

Введение в искусственный интеллект. Машинное обучение

Тема семинара: обзор, кросс-валидация

Бабин Д.Н., Иванов И.Е.

кафедра Математической Теории Интеллектуальных Систем



1 Организационные вопросы

План семинара

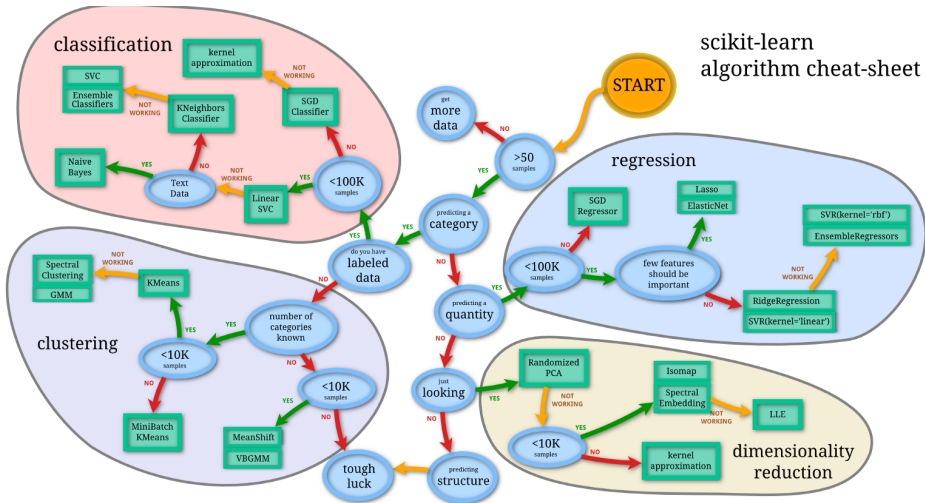
- 1 Организационные вопросы
- 2 Обзор семинарских занятий

- 1 Организационные вопросы
- 2 Обзор семинарских занятий
- 3 Скользящий контроль

- Данный курс является частью программы **SHARE**
 - **SHARE** = School of Huawei Advanced Research Education
 - **SHARE** = Школа опережающего научного образования Хуавэй
 - e-mail: share@intsys.msu.ru
 - Канал SHARE: https://t.me/joinchat/AAAAAE_r4XKzEDaUKy1FwA

- Страница курса: <https://github.com/mlcoursemm/ml2021autumn>
- Главный ресурс по курсам “Введение в компьютерный интеллект”:
<https://github.com/mlcoursemm>
- Телеграмм-канал: <https://t.me/joinchat/AAAAAEUmx5cJL0dLXs0t8g>
- Группа обсуждения: <https://t.me/joinchat/AAAAAEx8IrWw-nYJPo6smQ>
- Почта курса: mlcoursemm@gmail.com
- Телеграмм-бот для посылки домашних заданий [@ml2021sharebot](#)

Дорожная карта Scikit-Learn¹



¹https://scikit-learn.org/stable/tutorial/machine_learning_map/

- Алгоритмы k -ближайших соседей (k-Nearest Neighbor)
 - Самый базовый алгоритм **классификации** — выбор ближайшего элемента из обучающего множества и вывод соответствующего ему класса
 - Различается по количеству ближайших точек (k), используемой метрике близости, наличию ускоренных процедур, методам приближенных вычислений и т.д.

- Алгоритмы k -ближайших соседей (k-Nearest Neighbor)
 - Самый базовый алгоритм **классификации** — выбор ближайшего элемента из обучающего множества и вывод соответствующего ему класса
 - Различается по количеству ближайших точек (k), используемой метрике близости, наличию ускоренных процедур, методам приближенных вычислений и т.д.
- Алгоритмы кластеризации k -средних (k-means clusterization)
 - Базовый метод обучения **без учителя**: входной неразмеченный массив примеров необходимо разбить на классы (кластеры)
 - Различается по начальной инициализации алгоритма, методам подсчета центроида классов, работе с выбросами и т.д.



- Алгоритмы k -ближайших соседей (k-Nearest Neighbor)
 - Самый базовый алгоритм **классификации** — выбор ближайшего элемента из обучающего множества и вывод соответствующего ему класса
 - Различается по количеству ближайших точек (k), используемой метрике близости, наличию ускоренных процедур, методам приближенных вычислений и т.д.
- Алгоритмы кластеризации k -средних (k-means clusterization)
 - Базовый метод обучения **без учителя**: входной неразмеченный массив примеров необходимо разбить на классы (кластеры)
 - Различается по начальной инициализации алгоритма, методам подсчета центроида классов, работе с выбросами и т.д.
- Регрессия и оценка качества
 - Рассматриваются базовые методы линейной **регрессии** для восстановления целевой зависимости, а также критерий оценки качества этого восстановления
 - Различаются по наличию аналитического решения, видам регуляризации и т.д.



- Метрики качества классификаторов
 - Будут разобраны основные методики оценки качества как бинарных, так и многоклассовых классификаторов
 - Узнаете, что такое ошибки первого и второго рода, ROC-кривая и площадь под ней (AUC), true/false positive/negative ответы и соответственно их доля (rate), точность и полнота (precision and recall) ответов классификатора. Будет примерно указано, какую метрику и где лучше использовать и т.д.

- Метрики качества классификаторов
 - Будут разобраны основные методики оценки качества как бинарных, так и многоклассовых классификаторов
 - Узнаете, что такое ошибки первого и второго рода, ROC-кривая и площадь под ней (AUC), true/false positive/negative ответы и соответственно их доля (rate), точность и полнота (precision and recall) ответов классификатора. Будет примерно указано, какую метрику и где лучше использовать и т.д.
- Построение машины опорных векторов
 - Support Vector Machine (SVM) — один из наиболее устойчивых и распространенных классификаторов в классическом машинном обучении
 - Будет подробно разобран пример построения такого классификатора аналитически



- Метрики качества классификаторов
 - Будут разобраны основные методики оценки качества как бинарных, так и многоклассовых классификаторов
 - Узнаете, что такое ошибки первого и второго рода, ROC-кривая и площадь под ней (AUC), true/false positive/negative ответы и соответственно их доля (rate), точность и полнота (precision and recall) ответов классификатора. Будет примерно указано, какую метрику и где лучше использовать и т.д.
- Построение машины опорных векторов
 - Support Vector Machine (SVM) — один из наиболее устойчивых и распространенных классификаторов в классическом машинном обучении
 - Будет подробно разобран пример построения такого классификатора аналитически
- Работа с пропущенными значениями и выбор признаков
 - Зачастую в реальной жизни не все признаки и не у всех объектов известны. Также порой не известно заранее, какие признаки важны, а какие наоборот мусорные
 - Будут рассмотрены различные методы заполнения пропущенных значений на основе статистик (среднее, минимум или максимум и т.д.), а также способы оценки и выбора наиболее важных признаков

- Регрессия на основе опорных векторов
 - Support Vector Regression (SVR) является расширением машины опорных векторов, но для задачи восстановления регрессии
 - Будут рассмотрены аналитическая формулировка и методы решения

- Регрессия на основе опорных векторов
 - Support Vector Regression (SVR) является расширением машины опорных векторов, но для задачи восстановления регрессии
 - Будут рассмотрены аналитическая формулировка и методы решения
- Ансамбли
 - Ансамблирование — основной способ “дожатия” последних процентов на тесте в соревнованиях по машинному обучению, а также создания более устойчивых систем, в которых ошибки разных моделей будут нивелироваться
 - Будет дан обзор методов ансамблирования в scikit-learn



- Регрессия на основе опорных векторов
 - Support Vector Regression (SVR) является расширением машины опорных векторов, но для задачи восстановления регрессии
 - Будут рассмотрены аналитическая формулировка и методы решения
- Ансамбли
 - Ансамблирование — основной способ “дожатия” последних процентов на тесте в соревнованиях по машинному обучению, а также создания более устойчивых систем, в которых ошибки разных моделей будут нивелироваться
 - Будет дан обзор методов ансамблирования в scikit-learn
- Разбор домашних заданий курса
 - На последнем (-их) занятии (-ях) будем разбирать домашние задания
 - Также будут предложены современные научные статьи для разбора по желанию





Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров

²Image source: <https://scikit-learn.org/>

Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное

²Image source: <https://scikit-learn.org/>

Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное
- 3 Обучаем модель

²Image source: <https://scikit-learn.org/>

Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное
- 3 Обучаем модель
- 4 Проводим процедуру кросс-валидации по пространству гиперпараметров и находим оптимальные их значения

²Image source: <https://scikit-learn.org/>

Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное
- 3 Обучаем модель
- 4 Проводим процедуру кросс-валидации по пространству гиперпараметров и находим оптимальные их значения
- 5 Обучаем на полном обучающем множестве с подобранными гиперпараметрами

²Image source: <https://scikit-learn.org/>

Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное
- 3 Обучаем модель
- 4 Проводим процедуру кросс-валидации по пространству гиперпараметров и находим оптимальные их значения
- 5 Обучаем на полном обучающем множестве с подобранными гиперпараметрами
- 6 Проверяем на тесте!

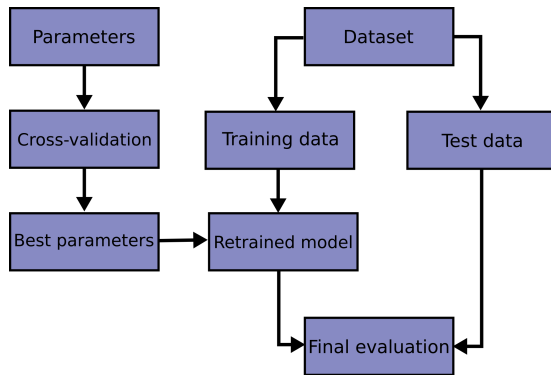
²Image source: <https://scikit-learn.org/>



Кросс-валидация: основные этапы корректного обучения

- 1 Придумываем модель и пространство гиперпараметров
- 2 Из исходных данных выделяем тестовое множество, а оставшееся множество делим на обучающее и валидационное
- 3 Обучаем модель
- 4 Проводим процедуру кросс-валидации по пространству гиперпараметров и находим оптимальные их значения
- 5 Обучаем на полном обучающем множестве с подобранными гиперпараметрами
- 6 Проверяем на тесте!

Общая схема²:



²Image source: <https://scikit-learn.org/>

- Иногда нужно поделить исходное множество примеров на обучающее и тестовое подмножества

Кросс-валидация: hold-out

- Иногда нужно поделить исходное множество примеров на обучающее и тестовое подмножества
- Простейшая кросс-валидация — это контроль на отложенном множестве (hold-out), при котором происходит однократное разделение множества



- Иногда нужно поделить исходное множество примеров на обучающее и тестовое подмножества
- Простейшая кросс-валидация — это контроль на отложенном множестве (hold-out), при котором происходит однократное разделение множества
- И то, и другое можно сделать одной процедурой



- Для начала разобьем общий массив данных на обучающее и тестовое подмножества

- Для начала разобьем общий массив данных на обучающее и тестовое подмножества
- Это делается с помощью функции `sklearn.model_selection.train_test_split()`:

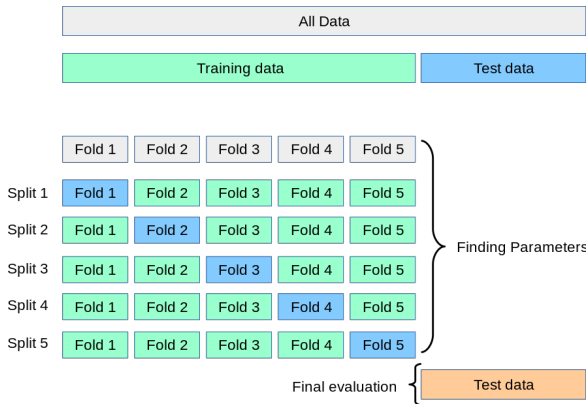
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets

X, y = datasets.load_iris(return_X_y=True) # X.shape = (150, 4), y.shape = (150,)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
# X_train.shape=(90, 4), y_train.shape=(90, ), X_test.shape=(60, 4), y_test.shape=(60, )
```



Кросс-валидация: k-fold валидация

Контроль по k блокам (k-fold валидация)³:



³Image source: <https://scikit-learn.org/>

- Теперь посмотрим на k-fold кросс-валидацию

- Теперь посмотрим на k-fold кросс-валидацию
- Это делается с помощью функций `sklearn.model_selection.cross_val_score()` и `cross_val_predict()`:

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X_train, y_train, cv=5)
# scores = [1., 1., 1., 1., 0.94444444]
```



- Теперь посмотрим на k-fold кросс-валидацию
- Это делается с помощью функций `sklearn.model_selection.cross_val_score()` и `cross_val_predict()`:

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X_train, y_train, cv=5)
# scores = [1., 1., 1., 1., 0.94444444]
```

- Параметр `cv` — это число блоков k



- Теперь посмотрим на k-fold кросс-валидацию
- Это делается с помощью функций `sklearn.model_selection.cross_val_score()` и `cross_val_predict()`:

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X_train, y_train, cv=5)
# scores = [1., 1., 1., 1., 0.94444444]
```

- Параметр `cv` — это число блоков k
- Функция `sklearn.model_selection.cross_val_predict()` выдает также значения на валидации для каждого примера из каждого тестового блока (заметим, что каждый пример входит ровно в один тестовый блок)



Кросс-валидация: scikit-learn

- В заключение темы кросс-валидации рассмотрим в `scikit-learn` специальные итераторы, которые разбивают исходное множество согласно разным правилам:

Кросс-валидация: scikit-learn

- В заключение темы кросс-валидации рассмотрим в scikit-learn специальные итераторы, которые разбивают исходное множество согласно разным правилам:
 - Разбиение множества согласно k-fold валидации: `sklearn.model_selection.KFold()`

```
from sklearn.model_selection import KFold

X = ["a", "b", "c", "d"]
kf = KFold(n_splits=2)
for train, test in kf.split(X):
    print("%s□%s" % (train, test))
# out: [2 3] [0 1]
# out: [0 1] [2 3]
```



Кросс-валидация: scikit-learn

- В заключение темы кросс-валидации рассмотрим в scikit-learn специальные итераторы, которые разбивают исходное множество согласно разным правилам:
 - Разбиение множества согласно k-fold валидации: `sklearn.model_selection.KFold()`

```
from sklearn.model_selection import KFold

X = ["a", "b", "c", "d"]
kf = KFold(n_splits=2)
for train, test in kf.split(X):
    print("%s_ %s" % (train, test))
# out: [2 3] [0 1]
# out: [0 1] [2 3]
```

- Разбиение множества согласно leave-one-out валидации:
`sklearn.model_selection.LeaveOneOut()`

```
from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()
for train, test in loo.split(X):
    print("%s_ %s" % (train, test))
# out: [1 2 3] [0]
# out: [0 2 3] [1]
# out: [0 1 3] [2]
# out: [0 1 2] [3]
```



- Разбиение множества согласно множественной k-fold валидации:
`sklearn.model_selection.RepeatedKFold()`

```
from sklearn.model_selection import RepeatedKFold

rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=123456)
for train, test in rkf.split(X):
    print("%s_□%s" % (train, test))
# out (repeat 1): [1 2] [0 3]
# out (repeat 1): [0 3] [1 2]
# out (repeat 2): [0 1] [2 3]
# out (repeat 2): [2 3] [0 1]
```

⁴https://scikit-learn.org/stable/modules/cross_validation.html

- Разбиение множества согласно множественной k-fold валидации:
`sklearn.model_selection.RepeatedKFold()`

```
from sklearn.model_selection import RepeatedKFold

rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=123456)
for train, test in rkf.split(X):
    print("%s_□%s" % (train, test))
# out (repeat 1): [1 2] [0 3]
# out (repeat 1): [0 3] [1 2]
# out (repeat 2): [0 1] [2 3]
# out (repeat 2): [2 3] [0 1]
```

- Параметры в примерах: *n_splits* — количество блоков (*cv*), *n_repeats* — количество повторений

⁴https://scikit-learn.org/stable/modules/cross_validation.html

- Разбиение множества согласно множественной k-fold валидации:
`sklearn.model_selection.RepeatedKFold()`

```
from sklearn.model_selection import RepeatedKFold

rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=123456)
for train, test in rkf.split(X):
    print("%s_□%s" % (train, test))
# out (repeat 1): [1 2] [0 3]
# out (repeat 1): [0 3] [1 2]
# out (repeat 2): [0 1] [2 3]
# out (repeat 2): [2 3] [0 1]
```

- Параметры в примерах: *n_splits* — количество блоков (*cv*), *n_repeats* — количество повторений
- Остальные примеры работы с процедурой кросс-валидации можно посмотреть на сайте scikit-learn⁴

⁴https://scikit-learn.org/stable/modules/cross_validation.html

- Итак, мы умеем разбивать множество примеров на обучающее и тестовое подмножества

- Итак, мы умеем разбивать множество примеров на обучающее и тестовое подмножества
- Также умеем подразбивать обучающее подмножество для проведения процедуры валидации и даже замерять качество при этой процедуре

- Итак, мы умеем разбивать множество примеров на обучающее и тестовое подмножества
- Также умеем подразбивать обучающее подмножество для проведения процедуры валидации и даже замерять качество при этой процедуре
- Но главная цель — подобрать оптимальный набор гиперпараметров!

Настройка гиперпараметров: scikit-learn

- В scikit-learn есть два важных класса для этого:
`sklearn.model_selection.GridSearchCV` и
`sklearn.model_selection.RandomizedSearchCV`

Настройка гиперпараметров: scikit-learn

- В scikit-learn есть два важных класса для этого:
`sklearn.model_selection.GridSearchCV` и
`sklearn.model_selection.RandomizedSearchCV`
- `GridSearchCV` проходит по заранее определенному диапазону гиперпараметров

Настройка гиперпараметров: scikit-learn

- В scikit-learn есть два важных класса для этого:
`sklearn.model_selection.GridSearchCV` и
`sklearn.model_selection.RandomizedSearchCV`
- `GridSearchCV` проходит по заранее определенному диапазону гиперпараметров
- `RandomizedSearchCV` генерирует гиперпараметры случайно (согласно их заданным распределениям)



Настройка гиперпараметров: scikit-learn

- В scikit-learn есть два важных класса для этого:
`sklearn.model_selection.GridSearchCV` и
`sklearn.model_selection.RandomizedSearchCV`
- `GridSearchCV` проходит по заранее определенному диапазону гиперпараметров
- `RandomizedSearchCV` генерирует гиперпараметры случайно (согласно их заданным распределениям)
- Лучшие найденные параметры будут находиться в поле `best_params_`
- Точность на кросс-валидации для лучших гиперпараметров будет находиться в поле `best_score_`
- Точность на тесте для лучших гиперпараметров считается через метод `score()`



Настройка гиперпараметров: перебор по сетке

- Рассмотрим класс `sklearn.model_selection.GridSearchCV`

Настройка гиперпараметров: перебор по сетке

- Рассмотрим класс `sklearn.model_selection.GridSearchCV`
- Для перебора по сетке нужно задать **полный** диапазон поиска по каждому из гиперпараметров (либо оставить используемые по умолчанию)

```
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1.0, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
svc = svm.SVC()
clf = GridSearchCV(svc, param_grid, cv=5, refit=True) # Refit an estimator using the
              best found parameters on the whole dataset
clf.fit(X_train, y_train) # Run CV grid search
clf.score(X_test, y_test) # Test on the best parameters
# score = 0.91666666
```



Настройка гиперпараметров: случайный перебор

- Для случайного перебора нужно задать **распределение** каждого из гиперпараметров (либо оставить используемые по умолчанию), количество случайных попыток задается через параметр *n_iter* конструктора `RandomizedSearchCV`

Настройка гиперпараметров: случайный перебор

- Для случайного перебора нужно задать **распределение** каждого из гиперпараметров (либо оставить используемые по умолчанию), количество случайных попыток задается через параметр *n_iter* конструктора RandomizedSearchCV
- Распределения можно использовать из пакета `scipy.stats` (например, равномерное, нормальное, лог-равномерное и т.п.)

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform

param_dist = {'C': loguniform(0.1, 1000), 'gamma': loguniform(0.0001, 1.0)}
svc = svm.SVC()
clf = RandomizedSearchCV(svc, param_dist, cv=5, random_state=123456, refit=True) #
    Refit an estimator using the best found parameters on the whole dataset
clf.fit(X_train, y_train) # Run CV grid search
clf.score(X_test, y_test) # Test on the best parameters
# score can be different — in my experiments from 0.9 to 0.96666
```



Настройка гиперпараметров: сравнение методов перебора

- Сравним GridSearchCV и RandomizedSearchCV на решетке (количество генераций гиперпараметров) размера примерно 100

```
# GridSearchCV params: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}  
# GridSearchCV test score: 0.9833333333333333  
  
# RandomizedSearchCV params: {'C': 24.193643927107065, 'gamma':  
    0.0006301690253315956, 'kernel': 'rbf'}  
# RandomizedSearchCV test score: 0.9888888888888889
```

Настройка гиперпараметров: сравнение методов перебора

- Сравним GridSearchCV и RandomizedSearchCV на решетке (количество генераций гиперпараметров) размера примерно 100

```
# GridSearchCV params: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}  
# GridSearchCV test score: 0.9833333333333333  
  
# RandomizedSearchCV params: {'C': 24.193643927107065, 'gamma':  
    0.0006301690253315956, 'kernel': 'rbf'}  
# RandomizedSearchCV test score: 0.9888888888888889
```

- Как видно, разницы особой нет — и если заранее не нужно проверять **конкретные** значения гиперпараметров, то лучше ограничиться случайным поиском





Спасибо за внимание!