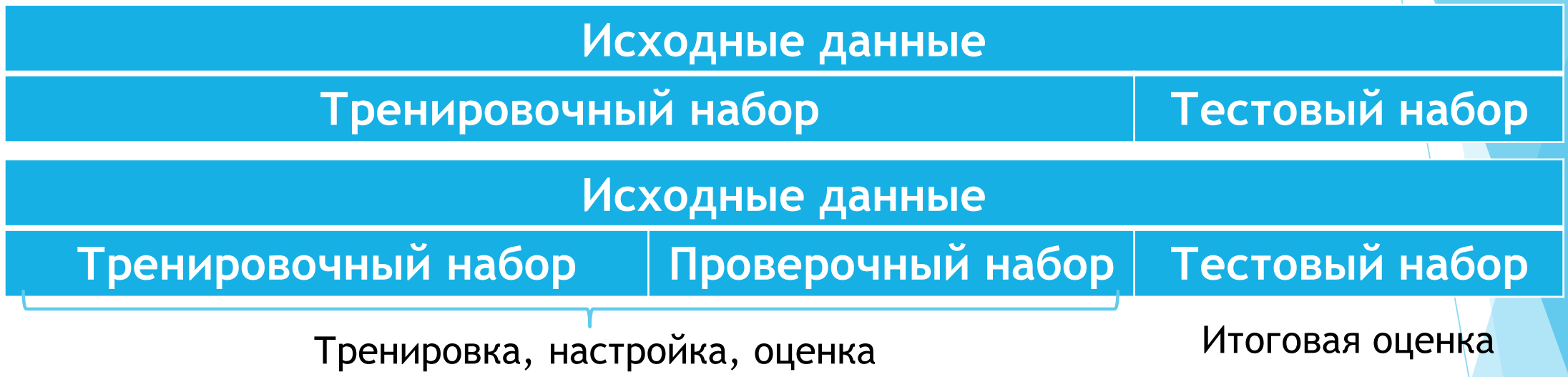
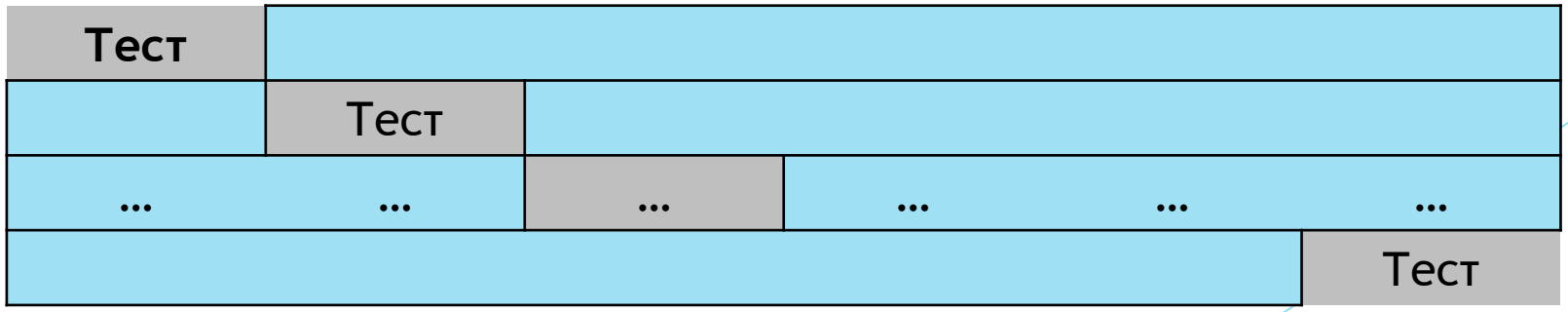
кросс-валидация, поиск лучших параметров

Кросс-валидация (перекрестная проверка)

- Основная задача – проверить способность алгоритма обобщать данные



- Случайным образом подразделяем тренировочный набор данных на k блоков без возврата
- k-1 блок для тренировки, 1 блок для тестирования



Настройка гиперпараметров

- Гиперпараметр – параметр (внешний по отношению к модели), который устанавливается перед обучением, во время обучения постоянный (априорный параметр)
- Примеры:
  - kNN: **n_neighbors** – количество соседей, **metric** – функция расстояний
  - Персептрон: **max_iter** –количество эпох обучения,

    eta0 – темп обучения
  - Логистическая регрессия: **C** – параметр регуляризации

    penalty: тип регуляризаци {‘l1’, ‘l2’}
  - SVM: **kernel** - Тип ядра {'linear', 'poly', 'rbf', 'sigmoid'}

    C – параметр регуляризации

    gamma – коэффициент ядра

Кросс-валидация. Для чего используется

- Усредненная оценка качества модели – менее чувствительная к разделению данных
- Оценка при настройке гиперпараметров
- Разброс характеристики качества

- Различные типы разбиений:
  - Kfold
  - StratifiedKFold
  - TimeSeriesSplit

Kfold - получение N наборов тренировочных и тестовых индексов

```python
import numpy as np
from sklearn.model_selection import KFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=4)
```

```python
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [1 2 3] TEST: [0]
TRAIN: [0 2 3] TEST: [1]
TRAIN: [0 1 3] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
```

```python
KFold(n_splits=3, shuffle=True)
```

StratifiedKfold – получение N наборов тренировочных и тестовых индексов с одинаковыми пропорциями классов

```python
import numpy as np
from sklearn.model_selection import StratifiedKFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
skf = StratifiedKFold(n_splits=2)
```

```python
for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

```

```
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
```

TimeSeriesSplit – получение N наборов тренировочных данных, последовательных по времени

```
1  import numpy as np
2  from sklearn.model_selection import TimeSeriesSplit
3  X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
4  y = np.array([1, 2, 3, 4, 5, 6])
5  tscv = TimeSeriesSplit(n_splits=5)
```

```
1  for train_index, test_index in tscv.split(X):
2      print("TRAIN:", train_index, "TEST:", test_index)
3      X_train, X_test = X[train_index], X[test_index]
```

```
TRAIN: [0] TEST: [1]
TRAIN: [0 1] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
TRAIN: [0 1 2 3] TEST: [4]
TRAIN: [0 1 2 3 4] TEST: [5]
```

Усредненная оценка качества модели

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
X, y = datasets.load_iris(return_X_y=True)
```

```
clr = DecisionTreeClassifier(max_depth=3)
kfold = KFold(n_splits=5, shuffle=True, random_state=0)
```

```
mean = []
for train_index, test_index in kfold.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clr.fit(X_train, y_train)
    y_pred = clr.predict(X_test)
    mean.append(accuracy_score(y_test, y_pred))
```

```
print(mean)
print(np.array(mean).mean())
```

```
[0.9666666666666667, 0.9, 1.0, 0.9666666666666667, 0.9333333333333333]
0.9533333333333334
```

Усредненная оценка 2 - cross_val_score

```
1 from sklearn import datasets
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.metrics import accuracy_score
4 from sklearn.model_selection import cross_val_score
5 X, y = datasets.load_iris(return_X_y=True)
```

```
1 clr2 = DecisionTreeClassifier(max_depth=3)
```

```
1 res = cross_val_score(clr2, X, y, cv=5)
```

```
1 print(res)
2 print(res.mean())
```

```
[0.96666667 0.96666667 0.93333333 0.93333333 1.        ]
0.96
```

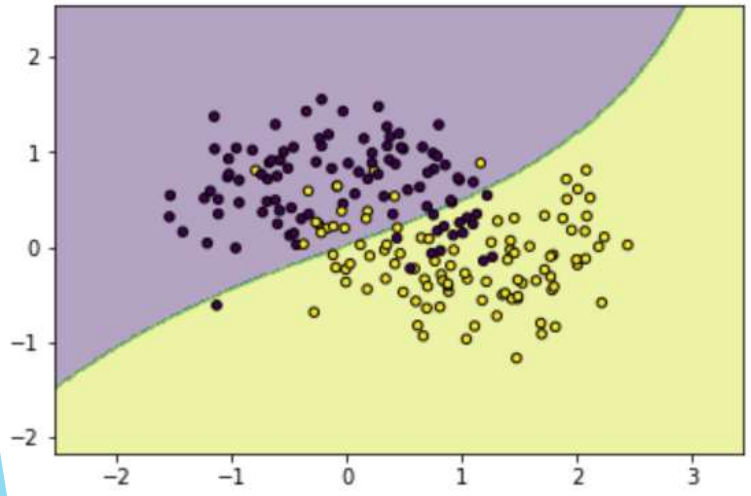
```
1 skf = StratifiedKFold(n_splits=5, random_state=42)
2 res = cross_val_score(clr2, X, y, cv=skf)
```

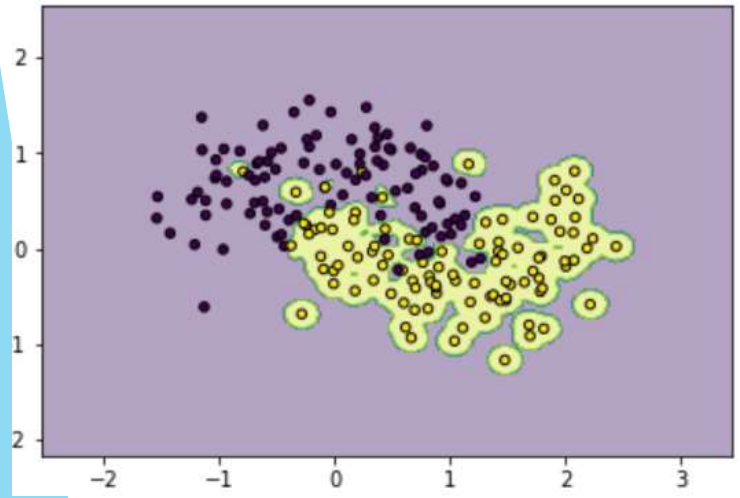
```
1 print(res)
2 print(res.mean())
```

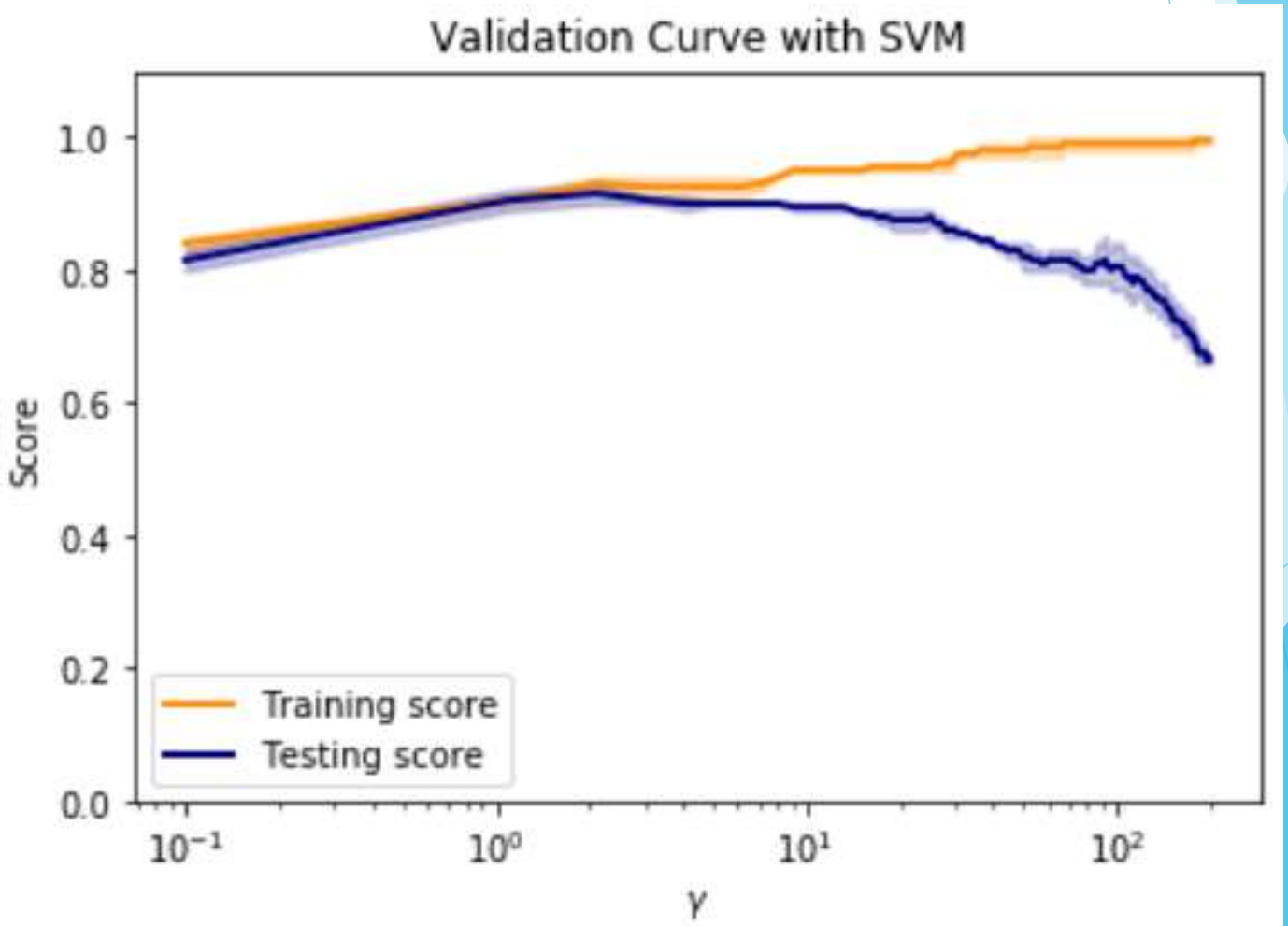
```
[0.96666667 0.96666667 0.93333333 1.         1.        ]
0.9733333333333334
```

Кривая проверки (validation curve)

- Будем изменять параметр gamma [0.1, 200]
- Выведем результат работы алгоритма на тестовых данных и на проверочных







Какие гиперпараметры есть у алгоритма

- BaseEstimator – базовый класс для всех алгоритмов
- get_params() – метод базового класса

```
1 SVC().get_params()
```

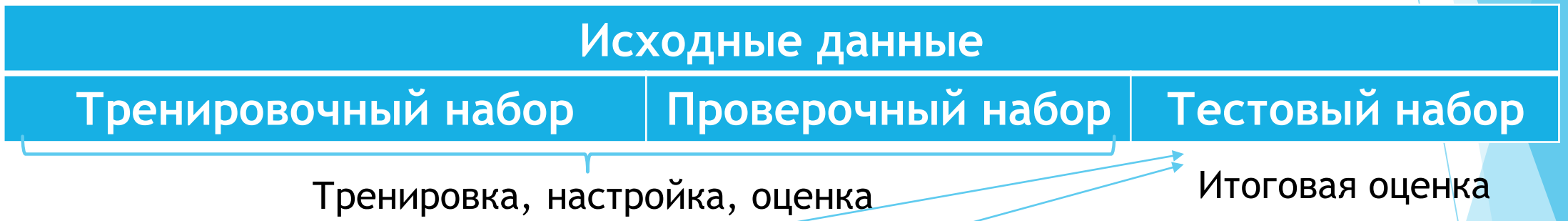
```
{'C': 1.0,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'auto_deprecated',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

```
2 LogisticRegression().get_params()
```

```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'max_iter': 100,
 'multi_class': 'warn',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'warn',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

GridSearchCV – поиск по сетке

- Задача – перебрать параметры из списка и определить при каких параметрах у алгоритма лучшее качество
- Задаем список параметров для перебора
- Передаем алгоритм, список параметров, задаем тип кросс-валидации




```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=12)
```

```
Cs = [0.01, 0.1, 1, 10, 100]
Penalties = ['l1','l2']
param_grid = {'C' : Cs, 'penalty': Penalties}
clfr = LogisticRegression(multi_class='auto', solver='liblinear', max_iter=1000)
grid_search = GridSearchCV(clfr, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

GridSearchCV - результаты

- grid_search.cv_results_ - полное описание всех итераций
- grid_search.best_params_ - лучшие параметры

```
{'C': 1, 'penalty': 'l1'}
```

- grid_search.best_index_ - индекс набора лучших параметров

```
4
```

- grid_search.best_estimator_ - лучший алгоритм (алгоритм, чьи параметры оказались лучшими)

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=1000, multi_class='auto',
          n_jobs=None, penalty='l1', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm start=False)
```

```
1 print(accuracy_score(y_test, grid_search.best_estimator_.predict(X_test)))
2 print(accuracy_score(y_test, grid_search.predict(X_test)))
```

```
0.977777777777777
0.977777777777777
```

Итоговая оценка на тестовом наборе

GridSearchCV – результаты 2. cv_results_ в читаемом виде

```
res = (
    p.DataFrame({
        "mean_test_score": grid_search.cv_results_["mean_test_score"],
        "mean_fit_time": grid_search.cv_results_["mean_fit_time"]})
     .join(p.io.json.json_normalize(grid_search.cv_results_["params"]).add_prefix("param_"))
)
```

	mean_test_score	mean_fit_time	param_C	param_penalty
0	0.342857	0.000801	0.01	l1
1	0.657143	0.001000	0.01	l2
2	0.666667	0.002402	0.10	l1
3	0.761905	0.000801	0.10	l2
4	0.952381	0.007203	1.00	l1
5	0.952381	0.001001	1.00	l2
6	0.942857	0.009805	10.00	l1
7	0.952381	0.001401	10.00	l2

RandomizedSearchCV

- Перебор всех комбинаций параметров - трудоемкая задача
- Идея - перебор первых n случайных комбинаций

```
1  rs = RandomizedSearchCV(clfr, param_grid, iid=False, cv=5, n_iter=3)
2  rs.fit(X_train, y_train)
```

	mean_test_score	mean_fit_time	param_C	param_penalty
0	0.657273	0.0008	0.01	l2
1	0.951905	0.0136	1.00	l1
2	0.941905	0.0032	100.00	l2

GridSearch для PipeLine

```
1  from sklearn.pipeline import Pipeline
2  ppln = Pipeline([("scalar", StandardScaler()), ("estimator", SVC())])
```

```
1  ppln.get_params()
```

```
{'memory': None,
 'steps': [('scalar',
 'scalar__with_std': True,
 'estimator__C': 1.0,
 'estimator__cache_size': 200,
 'estimator__class_weight': None,
 'estimator__degree': 3,
 'estimator__gamma': 'auto_deprecated',
 'estimator__kernel': 'rbf',
```

```
Cs = [0.01, 0.1, 1, 10, 100]
Krns = ['linear', 'poly', 'rbf', 'sigmoid']
Gammas = [1, 10, 100]
param_grid = {'estimator__C' : Cs, 'estimator__kernel': Krns, 'estimator__gamma' : Gammas}
```

Пример из ДЗ, где делали прогноз по RGB кислотности среды

Исходные данные		
Тренировочный набор	Проверочный набор	Тестовый набор

Тренировка, настройка, оценка

- Train – данные для тренировки модели
- Validation – данные для оценки точности натренированной модели при настройке параметров модели
- Test – Данные, которые ни разу не участвовали в процессе настройки – для окончательной оценки модели

Итоговая оценка

R, G, B

[255, 38, 0]
[255, 124, 0]
[141, 250, 0]
[179, 68, 198]
[111, 43, 142]

- У нас была разбивка train/test. На train (тренировочный)/test(**проверочный**) мы выбирали алгоритм
- В качестве тестового набора у нас выступал набор из 5ти образцов
- Если разбивку сделать 2/3, то по точности будет другой алгоритм, но после кроссвалидации найдётся правильный

Что бы мы могли сделать для выбора алгоритма?

- GridSearch для поиска гиперпараметров
- Кросс-валидация на X,y (на всём наборе, так как он разбивается на тренировочный/проверочный)
- Получение итоговой оценки (на базе оценки не перевыбирать алгоритм)

Домашнее задание

- Загрузить базу данных по недвижимости
- Отложить 15% данных для тестирования (итоговой оценки)
- Создать конвейер производящий масштабирование, создающий полиномиальные признаки и оценку с регуляризацией L1 (что раньше, масштабирование или полиномиальные признаки?)
- Подобрать параметры – степень масштабирования, коэффициент регуляризации. При подборе использовать кросс-валидацию по 5ти блокам. В качестве оценки использовать метрику R^2
- Вывести сколько получилось признаков после масштабирования, сколько признаков значимо (имеют абсолютное значение больше единицы)
- Оценить качество на отложенной выборке. RMSE, R^2
- Проанализировать остатки