



Лекция 5: Контроль качества и выбор модели

Валидация, кросс-валидация и бутстреппинг

- Эти методы позволяют:
 - оценить ошибки прогнозирования тестового набора
 - найти стандартное отклонение и оценки параметров модели
 - выбрать лучшую модель
- Различия между *ошибкой тестирования* и *ошибкой обучения*:
 - Ошибка тестирования - это усредненная ошибка, которая возникает в результате применения модели машинного обучения для прогнозирования отклика на новом наблюдении, которое не было задействовано в процессе обучения.
 - Ошибка обучения вычисляется во время применения метода машинного обучения к наблюдениям, используемым в обучении.

Применение валидационного набора

- Разделим случайным образом имеющийся набор образцов на две части: обучающую и валидационную выборки.



- Построим модель на обучающем наборе и используем ее для прогнозирования откликов наблюдений в валидационном наборе.
- Полученная ошибка на валидационном множестве дает оценку тестовой ошибки.

$$HO(\mu, Z, Z_{val}) = Q(\mu(Z \setminus Z_{val}), Z_{val})$$

Использование валидационного набора данных

Training Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Validation Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Основные методы генерации валидационного набора:

- Случайная выборка
- Стратифицированная выборка (сохраняем распределение выбранных переменных)
- Кластерная выборка (сохраняем пропорции кластеров)

Оценка моделей

Training Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Validation Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |



Оценка качества моделей
на валидационном наборе

Сложность Валидационная
модели оценка

Выбор модели

Training Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Validation Data

| | <i>inputs</i> | | | <i>target</i> |
|--|---------------|--|--|---------------|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |



Самая простая модель среди
самых лучших на
валидационном наборе

Сложность модели Валидационная
оценка

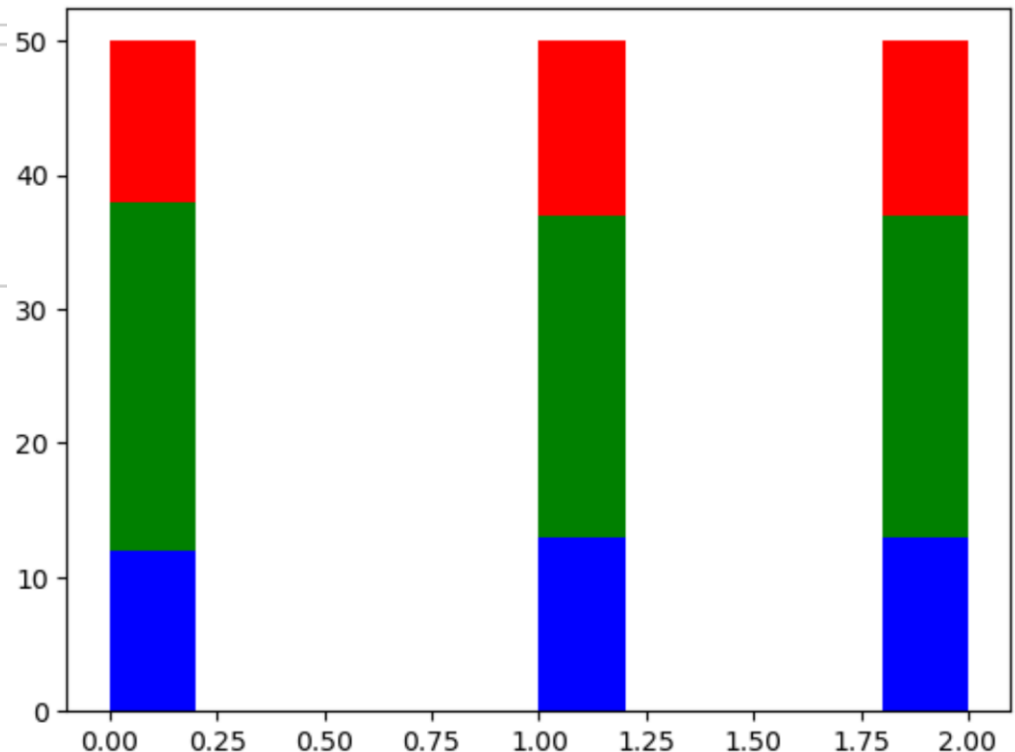
Пример (Python)

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
```

```
iris = load_iris()
X_train, X_test, y_train, y_test, = train_test_split(iris.data, iris.target, test_size=0.25,
    shuffle=True, # Random select
    stratify=iris.target, # Stratification
    random_state=123)
```

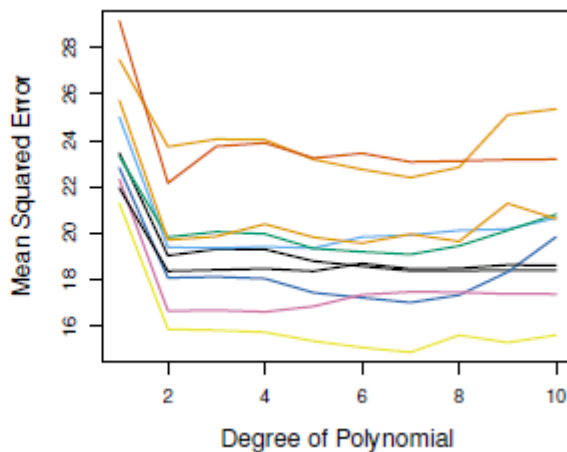
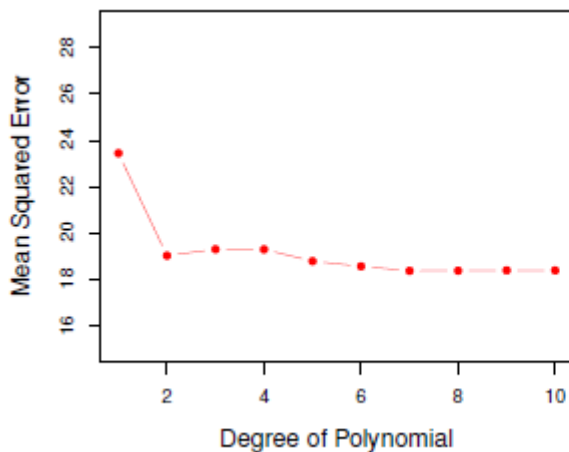
```
plt.hist(iris.target, color="red")
plt.hist(y_train, color="green")
plt.hist(y_test, color="blue")
pass
```

Для кластерной выборки
передаем в качестве stratify
метки кластеров



Пример

- Хотим сравнить регрессионные модели с разными степенями полинома
- Разделим случайным образом 392 наблюдения на две группы: обучающий набор, содержащий 196 объектов и валидационный набор, содержащий оставшиеся 196 объектов.



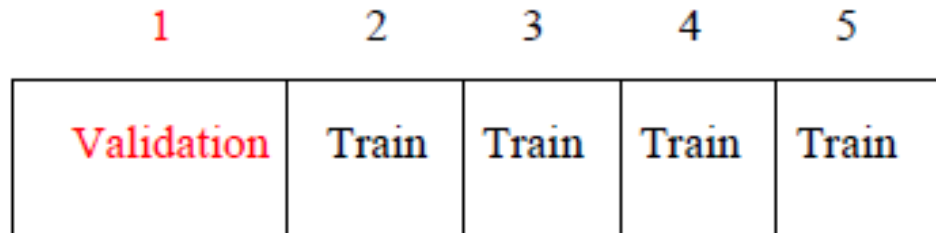
Слева показано одиночное разбиение, справа - множественное

Недостатки подхода применения валидационного набора

- Если плохое разбиение:
 - Валидационная оценка ошибки тестирования может сильно варьироваться в зависимости от того, какие именно наблюдения включены в обучающий набор, а какие в валидационный.
- Не вся информация используется при обучении:
 - При валидационный подходе только подмножество наблюдений (те, которые включены в обучающий набора, а не в валидационный) используются для построения модели.
- Чрезмерный оптимизм:
 - Ошибка на валидационном наборе может иметь тенденцию *переоценивать* ошибку тестирования

Кросс-валидация

- Широко используемый подход для оценки ошибки тестирования.
- Оценки могут быть использованы для:
 - выбора оптимальной модели,
 - оценки тестовой ошибки результирующей выбранной модели.
- Идея - разделить данные на K частей равного размера. Мы удаляем часть k , строим модель на оставшихся частях, а затем получаем прогнозы для удаленной k -ой части.

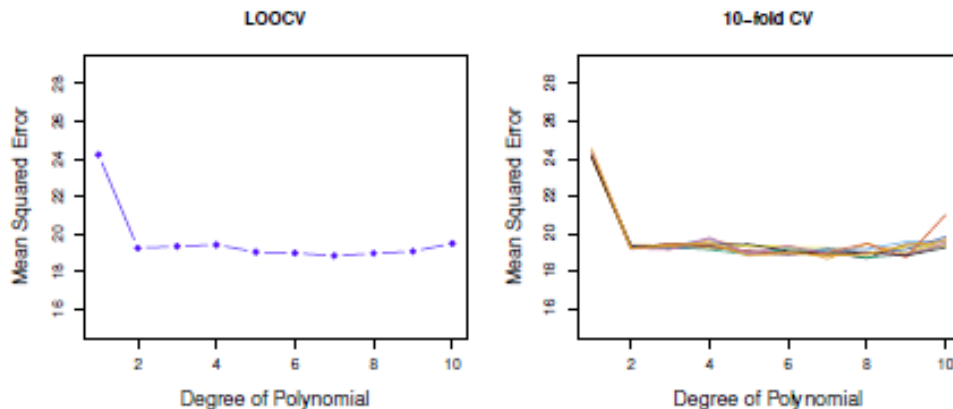


- Это делается в свою очередь для каждой части $k = 1, 2, \dots, K$, а затем результаты объединяются.

Кросс-валидация для оценки ошибки

- Обозначим K частей как Z_1, \dots, Z_K , где Z_k - это индексы наблюдений в части k . Есть l_k наблюдений в части k : если l кратно K , то удобно $l_k = l/K$
- Вычислим:

$$CV_Z(\mu) = \sum_{k=1}^K \frac{l_k}{l} Q(\mu(Z \setminus Z_k), Z_k)$$



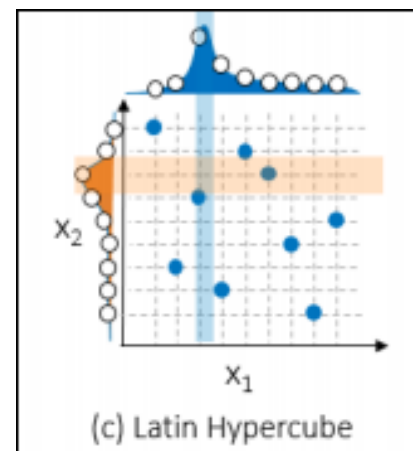
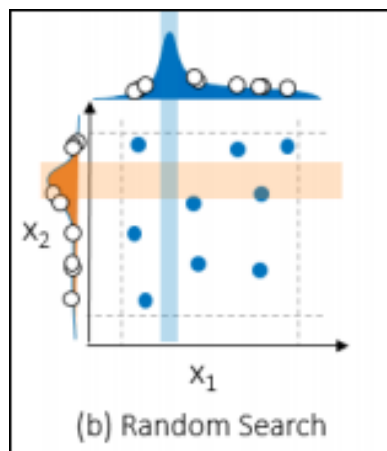
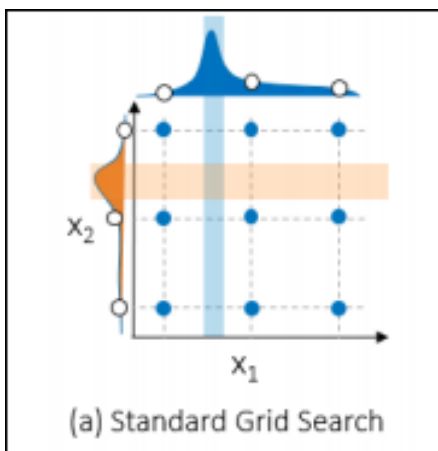
- При $K = l$ имеем l папок или кросс-валидацию с попеременным исключением одного наблюдения – скользящий контроль (*leave-one out cross-validation*, LOOCV).



Кросс-валидация для оценки метопараметров и выбора модели

- Задают стратегию перебора вариантов метопараметров
- Запускают кросс-валидацию для разных значений метопараметров
- Рассчитывают кросс-валидационные ошибки для каждого варианта
- Выбирают лучшее значение метопараметра по кросс-валидационной ошибке
- Перестраивают модель на всей выборке с этим значением метопараметра

Кросс-валидация и валидация для выбора метопараметров



■ AutoML:

- ☐ Поиск по решетке
- ☐ Случайный поиск
- ☐ Латинский гиперкуб
- ☐ Эволюционные и генетические алгоритмы поиска
- ☐ «Мета» оптимизация
- ☐ Байесовская оптимизация

■ Оценка качества:

- ☐ Не обязательно (и даже как правило) оценка качества для выбора модели совпадает с функцией потерь для обучения модели!!!!

Пример (Python)

```
from sklearn.utils import resample
```

```
X, y = fetch_california_housing(return_X_y=True, as_frame=True)  
X
```

| | MedInc | HouseAge | AveRooms | AveBedrms | Population | AveOccup | Latitude | Longitude |
|-------|--------|----------|----------|-----------|------------|----------|----------|-----------|
| 0 | 8.3252 | 41.0 | 6.984127 | 1.023810 | 322.0 | 2.555556 | 37.88 | -122.23 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 | 2.109842 | 37.86 | -122.22 |
| 2 | 7.2574 | 52.0 | 8.288136 | 1.073446 | 496.0 | 2.802260 | 37.85 | -122.24 |
| 3 | 5.6431 | 52.0 | 5.817352 | 1.073059 | 558.0 | 2.547945 | 37.85 | -122.25 |
| 4 | 3.8462 | 52.0 | 6.281853 | 1.081081 | 565.0 | 2.181467 | 37.85 | -122.25 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 20635 | 1.5603 | 25.0 | 5.045455 | 1.133333 | 845.0 | 2.560606 | 39.48 | -121.09 |
| 20636 | 2.5568 | 18.0 | 6.114035 | 1.315789 | 356.0 | 3.122807 | 39.49 | -121.21 |
| 20637 | 1.7000 | 17.0 | 5.205543 | 1.120092 | 1007.0 | 2.325635 | 39.43 | -121.22 |
| 20638 | 1.8672 | 18.0 | 5.329513 | 1.171920 | 741.0 | 2.123209 | 39.43 | -121.32 |
| 20639 | 2.3886 | 16.0 | 5.254717 | 1.162264 | 1387.0 | 2.616981 | 39.37 | -121.24 |

20640 rows × 8 columns

Пример – Grid Search (Python)

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.datasets import fetch_california_housing
```

```
X, y = fetch_california_housing(return_X_y=True)
```

```
N = 5000
X, y = X[:N], y[:N]
X.shape, y.shape
```

```
((5000, 8), (5000,))
```

```
scaler = StandardScaler()
VT = VarianceThreshold() # Preprocessing
KNN = KNeighborsRegressor() # Regressor
```

```
# Combined model - encapsulates all stages
model = Pipeline([("scaler", scaler), ("VT", VT), ("KNN", KNN)])
```

Пример – Grid Search (Python)

```
# Parameters to cycle through  
# Pipeline parameters are passed as <STAGE>__<PARAMETER NAME>  
parameters = {"KNN__n_neighbors": range(2, 20),  
              "VT__threshold": [0, 1]}
```

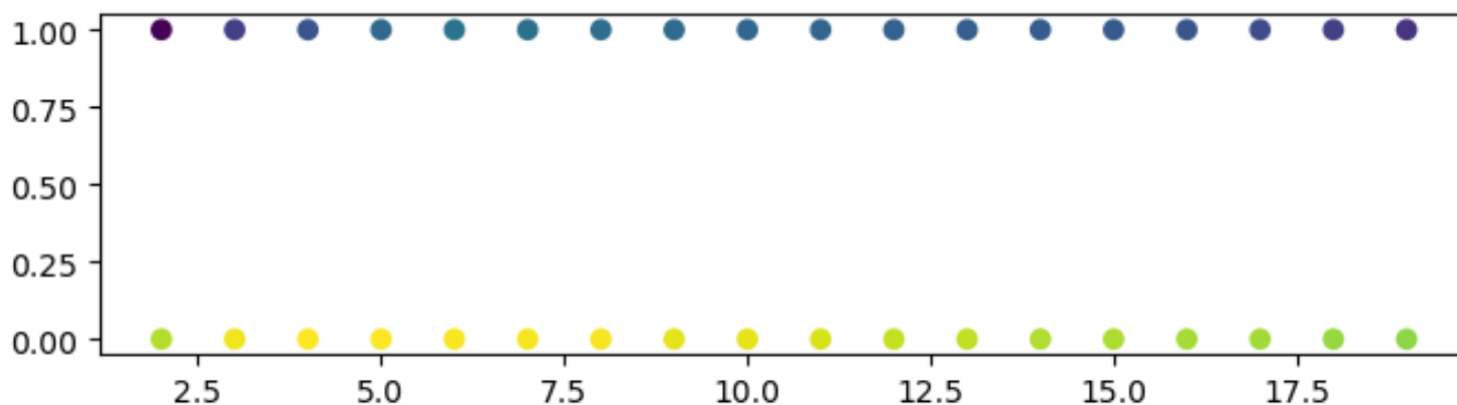
```
# 5-fold cross-validation  
GSCV = GridSearchCV(model, parameters, cv=5)  
GSCV.fit(X, y)  
pass
```

```
GSCV.best_params_  
  
{'KNN__n_neighbors': 4, 'VT__threshold': 0}
```

```
pred = GSCV.predict(X) # GSCV is equal to the best estimator
```


Пример – Grid Search (Python)

```
plt.scatter(*pd.DataFrame(GSCV.cv_results_["params"]).T.values, c=GSCV.cv_results_["mean_test_score"])  
plt.gcf().set_size_inches(8, 2)
```



Пример – Случайный поиск (Python)

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

```
model = Pipeline([("scaler", scaler),
                  ("VT", VarianceThreshold()),
                  ("KNN", KNeighborsRegressor())])
```

```
distributions = {"KNN__n_neighbors": randint(2, 20),
                "VT__threshold": uniform(0, 1)}
```

```
# 5-fold cross-validation
```

```
RSCV = RandomizedSearchCV(model, distributions, n_iter=20,
                          cv=5, random_state=123)
```

```
RSCV.fit(X, y)
```

```
pass
```

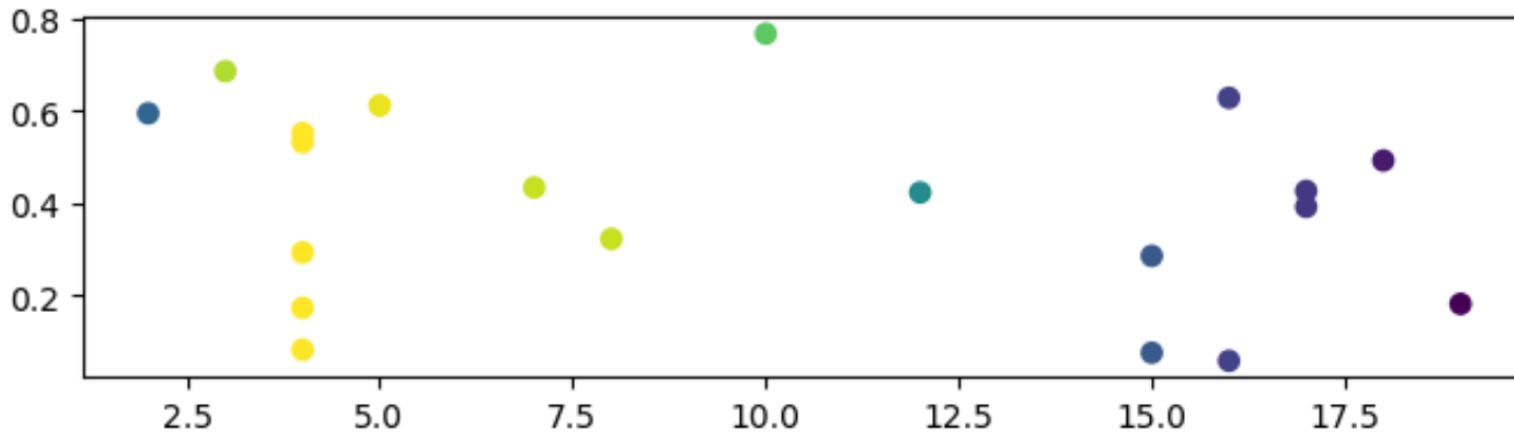
Пример – Случайный поиск (Python)

```
RSCV.best_params_
```

```
{'KNN__n_neighbors': 4, 'VT__threshold': 0.5513147690828912}
```

```
pred = RSCV.predict(X) # RSCV is equal to the best estimator
```

```
plt.scatter(*pd.DataFrame(RSCV.cv_results_["params"]).T.values, c=RSCV.cv_results_["mean_test_score"])  
plt.gcf().set_size_inches(8, 2)
```



Пример – Отбор (Python)

```
from sklearn.experimental import enable_halving_search_cv # Required import
from sklearn.model_selection import HalvingGridSearchCV, HalvingRandomSearchCV
```

```
model = Pipeline([("scaler", scaler),
                  ("VT", VarianceThreshold()),
                  ("KNN", KNeighborsRegressor())])
```

```
distributions = {"KNN__n_neighbors": randint(2, 20),
                "VT__threshold": uniform(0, 1)}
```

```
HRSV = HalvingRandomSearchCV(model, distributions, cv=5,
                             factor=2, # Candidate selection cut-off
                             # Resource increasing during selection:
                             resource="n_samples",
                             min_resources=100)
```

```
HRSV.fit(X, y)
pass
```

Пример – Отбор (Python)

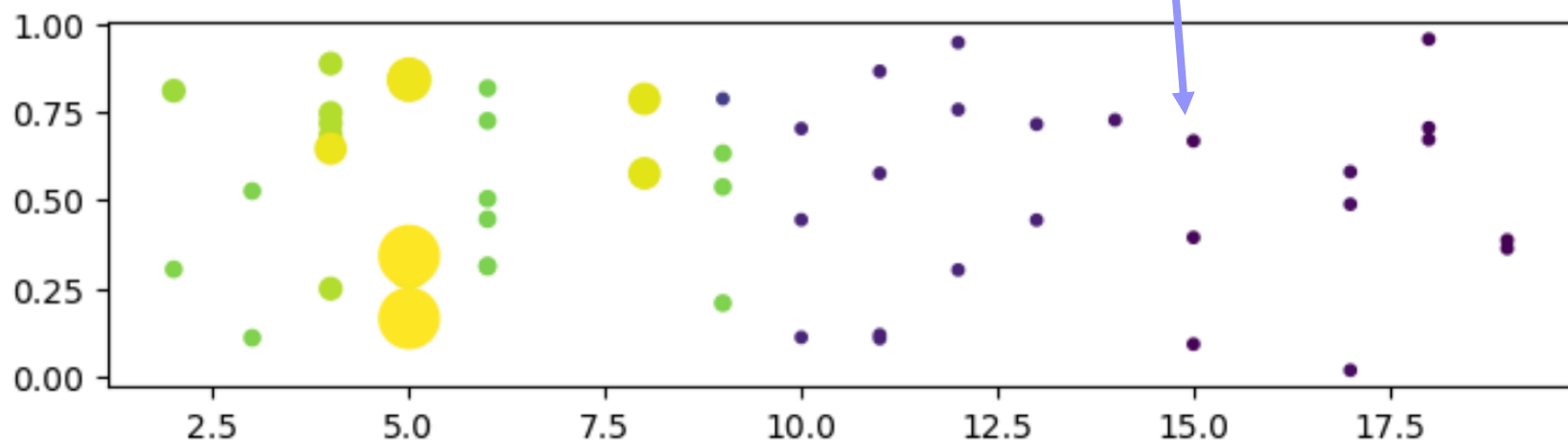
```
HRSV.best_params_
```

```
{'KNN__n_neighbors': 5, 'VT__threshold': 0.3417047325655804}
```

```
pred = HRSV.predict(X) # RSCV is equal to the best estimator
```

```
plt.scatter(*pd.DataFrame(HRSV.cv_results_["params"]).T.values,  
            c=HRSV.cv_results_["mean_test_score"],  
            s=HRSV.cv_results_["n_resources"]/10)  
plt.gcf().set_size_inches(8, 2)
```

Размер отвечает за число семплов



Бутстрэппинг

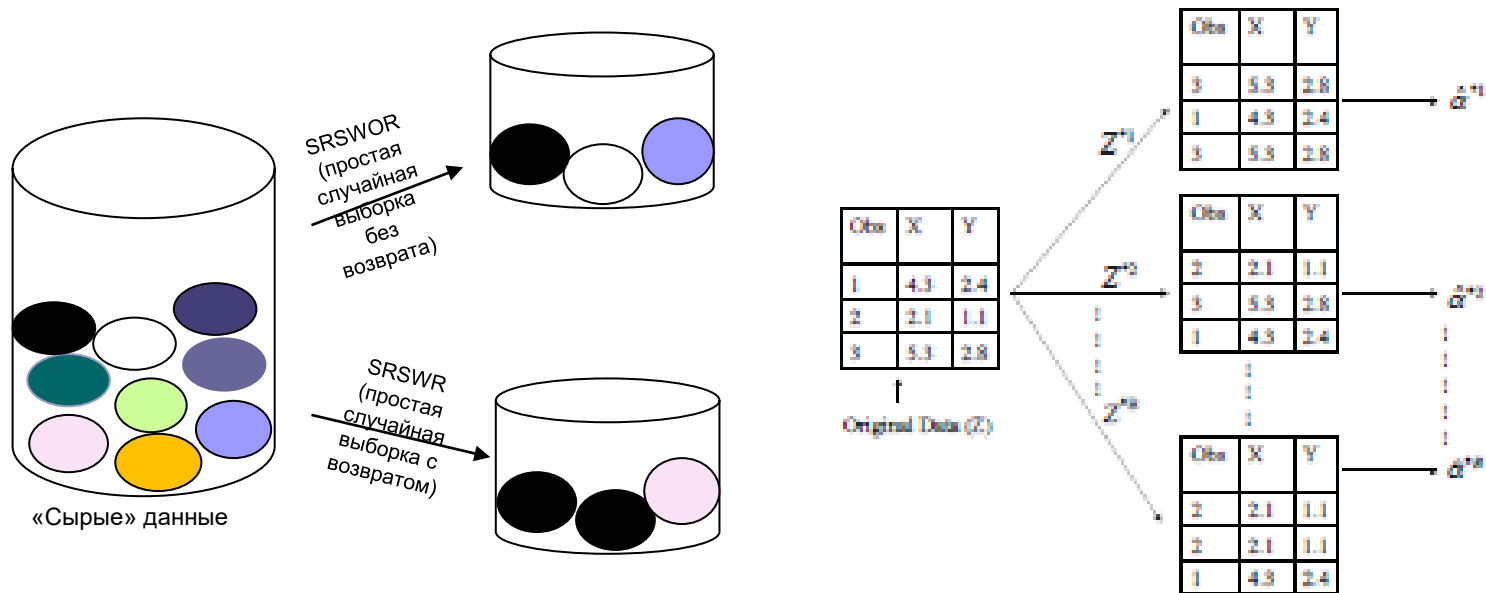
- *Бутстрэппинг* представляет собой мощный статистический инструмент, который может быть использован для количественной оценки неопределенности.
- Например, он может позволить произвести оценку стандартной ошибки коэффициента или доверительного интервала для этого коэффициента.
- Использование термина бутстреппинг происходит от фразы, чтобы *to pull oneself up by one's bootstraps*, - цитата из книги «Удивительные приключения барона Мюнхгаузена»

Барон упал на дно глубокого озера. Когда казалось, что все было потеряно, он решил вытащить себя своими собственными силами.

Бутстрэппинг

- Подход бутстрэппинга позволяет имитировать процесс получения новых случайных наборов данных, так что мы можем оценить дисперсию нашей оценки, не создавая дополнительных образцов.
- Вместо того, чтобы постоянно получать независимые наборы данных, мы получаем различные наборы путем многократной выборки наблюдений из исходного набора с *замещением* (или с *возвращением*).
- Каждый из этих "наборов данных" создается путем выборки с *замещением* и имеет *такой же размер* как наш исходный набор данных. В результате некоторые наблюдения могут появляться более одного раза в наборе данных бутстрэппинга, а некоторые нет вообще.

Демонстрационный пример



- Графическая иллюстрация бутстреппингового подхода на маленькой выборке
- Каждый бутстреппинговый набор данных содержит наблюдения, отобранные с заменой из исходного набора.
- Каждый такой набор данных начальной используется для получения оценки

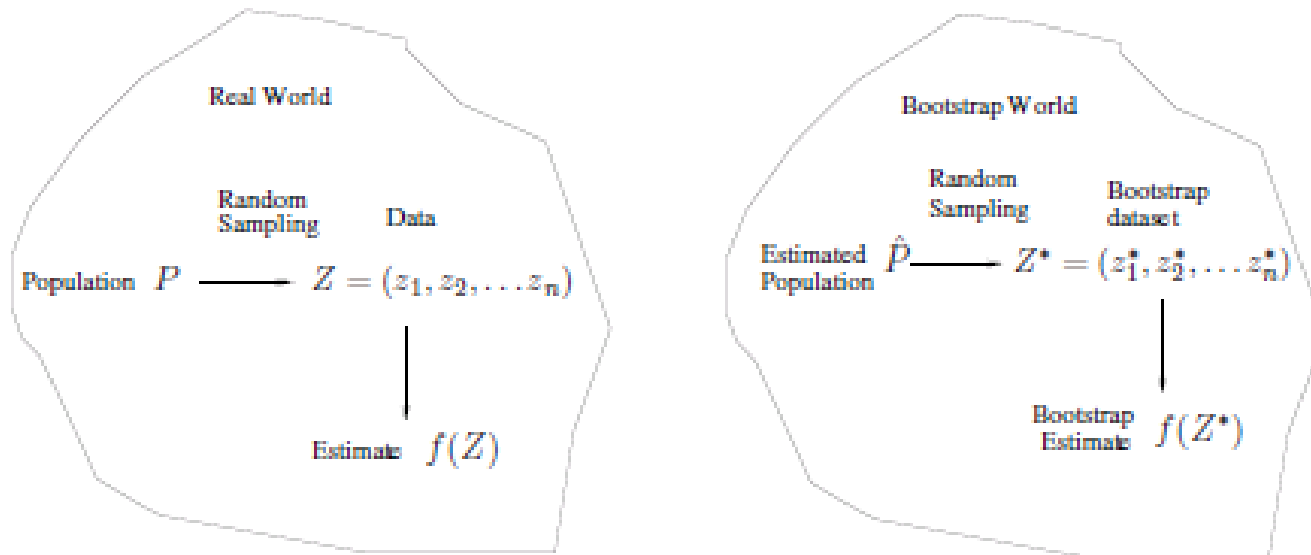
Бутстрэппинг

- Обозначая первый набор данных бутстреппинга как Z^{*k} , мы используем его, чтобы выполнить новую оценку для a^{*k}
- Эта процедура повторяется B раз для некоторого большого значения B (например, 100 или 1000), чтобы получить B различных наборов данных бутстреппинга $Z^{*1}, Z^{*2}, \dots, Z^{*B}$, и B соответствующих оценок $a^{*1}, a^{*2}, \dots, a^{*B}$
- Оценим среднее и стандартную ошибку этих оценок бутстреппинга:

$$\tilde{a} = \frac{1}{B} \sum_{r=1}^B (a^{*r}), SE_B(a) = \sqrt{\frac{1}{B-1} \sum_{r=1}^B (\tilde{a} - a^{*r})^2}$$

- Они служат в качестве оценки, полученной на тестовом наборе данных.

Общая схема бутстреппинга



- В более сложных ситуациях, определение подходящего способа для получения выборок бутстреппинга может потребовать значительных усилий.
- Например, если данные представляют собой временные ряды, мы не можем просто выбирать наблюдения с замещением

Как бутстрепинг оценивает ошибку прогнозирования

- При кросс-валидации каждая из K папок валидации отличается от других $K - 1$ папок, используемых для обучения: *перекрытия нет*.
- Для оценки ошибки прогнозирования с помощью бутстреппинга мы могли бы подумать об использовании каждого набора данных бутстреппинга в нашей обучающей выборке и исходного набора данных как валидационного набора (или наоборот).
 - Но каждая выборка бутстреппинга имеет значительное перекрытие с исходным набором (около двух третей).
 - Это приведет бутстреппинг к существенному недооцениванию истинной ошибки прогнозирования
- Удаление перекрытия (*out of bag*) - можно частично решить эту проблему, используя для оценки только те наблюдения, которые не появились (случайно) в текущей выборке бутстреппинга.

$$OOB(\mu) = \sum_{k=1}^K \frac{l_k}{l} Q(\mu(Z^{*k}), Z \setminus Z^{*k}), |OOB| = \left(1 - \frac{1}{l}\right)^l \approx e^{-1} \approx 0.37$$

Пример (Python)

```
ITER = 100
SAMPLES = 100
frame = []
for i in range(ITER):
    sample = resample(X, replace=True, n_samples=SAMPLES, stratify=None)
    stat = sample["HouseAge"].mean()
    frame.append(stat)
frame = np.array(frame).flatten()
frame = pd.Series(frame).sort_values()
```

Доверительные интервалы 90% для среднего возраста жилища:

```
frame.quantile(0.05), frame.quantile(0.95)
```

(26.248, 30.6515)

Бутстреп-регрессия (Python)

```
from sklearn.ensemble import BaggingRegressor
```

```
estimator = BaggingRegressor(LinearRegression(), n_estimators=100,  
                             bootstrap=True, max_samples=0.1,  
                             random_state=42)
```

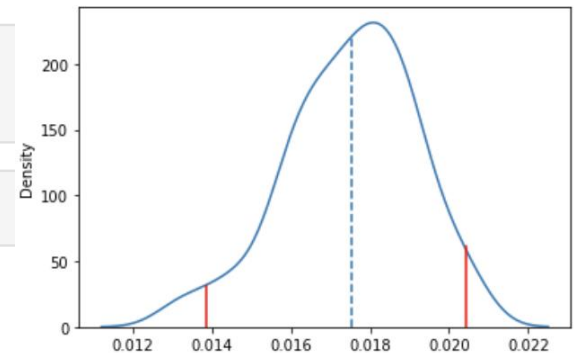
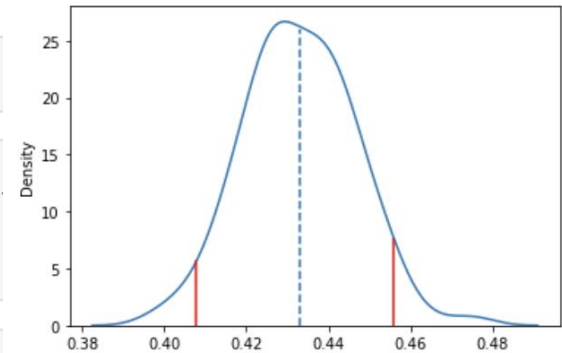
```
features = X[["MedInc", "HouseAge"]]
```

```
estimator.fit(features, y)  
pass
```

```
coefs = np.array([x.coef_ for x in estimator.estimators_])
```

```
sns.kdeplot(data=coefs[:, 0])  
plt.axvline(coefs[:, 0].mean(), 0, 0.93, ls="--")  
plt.axvline(np.quantile(coefs[:, 0], 0.025), 0, 0.20, c="red")  
plt.axvline(np.quantile(coefs[:, 0], 0.975), 0, 0.27, c="red")
```

```
sns.kdeplot(data=coefs[:, 1])  
plt.axvline(coefs[:, 1].mean(), 0, 0.91, ls="--")  
plt.axvline(np.quantile(coefs[:, 1], 0.025), 0, 0.13, c="red")  
plt.axvline(np.quantile(coefs[:, 1], 0.975), 0, 0.25, c="red")  
~~~~~
```



Бутстреп-регрессия (Python)

```
pred = np.array([x.predict(features.values) for x in estimator.estimators_]).T  
pred.shape
```

```
(20640, 100)
```

```
plt.plot(np.percentile(pred, q=5, axis=1)[:25], c="blue")  
plt.plot(np.percentile(pred, q=95, axis=1)[:25], c="red")  
plt.scatter(range(25), estimator.predict(features)[:25], c="green")  
pass
```

