

Технологии программирования

Лекция 12 | Что есть ИИ на самом деле





Основные понятия AI и ML

Что такое, сферы применения, какие ограничения при использовании

Линейные алгоритмы

Область применения, полезность и откуда надежность

Нелинейная оптимизация

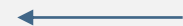

Не ML, но близко...

Гибридизация

Как получить лучшее из всех алгоритмов

Итоги

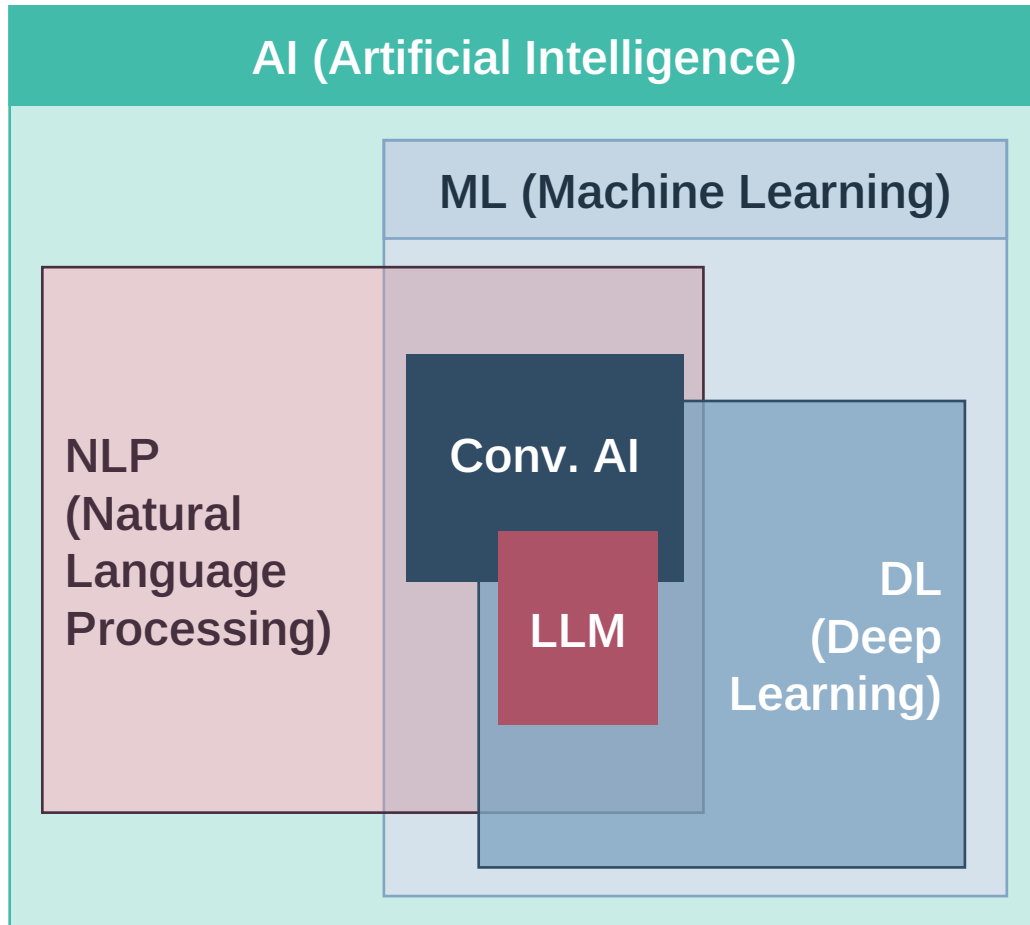
Выводы и что делать дальше?



Мы здесь



Что понимается под AI



Artificial Intelligence – это направление науки, которое занимается разработкой компьютерных систем, способных выполнять задачи, свойственные человеческому интеллекту

Сюда входит:

- Анализ данных
- Распознавание образов
- Обработка текстов и запросов, сформулированных естественным языком
- Обучение на потоках данных
- Принятие решений

История развития искусственного интеллекта

1940 – 1950

Основы AI заложены в работах Алана Тьюринга (тест Тьюринга) и Джона Маккарти

1956

Термин «искусственный интеллект»

1960 – 1970

Первые экспертные системы (MYCIN для диагностики болезней) и логические алгоритмы

1980 – 1990

Расцвет нейросетей и ML, но ограниченный вычислительными ресурсами

1997

IBM Deep Blue побеждает Гарри Каспарова в шахматах

2000е

Прорыв благодаря Big Data, GPU и глубокому обучению (AlphaGo, GPT, Dall-E)

2012

Нейросети побеждают в ImageNet, открывая эру глубокого обучения

Классы Artificial Intelligence

Weak AI (Слабый AI)

Решает узкие задачи
Не обладает сознанием
или самосознанием

Пример:

ChatGPT, который генерирует
текст, но не «понимает» его
Другие чат-боты, система
распознавания лиц

Strong AI (Сильный AI)

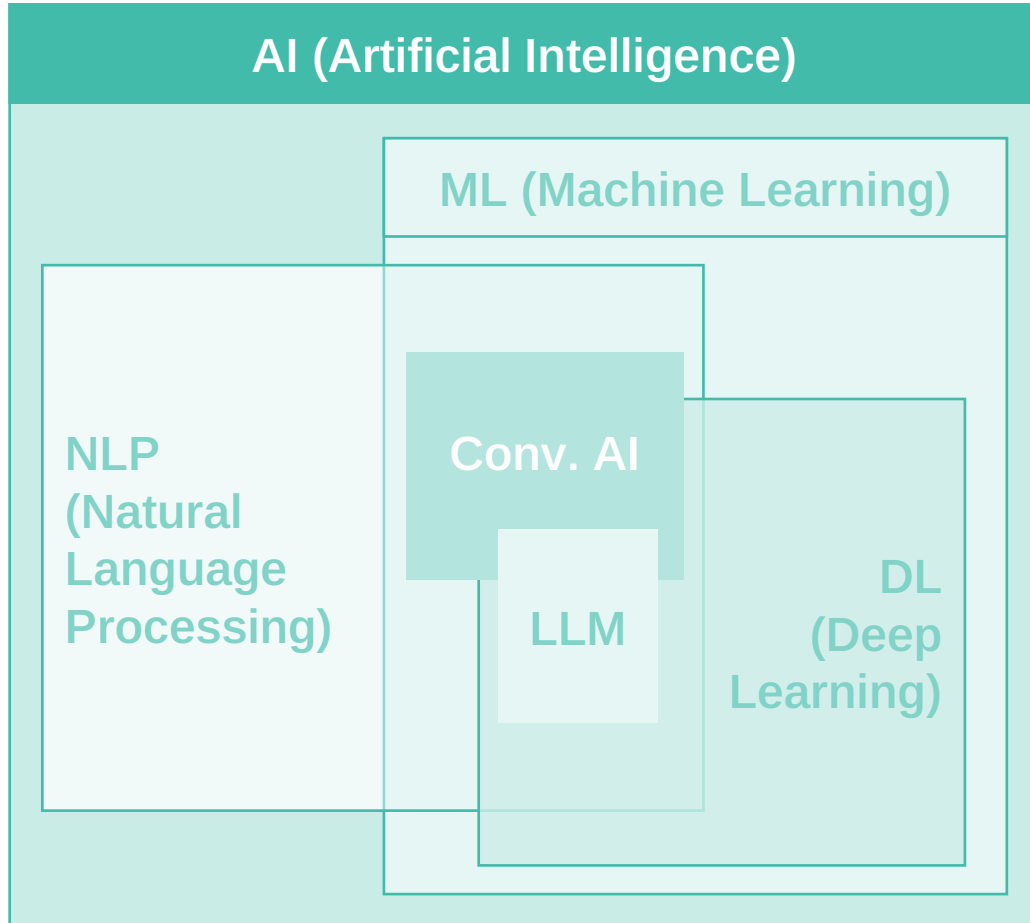
Гипотетическая система,
способная мыслить как человек
(обобщать знания, проявлять
креативность)

Artificial General Intelligence (AGI)

Уровень AI, способный решать
любые интеллектуальные задачи
на уровне человека.

Пока не существует

Artificial Intelligence в контексте лекции



AI – это любой код, исполненный на ПК, но мы будем рассматривать его как сложные модели с комплексной логикой, которые:

- Самообучаются на данных
- Адаптируются к новым условиям – генетические алгоритмы, Reinforcement Learning
- Решают неочевидные задачи через анализ скрытых паттернов (например, предсказание биржевых трендов)

ResNet

GPT-4

Random
Forest

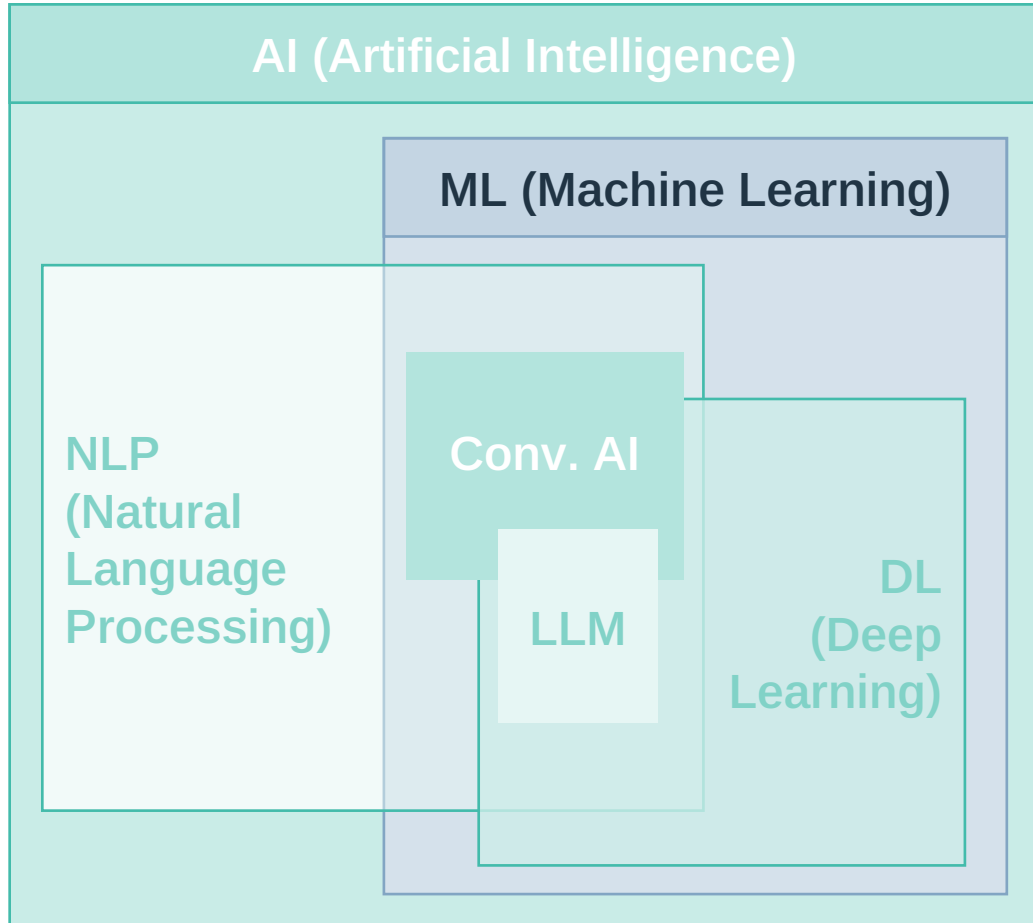
XGBoost

LSTM

Что НЕ считается AI в этой лекции:

- Простые алгоритмы без обучения
- Жестко запрограммированные правила

Machine Learning



Machine Learning (ML) – подраздел AI, в котором алгоритмы учатся на данных без явного программирования

Классическое программирование

правила вместе данными превращаются ответ

VS

ML

данные с ответами находят правила

Сферы применения ML

Сфера	Пример
Медицина	Диагностика рака по снимкам MRI
Финтех	Прогнозирование биржевых цен
Ритейл	Оптимизация запасов на складах
Логистика	Маршрутизация доставки (например, Яндекс.Такси)
Игры	AlphaGo, ИИ в компьютерных играх

Категории ML

Обучение с учителем (Supervised Learning)

Предсказать значение по историческим данным

Классификация (спам / не спам)

Регрессия (прогноз цены акций)

Обучение без учителя (Unsupervised Learning)

Найти скрытые структуры в данных

Кластеризация (сегментация клиентов)

Анализ аномалий (обнаружение мошенничества)

С обучением с подкреплением (Reinforcement Learning)

Научиться принимать решения через поощрения
и штрафы

AlphaGo, роботы, обучающиеся ходить

Глубокое обучение (Deep Learning)


Самостоятельно выявить признаки и найти решение
используя сети с множеством слоев

Распознавание речи (Siri)

Пример Классификации

Задача:
Определить, болен ли пациент диабетом
(на основе анализов)

Данные:
Возраст, уровень глюкозы, давление, ИМТ и др



Модель:
Логистическая регрессия / Random Forest

Результат:
Вероятность болезни (0 – здоров, 1 – болен)

Возраст	Глюкоза (мг/дл)	Давление (мм рт.ст.)	ИМТ	Диабет (0/1)
20	150	85	29.5	1
32	95	70	22.0	0
18	180	90	34.2	1
28	88	65	20.1	0
19	140	88	31.0	1
35	100	72	24.5	0
70	200	95	38.7	1
40	130	80	28.9	0
55	160	92	33.0	1
30	90	68	21.3	0

Пример Классификации

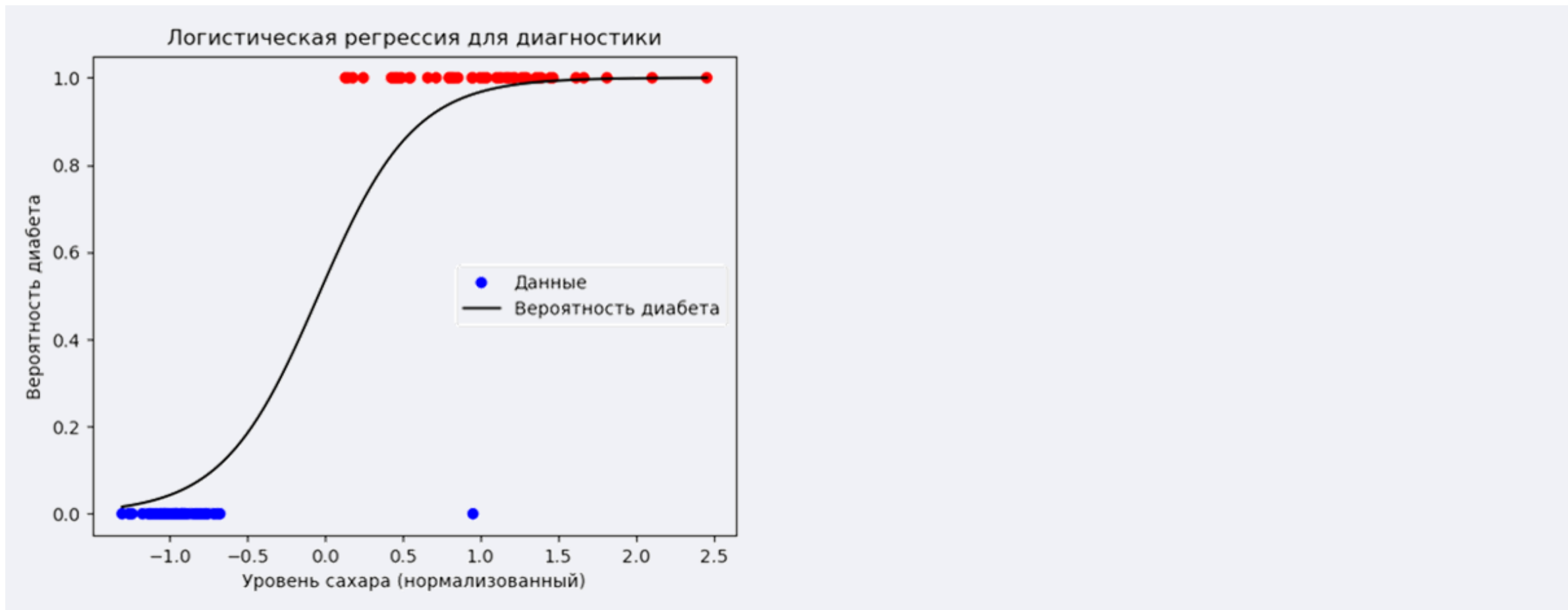
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification

# Генерация синтетических данных: уровень сахара и метка (0 = здоров, 1 = диабет)
X, y = make_classification(n_samples=100, n_features=1, n_informative=1, n_redundant=0, n_clusters_per_class=1,
random_state=42)

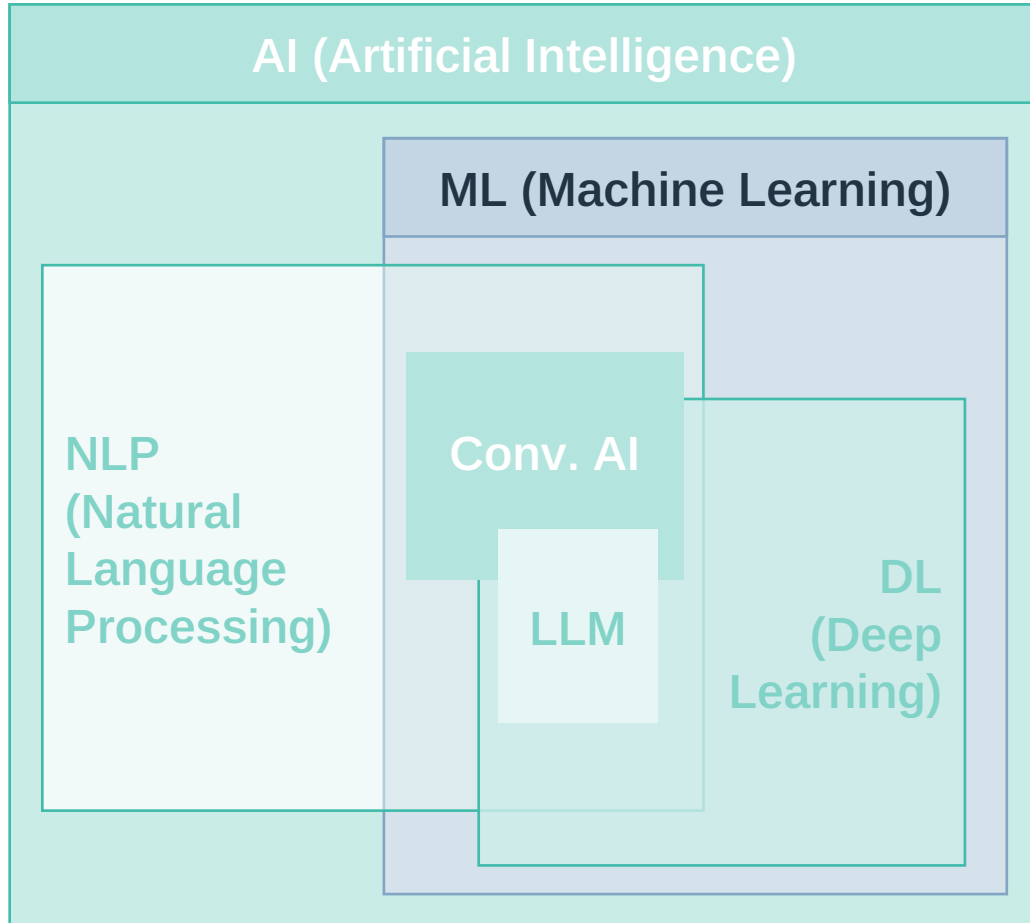
# Создание и обучение модели логистической регрессии
model = LogisticRegression()
model.fit(X, y)

# Визуализация
plt.scatter(X, y, c=y, cmap='bwr', s=30, label='Данные')
x_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
y_prob = model.predict_proba(x_range)[:, 1]
```

Пример Классификации



Компьютерное зрение (Computer Vision)



Распознавание лиц:

- Разблокировка телефона (Face ID)
- Поиск преступников в системах видеонаблюдения

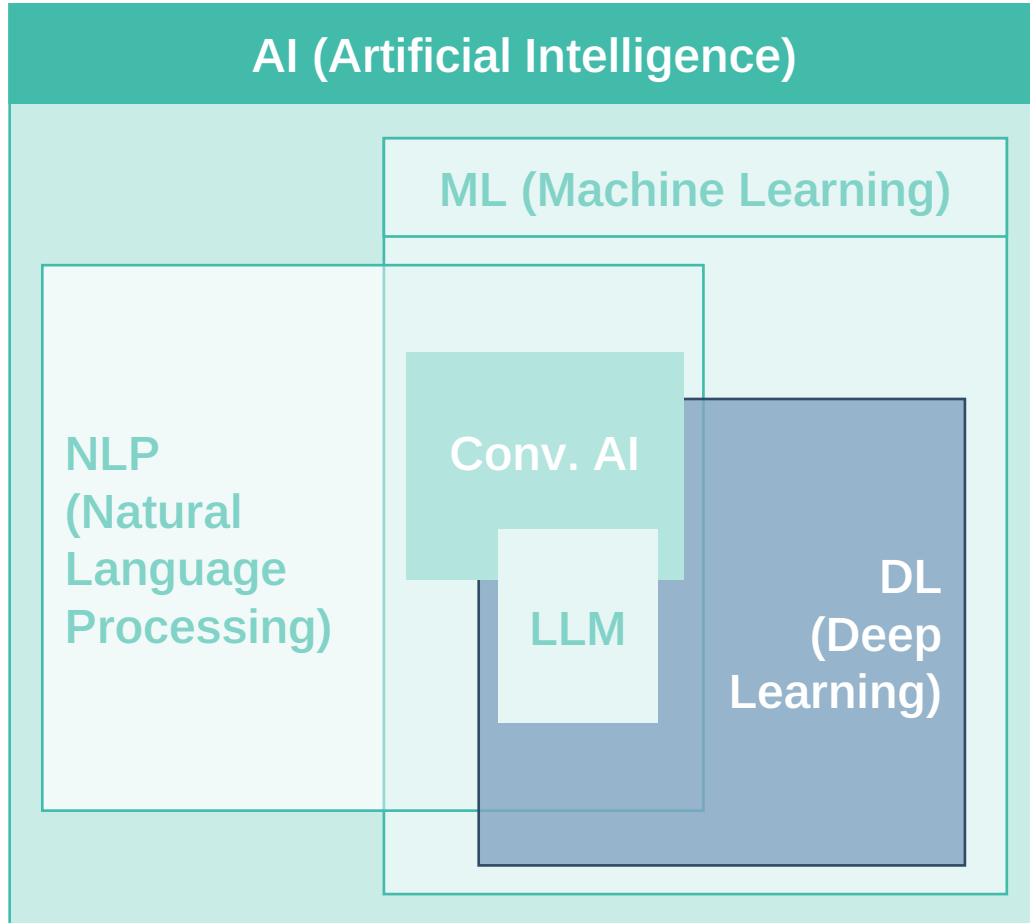
Медицинская диагностика:

- Анализ рентгеновских снимков (например, обнаружение опухолей)
- Диабетическая ретинопатия – AI от Google Health диагностирует болезнь по фото сетчатки

Автономные автомобили:

- Tesla использует компьютерное зрение для обнаружения пешеходов, знаков и других машин

Deep Learning



Deep Learning – подраздел Machine Learning, использующий многослойные нейронные сети для автоматического извлечения признаков из данных

Подкатегории:

- **Сверточные сети (CNN):**
Для изображений (например, ResNet)
- **Рекуррентные сети (RNN):**
Для временных рядов и текста (LSTM, GRU)
- **Генеративные сети (GAN):**
Создание новых данных
- **Трансформеры:**
NLP (GPT, BERT)

Deep Learning | Распознавание рукописных цифр

```

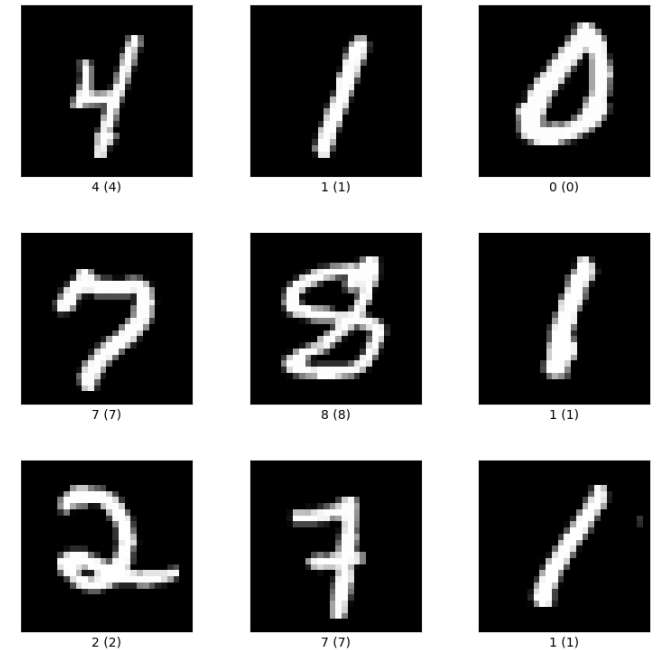
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2d # Загрузка данных

(X_train, y_train), (X_test, y_test) = mnist.load_data() # Создание модели

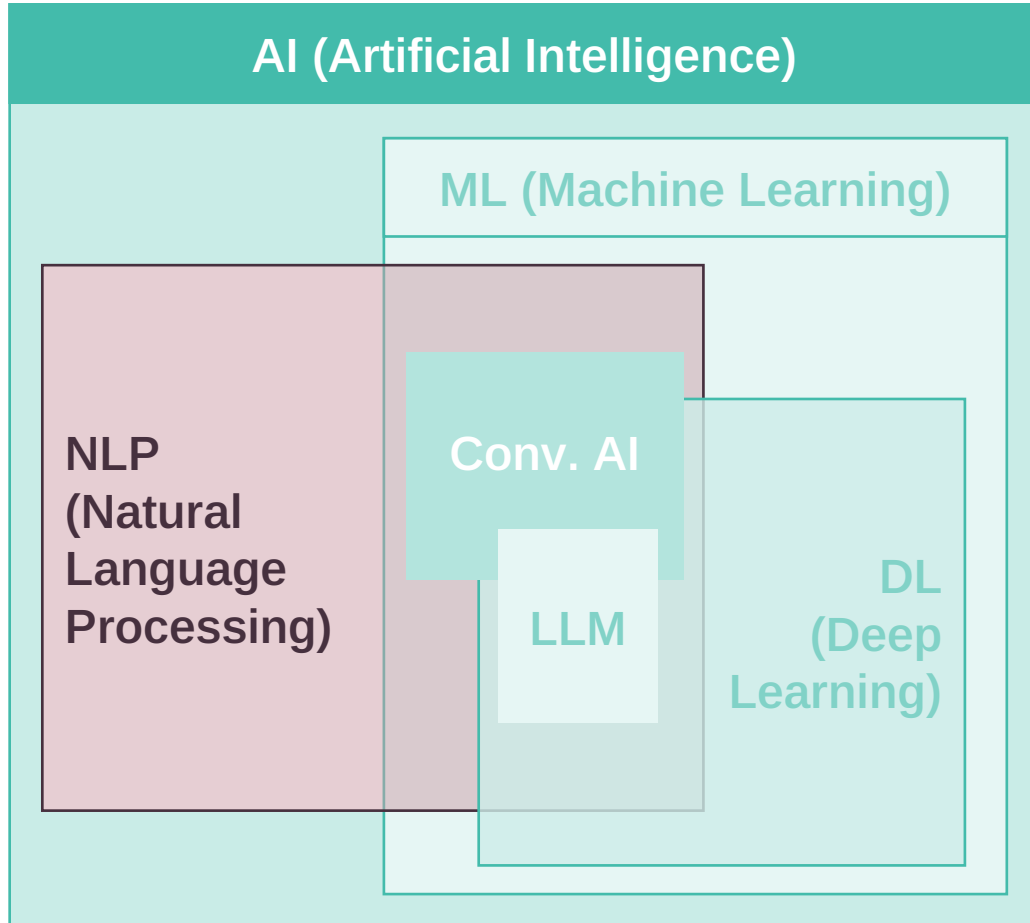
model = Sequential([
    Flatten(input_shape=(28, 28)), # Преобразует изображение 28x28 в вектор
    Conv2D(32, kernel_size=(3, 3), activation='relu',
        input_shape=(28, 28, 1)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax') # 10 классов (цифры 0-9)
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5)

```



NLP



NLP – область AI, которая помогает компьютерам понимать, анализировать и генерировать человеческий язык

Сферы применения:

- **Машинный перевод:**
Google Translate, DeepL
- **Анализ тональности:**
Оценка отзывов на Amazon
- **Чат-боты:**
Поддержка клиентов
- **Извлечение информации:**
Поиск ключевых фраз в документах

Подкатегории и виды NLP

Подкатегории

Текстовое
представление
Word2Vec, BERT

Классификация текста
Спам vs не спам

Генерация текста
GPT-3, ChatGPT

NER (Named Entity
Recognition)
Извлечение имен, дат,
локаций

Виды

Понимание
естественного языка
(NLU)

Генерация
естественного языка
(NLG)

Обработка
естественного языка
(NLP)
Охватывает как NLU,
так и NLG

Взаимодействие на
естественном языке
(NLI)

Классификация: Определение тональности твитов

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC

# Данные: твиты и их тональность (0 — негатив, 1 — позитив)
tweets = ["I love this movie!", "This product is terrible..."]
labels = [1, 0]

# Векторизация текста
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(tweets)

# Обучение модели
model = SVC()
model.fit(X, labels)

# Предсказание
test_tweet = ["This is amazing!"]
test_X = vectorizer.transform(test_tweet)

print(model.predict(test_X)) # Output: [1]
```

Извлечение именованных сущностей (NER)

```
import spacy

nlp = spacy.load("en_core_web_sm")
text = "Apple is looking to buy a U.K. startup for $1 billion."

doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)

# Output:
# Apple ORG
# U.K. GPE
# $1 billion MONEY
```

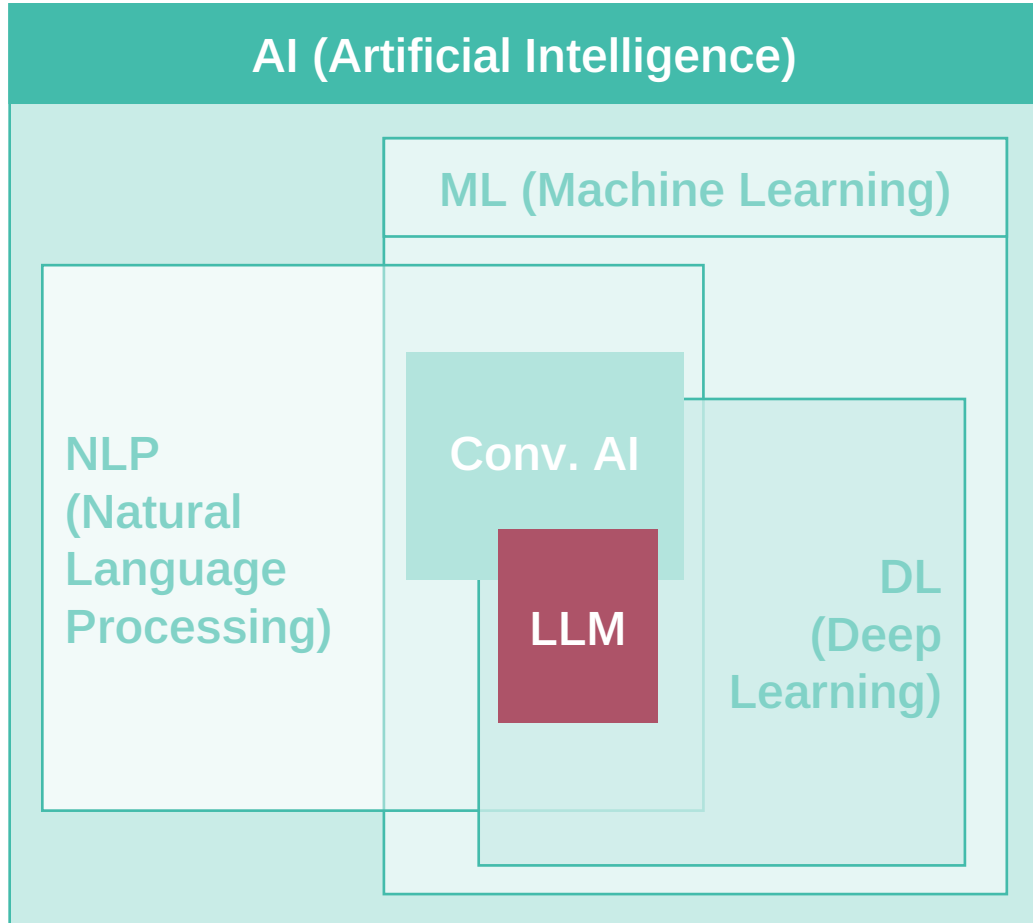
Кластеризация новостных статей по темам

```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer

# Векторизация текстов
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(news_articles)

# Кластеризация
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
```

Large Language Models (LLM)



LLM – это нейросетевые модели, обученные на огромных объемах текстовых данных для выполнения задач, связанных с пониманием и генерацией естественного языка

1

Основа

Архитектура трансформеров (2017), механизм внимания (attention)

2

Примеры моделей

GPT-4, LLaMA, PaLM, BERT

3

Ключевая особенность

Способность работать с контекстом длиной в тысячи слов и выполнять задачи без явного переобучения

Ограничения LLM

Вычислительная стоимость

Обучение GPT-3 требует миллионов долларов и тысяч GPU

Галлюцинации

Модель генерирует ложные факты (например, выдуманные цитаты)

Предвзятость данных

Усиление стереотипов из-за токсичных данных в обучающих корпусах

Слабая интерпретируемость

Невозможно точно понять, как модель принимает решения

Чем LLM отличаются от предыдущих моделей?

Критерий	Старые модели (RNN, LSTM)	LLM на трансформерах
Обработка контекста	Ограниченная длина (проблема забывания)	Контекст до 100 тыс. токенов (например, GPT-4 Turbo)
Обучение	Требуется тонкая настройка под каждую задачу	Few-shot learning (решение задач через примеры в промпте)
Архитектура	Последовательная обработка	Параллельная через self-attention
Масштаб	Миллионы параметров	Миллиарды параметров

Дальнейшее развитие LLM и трансформеров

Оптимизация размеров

Модели-«лайт» версии (например, [DistilBERT](#), [TinyLlama](#)) для работы на смартфонах

Повышение точности:

Борьба с галлюцинациями через [RAG](#) (Retrieval-Augmented Generation)

Мультимодальность:

Модели, работающие с текстом, изображениями и звуком (например, [GPT-4V](#), [Gemini](#))

Энергоэффективность:

Алгоритмы квантования и уменьшения требуемых ресурсов

Сокращение научных статей до абзаца

```
from transformers import pipeline

summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
text = "Длинный текст о климатических изменениях..."

summary = summarizer(text, max_length=130)

print(summary[0]['summary_text'])
```

Зачем нам ML?

Автоматизация
сложных задач

Масштабируемость

Гибкость

Раньше

Писали жесткие правила для фильтрации спама (if "виагра" in subject: spam = True)



Сейчас

ML сам учится находить сложные паттерны в письмах на основе их метаданных

Одна обученная модель может обрабатывать миллионы запросов
Например, рекомендации Netflix)

Один и тот же алгоритм (например, Random Forest) решает задачи от предсказания цен до диагностики болезней

Устойчивость результата

Что влияет на результат?

Качество данных

Если в тренировочных данных есть смещение (bias), модель повторит его

Пример: Модель для найма, обученная на данных с гендерным перекосом, будет дискриминировать женщин

Изменения в данных

Модель, обученная на данных 2020 года, может не работать в 2024 (например, тренды в соцсетях)

ML-модель – не «раз и навсегда», её нужно постоянно мониторить и обновлять

Устойчивость результата

! Подготовка данных решает больше, чем архитектура сети

Грязные данные гарантируют Плохие предсказания – даже GPT-4 выдаст ерунду, если входной текст бессмысленный

Задача:
Предсказать
стоимость дома

Плохие данные

В датасете нет информации
о районе или площади



Лучшая нейросеть даст точность **60%**

Хорошие данные

Добавили этажность, год постройки,
транспортную доступность



Линейная регрессия даст **85%** точности

«Лучше простая модель на чистых данных, чем сложная модель на мусоре»

Почему ML ненадёжный?

Галлюцинации

Модель уверенно генерирует ложную информацию

Хрупкость

Незначительные изменения ввода ломают предсказания

Пример: Изменение одного пикселя в изображении → модель путает панду с гиббоном (атаки adversarial attacks)

Чёрный ящик

Невозможно точно понять, как модель приняла решение
(проблема в медицине, юриспруденции)

Выход – грамотное использование детерминированных алгоритмов точно применяя ML

Только как этого добиться?



Основные понятия AI и ML

Что такое, сферы применения, какие ограничения при использовании

Линейные алгоритмы

Область применения, полезность и откуда надежность

Нелинейная оптимизация

Не ML, но близко...

Гибридизация

Как получить лучшее из всех алгоритмов

Итоги

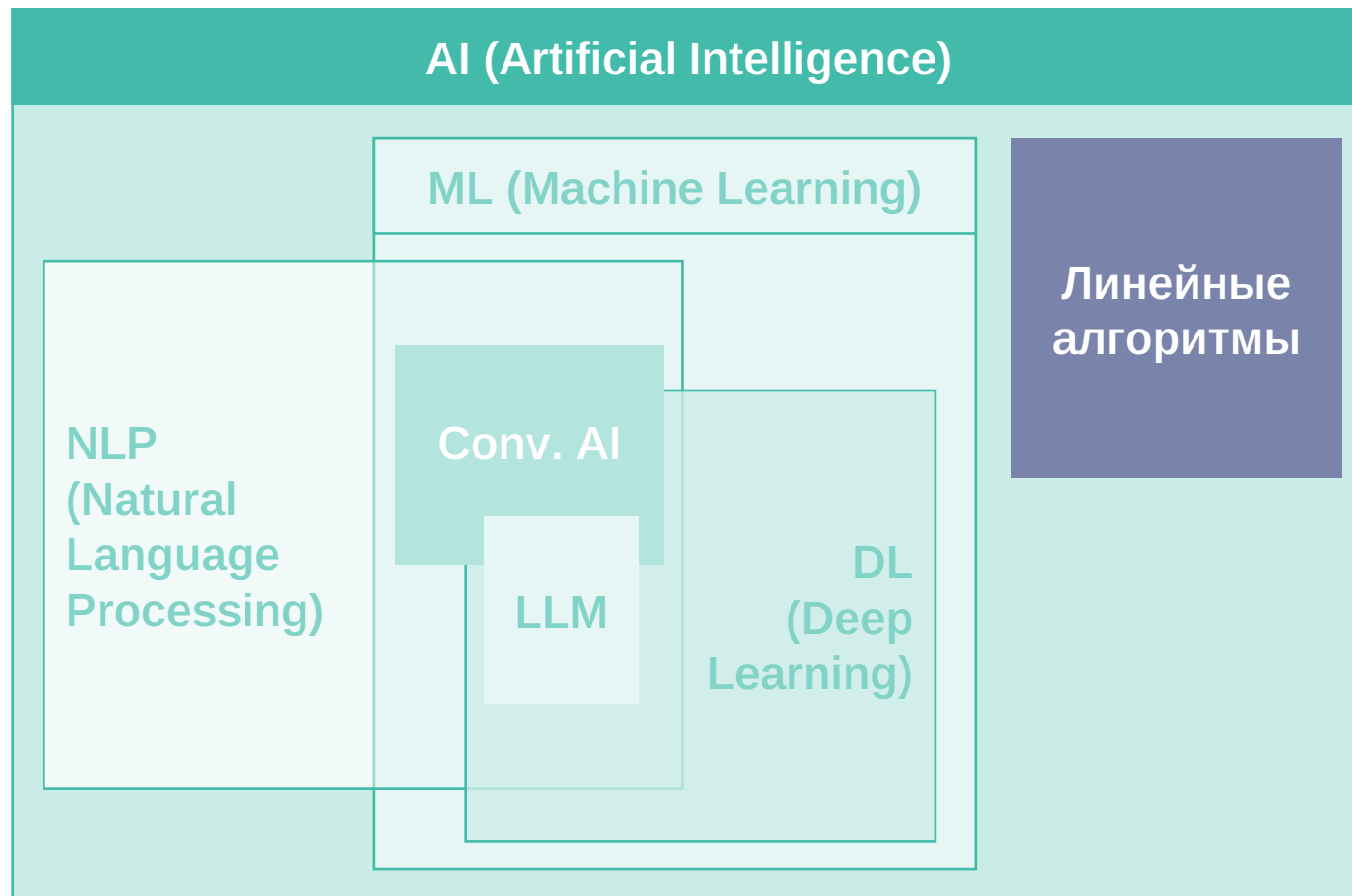
Выводы и что делать дальше?



← Мы здесь



Линейные алгоритмы



Линейные алгоритмы – методы, которые моделируют зависимости между переменными, предполагая, что эти зависимости можно описать линейными функциями (прямыми линиями или гиперплоскостями в многомерных пространствах)

Линейные алгоритмы

Линейные алгоритмы – это подмножество AI, где делается упрощающее предположение о линейной зависимости данных

Они основаны на предположении, что выходные данные можно выразить как линейную комбинацию входных данных с некоторыми коэффициентами

Если мы хотим предсказать цену квартиры (зависимая переменная) на основе её площади (независимая переменная), линейный алгоритм предположит, что цена растёт пропорционально площади:

- $\text{Цена} = a * \text{площадь} + b$, где a и b — параметры модели.

В ML линейные алгоритмы часто используются как начальная точка или базовые модели (baseline), поскольку они просты, быстры и требуют меньше вычислительных ресурсов, чем сложные нелинейные модели (например, глубокие нейронные сети)

Для лекции мы примем следующие ключевые утверждения о линейных алгоритмах:

Линейность: Алгоритм предполагает, что данные можно описать линейной функцией вида $y = w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n + b$, где w_i — веса, x_i — входные переменные, b — свободный член

Простота и интерпретируемость: Линейные алгоритмы легко интерпретировать, так как их параметры (веса) напрямую показывают вклад каждой переменной

Ограничения: Они работают хорошо, если данные действительно имеют линейную зависимость. Если зависимости сложные (нелинейные), линейные алгоритмы могут быть недостаточно точными

Основа для гибридных подходов: Линейные алгоритмы часто используются как базовые компоненты в более сложных моделях (например, в ансамблях или нейронных сетях)

Мир как математическая модель

Математическое моделирование – метод исследования реальных объектов и явлений через создание и изучение их математических моделей

Это процесс создания и исследования математических моделей, позволяющий изучать объект в виртуальной среде

Математическая модель – упрощенное математическое представление реального объекта, явления или процесса

Примеры применения:

Физика

Моделирование движения тел, процессов в термодинамике и т.д.

Экономика

Моделирование рынков, инвестиций, прогнозирование

Инженерия

Моделирование процессов в машиностроении, электротехнике, строительстве

Алгоритмы наше всё

! Поэтому большинство алгоритмов полностью опираются на математические модели

Черты алгоритмов:

Математическое описание

Линейные зависимости выражаются через уравнения (например, $y=ax+b$) – это позволяет строго формализовать задачу

Оптимизация

Обучение линейных моделей сводится к минимизации функции потерь*, с использованием методов линейной алгебры и оптимизации (например, через градиентный спуск)

Интерпретируемость

Математическая строгость позволяет точно объяснить, как модель принимает решения
Например, какой вклад вносит каждая переменная

Универсальность

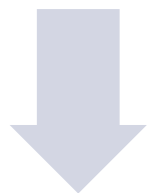
Математические основы (линейная алгебра, теория вероятностей) универсальны и применимы как к линейным, так и к нелинейным моделям, что делает их фундаментом ML

* Например, среднеквадратичная ошибка

Детерминированность

Линейные алгоритмы часто **детерминированы**, то есть при одинаковых входных данных и параметрах они всегда дают одинаковый результат

Например, в линейной регрессии для фиксированного набора данных и метода оптимизации (например, обычного метода наименьших квадратов) вы получите одну и ту же модель



Это отличает их от некоторых нелинейных алгоритмов, которые могут давать разные результаты при каждом запуске (например, нейронных сетей с случайной инициализацией начальных весов)

Детерминированность | Почему это залог успеха?

Предсказуемость

Точная воспроизводимость при одних и тех же данных

Интерпретируемость

Легко объяснить и представить в голове

Например, вы можете сказать:
«Этот коэффициент показывает, что увеличение площади дома на 1 м² увеличивает цену на 1000 рублей»

Доверие

Детерминированные модели вызывают больше доверия у пользователей, так как их поведение понятно и предсказуемо

Больше примеров

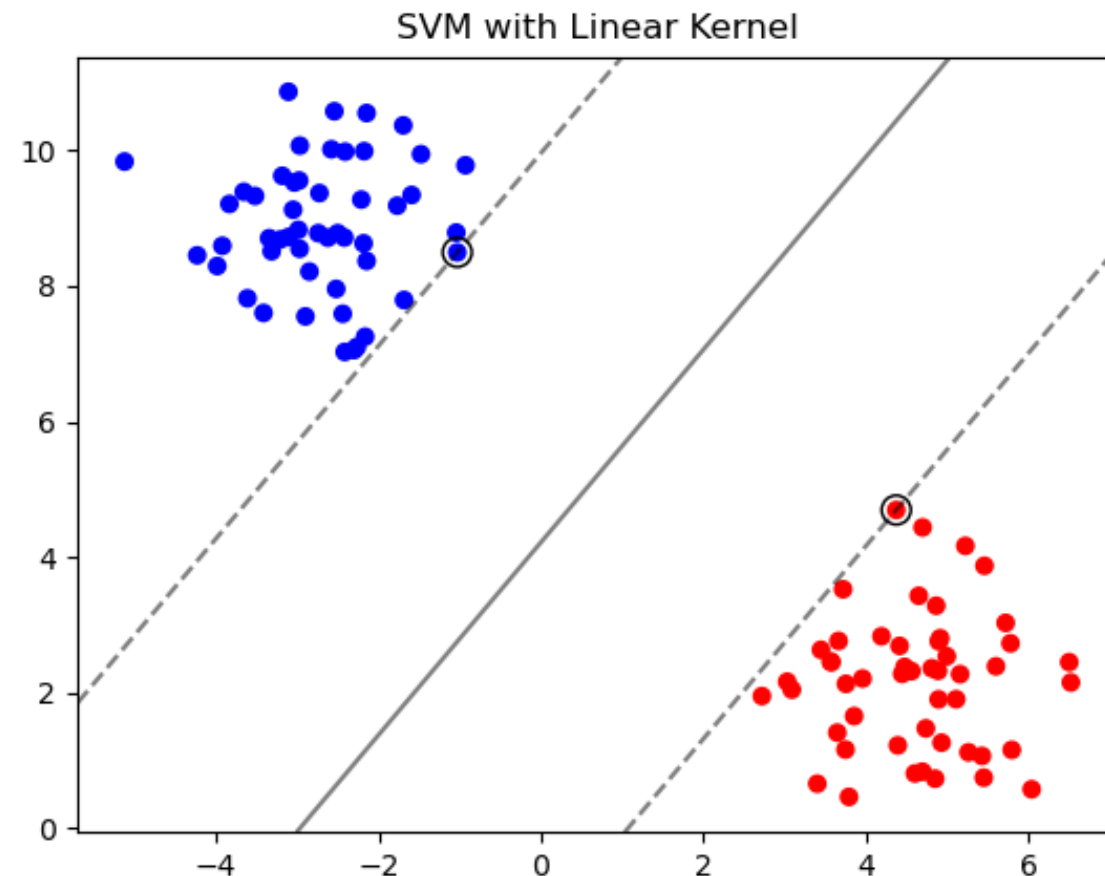
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_blobs

# Генерация синтетических данных
X, y = make_blobs(n_samples=100, centers=2, random_state=42)

# Создание и обучение модели SVM с линейным ядром
model = SVC(kernel='linear', C=1.0)
model.fit(X, y)

# Создание сетки для отображения границы решения
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

Z = model.decision_function(xy).reshape(XX.shape)
```



Детекция лиц в видеопотоке с веб-камеры

**Какой метод вы бы
выбрали для решения
задачи?**

Без учета навыков и умений, какой подход
из известного вам кажется лучшим выбором?

Варианты:

- ML (CNN)
- Template Matching (матем-кие операции)
- Feature Matching
- HOG + SVM

• **Каскады Хаара**



Каскады Хаара как альтернатива ML

Каскады Хаара – алгоритм обнаружения объектов на изображениях, основанный на **признаках Хаара** (простые паттерны яркости) и **бустинге** (AdaBoost)

Основное применение:

Быстрое обнаружение стандартных объектов (лица, глаза, машины) в реальном времени

Преимущества

Высокая скорость работы

Подходит для embedded-систем и мобильных приложений

Требует мало вычислительных ресурсов

Простая реализация

Недостатки

Низкая точность при сложных ракурсах, освещении или нестандартных объектах

Требует предобученных XML-каскадов
Не гибкие для кастомных задач

Примеры задач для каскадов Хаара

- Детекция лиц в видеопотоке с веб-камеры (*данная задача*)
- Обнаружение автомобильных номеров на парковке
- Отслеживание глаз в приложениях дополненной реальности



Детекция лиц в видеопотоке с веб-камеры

Классический ML

```
# Загрузка пред обученной модели ResNet из Caffe
model_weights = "res10_300x300_ssd_iter_140000.caffemodel"
config = "deploy.prototxt"
net = cv2.dnn.readNetFromCaffe(config, model_weights)

# Загрузка изображения
img = cv2.imread('test_face.jpg')
h, w = img.shape[:2]
blob = cv2.dnn.blobFromImage(img, 1.0, (300, 300), [104, 117, 123])

# Детекция лиц
net.setInput(blob)
detections = net.forward()

# Отрисовка результатов
for i in range(detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.7:
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        x1, y1, x2, y2 = box.astype(int)
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

cv2.imshow('CNN Faces', img)
cv2.waitKey(0)
```

Каскады Хаара

```
import cv2

# Загрузка предобученного каскада для лиц
face_cascade = cv2.CascadeClassifier(cv2.data.harcascades +
                                     'haarcascade_frontalface_default.xml')

# Загрузка изображения
img = cv2.imread('test_face.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Детекция лиц
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

# Отрисовка прямоугольников вокруг лиц
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2.imshow('Haar Cascade Faces', img)
cv2.waitKey(0)
```

Детекция лиц в видеопотоке с веб-камеры

Критерий	CNN (ResNet)	Каскады Хаара
Время обработки (на CPU)	~500 мс на изображение 640x480	~10 мс на изображение 640x480
Точность	95-98%	70-80%
Гибкость	Любые объекты	Только лица

Каскады Хаара как альтернатива ML

Критерий	Классический ML (CNN, YOLO)	Каскады Хаара
Скорость	Зависит от модели (ResNet медленнее YOLO)	Очень высокая
Точность	Высокая (адаптируется под любые данные)	Средняя (только для стандартных объектов)
Гибкость	Высокая (можно обучать на своих данных)	Низкая (только предобученные классы)
Ресурсы	Требует GPU для сложных моделей	Минимум (работает на CPU)


Метод опорных векторов | Классификация препятствий

Задача:

Определить, является ли объект препятствием или свободным путём, на основе данных лидара

Описание:

Код использует SVM с линейным ядром для классификации и визуализирует разделяющую гиперплоскость



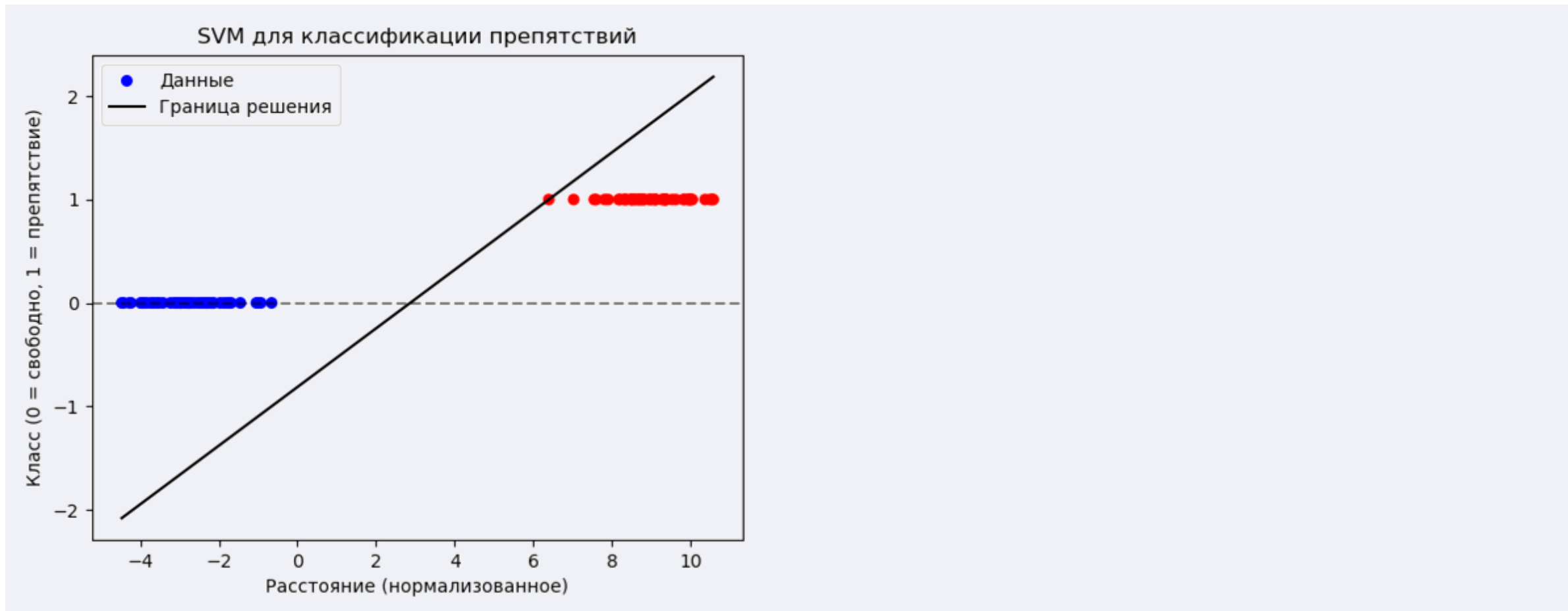
```
from sklearn.svm import SVC
from sklearn.datasets import make_blobs

# Генерация синтетических данных: расстояние и
# метка (0 = свободно, 1 = препятствие)
X, y = make_blobs(n_samples=100, centers=2, n_features=1,
                  random_state=42)

# Создание и обучение модели SVM с линейным ядром
model = SVC(kernel='linear', C=1.0)
model.fit(X, y)

# Визуализация
plt.scatter(X, y, c=y, cmap='bwr', s=30, label='Данные')
x_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
Z = model.decision_function(x_range)
```

Метод опорных векторов | Классификация препятствий



Локализация (объединение данных)

Задача:

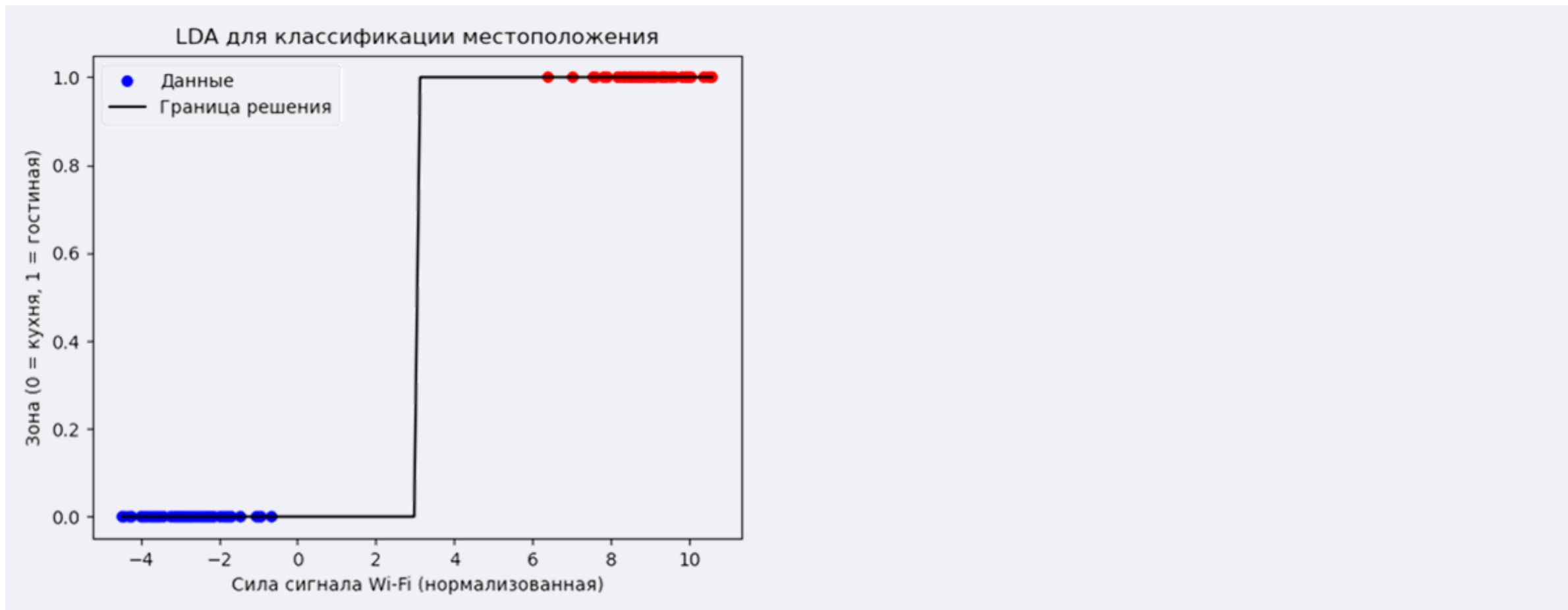
Определить, в какой зоне (например, «кухня» или «гостиная») находится робот, на основе силы сигнала Wi-Fi


Описание:

Код классифицирует зоны с помощью LDA и визуализирует разделение

```
from sklearn.discriminant_analysis import  
LinearDiscriminantAnalysis  
from sklearn.datasets import make_blobs  
  
# Генерация синтетических данных: сила сигнала Wi-Fi  
и метка (0 = кухня, 1 = гостиная)  
X, y = make_blobs(n_samples=100, centers=2, n_features=1,  
random_state=42)  
  
# Создание и обучение модели LDA  
model = LinearDiscriminantAnalysis()  
model.fit(X, y)  
  
# Визуализация  
plt.scatter(X, y, c=y, cmap='bwr', s=30, label='Данные')  
x_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)  
y_pred = model.predict(x_range)
```

Локализация (объединение данных)





Основные понятия AI и ML

Что такое, сферы применения, какие ограничения при использовании

Линейные алгоритмы

Область применения, полезность и откуда надежность

Нелинейная оптимизация


Не ML, но близко...

Гибридизация

Как получить лучшее из всех алгоритмов

Итоги

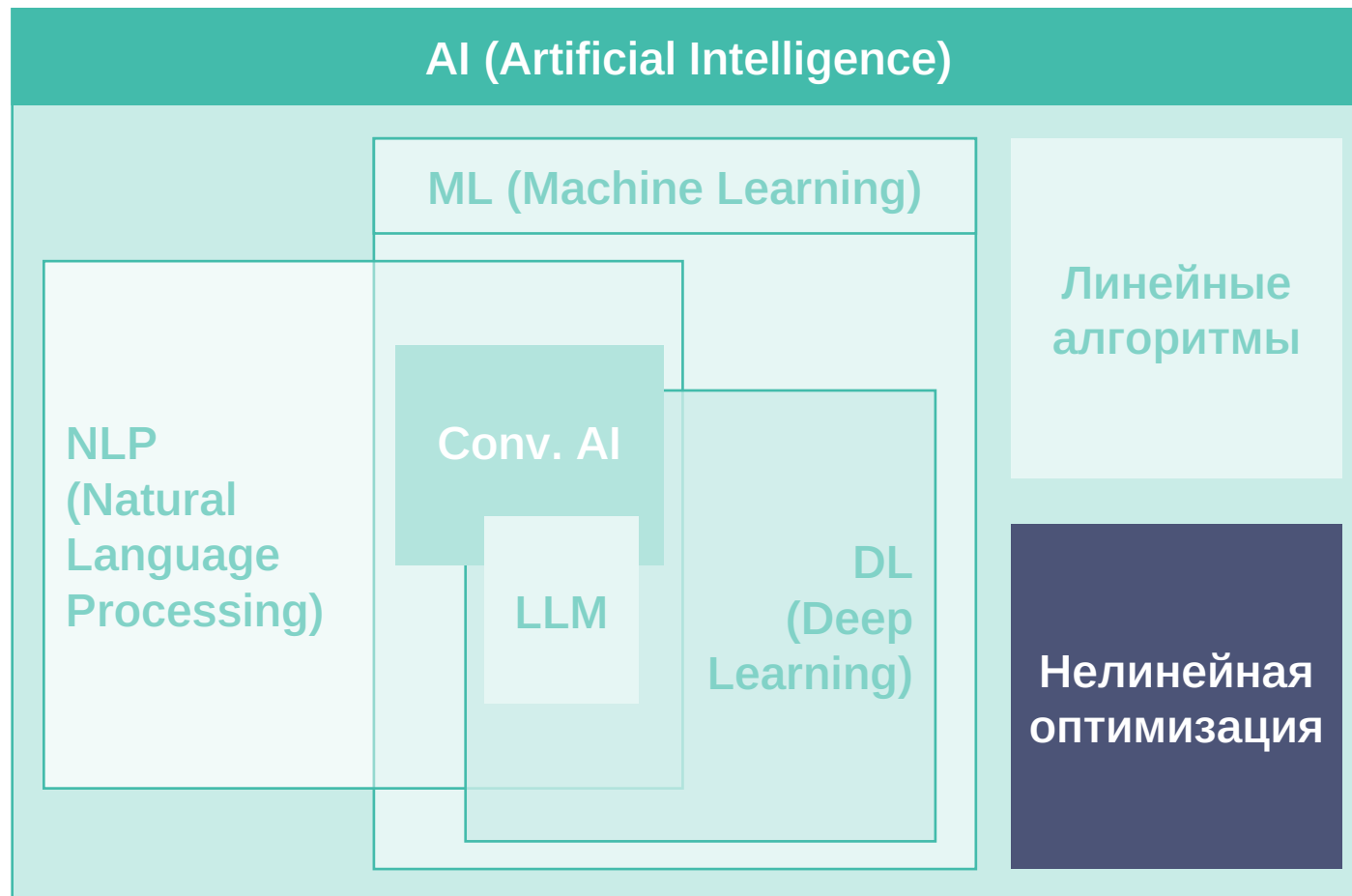
Выводы и что делать дальше?



← Мы здесь



А что дальше?



Нелинейная оптимизация – это область математики, которая ищет наилучшее решение для задач, где целевая функция $f(x)$ или ограничения $g(x) \leq 0, h(x) = 0$ нелинейны

Пример задачи

$$\min_x f(x), \quad \text{при } g(x) \leq 0, \quad h(x) = 0$$

Нелинейная оптимизация

Основные особенности

Нелинейные функции могут иметь несколько локальных минимумов, что усложняет поиск глобального оптимума

Решение требует численных методов, так как аналитические решения редки

Итог зависит от начальной точки и параметров алгоритма

Отличия от ML

Сложность

Линейные задачи решаются быстрее $O(n)$ $\log(n)$, нелинейные – сложнее (иногда $n!$)

Оптимальность

В линейной локальный оптимум = глобальный, в нелинейной – нет

Методы

Линейная – симплекс-метод, нелинейная – градиентные, квазиньютоновские

Чувствительность

Нелинейные задачи зависят от начальных условий

Оптимизация классифицируется в зависимости от структуры задачи

Безусловная оптимизация

Поиск минимума или максимума функции без ограничений (например, $\min_x f(x)$)

Условная оптимизация

Оптимизация с ограничениями, которые могут быть равенствами ($h(x)=0$) или неравенствами ($g(x)\leq 0$)

Глобальная оптимизация

Нахождение глобального оптимума в присутствии множества локальных минимумов

Стохастическая оптимизация

Работа с функциями, содержащими шум или неопределённость

Основные алгоритмы нелинейной оптимизации

Градиентные методы

- **Градиентный спуск:** Итеративно движется в направлении антиградиента для минимизации функции
- **Метод сопряжённых градиентов:** Ускоряет сходимость для задач, близких к квадратичным

Методы второго порядка

- **Метод Ньютона:**
Использует матрицу вторых производных для быстрой сходимости, но требует больших вычислительных ресурсов

Эволюционные и стохастические методы

- **Генетические алгоритмы:** Имитируют естественный отбор для поиска глобального оптимума
- **Метод имитации отжига:** Использует вероятностный подход, чтобы избежать локальных минимумов

Методы без градиента

- **Метод Нелдера-Мида (симплекс-метод):**
Работает с геометрическими преобразованиями

Эти алгоритмы подходят для разных типов задач: градиентные – для гладких функций, стохастические – для поиска глобального оптимума, а методы без градиента – для случаев, когда производные вычислить сложно

Матан, матан и ещё раз матан

! Нелинейная оптимизация математически сложнее линейной по нескольким причинам:

Нелинейность:

Линейные задачи решаются аналитически (например, через симплекс-метод), а нелинейные требуют итеративных методов, таких как градиентный спуск

Производные:

Алгоритмы часто используют первые и вторые производные (гессиан), что увеличивает объём вычислений

Условия оптимальности:

Применяются сложные методы, такие как условия Каруша-Куна-Таккера (ККТ), которые обобщают лагранжиан для задач с неравенствами

Локальные vs глобальные минимумы:

В линейной оптимизации локальный оптимум всегда глобальный, а в нелинейной нужно проверять, не застрял ли алгоритм

Пример комплексной функции для оптимизации с жесткими ограничениями:

$$f(x_1, x_2) = c_1 x_1^2 + c_2 x_2^2 + c_3 x_1 x_2 + c_4 \ln(1 + x_1) + c_5 e^{x_2}$$

$$\begin{aligned} a_1 x_1 + a_2 x_2 &\leq b_1 \\ x_1^2 + x_2^2 + x_1 x_2 &\leq b_2 \\ x_1 &\geq 0, x_2 \geq 0 \end{aligned}$$

Применимость

- Инженерия – проектирование мостов, минимизация веса конструкций
- Экономика – оптимизация инвестиционных портфелей
- Логистика – планирование маршрутов с нелинейными затратами
- Энергетика – управление сетями с нелинейными потерями
- Химия – оптимизация реакций с нелинейной кинетикой



А что по ресурсам?

! Нелинейная оптимизация требует больше усилий

Вычисления

Итеративные методы и вычисление производных увеличивают нагрузку

Настройка

Нужно подбирать шаг, параметры, начальные точки

Анализ

Проверка сходимости к глобальному оптимуму требует дополнительных тестов

Реализация

Код сложнее, часто нужны библиотеки (например, SciPy)

Линейные алгоритмы проще и быстрее, но менее гибки

Как пример сложности – навигация через алгоритм Дейкстры и минимизации функции, описывающую модель передвижения робота

Детерминированность

! Нелинейная оптимизация даёт одинаковый результат при тех же входных данных, что важно для воспроизводимости



Гарантии

Многие алгоритмы имеют теоремы сходимости к оптимуму



Точность

В задачах с чёткой постановкой (например, проектирование) она превосходит аппроксимации ML

Пример | Минимизация затрат производства

Предположим, что компания оптимизирует производство двух видов продукции x_1, x_2

Затраты на производство зависят от объёмов продукции нелинейным образом, а ограничения связаны с доступными ресурсами и технологическими особенностями

Цель – минимизировать общие затраты, учитывая эти условия

$$f(x_1, x_2) = c_1 x_1^2 + c_2 x_2^2 + c_3 x_1 x_2 + c_4 \ln(1 + x_1) + c_5 e^{x_2}$$

Ограничения алгоритма:

$$\begin{aligned} a_1 x_1 + a_2 x_2 &\leq b_1 \\ x_1^2 + x_2^2 + x_1 x_2 &\leq b_2 \\ x_1 &\geq 0, x_2 \geq 0 \end{aligned}$$

Пример | Минимизация затрат производства

Коэффициенты (примерные значения)

`c1, c2, c3, c4, c5 = 1.0, 1.0, 0.5, 1.0, 0.1`

`a1, a2, b1 = 1.0, 1.0, 5.0`

`b2 = 10.0`

`b3 = 5.0`

Целевая функция

`def objective(x):`

`x1, x2 = x`

`return c1 * x1**2 + c2 * x2**2 + c3 * x1 * x2 + \`
`c4 * np.log(1 + x1) + c5 * np.exp(x2)`

Ограничения

`constraints = [`

`# $a_1x_1 + a_2x_2 \leq b_1$`

`{'type': 'ineq', 'fun': lambda x: b1 - a1 * x[0] - a2 * x[1]},`

`# $x_1^2 + x_2^2 + x_1x_2 \leq b_2$`

`{'type': 'ineq', 'fun': lambda x: b2 - x[0]**2 - x[1]**2 - x[0]*x[1]},`

`# $e^{x_1} + x_2^2 \leq b_3$`

`{'type': 'ineq', 'fun': lambda x: b3 - np.exp(x[0]) - x[1]**2}`

`]`

Границы переменных ($x_1 \geq 0, x_2 \geq 0$)

`bounds = [(0, None), (0, None)]`

Начальное приближение

`x0 = [0.5, 0.5]`

Вызов функции minimize с методом SLSQP


`result = minimize(objective, x0,`
`method='SLSQP', bounds=bounds, constraints=constraints)`

Оптимальное решение:

`[4.44089210e-16 2.22044605e-16]`

Минимальное значение функции:

`0.1000000000000000048`



Основные понятия AI и ML

Что такое, сферы применения, какие ограничения при использовании

Линейные алгоритмы

Область применения, полезность и откуда надежность

Нелинейная оптимизация


Не ML, но близко...

Гибридизация

Как получить лучшее из всех алгоритмов

Итоги

Выводы и что делать дальше?



← Мы здесь

Гибриды?

Гибриды в контексте алгоритмов – это комбинация различных методов (таких как линейные, нелинейные алгоритмы и машинное обучение) для решения сложных задач

Такой подход позволяет объединить сильные стороны каждого метода, чтобы добиться более эффективных и точных результатов, чем при использовании одного типа алгоритмов

Гибридные подходы используются, когда одна методика не справляется с задачей полностью



Например, можно сначала применить линейные алгоритмы для упрощения или предобработки данных, а затем подключить нелинейные методы или ML для более глубокого анализа или предсказаний

В задаче прогнозирования спроса:

Линейная регрессия
Определение общего тренда

+

Нейронная сеть
Выявление сложных сезонных колебаний

Примеры гибридов

Ансамбли в машинном обучении

Это гибридный подход, при котором несколько моделей (например, деревья решений) объединяются для повышения точности предсказаний

Пример:

алгоритм Random Forest, который комбинирует множество деревьев решений

Гибридные системы управления

Линейные регуляторы (например, PID-контроллер) используются для базового управления системой, а нейронные сети добавляются для адаптации к изменениям в окружающей среде

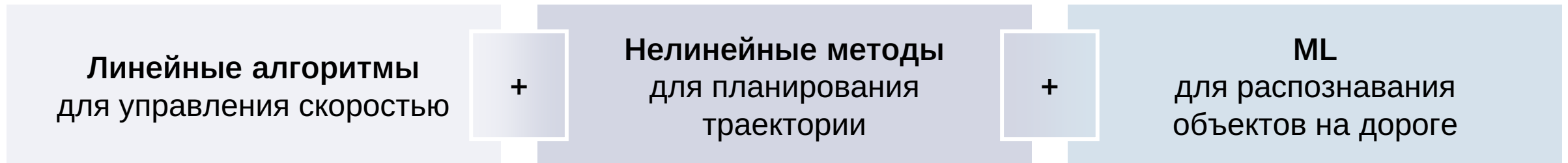
Решение задач | Разбиение на подзадачи

Разбиение сложных задач на более мелкие подзадачи – это процесс декомпозиции, который делает задачу более понятной и управляемой

! Определите этапы решения задачи и разбейте её на логические части

- Упрощает процесс решения, позволяя использовать наиболее подходящий метод для каждой подзадачи
- Повышает общую эффективность и точность, так как разные методы лучше подходят для разных типов проблем

В задаче автономного вождения:

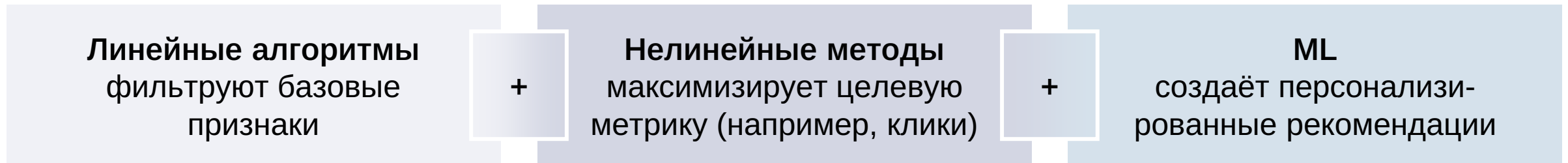


Решение задач | Разбиение на подзадачи

Линейные алгоритмы	Нелинейные алгоритмы	Машинное обучение
Применяются для простых, структурированных подзадач, таких как предобработка данных или базовые вычисления	Используются для задач с нелинейными зависимостями, например, оптимизация сложных процессов	Подходит для анализа больших данных или задач с нечёткими закономерностями, таких как предсказания

- Результаты одной подзадачи передаются как входные данные для следующей
Например, линейные алгоритмы обрабатывают данные, которые затем используются ML-моделью
- Для связи между модулями часто применяются интерфейсы или API

В системе рекомендаций:



Решение задач | Паттерны проектирования

! Паттерны проектирования идеально подходят для реализации гибридных систем, так как они обеспечивают структурированный способ интеграции и замены алгоритмов. Они позволяют организовать код так, чтобы различные алгоритмы могли легко взаимодействовать и заменяться без необходимости переписывать всю систему.

Зачем это делать:

Обеспечивает гибкость

Можно быстро адаптировать систему к новым требованиям

Упрощает поддержку и масштабирование

Добавление новых алгоритмов не нарушает существующую логику

Решение задач | Паттерны проектирования

- **Стратегия:**

Позволяет выбирать алгоритм во время выполнения программы

Например, переключение между линейной и нелинейной оптимизацией в зависимости от задачи

- **Декоратор:**

Добавляет дополнительную функциональность к существующим алгоритмам, не изменяя их код — *Например, кэширование результатов вычислений*

- **Фабричный метод:**

Используется для создания объектов алгоритмов, скрывая детали их реализации
Это упрощает добавление новых методов в систему

Причины популярности гибридных подходов

Эффективность

Комбинирование методов часто даёт лучшие результаты, чем использование одного подхода, особенно для сложных задач

Масштабируемость

Системы легко адаптируются к новым данным или требованиям, что важно в динамичных бизнес-условиях

Командная работа

Разные команды могут специализироваться на своих областях (например, линейные алгоритмы или ML), что ускоряет разработку

Хакатоны как идеальный вариант для обучения

Хакатоны – это мероприятия, где команды за короткое время создают прототипы решений, что делает их отличной площадкой для обучения гибридным подходам

Преимущества

- Участники могут экспериментировать с комбинацией разных алгоритмов, быстро проверяя идеи на практике
- Развивают креативность и навыки командной работы, которые важны для разработки гибридных систем
- Учат принимать решения в условиях ограниченного времени, что полезно для реальных проектов



Основные понятия AI и ML

Что такое, сферы применения, какие ограничения при использовании

Линейные алгоритмы

Область применения, полезность и откуда надежность

Нелинейная оптимизация


Не ML, но близко...

Гибридизация

Как получить лучшее из всех алгоритмов

Итоги

Выводы и что делать дальше?



Мы здесь

Последовательное развитие проекта

1

Начинайте с простых правил, затем переходите к линейным моделям, нелинейным алгоритмам, классическому ML и только в конце – к глубокому обучению

2

Усложняйте архитектуру только тогда, когда простые методы не дают нужной точности

Паттерны для удобства кода:

Модульность

разделяет код на компоненты
(препроцессинг, модели, оценка)

Конфигурационные
файлы упрощают управление
параметрами

Паттерн «Стратегия»

позволяет гибко переключаться
между моделями

Где применять методы?

Линейные модели

Для прозрачных задач с линейными зависимостями
(прогноз продаж)

Нелинейные модели (XGBoost)

Когда важны интерпретируемость и неочевидные паттерны

ML / Deep Learning

Для неструктурированных данных (текст, изображения)
и максимальной точности

Главные выводы

- **Данные важнее моделей:**
80% успеха — качественная подготовка данных
- **Избегайте overengineering:**
Сначала пробуйте простые решения, даже если задача кажется сложной
- **Мониторинг и гибкость:**
Модели нужно постоянно тестировать и обновлять под меняющиеся условия



Спасибо за внимание!
Ваши вопросы?

