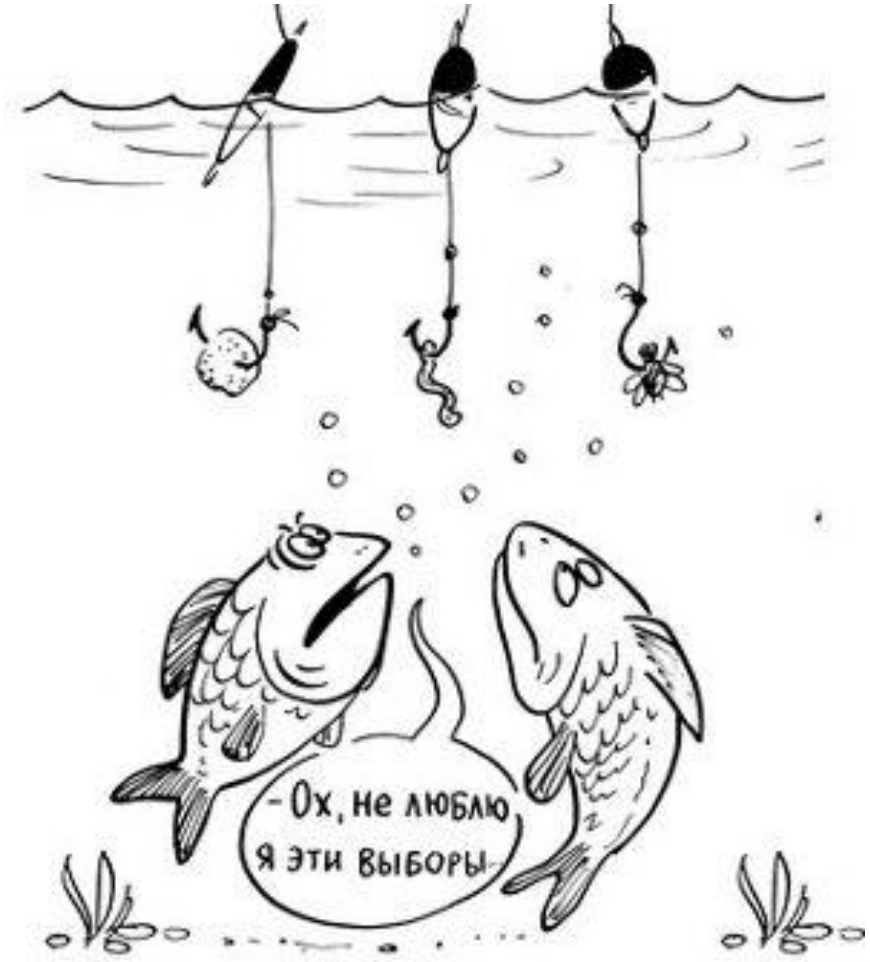
Машинное обучение и анализ данных

Контроль качества и выбор модели

Дьяконов А.Г.

Московский государственный университет
имени М.В. Ломоносова (Москва, Россия)

Контроль качества и выбор модели

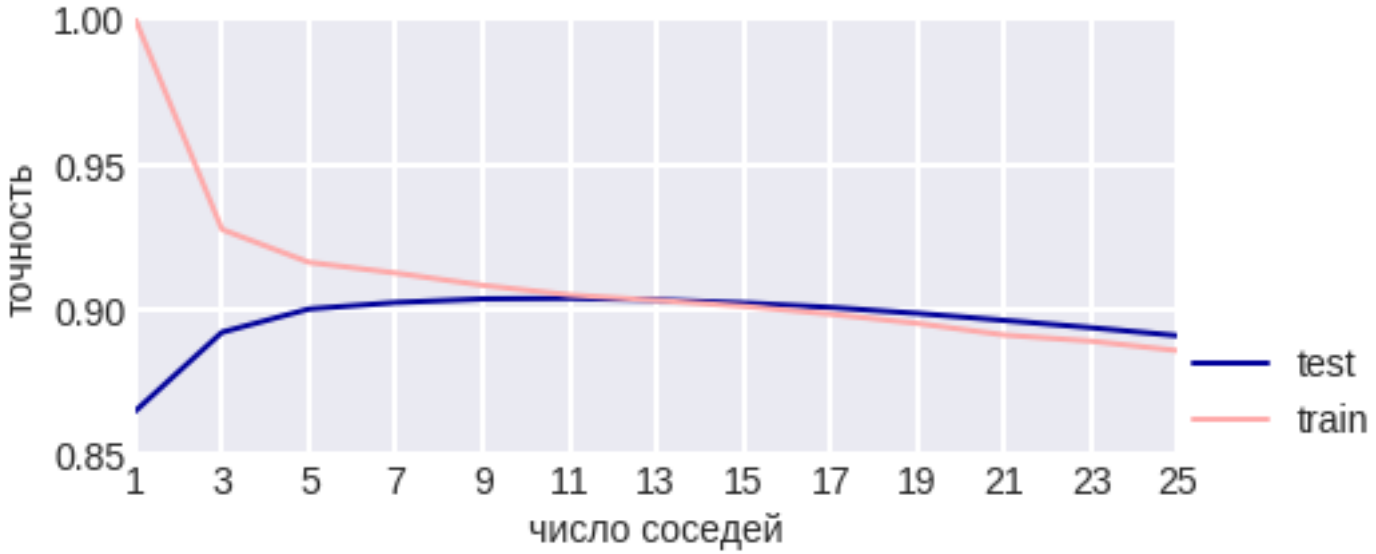


Проблема контроля качества

Как оценить качество / ошибку алгоритма?

Ошибка на обучении (train error) и ошибка на контроле (test error) – как правило очень различаются!

Нужен способ оценить качество работы (в будущем) алгоритма



Проблема выбора модели (Model Selection)

Моделей очень много… (но конечное число)

- **kNN**
- **SVM**
- **Linear**
- **NN**

…

Как выбрать модель?

В параметрических моделях – как выбрать параметры?

Как выбрать гиперпараметры (ex: k для kNN)?

Обычные параметры настраиваются на обучении

Очевидно, надо смотреть на качество – см. проблему выше…

MS может быть и для обучения без учителя!

Проблема выбора модели (Model Selection)

Выбор модели в широком смысле:

- **выбор модели алгоритмов**
- **выбор гиперпараметров**
- **выбор признаков**
- **выбор способа предобработки данных**

(заполнения пропусков, детектирования и удаления выбросов и т.п.)

Способы контроля

- **отложенный контроль (held-out data, hold-out set)**
- **скользящий контроль (cross-validation)**
- **бутстреп (bootstrap)**
- **контроль по времени (Out-of-time-контроль)**

используется для выбора модели и выбора гиперпараметров конкретной модели (выбирается модель с наименьшей ошибкой)

Общая схема / термины

Обучающая выборка – Training Set

обучение модели (настройка её параметров)

Валидационная выборка – Validation Set

настройка модели / (гипер)параметров модели /
выбор признакового пространства / …
иногда: локальный контроль

Тестовая выборка – Test Set

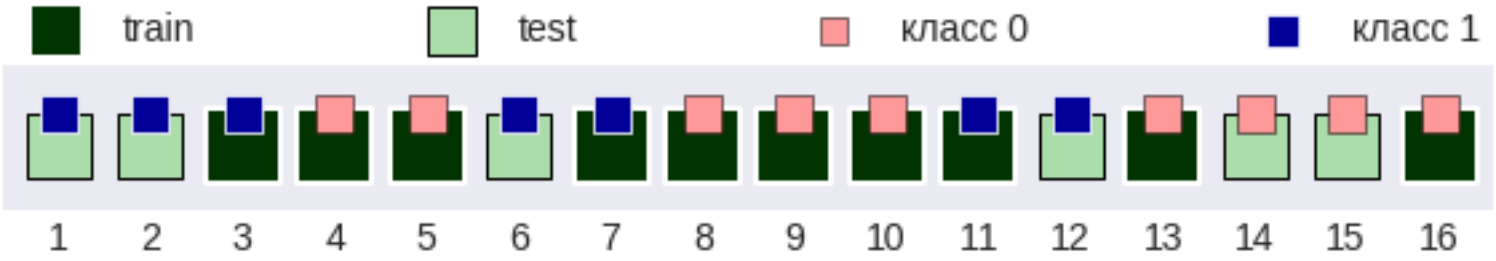
оценка качества модели
иногда: итоговая оценка

Отложенный контроль (held-out / validation data)

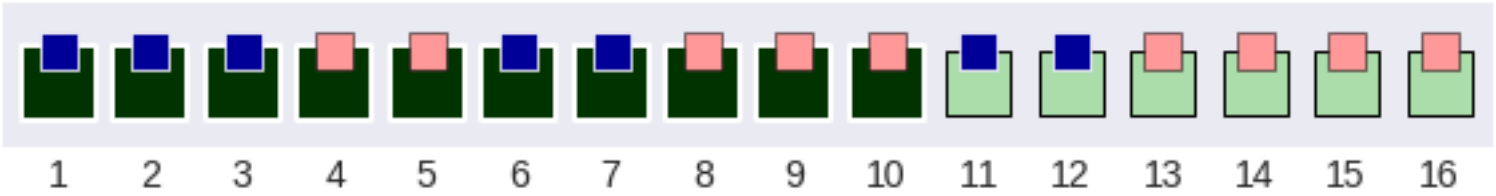
Выборку делим на две части:

- **обучение** – здесь настройка алгоритма
- **отложенный контроль** – здесь оценка качества, выбор алгоритма с наименьшей ошибкой

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33,
                                                    random_state=41)
```




```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33,
                                                    shuffle=False)
```



Отложенный контроль (held-out / validation data)

Оценка ошибки зависит от конкретной выбранной валидационной (отложенной выборки), часто сильно меняется при другом выборе,

Если переобучить алгоритм для всех данных, то мы не знаем оценку его ошибки (в каком-то смысле, неустранимый недостаток)

В какой пропорции делить…
(обычно ~20% от выборки, это не тестовые данные!!!)

Большое обучение

- алгоритм больше похож на обученный по всем данным
- обучающая выборка более репрезентативная

Большой контроль

- оценка качества более надёжна

Три правила разбиения выборки

- **Валидация моделирует реальную работу алгоритма!**

- **Нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку (качество)**

- **Тест должен быть случайным (или специально подготовленным Вами)**

Первое правило при разбиении выборки

Валидация моделирует реальную работу алгоритма!

Пример: если алгоритм должен работать на новых пользователях, для которых нет статистики, то и в валидационной выборке должны быть такие новые пользователи

Следствие: пропорции классов должны сохраняться

`shuffle=True` **(в новой версии)**

вообще, распределения в обучении и тесте д.б. одинаковыми

вопрос: как с вещественным признаком?

Проблема реальности: нестационарность (Nonstationarity)

– изменение параметров со временем

- **Covariate shift – меняются распределения**

(популярность постов, доходы и т.п.)

- **Concept drift – меняются правильные ответы**

(купил товар, уже не нужен)

Первое правило при разбиении выборки

Проблемы:

- **наличие / отсутствие дубликатов (почти дубликатов)**
- **вхождение целиком групп данных**

пример: всё транзакции конкретного пользователя

Второе правило при разбиении выборки

Для любых целей:

Нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку (качество)

Пример (курс Тибширани): нельзя найти подмножество признаков, максимально коррелирующих с целевым, а потом методом k-fold CV оценить ошибку этого метода – будет неверной!

Третье правило при разбиении выборки

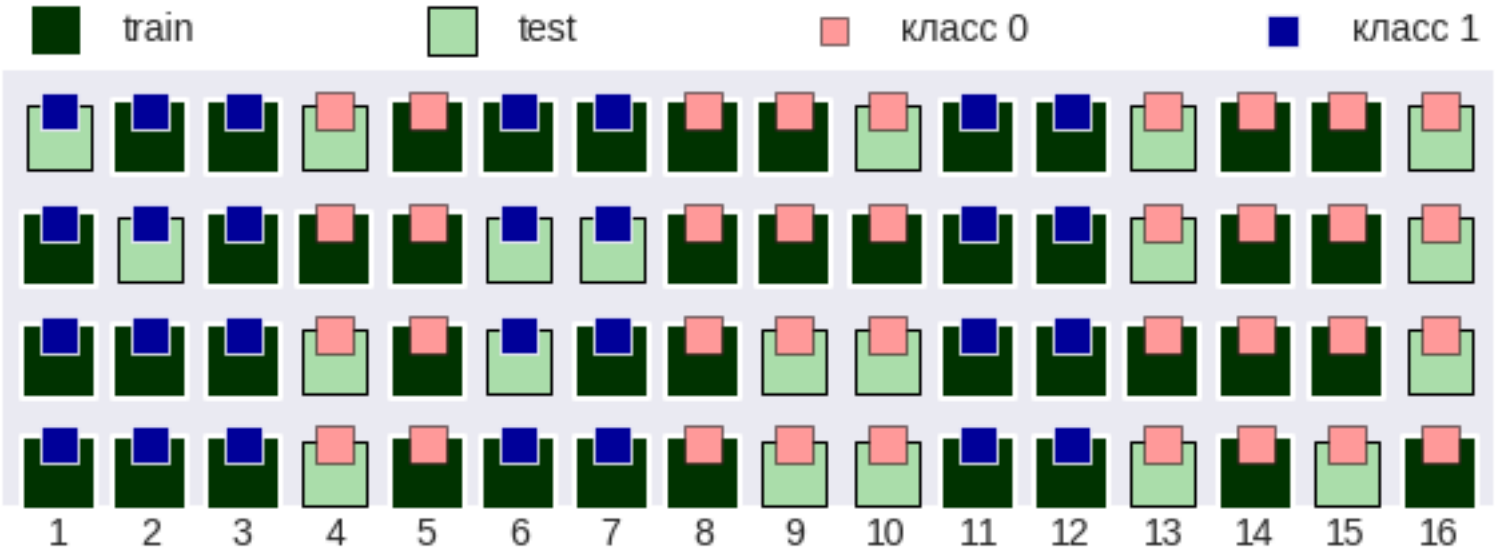
Тест должен быть случайным
(или специально подготовленным Вами)

По умолчанию: `shuffle=True`

Random Subsampling Cross-Validation

k раз случайно выбираем отложенный контроль, усредняем ошибки на всех отложенных выборках

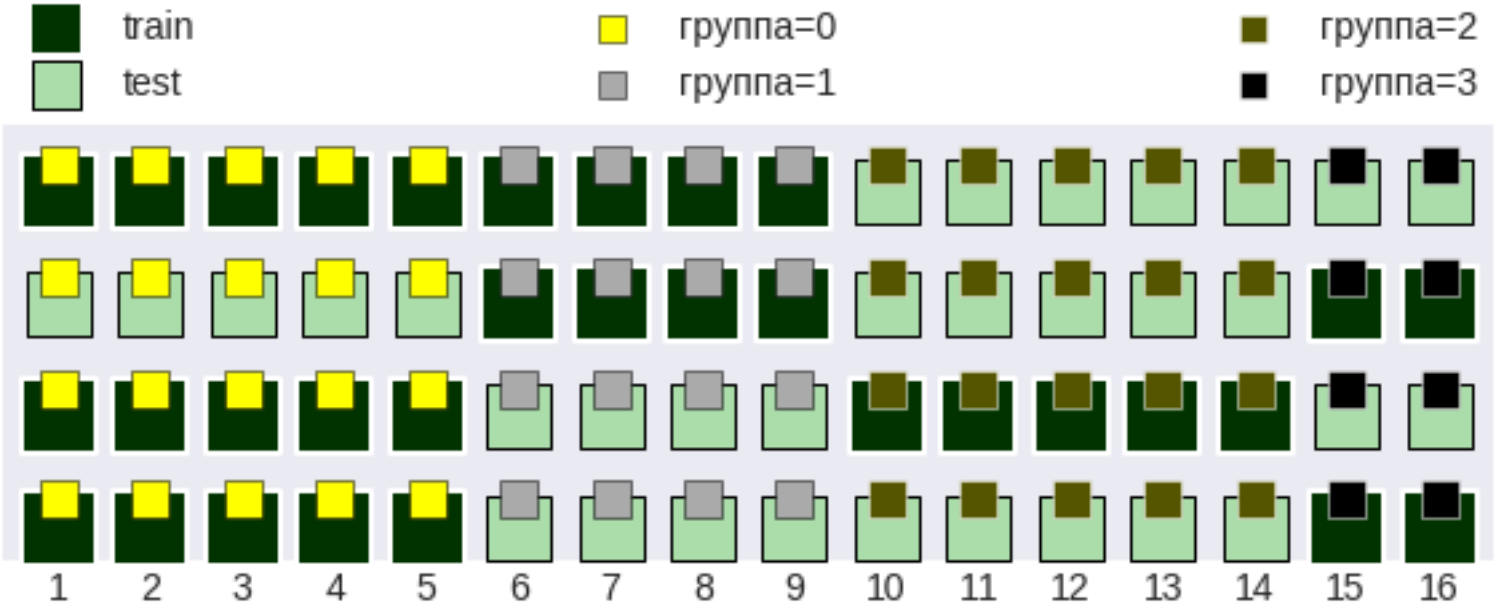
```
sklearn.model_selection.ShuffleSplit(n_splits=4,
                                      test_size=0.3,
                                      train_size=None,
                                      random_state=None)
```



Random Subsampling Cross-Validation: не забываем правила

без разбиения групп

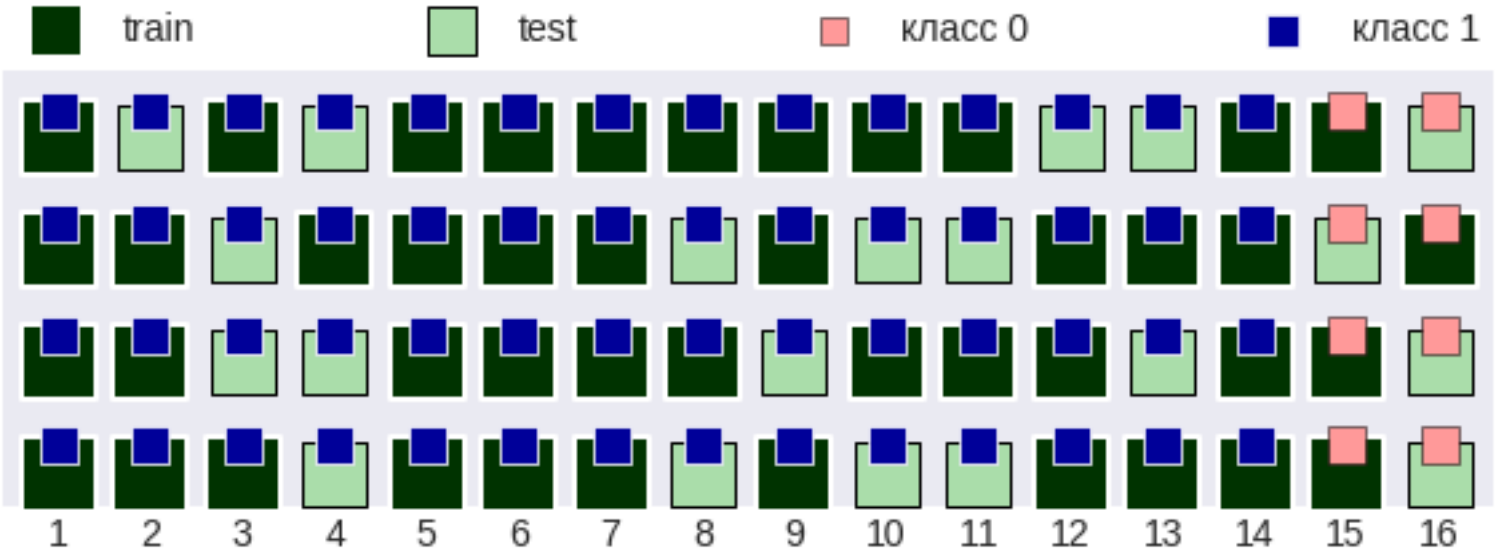
```
sklearn.model_selection.GroupShuffleSplit(n_splits=4,
                                          test_size=0.3,
                                          train_size=None,
                                          random_state=None)
for t, (itrain, itest) in enumerate(cv.split(x, groups=g)):
    ...
```



Random Subsampling Cross-Validation: не забываем правила

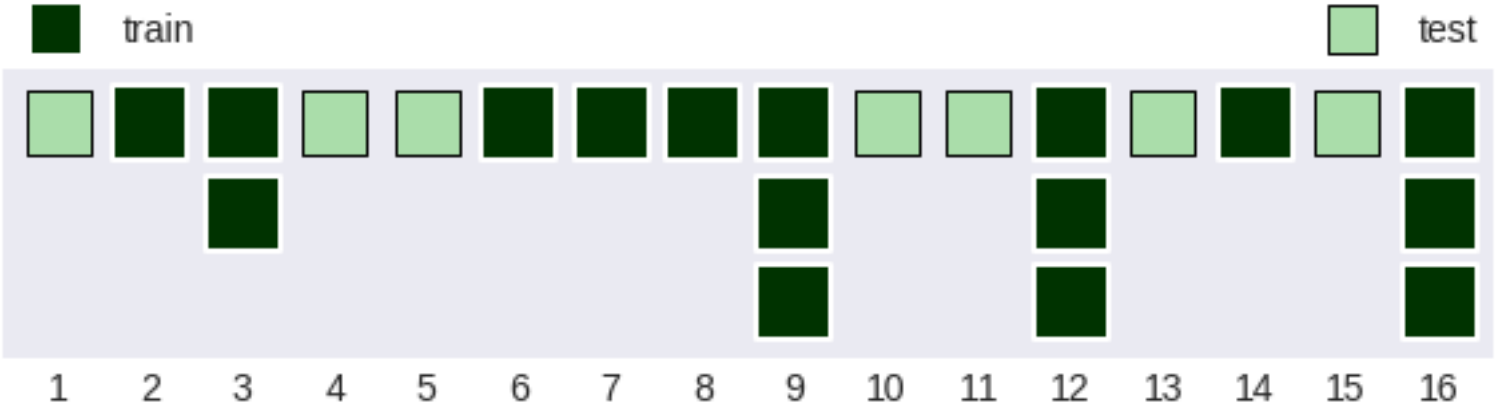
случайные разбиения сохраняя пропорции классов

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=4,
                                                test_size=0.3,
                                                train_size=None,
                                                random_state=None)
for t, (itrain, itest) in enumerate(cv.split(x, groups=y)):
    ...
```



Бутстреп (Bootstrap)

с помощью выбора с возвращением
формируется подвыборка полного объёма *m*,
на которой производится обучение модели
на остальных объектах (которые не попали в обучение) – контроль




```
i_train = [9, 16, 14, 9, 7, 12, 3, 12, 9, 8, 3, 2, 16, 12, 6, 16]
i__test =  [1, 4, 5, 10, 11, 13, 15]
```

Бутстреп (Bootstrap)

В контроль попадает примерно

$$\left(1-\frac{1}{m}\right)^m \approx e^{-1} \approx 0.37 = 37\%$$ **выборки.**

Чем хорошо:

- **модель учится на выборке того же объёма, что и итоговая (которую мы обучим по всей)**

Но… использует не все данные!

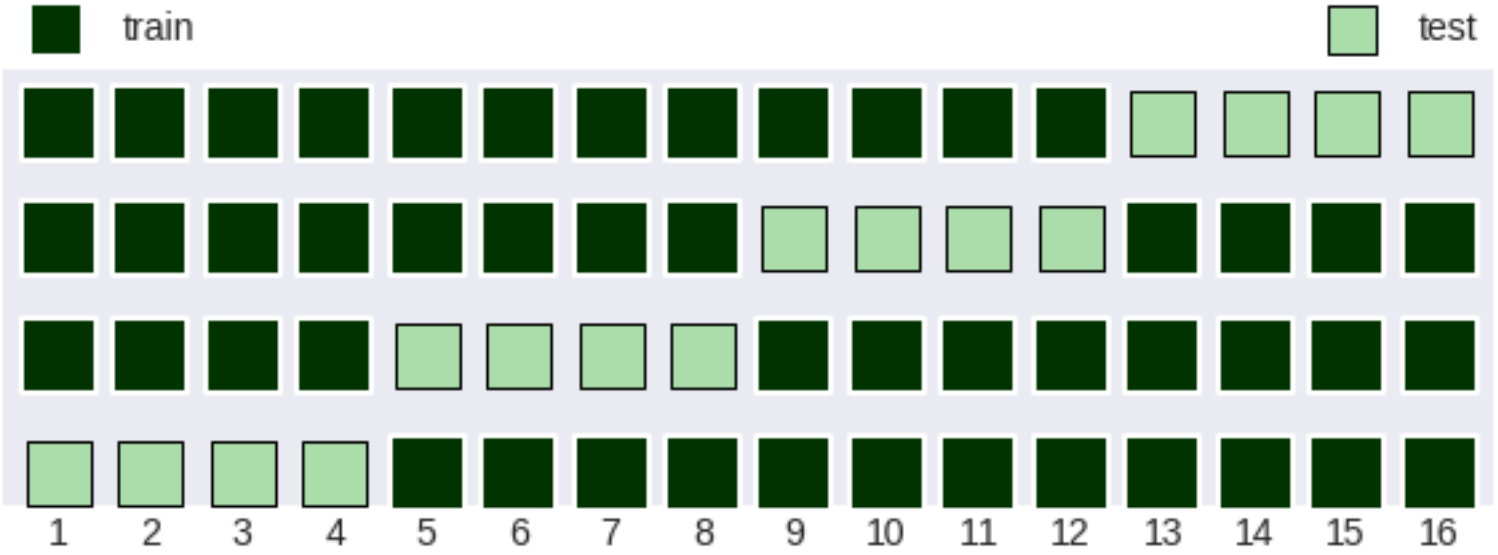
есть дубликаты

- **с точки зрения распределения бутстреп-выборка похожа на исходную**

Перекрёстная проверка по фолдам (k-fold cross-validation)

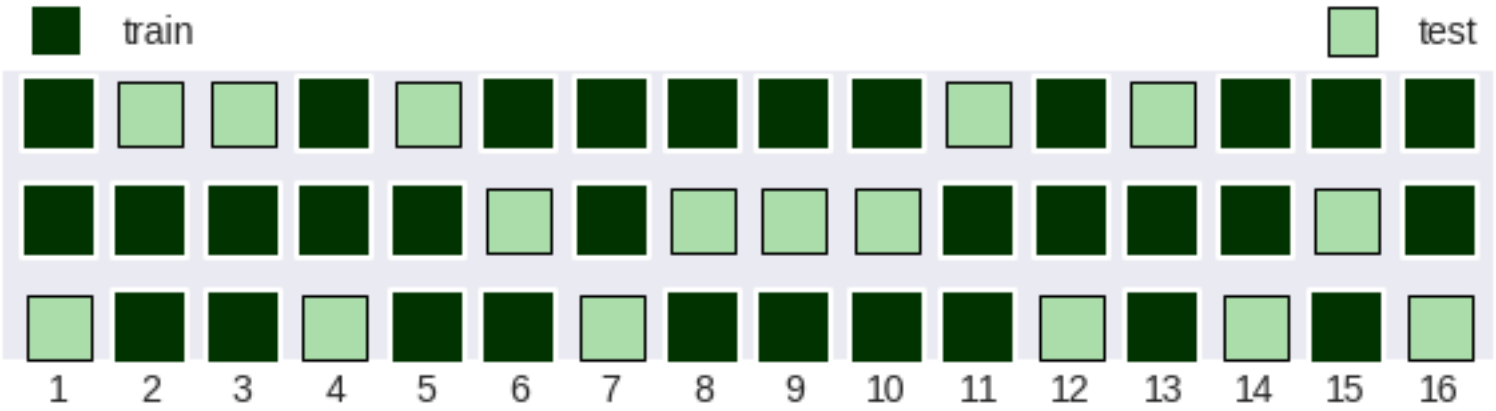
- **Разделить выборку на k примерно равных частей**
- **цикл по i=1…k**
  - **использовать i-ю часть для валидации (вычислить ошибку на ней), а объединение остальных – для обучения**
- **усреднить k ошибок, вычисленных на разных итерациях цикла (можно использовать дисперсию для оценки доверия к полученному качеству)**

```
sklearn.model_selection.KFold(n_splits='warn',
                              shuffle=False,
                              random_state=None)
```

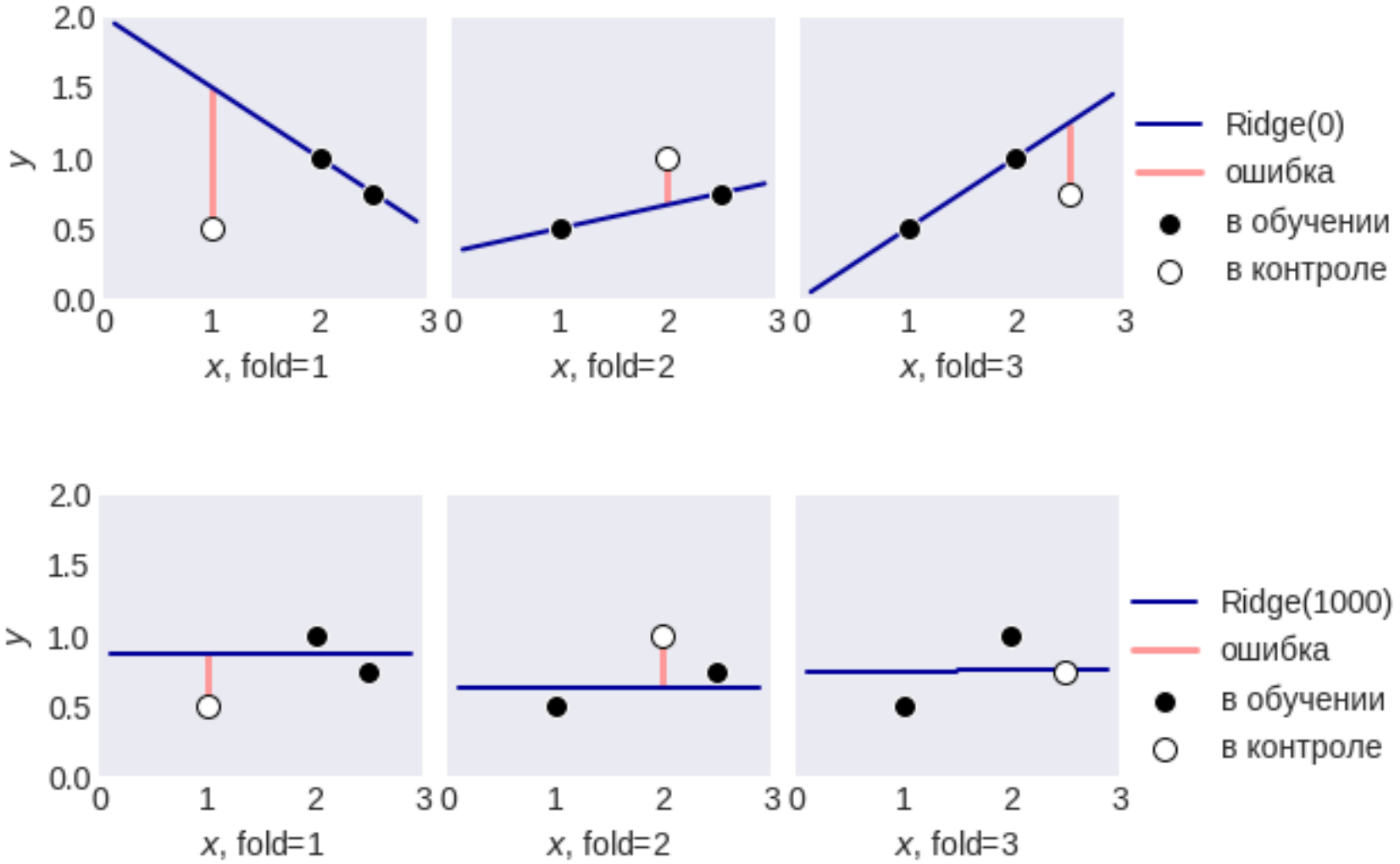


Перекрёстная проверка по фолдам (k-fold cross-validation)

```
sklearn.model_selection.KFold(n_splits=3, shuffle=True, random_state=None)
```



Перекрёстная проверка по фолдам: иллюстрация



Перекрёстная проверка по фолдам: тонкости

«k примерно равных частей»

в последней части может быть меньше объектов, чем в остальных, если k не делит нацело объём выборки

Обычно выбирают k=10

Большие k –

- **надёжнее оценка качества**
- **обучающая выборка больше походит на все данные**
- **время контроля возрастает (линейно)!**
- **не любое качество адекватно оценивается на меленьких подвыборках**

Перекрёстная проверка по фолдам: не забываем правила

сохраняем пропорцию классов

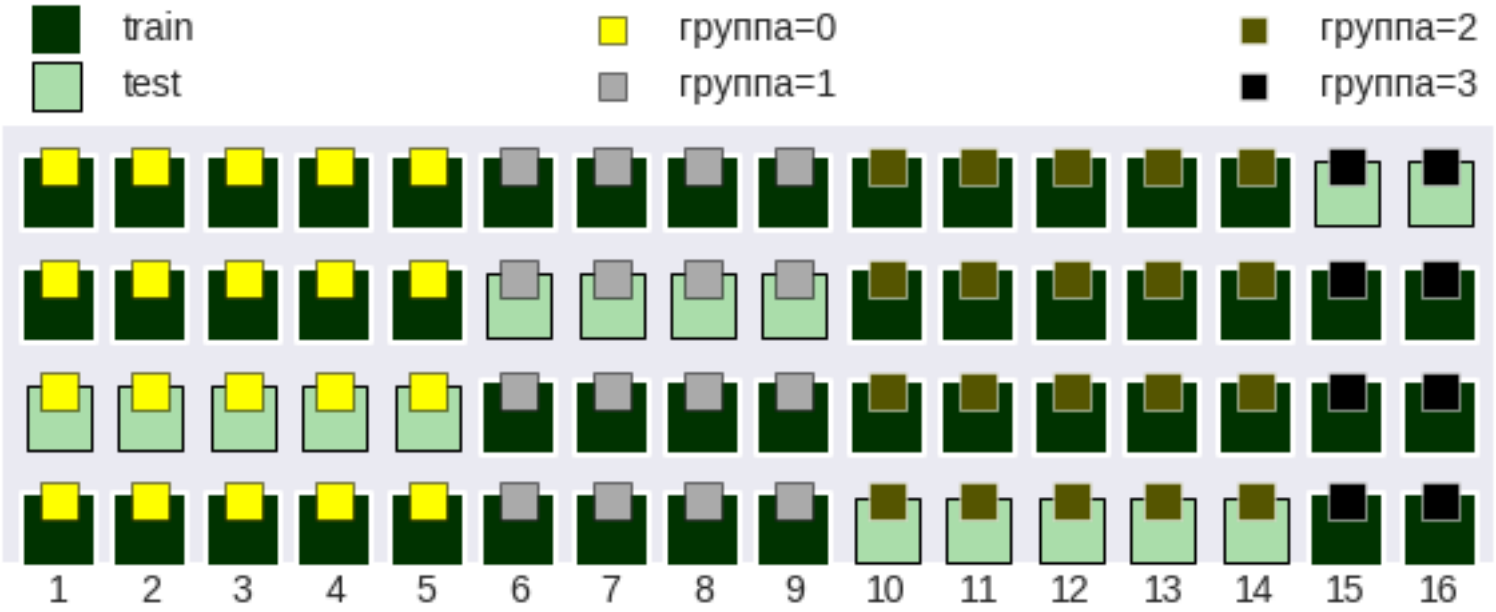
```
sklearn.model_selection.StratifiedKFold(n_splits='warn',
                                        shuffle=False,
                                        random_state=None)
```

перемешиваем: `shuffle=True`

Перекрёстная проверка по фолдам: не забываем правила

не разбиваем группы

```
sklearn.model_selection.GroupKFold(n_splits='warn')
```



вопрос: когда это нужно?

ещё есть: `sklearn.model_selection.PredefinedSplit` **– разбиение индуцированное группами**

Leave-one out cross-validation (LOOCV)

k-fold при k=n

`sklearn.model_selection.LeaveOneOut`

показаны только первые 4 разбивки...

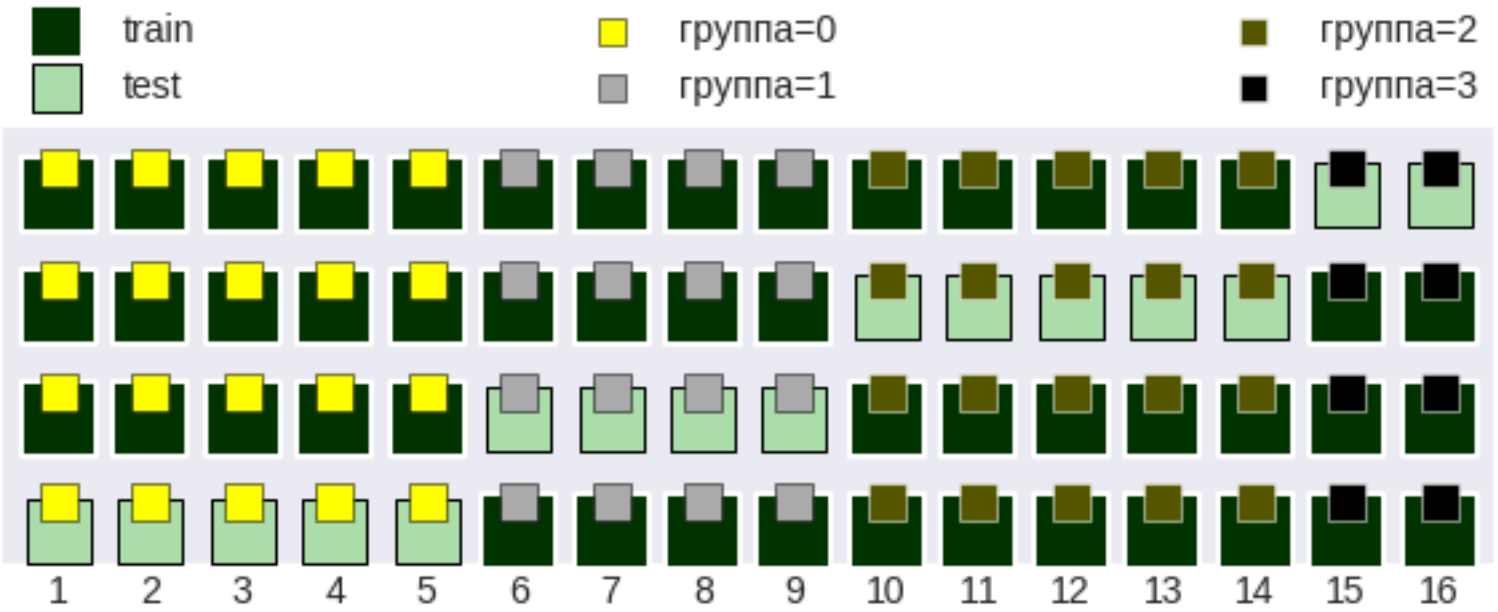
- **может слишком долго вычисляться**

ещё есть `sklearn.model_selection.LeavePOut` **– всевозможные n-ки**

Контроль по группам: LeaveOneGroupOut

LeaveOneGroupOut: Контроль по одной группе

```
from sklearn.model_selection import LeaveOneGroupOut
```



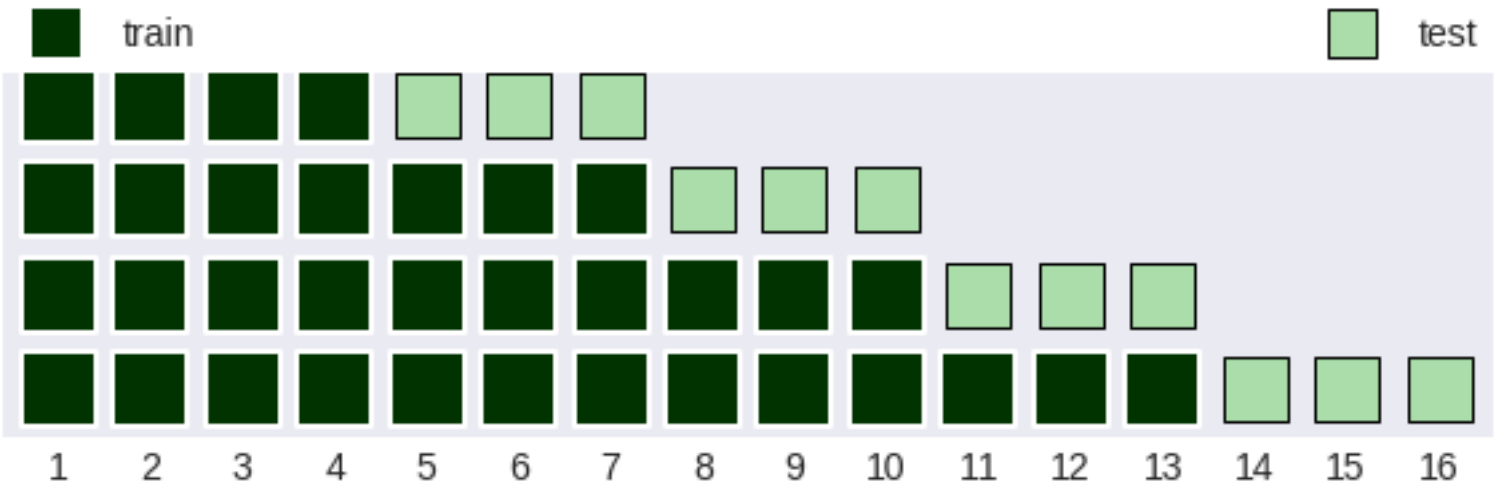
тонкость:
при оценки ошибки можно(нужно?)
учитывать мощность групп

$$e = \sum_{t=1}^{k} \frac{m_t}{m} e_t$$

Контроль по времени (Out-of-time-контроль)

TimeSeriesSplit: разбиения временных рядов (Time series cross-validation)

```
sklearn.model_selection.TimeSeriesSplit(n_splits='warn',
                                        max_train_size=None)
```



- **часто не получится сделать много контролей (слишком маленькая предыстория)**
- **первое правило ⇒ знаем как организовывать**

Терминология

user index	Training Data	in sample out of time
	out of sample in time	out of sample out of time

time

Локальный контроль

– организация контроля, когда итоговое качество алгоритма будет оцениваться на заранее заданной выборке (глобальном контроле)

- **соревнования**
- **обучение с частичной разметкой (semi-supervised learning)**

Вспоминаем первое правило…

- **распределения по признакам должны совпадать**
- **распределение выбросов, пропусков…**
- **если прогнозирование – на похожие период**

(с того же дня недели, столько же праздников впереди и т.п.)

Корректность локального контроля при использования кодировок

Разбиения корректные (например, OOT)

Локально:	Глобально:
Определяем кодировки на train_code	Определяем кодировки на train
Кодируем train_code и valid	Кодируем train и test
Регуляризуем на train_code	Регуляризуем на train
Валидируем на valid	Обучаем на train, предсказываем на test

Где используется выбор CV-контроля

- **оценка качества модели**

- **получение «честных» ответов на обучении**
  - как бы алгоритм отвечал,
  - если бы не обучался на этих объектах
  - **контроль алгоритмов**
  - **ансамблирование (метапризнаки)**

- **настройка гиперпараметров**

- **построение кривых зависимостей качества от параметров**
  - **validation curves (от значений)**
  - **learning curves (от объёма выборки)**

Оценка модели с помощью выбранного контроля: минутка кода

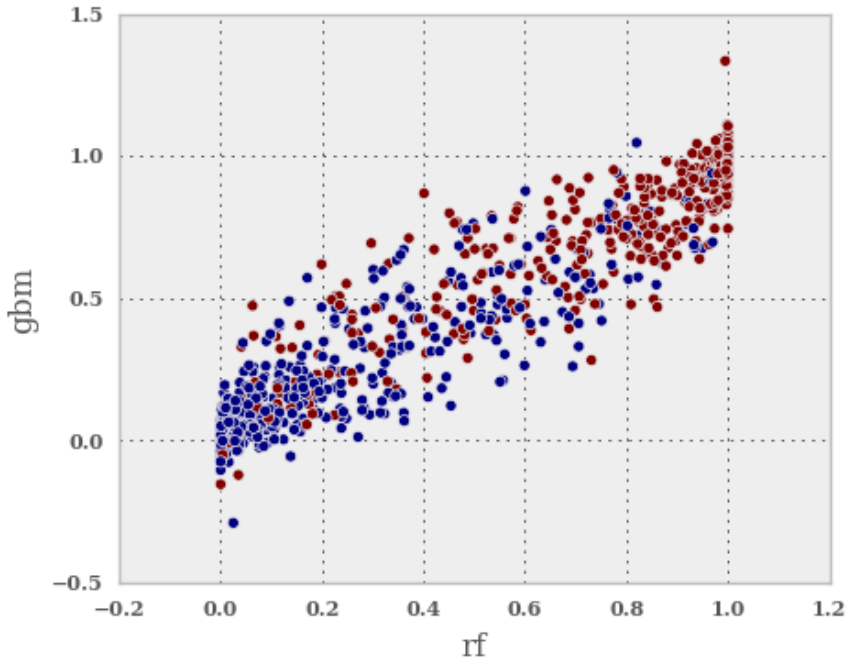
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                  train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

У этих функций много параметров…

Они (функции) «понимают» друг друга

Если не указываем скорер – используется встроенный (в модель)

Ответы алгоритма с помощью выбранного контроля: минутка кода

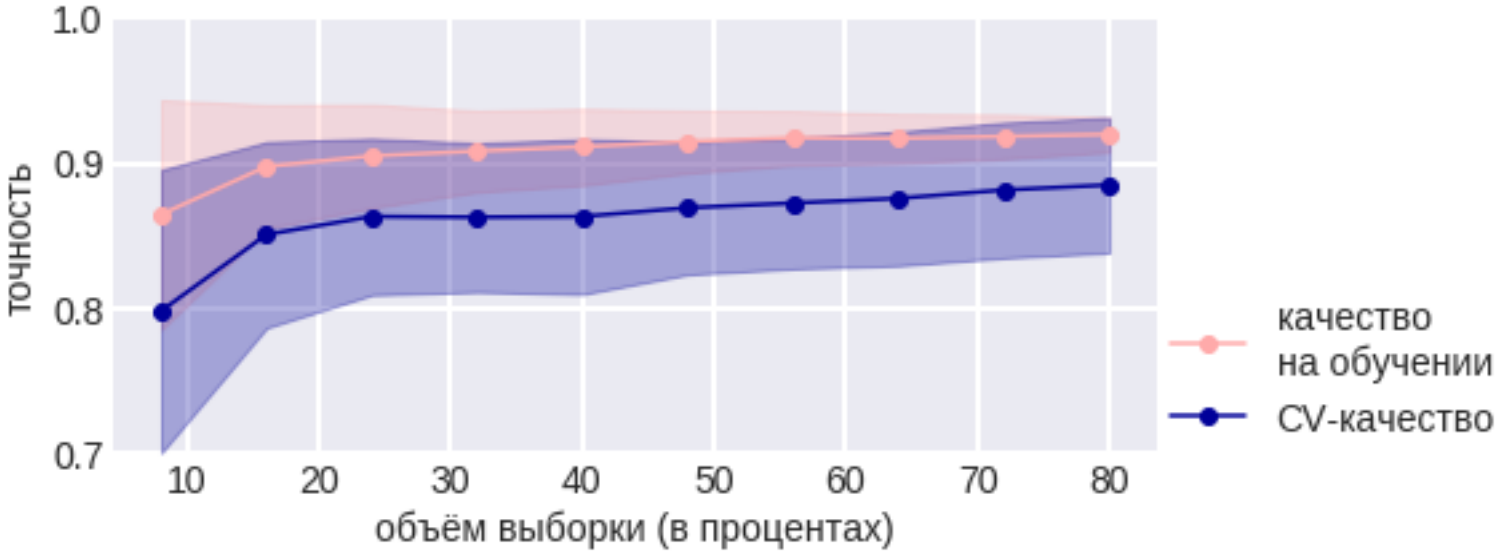



```
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
cv = KFold(n_splits=10, shuffle=True, random_state=1)
a_rf = cross_val_predict(rf, X, y, cv=cv) # ответы rf на CV
a_gbm = cross_val_predict(gbm, X, y, cv=cv) # ответы gbm на CV

plt.scatter(a_rf, a_gbm, c=y)
plt.xlabel('rf')
plt.ylabel('gbm')
```

Кривые обучения (Learning Curves)

Делим данные на обучение и контроль (м.б. очень много раз)
Обучаемся на k% от обучающей выборки для разных k
Строим графики ошибок/качества на train/CV от k

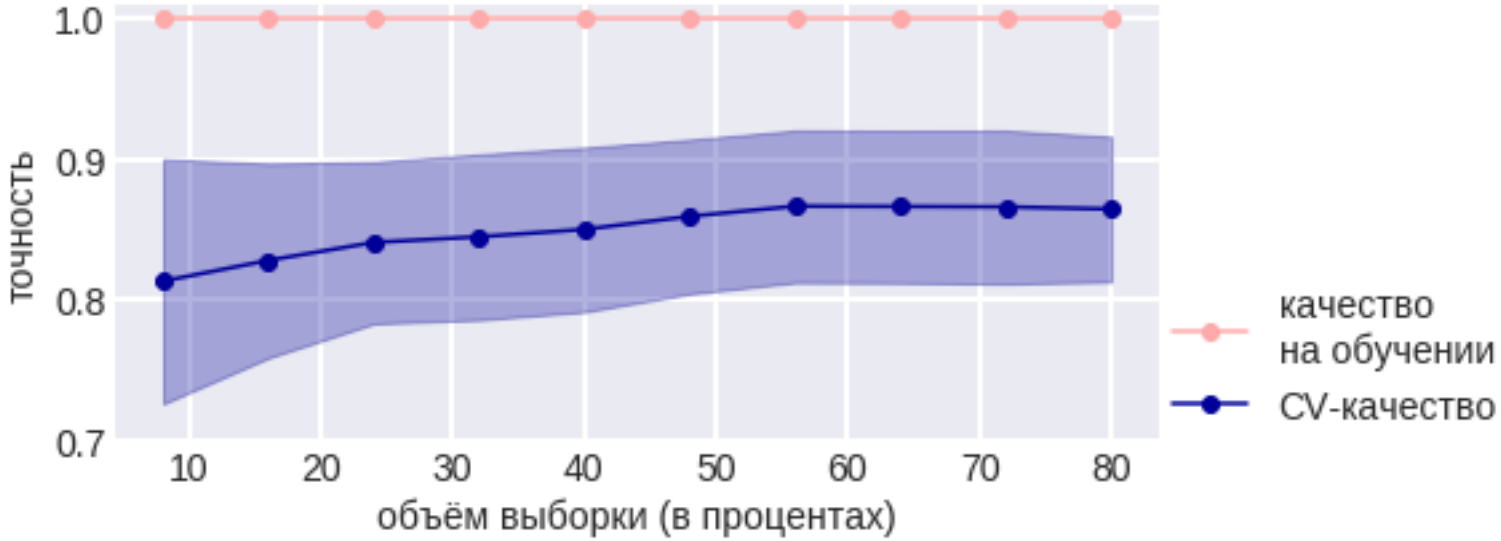
`model_selection.learning_curve`



Есть зазор между обучением и CV

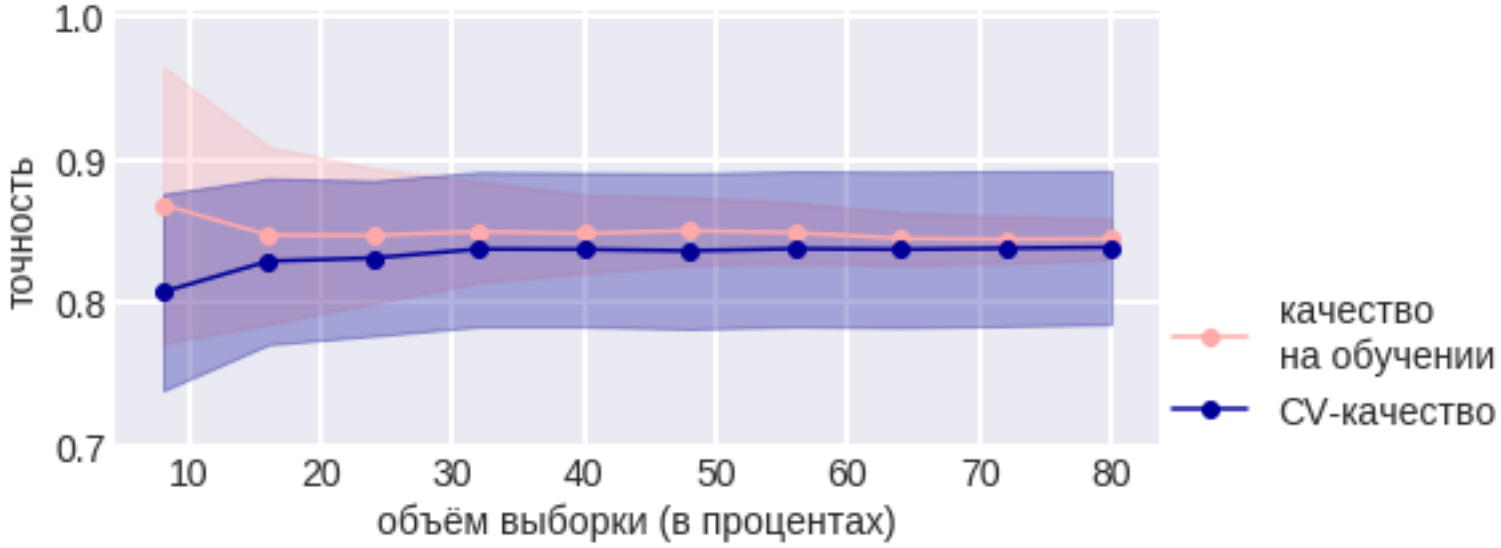
Тонкость: 100% – вся выборка, но здесь `test_size=0.2`

Кривые обучения (Learning Curves)



Переобучение:
100% качество на обучении!
1NN?

Нет выгоды от объёма!

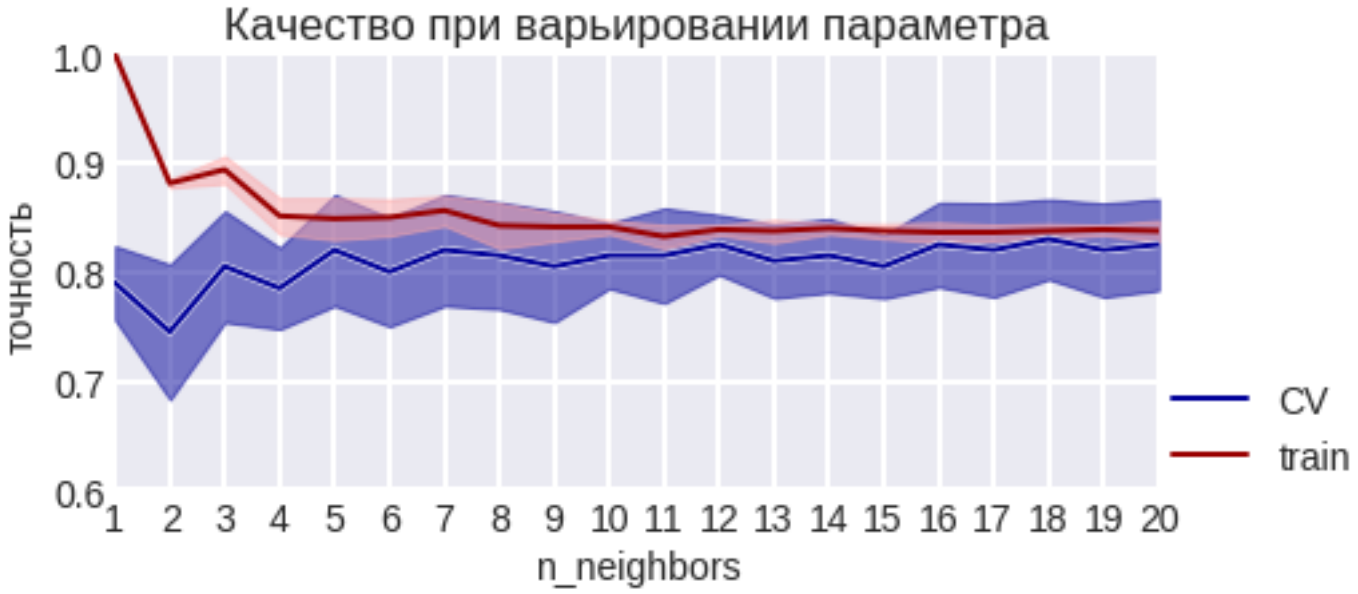


Полная согласованность между обучением и CV, но качество низкое

Качество от параметров

Делим данные на обучение и контроль (м.б. очень много раз)
При разных значениях параметров обучаемся и проверяем качество
Строим графики ошибок/качества на train/CV от значения параметра

`sklearn.model_selection.validation_curve`



Перебор параметров

Делим данные на обучение и контроль (м.б. очень много раз)
При разных значениях параметров обучаемся и проверяем качество

`sklearn.model_selection.GridSearchCV`

	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$
euclidean	76.0	77.0	79.0	78.5	80.5	82.5
manhattan	74.0	74.0	79.0	79.5	80.5	81.0
chebyshev	76.5	78.5	80.0	80.0	81.0	81.5

Есть также случайный поиск
`model_selection.RandomizedSearchCV`
(тут есть «число итераций»)

Ссылки

- **презентация по sklearn**

https://github.com/Dyakonov/IML/blob/master/IML2018_06_scikitlearn_10.pdf

– есть код для различных организаций контроля