

Отлично, давайте начнем с первого урока!

Урок 1: Введение в Машинное Обучение и его Основные Парадигмы

Цель урока:

Понять, что такое машинное обучение (ML), какие существуют основные подходы, и какие задачи можно решать с его помощью.

Теоретическая часть (15 минут)

1. Что такое Машинное Обучение (ML)?

- Машинное обучение — это область искусственного интеллекта, которая позволяет системам автоматически учиться и улучшаться на основе опыта без явного программирования на каждый шаг.
- ML используется для решения задач, которые трудно описать алгоритмически, но которые можно решить на основе данных, таких как распознавание изображений, обработка естественного языка, прогнозирование и рекомендации.

2. Основные типы машинного обучения:

- **Обучение с учителем (Supervised Learning):** Модель обучается на размеченных данных, где входные данные (features) соответствуют известным выходным значениям (labels). Примеры: линейная регрессия, логистическая регрессия, деревья решений.
- **Обучение без учителя (Unsupervised Learning):** Модель обучается на неразмеченных данных, пытаясь найти скрытые структуры в данных. Примеры: кластеризация (k-means), методы понижения размерности (PCA).
- **Обучение с подкреплением (Reinforcement Learning):** Модель обучается через взаимодействие с окружением и получает обратную связь в виде наград и наказаний. Примеры: алгоритмы Q-learning, Deep Q-Networks (DQN).

3. Основные задачи и примеры алгоритмов для каждого типа:

- **Классификация:** Определение категории объекта (например, спам/не спам для писем).
- **Регрессия:** Прогнозирование непрерывных значений (например, прогноз цен на жилье).
- **Кластеризация:** Группировка объектов без предварительных меток (например, сегментация клиентов).
- **Поиск аномалий:** Обнаружение редких событий или наблюдений (например, обнаружение мошенничества).
- **Обучение с подкреплением:** Игра в шахматы или управление беспилотным автомобилем.

Практическая часть (15 минут)

Теперь давайте настроим окружение и выполним простую задачу классификации с использованием библиотеки `scikit-learn`.

1. Настройка окружения:

- Убедитесь, что у вас установлены Python и необходимые библиотеки: `scikit-learn`, `numpy`, `pandas`.
- Если нужно установить библиотеки, выполните следующие команды:

```
pip install numpy pandas scikit-learn
```

2. Пример задачи классификации с использованием `Iris dataset`:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Загрузка датасета Iris
iris = datasets.load_iris()
X = iris.data # Признаки
y = iris.target # Метки классов

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Создание и обучение модели логистической регрессии
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Прогнозирование и оценка точности
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Точность модели: {accuracy:.2f}")
```

Этот код выполняет несколько шагов:

- Загружает датасет `Iris` из `scikit-learn`.
- Разделяет данные на обучающую и тестовую выборки.
- Обучает модель логистической регрессии на обучающих данных.

- Делает прогнозы и оценивает точность модели на тестовых данных.

Что дальше?

Пожалуйста, попробуйте выполнить этот пример на вашей машине. Если возникнут вопросы по коду или теории, или если вы хотите обсудить конкретные моменты более подробно, дайте знать! Мы можем углубиться в любой аспект или перейти к следующему уроку.

Отличные вопросы! Давайте рассмотрим их по порядку.

Что из себя представляет датасет Iris?

Датасет Iris — один из самых известных и часто используемых учебных наборов данных в машинном обучении. Он содержит информацию о 150 экземплярах цветов ириса, принадлежащих к одному из трех видов:

1. **Iris-setosa**
2. **Iris-versicolor**
3. **Iris-virginica**

Каждый экземпляр представляет собой цветок, и для каждого цветка в датасете заданы 4 характеристики (признака), которые измерены в сантиметрах:

- **Длина чашелистика (sepal length)**
- **Ширина чашелистика (sepal width)**
- **Длина лепестка (petal length)**
- **Ширина лепестка (petal width)**

Датасет выглядит примерно так:

ID	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
...
50	7.0	3.2	4.7	1.4	Iris-versicolor
...

ID	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
100	6.3	3.3	6.0	2.5	Iris-virginica

Таким образом, датасет состоит из 50 экземпляров каждого вида ириса (итого 150 строк).

Что такое “признаки” и “метки классов”?

- **Признаки (Features):** Это атрибуты или свойства данных, которые используются для обучения модели. В контексте датасета Iris, признаки — это количественные характеристики каждого цветка ириса, такие как длина и ширина чашелистика и лепестка. В нашем примере это четыре числовых столбца:

- Длина чашелистика
- Ширина чашелистика
- Длина лепестка
- Ширина лепестка

Эти признаки используются моделью для того, чтобы “понять”, как отличить один вид ириса от другого.

- **Метки классов (Target Labels):** Это выходные значения или категории, которые модель пытается предсказать. В задачах классификации, таких как наш пример, метки классов представляют собой категории, к которым принадлежат данные. В случае с датасетом Iris, метки классов — это виды ирисов:

- Iris-setosa
- Iris-versicolor
- Iris-virginica

В `scikit-learn` эти классы обычно представлены в виде целых чисел: 0 для `Iris-setosa`, 1 для `Iris-versicolor`, и 2 для `Iris-virginica`.

Пример в контексте кода:

В приведенном выше примере кода:

- `x` (признаки) — это массив размером (150, 4), где каждая строка соответствует одному цветку, а каждый столбец — одному признаку.
- `y` (метки классов) — это массив размером (150,), который содержит метки классов (0, 1 или 2) для каждого цветка.

Когда мы обучаем модель на `x` и `y`, модель пытается найти зависимость между признаками (длиной и шириной чашелистика и лепестка) и метками классов (виды ирисов). После обучения

модель может предсказывать вид цветка, исходя из его измерений.

Если у вас есть еще вопросы или хотите углубиться в какой-то аспект, дайте знать!

Да, вы правильно понимаете основную идею функции `train_test_split`. Давайте рассмотрим её параметры более подробно, включая последние три параметра, которые вы упомянули.

Обзор функции `train_test_split`

Функция `train_test_split` из библиотеки `scikit-learn` используется для разделения исходного датасета на две (или больше) части: обучающую (`train`) и тестовую (`test`) выборки. Это важно для оценки производительности модели: модель обучается на обучающей выборке, а затем тестируется на данных, которые она не видела во время обучения (тестовая выборка).

Основные параметры `train_test_split`:

- `test_size`: Доля или абсолютное количество данных, которые будут использоваться для тестовой выборки. Например, `test_size=0.2` означает, что 20% данных будут использованы для тестирования. Если указано целое число, то это будет количество примеров в тестовой выборке.
- `train_size`: Доля или абсолютное количество данных, которые будут использоваться для обучающей выборки. Обычно, если указано значение `test_size`, `train_size` не требуется, так как оставшаяся часть данных используется для обучения.

Теперь давайте рассмотрим три параметра, которые вы упомянули:

1. `random_state`

- **Описание:**

Этот параметр используется для инициализации генератора случайных чисел, который отвечает за случайное разделение данных.

- Если установить `random_state` в определенное значение (например, `random_state=42`), вы получите одно и то же разделение данных при каждом выполнении кода. Это полезно для воспроизводимости результатов.
- Если оставить `random_state=None` (по умолчанию), генератор случайных чисел будет инициализирован системой (например, текущим временем), что приведет к разным результатам при каждом запуске функции.

- **Когда использовать:**

Используйте фиксированное значение `random_state` во время разработки или экспериментов, чтобы гарантировать, что вы всегда получите одно и то же разделение

данных. Это упрощает отладку и сравнение моделей. Если вас не интересует воспроизводимость (например, при финальном запуске модели), можно оставить `None`.

2. `shuffle`

- **Описание:**

Этот параметр определяет, нужно ли перемешивать данные перед разделением.

- По умолчанию `shuffle=True`, что означает, что данные будут случайным образом перемешаны перед разделением на обучающую и тестовую выборки. Это важно, чтобы избежать смещения, если данные изначально отсортированы каким-либо образом.
- Если `shuffle=False`, данные не будут перемешиваться, и разбиение будет производиться последовательно. Это может быть полезно в некоторых временных рядах или других специфических задачах, где порядок имеет значение.

- **Когда использовать:**

В большинстве случаев лучше оставить `shuffle=True`, чтобы убедиться, что выборка репрезентативна и не имеет смещения. Если порядок данных важен (например, временные ряды), тогда `shuffle` следует установить в `False`.

3. `stratify`

- **Описание:**

Этот параметр используется для стратифицированного разделения данных, чтобы гарантировать, что распределение целевой переменной (меток классов) в обучающей и тестовой выборках одинаково.

- Если указать `stratify=y`, то функция `train_test_split` постарается сделать так, чтобы пропорции каждого класса в обучающей и тестовой выборках соответствовали их пропорциям в исходных данных.
- Если оставить `stratify=None` (по умолчанию), то стратификация не выполняется.

- **Когда использовать:**

Рекомендуется использовать `stratify` в задачах классификации, особенно если у вас есть дисбаланс классов (например, один класс встречается гораздо чаще других). В противном случае тестовая выборка может не представлять весь набор данных.

Пример использования параметров:

```
from sklearn.model_selection import train_test_split

# Пример с параметрами random_state, shuffle и stratify
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)
```

В этом примере:

- `test_size=0.2` : 20% данных идут в тестовую выборку.
- `random_state=42` : Обеспечивает воспроизводимость разбиения данных.
- `shuffle=True` : Перемешивает данные перед разделением.
- `stratify=y` : Делает стратифицированное разделение, чтобы классы в `y` были пропорционально распределены в обучающей и тестовой выборках.

Использование этих параметров помогает создать сбалансированные и репрезентативные обучающую и тестовую выборки, что важно для правильной оценки моделей машинного обучения. Если у вас возникли еще вопросы или нужно подробнее объяснить какой-то из параметров, дайте знать!

Да, вы правильно понимаете общий процесс создания и обучения модели, а также значение параметра `sample_weight` для метода `fit`. Давайте рассмотрим это подробнее.

Создание модели и обучение

1. Создание модели:

```
model = LogisticRegression(max_iter=200)
```

Здесь мы создаем экземпляр модели логистической регрессии. `LogisticRegression` — это модель классификации, которая используется для прогнозирования вероятности принадлежности входных данных к одному из двух классов (в случае бинарной классификации) или к одному из нескольких классов (в случае многоклассовой классификации).

2. Обучение модели:

```
model.fit(X_train, y_train)
```

Метод `fit` используется для обучения модели на обучающих данных `X_train` и соответствующих метках классов `y_train`.

Параметры LogisticRegression

Целесообразно рассмотреть основные параметры `LogisticRegression`, так как они могут существенно повлиять на производительность и поведение модели. Вот некоторые ключевые параметры:

- **penalty** : Тип регуляризации, используемый для предотвращения переобучения. Значения могут быть:
 - `'l1'` : L1-регуляризация (Lasso).
 - `'l2'` : L2-регуляризация (Ridge). Это значение по умолчанию.
 - `'elasticnet'` : Комбинация L1 и L2.
 - `'none'` : Без регуляризации.
- **c** : Инверсия силы регуляризации; меньшие значения указывают на более сильную регуляризацию. По умолчанию `c=1.0`.
- **solver** : Алгоритм оптимизации, используемый для нахождения коэффициентов модели. Выбор солвера может зависеть от размера данных и наличия регуляризации. Например:
 - `'liblinear'` : Для маленьких датасетов и для L1-регуляризации.
 - `'saga'` : Подходит для больших датасетов и поддерживает `'elasticnet'`.
- **max_iter** : Максимальное количество итераций для солвера, чтобы сойтись. Значение по умолчанию — `100`. В примере установлено `200`, что может быть полезно, если модель не сходится за меньшее число итераций.
- **random_state** : Определяет генератор случайных чисел для инициализации солвера. Установка фиксированного значения этого параметра позволяет получить воспроизводимые результаты.

Параметр `sample_weight` метода `fit`

Параметр `sample_weight` действительно представляет собой массив весов той же длины, что и количество наблюдений в `X_train`.

- **Описание:**
`sample_weight` позволяет назначить каждому примеру из обучающей выборки свой вес. Это полезно, когда ваши данные неравномерно распределены или когда определенные примеры более важны для модели, чем другие.
- **Пример использования:**
Представим, что у вас есть датасет, где некоторые наблюдения являются более “значимыми” или “важными” для модели (например, редкие события или примеры, которые лучше представляют целевой класс). Вы можете назначить больший вес этим примерам, чтобы модель уделяла им больше внимания во время обучения.


```
# Пример задания весов
sample_weights = [1, 1, 2, 1, 1, 3, 1, 1, ...] # Должен иметь такую же длину, как X_train

# Обучение модели с учетом весов
model.fit(X_train, y_train, sample_weight=sample_weights)
```

- **Когда использовать:**

- В случае, когда у вас есть разбалансированные классы, и вы хотите компенсировать это, добавляя больший вес меньшему классу.
- Когда определенные наблюдения являются более “надежными” или “важными” для обучения модели.
- Когда у вас есть дополнительные данные о том, насколько часто встречаются или важны те или иные признаки в реальном мире.

Целесообразность разбора всех параметров LogisticRegression

На данном этапе было бы полезно знать о параметрах, которые могут существенно повлиять на обучение и результаты модели. Разбор всех параметров может быть полезен, но ключевые параметры, такие как `penalty`, `C`, `solver`, и `max_iter`, являются наиболее важными для понимания того, как модель будет обучаться и работать.

Если вам нужно углубиться в какой-то из параметров или вы хотите узнать больше о конкретном аспекте, пожалуйста, дайте знать! Мы также можем продолжить с другим уроком или темой по вашему выбору.

Да, вы правильно понимаете. Параметр `C` в `LogisticRegression` действительно определяет «вес» регуляризации: он является обратной величиной к силе регуляризации. Чем меньше значение `C`, тем сильнее регуляризация.

Давайте рассмотрим, что из себя представляют L1 (Lasso) и L2 (Ridge) регуляризации и как они работают.

Регуляризация в машинном обучении

Регуляризация — это техника, используемая для предотвращения переобучения (overfitting) модели, особенно когда модель слишком сложная и может подстраиваться под шум в данных.

В логистической регрессии (как и в других моделях) регуляризация добавляет штраф к функции потерь, который увеличивается с ростом значений коэффициентов модели. Это помогает

«упрощать» модель, делая её менее подверженной переобучению.

L1-регуляризация (Lasso)

- **Описание:**

L1-регуляризация, также известная как Lasso (Least Absolute Shrinkage and Selection Operator), добавляет к функции потерь сумму абсолютных значений коэффициентов модели (весов).

- **Формула:**

В случае L1-регуляризации, мы минимизируем следующую функцию потерь:

$$J(w) = \text{Loss}(w) + \lambda \sum_{j=1}^n |w_j|$$

где:

- $\text{Loss}(w)$ — стандартная функция потерь (например, логистическая для логистической регрессии).
- λ — коэффициент регуляризации (чем больше λ , тем сильнее регуляризация).
- w_j — коэффициенты модели (веса).

- **Эффект:**

L1-регуляризация способствует разреженности модели. Это означает, что многие коэффициенты модели могут быть сокращены до нуля. Таким образом, L1-регуляризация используется для выполнения отбора признаков (feature selection), так как она может полностью исключить некоторые из них.

- **Пример использования:**

В задачах, где у вас много признаков, и вы хотите найти наиболее важные из них, L1-регуляризация поможет «выключить» наименее важные признаки, присвоив их коэффициентам значение 0.

L2-регуляризация (Ridge)

- **Описание:**

L2-регуляризация, также известная как Ridge-регуляризация, добавляет к функции потерь сумму квадратов коэффициентов модели (весов).

- **Формула:**

В случае L2-регуляризации, мы минимизируем следующую функцию потерь:

$$J(w) = \text{Loss}(w) + \lambda \sum_{j=1}^n w_j^2$$

где:

- $\text{Loss}(w)$ — стандартная функция потерь.
- λ — коэффициент регуляризации.
- w_j — коэффициенты модели.

- **Эффект:**

L2-регуляризация накладывает штраф на большие коэффициенты, приводя их к меньшим значениям, но не к нулю. Это делает модель менее подверженной переобучению, так как она избегает сильных зависимостей от отдельных признаков.

- **Пример использования:**

L2-регуляризация полезна, когда у вас есть признаки, которые могут быть коррелированы или когда вы хотите уменьшить влияние отдельных признаков на модель, избегая переобучения.

Сравнение L1 и L2 регуляризаций

- **Сжатие коэффициентов:** L1 может привести коэффициенты к нулю, тогда как L2 приводит их к малым значениям, но не к нулю.
- **Отбор признаков:** L1 подходит для задач отбора признаков, так как оно приводит к разреженности модели. L2 не приводит к разреженности, но уменьшает значение всех коэффициентов.
- **Устойчивость к мультиколлинеарности:** L2 регуляризация лучше справляется с мультиколлинеарностью (когда признаки сильно коррелируют между собой).

Elastic Net

- **Описание:**

Elastic Net — это комбинация L1 и L2 регуляризаций, что позволяет получить преимущества обоих подходов: отбор признаков (L1) и уменьшение мультиколлинеарности (L2).

Используется в задачах, где у нас много признаков и коррелированные признаки.

- **Формула:**

$$J(w) = \text{Loss}(w) + \alpha \left(\lambda_1 \sum_{j=1}^n |w_j| + \lambda_2 \sum_{j=1}^n w_j^2 \right)$$

где λ_1 и λ_2 регулируют влияние L1 и L2 регуляризаций соответственно.

Примеры и наглядность

Чтобы лучше понять, как работают регуляризации, можно визуализировать зависимости между значениями параметров (весов) и их влиянием на функцию потерь. Регуляризация делает модель более «гладкой», ограничивая сложность модели и помогая избежать переобучения.

Если у вас есть еще вопросы или вы хотите, чтобы я подготовил пример с использованием L1 или L2 регуляризации, дайте знать!

Да, вы всё правильно понимаете!

Переобучение (Overfitting)

Переобучение (overfitting) — это ситуация, когда модель слишком хорошо подстраивается под обучающие данные, в том числе под их шум и случайные колебания. В результате такая модель плохо обобщает знания на новые, ранее не виденные данные. Это приводит к высокой точности на обучающем наборе данных и низкой точности на тестовом наборе.

Когда модель переобучена, она слишком сложна и имеет слишком много параметров, что позволяет ей выучить не только основные закономерности, но и шум в данных. Регуляризация помогает уменьшить вероятность переобучения, добавляя штраф за слишком большие коэффициенты модели.

Регуляризация L1 и L2: Подробное объяснение

- **L1-регуляризация (Lasso):**
 - **Как она работает:** L1-регуляризация добавляет штраф к функции потерь, пропорциональный сумме абсолютных значений коэффициентов. В результате некоторые коэффициенты могут стать равными нулю.
 - **Эффект:** L1-регуляризация может “обнулить” коэффициенты признаков, которые считаются менее значимыми для модели. Это делает модель более разреженной (модель имеет меньше ненулевых коэффициентов), что эффективно отбирает наиболее важные признаки. Например, если у вас есть 100 признаков, но только 10 действительно важны для прогнозирования, L1-регуляризация может уменьшить коэффициенты остальных 90 признаков до нуля, упрощая модель и улучшая её интерпретируемость.
- **L2-регуляризация (Ridge):**
 - **Как она работает:** L2-регуляризация добавляет штраф к функции потерь, пропорциональный сумме квадратов коэффициентов. Это не приводит коэффициенты к нулю, но уменьшает их значения, особенно если они велики.

- **Эффект:** L2-регуляризация “сглаживает” модель, уменьшая влияние каждого признака. Она делает это путем сокращения значений коэффициентов, чтобы модель не зависела слишком сильно от отдельных признаков. В отличие от L1, L2 не делает коэффициенты равными нулю, но заставляет модель распределять “вес” между связанными признаками, избегая слишком больших коэффициентов.

Сравнение L1 и L2 регуляризаций в контексте упрощения модели

1. L1 (Lasso) регуляризация:

- **Упрощение модели:** За счет “обнуления” некоторых коэффициентов, модель становится проще, так как она эффективно исключает менее важные признаки. Это приводит к более разреженной модели, которая лучше подходит для интерпретации и может предотвратить переобучение, удаляя нерелевантные признаки.
- **Когда использовать:** Полезно, когда у вас много признаков, и вы хотите сократить их количество, оставив только наиболее важные.

2. L2 (Ridge) регуляризация:

- **Упрощение модели:** Уменьшает значения всех коэффициентов, что делает модель менее чувствительной к отдельным признакам и предотвращает переобучение. Это полезно, когда признаки коррелируют друг с другом (мультиколлинеарность).
- **Когда использовать:** Предпочтительна, когда все признаки могут быть полезны, но вы хотите избежать больших коэффициентов и уменьшить влияние шума в данных.

Пример применения L1 и L2

Вот пример, показывающий использование L1 и L2 регуляризаций с логистической регрессией в Python:

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

# Загрузка датасета Iris
iris = load_iris()
X, y = iris.data, iris.target

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# L1-регуляризация (Lasso)
model_l1 = LogisticRegression(penalty='l1', solver='saga', C=1.0, max_iter=200)
model_l1.fit(X_train, y_train)

# L2-регуляризация (Ridge)
model_l2 = LogisticRegression(penalty='l2', solver='saga', C=1.0, max_iter=200)
model_l2.fit(X_train, y_train)

# Вывод коэффициентов для сравнения
print("Коэффициенты модели с L1-регуляризацией (Lasso):", model_l1.coef_)
print("Коэффициенты модели с L2-регуляризацией (Ridge):", model_l2.coef_)
```

В этом примере:

- Модель с L1-регуляризацией (`penalty='l1'`) может иметь несколько нулевых коэффициентов, если она считает, что некоторые признаки не значимы.
- Модель с L2-регуляризацией (`penalty='l2'`) будет иметь более сглаженные коэффициенты, но все они будут ненулевыми.

Итог

Вы правильно понимаете ключевую разницу между L1 и L2 регуляризациями:

- **L1 (Lasso):** Находит и “убирает” менее значимые признаки, делая их коэффициенты равными нулю.
- **L2 (Ridge):** Уменьшает значение всех коэффициентов, делая модель более устойчивой к переобучению, но не исключает признаки полностью.

Если вам нужно больше примеров или разъяснений по какому-либо из этих методов, не стесняйтесь спрашивать!