In [9]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.metrics import accuracy_score
from sklearn import datasets
import scipy.stats as sps
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings
sns.set('notebook', style='whitegrid', font_scale=1.3)
```

Обзор методов поиска гиперпараметров

В машинном обучении существуют различные семейства моделей, выбор которых зависит от задачи, которая перед вам стоит.

Предположим, вы выбрали семейство моделей (например, линейная регрессия). Теперь вам нужно настроить гиперпараметры данной модели. Для начала нужно определить, что в машинном обучении называют гиперпараметрами.

Гиперпараметры - это параметры модели, которые задаются ДО ее обучения, т.е. они не являются обучаемыми параметрами и не изменяются в ходе обучения модели.

Пример: выбор способа регуляризации в модели линейной регрессии. Строго говоря, выбор семейства моделей также является гиперпараметром. В данном ноутбуке мы изучим различные методы подбора гиперапараметров, затем применим их к ридж-регрессии.

В контексте линейной регрессии мы будем работать с гиперпараметром, который является коэффициентом регуляризации. Этот гиперпараметр относится к группе, которую мы в дальнейшем для удобства будем называть красными гиперпараметрами:

- красные гиперпараметры - гиперпараметры, ограничивающие модель. Если увеличить значение гиперпараметра такого типа, то можно снизить переобучение модели. И наоборот, уменьшение значений даст модели возможность лучше улавливать зависимости в данных.
  Пример: коэффициент регуляризации в Ridge или Lasso регрессии. Чем больше данный коэффициент, тем меньше зависимостей улавливает модель, именно поэтому при больших значениях параметра регуляризации коэффициенты признаков зануляются.

О других видах гиперпараметров вы узнаете позже.

Поиск гиперпараметров для Ridge регрессии

Будем предсказывать цены квартир в Бостоне при помощи линейной регрессии с l_2-регуляризацией.

Вспомним, что в Ridge-регрессии мы оптимизируем $||Y - X\theta||_2^2 + \alpha \cdot ||\theta||_2^2$, где Y - истинные значения целевой переменной, X - матрица "объект-признак", θ - параметр модели, α - параметр регуляризации.

In [3]:

```
boston = datasets.load_boston()  # данные о ценах квартир в Бостоне
X = pd.DataFrame(data=boston['data'], columns=boston['feature_names'])
y = boston['target']
```

Описание датасета:

In [5]:

```
1 print(boston['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median V
alue (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over
25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds r
iver; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1
940
        - DIS      weighted distances to five Boston employment centre
s
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of bla
cks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (ht
tps://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at
Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedon
ic
prices and the demand for clean air', J. Environ. Economics & Manageme
nt,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diag
nostics
...', Wiley, 1980.   N.B. Various transformations are used in the tabl
e on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning pap
ers that address regression
```

```
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influ
ential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learn
ing. In Proceedings on the Tenth International Conference of Machine L
earning, 236-243, University of Massachusetts, Amherst. Morgan Kaufman
n.
```

In [7]:

```
1 X.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

Разберемся, что означают эти признаки:

- `CRIM` - уровень преступности на душу населения
- `ZN` - доля жилой земли на каждые 25 000 км^2
- `INDUS` - доля неторговых площадей в районе
- `CHAS` - индиактор того, что неподалеку река Чарльз
- `NOX` - концентрация оксида азота
- `RM` - среднее количество комнат в доме
- `AGE` - доля зданий, построенных до 1940 года
- `DIS` - взвешенные расстояния до пяти бостонских центров занятости
- `RAD` - индекс доступности радиальных магистралей
- `TAX` - ставка налога на полную стоимость имущества за 10 000 долларов
- `PTRATIO` - количество учеников на учителя
- `B` - доля темнокожих
- `LSTAT` - доля населения с низким статусом

Получить гиперпараметры модели можно при помощи метода `get_params()` .

In [22]:

```
model = Ridge()
model.get_params()
```

Out[22]:

```
{'alpha': 1.0,
 'copy_X': True,
 'fit_intercept': True,
 'max_iter': None,
 'normalize': False,
 'random_state': None,
 'solver': 'auto',
 'tol': 0.001}
```

Самым важным гиперпараметром тут является `alpha` — коэффициент регуляризации. Именно его мы и будем искать при помощи техники `GridSearch` — полный перебор по сетке.

Суть перебора по сетке заключается в следующем: вы задаете область значений гиперпараметров, которые хотите оптимизировать, обучаете все модели, которые возможно получить в рамках заданного пространства поиска гиперпараметров. Затем выбираете из данных моделей ту, которая дала наилучшее качество на кросс-валидации. Этой модели соответствует конкретный набор значений гиперпараметров — его мы и ищем.

В sklearn вам нужно просто указать одномерную сетку отдельно для каждого из гиперпараметров, построение многомерного пространства поиска оптимальных гиперпараметров спрятано в реализации метода. Применим `GridSearch` на примере:

In [24]:

```
# задаем пространство поиска
parameters_grid = {
    'alpha' : np.linspace(0.00001, 2, num=1000)
}

# задаем стратегию кросс-валидации
ss = ShuffleSplit(n_splits=5, test_size=0.25, random_state=0)

# задаем модель
model = Ridge()

# определяем поиск по сетке
gs = GridSearchCV(
    # модель для обучения, в нашем случае Ridge
    estimator=model,
    # сетка значений гиперпараметров
    param_grid=parameters_grid,
    # метрика качества, берем MSE
    scoring='neg_mean_squared_error',
    # GridSearch отлично параллелится, указываем количество параллельных джоб
    n_jobs=-1,
    # стратегия кросс-валидации
    cv=ss,
    # сообщения с логами обучения: больше значение - больше сообщений
    verbose=10,
    # значение, присваиваемое scorer в случае ошибки при обучении
    error_score='raise'
)
```

In [25]:

```
%%time
# выполняем поиск по сетке
gs.fit(X, y)
```

```
Fitting 5 folds for each of 1000 candidates, totalling 5000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1848s.) Setting b
atch_size=2.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done  61 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.0228s.) Setting b
atch_size=34.
[Parallel(n_jobs=-1)]: Done  86 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 212 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-1)]: Done 722 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 1300 tasks      | elapsed:    3.7s
[Parallel(n_jobs=-1)]: Done 1878 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done 2524 tasks      | elapsed:    4.1s
[Parallel(n_jobs=-1)]: Done 3170 tasks      | elapsed:    4.5s
[Parallel(n_jobs=-1)]: Done 3884 tasks      | elapsed:    4.9s

CPU times: user 2.48 s, sys: 188 ms, total: 2.66 s
Wall time: 5.29 s

[Parallel(n_jobs=-1)]: Done 5000 out of 5000 | elapsed:    5.3s finish
ed
```

Out[25]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=5, random_state=0, test_size=0.2
5, train_size=None),
             error_score='raise',
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=Tru
e,
                             max_iter=None, normalize=False, random_st
ate=None,
                             solver='auto', tol=0.001),
             iid='warn', n_jobs=-1,
             param_grid={'alpha': array([1.00000000e-05, 2.01199199e-0
3, 4.01398398e-03, 6.01597598e-03,
       8.017...
       1.96196215e+00, 1.96396414e+00, 1.96596614e+00, 1.96796813e+00,
       1.96997012e+00, 1.97197211e+00, 1.97397410e+00, 1.97597610e+00,
       1.97797809e+00, 1.97998008e+00, 1.98198207e+00, 1.98398406e+00,
       1.98598606e+00, 1.98798805e+00, 1.98999004e+00, 1.99199203e+00,
       1.99399402e+00, 1.99599602e+00, 1.99799801e+00, 2.00000000e+0
0])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='neg_mean_squared_error', verbose=10)
```

Итак, мы выполнили полный перебор по сетке. Давайте посмотрим, какие атрибуты есть у `GridSearch` и чему они равны. Атрибут `cv_results_` возвращает информацию о времени выполнения обучения и подробности о значениях метрик. Значения данного атрибута удобнее отображать через `pd.DataFrame` .

In [31]:

```
results = gs.cv_results_
pd.DataFrame(results)
```

Out[31]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	
0	0.008842	0.000738	0.002879	0.002724	1e-05	{'alph
1	0.009286	0.001412	0.002396	0.001710	0.00201199	0.00201199199
2	0.009323	0.003790	0.002253	0.000697	0.00401398	0.00401398398
3	0.003256	0.000623	0.001309	0.000101	0.00601598	0.00601597597
4	0.002411	0.000136	0.001086	0.000054	0.00801797	0.00801796796
...	...	...	...	...	...	
995	0.002493	0.000283	0.001041	0.000233	1.99199	1.99199203
996	0.002337	0.000400	0.001294	0.000161	1.99399	1.99399402
997	0.001685	0.000187	0.000815	0.000109	1.996	1.99599601
998	0.001543	0.000070	0.000746	0.000005	1.998	1.99799800
999	0.001406	0.000049	0.000700	0.000026	2	{'a

1000 rows × 14 columns

Каждой строке в этой таблице соответствуют результаты эксперимента для конкретного значения гиперпараметра. Для поиска оптимального параметра регуляризации мы выбрали сетку из 1000 элементов, соответственно, в данной таблице 1000 строк. Также напомним, что в каждом эксперименте проводится процедура кросс-валидации, то есть модель обучается несколько раз на разных батчах. Разберемся, что в каждом столбце таблицы выше:

- `mean_fit_time` - среднее время обучения модели.
- `std_fit_time` - стандартное отклонение времени обучения модели.
- `mean_score_time` - среднее время предсказания модели.
- `std_score_time` - стандартное отклонение времение предсказания модели.
- `param_alpha` - значение гиперпараметра.
- `params` - значения всех гиперпараметров в виде словаря.
- `split0_test_score` и тд - значения метрики на каждом из этапов кросс-валидации.
- `mean_test_score` - усредненное значение метрики на тестовых батчах.

- `std_test_score` - стандартное отклонение значеняи метрики на тестовых батчах.
- `rank_test_score` - ранг эксперимента. Все эксприменты отсортированы по `mean_test_score`, ранг - это место эксперимента в отсортированном списке экспериментов.

Лучшая модель:

In [27]:

```
1 gs.best_estimator_
```

Out[27]:

```
Ridge(alpha=1e-05, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Значение метрики лучшей модели:

In [28]:

```
1 gs.best_score_
```

Out[28]:

```
-26.99855610530455
```

Значения гиперпараметров лучшей модели:

In [29]:

```
1 gs.best_params_
```

Out[29]:

```
{'alpha': 1e-05}
```

Время обучения лучшей модели в секундах:

In [30]:

```
1 gs.refit_time_
```

Out[30]:

```
0.0022339820861816406
```