

«Машинное обучение и анализ данных»

Контроль качества и выбор модели

Александр Дьяконов



5 октября 2020 года

План

Проблема выбора модели (в широком смысле)

Способы контроля

отложенный / скользящий / перекрёстный / бутстреп / по времени

Три золотых правила разбиения выборки

моделируем реальность / нет утечкам / случайность

Локальный контроль

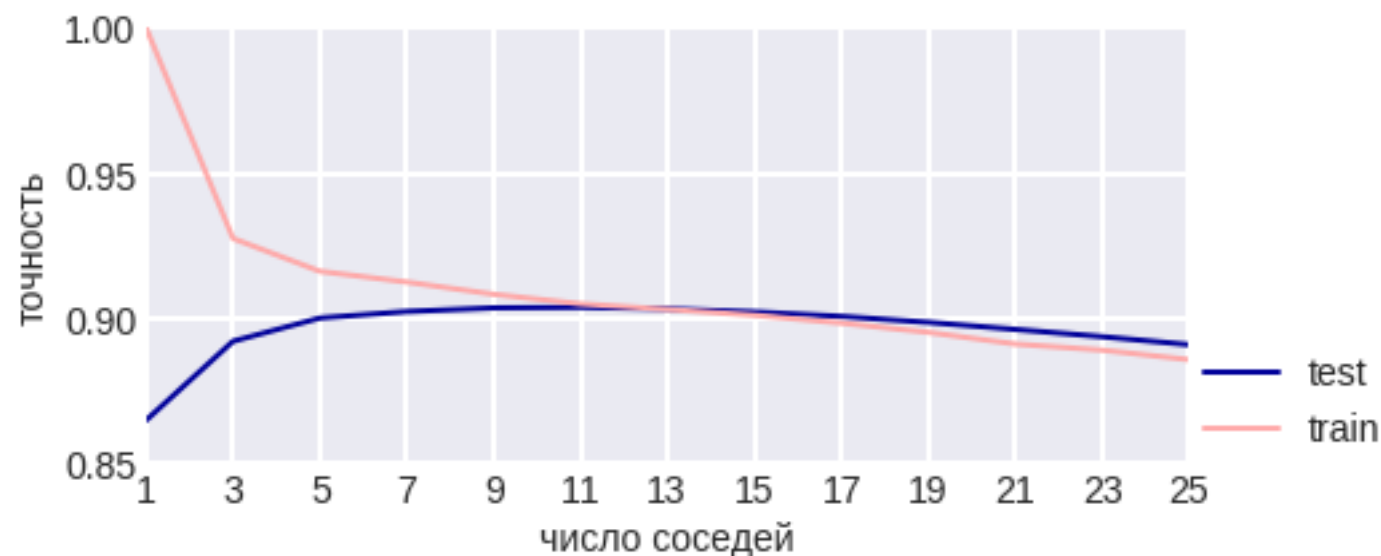
Где используется выбор CV-контроля

Проблема контроля качества

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

Как оценить качество / ошибку алгоритма?
model performance / model error

Нужен способ оценить качество работы (в будущем) алгоритма



Модель (Model)

– параметрическое семейство алгоритмов

алгоритм (параметры, **гиперпараметры**)

(обычные) параметры (model parameters) – настраиваются в результате обучения

гиперпараметры (hyperparameters) – выбираются до обучения экспертно / настраиваются переборно

Моделей очень много... (но конечное число)

- kNN
- NearCentroid
- SVM
- Linear
- NN
- ...

Как выбрать модель?

Проблема выбора модели (Model Selection)

Гиперпараметры часто как бы задают разные подмодели...

1NN

3NN

5NN

...

Как выбрать модель?

Очевидно, надо смотреть на качество – см. проблеме выше...

MS может быть и для обучения без учителя!

Проблема выбора модели (Model Selection)

Выбор модели в широком смысле – «пайплайна»:

- **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

выбор пайплайна (модели / гиперпараметров / признаков)

иногда: локальный контроль

Тестовая выборка – Test Set

оценка качества алгоритма

иногда: итоговая оценка

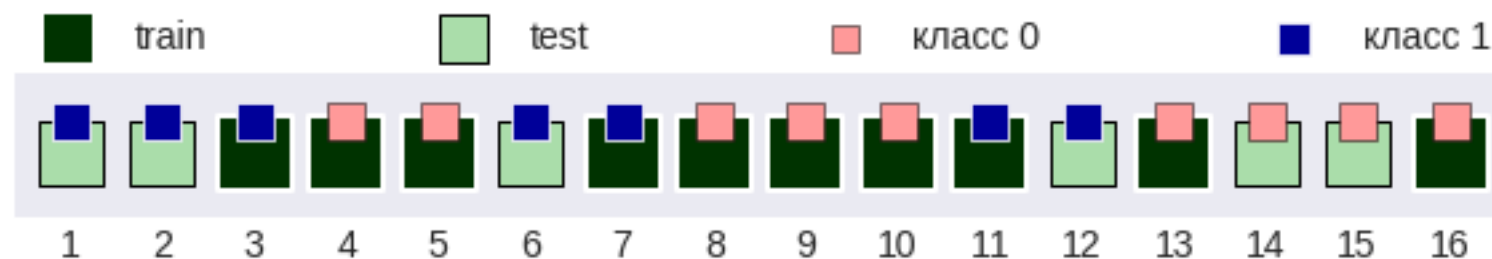


Отложенный контроль (held-out / validation data)

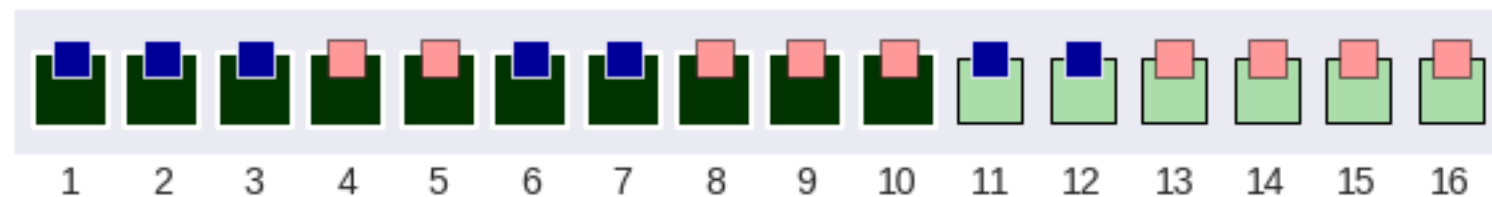
Выборку делим на две части:

- **обучение** – здесь обучение 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 золотых 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: 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?

Первое правило: валидация моделирует реальную работу алгоритма

Проблемы реализации:

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

пример: всё транзакции конкретного пользователя

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)**

Второе правило: нельзя явно или неявно использовать метки объектов, на которых оцениваешь ошибку (качество)

Это для любых целей!

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: тест должен быть случайным (или подготовленным)

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)
```

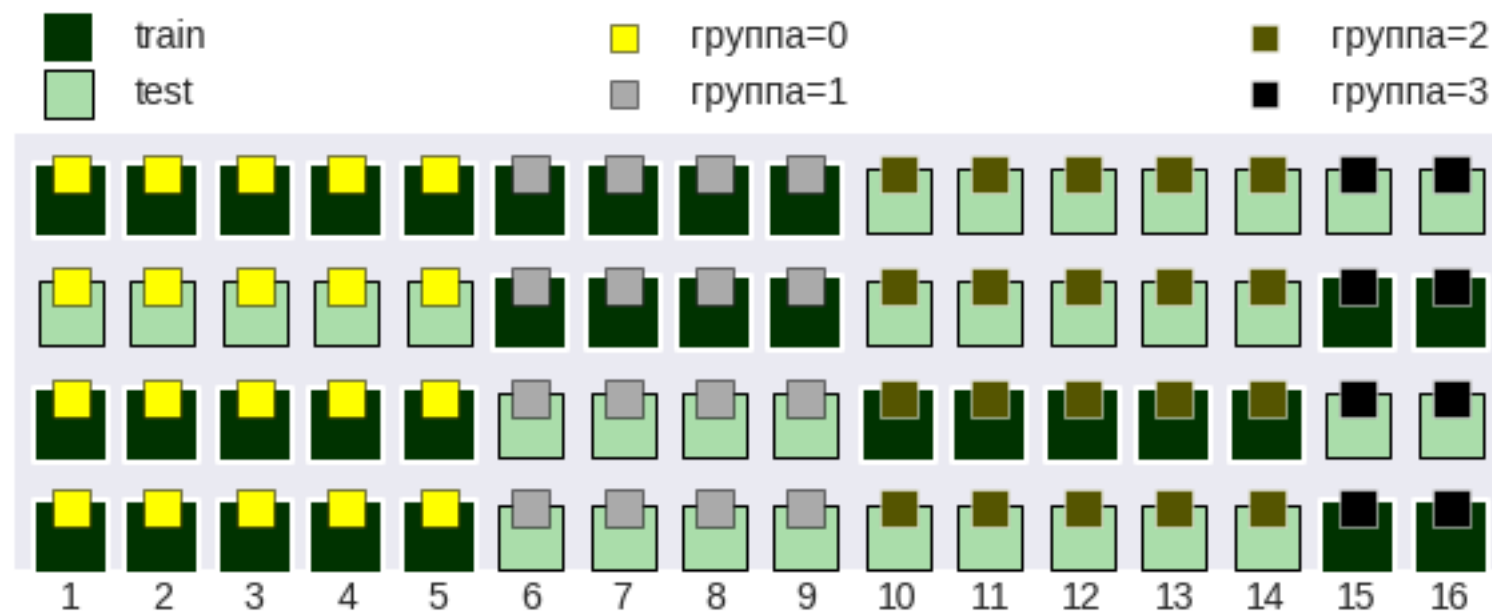


Random Subsampling Cross-Validation: не забываем правила

без разбиения групп **вопрос: когда это нужно?**

```
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

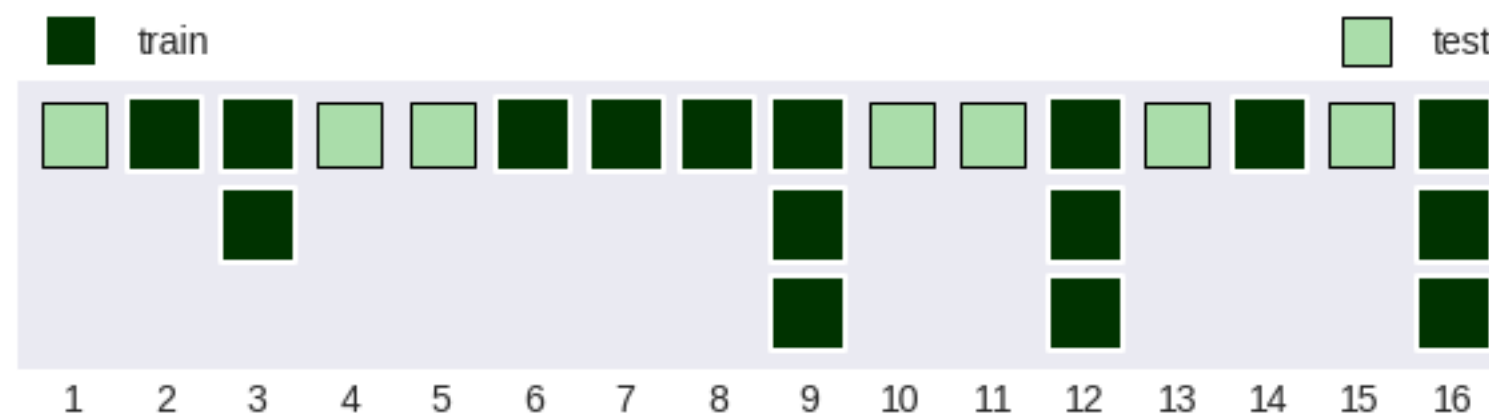
```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=4,  
                                                test_size=0.3,  
                                                train_size=None,  
                                                random_state=None)
```

```
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

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

Чем хорошо:

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

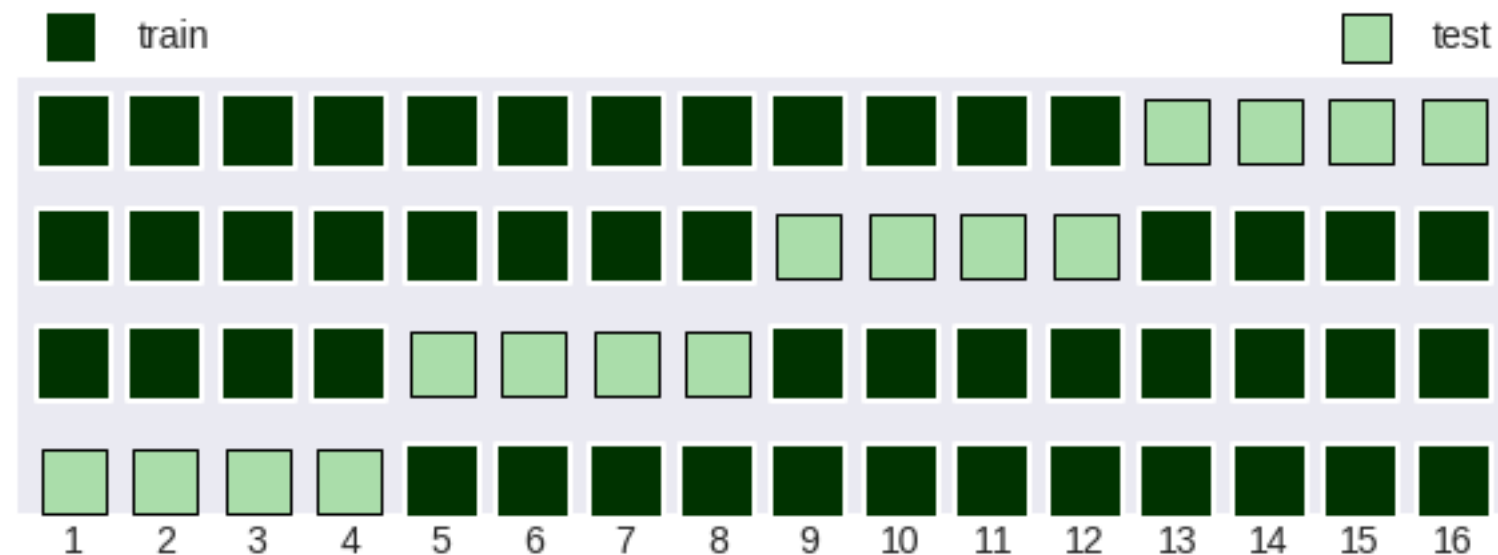
**Но... использует не все данные!
есть дубликаты**

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

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

- Разделить выборку на k примерно равных частей
- цикл по $i = 1 \dots k$
 - 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)
```



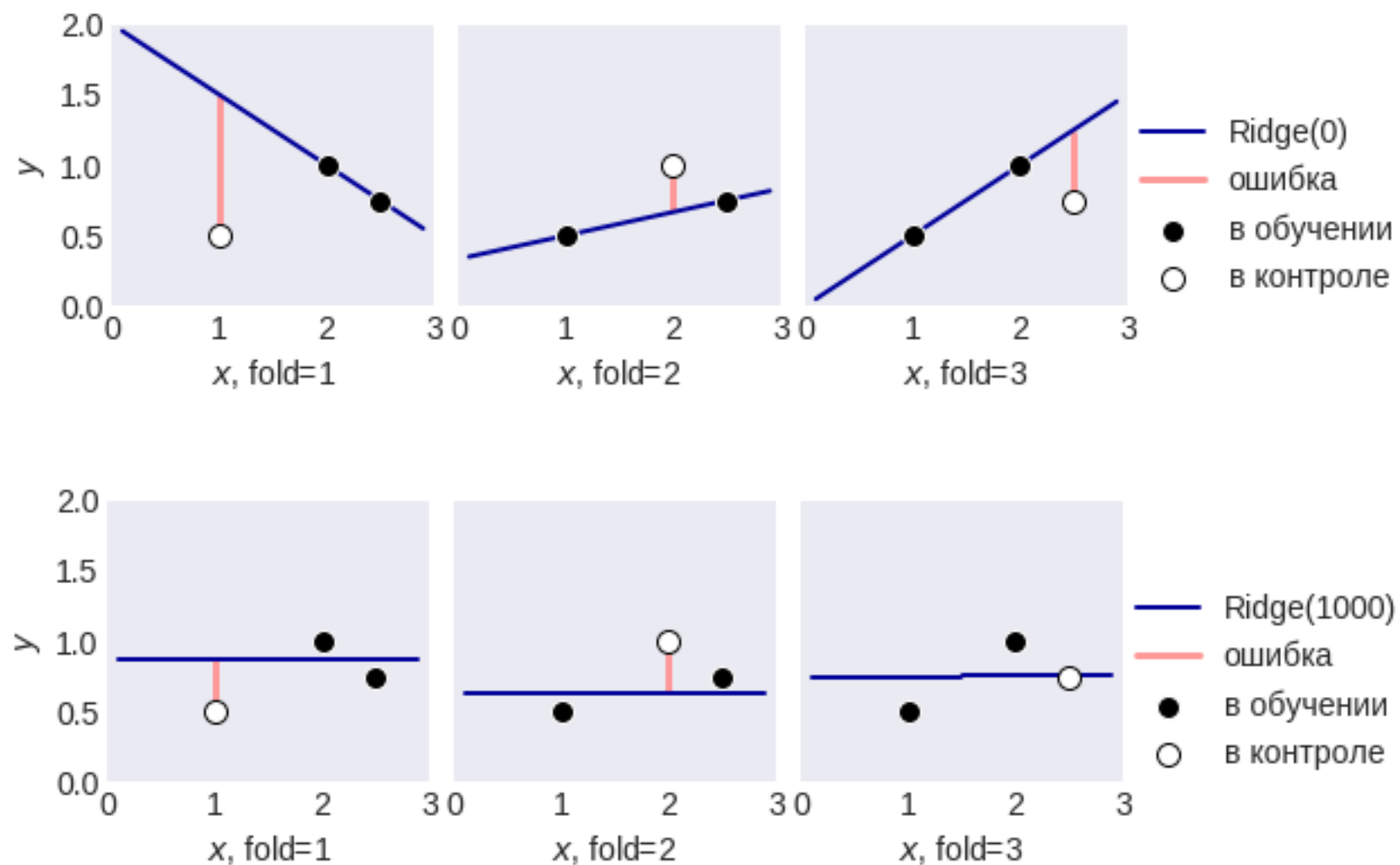
Перекры́стная проверка по фолдам (k-fold cross-validation)

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



k-fold CV = k-fold cross-validation

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 маленьких подвыборках**

Перекры́стная проверка по фолдам: не забываем правила

сохраняем пропорцию классов

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

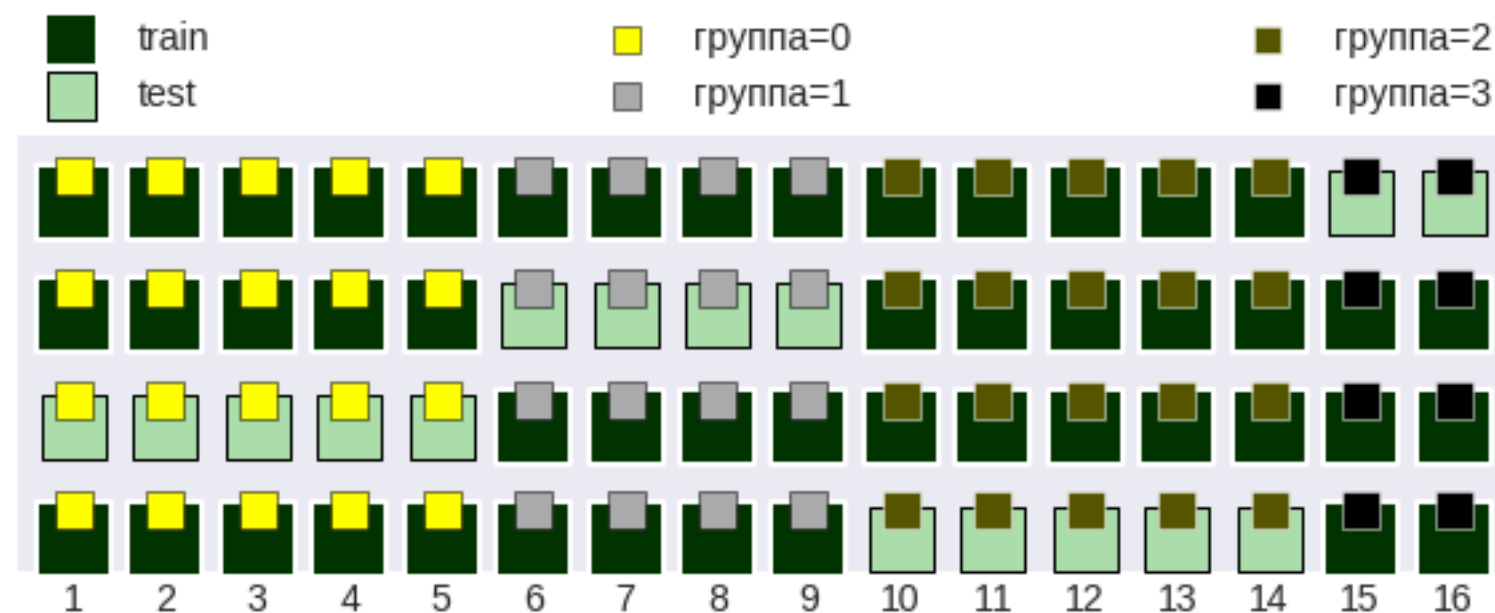


перемешиваем: `shuffle=True`

Перекрёстная проверка по фолдам: не забываем правила

не разбиваем группы

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

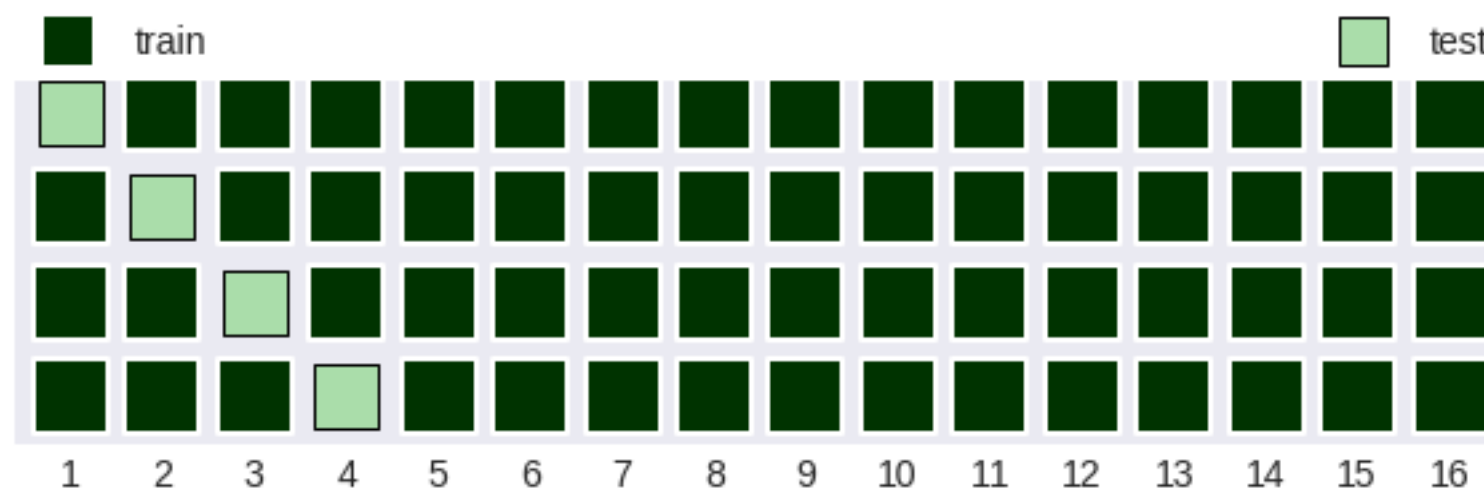


ещё есть `sklearn.model_selection.PredefinedSplit` – **разбиение индуцированное группами**

Leave-one out cross-validation (LOOCV)

k-fold при *k*=*m*

`sklearn.model_selection.LeaveOneOut`



показаны только первые 4 разбивки...

- **МОЖЕТ СЛИШКОМ ДОЛГО ВЫЧИСЛЯТЬСЯ**

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

Leave-one out cross-validation (LOOCV)

**Для справки: в статистике аналогичный метод «Складной нож» (jackknife)
для оценки параметров**

$$\bar{x}_{-i} = \frac{1}{m-1} \sum_{j \neq i} x_j$$

$$\bar{x} = \frac{1}{m} \sum_{i=1}^m \bar{x}_{-i}$$

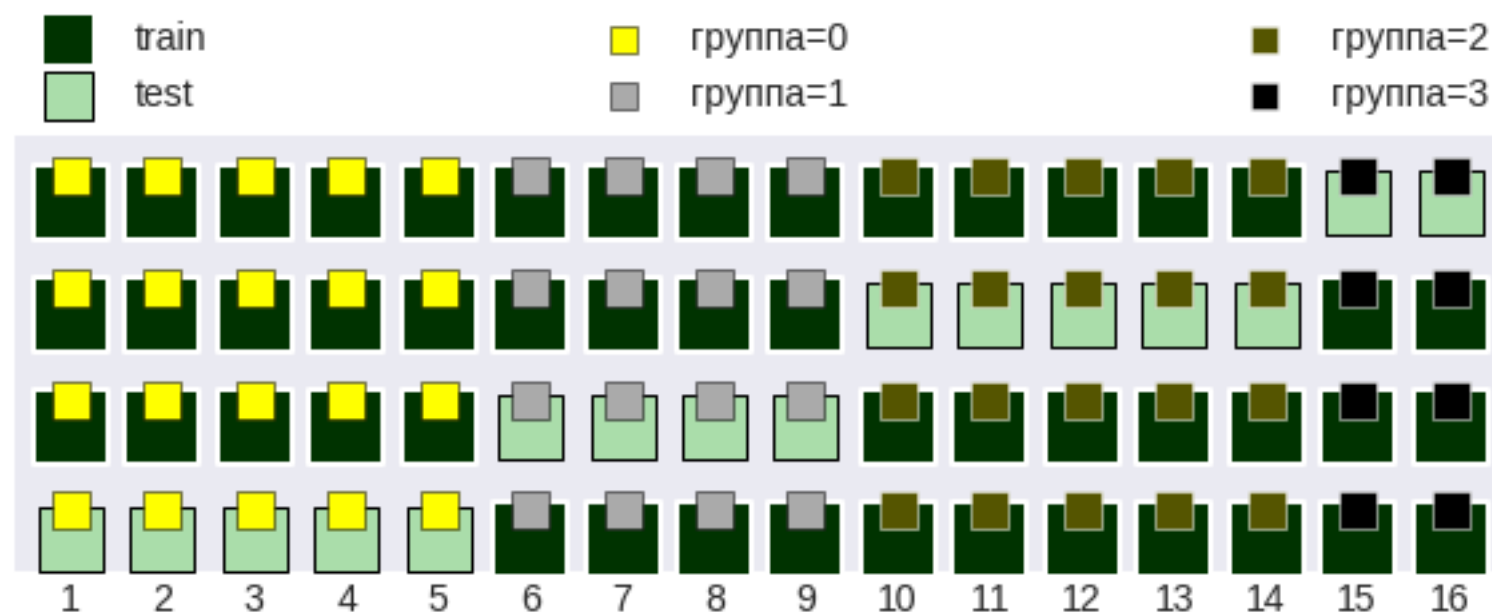
$$\overline{\text{var}} = \frac{m-1}{m} \sum_{i=1}^m (\bar{x} - \bar{x}_{-i})^2$$

на практике чаще применяют бутстреп-оценки

Контроль по группам: LeaveOneGroupOut

LeaveOneGroupOut: Контроль по одной группе

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



ТОНКОСТЬ:

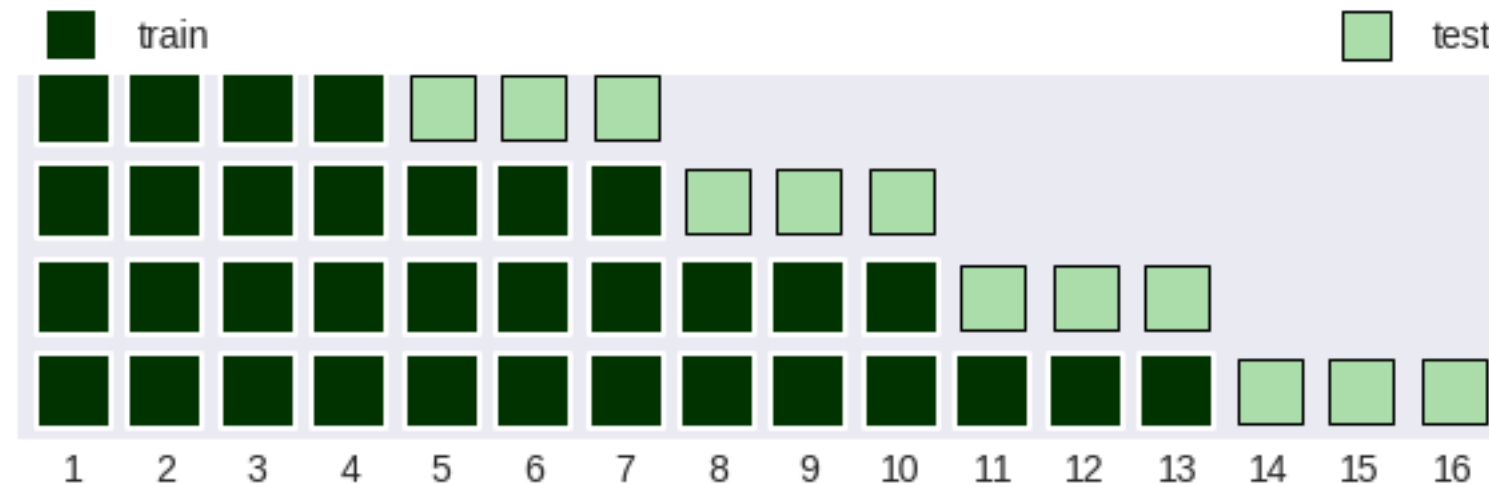
**при оценке ошибки можно (нужно?)
учитывать мощность групп**

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

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 \Rightarrow знаем как организовывать

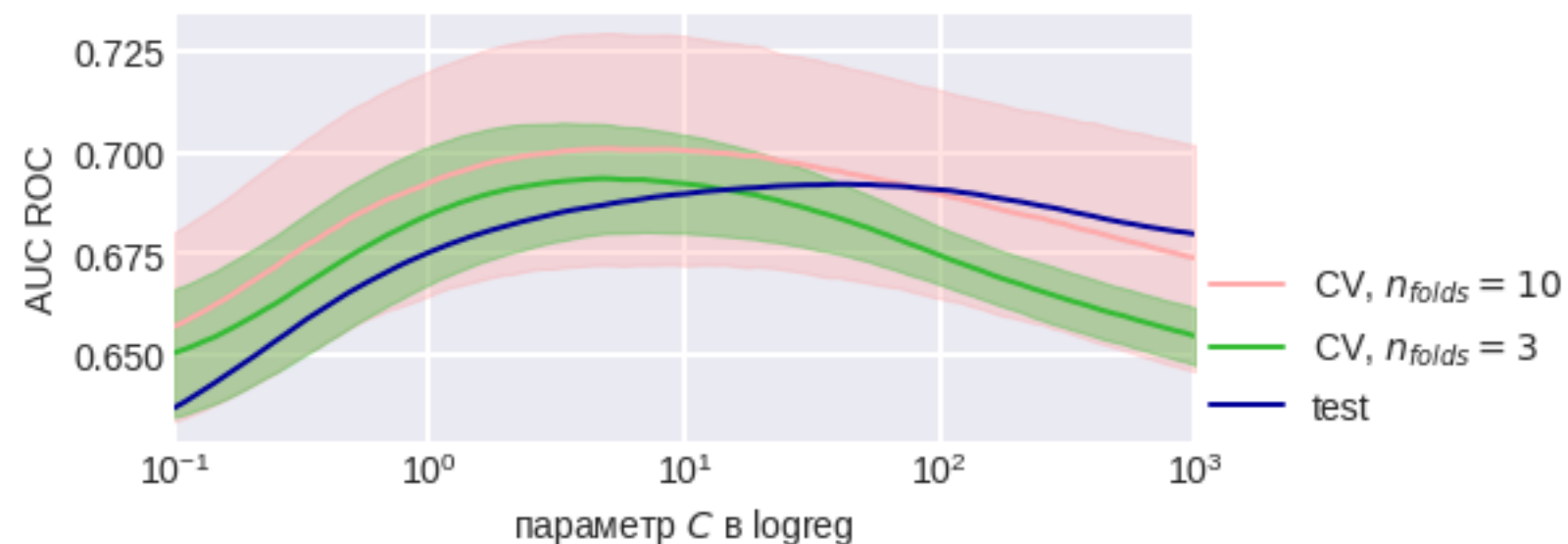
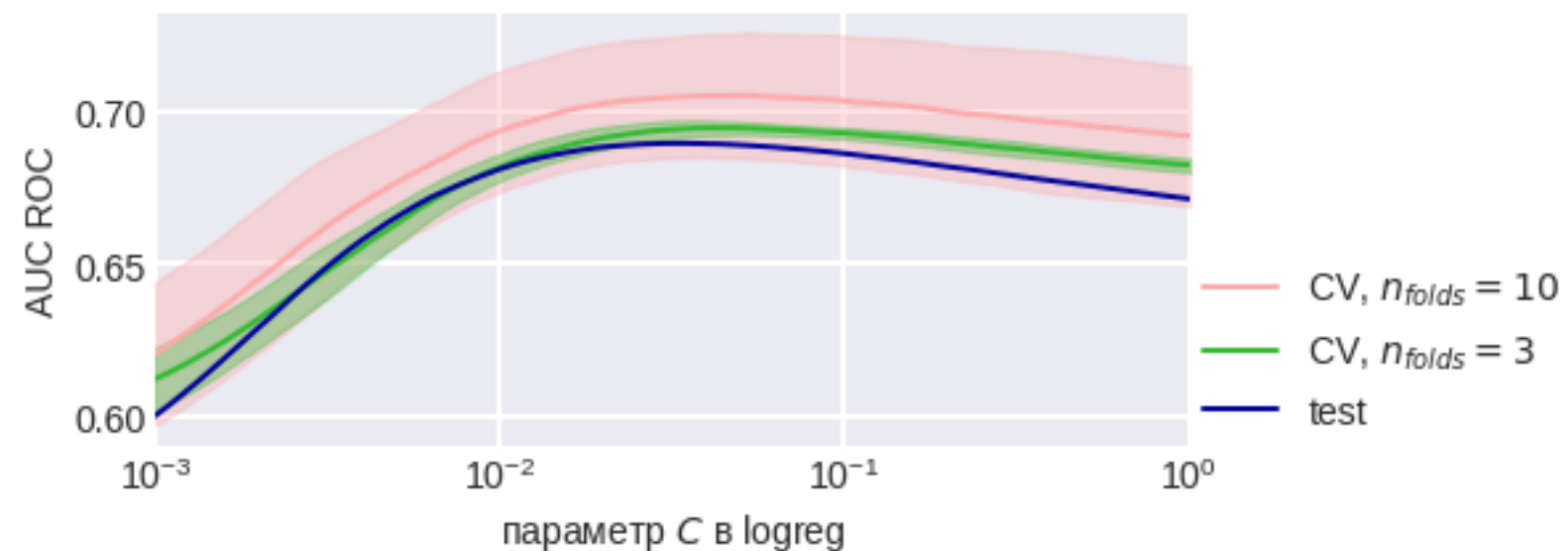
Терминология

user index

Training Data	in sample out of time
out of sample in time	out of sample out of time

time

Сравнение CV и настоящей ошибки



Две похожие задачи: есть и нет согласование CV / test

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 золотое 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.)

Январь							Февраль							Март							Апрель							Май							Июнь						
пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс
31	1	2	3	4	5	6	28	29	30	31	1	2	3	25	26	27	28	1	2	3	1	2	3	4	5	6	7	29	30	1	2	3	4	5	27	28	29	30	31	1	2
7	8	9	10	11	12	13	4	5	6	7	8	9	10	4	5	6	7	8	9	10	8	9	10	11	12	13	14	6	7	8	9	10	11	12	3	4	5	6	7	8	9
14	15	16	17	18	19	20	11	12	13	14	15	16	17	11	12	13	14	15	16	17	15	16	17	18	19	20	21	13	14	15	16	17	18	19	10	11	12	13	14	15	16
21	22	23	24	25	26	27	18	19	20	21	22	23	24	18	19	20	21	22	23	24	22	23	24	25	26	27	28	20	21	22	23	24	25	26	17	18	19	20	21	22	23
28	29	30	31	1	2	3	25	26	27	28	1	2	3	25	26	27	28	29	30	31	29	30	1	2	3	4	5	27	28	29	30	31	1	2	24	25	26	27	28	29	30
																												train							test						

Корректность локального контроля при использования кодировок



Разбиения корректные (например, OOT)

Локально:

Определяем кодировки на train_code

Кодируем train_code и valid

Регуляризуем на train_code

Валидируем на valid

Глобально:

Определяем кодировки на train

Кодируем train и test

Регуляризуем на train

Обучаем на train, предсказываем на test

 – можно пропустить

Где используется выбор CV-контроля

- **оценка качества модели**
было выше – оцениваем качество
- **настройка гиперпараметров**
перебор значений гиперпараметров
- **получение «честных» ответов на обучении**
как бы алгоритм отвечал,
если бы не обучался на этих объектах
 - контроль алгоритмов
 - ансамблирование (метапризнаки)
- **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

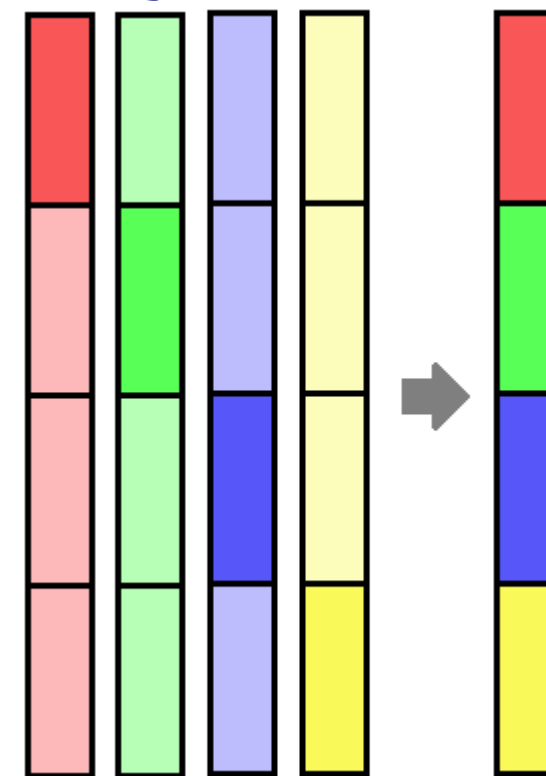
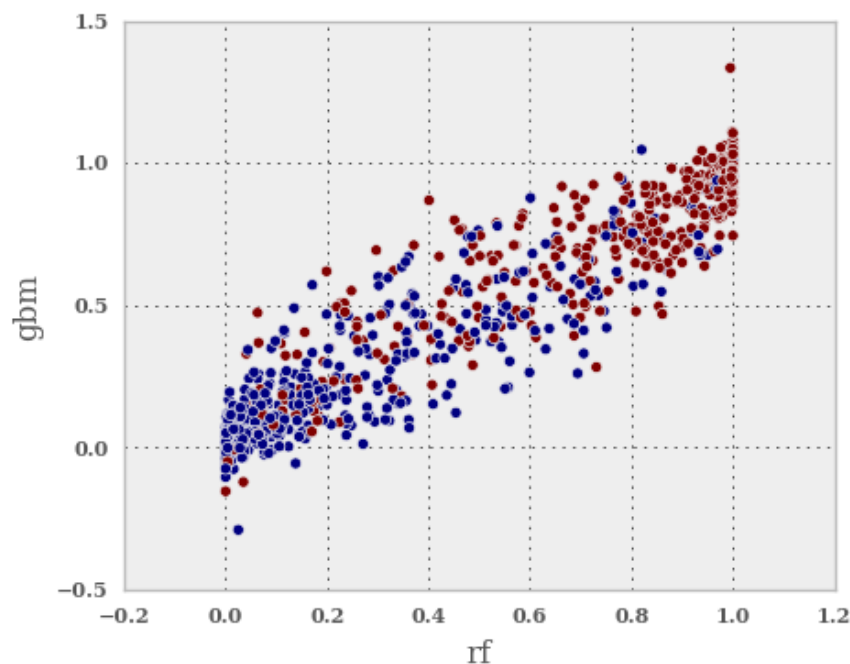
logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                 train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

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')
```

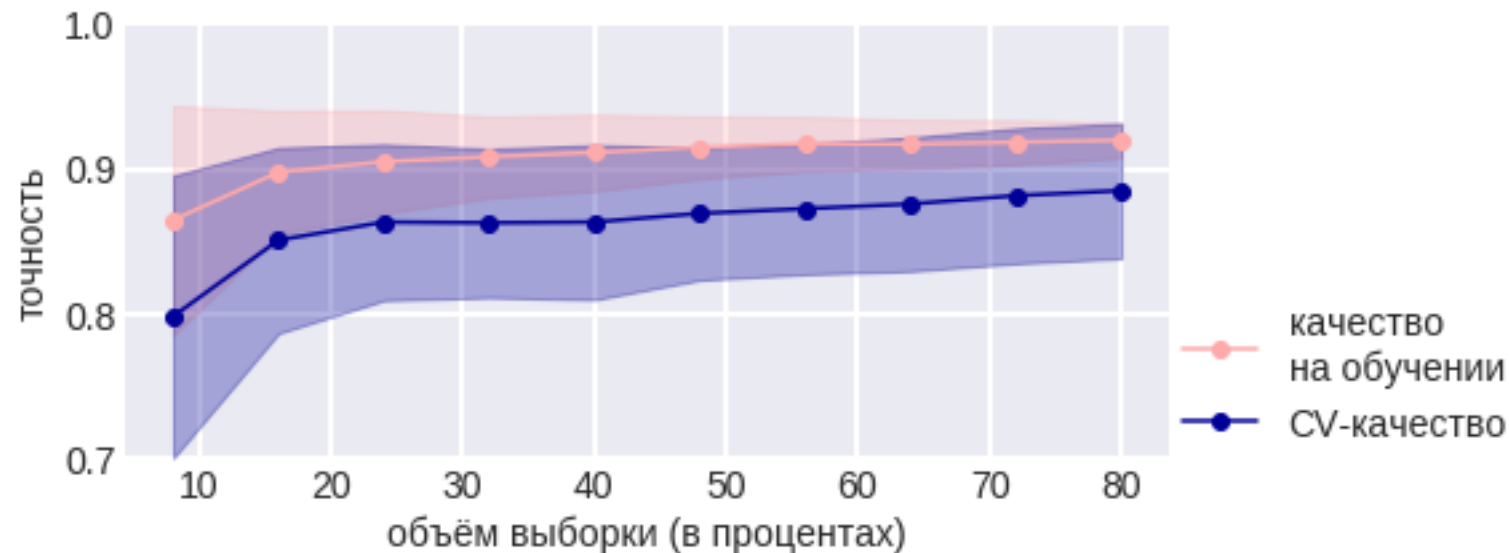
Кривые обучения (Learning Curves)

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

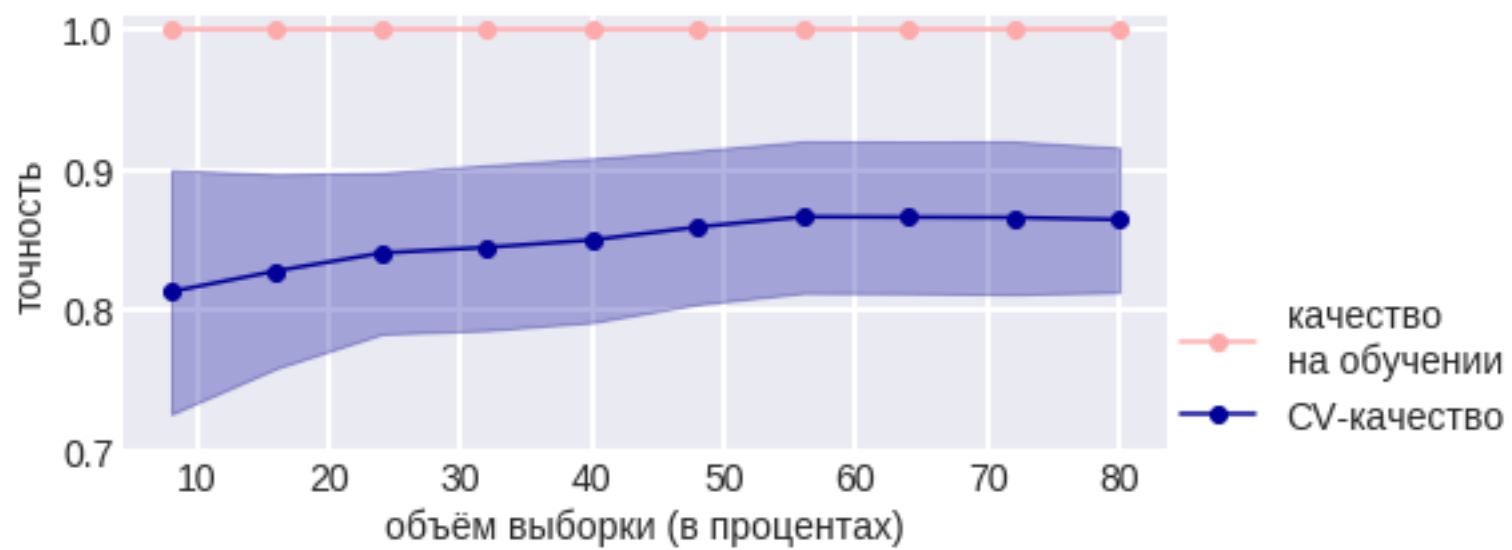
```
model_selection.learning_curve  
train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv,  
                                                         n_jobs=n_jobs, train_sizes=train_sizes)
```



Есть зазор между обучением и CV

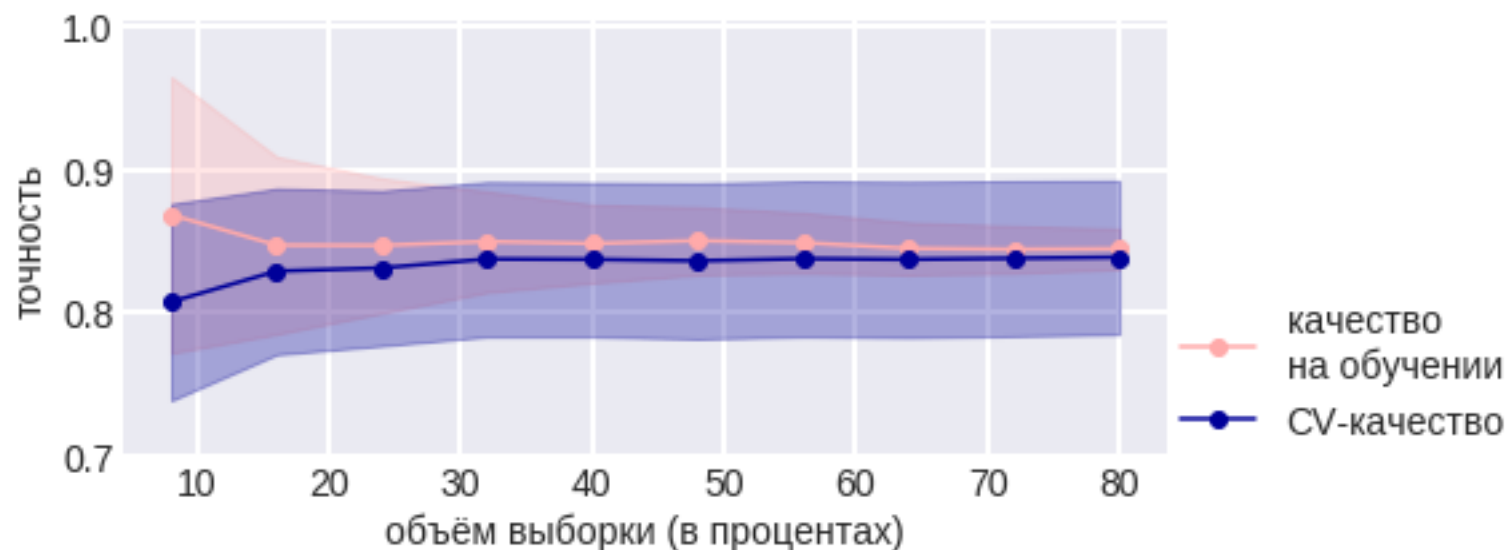
Тонкость: 100% – вся выборка, но здесь `test_size=0.2`

Кривые обучения (Learning Curves)



Переобучение:
100% качество на обучении!
1NN?

Нет выгоды от объёма!



Полная согласованность между обучением и CV,
но качество низкое

малая сложность модели

Переобучение / недообучение

overfitting

плохо, когда качество на обучении ↑,
а на контроле низкое или ↓

Причины переобучения: сложность модели, шум, нерепрезентативность

underfitting

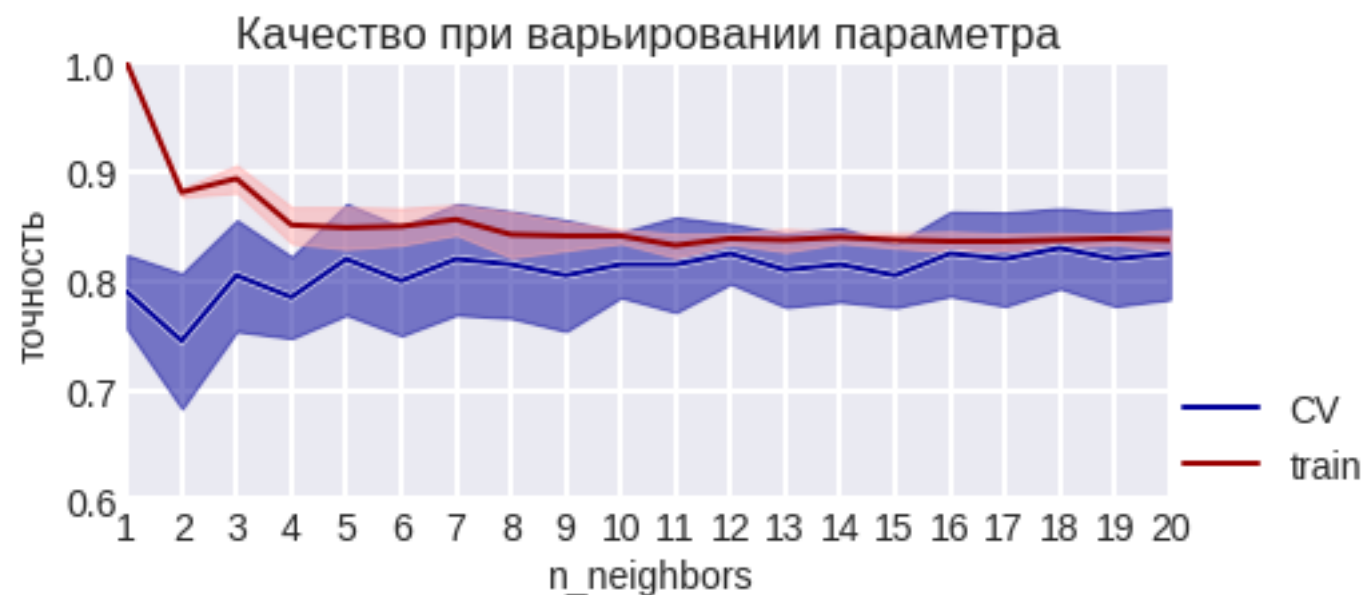
плохо, когда на обучении и контроле качество совпадает
и оно низкое

**дальше будет: более сложную модель / меньше регуляризацию,
больше тренировать,
больше данных,
другую модель**

Качество от параметров

Валидационная кривая (Validation Curve) показывает зависимость качества / ошибки при выбранной схеме контроля от значений гиперпараметров.

```
sklearn.model_selection.validation_curve
```



Настройка параметров (вообще)

**градиентные методы
(gradient-based)**

Метаоптимизация

**Байесовская оптимизация
(Bayesian model-based optimization)**

Перебор

Ручной перебор (Manual)

(Квази)полный перебор (Grid search)

Стохастические методы

Случайный поиск (Random search)

Эволюционные алгоритмы (evolutionary)

Замечание

Многие методы (например, RF, GBM) проще оптимизировать «вручную»!

дальше будем

Перебор значений гиперпараметров

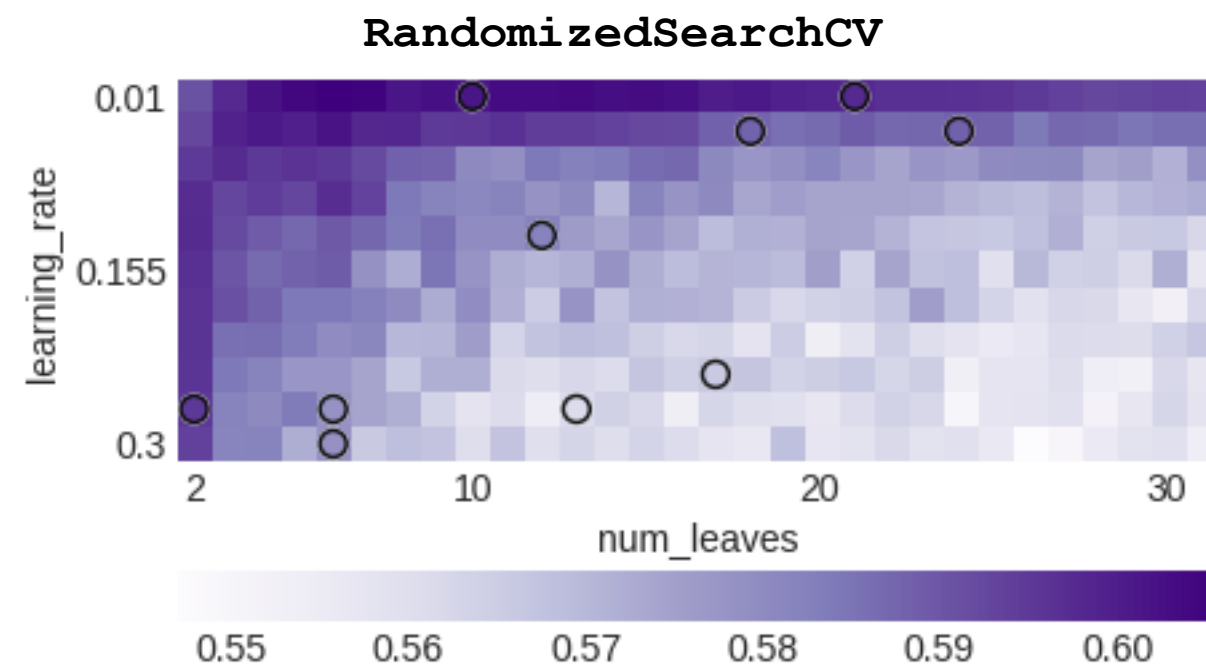
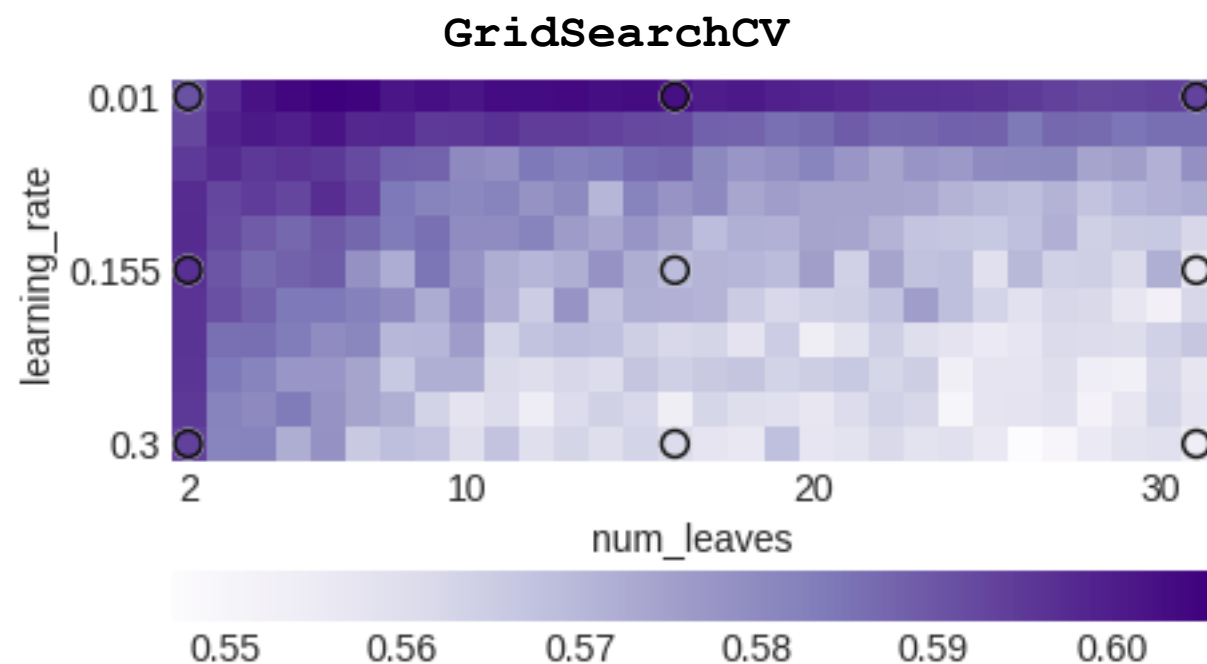
Делим данные на обучение и контроль (м.б. очень много раз)
При разных значениях параметров обучаемся и проверяем качество

```
from sklearn.model_selection import GridSearchCV
parameters = {'metric': ('euclidean', 'manhattan', 'chebyshev'),
              'n_neighbors': [1, 3, 5, 7, 9, 11], 'scoring': 'roc_auc'}
clf = GridSearchCV(estimator, parameters, cv=5)
clf.fit(X, y)
clf.cv_results_['mean_test_score']
```

	$k = 1$	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 11$
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», можно передавать распределения параметров)

Перебор параметров



```
import lightgbm as lgb
model = lgb.LGBMClassifier(n_estimators=100, subsample=0.75, colsample_bytree=0.75)

from sklearn.model_selection import GridSearchCV
parameters = {'num_leaves': np.arange(2, 32), 'learning_rate': np.linspace(0.01, 0.3, 11)}
clf = GridSearchCV(model, parameters, cv=5, scoring='roc_auc')
clf.fit(X, y)
```

Случайный поиск считают предпочтительным
(для «чёрных ящиков» с большим числом параметров)

Байесовская оптимизация * **(Bayesian model-based optimization)**

Идея: есть априорная вероятностная модель целевой функции (ошибки – the objective function) – «surrogate probability model»
в отличие от функции ошибки она будет её приближением,
её просто оптимизировать!

Способы представления SPM

Гауссовские процессы (Gaussian Processes)

Случайный лес (Random Forest Regressions)

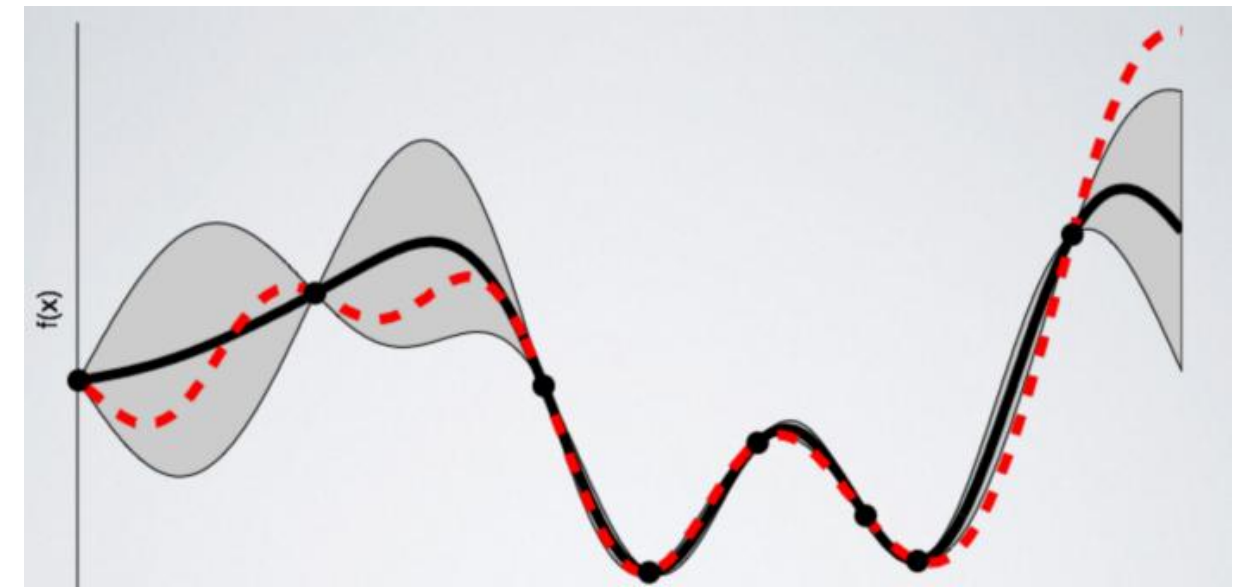
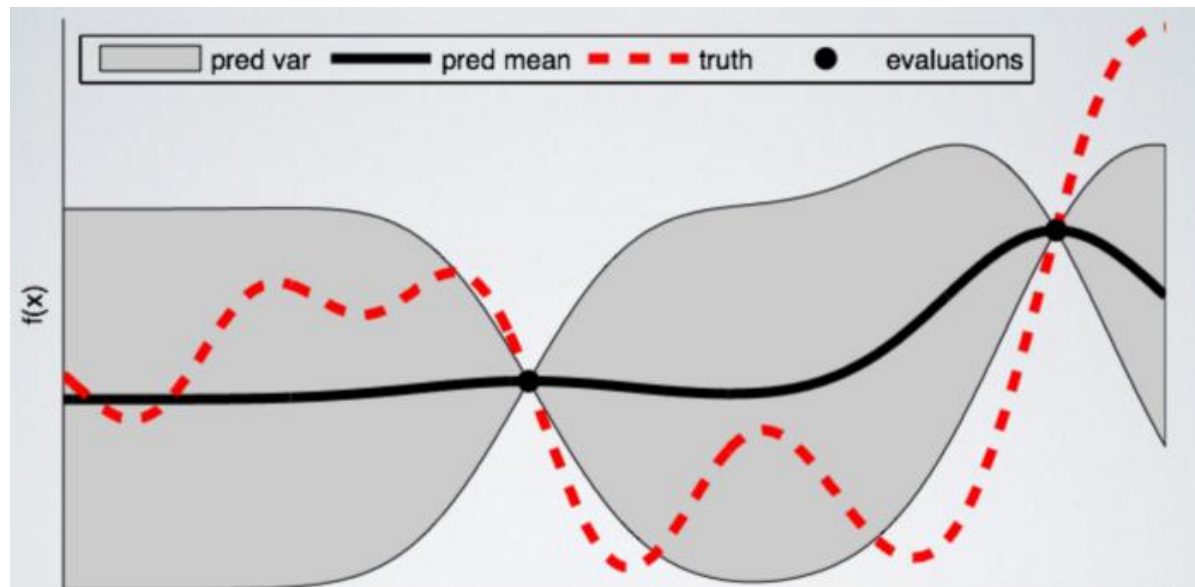
Tree Parzen Estimators (TPE)

...

находим argmax SPM / или точку для лучшего уточнения SPM
оцениваем значение в нём (ошибка при таких параметрах)
уточняем SPM

https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

Байесовская оптимизация *

**Algorithm 1** Sequential Model-Based Optimization**Input:** $f, \mathcal{X}, S, \mathcal{M}$ $\mathcal{D} \leftarrow \text{INITSAMPLES}(f, \mathcal{X})$ **for** $i \leftarrow |\mathcal{D}|$ **to** T **do** $p(y | \mathbf{x}, \mathcal{D}) \leftarrow \text{FITMODEL}(\mathcal{M}, \mathcal{D})$ $\mathbf{x}_i \leftarrow \arg \max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}, p(y | \mathbf{x}, \mathcal{D}))$ $y_i \leftarrow f(\mathbf{x}_i)$ \triangleright Expensive step $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$ **end for**

Байесовская оптимизация *

МОЖНО ПОЧИТАТЬ...

«Bayesian Optimization Primer»

https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf

Пример HyperOpt

<https://github.com/WillKoehrsen/hyperparameter-optimization/blob/master/Introduction%20to%20Bayesian%20Optimization%20with%20Hyperopt.ipynb>

«Algorithms for Hyper-Parameter Optimization»

<https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>

ссылки на библиотеки и код

<https://www.jeremyjordan.me/hyperparameter-tuning/>

Минутка кода

Как получить параметры модели

```
model.get_params()  
{'boosting_type': 'gbdt',  
 'class_weight': None,  
 'colsample_bytree': 0.75,  
 'importance_type': 'split',  
 'learning_rate': 0.1,  
 'max_depth': -1,  
 'min_child_samples': 20,  
 'min_child_weight': 0.001,  
 'min_split_gain': 0.0,  
 'n_estimators': 100,  
 'n_jobs': -1,  
 'num_leaves': 31,  
 'objective': None,  
 'random_state': None,  
 'reg_alpha': 0.0,  
 'reg_lambda': 0.0,  
 'silent': True,  
 'subsample': 0.75,  
 'subsample_for_bin': 200000,  
 'subsample_freq': 0}
```


Советы

Не забывайте указать метрику качества
`score`

а лучше несколько

Распараллеливание
`n_jobs=-1`

Можно сделать вычисления устойчивым к ошибкам
`error_score=0`

Оптимизировать целый пайплайн!

Итог

Правильная организация контроля – важная часть обучения

Помним 3 золотых правила:

- моделируем реальность,
- не допускаем утечек,
- случайность

CV для

- контроля качества
- формирования ответов на обучении
 - ансамблировании **будет**
- настройка гиперпараметров

Кривые качества

- validation curves (от значений)
- learning curves (от объёма выборки)

Ссылки

- презентация по sklearn

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

– есть код для различных организаций контроля

Только ослы
выбирают
до смерти



Бурданный осел