KARPOV.COURSES >>>
КОНСПЕКТ

Конспект > 5 урок > Обобщающая способность модели. Метод отложенной выборки. Кросс-Валидация.

>Оглавление 5 урока

>Обобщающая способность модели

Обычно, при построении модели хочется снизить или хотя бы предсказывать вероятность ошибки на тестовой выборке с объектами, о которых пока ещё ничего не известно (т.е на данных, содержащих объекты, которые модель еще не видела).

В обучении с учителем модель должна отлично справляться с новыми объектами, то есть обладать хорошей **обобщающей способностью**.

Обобщающая способность(generalization ability) - это способность модели выдавать правильные результаты не только для тренировочных примеров,

участвовавших в процессе обучения, а на всем множестве исходных данных (во всех сценариях, не только на тренировочных данных).

Но как измерить качество метода обучения? Как алгоритм будет вести себя на новых данных? Какая у него будет доля ошибок?

>Метод отложенной выборки

Самый простой способ оценить качество алгоритма — использование **отложенной выборки.** Представим, у нас есть некоторая выборка. Оценить работоспособность модели можно разбив выборку на две части (с красными и зелеными точками, к примеру), одна из которых (красная) будет использоваться для обучения модели, а другая (зеленая) для измерения качества метода обучения.

Выборка(sample, set) это лишь небольшая часть генеральной совокупности(всего объема данных), т.е нет никаких гарантий, что модель обученная на этой небольшой части будет работать качественно для всей генеральной совокупности. Но, если мы поделим выборку на 2 части(обучающую и тестовую), построим модель на **обучающей выборке**, убедимся, что она хорошо предсказывает ещё и **тестовую**(или контрольную) **выборку**, тогда у нас будет возможность считать, что у нашей модели хорошая **обобщающая способность.** А значит, модель будет отлично справляться с новыми объектами.

В идеале делить выборку на:

- **Обучающую выборку** (training sample) - это набор данных, который используется для разработки и настройки модели машинного обучения.
- **Валидационную выборку** (validation sample) - это набор данных, по которым осуществляется выбор наилучшей модели из множества моделей, построенных по обучающей выборке.
- **Тестовую выборку** (test sample) - это набор данных, который используется для окончательной оценки модели машинного обучения. Ещё её называют контрольной выборкой (в идеальном случае, независима от обучающей).

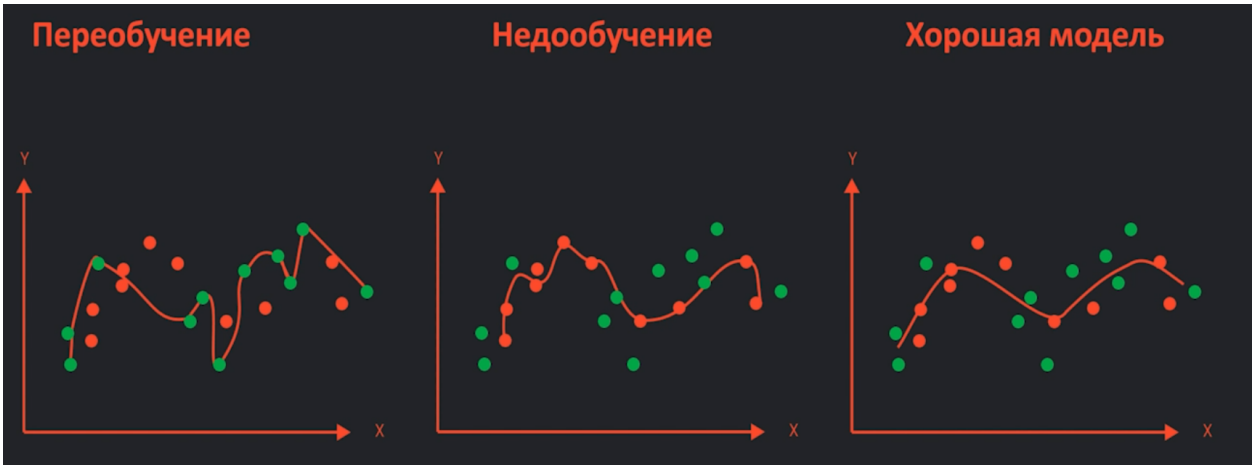
Только будьте внимательны, метод отложенной выборки подходит только если данных очень много.

>Переобучение и недообучение

При создании аналитических моделей следует избегать двух крайностей — недообучения (underfitting) и переобучения (overfitting). В первом случае алгоритм делает поспешные выводы, во втором строит функцию слишком близко к исходным данным.

Когда модель хорошо объясняет примеры из обучающей выборке(train), но ужасно работает на тестовой выборке(test) - мы говорим, что модель переобучилась.

То есть, нам необходимо находить такие модели, которые хорошо балансируют между качеством на обучающей и тестовой выборках.



А что если ни для одной модели не получилось хорошее качество всюду? Почему модели могут обладать плохой обобщающей способностью? Может быть несколько причин:

- Модель чрезмерно сложная или чрезмерно простая
- Неправильно спецификация задачи
- В какой-то из выборок много выбросов
- Данные засплитились сегментированно

>Кросс-валидация

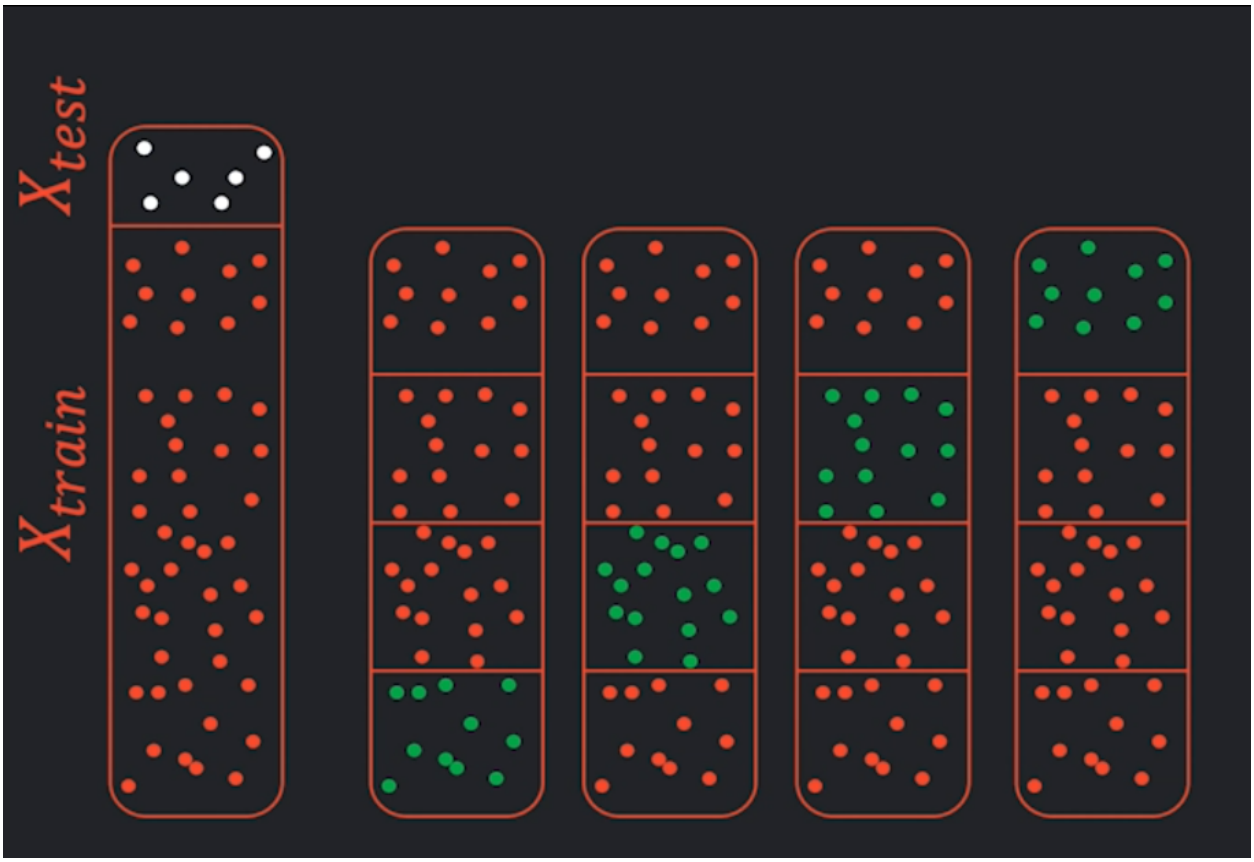
Необходимо иметь метод(дизайн) деления выборки на train/test и дизайн того как замерять качество моделей, чтобы все проблемы с данными учитывались. Т.е

немного модифицировать отложенную выборку, чтобы более правдоподобно показывать обобщающую способность моделей.

Кросс-валидация (перекрестная проверка, скользящий контроль) – это метод оценки

Представим, что у нас есть некий набор данных.

- Делим исходные данные на **k** фолдов(частей) примерно одинаково размера.
- Затем на (**k** - 1) блоке обучаем модели, а оставшуюся часть используем для тестирования.
- Повторяем процедуру **k** раз, при этом на каждой итерации для проверки выбирается новый блок, а обучение производится на оставшихся.
- Замеряем **k** средних ошибок на валидационной выборке (ее мы отложили ранее) и если их среднее/их распределение нас устраивает, то строим финальную модель



Но как получить финальную модель?

Либо выбрать лучшую модель, обучить её на всём объеме данных и потом замерить её качество на валидации. Либо усреднить все модели, которые получаются во время кросс-валидации. Либо выбрать ту модель, которая лучше всего справилась с данными на валидации.

По умолчанию **k** берут равным 5 или 10, но можно принимать и любое другое значение.

При небольшом числе разбиений оценки будут смещенными, поскольку размер обучения будет существенно отличаться от размера полной выборки, и обученные в процессе кросс-валидации алгоритмы могут иметь более низкое качество. Дисперсия оценок будет небольшой

>Переобучение модели в python

Пусть имеем две модели: полиноминальную и обыкновенную немодифицированную линейную без свободного коэффициента

$$\alpha_1(\chi) = \beta_1 \cdot \delta + \beta_2 \cdot \delta^2 + ... + \beta_{25} \cdot \delta^{25} + \beta_0$$

$$\alpha_2(\chi) = \beta_1 \cdot \delta$$

Давайте сгенерируем выборку (X, Y) следующим образом. Скажем, что X - какие-то случайные числа, равномерно лежащие в отрезке [0, 20]. Ответы (вектор таргетов Y) - удвоенные X-ы, сгенерированные с некоторым шумом, распределенным тоже равномерно на отрезке [-6, 6].

Разделим выборку на тренировочную и тестовую (в пропорции 4 к 1), и проверим, в каком случаем модель ошибается сильнее: при сложном моделировании или простом?

```
import numpy as np

X = np.array(sorted([20*x for x in np.random.rand(120,1)]))
Y = np.array([2 * el + np.random.choice([-1, 1]) * 6 * np.random.rand() for el in X])
```

Теперь разобьем нашу выборку на тренировочную и тестовую выборку с помощью train_test_split из библиотеки `sklearn.model_selection`:

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4)
```

Для разбиения выборки на **k** частей также можно воспользоваться методом KFold() из той же библиотеки `sklearn.model_selection`. Ещё данной библиотеке существует метод StratifiedKFold(), который производит стратифицированное разделение выборки на фолды. Кроме того, можно воспользоваться методом cross_val_score(), который принимает на вход выбранный алгоритм построения и сразу выдает результаты.

Теперь, сохраним упорядоченные индексы наших элементов(это далее понадобится для визуализации)

```
index_argsort = np.argsort(X_train.reshape(72,))
```

Теперь построим линейную модель, т.е попытаемся связать X с Y с помощью LinearRegression из библиотеки `sklearn.linear_model`:

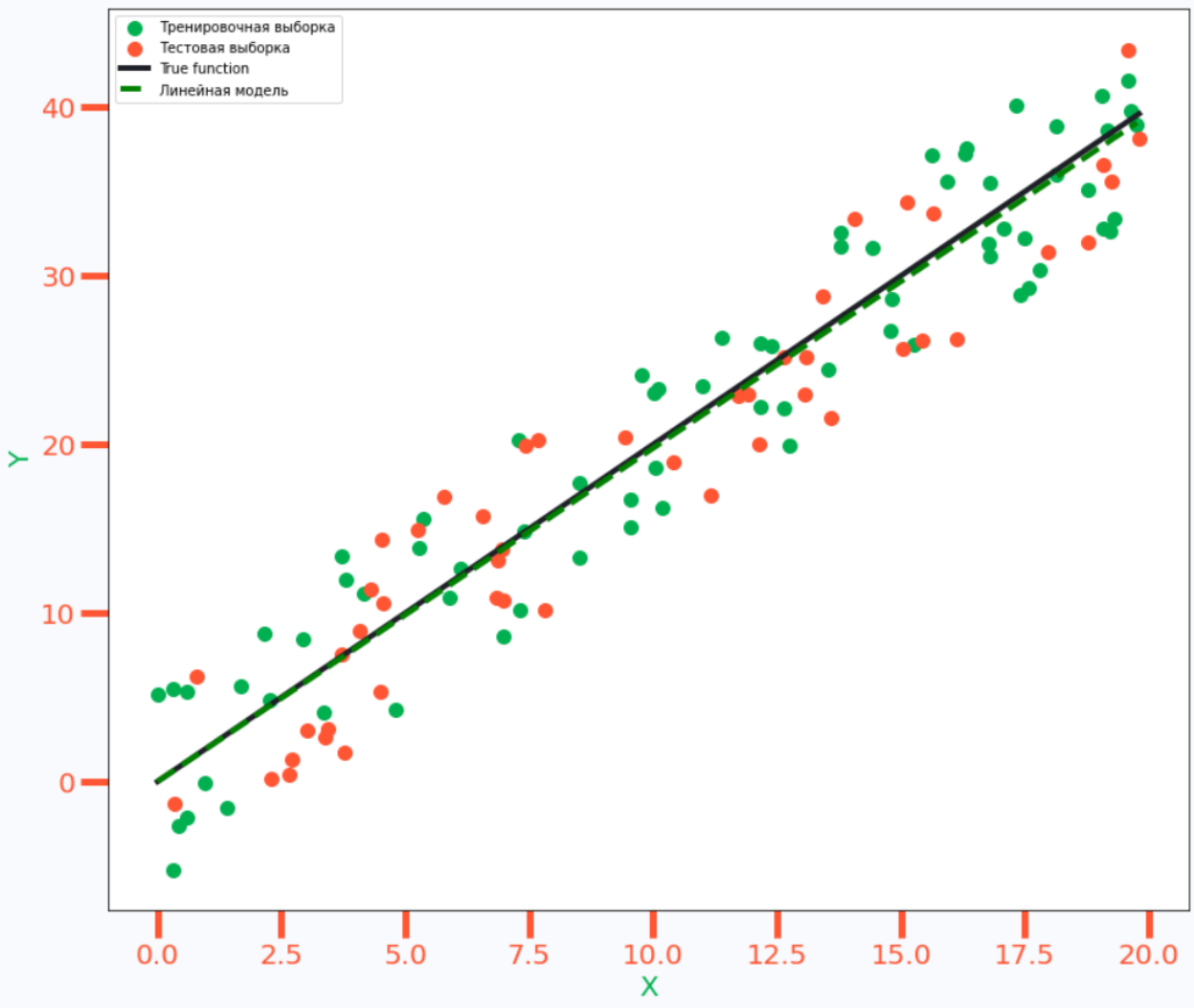
```
from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept=False) # fit_intercept наличие свободной переменной
model.fit(X_train, Y_train) # Передаём тренировочную часть
```

Теперь настроим красивые дефолтные настройки matplotlib и визуализируем полученные данные. Создадим полотно вызвав объект `figure`.

```
import matplotlib.pyplot as plt

# Создадим полотно вызвав объект figure
fig = plt.figure()
fig.set_size_inches(14, 12)

# Отметим точки
plt.scatter(X_train, Y_train, c='#00B050', s=100)
plt.scatter(X_test, Y_test, c='#FF5533', s=100)
```

```
# Построим идеальное распределение
plt.plot(X, [2*x for x in X], '#1E2027', linewidth=4)

plt.plot(X_train[index_argsort],
         model.predict(X_train[index_argsort]),
         '--g', linewidth=4)

# Установим легенды и подпишем оси
plt.legend(['Тренировочная выборка', 'Тестовая выборка', 'True function',
            'Линейная модель'], loc = 'upper left')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Получим такой график:



Линейная модель почти точно совпала с черной, идеальной моделью(True function). С нашей моделью всё отлично, она и не недообучилась и не переучилась.

Теперь построим полиномиальную модель. Для этого нужно немного обработать данные:

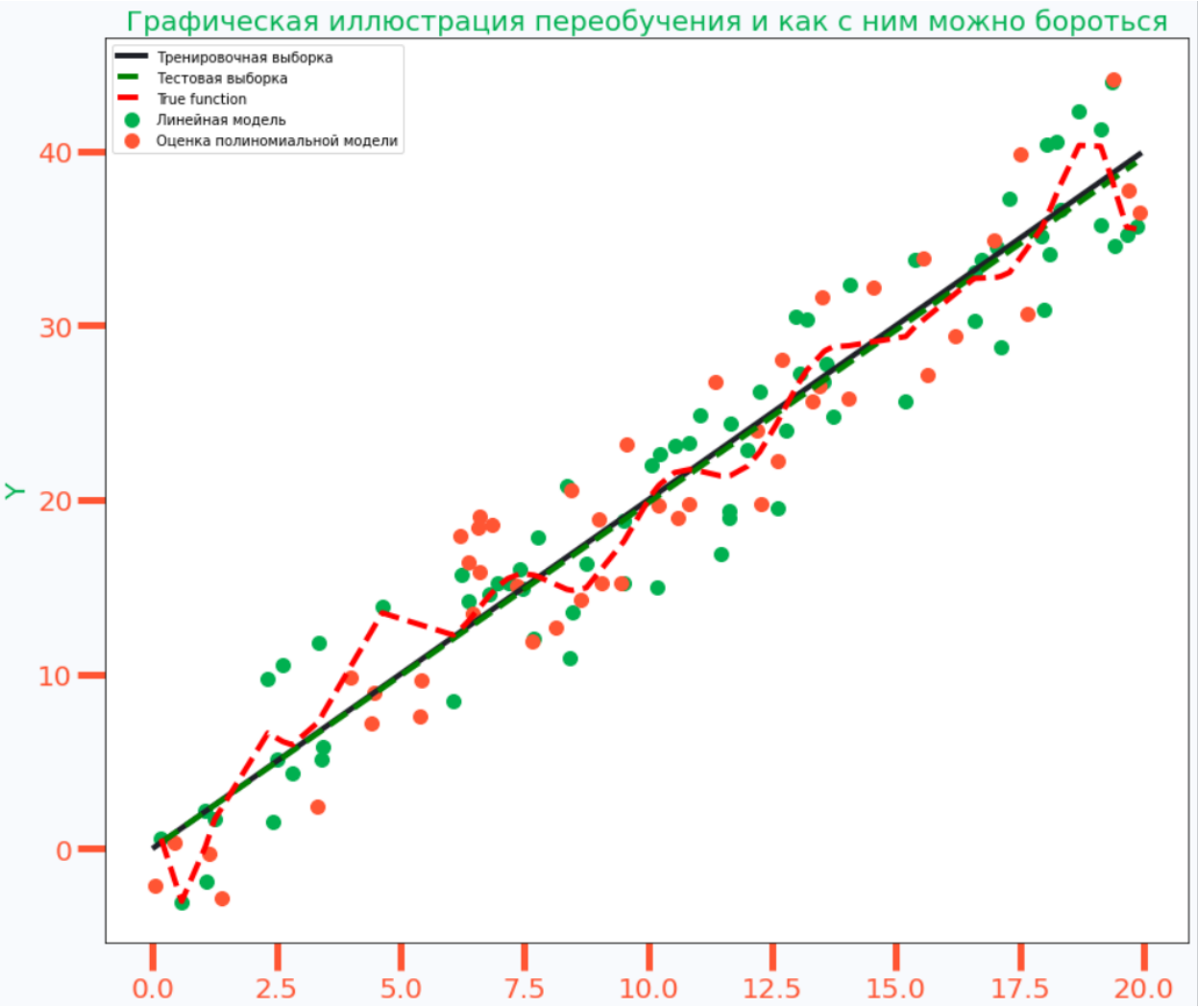
```
X_pol = X_train.copy()

for k in range(2, 26):
    X_pol = np.append(X_pol,
                      np.array([x**k for x in X_pol[:, 0]]).reshape(72, -1),
                      axis=1)
# Теперь мы получили массив из 25 столбцов

# Создадим новую модель model_pol
model_pol = LinearRegression(normalize=True)
model_pol.fit(X_pol, Y_train)
```

Мы построили полиномиальную модель, с 25 признаками(плюс свободный коэфф.). Теперь изобразим всё полученное нами на графике:

```
import matplotlib.pyplot as plt

# Создадим полотно вызвав объект figure
fig = plt.figure()
fig.set_size_inches(14, 12)

# Отметим точки
plt.scatter(X_train, Y_train, c='#00B050', s=100)
plt.scatter(X_test, Y_test, c='#FF5533', s=100)
plt.plot(X, [2*x for x in X], '#1E2027', linewidth=4)
plt.plot(X_train[index_argsort],
         model_just_linear.predict(X_train[index_argsort]),
         '--g', linewidth=4)
# Добавим предсказание к точкам созданного ранее model_pol
plt.plot(X_train[index_argsort],
         model_pol.predict(X_pol[index_argsort]),
         '--r', linewidth=4)


plt.legend(['Тренировочная выборка', 'Тестовая выборка', 'True function',
            'Линейная модель', 'Оценка полиномиальной модели'],
            loc = 'upper left')
plt.xlabel('X')
plt.ylabel('Y')
```

```
plt.title('Графическая иллюстрация переобучения и как с ним можно бороться')

plt.show()
```

Получим такой график:



Наша полиномиальная модель переобучилась. Она далека от идеальной модели, которую мы ожидали получить. Модель сильно подстроилась под тренировочную выборку, это сигнал о том, что модель переобучилась. Можно сделать вывод, что когда данные распределены просто, то сложную модель строить не надо.

Чтобы убедиться, что полиномиальная модель переобучилась - сравним среднюю квадратическую ошибку (MSE) первой и второй модели

```
# обыкновенная немодифицированная линейная модель без свободного коэффициента
np.mean((model.predict(X_train)-Y_train)**2)
# полиномиальная модель
np.mean((model_pol.predict(X_pol)-Y_train)**2)
Output:
10.662141223891416

26.88372246737598
```

На тренировочных данных полиномиальная модель работает лучше, но давайте теперь сгенерируем полиномиальные признаки для **тестовых данные** и посчитаем среднюю квадратическую ошибку**:**

```
X_pol_test = X_test.copy()

for k in range(2, 26):
    X_pol_test = np.append(X_pol_test,
                           np.array([x**k for x in X_pol_test[:, 0]]).reshape(48, -1),
                           axis=1)
np.mean((model_pol.predict(X_pol_test)-Y_test)**2)

Output:
682.529835293346
```

Видно, что на тестовых данных у полиномиальной модели появляются проблемы. Ошибок почти в 30 раз больше, чем на тренировочных данных. Это явный сигнал о том, что наша полиномиальная модель переобучается

>Замерим качество на Кросс-валидации

Представим, у нас есть модель с небольшой средне квадратической ошибкой на тренировочных данных, но у нас нет никакой информации о качестве модели на новых данных и мы хотим прогнать нашу модель через метод кросс-валидации перед тем как выкатывать ее в прод.

Чтобы реализовать метод кросс-валидации разобьём выборки на 10 частей методом KFold() из библиотеки `sklearn.model_selection`.

```
from sklearn.model_selection import KFold
```

```
kf = KFold(n_splits=10, shuffle=True, random_state=33)
kf.split(x)   # x это датафрейм
```

`kf.split(x)` - передает список различных индексов для тренировочной и тестовой выборок(причем так, чтобы тест нигде не повторялся и не пересекался). Давайте для каждого набора индексов создадим новые объекты X_train, X_test, Y_train, Y_test

```
# Тут мы храним ошибки модели
losses_test = []
losses_train = []

for train_index, test_index in kf.split(x):
    X_train, X_test = x.values[train_index], x.values[test_index]
    Y_train, Y_test = y.values[train_index], y.values[test_index]
    # На каждой итерации строим лин.регрессию на train
    model = LinearRegression()
    model.fit(X_train, Y_train)
    # Замеряем качество модели(RMSE)
    losses_test.append(np.mean((model.predict(X_test)-Y_test)**2)**(1/2))
    losses_train.append(np.mean((model.predict(X_train)-Y_train)**2)**(1/2))
```

Что мы должны сделать, чтобы понять, переобучилась ли наша модель? Мы усредним ошибки на тестовых и тренировочных данных и сравним:

```
np.mean(losses_test)
np.mean(losses_train)

Output:
91114.72921295118
96610.06854595507
```

Ошибка на кросс-валидации не сильно отличаются. Подозрения, что модель переобучилась пропадают.