Глоссарий

- **Нейронная сеть** — это модель, вдохновлённая работой мозга. Она учится находить закономерности в данных.
- **Рекуррентная нейронная сеть (RNN)** — вид нейронной сети, которая умеет запоминать информацию из прошлого.
- **Память** — способность сети помнить, что было раньше и использовать это для принятия решений в будущем.
- **LSTM (Long Short-Term Memory)** — улучшенная версия RNN, которая лучше запоминает важную информацию и забывает ненужную, благодаря специальным механизмам управления памятью.
- **GRU (Gated Recurrent Unit)** — ещё одна улучшенная версия RNN, которая работает быстрее, чем LSTM, но тоже хорошо справляется с задачами памяти, используя более простую архитектуру.
- **Секвенциальные данные** — данные, которые идут друг за другом: текст, музыка, временные ряды, последовательности действий.
- **Гейт (ворота)** — механизмы внутри LSTM и GRU, которые решают, что запомнить, а что забыть.
- **Градиент** — значение, показывающее, как сильно изменится результат сети при изменении параметров. Используется для обучения модели.
- **Обучение** — процесс настройки нейронной сети для улучшения её работы на основе данных.
- **Transformer** — архитектура нейронной сети, предназначенная для работы с последовательностями данных, заменяя RNN и LSTM.
- **Self-Attention (внимание к самому себе)** — механизм, который помогает сети обращать внимание на важные части данных внутри последовательности.
- **Encoder (энкодер)** — часть трансформера, которая анализирует входные данные и создаёт их представление.
- **Decoder (декодер)** — часть трансформера, которая генерирует вывод на основе данных от энкодера.
- **Attention Head (голова внимания)** — компонент, который фокусируется на разных частях данных одновременно.
- **Multi-Head Attention (многоголовое внимание)** — несколько attention heads, работающих параллельно для лучшего понимания данных.
- **Position Encoding (позиционное кодирование)** — метод добавления информации о порядке элементов в последовательности, так как трансформеры не обрабатывают данные по порядку, как RNN.

- **Сверточная нейронная сеть (CNN)** — тип нейронной сети, специально разработанный для работы с изображениями и анализа визуальных данных.
- **Свертка (Convolution)** — операция, которая ищет важные признаки на изображении, такие как края, углы и текстуры, сканируя его с помощью фильтра.
- **Фильтр (Filter) или ядро (Kernel)** — маленькая матрица с весами, которая скользит по изображению, чтобы выявить специфические признаки.
- **Карта признаков (Feature Map)** — результат применения фильтров к изображению, показывающий, где были обнаружены важные элементы.
- **Пулинг (Pooling)** — метод уменьшения размера данных, сохраняя при этом наиболее важную информацию.
- **Макс-пулинг (Max Pooling)** — выбирает наибольшее значение в области, чтобы уменьшить размер данных и выделить важные признаки.
- **Релу (ReLU)** — функция активации, которая добавляет в модель нелинейность, помогая лучше обучаться.
- **ResNet (Residual Network)** — улучшенная архитектура CNN, которая позволяет строить очень глубокие сети без потери качества благодаря остаточным связям.
- **Остаточная связь (Residual Connection или Skip Connection)** — механизм, который позволяет пропускать данные через несколько слоев, минуя их, чтобы избежать потери информации.
- **Глубина сети** — количество слоев в нейронной сети.
- **Градиентный спуск** — алгоритм оптимизации, который помогает обучать нейронную сеть, изменяя веса для минимизации ошибки.
- **Переобучение (Overfitting)** — когда модель слишком хорошо запоминает обучающие данные и плохо работает на новых данных.
- **Машинное обучение (ML)** — область искусственного интеллекта, которая учит компьютеры делать выводы на основе данных.
- **Обучение с учителем (Supervised Learning)** — тип машинного обучения, когда модель обучается на размеченных данных (где известен правильный ответ).
- **Модель (Model)** — алгоритм, который принимает данные и делает предсказания или принимает решения.
- **Размеченные данные (Labeled Data)** — данные, которые содержат входные значения и правильные ответы.
- **Фича (Feature)** — характеристика или признак данных, используемый для обучения модели.
- **Целевая переменная (Target)** — правильный ответ, который модель должна предсказать.
- **Регрессия (Regression)** — задача прогнозирования числовых значений (например, прогнозирование температуры).

- **Классификация (Classification)** — задача определения категории объекта (например, распознавание, кошка это или собака).
- **Ошибка (Loss)** — разница между предсказанием модели и правильным ответом.
- **Градиентный спуск (Gradient Descent)** — алгоритм для минимизации ошибки и улучшения модели.
- **Обучающая выборка (Training Set)** — данные, на которых модель обучается.
- **Тестовая выборка (Test Set)** — данные, на которых проверяют, как хорошо обучилась модель.

Вопрос №1 Рекуррентные нейронные сети. RNN, LSTM, GRU

Рекуррентные нейронные сети (RNN)

Что это?

Обычные нейронные сети хорошо работают с картинками или числами. Но когда нужно анализировать последовательности, например текст, музыку или временные ряды, обычные сети не справляются. Для этого придумали **рекуррентные нейронные сети (RNN)**.

RNN умеет запоминать, что было раньше, чтобы использовать эту информацию в будущем. Представь, что ты читаешь книгу и понимаешь текущую главу, помня, что было в прошлых главах. Вот так работает RNN.

Как работает RNN?

В RNN информация идёт по цепочке: каждое новое "звено" цепи помнит, что было в предыдущих. Например, если мы анализируем текст:

1. Сеть видит первое слово и делает выводы.
2. Потом видит второе слово и думает: "Ага, я помню, что было первое слово, значит, могу лучше понять второе".
3. И так далее для каждого следующего слова.

Важный элемент RNN — это скрытое состояние (hidden state), которое обновляется на каждом шаге и хранит информацию о предыдущих шагах.

Проблемы RNN

- **Затухание градиента**: когда сеть забывает старую информацию слишком быстро. Это происходит, когда значения градиента становятся очень маленькими, и сеть не может

эффективно обучаться.
- **Взрыв градиента**: наоборот, когда значения градиента становятся слишком большими, что делает обучение нестабильным.

Эти проблемы ограничивают способность RNN запоминать информацию на долгие промежутки времени. Чтобы решить эти проблемы, придумали улучшенные версии RNN: **LSTM** и **GRU**.

LSTM (Long Short-Term Memory)

Что это?

LSTM — это умная версия RNN. Она знает, что нужно помнить важное, а ненужное забывать. Это как дневник: ты записываешь туда важные события, но не записываешь каждую мелочь.

Как работает LSTM?

Внутри LSTM есть специальные "ворота", которые помогают управлять потоком информации:

4. **Ворота забывания (forget gate)**: решает, какую информацию нужно забыть. Например, если старые данные больше не нужны, ворота их "удаляют".
5. **Ворота запоминания (input gate)**: решает, какую новую информацию стоит запомнить. Они анализируют входные данные и добавляют важную информацию в память.
6. **Ворота вывода (output gate)**: решает, какую информацию нужно использовать прямо сейчас для принятия решений.

Эти ворота работают как фильтры, помогая сети сохранять только важные данные.

Пример для понимания

- Текст: "Маша пошла в магазин. Она купила молоко."
- Чтобы понять, кто такая "она", нужно помнить про "Машу". LSTM не забудет об этом, правильно определив, что "она" — это Маша.

GRU (Gated Recurrent Unit)

Что это?

GRU — это ещё одна версия RNN, похожая на LSTM, но проще и быстрее. GRU работает почти так же, но у неё меньше "ворот", поэтому она быстрее обучается и требует меньше

вычислительных ресурсов.

Как работает GRU?

В GRU есть два главных "ворота":

7. **Ворота обновления (update gate)**: решает, сколько старой информации нужно оставить и как её комбинировать с новой.
8. **Ворота сброса (reset gate)**: решает, сколько новой информации нужно добавить и какие старые данные можно проигнорировать.

GRU легче настраивать и она быстрее работает на больших данных, что делает её популярной для задач, где важна скорость.

Пример для понимания

- Анализ временного ряда: прогнозирование температуры на завтра на основе данных за последние дни. GRU быстро обучается, запоминая нужные данные и отбрасывая лишние.

Итоги

- **RNN** хороша для работы с последовательностями, но плохо запоминает долгосрочную информацию.
- **LSTM** решает эту проблему с помощью умных "ворот" для управления памятью, помогая сети помнить важные данные дольше.
- **GRU** делает то же самое, но проще и быстрее, благодаря своей более лёгкой архитектуре.

Эти сети применяются для анализа текста, перевода языков, распознавания речи, прогнозирования временных рядов и многого другого.

Вопрос №2 Transformer

Что такое Transformer?

Transformer — это нейронная сеть, которая работает с последовательностями данных (текст, аудио и т.д.) без использования RNN. Вместо запоминания последовательности, как в RNN, он использует механизм внимания (attention), чтобы понимать, какие части данных важны.

Представь, что ты читаешь книгу и не просто следишь за каждым словом по порядку, а сразу видишь важные моменты на всей странице. Вот так работает Transformer.

Как работает Transformer?

Transformer состоит из двух частей:

9. **Энкодер (Encoder)**: принимает входные данные и преобразует их в более удобный для обработки вид.
10. **Декодер (Decoder)**: использует данные от энкодера, чтобы генерировать результаты (например, перевод текста на другой язык).

Важные элементы:

- **Self-Attention**: позволяет модели понять, какие части текста важны. Например, в предложении "Маша купила яблоки, потому что она была голодна" модель поймёт, что "она" относится к "Маше".
- **Multi-Head Attention**: несколько attention-механизмов работают параллельно, чтобы уловить разные аспекты данных.
- **Position Encoding**: добавляет информацию о порядке слов, чтобы модель знала, что "собака кусает человека" и "человек кусает собаку" — это разные вещи.

Пример для понимания

- **Перевод текста**: Transformer берёт предложение на английском языке и с помощью энкодера анализирует его, а декодер переводит на русский, учитывая важные слова и их взаимосвязь.

Преимущества Transformer

- Работает быстрее, чем RNN и LSTM, особенно с большими данными.
- Легче параллелить на современных компьютерах.
- Лучше справляется с задачами перевода, анализа текста и создания текста.

Вопрос №3 CNN. ResNet

Сверточные нейронные сети (CNN)

Что это?

Сверточные нейронные сети (CNN) — это особый тип нейронных сетей, который используется для обработки изображений и анализа визуальной информации. CNN может распознавать объекты на картинках, выделять важные детали и классифицировать изображения.

Представь, что ты смотришь на фотографию: сначала ты видишь общие очертания, затем замечаешь мелкие детали. CNN работает примерно так же: она сначала находит простые формы (например, линии и углы), а затем — более сложные детали (например, лицо или автомобиль).

Как работает CNN?

1. **Слой свертки (Convolutional Layer):** на этом этапе изображение анализируется с помощью фильтров. Фильтр скользит по изображению и "выделяет" важные детали, такие как границы объектов.
2. **Функция активации (ReLU):** после свертки применяется ReLU, чтобы модель могла обучаться сложным закономерностям, добавляя нелинейность в данные.
3. **Слой пулинга (Pooling Layer):** уменьшает размер данных, чтобы ускорить обучение и сделать модель более устойчивой к небольшим изменениям в изображении. Например, если объект чуть сдвинут, модель всё равно его распознает.
4. **Полносвязные слои (Fully Connected Layers):** объединяют все извлечённые признаки и принимают решение о классификации. Например, определяют, есть ли на изображении кот или собака.

Пример для понимания

- У нас есть фото кошки.
- **Слой свертки** находит контуры ушей и глаз.
- **ReLU** выделяет важные признаки.
- **Пулинг** уменьшает изображение, оставляя только значимые детали.
- **Полносвязный слой** анализирует эти признаки и делает вывод: "Это кошка!"

Проблемы классических CNN

- Трудности при обучении очень глубоких сетей (больше 20–30 слоев).
- Потеря информации при прохождении данных через множество слоев.
- Затухание градиента, когда обучение становится неэффективным в глубоких сетях.

Чтобы решить эти проблемы, была разработана архитектура **ResNet**.

ResNet (Residual Network)

Что это?

ResNet (Residual Network) — это архитектура нейронной сети, которая позволяет строить очень глубокие модели (сотни и даже тысячи слоев), не теряя качество обучения. ResNet

была создана для решения проблемы затухания градиента и ухудшения производительности в глубоких нейронных сетях.

Как работает ResNet?

В отличие от обычных CNN, ResNet использует **остаточные связи (Residual Connections)**. Они позволяют данным "перескакивать" через несколько слоев, минуя их. Это помогает сохранить важную информацию и ускорить обучение.

Остаточные блоки (Residual Blocks)

Остаточный блок — это комбинация нескольких свёрточных слоев с добавлением исходных данных (skip connection). Идея проста:

- Вместо того чтобы сеть училась напрямую сложной функции, она учится только небольшим изменениям (остаткам) по сравнению с входными данными.
- Итоговый результат = Входные данные + "Изменённые" данные после нескольких слоев.

Преимущества ResNet

- **Обучение очень глубоких сетей:** можно строить модели с сотнями слоев без ухудшения качества.
- **Сохранение информации:** остаточные связи помогают не терять важные данные.
- **Быстрое обучение:** ResNet обучается быстрее, чем классические глубокие сети.

Пример для понимания

- У нас есть фото собаки.
- Данные проходят через много слоев, и часть важной информации может потеряться.
- **ResNet добавляет остаточные связи**, чтобы избежать потери этих данных.
- В итоге модель уверенно распознаёт: "Это собака!"

Вопрос №4 Машинное обучение с учителем

Что это?

Обучение с учителем (Supervised Learning) — это тип машинного обучения, при котором модель обучается на размеченных данных. Это значит, что у нас есть данные (входные признаки) и правильные ответы (целевая переменная). Модель анализирует эти данные и учится находить закономерности, чтобы делать правильные предсказания в будущем.

Представь, что ты учишься распознавать фрукты. Тебе показывают яблоко и говорят: "Это яблоко". Потом показывают банан и говорят: "Это банан". Через некоторое время ты сможешь сам отличать яблоко от банана. Вот так работает обучение с учителем.

Как работает обучение с учителем?

1. **Сбор данных:** сначала нужно собрать данные. Например, изображения фруктов с подписями "яблоко", "банан" и т.д.
2. **Обработка данных:** данные подготавливаются для обучения, например, изображения преобразуются в числовые данные.
3. **Обучение модели:** модель анализирует данные, ищет закономерности и "учится" делать правильные предсказания.
4. **Оценка модели:** проверяют, как хорошо модель работает на новых данных, которые она не видела раньше.
5. **Использование модели:** модель используется для решения реальных задач, например, для автоматической сортировки фруктов на конвейере.

Виды задач в обучении с учителем

1. Классификация

Классификация — это задача, при которой нужно определить, к какой категории относится объект.

Примеры:

- Определение, является ли письмо спамом.
- Распознавание рукописных цифр.
- Классификация фотографий: кошка или собака.

2. Регрессия

Регрессия — это задача прогнозирования числового значения.

Примеры:

- Прогнозирование цены на жильё по характеристикам (площадь, район, количество комнат).
- Оценка температуры на завтра по текущим погодным условиям.
- Предсказание роста дохода компании.

Методы обучения с учителем

6. **Линейная регрессия:** простой алгоритм для прогнозирования числовых значений на основе линейной зависимости.
7. **Логистическая регрессия:** используется для задач классификации, чтобы определить вероятность того, что объект относится к определённой категории.
8. **Деревья решений (Decision Trees):** алгоритм, который принимает решения, двигаясь по "дереву" условий.
9. **Метод опорных векторов (SVM):** находит границу, которая разделяет данные на разные классы.
10. **Нейронные сети:** сложные модели, которые могут решать как задачи классификации, так и регрессии.

Пример для понимания

Задача: Определить, является ли животное кошкой или собакой.

11. **Сбор данных:** фотографии кошек и собак с подписями.
12. **Обработка данных:** преобразование изображений в числовой формат.
13. **Обучение модели:** модель анализирует фотографии и учится различать признаки (ушки, мордочка, хвост).
14. **Оценка модели:** проверяют модель на новых фото, которые она раньше не видела.
15. **Использование:** теперь модель может автоматически распознавать, кошка это или собака.

Преимущества и недостатки обучения с учителем

Преимущества:

- Высокая точность на размеченных данных.
- Прямой контроль за процессом обучения.
- Подходит для широкого круга задач (распознавание изображений, анализ текста, прогнозирование).

Недостатки:

- Требует большого количества размеченных данных.
- Может плохо работать на новых, неожиданных данных.
- Риск переобучения, если модель слишком хорошо запоминает обучающие данные.

Вопрос №5 Машинное обучение без учителя

Что это?

Обучение без учителя (Unsupervised Learning) — это тип машинного обучения, при котором модель обучается на неразмеченных данных. Это значит, что у нас есть только данные (входные признаки) без правильных ответов (целевых переменных). Модель пытается самостоятельно найти закономерности, структуры или скрытые зависимости в данных.

Представь, что тебе показывают корзину с разными фруктами, но никто не говорит, что это за фрукты. Ты начинаешь группировать их по цвету, форме или размеру, чтобы найти какие-то общие признаки. Вот так работает обучение без учителя.

Как работает обучение без учителя?

1. **Сбор данных:** сначала нужно собрать данные, например, изображения фруктов без подписей.
2. **Обработка данных:** данные подготавливаются для анализа, например, преобразуются в числовой формат.
3. **Обучение модели:** модель анализирует данные и ищет скрытые закономерности или группы.
4. **Интерпретация результатов:** результаты анализа интерпретируются, чтобы понять, что именно обнаружила модель.
5. **Использование модели:** модель применяется для кластеризации, выявления аномалий и других задач.

Виды задач в обучении без учителя

1. Кластеризация

Кластеризация — это задача группировки объектов на основе их сходства.

Примеры:

- Разделение клиентов на группы по поведению в интернет-магазине.
- Группировка новостей по темам.
- Анализ ДНК для выявления схожих генетических структур.

2. Поиск аномалий

Поиск аномалий — это задача выявления необычных данных, которые сильно отличаются от остальных.

Примеры:

- Обнаружение мошеннических операций в банке.

- Выявление сбоев в работе оборудования.
- Нахождение нетипичных покупок у клиентов.

3. Снижение размерности

Снижение размерности — это задача упрощения данных без потери важной информации.

Примеры:

- Визуализация сложных многомерных данных на двумерных графиках.
- Ускорение работы моделей за счёт уменьшения количества признаков.
- Удаление лишнего шума из данных.

Методы обучения без учителя

6. **Метод k-средних (k-means):** алгоритм для группировки данных в к кластеров на основе сходства.
7. **Иерархическая кластеризация:** создаёт древовидную структуру кластеров для анализа данных на разных уровнях.
8. **Метод главных компонент (PCA):** используется для снижения размерности данных и выделения главных признаков.
9. **Автокодировщики (Autoencoders):** нейронные сети, которые обучаются сжимать и восстанавливать данные, выявляя их скрытые закономерности.
10. **Методы плотностной кластеризации (DBSCAN):** группируют данные на основе плотности точек в пространстве.

Пример для понимания

Задача: Группировка фруктов без подписей.

11. **Сбор данных:** фотографии разных фруктов без указания их названий.
12. **Обработка данных:** преобразование изображений в числовой формат (например, по цвету, форме, размеру).
13. **Обучение модели:** алгоритм кластеризации группирует фрукты на основе сходства.
14. **Интерпретация результатов:** анализируем, какие группы сформировались — например, красные круглые фрукты в одной группе, жёлтые продолговатые — в другой.

Преимущества и недостатки обучения без учителя

Преимущества:

- Не требует размеченных данных.

- Помогает находить скрытые закономерности и структуры.
- Полезно для предварительного анализа данных и выявления аномалий.

Недостатки:

- Сложно интерпретировать результаты без экспертной оценки.
- Нет гарантии, что найденные закономерности будут полезны.
- Может быть чувствительно к настройкам алгоритмов и параметров.

Вопрос №6 Деревья решений и ансамблевые методы

Что это?

Деревья решений (Decision Trees) — это один из популярных алгоритмов машинного обучения, который используется как для задач классификации, так и для регрессии. Они представляют собой древовидную структуру, где каждый внутренний узел — это условие, ветви — это варианты ответов, а листья — конечные классы или числовые значения.

Представь, что ты хочешь купить новый телефон. Ты можешь задать себе вопросы: "Какой у меня бюджет?", "Какой бренд мне нравится?", "Какая камера лучше?". В зависимости от ответов ты сужаешь выбор, пока не найдёшь подходящую модель. Так работают деревья решений.

Как работают деревья решений?

1. **Сбор данных:** сначала нужно получить данные с признаками и целевыми значениями.
2. **Выбор признаков:** определяется, какие признаки лучше всего разделяют данные.
3. **Построение дерева:** алгоритм рекурсивно разбивает данные на основе оптимальных признаков.
4. **Обрезка дерева (Pruning):** устраняются лишние ветви, чтобы избежать переобучения.
5. **Использование модели:** дерево применяется для предсказаний на новых данных.

Виды задач для деревьев решений

1. Классификация

Классификационные деревья решений используются, когда целевой признак является категориальным.

Примеры:

- Определение, является ли клиент платёжеспособным (да/нет).
- Распознавание рукописных цифр.
- Классификация изображений фруктов.

2. Регрессия

Регрессионные деревья решений предсказывают числовые значения.

Примеры:

- Прогнозирование цены недвижимости.
- Определение количества заказов в ресторане на основе погоды.
- Оценка заработной платы по возрасту и стажу.

Как строятся деревья решений?

Алгоритм построения дерева решений включает несколько этапов:

6. **Выбор лучшего признака для разделения**
   - Используются критерии:
     - **Gini (индекс Джини)** – мера неоднородности классов в узле.
     - **Entropия (Entropy)** – степень неопределённости или хаотичности данных.
     - **MSE (среднеквадратическая ошибка)** – для регрессионных задач.
7. **Рекурсивное разбиение**
   - Данные делятся на подмножества, пока не будут достигнуты условия остановки (например, максимальная глубина дерева).
8. **Обрезка дерева (Pruning)**
   - Уменьшает переобучение, удаляя незначительные ветви.

Преимущества и недостатки деревьев решений

Преимущества:

- Простая интерпретация результатов.
- Может работать с разными типами данных.
- Не требует масштабирования признаков.

Недостатки:

- Склонность к переобучению (если дерево слишком глубокое).
- Нестабильность (малые изменения в данных могут сильно изменить структуру дерева).

- Не всегда хорошо работает с линейно разделимыми данными.

Ансамблевые методы

Ансамблевые методы (Ensemble Methods) — это техника, которая объединяет несколько моделей для улучшения точности предсказаний. Они работают по принципу "вместе мы сильнее", так как объединение слабых моделей часто даёт более устойчивые результаты.

Виды ансамблевых методов

1. Бэггинг (Bagging)

Бэггинг (Bootstrap Aggregating) — это метод, при котором создаётся множество деревьев решений на случайных подвыборках данных, а затем их предсказания усредняются.

Пример:

- **Random Forest (Случайный лес)** — строит много деревьев решений и усредняет их предсказания.

Преимущества:

- Снижает переобучение.
- Устойчив к шуму в данных.

Недостатки:

- Может быть медленным при большом количестве деревьев.

2. Бустинг (Boosting)

Бустинг (Boosting) — метод, при котором модели обучаются последовательно, каждая следующая исправляет ошибки предыдущей.

Примеры:

- **AdaBoost (Adaptive Boosting)** — обучает слабые модели (обычно деревья глубины 1) и корректирует их ошибки.
- **Gradient Boosting (Градиентный бустинг, GBM)** — использует градиентный спуск для оптимизации ошибок.
- **XGBoost (Extreme Gradient Boosting)** — улучшенная версия GBM с высокой скоростью.

- **LightGBM** — ускоренная реализация градиентного бустинга.

Преимущества:

- Высокая точность.
- Хорошо работает с небольшими и средними датасетами.

Недостатки:

- Может переобучаться.
- Чувствителен к шуму в данных.

3. Стэкинг (Stacking)

Стэкинг (Stacking) — это метод, в котором несколько моделей объединяются, а их предсказания подаются на вход другой модели (мета-алгоритму).

Пример:

- Комбинация логистической регрессии, случайного леса и градиентного бустинга с нейронной сетью как мета-моделью.

Преимущества:

- Может значительно повысить точность.
- Учитывает разные алгоритмы.

Недостатки:

- Более сложная настройка.
- Требует больших вычислительных ресурсов.

Пример применения ансамблевых методов

Задача: Определение вероятности одобрения кредита.

9. **Сбор данных:** История платежей, доход, возраст, количество кредитов.
10. **Обработка данных:** Заполнение пропущенных значений, нормализация.
11. **Обучение модели:**

- **Random Forest** для оценки общей важности признаков.
- **Gradient Boosting** для улучшения точности.

- **Stacking** для финального предсказания.

12. **Оценка модели:** ROC-кривая, точность, полнота.
13. **Использование:** Применение модели в банке для автоматической оценки заявок.

Заключение

Деревья решений:

✓ Простые и интерпретируемые.
✕ Склонны к переобучению.

Бэггинг:

✓ Устойчив к шуму.
✕ Требует больше ресурсов.

Бустинг:

✓ Высокая точность.
✕ Чувствителен к переобучению.

Стэкинг:

✓ Комбинирует сильные модели.
✕ Сложность настройки.

Вопрос №7 Подбор гиперпараметров моделей машинного обучения

Что это?

Подбор гиперпараметров (Hyperparameter Tuning) — это процесс поиска наилучших значений гиперпараметров модели машинного обучения, которые нельзя изменить во время обучения. Гиперпараметры определяют, как именно модель будет обучаться, а правильный их выбор влияет на итоговую точность и обобщающую способность модели.

Различие параметров и гиперпараметров

- **Параметры модели** — это значения, которые модель учит из данных во время обучения. Например, коэффициенты (веса) линейной регрессии или параметры слоёв нейронной сети.
- **Гиперпараметры модели** — это настройки, задаваемые до начала обучения, которые управляют процессом тренировки модели. Они включают глубину деревьев, скорость обучения, количество нейронов в скрытых слоях и т. д.

Почему подбор гиперпараметров важен?

Правильный подбор гиперпараметров критически важен, так как неправильные настройки могут привести к:

1. **Переобучению (overfitting)** — модель слишком хорошо подстраивается под обучающие данные, но плохо работает на новых данных.
2. **Недообучению (underfitting)** — модель слишком простая и не может выявить сложные закономерности в данных.
3. **Медленной или нестабильной тренировке** — если параметры подобраны плохо, модель может обучаться очень долго или не достигать хорошей сходимости.

Важные гиперпараметры в популярных моделях

В зависимости от типа модели машинного обучения, гиперпараметры могут отличаться.

1. Линейная регрессия

- **alpha (λ)** — коэффициент регуляризации (в Lasso и Ridge регрессии). Чем больше значение, тем сильнее модель штрафует большие коэффициенты, что снижает переобучение.
- **fit_intercept** — учитывать ли свободный член (intercept) в уравнении.

2. Деревья решений (Decision Trees)

- **max_depth** — максимальная глубина дерева решений (чем глубже, тем сложнее модель).
- **min_samples_split** — минимальное количество примеров, необходимое для разделения узла.
- **min_samples_leaf** — минимальное количество объектов в листе дерева.
- **criterion** — критерий разбиения (например, Gini или Entropy для классификации).

3. Случайный лес (Random Forest)

- **n_estimators** — количество деревьев в ансамбле.
- **max_features** — максимальное число признаков, используемых при построении каждого дерева.
- **bootstrap** — использовать ли случайную выборку данных при обучении деревьев.

4. Градиентный бустинг (XGBoost, LightGBM)

- **learning_rate (eta)** — шаг градиентного спуска (меньшее значение делает обучение медленнее, но повышает качество).
- **n_estimators** — количество деревьев в ансамбле.
- **max_depth** — глубина деревьев решений.
- **subsample** — процент выборки, используемый для построения каждого дерева.

5. Нейронные сети

- **learning_rate** — скорость обучения (определяет, насколько сильно изменяются веса модели на каждой итерации).
- **batch_size** — количество примеров, используемых за одну итерацию обновления весов.
- **epochs** — количество полных проходов по обучающему набору данных.
- **activation** — функция активации (ReLU, Sigmoid, Tanh).
- **dropout** — вероятность отключения нейронов в слое для предотвращения переобучения.

Методы подбора гиперпараметров

1. Grid Search (Перебор по сетке)

Grid Search (поиск по сетке) — это метод полного перебора всех возможных комбинаций гиперпараметров из заданного списка. Каждая комбинация тестируется, а затем выбирается та, которая даёт лучший результат.

Как работает?

4. Определяется диапазон значений для каждого гиперпараметра.
5. Генерируются все возможные комбинации параметров.
6. Обучается модель на каждой комбинации.
7. Выбирается лучшая конфигурация по метрике (например, accuracy, F1-score, RMSE).

Плюсы:

- Простота реализации.
- Гарантированно находит оптимальную комбинацию (если перебор полный).

Минусы:

- Медленный и требовательный к ресурсам, особенно при большом количестве параметров.

Пример кода с GridSearchCV (scikit-learn):

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Лучшие параметры:", grid_search.best_params_)
```

---

2. Random Search (Случайный поиск)

Вместо полного перебора, **Random Search (случайный поиск)** выбирает случайные комбинации гиперпараметров и тестирует их.

Как работает?

8. Определяются диапазоны значений гиперпараметров.
9. Генерируются случайные комбинации параметров.
10. Обучается модель на каждой комбинации.
11. Выбирается лучшая комбинация.

Плюсы:

- Быстрее, чем Grid Search.

- Хорошо работает, если некоторые параметры менее важны.

Минусы:

- Может пропустить оптимальные значения.

Пример кода с RandomizedSearchCV (scikit-learn):

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

param_dist = {
    'n_estimators': [10, 50, 100],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10]
}

random_search = RandomizedSearchCV(RandomForestClassifier(), param_dist,
n_iter=10, cv=5, scoring='accuracy')
random_search.fit(X_train, y_train)

print("Лучшие параметры:", random_search.best_params_)
```

---

3. Байесовская оптимизация

Байесовская оптимизация (Bayesian Optimization) — это метод, который использует вероятностную модель для предсказания лучших гиперпараметров.

Как работает?

12. Создаётся начальная выборка случайных гиперпараметров.
13. Строится модель (например, гауссовский процесс) для предсказания лучших значений.
14. Новые точки выбираются стратегически, а не случайно.
15. Оптимальный набор параметров ищется быстрее, чем при случайном поиске.

Плюсы:

- Требует меньше итераций, чем Grid Search и Random Search.
- Эффективен при сложных гиперпараметрических пространствах.

Минусы:

- Реализация сложнее, требует специальных библиотек (например, `hyperopt`, `optuna`).

Вывод

Какой метод выбрать?

- **Grid Search** – если гиперпараметров мало и вычислительные ресурсы не ограничены.
- **Random Search** – если параметры можно варьировать в широком диапазоне.
- **Байесовская оптимизация** – если нужно быстро найти лучшие параметры без полного перебора.

Итог

Подбор гиперпараметров – ключевой этап машинного обучения. Автоматизированные методы, такие как Grid Search, Random Search и Bayesian Optimization, помогают находить оптимальные настройки модели, увеличивая её точность и производительность.

Вопрос №8 Нейронные сети. Метод обратного распространения ошибки

Что такое нейронные сети?

Нейронные сети (Artificial Neural Networks, ANN) — это класс алгоритмов машинного обучения, вдохновленный работой биологических нейронов мозга. Они состоят из множества взаимосвязанных узлов (нейронов), организованных в слои. Нейронные сети могут решать сложные задачи, такие как обработка изображений, распознавание речи, прогнозирование временных рядов и многое другое.

Структура нейронной сети

Нейронная сеть состоит из трёх основных типов слоев:

1. **Входной слой (Input Layer)** — принимает входные данные.
2. **Скрытые слои (Hidden Layers)** — обрабатывают информацию и ищут закономерности.
3. **Выходной слой (Output Layer)** — выдаёт итоговый результат (например, класс объекта).

Каждый нейрон соединён с нейронами следующего слоя и передаёт им взвешенную сумму входных сигналов, обработанных с помощью функции активации.

Основные элементы нейронной сети

- **Вес (Weight, w)** — коэффициент, определяющий, насколько важно входное значение.
- **Смещение (Bias, b)** — дополнительный параметр, который помогает смещать функцию активации.
- **Функция активации (Activation Function, $f(x)$)** — преобразует входные данные в нужный диапазон.

Функции активации

Функции активации необходимы для преобразования входных данных в выходные значения нейрона. Они позволяют нейросети моделировать сложные нелинейные зависимости.

1. Сигмоидная функция (Sigmoid)

Формула:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Используется в бинарной классификации.
- Выходное значение находится в диапазоне $0,1$, что удобно для вероятностных предсказаний.
- Главный недостаток: **затухающий градиент** (малые значения градиента при больших $|x|$).

2. ReLU (Rectified Linear Unit)

Формула:

$$f(x) = \max(0, x)$$

- Быстро вычисляется.
- Хорошо работает в глубоких нейросетях, поскольку не имеет проблемы затухающих градиентов.
- Недостаток: **"умирающие" нейроны** — если входное значение отрицательное, градиент становится нулевым.

3. Гиперболический тангенс (Tanh)

Формула:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Значения функции находятся в диапазоне ((-1,1)), что позволяет центрировать данные.
- Улучшенная версия сигмоиды, так как градиенты больше.
- Всё ещё подвержена проблеме затухающих градиентов.

Метод обратного распространения ошибки (Backpropagation)

Метод обратного распространения ошибки (Backpropagation) — это алгоритм обучения нейронных сетей, который корректирует веса модели, минимизируя ошибку.

Основные этапы работы алгоритма

1. Прямой проход (Forward Pass)

- Входные данные проходят через слои сети.
- Каждый нейрон передаёт взвешенную сумму входных данных через функцию активации.
- На выходе получается предсказание модели.

2. Вычисление ошибки (Loss Calculation)

Ошибка вычисляется с помощью функции потерь. Для разных задач используются различные функции:

- **Среднеквадратичная ошибка (MSE, Mean Squared Error)** — для задач регрессии:

$$L = \frac{1}{N} \sum (y_{\text{true}} - y_{\text{pred}})^2$$

- **Кросс-энтропия (Cross-Entropy Loss)** — для задач классификации:

$$L = -\sum y_{\text{true}} \log(y_{\text{pred}})$$

3. Обратное распространение ошибки (Backward Pass)

Обратное распространение ошибки используется для вычисления градиентов и корректировки весов. Это делается с помощью **правила цепного дифференцирования**:

1. Вычисляется градиент ошибки по отношению к выходу нейрона.

2. Используется производная функции активации.
3. Ошибка передаётся назад по сети.

4. Обновление весов (Weight Update)

Веса обновляются с помощью **градиентного спуска (Gradient Descent)**:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

---

1. Сигмоидная функция (Sigmoid)

Формула:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Используется в бинарной классификации.
- Выходное значение находится в диапазоне (0,1), что удобно для вероятностных предсказаний.
- Главный недостаток: **затухающий градиент** (малые значения градиента при больших (|x|)).

2. ReLU (Rectified Linear Unit)

Формула:

$$f(x) = \max(0, x)$$

- Быстро вычисляется.
- Хорошо работает в глубоких нейросетях, поскольку не имеет проблемы затухающих градиентов.
- Недостаток: **"умирающие" нейроны** — если входное значение отрицательное, градиент становится нулевым.

3. Гиперболический тангенс (Tanh)

Формула:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Значения функции находятся в диапазоне (-1,1), что позволяет центрировать данные.

- Улучшенная версия сигмоиды, так как градиенты больше.
- Всё ещё подвержена проблеме затухающих градиентов.

Градиентный спуск и его вариации

Градиентный спуск — это метод оптимизации, который минимизирует функцию ошибки.

1. Обычный градиентный спуск (Batch Gradient Descent)

- Обновляет веса на основе всей обучающей выборки.
- Точный, но медленный.

2. Стохастический градиентный спуск (SGD)

- Обновляет веса после каждого примера.
- Быстрее, но шумнее.

3. Мини-пакетный градиентный спуск (Mini-Batch Gradient Descent)

- Компромисс между первым и вторым методом: обновление происходит по мини-батчам.

4. Адаптивные алгоритмы (Adam, RMSprop, Adagrad)

- Улучшают сходимость градиентного спуска.

Пример использования Adam в TensorFlow/Keras:

```
import tensorflow as tf

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Пример работы обратного распространения ошибки

Допустим, у нас есть простая сеть с одним скрытым слоем:

7. **Прямой проход**:
    - Вычисляем выходы нейронов.
    - Применяем функции активации.
    - Получаем финальный результат.
8. **Вычисление ошибки**:
    - Находим разницу между предсказанием и реальным значением.
9. **Обратное распространение**:
    - Вычисляем градиенты.
    - Обновляем веса.

Преимущества и недостатки метода обратного распространения ошибки

Преимущества:

✓ Позволяет обучать глубокие нейронные сети.
✓ Дает возможность автоматического обновления весов.
✓ Хорошо работает в сочетании с методами оптимизации (Adam, SGD).

Недостатки:

✕ Подвержен **затухающему градиенту** в глубоких сетях.
✕ Требует больших вычислительных мощностей.
✕ Чувствителен к выбору гиперпараметров (learning rate, batch size).

Итог

Метод обратного распространения ошибки является основой обучения нейронных сетей. Он позволяет минимизировать ошибку модели, используя градиентный спуск. В сочетании с современными методами оптимизации этот алгоритм делает нейронные сети мощным инструментом для решения сложных задач.

Вопрос №9 Обучение с подкреплением (Reinforcement Learning)

Что такое обучение с подкреплением?

Обучение с подкреплением (Reinforcement Learning, RL) — это метод машинного обучения, в котором агент учится принимать решения, взаимодействуя с окружающей средой. В отличие от обучения с учителем и без учителя, RL использует систему **вознаграждений и штрафов**, направляя агента к достижению **оптимальной стратегии**.

Основные элементы обучения с подкреплением:

1. **Агент (Agent)** — объект, который принимает решения и взаимодействует со средой.
2. **Среда (Environment)** — мир, в котором действует агент.
3. **Состояние (State, (S))** — текущее описание среды.
4. **Действие (Action, (A))** — выбор агента в данной ситуации.
5. **Вознаграждение (Reward, (R))** — числовая оценка за выполнение действия.
6. **Политика (Policy, (\pi))** — стратегия агента при выборе действий.
7. **Функция ценности (Value Function, (V(s)))** — оценка ожидаемого будущего вознаграждения.
8. **Функция действия-ценности (Q-функция, (Q(s, a)))** — оценка качества конкретного действия в данном состоянии.

Основной цикл обучения с подкреплением

Алгоритм RL обычно работает в следующем цикле:

1. Агент наблюдает текущее **состояние** (S_t).
2. Агент выбирает **действие** (A_t) на основе политики (\pi(S_t)).
3. Агент выполняет действие, изменяя состояние среды на (S_{t+1}).
4. Среда предоставляет **вознаграждение** (R_t).
5. Агент обновляет свою стратегию, используя полученную информацию.
6. Повторение процесса, пока не будет достигнута **оптимальная политика**.

Методы обучения с подкреплением

1. Методы обучения с использованием ценности (Value-Based Methods)

В этих методах агент учится **оценивать состояние** и выбирать действия, которые ведут к максимальному суммарному вознаграждению.

- **Функция ценности состояния**:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s\right]$$

  где (\gamma) — коэффициент дисконтирования, определяющий, насколько важны будущие награды.
- **Q-функция (Функция ценности действий)**:

$$Q(s,a) = \mathbb{E}\left[R_t + \gamma \max_{a'} Q(S_{t+1}, a') \mid S_t = s, A_t = a\right]$$

  Эта функция показывает, насколько хорошее действие (a) в состоянии (s).

Пример алгоритма: **Q-Learning**.

2. Методы обучения с использованием политики (Policy-Based Methods)

Эти методы оптимизируют стратегию агента, параметризуя политику

Функция градиента политики:

$$\nabla J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi(A_t \mid S_t, \theta) Q(S_t, A_t)\right]$$

Примеры алгоритмов: **REINFORCE, PPO, A2C**.

3. Методы обучения с испольованием актёра-критика (Actor-Critic Methods)

Комбинируют два подхода:

- **Actor** (Актёр) обновляет политику.
- **Critic** (Критик) оценивает действия, предоставляя ценность состояния.

Формула обновления актёра:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi(A_t \mid S_t, \theta) \cdot Q(S_t, A_t)$$

Примеры алгоритмов: **A2C, A3C, DDPG**.

Пример Q-Learning

Q-Learning — это один из самых известных алгоритмов RL, который использует Q-функцию для выбора наилучших действий.

Обновление Q-функции:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

где:

$$\alpha \quad \text{— скорость обучения (learning rate)},$$

$$\gamma \quad \text{— скорость обучения (learning rate)}$$

Шаги алгоритма Q-Learning:

1. **Инициализация**: Заполнить Q-таблицу случайными значениями.
2. Выбор действия: Использовать ϵ-жадную стратегию для выбора A_t
3. Выполнение действия: Обновить состояние S_{t+1} и получить вознаграждение R_t
4. **Обновление Q-значений**: Использовать формулу обновления.
5. **Повторять**, пока агент не обучится.

Пример на Python (Q-Learning)

```python
import numpy as np

# Параметры среды
states = 5
actions = 2
gamma = 0.9  # Коэффициент дисконтирования
alpha = 0.1  # Скорость обучения
epsilon = 0.1  # Вероятность случайного выбора действия

# Инициализация Q-таблицы
Q = np.zeros((states, actions))

def choose_action(state):
    if np.random.rand() < epsilon:
        return np.random.choice(actions)  # Случайное действие
```

```
    else:
        return np.argmax(Q[state])  # Действие с максимальным Q-значением

# Обучение агента
for episode in range(1000):
    state = np.random.randint(0, states)  # Случайное начальное состояние
    done = False

    while not done:
        action = choose_action(state)
        next_state = np.random.randint(0, states)  # Симуляция перехода
        reward = np.random.randn()  # Случайное вознаграждение

        # Обновление Q-значений
        Q[state, action] = Q[state, action] + alpha * (
            reward + gamma * np.max(Q[next_state]) - Q[state, action]
        )

        state = next_state

print("Обученная Q-таблица:\n", Q)
```

Преимущества и недостатки обучения с подкреплением

Преимущества:

✓ Позволяет агенту обучаться **без размеченных данных**.
✓ Эффективно решает сложные задачи **управления** и **оптимизации**.
✓ Может адаптироваться к **изменяющейся среде**.

Недостатки:

✕ **Медленное обучение** — агенту требуется много итераций.
✕ **Высокие вычислительные затраты**, особенно в средах с большим количеством состояний.
✕ **Баланс между исследованием (exploration) и использованием (exploitation)** сложно настроить.

Итог

Обучение с подкреплением — это мощный метод машинного обучения, применяемый в робототехнике, игровой индустрии, автономных системах и финансах. Различные подходы, такие как Q-Learning, Policy Gradient и Actor-Critic, позволяют агенту обучаться оптимальному поведению, взаимодействуя со средой и получая вознаграждения.

Вопрос №10 Обработка данных при работе с методами машинного обучения

Что такое обработка данных?

Обработка данных (Data Preprocessing) — это важный этап машинного обучения, на котором данные подготавливаются для использования в моделях. Качество обработки данных напрямую влияет на точность модели и её способность обобщать закономерности.

Обработка данных включает следующие этапы:
6. **Сбор данных** — получение информации из различных источников.
7. **Очистка данных** — удаление пропущенных и ошибочных значений.
8. **Преобразование данных** — нормализация, кодирование категориальных признаков.
9. **Разделение данных** — разбиение на обучающую и тестовую выборку.
10. **Генерация признаков** — создание новых информативных признаков.

---

1. Очистка данных

1.1 Обнаружение и обработка пропущенных значений

Пропущенные значения могут снижать качество модели. Способы их обработки:

- **Удаление строк или столбцов с пропущенными значениями** (если их немного).
- **Замена средним, медианой или модой**:

$$x_{\text{new}} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

- **Использование моделей для восстановления значений** (например, `KNNImputer`).

Пример кода на Python:

```
import pandas as pd
from sklearn.impute import SimpleImputer

df = pd.DataFrame({'A': [1, 2, None, 4], 'B': [None, 2, 3, 4]})
imputer = SimpleImputer(strategy='mean')
df_filled = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

1.2 Обнаружение и обработка выбросов

Выбросы — это аномальные значения, которые могут исказить обучение модели. Методы их обработки:

- Использование **z-оценки**:

$$z = \frac{x - \mu}{\sigma}$$

  Значения |z|>3 считаются выбросами.
- Использование **межквартильного размаха (IQR)**:

$$\text{IQR} = Q3 - Q1$$

  Выбросы: значения, выходящие за пределы

$$Q1 - 1.5 \times \text{IQR}\ \text{Q3} + 1.5 \times IQRQ3 + 1.5 \times \text{IQR}.$$

Пример кода:

```
import numpy as np

Q1 = df['A'].quantile(0.25)
Q3 = df['A'].quantile(0.75)
IQR = Q3 - Q1

df_filtered = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 *
IQR))).any(axis=1)]
```

---

2. Преобразование данных

2.1 Масштабирование (Normalization & Standardization)

Многие модели машинного обучения чувствительны к масштабам признаков, особенно алгоритмы, использующие расстояния между точками (например, kNN, SVM, линейная регрессия). Поэтому перед обучением данные часто масштабируют. Основные методы:

- **Мин-Макс нормализация (Min-Max Scaling)**
  Приводит все значения к диапазону ([0,1]) или ([-1,1]), что делает их сопоставимыми по масштабу:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- **Стандартизация (Z-score Normalization)**
  Преобразует данные так, чтобы среднее значение было равно 0, а стандартное отклонение — 1:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}$$

  где:

$$x \quad \text{— исходное значение}$$

$$\mu \quad \text{— среднее значение признака}$$

$$\sigma \quad \text{—стандартное отклонение}$$

Мин-Макс нормализация лучше подходит, когда данные ограничены фиксированным диапазоном, например, значения пикселей изображения. **Стандартизация** эффективнее, когда данные имеют нормальное распределение или содержат выбросы.

2.2 Кодирование категориальных признаков

Некоторые алгоритмы машинного обучения не работают с категориальными данными. Методы кодирования:

- **One-Hot Encoding** (разбивает категориальный признак на несколько бинарных):

```
Цвет:
Красный → [1,0,0]
Зелёный → [0,1,0]
Синий → [0,0,1]
```

- **Label Encoding** (преобразует категории в числа):

```
Красный → 0
Зелёный → 1
Синий → 2
```

Пример кода:

```python
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

label_encoder = LabelEncoder()
df['Category'] = label_encoder.fit_transform(df['Category'])
```

---

3. Разделение данных на обучающую и тестовую выборку

Разделение данных необходимо для оценки качества модели. Обычно используется разбиение:

- **Обучающая выборка (Training Set)** — 70-80%
- **Тестовая выборка (Test Set)** — 20-30%
- Иногда выделяется **валидационная выборка (Validation Set)** — 10-20% (при настройке гиперпараметров).

Пример кода:

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

---

4. Генерация признаков (Feature Engineering)

Этот этап позволяет создать новые информативные признаки. Методы:

- **Преобразование существующих признаков** (например, логарифмирование):

$$x_{\text{new}} = \log(x + 1)$$

- **Создание полиномиальных признаков**:

$$x_{\text{poly}} = x^2, x^3$$

- **Объединение категориальных данных** (например, комбинация "город + профессия").

Пример кода:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
```

Итог

Обработка данных — это ключевой этап машинного обучения, который включает очистку, масштабирование, кодирование и разделение данных. **Хорошо подготовленные данные значительно улучшают качество моделей** и позволяют добиться высокой точности прогнозирования.

Основные выводы:

✓ Очистка данных предотвращает ошибки, связанные с пропущенными значениями и выбросами.
✓ Масштабирование данных важно для алгоритмов, чувствительных к масштабу признаков.
✓ Кодирование категориальных признаков делает данные понятными для моделей.
✓ Разделение данных на обучающую и тестовую выборку позволяет избежать переобучения.
✓ Генерация признаков может значительно повысить точность модели.