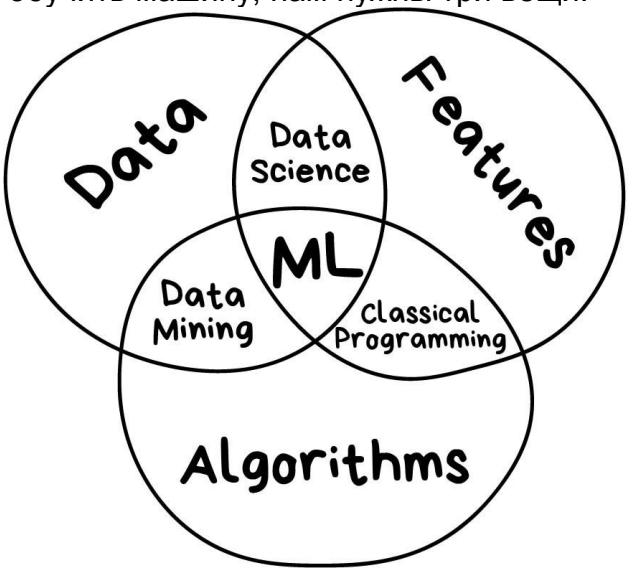
Введение в машинное обучение

Три составляющих обучения

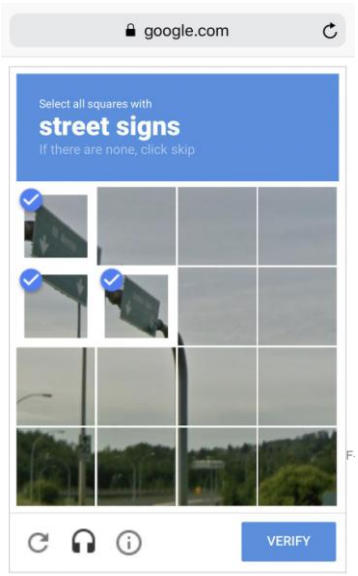
Цель машинного обучения – предсказать результат по входным данным. Чем разнообразнее входные данные, тем проще машине найти закономерности и тем точнее результат.

Итак, если мы хотим обучить машину, нам нужны три вещи:



Данные. Хотим определять спам – нужны примеры спам-писем, предсказывать курс акций – нужна история цен, узнать интересы пользователя – нужны его лайки или посты. Данных нужно как можно больше. Десятки тысяч примеров – это иногда самый минимум.

Данные собирают как могут. Кто-то вручную – получается дольше, меньше, зато без ошибок. Кто-то автоматически – просто сливает машине всё, что нашлось, и верит в лучшее. Самые хитрые, типа гугла, используют своих же пользователей для бесплатной разметки. Вспомните ReCaptcha, которая иногда требует «найти на фотографии все дорожные знаки» – это оно и есть.



За хорошими наборами данных (датасетами) идёт большая охота. Крупные компании, бывает, раскрывают свои алгоритмы, но датасеты – крайне редко.

Признаки. Мы называем их фичами (features). Фичи, свойства, характеристики, признаки – ими могут быть пробег автомобиля, пол пользователя, цена акций, даже счетчик частоты появления слова в тексте может быть фичей.

Машина должна знать, на что ей конкретно смотреть. Хорошо, когда данные просто лежат в табличках – названия их колонок и есть фичи. Когда признаков много, модель работает медленно и неэффективно. Зачастую отбор правильных фич занимает больше времени, чем всё остальное обучение. Но бывают и обратные ситуации, когда человек сам решает отобрать только «правильные» на его взгляд признаки и вносит в модель субъективность – модель начинает дико врать.

Алгоритм. Почти всегда одну задачу можно решить разными методами. От выбора метода зависит точность, скорость работы и размер готовой модели. Но есть один нюанс: если данные плохие, даже самый лучший алгоритм не поможет. Лучше не зацикливаться на процентах точности, а собрать побольше данных.

Обучение vs Интеллект

Искусственный интеллект – название всей области, как биология или химия.

Машинное обучение – это раздел искусственного интеллекта. Важный, но не единственный.

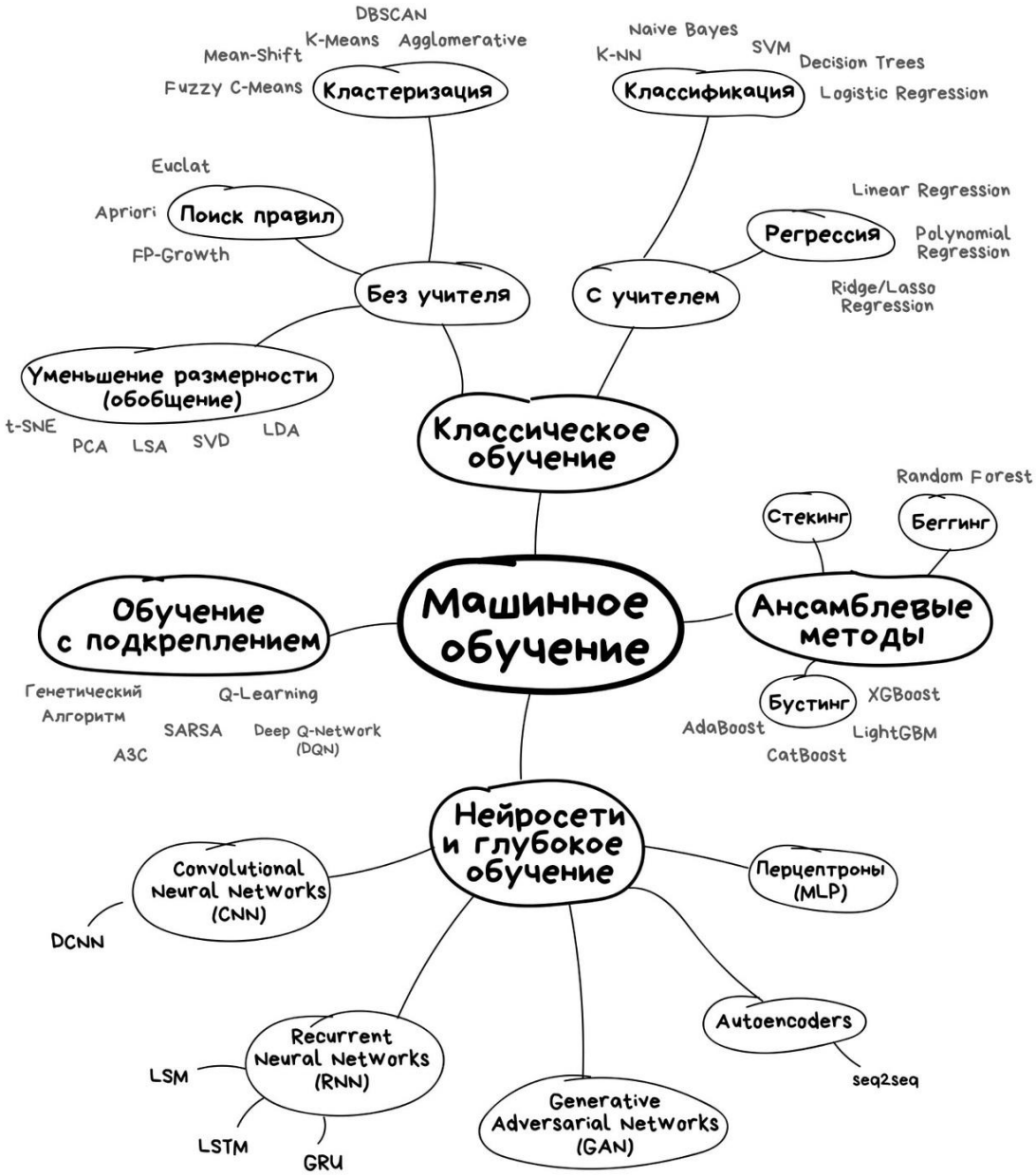
Нейросети – один из видов машинного обучения. Популярный, но есть и другие, не хуже.

Глубокое обучение – архитектура нейросетей, один из подходов к их построению и обучению. На практике сегодня мало кто отличает, где глубокие нейросети, а где не очень. Говорят название конкретной сети и всё (AlexNet, GoogleNet, ResNet т.д.).

Вот что машины сегодня умеют, а что не под силу даже самым обученным.

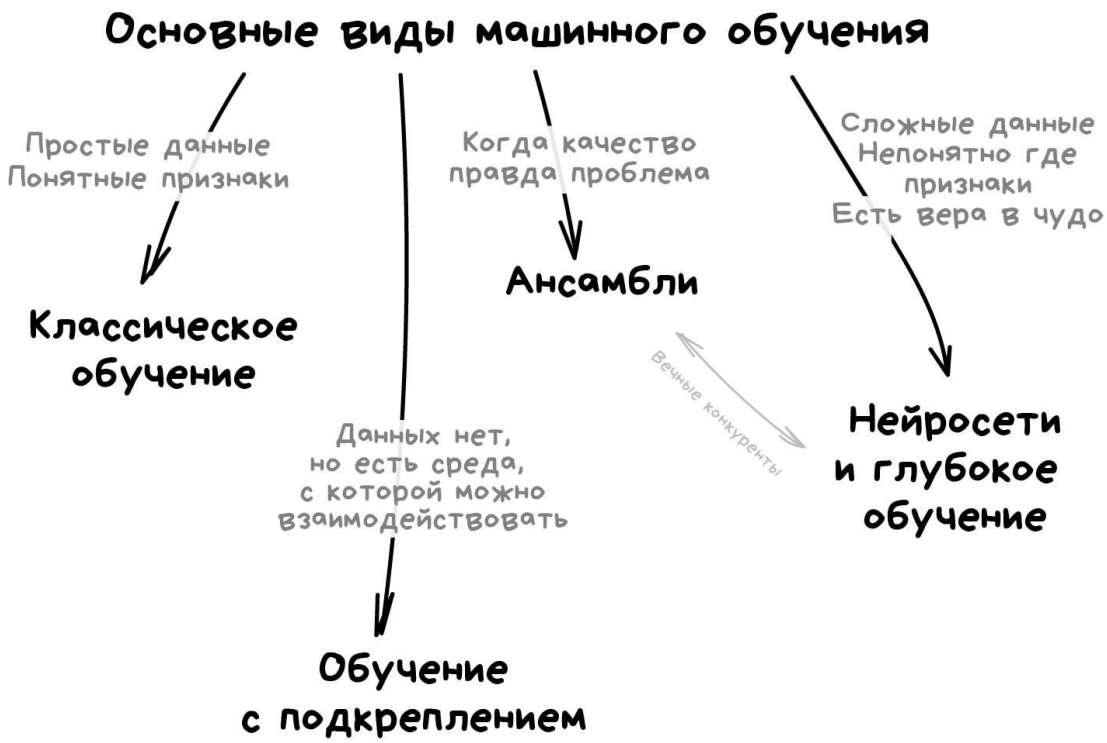
Машина может	Машина не может
Предсказывать	Создавать новое
Запоминать	Резко поумнеть
Воспроизводить	Выйти за рамки задачи
Выбирать лучшее	Убить всех людей

Карта мира машинного обучения



Классифицировать алгоритмы можно десятком способов. Я выбрал этот, потому что он мне кажется самым удобным для повествования. Надо понимать, что не бывает так, чтобы задачу решал только один метод. Я буду упоминать известные примеры применений, но держите в уме, что всё это может решить нейросетями.

Сегодня в машинном обучении есть всего четыре основных направления.



Часть 1. Классическое обучение

Первые алгоритмы пришли к нам из чистой статистики еще в 1950-х. Они решали формальные задачи – искали закономерности в цифрах, оценивали близость точек в пространстве и вычисляли направления.

Сегодня на классических алгоритмах держится добрая половина интернета. Когда вы встречаете блок «Рекомендованные статьи» на сайте, или банк блокирует все ваши деньги на карточке после первой же покупки кофе за границей – это почти всегда дело рук одного из этих алгоритмов.

Да, крупные корпорации любят решать все проблемы нейросетями. Потому что лишние 2% точности для них легко конвертируются в дополнительные 2 миллиарда прибыли. Остальным же стоит включать голову. Когда задача решаема классическими методами, дешевле реализовать сколько-нибудь полезную для бизнеса систему на них, а потом думать об улучшениях. А если вы не решили задачу, то не решить её на 2% лучше вам не особо поможет.

При всей своей популярности, классические алгоритмы настолько просты, что их очень легко объяснить. Сегодня они как основы арифметики – пригождаются постоянно.

Обучение с учителем.
Классическое обучение любят делить на две категории – с учителем и без. Часто можно встретить их английские наименования – Supervised и Unsupervised Learning.

В первом случае у машины есть некий учитель и это мы с вами, который говорит ей как правильно, а как нет. Рассказывает, что на этой картинке кошка, а на этой собака. То есть учитель уже заранее разделил (разметил) все данные на кошек и собак, а машина учится на конкретных примерах.

В обучении без учителя, машине просто вываливают кучу фотографий животных на стол и говорят «разберись, кто здесь на кого похож». Данные не размечены, у машины нет учителя, и она пытается сама найти любые закономерности.

Очевидно, что с учителем машина обучится быстрее и точнее, потому в боевых задачах его используют намного чаще. Эти задачи делятся на два типа: классификация – предсказание категории или класса объекта, и регрессия – предсказание места на числовой прямой.

Классификация.
«Разделение объектов по заранее известному признаку. Носки по цветам, документы по языкам, музыку по жанрам»

Сегодня используют для:
- Спам-фильтры
- Определение языка
- Поиск похожих документов
- Анализ тональности

- Распознавание рукописных букв и цифр
- Определение подозрительных транзакций

Популярные алгоритмы: Наивный Байес, Деревья Решений, Логистическая Регрессия, K-ближайших соседей, Машины Опорных Векторов

Классификация вещей – самая популярная задача во всём машинном обучении. Машина в ней как ребёнок, который учится раскладывать игрушки: роботов в один ящик, танки в другой.

Для классификации всегда нужен учитель – размеченные данные с признаками и категориями, которые машина будет учиться определять по этим признакам. Дальше классифицировать можно что угодно: пользователей по интересам – так делают алгоритмические ленты, статьи по языкам и тематикам – важно для поисковиков, музыку по жанрам – вспомните плейлисты Спотифая и Яндекс.Музыки, даже письма в вашем почтовом ящике.

Раньше все спам-фильтры работали на алгоритме Наивного Байеса. Машина считала сколько раз слово «виагра» встречается в спаме, а сколько раз в нормальных письмах. Перемножала эти две вероятности по формуле Байеса и складывала результаты всех слов.

Здесь P(B|A) – вероятность данного значения признака при данном классе, P(A) – априорная вероятность данного класса, P(B) – априорная вероятность данного значения признака.

Возьмем другой пример полезной классификации. Вот берёте вы кредит в банке. Как банку удостовериться, вернёте вы его или нет? Точно никак, но у банка есть тысячи профилей других людей, которые уже брали кредит до вас. Там указан их возраст, образование, должность, уровень зарплаты и главное – кто из них вернул кредит, а с кем возникли проблемы.

Но проблема в том, что банк не может слепо доверять ответу машины, без объяснений. Вдруг сбой, злые хакеры или админ решил скриптик поправить.

Для этой задачи придумали Деревья Решений. Машина автоматически разделяет все данные по вопросам, ответы на которые «да» или «нет». Вопросы могут быть не совсем адекватными с точки зрения человека, например «зарплата заёмщика больше, чем 25934 рубля?», но машина придумывает их так, чтобы на каждом шаге разбиение было самым точным.

Так получается дерево вопросов. Чем выше уровень, тем более общий (абстрактный) вопрос. Потом даже можно загнать их аналитикам, и они навыдумывают почему так.

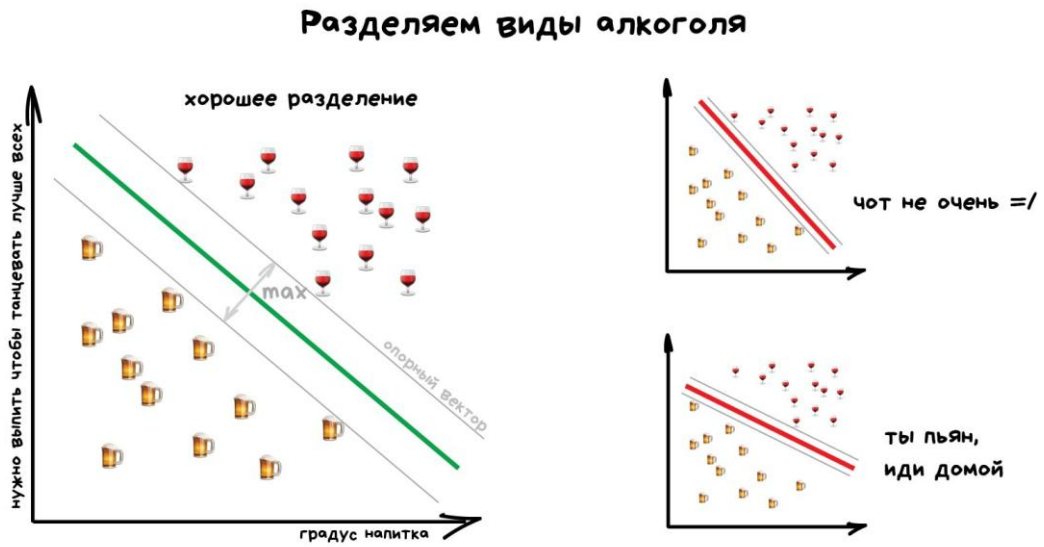
Деревья нашли свою нишу в областях с высокой ответственностью: диагностике, медицине, финансах.

В чистом виде деревья сегодня используют редко, но вот их ансамбли лежат в основе крупных систем и зачастую работают лучше нейросетей. Например, когда вы задаете вопрос Яндексу, именно толпа деревьев бежит ранжировать вам результаты.

Дерево Решений

Но самым популярным методом классической классификации заслуженно является Метод Опорных Векторов (SVM). Им классифицировали уже всё: виды растений, лица на фотографиях, документы по тематикам и т.д. Много лет он был главным ответом на вопрос «какой бы мне взять классификатор».

Идея SVM по своей сути проста – он ищет, как так провести две параллельные прямые (коридор) между категориями, чтобы между ними образовался наибольший зазор. На картинке видно нагляднее:



Метод Опорных Векторов

У классификации есть полезная обратная сторона – поиск аномалий. Когда какой-то признак объекта сильно не вписывается в наши классы, мы ярко подсвечиваем его на экране. Сейчас так делают в медицине: компьютер подсвечивает врачу все подозрительные области МРТ или выделяет отклонения в анализах. На биржах таким же образом определяют нестандартных игроков, которые скорее всего являются инсайдерами. Научив компьютер «как правильно», мы автоматически получаем и обратный классификатор – как неправильно.

Сегодня для классификации всё чаще используют нейросети, ведь по сути их для этого и изобрели.

Правило буравчика такое: сложнее данные – сложнее алгоритм. Для текста, цифр, табличек можно использовать классические методы и алгоритмы. Там модели меньше, обучаются быстрее и работают понятнее. Для картинок, видео и других больших данных – лучше сразу использовать нейросети.

Регрессия.
«Нарисуй линию вдоль точек. Да, это машинное обучение»

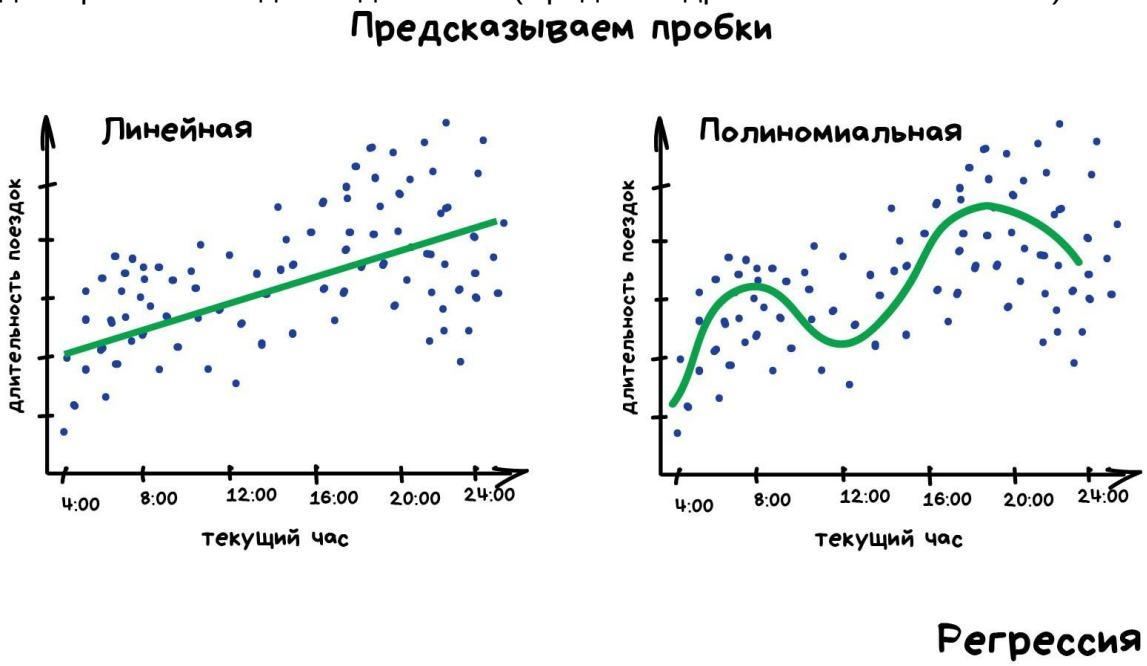
Сегодня используют для:

- Прогноз стоимости ценных бумаг
- Анализ спроса, объема продаж
- Медицинские диагнозы
- Любые зависимости числа от времени

Популярные алгоритмы: Линейная или Полиномиальная Регрессия

Регрессия – та же классификация, только вместо категории мы предсказываем число. Стоимость автомобиля по его пробегу, количество пробок по времени суток, объем спроса на товар от роста компании и.т.д. На регрессию идеально ложатся любые задачи, где есть зависимость от времени.

Регрессию очень любят финансисты и аналитики, она встроена даже в Excel. Внутри всё работает, опять же, банально: машина тупо пытается нарисовать линию, которая в среднем отражает зависимость. Правда, в отличии от человека с фломастером и вайтбордом, делает она это математически точно – считая среднее расстояние до каждой точки (Среднеквадратическое отклонение).



Когда регрессия рисует прямую линию, её называют линейной, когда кривую – полиномиальной. Это два основных вида регрессии, дальше уже начинаются редкие методы. Но также есть Логистическая Регрессия (или logit-функция), которая на самом деле не регрессия, а метод классификации, от чего у всех постоянно путаница.

Схожесть регрессии и классификации подтверждается еще и тем, что многие классификаторы, после небольшого тюнинга, превращаются в регрессоры. Например, мы можем не просто смотреть к какому классу принадлежит объект, а запоминать, насколько он близок – и вот, у нас регрессия.

Обучение без учителя.
Обучение без учителя (Unsupervised Learning) было изобретено позже, аж в 90-е, и на практике используется реже. Но бывают задачи, где у нас просто нет выбора.

Размеченные данные, как уже было сказано, дорогая редкость. Но что делать если мы хотим, например, написать классификатор автобусов – идти на улицу руками фотографировать миллион икарусов и подписывать где какой?

Либо, можно попробовать обучение без учителя. Хотя, честно говоря, из своей практики я не помню чтобы где-то оно сработало хорошо.

Обучение без учителя, всё же, чаще используют как метод анализа данных, а не как основной алгоритм.

Кластеризация.
«Разделяет объекты по неизвестному признаку. Машина сама решает как лучше»

Сегодня используют для:

- Сегментация рынка (типов покупателей, лояльности)
- Объединение близких точек на карте
- Сжатие изображений
- Анализ и разметки новых данных
- Детекторы аномального поведения

Популярные алгоритмы: Метод K-средних, Mean-Shift, DBSCAN

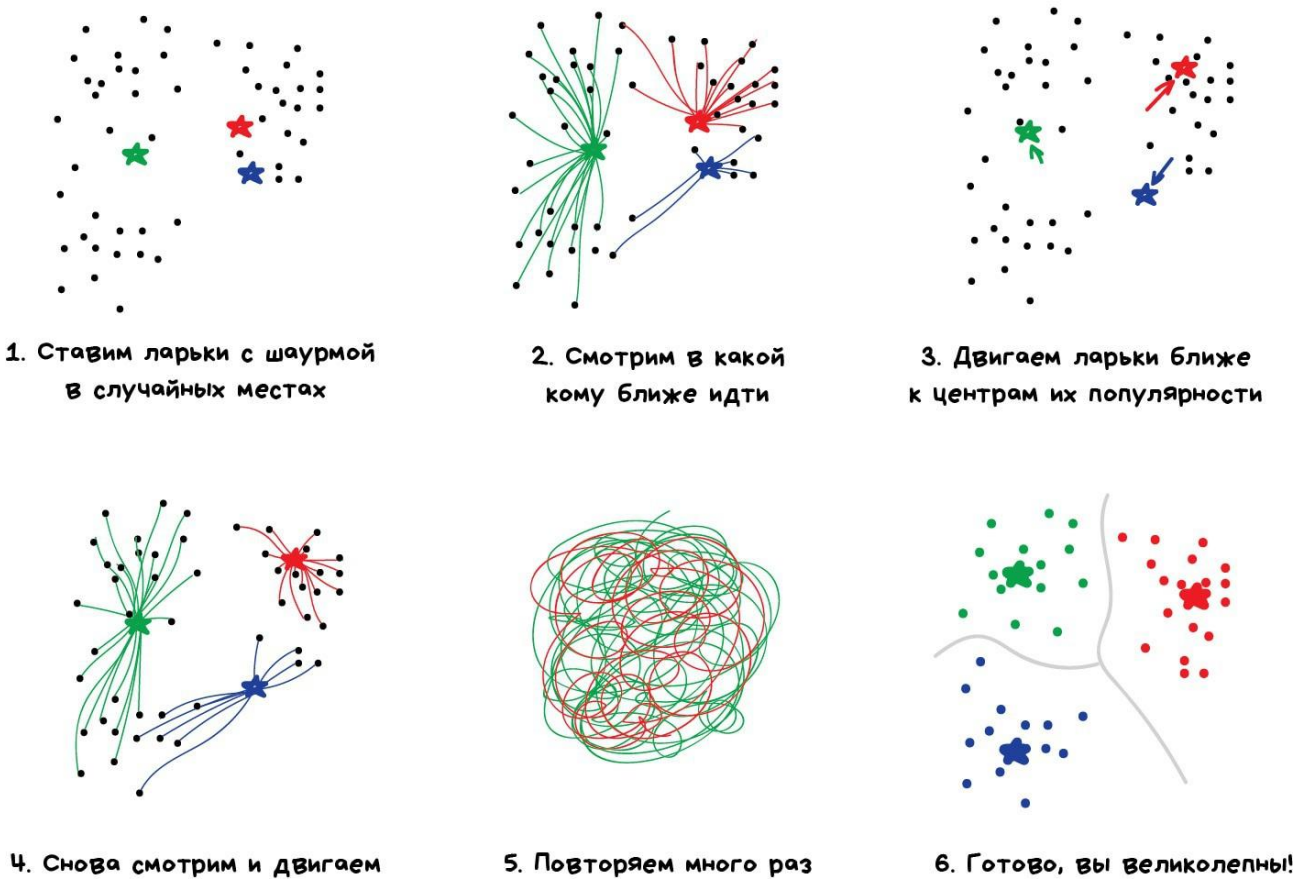
Кластеризация – это классификация, но без заранее известных классов. Она сама ищет похожие объекты и объединяет их в кластеры. Количество кластеров можно задать заранее или доверить это машине. Похожесть объектов машина определяет по тем признакам, которые мы ей разметили – у кого много схожих характеристик, тех собирай в один класс.

Отличный пример кластеризации – маркеры на картах в вебе. Когда вы ищете все бары в городе, приходится группировать их в кружочки с циферкой, иначе браузер зависнет в потугах нарисовать миллион маркеров.

Более сложные примеры кластеризации можно вспомнить в приложениях iPhoto или Google Photos, которые находят лица людей на фотографиях и группируют их в альбомы. Приложение не знает как зовут ваших друзей, но может отличить их по характерным чертам лица. Типичная кластеризация.

Сжатие изображений – еще одна популярная проблема. Сохраняя картинку в PNG, вы можете установить палитру, скажем, в 32 цвета. Тогда кластеризация найдёт все «примерно красные» пиксели изображения, высчитает из них «средний красный» и заменит все красные на него. Меньше цветов – меньше файл.

Проблема только, как быть с цветами типа Cyan – вот он ближе к зеленому или синему? Тут нам поможет популярный алгоритм кластеризации – Метод К-средних (K-Means). Мы случайным образом бросаем на палитру цветов наши 32 точки, обзывая их центроидами. Все остальные точки относим к ближайшему центроиду от них — получаются как бы созвездия из самых близких цветов. Затем двигаем центроид в центр своего созвездия и повторяем пока центроиды не перестанут двигаться. Кластеры обнаружены, стабильны и их ровно 32 как и надо было.



Искать центроиды удобно и просто, но в реальных задачах кластеры могут быть совсем не круглой формы. Вот вы геолог, которому нужно найти на карте схожие по структуре горные породы – ваши кластеры не только будут вложены друг в друга, но вы ещё и не знаете сколько их вообще получится.

Как и классификация, кластеризация тоже может использоваться как детектор аномалий. Поведение пользователя после регистрации резко отличается от нормального? Заблокировать его и создать тикет саппорту, чтобы проверили бот это или нет. При этом нам даже не надо знать, что есть «нормальное поведение» – мы просто выгружаем все действия пользователей в модель, и пусть машина сама разбирается кто тут нормальный.

Работает такой подход, по сравнению с классификацией, не очень хорошо.

Уменьшение Размерности (Обобщение).
«Собирает конкретные признаки в абстракции более высокого уровня»

Сегодня используют для:

- Рекомендательные Системы (★)
- Красивые визуализации
- Определение тематики и поиска похожих документов
- Анализ фейковых изображений
- Риск-менеджмент

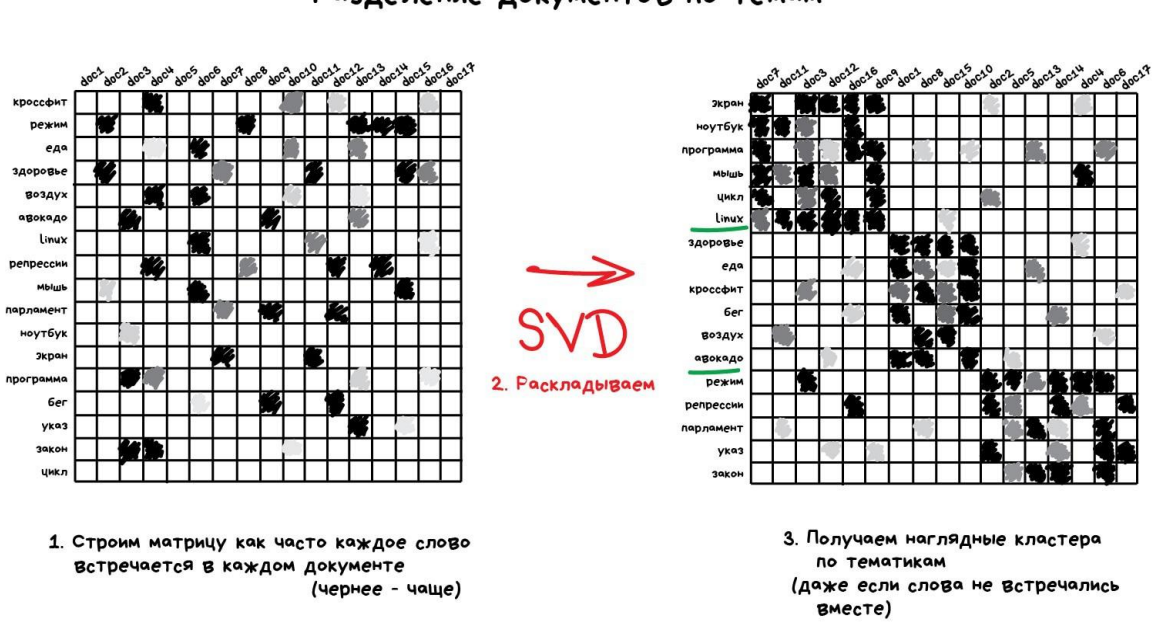
Популярные алгоритмы: Метод главных компонент (PCA), Сингулярное разложение (SVD), Латентное размещение Дирихле (LDA), Латентно-семантический анализ (LSA, pLSA, GLSA), t-SNE (для визуализации)

Изначально это были методы тех Data Scientist'ов, которым сгружали две фуры цифр и говорили найти там что-нибудь интересное. Когда просто строить графики в экселе уже не помогало, они придумали напрячь машины искать закономерности вместо них. Так у них появились методы, которые назвали Dimension Reduction или Feature Learning.

Для нас практическая польза их методов в том, что мы можем объединить несколько признаков в один и получить абстракцию. Например, собаки с треугольными ушами, длинными носами и большими хвостами соединяются в полезную абстракцию «овчарки». Да, мы теряем информацию о конкретных овчарках, но новая абстракция всяко полезнее этих лишних деталей. Плюс, обучение на меньшем количестве размерностей идёт сильно быстрее.

Инструмент на удивление хорошо подошел для определения тематик текстов (Topic Modelling). Мы смогли абстрагироваться от конкретных слов до уровня смыслов даже без привлечения учителя со списком категорий. Алгоритм назвали Латентно-семантический анализ (LSA), и его идея была в том, что частота появления слова в тексте зависит от его тематики: в научных статьях больше технических терминов, в новостях о политике – имён политиков. Да, мы могли бы просто взять все слова из статей и кластеризовать, как мы делали с ларьками выше, но тогда мы бы потеряли все полезные связи между словами, например, что батарейка и аккумулятор, означают одно и то же в разных документах.

Нужно как-то объединить слова и документы в один признак, чтобы не терять эти скрытые (латентные) связи. Отсюда и появилось название метода. Оказалось, что Сингулярное разложение (SVD) легко справляется с этой задачей, выявляя для нас полезные тематические кластеры из слов, которые встречаются вместе.



Другое популярное применение метода уменьшения размерности нашли в рекомендательных системах и коллаборативной фильтрации. Оказалось, если абстрагировать ими оценки пользователей фильмам, получается неплохая система рекомендаций кино, музыки, игр и чего угодно.

Полученная абстракция будет с трудом понимаема мозгом, но когда исследователи начали пристально рассматривать новые признаки, они обнаружили, что какие-то из них явно коррелируют с возрастом пользователя (дети чаще играли в Майнкрафт и смотрели мультфильмы), другие с определёнными жанрами кино, а третьи вообще с синдромом поиска глубокого смысла.

Машина, не знавшая ничего кроме оценок пользователей, смогла добраться до таких высоких материй, даже не понимая их.

Часть 2. Обучение с подкреплением

«Брось робота в лабиринт и пусть он сам ищет выход»

Сегодня используют для:

- Самоуправляемых автомобилей
- Роботов пылесосов
- Игр
- Автоматической торговли
- Управления ресурсами предприятий

Популярные алгоритмы: Q-Learning, SARSA, DQN, A3C, Генетический Алгоритм

Наконец мы дошли до вещей, которые, вроде, выглядят как настоящий искусственный интеллект.

Обучение с подкреплением используют там, где задачей стоит не анализ данных, а выживание в реальной среде.

Средой может быть даже видеоигра. Роботы, играющие в Марио, были популярны еще лет пять назад. Средой может быть реальный мир. Как пример – автопилот Теслы, который учится не сбивать пешеходов, или роботы-пылесосы, главная задача которых – напугать вашего кота с максимальной эффективностью.

Знания об окружающем мире такому роботу могут быть полезны, но чисто для справки. Не важно сколько данных он соберёт, у него всё равно не получится предусмотреть все ситуации. **Потому его цель – минимизировать ошибки, а не рассчитать все ходы**. Робот учится выживать в пространстве с максимальной выгодой: собранными монетками в Марио или временем поездки в Тесле.

Выживание в среде и есть идея обучения с подкреплением.

Часть 3. Ансамбли

«Куча глупых деревьев учится исправлять ошибки друг друга»

Сегодня используют для:

- Всего, где подходят классические алгоритмы (но работают точнее)
- Поисковые системы (★)
- Компьютерное зрение
- Распознавание объектов

Популярные алгоритмы: Random Forest, Gradient Boosting

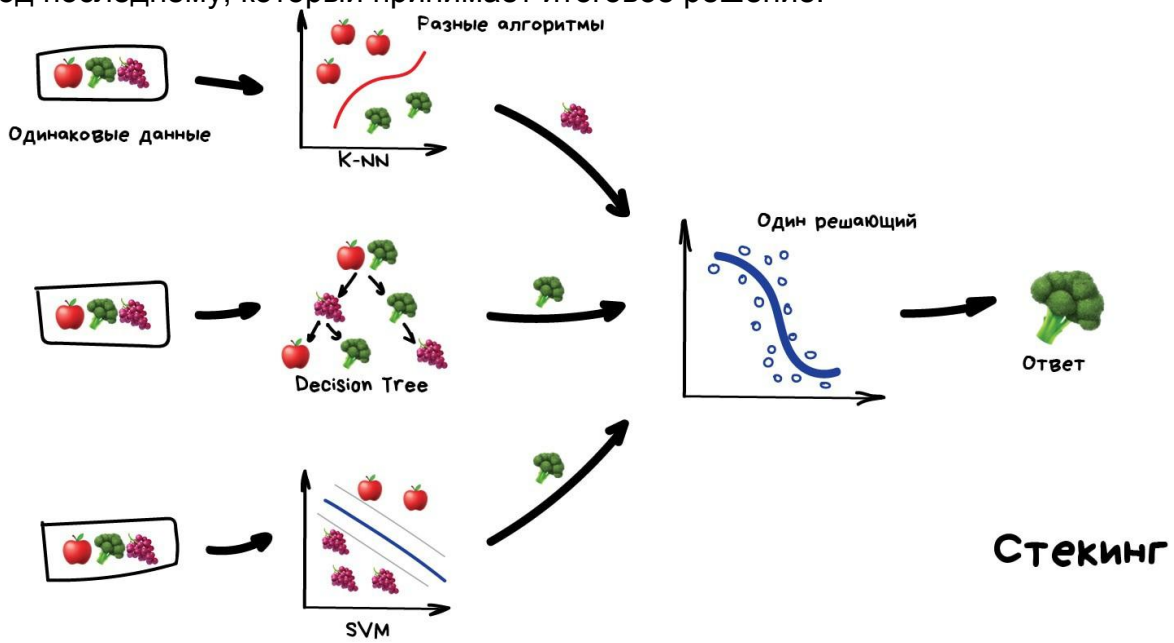
Ансамбли и нейросети – дают самые точные результаты и используются всеми крупными компаниями в продакшене.

При всей их эффективности, идея очень проста. Оказывается, если взять несколько не очень эффективных методов обучения и обучить исправлять ошибки друг друга, качество такой системы будет сильно выше, чем каждого из методов по отдельности.

Причём даже лучше, когда взятые алгоритмы максимально нестабильны и сильно плавают от входных данных. Поэтому чаще берут Регрессию и Деревья Решений, которым достаточно одной сильной аномалии в данных, чтобы поехала вся модель. А вот Байеса и K-NN не берут никогда – они хоть и простые, но очень стабильные.

Ансамбль можно собрать как угодно, хоть случайно. За точность, правда, тогда никто не ручается. Потому есть три проверенных способа делать ансамбли.

Стекинг. Обучаем несколько разных алгоритмов и передаём их результаты на вход последнему, который принимает итоговое решение.

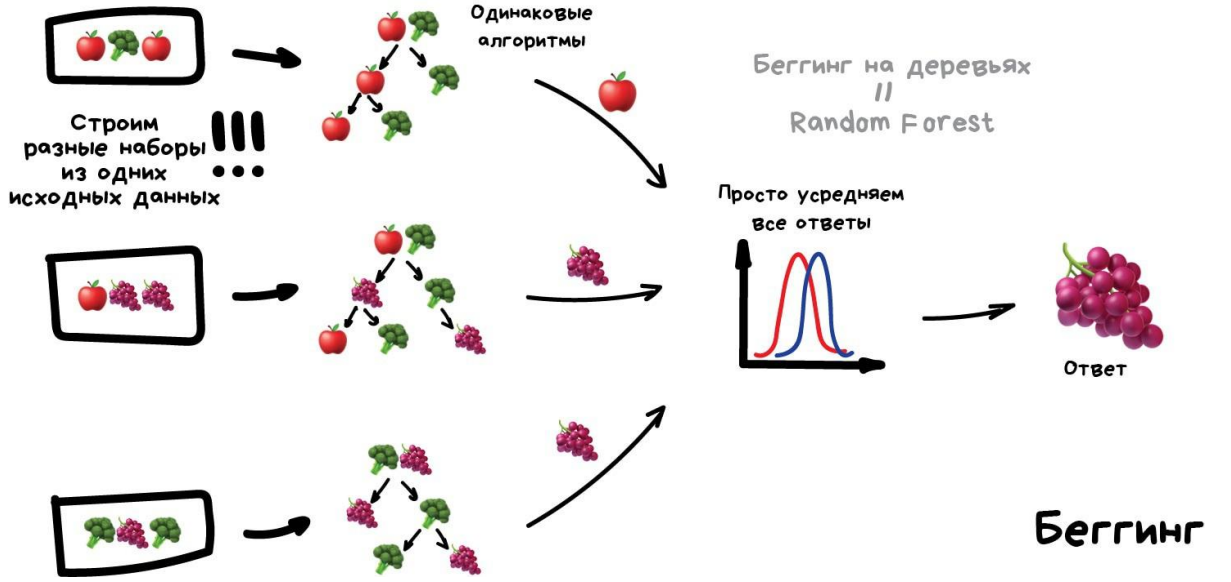


Ключевое слово – разных алгоритмов, ведь один и тот же алгоритм, обученный на одних и тех же данных не имеет смысла. Каких – ваше дело, разве что в качестве решающего алгоритма чаще берут регрессию.

Чисто из опыта – стекинг на практике применяется редко, потому что два других метода обычно точнее.

Беггинг. Он же Bootstrap AGGregatING. Обучаем один алгоритм много раз на случайных выборках из исходных данных. В самом конце усредняем ответы.

Данные в случайных выборках могут повторяться, то есть из набора 1-2-3 мы можем делать выборки 2-2-3, 1-2-2, 3-1-2 и так пока не надоест. На них мы обучаем один и тот же алгоритм несколько раз, а в конце вычисляем ответ простым голосованием.

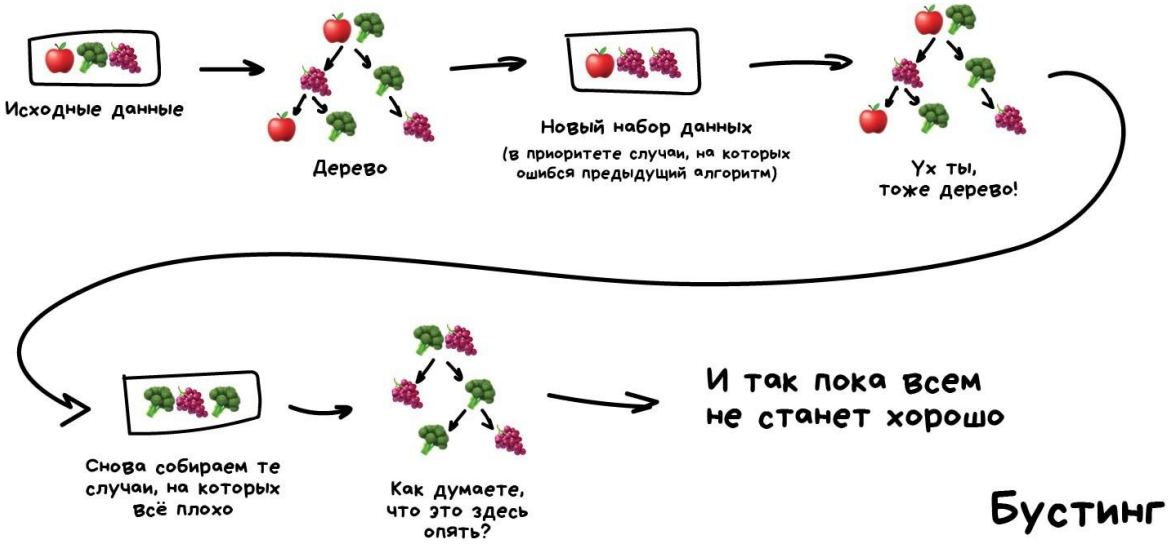


Самый популярный пример беггинга – алгоритм Random Forest, беггинг на деревьях, который и нарисован на картинке. Когда вы открываете камеру на телефоне и видите как она очертила лица людей в кадре желтыми прямоугольниками – скорее всего это их работа. Нейросеть будет слишком медлительна в реальном времени, а беггинг идеален, ведь он может считать свои деревья параллельно на всех шейдерах видеокарты.

Дикая способность параллелиться даёт беггингу преимущество даже над следующим методом, который работает точнее, но только в один поток. Хотя можно разбить на сегменты и запустить в несколько потоков.

Бустинг. Обучаем алгоритмы последовательно, каждый следующий уделяет особое внимание тем случаям, на которых ошибся предыдущий.

Как в беггинге, мы делаем выборки из исходных данных, но теперь не совсем случайно. В каждую новую выборку мы берём часть тех данных, на которых предыдущий алгоритм отработал неправильно. То есть, как бы доучиваем новый алгоритм на ошибках предыдущего.



Плюсы – высокая точность классификации.
Минусы уже названы – не параллелится, хотя всё равно работает быстрее нейросетей.

Нужен реальный пример работы бустинга – откройте Яндекс и введите запрос. Слышите, как Матрикснет грохочет деревьями и ранжирует вам результаты? Вот это как раз оно, Яндекс сейчас весь на бустинге.

Часть 4. Нейросети и глубокое обучение

«У нас есть сеть из тысячи слоёв, десятки видеокарт, но мы всё еще не придумали где это может быть полезно. Пусть рисует котиков и сочиняет песни!»

Сегодня используют для:

- Вместо всех вышеперечисленных алгоритмов вообще
- Определение объектов на фото и видео
- Распознавание и синтез речи
- Обработка изображений, перенос стиля
- Машинный перевод

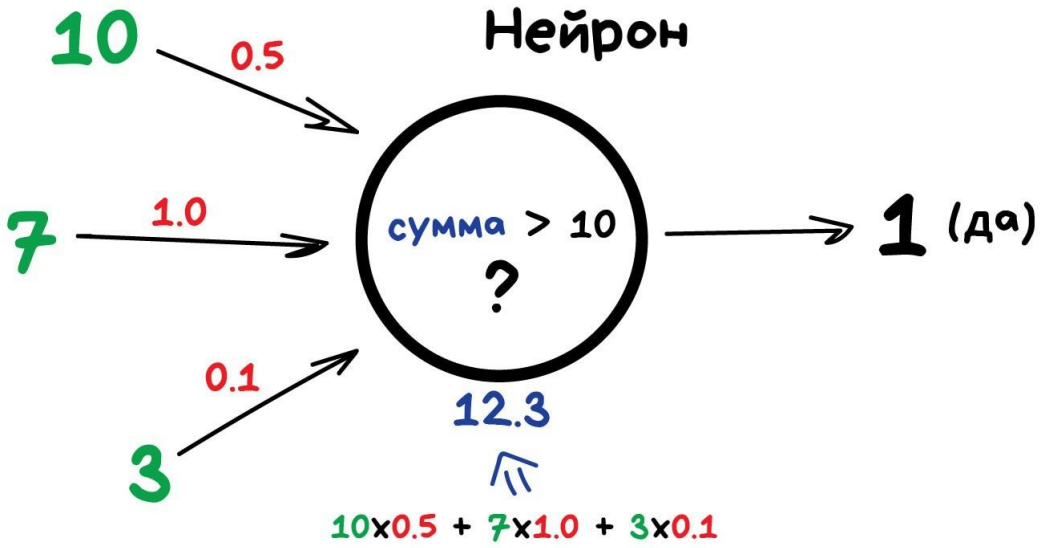
Популярные архитектуры: Перцептрон, Свёрточные Сети (CNN), Рекуррентные Сети (RNN), Автоэнкодеры

Любая нейросеть – это набор нейронов и связей между ними. Нейрон лучше всего представлять просто как функцию с кучей входов и одним выходом. Задача нейрона – взять числа со своих входов, выполнить над ними функцию и отдать результат на выход. Простой пример полезного нейрона: просуммировать все цифры со входов, и если их сумма больше N – выдать на выход единицу, иначе – ноль.

Связи или синапсы – это каналы, через которые нейроны шлют друг другу информацию. У каждой связи есть свой вес – её единственный параметр, который можно условно представить как прочность связи. Когда через связь с весом 0.5 проходит число 10, оно превращается в 5. Сам нейрон не разбирается, что к нему

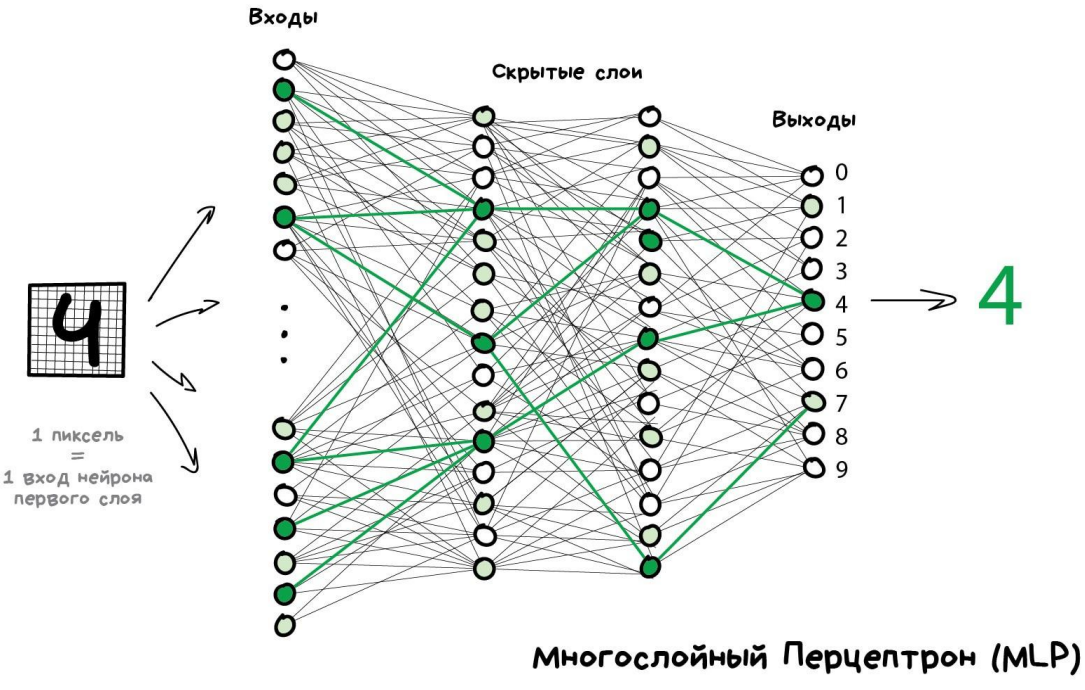
пришло и суммирует всё подряд – вот веса и нужны, чтобы управлять на какие входы нейрон должен реагировать, а на какие нет.

Условие перехода (если больше 10, то 1, если меньше, то 0) определяется функцией активации: линейные, пороговые, гиперболический тангенс и т.д.



Чтобы сеть не превратилась в анархию, нейроны решили связывать не как захочется, а по слоям. Внутри одного слоя нейроны никак не связаны, но соединены с нейронами следующего и предыдущего слоя. Данные в такой сети идут строго в одном направлении – от входов первого слоя к выходам последнего.

Если взять достаточное количество слоёв и правильно расставить веса в такой сети, получается следующее – подав на вход, скажем, изображение написанной от руки цифры 4, чёрные пиксели активируют связанные с ними нейроны, те активируют следующие слои, и так далее и далее, пока в итоге не загорится самый выход, отвечающий за четвёрку. Результат достигнут.



В реальном программировании, естественно, никаких нейронов и связей не пишут, всё представляют матрицами и считают матричными произведениями, потому что нужна скорость.

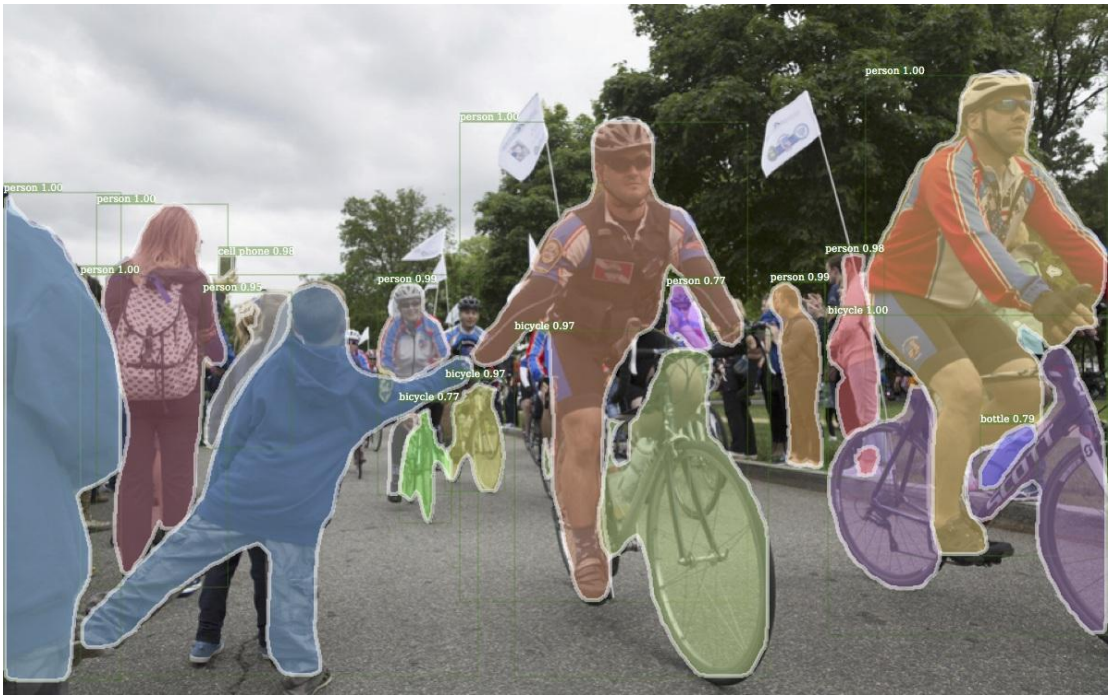
Когда мы построили сеть, наша задача правильно расставить веса, чтобы нейроны реагировали на нужные сигналы. Тут нужно вспомнить, что у нас же есть данные – примеры «входов» и правильных «выходов». Будем показывать нейросети рисунок той же цифры 4 и говорить «подстрой свои веса так, чтобы на твоём выходе при таком входе всегда загоралась четвёрка».

Сначала все веса просто расставлены случайно, мы показываем сети цифру, она выдаёт какой-то случайный ответ (весов-то нет), а мы сравниваем, насколько результат отличается от нужного нам. Затем идём по сети в обратном направлении, от выходов ко входам, и говорим каждому нейрону – так, ты вот тут зачем-то активировался, из-за тебя всё пошло не так, давай ты будешь чуть меньше реагировать на вот эту связь и чуть больше на вон ту, ок?

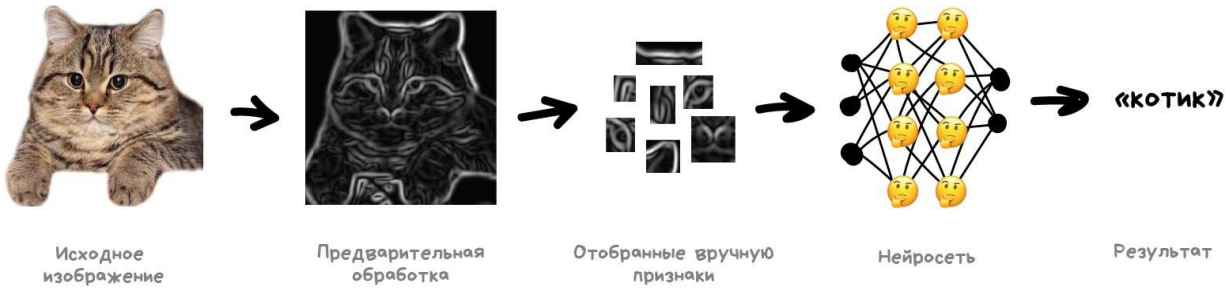
Через несколько таких циклов «прогнали-проверили-наказали» есть надежда, что веса в сети откорректируются так, как мы хотели. Научно этот подход называется Backpropagation или «Метод обратного распространения ошибки». Забавно то, что чтобы открыть этот метод понадобилось двадцать лет. До него нейросети обучали как могли.
В 2012 году свёрточная нейросеть порвала всех в конкурсе ImageNet, из-за чего в мире внезапно вспомнили о методах глубокого обучения, описанных еще в 90-х годах. Теперь-то у нас есть видеокарты!

Отличие глубокого обучения от классических нейросетей было в новых методах обучения, которые справлялись с большими размерами сетей. Однако сегодня лишь теоретики разделяют, какое обучение можно считать глубоким, а какое не очень. Мы же, как практики, используем популярные «глубокие» библиотеки типа Keras, TensorFlow и PyTorch даже когда нам надо собрать мини-сетку на пять слоёв. Просто потому что они удобнее всего того, что было раньше. Мы называем это просто нейросетями.

Свёрточные Нейросети (CNN)
Свёрточные сети сейчас на пике популярности. Они используются для поиска объектов на фото и видео, распознавания лиц, переноса стиля, генерации и дорисовки изображений, создания эффектов типа слоу-мо и улучшения качества фотографий. Сегодня CNN применяют везде, где есть картинки или видео.



Проблема с изображениями всегда была в том, что непонятно, как выделять на них признаки. Текст можно разбить по предложениям, взять свойства слов из словарей. Картинки же приходилось размечать руками, объясняя машине, где у котика на фотографии ушки, а где хвост. Такой подход даже назвали «handcrafting признаков» и раньше все так и делали.



Проблем у ручного крафтинга много.

Во-первых, если котик на фотографии прижал ушки или отвернулся – всё, нейросеть ничего не увидит.

Во-вторых, попробуйте сами сейчас назвать хотя бы десять характерных признаков, отличающих котиков от других животных. Я вот не смог. Однако когда ночью мимо меня пробегает чёрное пятно, даже краем глаза я могу сказать котик это или крыса. Потому что человек не смотрит только на форму ушей и количество лап – он оценивает объект по куче разных признаков, о которых сам даже не задумывается. А значит, не понимает и не может объяснить машине.

Получается, машине надо самой учиться искать эти признаки, составляя из каких-то базовых линий. Будем делать так: для начала разделим изображение на блоки 8x8 пикселей и выберем какая линия доминирует в каждом – горизонтальная [-], вертикальная [|] или одна из диагональных [/]. Могут и две, и три, так тоже бывает, мы не всегда точно уверены.

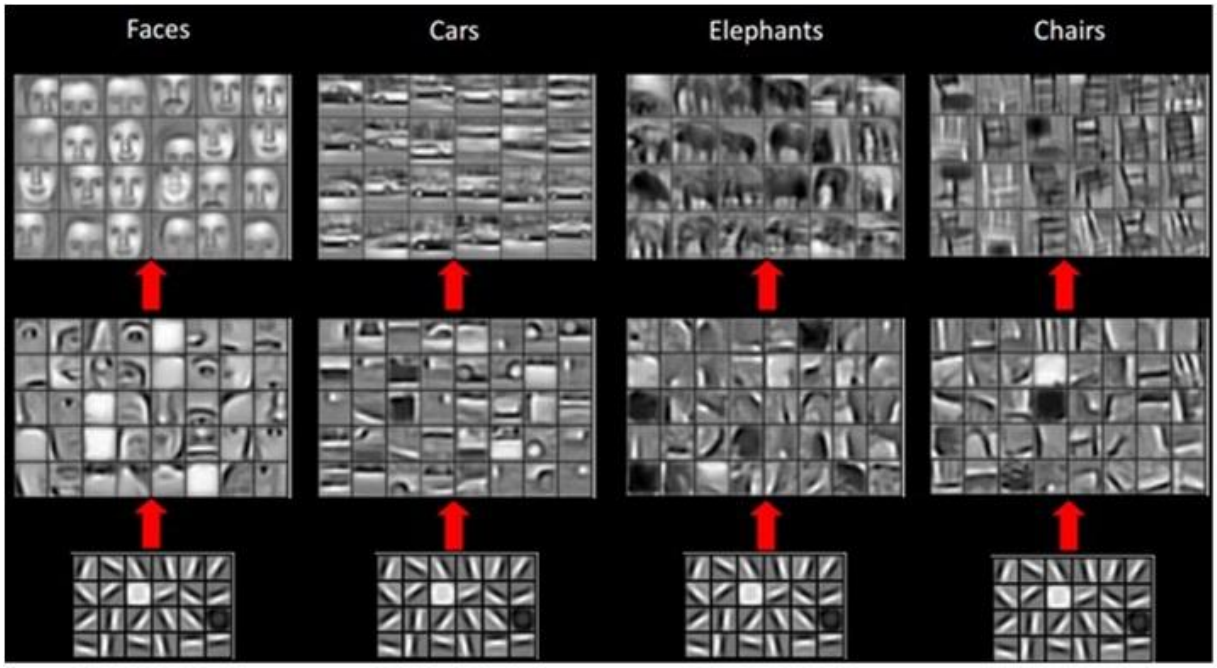
На выходе мы получим несколько массивов палочек, которые, по сути, являются простейшими признаками наличия очертаний объектов на картинке. По сути это тоже картинки, просто из палочек. Значит мы можем вновь выбрать блок 8x8 и посмотреть уже, как эти палочки сочетаются друг с другом. А потом еще и еще.

Такая операция называется свёрткой, откуда и пошло название метода. Свёртку можно представить как слой нейросети, ведь нейрон – абсолютно любая функция.

Свёрточная Нейросеть (CNN)

Когда мы прогоняем через нашу нейросеть кучу фотографий котов, она автоматически расставляет большие веса тем сочетаниям из палочек, которые увидела чаще всего. Причём неважно, это прямая линия спины или сложный геометрический объект типа мордочки – что-то обязательно будет ярко активироваться.

На выходе же мы поставим простой перцептрон, который будет смотреть какие сочетания активировались и говорить кому они больше характерны – кошке или собаке.



Красота идеи в том, что у нас получилась нейросеть, которая сама находит характерные признаки объектов. Нам больше не надо отбирать их руками. Мы можем сколько угодно кормить её изображениями любых объектов, просто

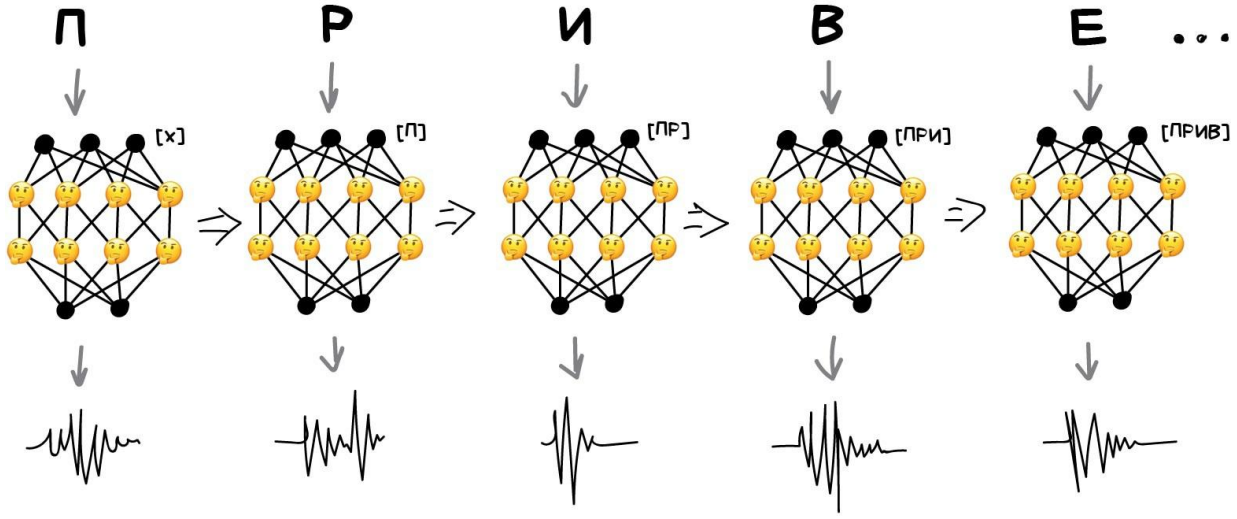
нагуглив миллион картинок с ними – сеть сама составит карты признаков из палочек и научится определять что угодно.

Рекуррентные Нейросети (RNN)
Вторая по популярности архитектура на сегодняшний день. Благодаря рекуррентным сетям у нас есть такие полезные вещи, как машинный перевод текстов и компьютерный синтез речи. На них решают все задачи, связанные с последовательностями – голосовые, текстовые или музыкальные.

Современные голосовые помощники обучают говорить не буквами, а фразами. Но сразу заставить нейросеть целиком выдавать фразы не выйдет, ведь тогда ей надо будет запомнить все фразы в языке и её размер будет исполинским. Тут на помощь приходит то, что текст, речь или музыка – это последовательности. Каждое слово или звук – как бы самостоятельная единица, но которая зависит от предыдущих. Когда эта связь теряется – получатся дабстеп.

Достаточно легко обучить сеть произносить отдельные слова или буквы. Берём кучу размеченных на слова аудиофайлов и обучаем по входному слову выдавать нам последовательность сигналов, похожих на его произношение. Сравниваем с оригиналом от диктора и пытаемся максимально приблизиться к идеалу. Для такого подойдёт даже перцептрон.

Вот только с последовательностью опять беда, ведь перцептрон не запоминает что он генерировал ранее. Для него каждый запуск как в первый раз. Появилась идея добавить к каждому нейрону память. Так были придуманы рекуррентные сети, в которых каждый нейрон запоминал все свои предыдущие ответы и при следующем запуске использовал их как дополнительный вход. То есть нейрон мог сказать самому себе в будущем – следующий звук должен звучать повыше, у нас тут гласная была (очень упрощенный пример).



Рекуррентная Нейросеть (RNN)

Была лишь одна проблема – когда каждый нейрон запоминал все прошлые результаты, в сети образовалось такое дикое количество входов, что обучить такое количество связей становилось нереально.

Когда нейросеть не умеет забывать – её нельзя обучить (у людей так же).

Сначала проблему решили в лоб – обрубили каждому нейрону память. Но потом придумали в качестве этой «памяти» использовать специальные ячейки, похожие на память компьютера или регистры процессора. Каждая ячейка позволяла записать в себя цифру, прочитать или сбросить – их назвали ячейки долгой и краткосрочной памяти (LSTM).

Когда нейрону было нужно поставить себе напоминалку на будущее – он писал это в ячейку, когда наоборот вся история становилась ненужной (предложение, например, закончилось) – ячейки сбрасывались, оставляя только «долгосрочные» связи, как в классическом перцептроне. Другими словами, сеть обучалась не только устанавливать текущие связи, но и ставить напоминалки.