорным вектором
- **Несбалансированность данных нежелательна**
  - простейший способ борьбы -- сэмплировать объекты доминирующего класса
  - в разных задачах бывает возможным генерировать объекты, похожие на объекты меньшего класса
- **Признаки объектов для линейных моделей нужно нормировать**
  - веса становятся интерпретируемыми
  - градиентные методы оптимизации лучше сходятся

Напоминание: метрические модели

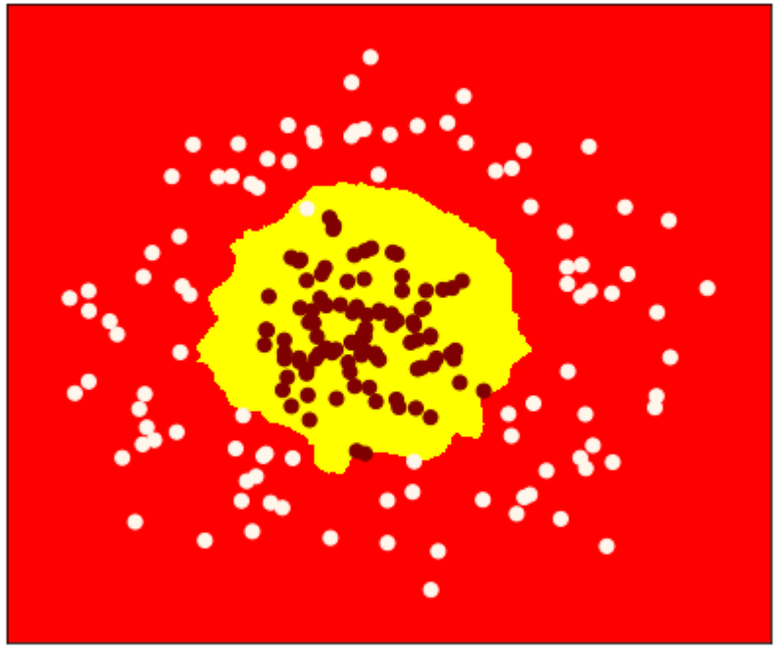
- Основываются на расстояних между объектами
- Основные функции расстояния:
  - Евклидово расстояние: $p(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$
  - Манхэттенское расстояние: $p(x, y) = \sum_i |x_i - y_i|$
  - Расстояние Чебышева: $p(x, y) = \max(|x_i - y_i|)$
  - Косинусное расстояние: $p(x, y) = \frac{\langle x, y \rangle}{||x|| * ||y||}$
- Могут использоваться как для обучения с учителем, так и без
- Самые известные представители: kNN и k-means

Напоминание: метрические модели

- Результаты работы сильно зависят от выбора функции близости и признакового описания
- Признаки нуждаются в нормализации:
  - можно делать *мини-макс нормализацию*: $\hat{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$
  - можно *Z-нормализацию*: $\hat{x} = \frac{x - \mathbb{E}(x)}{\sigma(x)}$
- kNN можно модифицировать для ускорения поиска ближайших соседей (возможно приближённого)
- возможны различные варианты учёта соседей и расстояний до них
- kNN может использоваться на практике для быстрой фильтрации кандидатов на более точную обработку

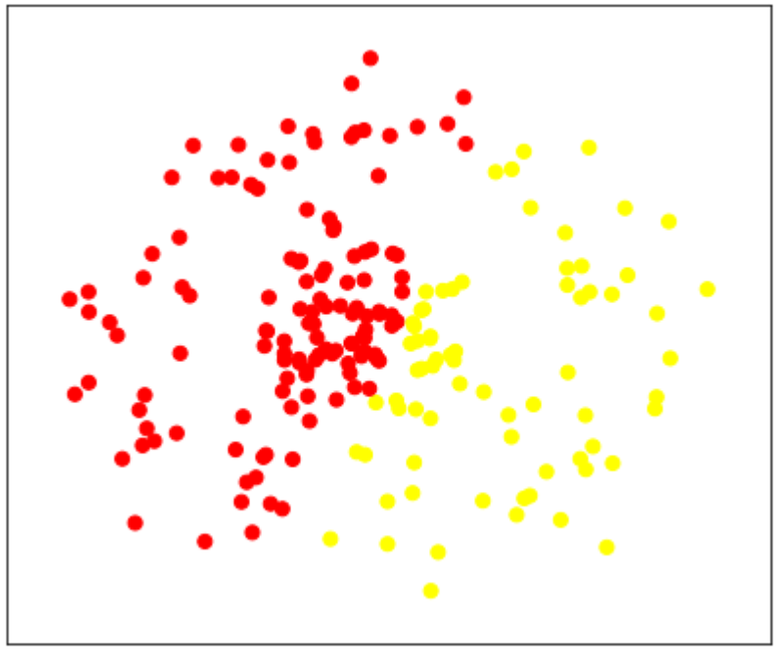
Применим kNN к последней выборке

In [92]:
```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X, y)
plot_results(model, X, y, cmap_plot=plt.cm.autumn)
```



А как поведёт себя кластеризация k-means?

In [107]:
```
from sklearn.cluster import k_means

model = k_means(X, n_clusters=2)
labels = model[1]
plot_results(model, X, labels, cmap_objects=plt.cm.autumn)
```



Частичное обучение

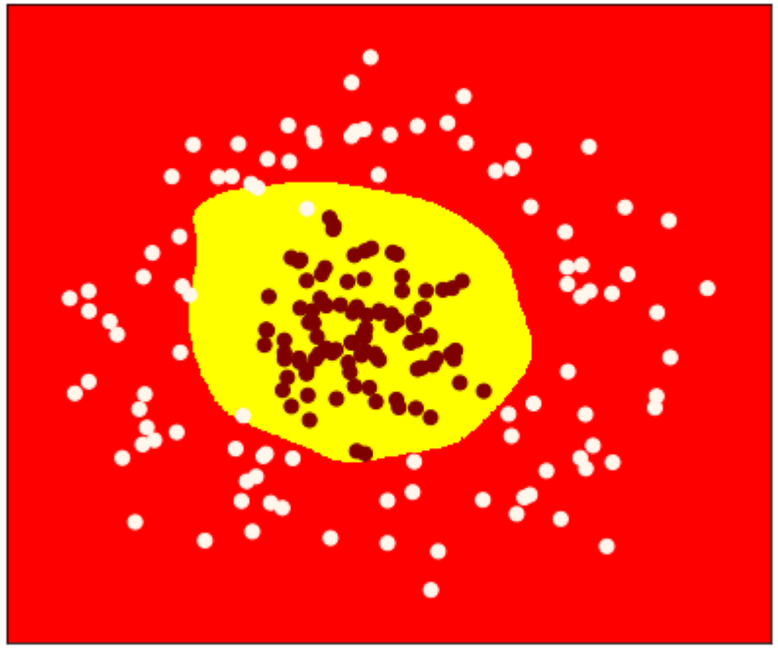
- Иногда у нас есть какое-то количество размеченных примеров
- Можно ими воспользоваться, для этого есть модуль `sklearn.semi_supervised`

Подготовим данные:

In [130]:
```
import numpy.random as rnd

indices = rnd.choice(range(len(y)), int(len(y) / 10), replace=False)
y_masked = np.full(len(y), -1)
y_masked[indices] = y[indices]
```

Частичное обучение

In [131]:
```
from sklearn.semi_supervised import LabelSpreading

model = LabelSpreading()
model.fit(X, y_masked)
plot_results(model, X, y, cmap_plot=plt.cm.autumn)
```



Напоминание: решающие деревья (логические модели)

- Decision Tree представляет собой бинарное дерево
    - в узлах - условия расшепления выборки
    - в листах - ответы модели
- Как правило, в жизни используются деревья с условием расщепления в виде банрного порога на значениях одного признака
- Деревья не используют расстояния между объектами, только взаимные положения на осях признаков
- Различные критерии оценки качества расщепления приводят к различным моделям
- Полноценное обучение дерева - вычислительно сложная задача, поэтому используются жадные алгоритмы

Напоминание: решающие деревья (логические модели)

Преимущества:

- простые и отлично интерпретируемые
- перебор по значению критерия расщепления обеспечивает встроенный отбор признаков
- устойчивыми к монотонным преобразованиям признаков и позволяет работать с признаками различной природы
- Деревья сильно переобучаются, поэтому в жизни используются редко
- Зато в составе композиций работаю очень круто

Недостатки:

- склонны к переобучению
- модель получается сложной при аппроксимации разделяющей поверхности, не параллельной признаковым осям координат
- добавление новых объектов требует переобучения дерева

Напоминание: композиции над решающими деревьями

- Сами по себе решающие деревья используются только в простейших задачах, требующих интерпретируемости

- А вот композиции деревьев оказались очень мощными моделями, способными решать сложные задачи

- Два основных типа композиций деревьев:

    - Случайный лес (Random Forest) - композиция сложных деревьев (Bagging + Random Subspace Method)

    - Градиентный бустинг - композиция простых деревьев, каждое следующее дерево обучается исправлять ошибки предшественников

- Отдельной разновидностью случайных лесов являются Extremely Randomized Trees - модификация, в которой выбор порога для каждого признака-кандидата в дереве производится случайно, а не перебором

Напоминание: Out-of-Bag Score для случайного леса

- Каждое дерево обучается по подмножеству объектов
- Для каждого дерева есть существенное подмножество выборки, которое является контрольным для этого дерева
- По этому подмножеству можно оценивать качество дерева
- Это даёт возможность не использовать выделенную валидационную выборку

Основные библиотеки

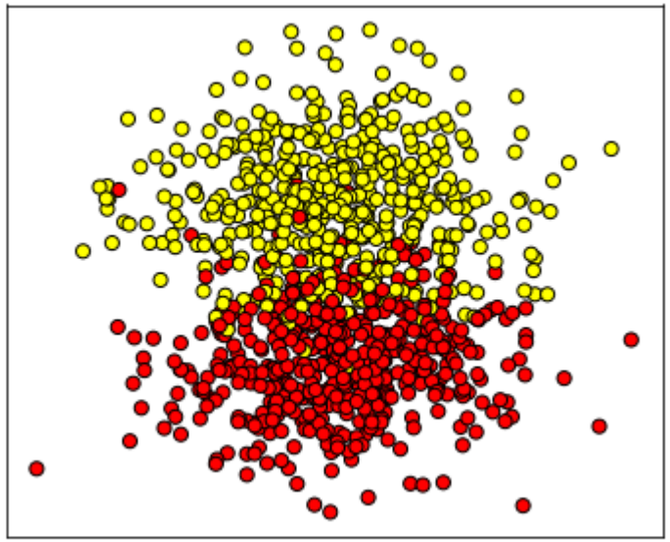
- Библиотеки общего назначения:
    - `sklearn`
- Библиотеки для обучения градиентного бустинга над решающими деревьями:
    - `XGBoost`
    - `LightGBM`
    - `CatBoost`
- Отличаются
    - скоростью
    - способом организацией параллелизма
    - способом оптимизации обучения и разбиений деревьев
    - способом обработки категориальных признаков

Композиции в sklearn

In [133]:
```
from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import BaggingRegressor

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import ExtraTreesRegressor

from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
```

Подготовка данных

In [160]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=2, n_informative=1, n_redundant=0,
                           n_classes=2, random_state=9, n_clusters_per_class=1, shuffle=True)

plt.figure(1, figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.autumn, s=50, edgecolors='k')
plt.xticks(())
plt.yticks(())
plt.show()
```



Обучение классификатора Random Forest

In [162]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=250, random_state=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

forest.fit(X_train, y_train)
```

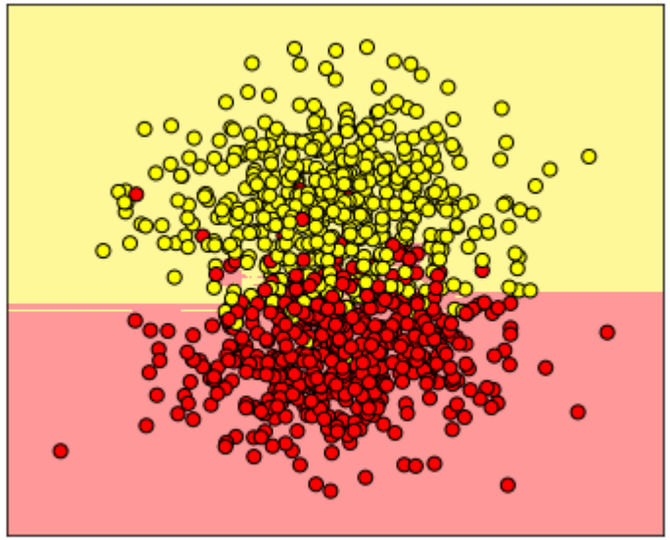
Out[162]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=250,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Основные параметры случайного леса в sklearn

- `n_estimators` - число деревьев в лесу
- `criterion` - критерий ветвления
- `min_samples_split` - минимальное число объектов в листе, допускающее деление
- `min_samples_leaf` - минимальное число элементов в листе
- `max_features` - максимально число перебираемых признаков при поиске лучшего ветвления
- `oob_score` - нужно ли использовать Out-of-bag score для оценивания обобщающей способности леса
- `n_jobs` - число параллельных потоков

Визуализация результатов

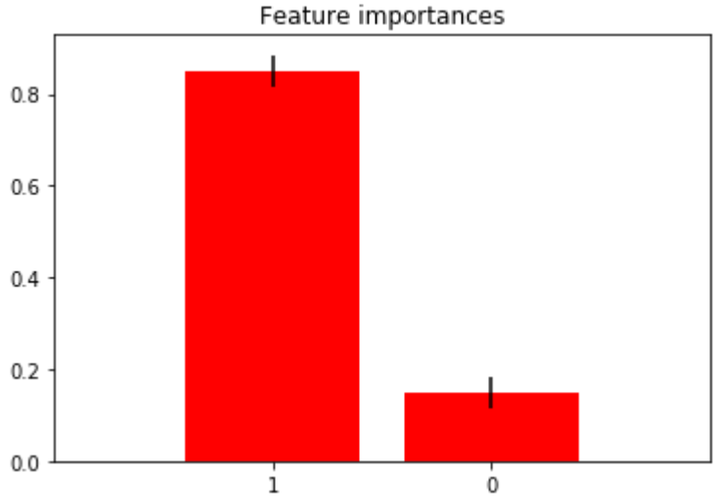
In [171]:
```
plot_results(forest, X, y)
```



Важность признаков

Помним, что выборка генерировалась с игнорированием одного из признаков

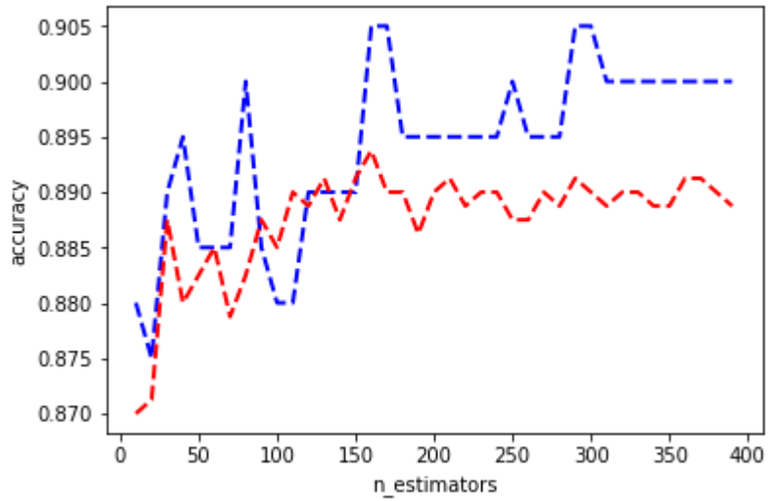
In [149]:
```
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
indices = np.argsort(importances)[::-1]

plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```



Out-of-Bag score

In [150]:
```
from sklearn.metrics import accuracy_score

oob_accuracy = []
test_accuracy = []
for i in range(10, 400, 10):
    forest = RandomForestClassifier(n_estimators=i, random_state=0, oob_score=True)
    forest.fit(X_train, y_train)

    oob_accuracy.append(forest.oob_score_)
    y_pred = forest.predict(X_test)
    test_accuracy.append(accuracy_score(y_test, y_pred))
```

In [152]:
```
plt.xlabel("n_estimators")
plt.ylabel("accuracy")
plt.plot(range(10, 400, 10), test_accuracy, 'b--',
         range(10, 400, 10), oob_accuracy, 'r--', linewidth=2)
plt.show()
```

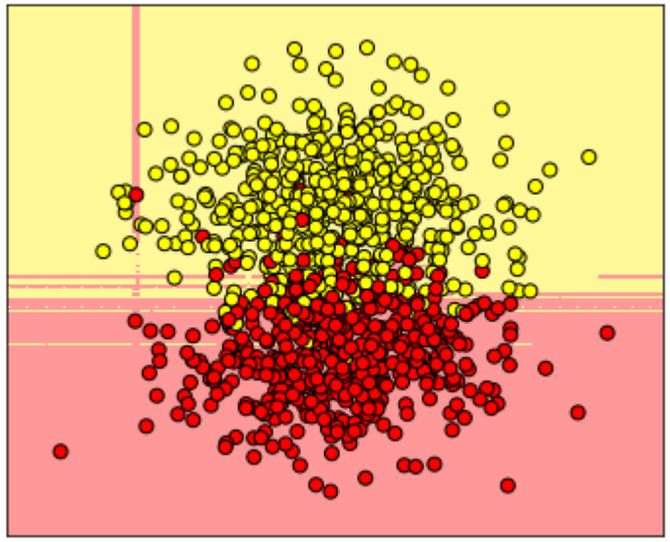


GB в sklearn

- Не самое эффективное по времени и памяти решение
- Лучше вместо этой реализации использовать xgboost или catboost
- Посмотрим для общего развития, принципиально ничего нового нет

Обучение классификатора Gradient Boosting

In [172]:
```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=250, random_state=0)
gbc.fit(X_train, y_train)
plot_results(gbc, X, y)
```

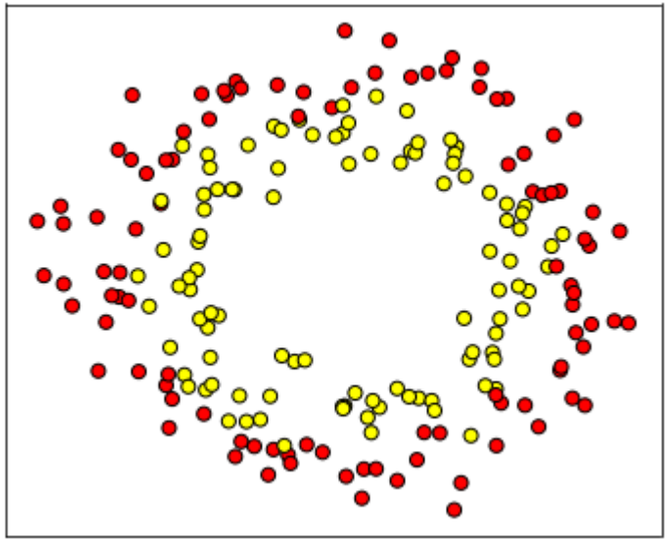


Основные параметры градиентного бустинга в sklearn

- `loss` - оптимизируемый функционал
- `learning_rate` - скорость обучения (по деревьям)
- `n_estimators` - число деревьев
- `subsample` - доля объектов для обучения каждого дерева
- `criterion` - критерий качества разбиения
- `min_samples_split` - минимальное число объектов в листе, допускающее деление
- `min_samples_leaf` - минимальное число элементов в листе
- `max_depth` - максимальная глубина каждого дерева
- `min_impurity_decrease` - минимальный выигрыш от разбиения, допускающий его проведение
- `max_features` - максимально число перебираемых признаков при поиске лучшего ветвления

Сгенерируем более сложные данные

In [173]:
```
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=200, shuffle=True, factor=0.7, noise=0.1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

plt.figure(1, figsize=(6, 5))
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.autumn, s=50, edgecolors='k')
plt.xticks(())
plt.yticks(())
plt.show()
```

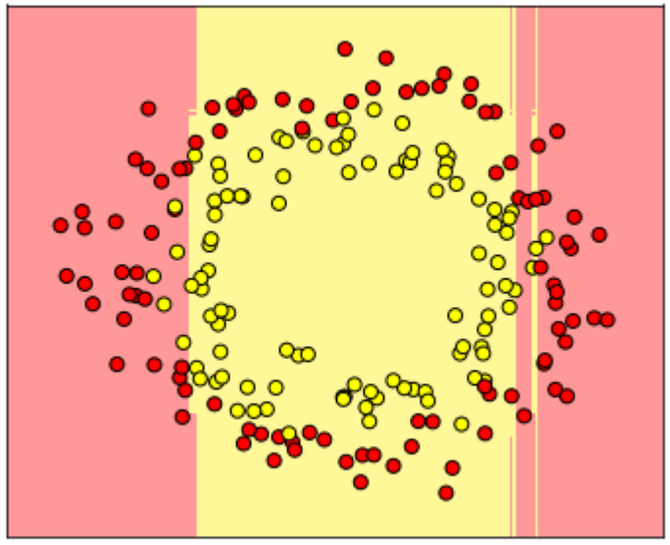


Запустим xgboost

- быстрая (параллельная), экономичная по памяти
- может запускаться на разных платформах и распределённых фреймворках
- допускает ускорение обучения с помощью GPU

In [174]:
```
import xgboost as xgb

xgbc = xgb.XGBClassifier(objective='reg:linear', colsample_bytree=0.3,
                         learning_rate=0.1, max_depth=3,
                         alpha=10, n_estimators=10)
xgbc.fit(X_train, y_train)
print(accuracy_score(y_test, xgbc.predict(X_test)))
plot_results(xgbc, X, y)
```

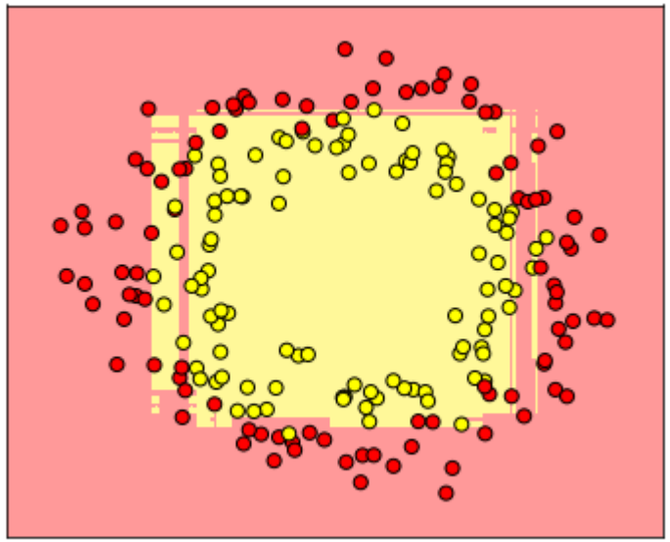
```
0.75
```



Попробуем другие параметры

In [175]:
```
xgbc = xgb.XGBClassifier(objective='reg:linear', colsample_bytree=0.3,
                         learning_rate=0.05, max_depth=3,
                         alpha=10, n_estimators=100)
xgbc.fit(X_train, y_train)

print(accuracy_score(y_test, xgbc.predict(X_test)))
plot_results(xgbc, X, y)
```

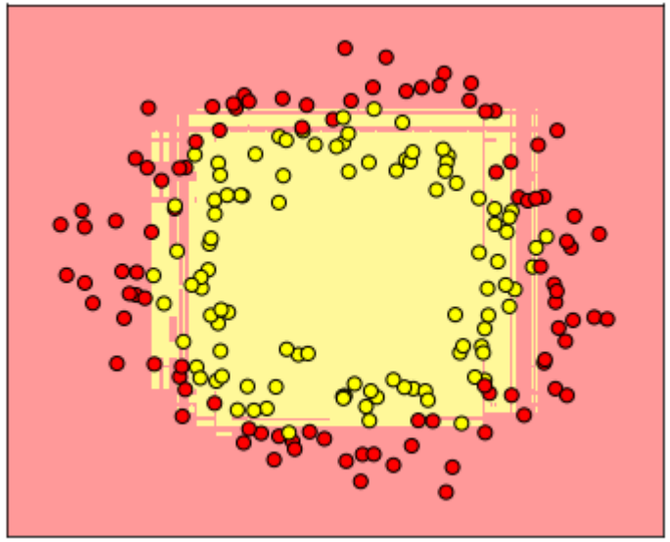
```
0.9
```



Попробуем другие параметры

In [176]:
```
xgbc = xgb.XGBClassifier(objective='reg:linear', colsample_bytree=0.3,
                         learning_rate=0.001, max_depth=3,
                         alpha=10, n_estimators=10000)
xgbc.fit(X_train, y_train)

print(accuracy_score(y_test, xgbc.predict(X_test)))
plot_results(xgbc, X, y)
```

```
0.925
```



Важные параметры xgboost

- `learning_rate` - скорость обучения (по деревьям)
- `max_depth` - максимальная глубина каждого дерева
- `subsample` - доля объектов для обучения каждого дерева
- `colsample_bytree` - максимально число перебираемых признаков при поиске лучшего ветвления
- `n_estimators` - число деревьев
- `objective` - оптимизируемый функционал

Одним из преимуществ xgboost является наличие регуляризации:

- `gamma` - коэффициент, влияющий на проверку уменьшения функционала перед ветвлением (ветвление может не произойти)
- `alpha` - коэффициент L_1-регуляризации для весов деревьев
- `lambda` - коэффициент L_2-регуляризации для весов деревьев

Пример из жизни: задача регрессии

- **Проблема:** у объектов много категориальных признаков

- **Инженерное решение**: изучить все категориальные признаки и с умом перекодировать их в числовые

- **Решение в лоб**:
    - перекодировать всё не глядя в one-hot
    - получится много признаков, больше, чем объектов в выборке ($\sim 10^5$)
    - применить матричное разложение, сократить размерность до нескольких тысяч
    - запустить на этом xgboost

- Во втором варианте изначально получилось качество, более низкое, чем требуемое
- Так вышло из-за того, что задание изначально было рассчитано на работу с признаками
- **Но!** Запуск на мощной машине большого ($\sim 10^4$ деревьев) бустинга в `xgboost` дал нужное качество
- В итоге время аналитика было замещено временем машины при том же результате

Библиотека `catboost`

- `CatBoost` способен упростить решение ещё сильнее
- `CatBoost` во многом похож на `XGBoost` (на GPU тоже может обучаться)
- Умеет из коробки работать с категориальными признаками
- В экспериментах авторов работает лучше `XGBoost` (как и обычно в научных статьях)
- Есть ещё полезные примочки, например визуализации обучения

Возьмём пример из документации

Посмотреть его и другие примеры можно [тут (https://catboost.ai/docs/concepts/python-usages-examples.html)](https://catboost.ai/docs/concepts/python-usages-examples.html)

In [4]:
```
from catboost import CatBoostRegressor
# Initialize data

train_data = [[1, 4, 5, 6],
              [4, 5, 6, 7],
              [30, 40, 50, 60]]

eval_data = [[2, 4, 6, 8],
             [1, 4, 50, 60]]

train_labels = [10, 20, 30]
# Initialize CatBoostRegressor
model = CatBoostRegressor(iterations=2,
                          learning_rate=1,
                          depth=2)
# Fit model
model.fit(train_data, train_labels)
# Get predictions
preds = model.predict(eval_data)
preds
```

```
0:      learn: 6.1237244        total: 384us    remaining: 384us
1:      learn: 4.5927933        total: 632us    remaining: 0us
```

Out[4]:
```
array([15.625, 18.125])
```

Задача понижения размерности

- Часто возникает необходимость перевести данные из одного признакового пространства в другое
- Иногда это можно сделать вручную, на чаще используются автоматические методы
- Один из возможных вариантов - уменьшение размерности:
    - часто используется в ситуации большого числа разреженных признаков (например, при обработке текстов или иных данных с категориальными признаками)
    - почти всегда это приводит к потере свойства интерпретируемости признаков (зависит от данных и метода)
- В качестве метода можно использовать различные матричные разложения
- Базовый вариант - сингулярное разложение (SVD)

Напоминание: сингулярное разложение

- Пусть есть матрица $X \in \mathbb{R}^{m \times n}$
- В этом случае её можно представить в виде следующего произведения: $X = U \Sigma V^T$
    - $\Sigma \in \mathbb{R}^{m \times n}$ - диагональная матрица, элементы главной диагонали - сингулярные числа (неотрицательные)
    - $U \in \mathbb{R}^{m \times m}$ и $V \in \mathbb{R}^{n \times n}$ - унитарные матрицы
- Разложение можно строить приближённо для k наиболее больших (то есть наиболее информативных) компонент Σ: $X = U_k \Sigma_k V_k^T$
- Тогда $U_k \Sigma_k$ становится новым признаковым описанием обучающей выборки
- Для получения признакового описания для тестовых данных $Z \in \mathbb{R}^{t \times n}$ достаточно умножить Z на $V_k \in \mathbb{R}^{n \times k}$
- SVD лежит в основе многих методов, например PCA или LSA

Снова sklearn

In [35]:
```
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import random as rnd

X = rnd(1000, 100, density=0.01)
svd = TruncatedSVD(n_components=10)
svd.fit(X)
print(f'V^T shape:\t{svd.components_.shape}')

X_transformed = svd.transform(X)
print(f'Transformes X shape:\t{X_transformed.shape}')

X_transformed_2 = X.dot(svd.components_.T)

print(f'Diff between transform and dot: {sum(sum(X_transformed - X_transformed_2))}')
```

```
V^T shape:      (10, 100)
Transformes X shape:    (1000, 10)
Diff between transform and dot: -2.190088388420719e-17
```

Спасибо за внимание!