

# ML Cheatsheets

**Полная коллекция шпаргалок по машинному  
обучению**

Владимир Гуровиц (школа "Летово")  
DeepSeek, Github Copilot, Perplexity Comet  
Всего шпаргалок: 422

## ◆ 1D-CNN and 3D-CNN

 17 Январь 2026

### ◆ 1. Обзор размерностей

Тип	Входные данные	Применение
1D-CNN	(batch, channels, length)	Временные ряды, текст, аудио
2D-CNN	(batch, channels, H, W)	Изображения
3D-CNN	(batch, channels, D, H, W)	Видео, медицинские 3D снимки

D — глубина (depth), H — высота, W — ширина

### ◆ 2. 1D-CNN: Основы

- **Идея:** свёртка вдоль одной оси (времени)
- **Kernel:** скользит по последовательности
- **Преимущества:** быстрее RNN, параллелизуется
- **Применение:** анализ сигналов, NLP, биоинформатика

**Формула выходного размера:**

```
output_length = (input_length + 2*padding - kernel_size) / stride + 1
```

### ◆ 3. 1D-CNN в PyTorch

```
import torch
import torch.nn as nn

class CNN1D(nn.Module):
    def __init__(self, input_channels=1, num_classes=10):
        super(CNN1D, self).__init__()

        self.conv1 = nn.Conv1d(
            in_channels=input_channels,
            out_channels=64,
            kernel_size=7,
            stride=1,
            padding=3
        )
        self.bn1 = nn.BatchNorm1d(64)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool1d(kernel_size=2)

        self.conv2 = nn.Conv1d(64, 128,
        kernel_size=5, padding=2)
        self.bn2 = nn.BatchNorm1d(128)

        self.conv3 = nn.Conv1d(128, 256,
        kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm1d(256)

        self.global_pool = nn.AdaptiveAvgPool1d(1)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        # x: (batch, channels, length)
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.pool(x)

        x = self.relu(self.bn2(self.conv2(x)))
        x = self.pool(x)

        x = self.relu(self.bn3(self.conv3(x)))
        x = self.pool(x)

        x = self.global_pool(x)  # (batch, 256, 1)
        x = x.squeeze(-1)       # (batch, 256)
        x = self.fc(x)
        return x

# Использование
model = CNN1D(input_channels=1, num_classes=10)
x = torch.randn(32, 1, 1000) # batch, channels, length
output = model(x)
print(output.shape) # (32, 10)
```

### ◆ 4. 1D-CNN: Применения

#### ✓ Временные ряды

- ✓ Финансовые данные
- ✓ Прогнозирование спроса
- ✓ Сенсорные данные IoT
- ✓ ЭКГ, ЭЭГ сигналы

#### ✓ Обработка текста

- ✓ Классификация текстов
- ✓ Sentiment analysis
- ✓ Named Entity Recognition
- ✓ Быстрая альтернатива RNN

#### ✓ Обработка аудио

- ✓ Распознавание речи
- ✓ Классификация звуков
- ✓ Детекция ключевых слов

#### ✓ Биоинформатика

- ✓ Анализ ДНК последовательностей
- ✓ Предсказание структуры белков

### ◆ 5. 1D-CNN с Keras/TensorFlow

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv1D, MaxPooling1D, GlobalAveragePooling1D,
    Dense, Dropout, BatchNormalization
)

model = Sequential([
    # Первый блок
    Conv1D(64, kernel_size=7, activation='relu',
           padding='same', input_shape=(1000, 1)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    # Второй блок
    Conv1D(128, kernel_size=5, activation='relu',
           padding='same'),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    # Третий блок
    Conv1D(256, kernel_size=3, activation='relu',
           padding='same'),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    # Классификатор
    GlobalAveragePooling1D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

## ◆ 6. Dilated (Atrous) Convolutions

- **Идея:** увеличение receptive field без увеличения параметров
- **Dilation rate:** шаг между элементами ядра
- **Применение:** WaveNet, TCN

```
# PyTorch
conv_dilated = nn.Conv1d(
    in_channels=64,
    out_channels=128,
    kernel_size=3,
    dilation=2 # расстояние между элементами
)

# Keras
from tensorflow.keras.layers import Conv1D
Conv1D(128, kernel_size=3, dilation_rate=2)

# Receptive field c dilation:
# dilation=1: kernel видит 3 точки подряд
# dilation=2: kernel видит 3 точки с шагом 2 (5 позиций)
# dilation=4: kernel видит 3 точки с шагом 4 (9 позиций)
```

## ◆ 7. 3D-CNN: Основы

- **Идея:** свёртка в пространстве и времени одновременно
- **Kernel:** 3D куб (глубина × высота × ширина)
- **Преимущество:** захват временной динамики
- **Недостаток:** много параметров, медленнее

**Форма входа:** (batch, channels, depth, height, width)

Для видео: **depth = количество кадров**

## ◆ 8. 3D-CNN в PyTorch

```
import torch
import torch.nn as nn

class CNN3D(nn.Module):
    def __init__(self, num_classes=400):
        super(CNN3D, self).__init__()

        self.conv1 = nn.Conv3d(
            in_channels=3, # RGB
            out_channels=64,
            kernel_size=(3, 7, 7), # (depth, height, width)
            stride=(1, 2, 2),
            padding=(1, 3, 3)
        )
        self.bn1 = nn.BatchNorm3d(64)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool3d(kernel_size=(1, 2, 2))

        self.conv2 = nn.Conv3d(64, 128,
            kernel_size=(3, 5, 5),
            stride=(1, 1, 1),
            padding=(1, 2, 2))
        self.bn2 = nn.BatchNorm3d(128)

        self.conv3 = nn.Conv3d(128, 256,
            kernel_size=(3, 3, 3),
            stride=(1, 1, 1),
            padding=(1, 1, 1))
        self.bn3 = nn.BatchNorm3d(256)

        self.global_pool =
nn.AdaptiveAvgPool3d((1, 1, 1))
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        # x: (batch, channels, depth, height, width)
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.pool(x)

        x = self.relu(self.bn2(self.conv2(x)))
        x = self.pool(x)

        x = self.relu(self.bn3(self.conv3(x)))
        x = self.pool(x)

        x = self.global_pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

# Использование
model = CNN3D(num_classes=400)
```

```
# batch=2, RGB, 16 frames, 224x224
x = torch.randn(2, 3, 16, 224, 224)
output = model(x)
print(output.shape) # (2, 400)
```

## ◆ 9. 3D-CNN: Применения

### ✓ Распознавание действий в видео

- ✓ C3D, I3D, R(2+1)D
- ✓ Спортивная аналитика
- ✓ Системы безопасности

### ✓ Медицинская визуализация

- ✓ CT, MRI сканы (3D объёмы)
- ✓ Сегментация опухолей
- ✓ Детекция аномалий

### ✓ Анализ видео

- ✓ Детекция событий
- ✓ Временная сегментация
- ✓ Video captioning

## ◆ 10. Популярные 3D архитектуры

Архитектура	Особенности	Год
C3D	Базовая 3D CNN, 3×3×3 ядра	2014
I3D	Inflated 2D → 3D, ImageNet pretrain	2017
R(2+1)D	Разделение на 2D spatial + 1D temporal	2018
SlowFast	Два пути: медленный (spatial) + быстрый (temporal)	2019
X3D	Эффективная 3D CNN с прогрессивным расширением	2020

## ◆ 11. R(2+1)D: Decomposed Convolution

**Идея:** разделить 3D свёртку на 2D spatial + 1D temporal

- 3D conv (3×3×3) → 2D conv (1×3×3) + 1D conv (3×1×1)
- Меньше параметров
- Больше нелинейностей (больше ReLU)
- Лучшие производительность

```
# Вместо одной 3D свёртки
nn.Conv3d(in_ch, out_ch, kernel_size=(3, 3, 3))

# Используем две последовательные
nn.Sequential(
    nn.Conv3d(in_ch, mid_ch, kernel_size=(1, 3,
3)), # spatial
    nn.ReLU(),
    nn.Conv3d(mid_ch, out_ch, kernel_size=(3, 1,
1)) # temporal
)
```

## ◆ 12. I3D (Inflated 3D ConvNet)

**Идея:** "раздуть" 2D CNN в 3D

- Взять предобученную 2D CNN (Inception)
- Повторить 2D веса N раз вдоль временной оси
- Разделить на N для сохранения выходов
- Fine-tune на видео данных

```
# Пример инфляции
# 2D фильтр: K × K
# 3D фильтр: N × K × K (повтор N раз)

# Веса инициализации:
# w_3d[t, i, j] = w_2d[i, j] / N для всех t
```

**Преимущество:** transfer learning из ImageNet

## ◆ 13. 1D vs 3D CNN: Сравнение

Аспект	1D-CNN	3D-CNN
Параметры	Мало	Много
Скорость	Быстро	Медленно
Память	Низкая	Высокая
Receptive field	1D (время)	3D (время+пространство)
Применение	Последовательности	Видео, 3D данные

## ◆ 14. Оптимизация 3D-CNN

- **Снижение разрешения:** 112×112 вместо 224×224
- **Меньше кадров:** 8-16 вместо 32-64
- **Temporal downsampling:** stride>1 по времени
- **Mixed precision:** FP16 вместо FP32
- **Gradient checkpointing:** экономия памяти
- **Decomposed conv:** (2+1)D вместо 3D

```
# Gradient checkpointing в PyTorch
from torch.utils.checkpoint import checkpoint

class OptimizedBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv3d(64, 128, 3,
padding=1)
        self.bn = nn.BatchNorm3d(128)
        self.relu = nn.ReLU()

    def forward(self, x):
        # Использовать checkpointing для экономии
памяти
        return checkpoint(self._forward, x)

    def _forward(self, x):
        return self.relu(self.bn(self.conv(x)))
```

## ◆ 15. Подготовка данных для видео

```
import cv2
import numpy as np

def load_video_clip(video_path, num_frames=16,
size=(224, 224)):
    """Загрузить видео клип"""
    cap = cv2.VideoCapture(video_path)
    total_frames =
int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Равномерная выборка кадров
    frame_indices = np.linspace(0, total_frames-1,
num_frames, dtype=int)

    frames = []
    for idx in frame_indices:
        cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
        ret, frame = cap.read()
        if ret:
            frame = cv2.resize(frame, size)
            frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
            frames.append(frame)

    cap.release()

    # (num_frames, H, W, C) → (C, num_frames, H,
W)
    video = np.array(frames).transpose(3, 0, 1, 2)
    video = video / 255.0 # нормализация

    return video

# Использование
video_clip = load_video_clip('video.mp4',
num_frames=16)
print(video_clip.shape) # (3, 16, 224, 224)
```

## ◆ 16. Чек-лист

### Для 1D-CNN:

- [ ] Нормализовать входные данные
- [ ] Выбрать kernel size (3, 5, 7, 9)
- [ ] Использовать dilated conv для большого context
- [ ] BatchNormalization после каждой свёртки
- [ ] Global pooling перед классификатором

### Для 3D-CNN:

- [ ] Снизить разрешение (112×112 или 128×128)
- [ ] Использовать 8-16 кадров
- [ ] Рассмотреть (2+1)D вместо 3D
- [ ] Использовать предобученные веса (I3D)
- [ ] Mixed precision training
- [ ] Gradient checkpointing для экономии памяти

### Объяснение заказчику:

«1D-CNN анализирует последовательности данных (как тренды во времени), а 3D-CNN видит видео целиком — не только что происходит в каждом кадре, но и как объекты движутся между кадрами».



# 3D Computer Vision

Январь 2026

## ◆ 1. Основы 3D Vision

3D координаты, проекции, камеры

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 2. Представление 3D данных

Point clouds, meshes, voxels

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 3. Point Cloud обработка

PointNet, PointNet++

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 4. 3D Object Detection

VoxelNet, PointPillars

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 5. 3D Reconstruction

MVS, NeRF, 3D Gaussian Splatting

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 6. Depth Estimation

Monocular, stereo, RGB-D

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 7. SLAM

### Simultaneous Localization and Mapping

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 8. 3D Pose Estimation

### 6DoF pose, camera pose

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 9. 3D Semantic Segmentation

### Point cloud segmentation

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 10. Libraries & Tools

Open3D, PyTorch3D, Kaolin

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

# activation Functions

 Январь 2026

## 1. Зачем нужны функции активации

- Нелинейность:** позволяют сети моделировать сложные зависимости
- Без активации:** сеть = линейная модель (бесполезно!)
- Где применяются:** после каждого слоя (кроме выходного)
- Выходной слой:** специальные активации (softmax, sigmoid)

## 2. Sigmoid ( $\sigma$ )

**Формула:**  $\sigma(x) = 1 / (1 + e^{-x})$

**Диапазон:** (0, 1)

```
# PyTorch
import torch.nn as nn
nn.Sigmoid()

# Keras
layers.Activation('sigmoid')
```

### ✓ Плюсы:

- Выход в диапазоне (0,1) — вероятность
- Гладкая функция

### ✗ Минусы:

- Исчезающий градиент (vanishing gradient)
- Не центрирована вокруг 0
- Медленные вычисления (exp)

**Когда использовать:** только для выходного слоя в бинарной классификации

## 3. Tanh (гиперболический тангенс)

**Формула:**  $\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$

**Диапазон:** (-1, 1)

```
# PyTorch
nn.Tanh()

# Keras
layers.Activation('tanh')
```

### ✓ Плюсы:

- Центрирована вокруг 0
- Лучше чем sigmoid для скрытых слоёв

### ✗ Минусы:

- Всё ещё исчезающий градиент
- Медленные вычисления

**Когда использовать:** RNN/LSTM (реже для скрытых слоёв)

## ◆ 4. ReLU (Rectified Linear Unit) ★

**Формула:**  $\text{ReLU}(x) = \max(0, x)$

**Диапазон:**  $[0, \infty)$

```
# PyTorch
nn.ReLU()

# Keras
layers.Activation('relu')
# или
layers.ReLU()
```

### ✓ Плюсы:

- Очень быстрая (простое сравнение)
- Нет исчезающего градиента для  $x > 0$
- Разреженная активация (многие нейроны = 0)
- Стандарт для CNN

### ✗ Минусы:

- "Dying ReLU": нейроны могут "умереть" (всегда 0)
- Не центрирована вокруг 0

**Когда использовать:** по умолчанию для большинства задач!

## ◆ 5. Leaky ReLU

**Формула:**  $\text{LeakyReLU}(x) = \max(\alpha x, x)$ , где  $\alpha = 0.01$

**Диапазон:**  $(-\infty, \infty)$

```
# PyTorch
nn.LeakyReLU(negative_slope=0.01)

# Keras
layers.LeakyReLU(alpha=0.01)
```

### ✓ Плюсы:

- Решает проблему dying ReLU
- Малый градиент для  $x < 0$  (не полностью обнуляет)

**Когда использовать:** если ReLU не работает, в GAN

## ◆ 6. ELU (Exponential Linear Unit)

**Формула:**

$\text{ELU}(x) = x$ , если  $x > 0$

$\text{ELU}(x) = \alpha(e^x - 1)$ , если  $x \leq 0$

```
# PyTorch
nn.ELU(alpha=1.0)

# Keras
layers.ELU(alpha=1.0)
```

### ✓ Плюсы:

- Нет dying ReLU
- Выходы близки к 0-mean
- Гладкая всюду

### ✗ Минусы:

- Медленнее ReLU (exp)

## ◆ 7. Swish / SiLU ★

**Формула:**  $\text{Swish}(x) = x \cdot \sigma(x) = x / (1 + e^{-x})$

**Открыта Google, 2017**

```
# PyTorch
nn.SiLU() # То же что Swish
```

```
# Keras
layers.Activation(tf.nn.swish)
```

### ✓ Плюсы:

- Часто лучше ReLU
- Гладкая, нелинейная
- Self-gated (x умножается на свою активацию)
- Используется в EfficientNet, MobileNet

### ✗ Минусы:

- Немного медленнее ReLU

## ◆ 8. GELU (Gaussian Error Linear Unit)



**Стандарт для Transformers!**

**Формула** (приближённо):

$$\text{GELU}(x) \approx 0.5x(1 + \tanh[\sqrt{(2/\pi)}(x + 0.044715x^3)])$$

```
# PyTorch
nn.GELU()
```

```
# Keras
layers.Activation(tf.nn.gelu)
```

### ✓ Плюсы:

- Используется в BERT, GPT
- Стохастическая интерпретация
- Гладкая

**Когда использовать:** Transformers, NLP модели

## ◆ 9. Softmax (выходной слой)

**Для многоклассовой классификации**

**Формула:**  $\text{Softmax}(x_i) = e^{x_i} / \sum_j e^{x_j}$

```
# PyTorch
nn.Softmax(dim=-1)
```

```
# Keras
layers.Activation('softmax')
# или в Dense
layers.Dense(10, activation='softmax')
```

**Свойства:**

- $\Sigma$  выходов = 1 (распределение вероятностей)
- Все выходы  $\in (0, 1)$

**⚠ Важно:** часто встроена в loss функцию  
(CrossEntropyLoss)

## ◆ 10. Сравнительная таблица

Функция	Диапазон	Скорость	Использование
ReLU	$[0, \infty)$	⚡⚡⚡	Стандарт для CNN
Leaky ReLU	$(-\infty, \infty)$	⚡⚡⚡	Вместо ReLU, GAN
ELU	$(-\alpha, \infty)$	⚡⚡	Альтернатива ReLU
Swish/SiLU	$(-\infty, \infty)$	⚡⚡	MobileNet, EfficientNet
GELU	$(-\infty, \infty)$	⚡⚡	Transformers
Sigmoid	$(0, 1)$	⚡	Только выходной слой
Tanh	$(-1, 1)$	⚡	RNN/LSTM
Softmax	$(0, 1), \Sigma=1$	⚡	Многоклассовая классификация

## ◆ 11. Примеры использования

```
# PyTorch CNN
model = nn.Sequential(
    nn.Conv2d(3, 64, 3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(), # ★ Стандарт для CNN
    nn.Conv2d(64, 128, 3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.Flatten(),
    nn.Linear(128 * 8 * 8, 10),
    nn.Softmax(dim=-1) # Выходной слой
)

# Transformer block
class TransformerBlock(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.attention =
            MultiHeadAttention(d_model)
        self.ffn = nn.Sequential(
            nn.Linear(d_model, 4 * d_model),
            nn.GELU(), # ★ Стандарт для Transformers
            nn.Linear(4 * d_model, d_model)
        )
```

## ◆ 12. Когда использовать какую активацию

Архитектура/Задача	Активация
CNN (скрытые слои)	ReLU
RNN/LSTM	Tanh
Transformer	GELU
MobileNet, EfficientNet	Swish/SiLU
GAN (discriminator)	Leaky ReLU
Бинарная классификация (выход)	Sigmoid
Многоклассовая (выход)	Softmax
Регрессия (выход)	Без активации (linear)

## ◆ 13. Проблема исчезающего градиента

**Проблема:** в глубоких сетях с sigmoid/tanh градиенты становятся очень малыми

**Почему:** производная sigmoid/tanh  $\leq 0.25$ , при умножении через слои  $\rightarrow 0$

**Решение:**

- Используйте ReLU/GELU/Swish
- Batch Normalization
- Residual connections (ResNet)
- Правильная инициализация весов

## ◆ 14. PReLU (Parametric ReLU)

### Обучаемый параметр $\alpha$

$\text{PReLU}(x) = \max(\alpha x, x)$ , где  $\alpha$  — обучаемый параметр

```
# PyTorch
nn.PReLU() # α обучается

# Keras
layers.PReLU()
```

Автоматически подбирает оптимальный наклон для отрицательных значений

## ◆ 15. Лучшие практики

- **По умолчанию:** используйте ReLU
- **Для Transformers:** GELU
- **Если ReLU не работает:** попробуйте Leaky ReLU или ELU
- **Современные архитектуры:** Swish/GELU часто лучше
- **Выходной слой:**
  - Бинарная классификация  $\rightarrow$  Sigmoid
  - Многоклассовая  $\rightarrow$  Softmax
  - Регрессия  $\rightarrow$  без активации
- **Не используйте sigmoid/tanh для скрытых слоёв в глубоких сетях!**

## ◆ 16. Чек-лист

- [ ] Для CNN — использовать ReLU
- [ ] Для Transformer — использовать GELU
- [ ] Правильная активация на выходном слое
- [ ] НЕ использовать sigmoid/tanh для скрытых слоёв (кроме RNN)
- [ ] Если dying ReLU — попробовать Leaky ReLU
- [ ] Экспериментировать с Swish для улучшения точности
- [ ] Мониторить "мёртвые" нейроны (всегда 0)

### 💡 Объяснение заказчику:

«Функция активации — это "решающий механизм" нейрона: должен ли он активироваться (передать сигнал дальше) или нет. ReLU — самая простая: если сигнал положительный — пропускаем, если отрицательный — блокируем. Это похоже на переключатель, который включается только при достаточно сильном сигнале».

# Active Learning

3 января 2026

## 1. Суть

- Цель:** минимизировать объем размеченных данных
- Итеративный процесс:** модель выбирает данные для разметки
- Query strategy:** выбираем самые информативные примеры
- Human-in-the-loop:** человек размечает выбранные данные

Выбираем: argmax Информативность(x)

## 2. Базовый процесс

- Начальная разметка:** небольшой размеченный набор
- Обучить модель:** на текущих данных
- Выбрать примеры:** по query strategy
- Разметить:** человек размечает выбранные
- Повторить:** шаги 2-4 до достижения цели

## 3. Базовый код

```
from modAL.models import ActiveLearner
from sklearn.ensemble import RandomForestClassifier

# Начальные данные (малый размеченный набор)
X_labeled = X_train[:100]
y_labeled = y_train[:100]
X_unlabeled = X_train[100:]

# Active learner
learner = ActiveLearner(
    estimator=RandomForestClassifier(),
    X_training=X_labeled,
    y_training=y_labeled
)

# Итеративное обучение
n_queries = 10
batch_size = 20

for i in range(n_queries):
    # Выбрать данные для разметки
    query_idx, query_inst = learner.query(
        X_unlabeled, n_instances=batch_size
    )

    # Получить метки (от эксперта)
    y_new = oracle(X_unlabeled[query_idx])

    # Обучить на новых данных
    learner.teach(X_unlabeled[query_idx], y_new)

    # Удалить из unlabeled
    X_unlabeled = np.delete(X_unlabeled,
                           query_idx, axis=0)

    # Оценка
    score = learner.score(X_test, y_test)
    print(f"Query {i}: Accuracy = {score:.3f}")
```

## 4. Стратегии выбора (Query Strategies)

Стратегия	Описание	Когда использовать
Uncertainty Sampling	Выбор наименее уверенных	По умолчанию, просто
Entropy Sampling	Максимальная энтропия	Многоклассовая классификация
Margin Sampling	Малая разница топ-2 классов	Бинарная классификация
Query-by-Committee	Разногласия в ансамбле	Есть ресурсы для ансамбля
Expected Model Change	Макс. изменение модели	Медленно, но эффективно

## ◆ 5. Uncertainty Sampling

```
# Выбор примеров с наименьшей уверенностью
def uncertainty_sampling(model, X_unlabeled,
n_instances=10):
    # Предсказываем вероятности
    proba = model.predict_proba(X_unlabeled)

    # Уверенность = макс. вероятность
    confidence = proba.max(axis=1)

    # Uncertainty = 1 - confidence
    uncertainty = 1 - confidence

    # Выбираем топ-n самых неуверенных
    query_idx = np.argsort(uncertainty)[-n_instances:]

    return query_idx

# Использование
query_idx = uncertainty_sampling(model,
X_unlabeled, n_instances=20)
X_to_label = X_unlabeled[query_idx]
```

## ◆ 7. Margin Sampling

```
def margin_sampling(model, X_unlabeled,
n_instances=10):
    # Предсказываем вероятности
    proba = model.predict_proba(X_unlabeled)

    # Сортируем вероятности
    proba_sorted = np.sort(proba, axis=1)

    # Margin = разница топ-2 классов
    margin = proba_sorted[:, -1] - proba_sorted[:, -2]

    # Выбираем топ-n с минимальным margin
    query_idx = np.argsort(margin)[:n_instances]

    return query_idx
```

## ◆ 8. Query-by-Committee

```
from sklearn.ensemble import
RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import
LogisticRegression

# Комитет из разных моделей
committee = [
    RandomForestClassifier(),
    GradientBoostingClassifier(),
    LogisticRegression()
]

# Обучаем все модели
for model in committee:
    model.fit(X_labeled, y_labeled)

# Выбор по разногласиям
def query_by_committee(committee, X_unlabeled,
n_instances=10):
    # Предсказания всех моделей
    predictions = np.array([
        model.predict(X_unlabeled) for model in
committee
    ])

    # Разногласия = разнообразие предсказаний
    # Vote entropy
    vote_entropy = []
    for i in range(len(X_unlabeled)):
        votes = predictions[:, i]
        unique, counts = np.unique(votes,
return_counts=True)
        proba = counts / len(committee)
        entropy = -np.sum(proba * np.log(proba +
1e-10))
        vote_entropy.append(entropy)

    vote_entropy = np.array(vote_entropy)

    # Выбираем топ-n с максимальными разногласиями
    query_idx = np.argsort(vote_entropy)[-n_instances:]
    return query_idx
```

## ◆ 6. Entropy Sampling

```
import numpy as np

def entropy_sampling(model, X_unlabeled,
n_instances=10):
    # Предсказываем вероятности
    proba = model.predict_proba(X_unlabeled)

    # Энтропия:  $-\sum p(y) \log p(y)$ 
    entropy = -np.sum(proba * np.log(proba + 1e-
10), axis=1)

    # Выбираем топ-n с максимальной энтропией
    query_idx = np.argsort(entropy)[-n_instances:]

    return query_idx

# Использование
query_idx = entropy_sampling(model, X_unlabeled,
n_instances=20)
```

## ◆ 9. Преимущества и недостатки

### ✓ Преимущества

- ✓ Меньше разметки (10-50% экономии)
- ✓ Фокус на сложных примерах
- ✓ Быстрее достигается высокая точность
- ✓ Экономия затрат на разметку

### ✗ Недостатки

- ✗ Требует итеративного процесса
- ✗ Нужен доступ к эксперту
- ✗ Модель может быть смещённой
- ✗ Не всегда работает (зависит от данных)

## ◆ 10. Когда использовать

### ✓ Хорошо подходит

- ✓ Дорогая разметка
- ✓ Много неразмеченных данных
- ✓ Доступен эксперт для разметки
- ✓ Классификация/регрессия

### ✗ Плохо подходит

- ✗ Дешевая разметка
- ✗ Мало данных вообще
- ✗ Нет доступа к эксперту
- ✗ Батчевое обучение

## ◆ 11. Pool-based vs Stream-based

Подход	Описание	Когда использовать
Pool-based	Есть пул неразмеченных данных	Оффлайн, датасет доступен
Stream-based	Данные приходят потоком	Онлайн, реал-тайм

## ◆ 12. Использование с scikit-learn

```
from sklearn.ensemble import
RandomForestClassifier
import numpy as np

# Начальные данные
labeled_idx = np.random.choice(len(X_train), 100,
replace=False)
X_labeled = X_train[labeled_idx]
y_labeled = y_train[labeled_idx]

unlabeled_idx = np.setdiff1d(range(len(X_train)),
labeled_idx)
X_unlabeled = X_train[unlabeled_idx]

# Модель
model = RandomForestClassifier()
model.fit(X_labeled, y_labeled)

# Active learning цикл
for iteration in range(10):
    # Query strategy (uncertainty sampling)
    proba = model.predict_proba(X_unlabeled)
    uncertainty = 1 - proba.max(axis=1)
    query_idx = np.argsort(uncertainty)[-20:] # топ-20

    # Симуляция разметки (в реальности - человек)
    X_new = X_unlabeled[query_idx]
    y_new = y_train[unlabeled_idx[query_idx]]

    # Обновить данные
    X_labeled = np.vstack([X_labeled, X_new])
    y_labeled = np.hstack([y_labeled, y_new])
    X_unlabeled = np.delete(X_unlabeled,
query_idx, axis=0)
    unlabeled_idx = np.delete(unlabeled_idx,
query_idx)

    # Переобучить
    model.fit(X_labeled, y_labeled)
    score = model.score(X_test, y_test)
    print(f"Iter {iteration}: {len(X_labeled)} samples, Acc={score:.3f}")
```

## ◆ 13. Визуализация процесса

```
import matplotlib.pyplot as plt

# Сохраняем историю
history = {'n_samples': [], 'accuracy': []}

# В каждой итерации
history['n_samples'].append(len(X_labeled))
history['accuracy'].append(model.score(X_test,
y_test))

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(history['n_samples'],
history['accuracy'], 'o-')
plt.xlabel('Число размеченных примеров')
plt.ylabel('Accuracy')
plt.title('Active Learning Progress')
plt.grid(True)
plt.show()
```

## ◆ 14. Практические советы

- **Начните с малого:** 50-100 примеров достаточно
- **Batch size:** 10-50 примеров за итерацию
- **Diversity:** комбинируйте uncertainty с diversity sampling
- **Валидация:** отслеживайте точность на hold-out set
- **Стоп-критерий:** когда точность перестает расти
- **Стратегия:** начните с uncertainty sampling

## ◆ 15. Чек-лист

- [ ] Подготовить начальный размеченный набор
- [ ] Выбрать query strategy
- [ ] Определить batch size
- [ ] Настроить процесс разметки
- [ ] Реализовать итеративный цикл
- [ ] Мониторить прогресс
- [ ] Определить стоп-критерий
- [ ] Сравнить с random sampling

*«Active Learning — мощная техника для минимизации затрат на разметку данных. Позволяет достичь высокой точности с 10-50% объема размеченных данных по сравнению с random sampling».*

## 🔗 Полезные ссылки

-  [modAL: Active Learning library](#)
-  [Active Learning Literature Survey](#)
-  [Wikipedia: Active Learning](#)

## Actor-Critic методы

 17 Январь 2026

### ◆ 1. Суть Actor-Critic

- **Actor:** политика  $\pi(a|s)$  - выбирает действия
- **Critic:** функция ценности  $V(s)$  - оценивает состояния
- **Комбинация:** policy gradient + value-based
- **Преимущество:** стабильность и эффективность

*Actor решает, что делать. Critic говорит, насколько хорошо получилось.*

### ◆ 2. Архитектура

```
# Actor: выбирает действия
π_θ(a|s) - политика с параметрами θ

# Critic: оценивает состояния
V_w(s) - функция ценности с параметрами w

# Advantage
A(s,a) = Q(s,a) - V(s)
          = r + γV(s') - V(s) (TD error)
```

### ◆ 3. Базовый алгоритм

1. Наблюдаем состояние  $s$
2. Actor выбирает действие  $a \sim \pi_\theta(a|s)$
3. Получаем reward  $r$  и новое состояние  $s'$
4. Critic вычисляет TD error:  $\delta = r + \gamma V_w(s') - V_w(s)$
5. Обновляем Critic:  $w \leftarrow w + \alpha_w \delta \nabla_w V_w(s)$
6. Обновляем Actor:  $\theta \leftarrow \theta + \alpha_\theta \delta \nabla_\theta \log \pi_\theta(a|s)$

### ◆ 4. Реализация PyTorch

```
import torch
import torch.nn as nn

class ActorCritic(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.shared = nn.Sequential(
            nn.Linear(state_dim, 128),
            nn.ReLU()
        )
        # Actor head
        self.actor = nn.Linear(128, action_dim)
        # Critic head
        self.critic = nn.Linear(128, 1)

    def forward(self, state):
        features = self.shared(state)
        action_probs =
F.softmax(self.actor(features), dim=-1)
        value = self.critic(features)
        return action_probs, value
```

### ◆ 5. Обучение

```
def train_step(state, action, reward, next_state,
done):
    # Forward pass
    probs, value = model(state)
    _, next_value = model(next_state)

    # TD error (advantage)
    td_target = reward + gamma * next_value * (1 -
done)
    advantage = td_target - value

    # Critic loss
    critic_loss = advantage.pow(2).mean()

    # Actor loss
    log_prob = torch.log(probs[action])
    actor_loss = -(log_prob *
advantage.detach()).mean()

    # Total loss
    loss = actor_loss + critic_loss

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

## ◆ 6. A2C (Advantage Actor-Critic)

```
# Синхронный parallel training
for episode in range(num_episodes):
    states, actions, rewards = [], [], []
    # Сбор траектории
    for step in range(max_steps):
        action_probs, _ = model(state)
        action = sample(action_probs)
        next_state, reward, done =
env.step(action)

        states.append(state)
        actions.append(action)
        rewards.append(reward)

        if done:
            break
    # Обновление
    update(states, actions, rewards)
```

## ◆ 7. A3C (Asynchronous)

```
import multiprocessing as mp

class Worker(mp.Process):
    def __init__(self, global_model):
        super().__init__()
        self.global_model = global_model
        self.local_model = ActorCritic()

    def run(self):
        while not done:
            # Собрать траекторию
            trajectory = collect_trajectory()

            # Вычислить градиенты
            gradients =
compute_gradients(trajectory)

            # Обновить глобальную модель
            self.global_model.apply_gradients(gradients)

            # Синхронизация
            self.local_model.load_state_dict(
                self.global_model.state_dict()
            )
```

## ◆ 8. GAE (Generalized Advantage)

```
# Сглаженная оценка advantage
δ_t = r_t + γV(s_{t+1}) - V(s_t)

A^{GAE}_t = Σ (γλ)^l δ_{t+l}

λ ∈ [0, 1] - trade-off:
λ = 0: только TD error
λ = 1: Monte Carlo return

def compute_gae(rewards, values, gamma=0.99,
    lam=0.95):
    advantages = []
    gae = 0

    for t in reversed(range(len(rewards))):
        delta = rewards[t] + gamma * values[t+1] -
values[t]
        gae = delta + gamma * lam * gae
        advantages.insert(0, gae)

    return advantages
```

## ◆ 10. Continuous actions

```
# Для непрерывных действий: Gaussian policy
class ContinuousActor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.mu = nn.Linear(state_dim, action_dim)
        self.log_std =
nn.Parameter(torch.zeros(action_dim))

    def forward(self, state):
        mu = self.mu(state)
        std = torch.exp(self.log_std)
        dist = Normal(mu, std)
        return dist

    # Sampling
    dist = actor(state)
    action = dist.sample()
    log_prob = dist.log_prob(action)
```

## ◆ 9. PPO (Proximal Policy Optimization)

```
# Clipped surrogate objective
r_t(θ) = π_θ(a|s) / π_θ_old(a|s) # importance
sampling ratio

L^{CLIP}(θ) = E[min(
    r_t(θ) * A_t,
    clip(r_t(θ), 1-ε, 1+ε) * A_t
)]
ε - clip parameter (обычно 0.2)
```

## ◆ 11. Tricks для стабильности

- Baseline subtraction:** убрать  $V(s)$  из returns
- Entropy bonus:**  $+β^*H(\pi)$  для exploration
- Value clipping:** ограничить изменения  $V$
- Gradient clipping:** norm  $\leq 0.5$
- Orthogonal initialization**

## ◆ 12. Сравнение методов

Метод	Плюсы	Минусы
A2C	Простота, стабильность	Медленнее A3C
A3C	Скорость (parallel)	Нестабильность
PPO	State-of-the-art	Сложность

## ◆ 13. Gym пример

```
import gym

env = gym.make("CartPole-v1")
model = ActorCritic(4, 2)
optimizer = torch.optim.Adam(model.parameters())

for episode in range(1000):
    state = env.reset()
    done = False

    while not done:
        # Select action
        probs, value =
model(torch.FloatTensor(state))
        action = torch.multinomial(probs,
1).item()

        # Environment step
        next_state, reward, done, _ =
env.step(action)

        # Train
        train_step(state, action, reward,
next_state, done)

        state = next_state
```

## ◆ 14. Hyperparameters

Параметр	Значение
Learning rate (actor)	1e-4
Learning rate (critic)	1e-3
Gamma ( $\gamma$ )	0.99
GAE lambda ( $\lambda$ )	0.95
Entropy coef	0.01

## ◆ 15. Best Practices

- [ ] Использовать GAE для advantage estimation
- [ ] Нормализовать rewards
- [ ] Gradient clipping
- [ ] Entropy bonus для exploration
- [ ] Правильная инициализация весов
- [ ] Логировать метрики (reward, loss)

## ◆ 16. Чек-лист

- [ ] Окружение выбрано
- [ ] Actor и Critic реализованы
- [ ] GAE настроен
- [ ] Гиперпараметры подобраны
- [ ] Мониторинг работает

«Actor-Critic — это как водитель (*actor*) и инструктор (*critic*): один управляет, другой оценивает и направляет обучение».

# AdaBoost

17 Январь 2026

## 1. Суть AdaBoost

**Adaptive Boosting:** последовательное обучение слабых классификаторов

- Идея:** каждый следующий классификатор фокусируется на ошибках предыдущих
- Веса примеров:** увеличиваются для неправильно классифицированных
- Веса классификаторов:** зависят от точности
- Финальное предсказание:** взвешенное голосование всех классификаторов
- Слабые классификаторы:** обычно decision stumps (деревья глубины 1)

 AdaBoost превращает множество слабых классификаторов в один сильный

## 2. Алгоритм AdaBoost

Шаги:

- Инициализация:  $w_i = 1/N$  для всех примеров
- для  $t = 1$  до  $T$ :
  - Обучить слабый классификатор  $h_t$  на взвешенных данных
  - Вычислить ошибку:  $\epsilon_t = \sum w_i * I(y_i \neq h_t(x_i))$
  - Вычислить вес классификатора:  
 $\alpha_t = 0.5 * \ln((1 - \epsilon_t) / \epsilon_t)$
  - Обновить веса примеров:  
 $w_i = w_i * \exp(-\alpha_t * y_i * h_t(x_i))$
  - Нормализовать веса:  $w_i = w_i / \sum w_i$
- Финальный классификатор:  
 $H(x) = \text{sign}(\sum \alpha_t * h_t(x))$

## 3. Базовый пример (Классификация)

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Генерация данных
X, y = make_classification(n_samples=1000, n_features=20,
                           n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Создание модели
ada = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    learning_rate=1.0,
    random_state=42
)

# Обучение
ada.fit(X_train, y_train)

# Предсказание
y_pred = ada.predict(X_test)
y_proba = ada.predict_proba(X_test)

# Оценка
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.3f}")
```

## 4. Параметры AdaBoost

Параметр	Описание	Рекомендации
n_estimators	Число слабых классификаторов	50-500
learning_rate	Вес каждого классификатора	0.1-1.0
base_estimator	Базовый классификатор	DecisionTreeClassifier
algorithm	'SAMME' или 'SAMME.R'	'SAMME.R' (для вероятностей)
random_state	Seed для воспроизводимости	Любое целое

## ◆ 5. Регрессия с AdaBoost

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# Создание данных
X, y = make_regression(n_samples=1000, n_features=10,
                       noise=10, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Модель
ada_reg = AdaBoostRegressor(
    base_estimator=DecisionTreeRegressor(max_depth=4),
    n_estimators=100,
    learning_rate=1.0,
    loss='linear', # 'linear', 'square', 'exponential'
    random_state=42
)

# Обучение
ada_reg.fit(X_train, y_train)

# Предсказание
y_pred = ada_reg.predict(X_test)

# Оценка
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"RMSE: {rmse:.3f}")
```

## ◆ 6. Подбор гиперпараметров

```
from sklearn.model_selection import GridSearchCV

# Сетка параметров
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5, 1.0],
    'base_estimator__max_depth': [1, 2, 3]
}

# Grid Search
ada = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(),
    random_state=42
)

grid_search = GridSearchCV(
    ada, param_grid,
    cv=5, scoring='accuracy',
    n_jobs=-1, verbose=1
)

grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best CV score: {grid_search.best_score_:.3f}")

# Лучшая модель
best_ada = grid_search.best_estimator_
test_score = best_ada.score(X_test, y_test)
print(f"Test score: {test_score:.3f}")
```

## ◆ 7. Важность признаков

```
import matplotlib.pyplot as plt

# Обучение модели
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)

# Важность признаков
importances = ada.feature_importances_
indices = np.argsort(importances)[::-1]

# Визуализация
plt.figure(figsize=(12, 6))
plt.title('Feature Importances (AdaBoost)')
plt.bar(range(X_train.shape[1]), importances[indices])
plt.xticks(range(X_train.shape[1]), indices)
plt.xlabel('Feature Index')
plt.ylabel('Importance')
plt.grid(axis='y', alpha=0.3)
plt.show()

# Топ-N признаков
top_n = 10
top_features = indices[:top_n]
print(f"Top {top_n} features: {top_features}")
print(f"Importances: {importances[top_features]}")
```

## ◆ 8. Staged predictions

```
# Staged predictions для анализа
import matplotlib.pyplot as plt

ada = AdaBoostClassifier(n_estimators=100, random_state=42)
ada.fit(X_train, y_train)

# Оценка на каждом этапе
train_scores = []
test_scores = []

for y_train_pred in ada.staged_predict(X_train):
    train_scores.append(accuracy_score(y_train, y_train_pred))

for y_test_pred in ada.staged_predict(X_test):
    test_scores.append(accuracy_score(y_test, y_test_pred))

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(train_scores, label='Training Accuracy', linewidth=2)
plt.plot(test_scores, label='Test Accuracy', linewidth=2)
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost: Training vs Test Accuracy')
plt.legend()
plt.grid(alpha=0.3)
plt.show()

# Определение оптимального числа классификаторов
optimal_n = np.argmax(test_scores) + 1
print(f"Optimal n_estimators: {optimal_n}")
print(f"Best test accuracy: {test_scores[optimal_n-1]:.3f}")
```

## ◆ 9. Сравнение с другими методами

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier

# Модели
models = {
    'Decision Tree': DecisionTreeClassifier(max_depth=5),
    'Random Forest':
    RandomForestClassifier(n_estimators=100),
    'AdaBoost': AdaBoostClassifier(n_estimators=100),
    'Gradient Boosting':
    GradientBoostingClassifier(n_estimators=100)
}

# Сравнение
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)
    results[name] = {'train': train_score, 'test': test_score}

# Визуализация
import pandas as pd

df_results = pd.DataFrame(results).T
df_results.plot(kind='bar', figsize=(10, 6))
plt.title('Model Comparison')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.xticks(rotation=15)
plt.legend(['Train', 'Test'])
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

print(df_results)
```

## ◆ 10. Преимущества и недостатки

### • ✓ Преимущества:

- Высокая точность для многих задач
- Мало гиперпараметров для настройки
- Автоматический feature selection
- Не требует масштабирования признаков
- Работает с категориальными и числовыми данными

### • ✗ Недостатки:

- Чувствителен к шуму и выбросам
- Склонен к переобучению
- Медленнее Random Forest (последовательное обучение)
- Плохо работает с несбалансированными данными
- Требует тщательного подбора n\_estimators

## ◆ 11. Работа с несбалансированными данными

```
from sklearn.utils import class_weight

# Вычисление весов классов
class_weights = class_weight.compute_class_weight(
    'balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weight_dict = dict(enumerate(class_weights))

# AdaBoost с весами
from sklearn.tree import DecisionTreeClassifier

base_clf = DecisionTreeClassifier(
    max_depth=1,
    class_weight=class_weight_dict
)

ada = AdaBoostClassifier(
    base_estimator=base_clf,
    n_estimators=100,
    random_state=42
)
ada.fit(X_train, y_train)

# SMOTE для балансировки
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

smote = SMOTE(random_state=42)
pipeline = ImbPipeline([
    ('smote', smote),
    ('adaboost', AdaBoostClassifier(n_estimators=100))
])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
```

## ◆ 12. Early Stopping

```
from sklearn.model_selection import train_test_split

# Разделение на train и validation
X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)

# Обучение с ранней остановкой
best_score = 0
best_n = 0
patience = 10
no_improvement = 0

for n in range(10, 501, 10):
    ada = AdaBoostClassifier(n_estimators=n,
                            random_state=42)
    ada.fit(X_tr, y_tr)
    val_score = ada.score(X_val, y_val)

    if val_score > best_score:
        best_score = val_score
        best_n = n
        no_improvement = 0
    else:
        no_improvement += 1

    if no_improvement >= patience:
        print(f"Early stopping at {n} estimators")
        break

print(f"Best n_estimators: {best_n}")
print(f"Best validation score: {best_score:.3f}")

# Обучение финальной модели
final_ada = AdaBoostClassifier(n_estimators=best_n,
                                random_state=42)
final_ada.fit(X_train, y_train)
```

## ◆ 13. Когда использовать AdaBoost

**Используйте AdaBoost когда:**

- Данные с небольшим шумом
- Нужна высокая точность
- Важна интерпретируемость (с decision stumps)
- Есть время на подбор параметров
- Данные достаточно сбалансированы

**НЕ используйте когда:**

- Много шума и выбросов в данных
- Данные сильно несбалансированы
- Нужна высокая скорость обучения
- Данные высокоразмерны с большим числом признаков

## ◆ 14. Чек-лист использования

1.  Проверить данные на выбросы и шум
2.  Выбрать базовый классификатор (обычно decision stump)
3.  Начать с n\_estimators=50-100
4.  Подобрать learning\_rate (0.1-1.0)
5.  Использовать cross-validation для оценки
6.  Проверить на переобучение (staged predictions)
7.  Настроить n\_estimators с early stopping
8.  Оценить важность признаков
9.  Сравнить с другими ансамблями
10.  Финальная оценка на тестовой выборке



# Adaptive Windowing

 Январь 2026

## ◆ 1. Суть

- **Адаптивное окно:** изменяемый размер окна для streaming данных
- **Concept drift:** изменение распределения данных во времени
- **Цель:** адаптация модели к изменениям без полного переобучения
- **Trade-off:** скорость адаптации vs стабильность

## ◆ 2. Типы concept drift

Тип	Описание	Пример
Sudden	Резкое изменение	Смена сезона
Gradual	Постепенный переход	Тренд
Incremental	Медленное изменение	Старение
Recurring	Циклическое	День/ночь

## ◆ 4. ADWIN алгоритм

### Adaptive Windowing:

- Автоматическое изменение размера окна
- Обнаружение drift через статистические тесты
- Удаление старых данных при обнаружении изменений

```
from river import drift
detector = drift.ADWIN(delta=0.002)

for i, (x, y) in enumerate(stream):
    # Обновить детектор
    detector.update(error)

    # Проверить drift
    if detector.drift_detected:
        print(f"Drift detected at {i}")
        # Сбросить/переобучить модель
        model = create_new_model()

    # Обучить модель
    model.learn_one(x, y)
```

## ◆ 3. Методы окон

- **Fixed Window:** постоянный размер
- **Sliding Window:** сдвиг на фиксированный шаг
- **Landmark Window:** от начала потока
- **Damped Window:** экспоненциальное забывание
- **Adaptive Window:** динамический размер

## ◆ 5. Exponential Weighted Moving Average

$$\text{EWMA}_t = \alpha * x_t + (1-\alpha) * \text{EWMA}_{t-1}$$

$\alpha \in (0, 1)$  - коэффициент забывания

```
import numpy as np

class EWMADetector:
    def __init__(self, alpha=0.1, threshold=3):
        self.alpha = alpha
        self.threshold = threshold
        self.mean = None
        self.std = None

    def update(self, value):
        if self.mean is None:
            self.mean = value
            self.std = 0
        else:
            # Обновление
            delta = value - self.mean
            self.mean += self.alpha * delta
            self.std = (1 - self.alpha) *
            (self.std + self.alpha * delta**2)**0.5

        # Drift detection
        if abs(value - self.mean) > self.threshold
        * self.std:
            return True # drift detected
        return False
```

## ◆ 6. Page-Hinkley Test

### Cumulative sum test:

```
from river import drift

ph = drift.PageHinkley(
    min_instances=30,
    delta=0.005,
    threshold=50,
    alpha=1-0.0001
)

for x, y in stream:
    error = abs(y_pred - y)
    ph.update(error)

    if ph.drift_detected:
        print("Drift detected!")
        # Reset or retrain
```

## ◆ 7. Sliding Window с размером по качеству

```
class QualityBasedWindow:
    def __init__(self, min_size=100,
                 max_size=1000,
                 quality_threshold=0.8):
        self.data = []
        self.min_size = min_size
        self.max_size = max_size
        self.threshold = quality_threshold

    def add(self, x, y, pred):
        self.data.append((x, y, pred))

        # Compute quality
        if len(self.data) > self.min_size:
            recent_acc =
            self._compute_recent_accuracy()

            # Shrink window if quality is low
            if recent_acc < self.threshold:
                self.data = self.data[-self.min_size:]

            # Grow if quality is high
            elif len(self.data) < self.max_size:
                pass # Keep growing

            # Maintain max size
            if len(self.data) > self.max_size:
                self.data = self.data[-self.max_size:]

    def _compute_recent_accuracy(self, n=100):
        recent = self.data[-n:]
        correct = sum(y == pred for _, y, pred in recent)
        return correct / len(recent)
```

## ◆ 8. Ensemble с разными окнами

- Несколько моделей с разными размерами окон
  - Взвешенное голосование по качеству
  - Покрывает разные типы drift
- ```
class MultiWindowEnsemble:
    def __init__(self, window_sizes=[100, 500,
                                    1000]):
        self.models = [create_model() for _ in
                      window_sizes]
        self.windows = [[] for _ in window_sizes]
        self.window_sizes = window_sizes
        self.weights = [1.0] * len(window_sizes)

    def predict(self, x):
        preds = [m.predict(x) for m in
                 self.models]
        # Взвешенное голосование
        weighted = sum(w * p for w, p in
                      zip(self.weights, preds))
        return weighted / sum(self.weights)

    def update(self, x, y):
        for i, (model, window, size) in enumerate(
            zip(self.models, self.windows,
                 self.window_sizes)):
            window.append((x, y))
            if len(window) > size:
                window.pop(0)

        # Переобучить на окне
        model.fit([x for x, y in window],
                  [y for x, y in window])

    # Обновить вес по качеству
    acc = compute_accuracy(model, window)
    self.weights[i] = acc
```

## ◆ 9. Метрики для оценки

- **Prequential accuracy:** тестировать перед обучением
- **Kappa statistic:** с учётом дисбаланса
- **Detection delay:** скорость обнаружения drift
- **False alarm rate:** ложные срабатывания

```
# Prequential evaluation
from river import metrics

metric = metrics.Accuracy()

for x, y in stream:
    # Predict THEN learn
    y_pred = model.predict_one(x)
    metric.update(y, y_pred)
    model.learn_one(x, y)

    if i % 100 == 0:
        print(f"Accuracy: {metric.get()}")
```

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Streaming данные
- ✓ Concept drift ожидается
- ✓ Нужна онлайн-адаптация
- ✓ Ограниченнная память

### ✗ Плохо

- ✗ Статичные данные
- ✗ Нет drift
- ✗ Batch обучение возможно

## ◆ 11. Библиотеки

| Библиотека       | Язык         | Особенности                |
|------------------|--------------|----------------------------|
| River            | Python       | Online ML, drift detection |
| scikit-multiflow | Python       | Streaming ML (deprecated)  |
| MOA              | Java         | Massive Online Analysis    |
| Spark Streaming  | Scala/Python | Distributed streaming      |

## ◆ 12. Чек-лист

- [ ] Определить тип ожидаемого drift
- [ ] Выбрать метод обнаружения drift
- [ ] Настроить параметры чувствительности
- [ ] Реализовать prequential evaluation
- [ ] Мониторить false alarm rate
- [ ] Сравнить с fixed window baseline
- [ ] Измерить задержку обнаружения

### 💡 Объяснение заказчику:

«Adaptive windowing — это как подстраивающееся окно просмотра. Когда данные меняются, мы автоматически забываем старые примеры и фокусируемся на новых. Это позволяет модели быстро адаптироваться к изменениям без полного переобучения».

# Advanced Model Ensembling

 17 Январь 2026

## 1. Stacking (Meta-Learning)

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Base models
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100)),
    ('svm', SVC(probability=True)),
    ('lr', LogisticRegression())
]

# Meta-model
stacking = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    cv=5 # Cross-validation для избежания overfitting
)

stacking.fit(X_train, y_train)
```

## 2. Blending

```
# Split data
X_train1, X_train2, y_train1, y_train2 =
train_test_split(
    X_train, y_train, test_size=0.5
)

# Train base models на первой части
models = [rf, svm, lr]
for model in models:
    model.fit(X_train1, y_train1)

# Предсказания на второй части
meta_features = np.column_stack([
    model.predict_proba(X_train2) for model in models
])

# Train meta-model
meta_model = LogisticRegression()
meta_model.fit(meta_features, y_train2)
```

## 3. Weighted Averaging

```
from scipy.optimize import minimize

def weighted_ensemble(predictions, weights):
    return np.average(predictions, axis=0, weights=weights)

def loss_func(weights, predictions, y_true):
    ensemble_pred = weighted_ensemble(predictions, weights)
    return -accuracy_score(y_true, ensemble_pred.argmax(axis=1))

# Оптимизация весов
result = minimize(
    loss_func,
    x0=[1/len(models)] * len(models),
    args=(val_predictions, y_val),
    bounds=[(0, 1)] * len(models),
    constraints={'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
)

optimal_weights = result.x
```

## 4. Bagging с различными подвыборками

- **Bootstrap aggregating:** случайные подвыборки с возвратом
- **Pasting:** без возврата
- **Random Subspaces:** случайные признаки
- **Random Patches:** случайные примеры + признаки

## ◆ 5. Snapshot Ensembles

```
import torch

class SnapshotEnsemble:
    def __init__(self, model, num_snapshots=5):
        self.model = model
        self.snapshots = []
        self.num_snapshots = num_snapshots

    def save_snapshot(self):
        snapshot = copy.deepcopy(self.model.state_dict())
        self.snapshots.append(snapshot)

    def predict(self, X):
        predictions = []
        for snapshot in self.snapshots:
            self.model.load_state_dict(snapshot)
            self.model.eval()
            with torch.no_grad():
                pred = self.model(X)
            predictions.append(pred)

        # Average predictions
        return torch.mean(torch.stack(predictions), dim=0)
```

## ◆ 7. Multi-Level Stacking

```
# Level 0: базовые модели
level0_models = [rf1, rf2, gbm1, gbm2, nn]

# Level 1: мета-модели
level1_models = [lr, ridge]

# Level 2: финальная модель
final_model = LogisticRegression()

# Обучение
for model in level0_models:
    model.fit(X_train, y_train)

level0_preds = [m.predict_proba(X_val) for m in
level0_models]
X_level1 = np.column_stack(level0_preds)

for model in level1_models:
    model.fit(X_level1, y_val)

level1_preds = [m.predict_proba(X_level1) for m in
level1_models]
X_level2 = np.column_stack(level1_preds)

final_model.fit(X_level2, y_val)
```

## ◆ 8. Сравнение методов

| Метод    | Complexity | Overfitting риск | Качество |
|----------|------------|------------------|----------|
| Voting   | Низкая     | Низкий           | Среднее  |
| Bagging  | Средняя    | Низкий           | Хорошее  |
| Stacking | Высокая    | Средний          | Отличное |
| Blending | Средняя    | Средний          | Хорошее  |

## ◆ 9. Чек-лист

- [ ] Использовать разнообразные base models
- [ ] Cross-validation для meta-model
- [ ] Не переобучать на train set
- [ ] Мониторить correlation между моделями
- [ ] Оптимизировать веса моделей
- [ ] Проверить улучшение vs single model
- [ ] Balance complexity vs performance

«Ансамбли работают лучше когда базовые модели разнообразны и некоррелированы.  
Diversity > количество моделей».

## ◆ 6. Diversity в ансамблях

- Разные алгоритмы:** RF + GBM + NN
- Разные features:** разные подмножества признаков
- Разные гиперпараметры:** вариации настроек
- Разная инициализация:** для NN



# Adversarial Attacks (Состязательные атаки)

17 Январь 2026

## ◆ 1. Что такое Adversarial Attacks

**Adversarial attack** — специально созданные небольшие возмущения входных данных, которые заставляют модель делать неправильные предсказания.

- **Цель:** обмануть нейросеть минимальными изменениями
- **Проблема:** модели уязвимы к imperceptible изменениям
- **Пример:** изображение панды + шум = гибсон (99.3% уверенности)
- **Применение:** тестирование робастности моделей
- **Защита:** adversarial training, certified defenses

*Adversarial примеры показывают, что нейросети "видят" данные иначе, чем люди, и могут быть легко обманут даже небольшими изменениями, невидимыми для человека.*

## ◆ 2. Типы атак

| Тип       | Описание               | Знание модели                |
|-----------|------------------------|------------------------------|
| White-box | Полный доступ к модели | Архитектура, веса, градиенты |
| Black-box | Только предсказания    | Входы и выходы               |
| Gray-box  | Частичный доступ       | Архитектура без весов        |

### По цели:

- **Untargeted:** любая ошибка классификации
- **Targeted:** конкретный неправильный класс

## ◆ 3. FGSM (Fast Gradient Sign Method)

Самая простая и быстрая атака. Использует знак градиента функции потерь.

```
# PyTorch реализация
import torch
import torch.nn.functional as F

def fgsm_attack(image, epsilon, data_grad):
    # Знак градиента
    sign_data_grad = data_grad.sign()
    # Создание возмущённого изображения
    perturbed_image = image + epsilon * sign_data_grad
    # Ограничение диапазона [0, 1]
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image

# Использование
model.eval()
image.requires_grad = True

output = model(image)
loss = F.nll_loss(output, target)
model.zero_grad()
loss.backward()

# Атака
perturbed_image = fgsm_attack(image, epsilon=0.1,
data_grad=image.grad.data)

# Epsilon контролирует силу атаки (обычно 0.01-0.3)
```

## ◆ 4. PGD (Projected Gradient Descent)

Итеративная версия FGSM. Сильнее и эффективнее.

```
def pgd_attack(model, images, labels, epsilon=0.3,
alpha=0.01, num_iter=40):
    # Копия изображения
    perturbed_images = images.clone().detach()
    perturbed_images.requires_grad = True

    for i in range(num_iter):
        outputs = model(perturbed_images)
        loss = F.cross_entropy(outputs, labels)

        model.zero_grad()
        loss.backward()

        # Шаг градиента
        data_grad = perturbed_images.grad.data
        perturbed_images =
perturbed_images.detach() + alpha *
data_grad.sign()

        # Проецирование в epsilon-шар
        eta = torch.clamp(perturbed_images -
images, -epsilon, epsilon)
        perturbed_images = torch.clamp(images +
eta, 0, 1).detach()
        perturbed_images.requires_grad = True

    return perturbed_images

# PGD — одна из самых сильных white-box атак
```

## ◆ 5. C&W Attack (Carlini & Wagner)

Оптимизационная атака, находит минимальное возмущение.

```
# Упрощённая версия C&W L2
def cw_l2_attack(model, image, target, c=1.0,
kappa=0, max_iter=1000, lr=0.01):
    # Переменные
    w = torch.zeros_like(image,
requires_grad=True)
    optimizer = torch.optim.Adam([w], lr=lr)

    for step in range(max_iter):
        # Создание возмущённого изображения
        perturbed = 0.5 * (torch.tanh(w) + 1)

        # Выход модели
        output = model(perturbed)

        # Loss C&W
        real = output[:, target]
        other = torch.max((1 - F.one_hot(target,
output.shape[1])) * output, dim=1)[0]
        f_loss = torch.clamp(real - other + kappa,
min=0)

        # L2 норма возмущения
        l2_loss = torch.norm(perturbed - image,
p=2)

        # Общая функция потерь
        loss = l2_loss + c * f_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return 0.5 * (torch.tanh(w) + 1)

# C&W находит минимальное возмущение, сложнее
детектировать
```

## ◆ 6. DeepFool

Находит кратчайший путь к границе решения.

```
def deepfool(model, image, num_classes=10,
max_iter=50, overshoot=0.02):
    f_image = model(image).detach()
    I = f_image.argmax().item()

    input_shape = image.shape
    pert_image = image.clone()

    w = torch.zeros(input_shape)
    r_tot = torch.zeros(input_shape)

    for iter in range(max_iter):
        pert_image.requires_grad = True
        fs = model(pert_image)

        # Градиенты
        grads = torch.zeros((num_classes,) +
input_shape[1:])
        for k in range(num_classes):
            model.zero_grad()
            fs[0, k].backward(retain_graph=True)
            grads[k] =
pert_image.grad.data.clone()

        # Находим ближайшую границу
        pert = float('inf')
        for k in range(num_classes):
            if k == I:
                continue

            w_k = grads[k] - grads[I]
            f_k = fs[0, k] - fs[0, I]

            pert_k = abs(f_k) /
torch.norm(w_k.flatten())

            if pert_k < pert:
                pert = pert_k
                w = w_k

        # Добавляем возмущение
        r_i = (pert + 1e-4) * w / torch.norm(w)
        r_tot += r_i

        pert_image = image + (1 + overshoot) *

r_tot

        # Проверяем изменение класса
        pred = model(pert_image).argmax().item()
        if pred != I:
            break
```

```
return r_tot, pert_image, pred
```

## ◆ 7. Сравнение атак

| Атака    | Скорость | Эффективность | Видимость |
|----------|----------|---------------|-----------|
| FGSM     | ⚡⚡⚡      | ★★★           | Средняя   |
| PGD      | ⚡⚡       | ★★★★★         | Средняя   |
| C&W      | ⚡        | ★★★★★         | Низкая    |
| DeepFool | ⚡⚡       | ★★★★★         | Низкая    |

## ◆ 8. Защита: Adversarial Training

Обучение модели на adversarial примерах.

```
def adversarial_training(model, train_loader,
epoches=10, epsilon=0.1):
    optimizer =
        torch.optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(epoches):
        for images, labels in train_loader:
            # Обычное обучение
            outputs = model(images)
            loss_clean = F.cross_entropy(outputs,
   labels)

            # Генерация adversarial примеров
            images.requires_grad = True
            outputs = model(images)
            loss = F.cross_entropy(outputs,
                                  labels)
            model.zero_grad()
            loss.backward()

            # FGSM атака
            perturbed = fgsm_attack(images,
                                     epsilon, images.grad.data)

            # Обучение на adversarial примерах
            outputs_adv = model(perturbed)
            loss_adv =
                F.cross_entropy(outputs_adv, labels)

            # Комбинированная loss
            total_loss = 0.5 * loss_clean + 0.5 *
            loss_adv

            optimizer.zero_grad()
            total_loss.backward()
            optimizer.step()

    return model

# Улучшает робастность, но может снизить точность
# на чистых данных
```

## ◆ 9. Защита: Input Transformations

```
# 1. Добавление шума
def add_noise_defense(image, std=0.05):
    noise = torch.randn_like(image) * std
    return torch.clamp(image + noise, 0, 1)

# 2. JPEG compression
from PIL import Image
import io

def jpeg_compression_defense(image, quality=75):
    # Конвертация в PIL
    pil_image = Image.fromarray((image.numpy() *
                                255).astype('uint8'))
    # Сжатие
    buffer = io.BytesIO()
    pil_image.save(buffer, format="JPEG",
                   quality=quality)
    buffer.seek(0)
    # Обратно в тензор
    compressed = Image.open(buffer)
    return torch.from_numpy(np.array(compressed))
/ 255.0

# 3. Median filtering
from scipy.ndimage import median_filter

def median_filter_defense(image, size=3):
    return torch.from_numpy(
        median_filter(image.numpy(), size=size)
    )

# Эти методы снижают эффективность атак, но могут
# ухудшить точность
```

## ◆ 10. Защита: Ensemble Defenses

```
# Ансамбль моделей как защита
class EnsembleDefense(nn.Module):
    def __init__(self, models):
        super().__init__()
        self.models = models

    def forward(self, x):
        outputs = []
        for model in self.models:
            # Разные трансформации для каждой модели
            x_transformed = self.transform(x)
            outputs.append(model(x_transformed))

        # Усреднение или голосование
        return torch.stack(outputs).mean(dim=0)

    def transform(self, x):
        # Случайные трансформации
        if random.random() > 0.5:
            x = add_noise_defense(x)
        if random.random() > 0.5:
            x = F.interpolate(x, scale_factor=0.9, mode='bilinear')
            x = F.interpolate(x, scale_factor=1.11, mode='bilinear')
        return x

# Transferability атак ниже между разными моделями
```

## ◆ 11. Детектирование атак

```
# 1. Проверка уверенности модели
def detect_by_confidence(model, image, threshold=0.95):
    output = F.softmax(model(image), dim=1)
    max_conf = output.max().item()

    # Подозрительно высокая уверенность
    return max_conf > threshold

# 2. Reconstruction error (Autoencoder)
def detect_by_reconstruction(autoencoder, image, threshold=0.1):
    reconstructed = autoencoder(image)
    error = F.mse_loss(image, reconstructed).item()

    # Adversarial примеры плохо реконструируются
    return error > threshold

# 3. Statistical tests
def detect_by_stats(image, clean_stats):
    mean = image.mean().item()
    std = image.std().item()

    # Сравнение со статистикой чистых изображений
    return abs(mean - clean_stats['mean']) > 3 * clean_stats['std']
```

## ◆ 12. Certified Defenses

Математически доказуемые гарантии робастности.

- **Randomized Smoothing:** добавление гауссового шума
- **Interval Bound Propagation (IBP):** формальная верификация
- **Lipschitz constraints:** ограничение градиентов
- **Certified radius:** гарантированная устойчивость к возмущениям

```
# Randomized Smoothing (упрощённо)
def randomized_smoothing_predict(model, image, num_samples=100, sigma=0.25):
    predictions = []

    for _ in range(num_samples):
        # Добавление шума
        noisy = image + torch.randn_like(image) * sigma
        output = model(noisy)
        predictions.append(output.argmax().item())

    # Мажоритарное голосование
    from collections import Counter
    most_common =
    Counter(predictions).most_common(1)[0]
    return most_common[0], most_common[1] / num_samples

# Гарантирует робастность в определённом радиусе
```

## ◆ 13. Практические рекомендации

### ✓ Best Practices

- ✓ Adversarial training на критичных моделях
- ✓ Комбинация нескольких защит
- ✓ Регулярное тестирование на робастность
- ✓ Мониторинг unusual predictions
- ✓ Ensemble моделей с разными архитектурами
- ✓ Ограничение доступа к модели (black-box)

### ✗ Частые ошибки

- ✗ Игнорирование проблемы безопасности
- ✗ Полагаться только на одну защиту
- ✗ Не тестировать на adversarial примерах
- ✗ Публиковать полные веса модели
- ✗ Переоценивать эффективность простых защит

## ◆ 14. Библиотеки для атак

```
# Foolbox
pip install foolbox

import foolbox as fb

model = fb.PyTorchModel(pytorch_model, bounds=(0, 1))
attack = fb.attacks.FGSM()
advs, clipped, is_adv = attack(model, images,
                                labels, epsilons=0.1)

# CleverHans
pip install cleverhans

from cleverhans.torch.attacks.fast_gradient_method
import fast_gradient_method

adv_x = fast_gradient_method(model, x, eps=0.1,
                             norm=np.inf)

# Adversarial Robustness Toolbox (ART)
pip install adversarial-robustness-toolbox

from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import
PyTorchClassifier

classifier = PyTorchClassifier(model, loss,
                               optimizer, input_shape, nb_classes)
attack = FastGradientMethod(estimator=classifier,
                            eps=0.1)
x_adv = attack.generate(x=x_test)
```

## ◆ 15. Метрики робастности

| Метрика                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Описание                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| Robust Accuracy                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Точность на adversarial примерах    |
| Attack Success Rate                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | % успешных атак                     |
| Perturbation Budget                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Максимальное epsilon для атаки      |
| Certified Radius                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Гарантированный радиус устойчивости |
| <pre># Оценка робастности def evaluate_robustness(model, test_loader,                         attack_fn, epsilon=0.1):     correct_clean = 0     correct_adv = 0     total = 0      for images, labels in test_loader:         # Чистые данные         outputs_clean = model(images)         correct_clean += (outputs_clean.argmax(1) ==                           labels).sum().item()          # Adversarial данные         images_adv = attack_fn(model, images,                                labels, epsilon)         outputs_adv = model(images_adv)         correct_adv += (outputs_adv.argmax(1) ==                        labels).sum().item()      total += labels.size(0)      clean_acc = 100 * correct_clean / total     robust_acc = 100 * correct_adv / total      print(f"Clean Accuracy: {clean_acc:.2f}%")     print(f"Robust Accuracy: {robust_acc:.2f}%")     print(f"Attack Success Rate: {100 - robust_acc:.2f}%")</pre> |                                     |

## ◆ 16. Чек-лист безопасности

- [ ] Тестировать модель на FGSM атаках
- [ ] Тестировать на PGD атаках (сильнее)
- [ ] Применить adversarial training
- [ ] Добавить input transformation
- [ ] Рассмотреть ensemble defenses
- [ ] Внедрить детектирование аномалий
- [ ] Ограничить доступ к модели
- [ ] Мониторить predictions в production
- [ ] Регулярно обновлять защиты
- [ ] Документировать уязвимости

### Объяснение заказчику:

«*Adversarial атаки — это как оптические иллюзии для AI. Мы добавляем невидимые для человека изменения к изображению, но нейросеть видит совсем другой объект. Это критично для безопасности систем распознавания — например, знак "стоп" может быть воспринят как знак ограничения скорости.*»

 Adversarial Attacks

17 Январь 2026

## ◆ 1. Что такое adversarial attacks?

**Adversarial attack** — добавление малых возмущений к входным данным для обмана модели.

- **Цель:** вызвать неправильное предсказание
- **Особенность:** возмущения незаметны для человека
- **Угроза:** безопасность ML систем
- **Применение:** CV, NLP, speech recognition

 "Небольшое изменение изображения может превратить панду в гиббона"

## ◆ 2. Типы атак

| Тип        | Описание                           |
|------------|------------------------------------|
| White-box  | Полный доступ к модели и весам     |
| Black-box  | Доступ только к выходам модели     |
| Gray-box   | Частичный доступ к архитектуре     |
| Targeted   | Целевая ошибка на конкретный класс |
| Untargeted | Любая неправильная классификация   |
| Physical   | Атаки в реальном мире (стикеры)    |

### ◆ 3. FGSM (Fast Gradient Sign Method)

Простейший и быстрый метод генерации adversarial examples.

**Формула:**  $x_{\text{adv}} = x + \epsilon \times \text{sign}(\nabla_x L(\theta, x, y))$

```
import torch
import torch.nn.functional as F

def fgsm_attack(model, x, y, epsilon=0.03):
    """
    FGSM атака на модель

    Args:
        model: нейронная сеть
        x: входное изображение
        y: истинная метка
        epsilon: сила возмущения
    """
    # Требуем градиенты
    x.requires_grad = True

    # Forward pass
    output = model(x)
    loss = F.cross_entropy(output, y)

    # Backward pass
    model.zero_grad()
    loss.backward()

    # Знак градиента
    data_grad = x.grad.data

    # Создаем adversarial example
    perturbed_x = x + epsilon * data_grad.sign()

    # Клиппинг к валидному диапазону
    perturbed_x = torch.clamp(perturbed_x, 0, 1)

    return perturbed_x

# Использование
x_adv = fgsm_attack(model, image, label,
                     epsilon=0.03)
pred_clean = model(image).argmax()
pred_adv = model(x_adv).argmax()
print(f"Clean: {pred_clean}, Adversarial: {pred_adv}")
```

### ◆ 4. PGD (Projected Gradient Descent)

Итеративный метод, более сильный чем FGSM.

- Несколько шагов с малым  $\epsilon$
- Проекция на допустимую область
- Случайная инициализация

```
def pgd_attack(model, x, y, epsilon=0.03,
               alpha=0.01, num_iter=40):
    """
    PGD атака - итеративный FGSM

    Args:
        alpha: размер шага
        num_iter: количество итераций
    """
    # Случайная инициализация
    delta = torch.zeros_like(x).uniform_(-epsilon,
   epsilon)
    delta.requires_grad = True

    for i in range(num_iter):
        # Forward
        output = model(x + delta)
        loss = F.cross_entropy(output, y)

        # Backward
        loss.backward()

        # Gradient ascent step
        delta.data = delta + alpha * \
            delta.grad.sign()

        # Проекция на epsilon-шар
        delta.data = torch.clamp(delta.data, -epsilon, epsilon)

        # Обнулить градиенты
        delta.grad.zero_()

    # Финальный adversarial example
    x_adv = torch.clamp(x + delta, 0, 1)
    return x_adv
```

### ◆ 5. C&W (Carlini & Wagner) Attack

Оптимизационный подход для поиска minimal perturbation.

**Формула:** minimize  $\|\delta\|_p + c \times f(x + \delta)$

```
def cw_attack(model, x, target, c=1.0,
              learning_rate=0.01, max_iter=1000):
    """
    C&W L2 атака - ищет минимальное возмущение
    """
    # Переменная для оптимизации
    delta = torch.zeros_like(x,
                            requires_grad=True)
    optimizer = torch.optim.Adam([delta],
                                 lr=learning_rate)

    for i in range(max_iter):
        # Adversarial example
        x_adv = x + delta

        # Logits модели
        logits = model(x_adv)

        # C&W loss function
        real = logits.gather(1,
                             target.unsqueeze(1))
        other = logits.gather(1,
                              logits.argsort(descending=True)[ :, 1:2])

        #  $f(x') = \max(0, \text{real} - \text{other} + \kappa)$ 
        f = torch.clamp(real - other + 0.0, min=0)

        # Total loss: L2 norm + classification
        loss = torch.norm(delta, p=2) + c * \
            f.sum()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return torch.clamp(x + delta, 0, 1)
```

## ◆ 6. DeepFool

Ищет closest decision boundary для минимального возмущения.

- Линеаризация границы решения
- Итеративное приближение
- Минимальное l2 возмущение

```
def deepfool_attack(model, x, num_classes=10,
                     max_iter=50, overshoot=0.02):
    """
    DeepFool атака - minimal perturbation
    """

    x_adv = x.clone().detach()
    x_adv.requires_grad = True

    # Исходный класс
    logits = model(x_adv)
    pred_class = logits.argmax().item()

    for i in range(max_iter):
        logits = model(x_adv)

        # Вычисляем градиенты для всех классов
        grads = []
        for k in range(num_classes):
            if k == pred_class:
                continue

            model.zero_grad()
            logits[0, k].backward(retain_graph=True)
            grads.append(x_adv.grad.clone())
            x_adv.grad.zero_()

        # Находим ближайшую границу
        min_dist = float('inf')
        best_grad = None

        for grad in grads:
            # Расстояние до boundary
            w_norm = torch.norm(grad.flatten())
            dist = abs(logits[0, pred_class] -
                       logits[0, k]) / w_norm

            if dist < min_dist:
                min_dist = dist
                best_grad = grad

        # Обновляем x_adv
        perturbation = (1 + overshoot) * min_dist
        * best_grad / torch.norm(best_grad)
        x_adv = x_adv + perturbation
```

```
# Проверяем изменение класса
if model(x_adv).argmax() != pred_class:
    break

return x_adv
```

## ◆ 7. Universal Adversarial Perturbations

Одно возмущение работает на многих примерах.

```
def universal_perturbation(model, dataloader,
                           epsilon=0.1,
                           max_iter=10):
    """
    Находит универсальное возмущение
    """

    # Инициализация
    v = torch.zeros(1, 3, 224, 224)

    for iteration in range(max_iter):
        print(f"Iteration {iteration+1}/{max_iter}")

        for batch_x, batch_y in dataloader:
            # Применяем текущее возмущение
            x_perturbed = batch_x + v

            # Проверяем предсказания
            pred_clean =
                model(batch_x).argmax(dim=1)
            pred_perturbed =
                model(x_perturbed).argmax(dim=1)

            # Для правильно классифицированных
            correct_mask = (pred_clean == batch_y)

            if correct_mask.sum() > 0:
                # Находим минимальное возмущение
                DeepFool
                    for i in range(len(batch_x)):
                        if correct_mask[i]:
                            delta = deepfool_attack(
                                model,
                                x_perturbed[i:i+1])
                            v += delta -
                                x_perturbed[i:i+1]

                # Проекция на epsilon-шар
                v = torch.clamp(v, -epsilon, epsilon)

    return v
```

## ◆ 8. Black-box атаки

Атаки без доступа к модели:

### 1. Transferability:

- Обучаем surrogate модель
- Генерируем атаку на surrogate
- Применяем к целевой модели

```
# Transfer attack
def transfer_attack(surrogate_model, target_model,
x, y):
    # Генерируем атаку на surrogate
    x_adv = pgd_attack(surrogate_model, x, y)

    # Проверяем на target
    pred_clean = target_model(x).argmax()
    pred_adv = target_model(x_adv).argmax()

    success = (pred_clean != pred_adv)
    return x_adv, success
```

### 2. Query-based атаки:

```
def query_based_attack(model_query_fn, x, y,
                       num_queries=1000):
    """
    Атака только с доступом к предсказаниям
    """
    best_x = x.clone()
    best_loss = float('inf')

    for i in range(num_queries):
        # Случайное возмущение
        noise = torch.randn_like(x) * 0.01
        x_perturbed = torch.clamp(x + noise, 0, 1)

        # Query модель
        pred = model_query_fn(x_perturbed)
        loss = F.cross_entropy(pred, y)

        # Сохраняем лучшее
        if loss < best_loss:
            best_loss = loss
            best_x = x_perturbed

    return best_x
```

## ◆ 9. Физические атаки

Adversarial examples в реальном мире:

- Adversarial patches:** стикеры на объектах
- 3D adversarial objects:** физические объекты
- Robust perturbations:** устойчивые к трансформациям

```
def adversarial_patch(model, target_class,
                      patch_size=50,
                      num_iter=1000):
    """
    Генерирует adversarial patch
    """
    # Инициализация патча
    patch = torch.rand(3, patch_size, patch_size,
                      requires_grad=True)
    optimizer = torch.optim.Adam([patch], lr=0.01)

    for i in range(num_iter):
        # Применяем патч к случайным изображениям
        images = get_random_images(batch_size=32)

        # Случайное размещение патча
        patched_images =
apply_patch_random(images, patch)

        # Loss для целевого класса
        logits = model(patched_images)
        target = torch.full((32,), target_class)
        loss = F.cross_entropy(logits, target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Клип патч к валидному диапазону
        patch.data = torch.clamp(patch.data, 0, 1)

    return patch
```

## ◆ 10. Метрики атак

| Метрика                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Описание                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| Attack Success Rate                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Доля успешных атак             |
| L $\infty$ distance                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Максимальное изменение пикселя |
| L2 distance                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Евклидово расстояние           |
| L0 distance                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Число измененных пикселей      |
| SSIM                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Структурное сходство           |
| Robustness                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Устойчивость к атакам          |
| <pre>def evaluate_attack(model, x_clean, x_adv, y_true):     """Оценка качества атаки"""      # Success rate     pred_clean = model(x_clean).argmax(dim=1)     pred_adv = model(x_adv).argmax(dim=1)     success = (pred_clean != pred_adv).float().mean()      # Perturbation metrics     l_inf = (x_adv - x_clean).abs().max()     l_2 = torch.norm(x_adv - x_clean, p=2)     l_0 = (x_adv != x_clean).float().sum()      return {         'success_rate': success.item(),         'l_inf': l_inf.item(),         'l_2': l_2.item(),         'l_0': l_0.item()     }</pre> |                                |

## ◆ 11. Защита: Adversarial Training

Обучение на adversarial examples:

```
def adversarial_training(model, train_loader,
                         epochs=10, epsilon=0.03):
    """
    Adversarial training для robustness
    """
    optimizer = torch.optim.Adam(model.parameters())

    for epoch in range(epochs):
        for x, y in train_loader:
            # Генерируем adversarial examples
            x_adv = pgd_attack(model, x, y,
                                epsilon=epsilon)

            # Смешиваем clean и adversarial
            x_mixed = torch.cat([x, x_adv], dim=0)
            y_mixed = torch.cat([y, y], dim=0)

            # Обычное обучение
            optimizer.zero_grad()
            output = model(x_mixed)
            loss = F.cross_entropy(output,
                                  y_mixed)
            loss.backward()
            optimizer.step()

            print(f"Epoch {epoch+1} completed")

    return model
```

## ◆ 12. Certified defenses

Гарантированная устойчивость:

- **Randomized smoothing**
- **Interval bound propagation**
- **Lipschitz constraints**

```
import torch.nn.functional as F

class RandomizedSmoothing:
    """Certified defense через случайный шум"""

    def __init__(self, model, sigma=0.1,
                 n_samples=100):
        self.model = model
        self.sigma = sigma
        self.n_samples = n_samples

    def predict(self, x):
        """Robust предсказание"""
        predictions = []

        # Множественные предсказания с шумом
        for _ in range(self.n_samples):
            noise = torch.randn_like(x) *
self.sigma
            x_noisy = x + noise
            pred = self.model(x_noisy).argmax()
            predictions.append(pred.item())

        # Возвращаем most common
        from collections import Counter
        return Counter(predictions).most_common(1)
[0][0]

    def certify(self, x, pred_class, alpha=0.001):
        """Вычисляем certified radius"""
        from scipy.stats import norm

        # Подсчитываем votes
        counts = [0] * 10
        for _ in range(self.n_samples):
            noise = torch.randn_like(x) *
self.sigma
            pred = self.model(x + noise).argmax()
            counts[pred.item()] += 1

        # Сертификат
        p_a = counts[pred_class] / self.n_samples

        if p_a > 0.5:
            radius = self.sigma * norm.ppf(p_a)
            return radius
```

```
else:
    return 0.0 # Not certified
```



# Adversarial Robustness

Январь 2026

## ◆ 1. Суть

- **Проблема:** ML модели уязвимы к adversarial примерам
- **Adversarial пример:** минимальное возмущение → неверное предсказание
- **Невидимость:** изменения незаметны для человека
- **Универсальность:** работает на разных моделях
- **Риск:** критично для безопасности (автопилот, медицина)

*Adversarial пример - специально созданный вход, вызывающий ошибку ML модели при малом возмущении*

## ◆ 2. FGSM атака

### Fast Gradient Sign Method:

```
x_adv = x + ε · sign(∇x L(θ, x, y))

где:
- x - оригинальное изображение
- ε - размер возмущения (обычно 0.01-0.3)
- L - функция потерь
- y - истинная метка

import torch
import torch.nn.functional as F

def fgsm_attack(model, x, y, epsilon=0.03):
    """
    FGSM атака на модель
    """
    x.requires_grad = True

    # Forward pass
    output = model(x)
    loss = F.cross_entropy(output, y)

    # Backward pass
    model.zero_grad()
    loss.backward()

    # Создание adversarial примера
    x_grad = x.grad.data
    x_adv = x + epsilon * x_grad.sign()

    # Обрезка до допустимого диапазона [0, 1]
    x_adv = torch.clamp(x_adv, 0, 1)

    return x_adv

# Использование
x_adv = fgsm_attack(model, images, labels,
                     epsilon=0.1)
output_adv = model(x_adv)
print(f"Точность на adversarial:
{output_adv.argmax(1).eq(labels).float().mean()}")
```

## ◆ 3. PGD атака

### Projected Gradient Descent:

```
def pgd_attack(model, x, y, epsilon=0.03,
               alpha=0.01, num_iter=40):
    """
    PGD - итеративная версия FGSM
    Сильнее и надежнее
    """
    x_adv = x.clone().detach()

    for i in range(num_iter):
        x_adv.requires_grad = True
        output = model(x_adv)
        loss = F.cross_entropy(output, y)

        model.zero_grad()
        loss.backward()

        # Шаг градиентного спуска
        with torch.no_grad():
            x_adv = x_adv + alpha *
            x_adv.grad.sign()

        # Проекция на epsilon-шар
        perturbation = torch.clamp(x_adv - x,
                                   -epsilon, epsilon)
        x_adv = torch.clamp(x + perturbation,
                           0, 1)

    return x_adv

# Использование
x_adv = pgd_attack(model, images, labels,
                    epsilon=0.1, alpha=0.01,
                    num_iter=40)
```

## ◆ 4. C&W атака

### Carlini & Wagner:

```
# Оптимизационная атака
minimize ||δ||_2^2 + c · f(x + δ)

где f - функция, максимизирующая вероятность
целевого класса
```

```
from cleverhans.torch.attacks import
carlini_wagner_l2

def cw_attack(model, x, y, c=1e-4, kappa=0,
max_iter=1000):
    """
    C&W L2 атака - одна из самых сильных
    """
    device = x.device
    batch_size = x.size(0)

    # Инициализация
    w = torch.zeros_like(x, requires_grad=True)
    optimizer = torch.optim.Adam([w], lr=0.01)

    for step in range(max_iter):
        # Преобразование w в x_adv
        x_adv = 0.5 * (torch.tanh(w) + 1)

        # Вычисление потерь
        output = model(x_adv)

        # L2 расстояние
        l2_dist = torch.sum((x_adv - x) ** 2, dim=[1, 2, 3])

        # Классификационная потеря
        target_logit = output[range(batch_size),
y]
        max_other_logit = torch.max(
            output - 1e-4 * F.one_hot(y,
output.size(1)),
            dim=1
        )[0]

        loss_cls = torch.clamp(max_other_logit -
target_logit + kappa, min=0)

        # Общая потеря
        loss = l2_dist + c * loss_cls

        optimizer.zero_grad()
        loss.mean().backward()
        optimizer.step()
```

```
return 0.5 * (torch.tanh(w) + 1)
```

## ◆ 5. DeepFool

```
def deepfool(model, x, max_iter=50):
    """
    DeepFool - находит минимальное возмущение
    для изменения предсказания
    """
    x_adv = x.clone().detach()
    pred = model(x_adv).argmax(1)

    for i in range(max_iter):
        x_adv.requires_grad = True
        output = model(x_adv)

        # Градиенты по всем классам
        gradients = []
        for c in range(output.size(1)):
            model.zero_grad()
            output[0,
c].backward(retain_graph=True)
            gradients.append(x_adv.grad.clone())

    # Находим минимальное возмущение
    pred_class = output.argmax(1).item()
    min_dist = float('inf')

    for k in range(output.size(1)):
        if k == pred_class:
            continue

        w = gradients[k] -
gradients[pred_class]
        f = output[0, k] - output[0,
pred_class]

        dist = abs(f.item()) /
torch.norm(w).item()

        if dist < min_dist:
            min_dist = dist
            best_w = w

    # Обновление
    r = (min_dist + 1e-4) * best_w /
torch.norm(best_w)
    x_adv = x_adv + r

    if model(x_adv).argmax(1) != pred:
        break

    return x_adv
```

## ◆ 6. Adversarial Training

### Основная защита:

```
def adversarial_training(model, train_loader,
optimizer,
                           epsilon=0.1,
num_epochs=10):
    """
        Обучение на adversarial примерах
    """
    for epoch in range(num_epochs):
        for images, labels in train_loader:
            # Генерация adversarial примеров
            x_adv = pgd_attack(model, images,
labels, epsilon=epsilon)

            # Обучение на смеси чистых и
adversarial
            images_mixed = torch.cat([images,
x_adv])
            labels_mixed = torch.cat([labels,
labels])

            optimizer.zero_grad()
            output = model(images_mixed)
            loss = F.cross_entropy(output,
labels_mixed)
            loss.backward()
            optimizer.step()

            print(f"Epoch {epoch+1} completed")

    return model

# Использование
model = adversarial_training(model, train_loader,
optimizer)
```

## ◆ 7. Методы защиты

| Метод                  | Идея                     | Эффективность |
|------------------------|--------------------------|---------------|
| Adversarial Training   | Обучение на атаках       | ★★★★★         |
| Input Transformation   | Сглаживание входа        | ★★            |
| Defensive Distillation | Смягчение выходов        | ★★★           |
| Randomization          | Случайные преобразования | ★★★           |
| Certified Defense      | Гарантии устойчивости    | ★★★★★         |
| Ensemble Methods       | Несколько моделей        | ★★★           |

## ◆ 8. Input Transformation

```
import torchvision.transforms as transforms

def input_transformation_defense(model, x):
    """
        Защита через преобразование входа
    """
    # Сжатие JPEG
    def jpeg_compression(x, quality=75):
        from PIL import Image
        import io

        buffer = io.BytesIO()
        img = transforms.ToPILImage()(x)
        img.save(buffer, format='JPEG',
quality=quality)
        buffer.seek(0)
        return transforms.ToTensor()(Image.open(buffer))

    # Медианный фильтр
    def median_filter(x, kernel_size=3):
        from scipy.ndimage import median_filter as
mf
        return torch.from_numpy(
            mf(x.cpu().numpy(), size=(1, 1,
kernel_size, kernel_size)))
    )

    # Применение преобразований
    x_defended = x.clone()
    for i in range(x.size(0)):
        x_defended[i] = jpeg_compression(x[i])

    return model(x_defended)

# Использование
output = input_transformation_defense(model,
x_adv)
```

## ◆ 9. Defensive Distillation

```
def defensive_distillation(teacher_model,
                           student_model,
                           train_loader,
                           temperature=20):
    """
        Обучение с повышенной temperature для
        устойчивости
    """
    # Шаг 1: Обучить teacher с высокой temperature
    for images, labels in train_loader:
        logits = teacher_model(images) /
        temperature
        soft_labels = F.softmax(logits, dim=1)

        # Сохранить soft labels

    # Шаг 2: Обучить student на soft labels
    optimizer =
    torch.optim.SGD(student_model.parameters(),
                    lr=0.01)

    for images, soft_labels in train_loader:
        logits = student_model(images) /
        temperature

        # Distillation loss
        loss = F.kl_div(
            F.log_softmax(logits, dim=1),
            soft_labels,
            reduction='batchmean'
        )

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return student_model
```

## ◆ 10. Certified Defense

### Randomized Smoothing:

```
def randomized_smoothing(model, x, sigma=0.25,
                           n_samples=100):
    """
        Сертифицированная защита через сглаживание
    """
    predictions = []

    for _ in range(n_samples):
        # Добавляем гауссовский шум
        noise = torch.randn_like(x) * sigma
        x_noisy = x + noise

        # Предсказание
        with torch.no_grad():
            pred = model(x_noisy).argmax(1)
            predictions.append(pred)

    # Голосование
    predictions = torch.stack(predictions)
    final_pred = predictions.mode(0)[0]

    return final_pred

# Сертифицированный радиус устойчивости
def certified_radius(sigma, prob_correct):
    """
        Вычисление радиуса, в котором предсказание
        гарантированно верно
    """
    from scipy.stats import norm
    return sigma * (norm.ppf(prob_correct) -
                    norm.ppf(0.5))
```

## ◆ 11. Оценка устойчивости

```
from foolbox import PyTorchModel, attacks

def evaluate_robustness(model, test_loader,
                        epsilon=0.3):
    """
        Оценка устойчивости к атакам
    """
    fmodel = PyTorchModel(model, bounds=(0, 1))

    # Различные атаки
    attack_methods = {
        'FGSM': attacks.FGSM(),
        'PGD': attacks.LinfPGD(),
        'C&W': attacks.L2CarliniWagnerAttack(),
        'DeepFool': attacks.LinfDeepFoolAttack()
    }

    results = {}

    for name, attack in attack_methods.items():
        correct = 0
        total = 0

        for images, labels in test_loader:
            _, x_adv, success = attack(fmodel,
   images, labels,
   epsilons=epsilon)

            pred_adv = model(x_adv).argmax(1)
            correct += (pred_adv ==
                        labels.sum().item())
            total += labels.size(0)

        accuracy = 100.0 * correct / total
        results[name] = accuracy
        print(f"{name}: {accuracy:.2f}%")

    return results
```

## ◆ 12. Лучшие практики

- **Adversarial Training:** самая эффективная защита
- **Разнообразие атак:** тестируйте на разных методах
- **Trade-off:** устойчивость vs точность
- **Размер epsilon:** выбирайте по задаче
- **Ensemble:** комбинируйте методы защиты
- **Мониторинг:** проверяйте в production

### ✓ Эффективно

- ✓ PGD adversarial training
- ✓ Certified randomized smoothing
- ✓ Ensemble моделей

### ✗ Слабо

- ✗ Gradient masking
- ✗ Input transformation только
- ✗ Obfuscated gradients



## Adversarial Training

 Январь 2026

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

### ◆ 1. Основы Adversarial Training

- **Определение:** ключевая концепция
- **Применение:** практическое использование
- **Преимущества:** почему это важно

# Affinity Propagation

17 Январь 2026

## 1. Суть метода

- Message passing:** точки обмениваются сообщениями о пригодности быть центром
- Автоматическое K:** сам определяет число кластеров и их центры (exemplars)
- Similarity-based:** работает с матрицей схожести между точками
- Детерминированный:** одинаковые результаты при фиксированных параметрах
- Основан на графах:** каждая точка может быть центром кластера

## 2. Базовый код

```
from sklearn.cluster import AffinityPropagation
import numpy as np

# Базовое использование
model = AffinityPropagation(
    damping=0.5,
    max_iter=200,
    random_state=42
)
labels = model.fit_predict(X)

# Получить exemplars (центры кластеров)
exemplars = model.cluster_centers_indices_
n_clusters = len(exemplars)

print(f"Найдено кластеров: {n_clusters}")
print(f"Индексы exemplars: {exemplars}")

# С предвычисленной матрицей схожести
from sklearn.metrics import pairwise_distances
S = -pairwise_distances(X, metric='euclidean')
model = AffinityPropagation(
    affinity='precomputed'
)
labels = model.fit_predict(S)
```

## 3. Ключевые параметры

| Параметр         | Описание                      | Совет                               |
|------------------|-------------------------------|-------------------------------------|
| damping          | Коэффициент затухания (0.5-1) | 0.5-0.9, больше = стабильнее        |
| max_iter         | Макс итераций                 | 200-500, увеличить если не сходится |
| convergence_iter | Итераций без изменений        | 15 по умолчанию                     |
| preference       | Предпочтение быть exemplar    | Медиана S или None (авто)           |
| affinity         | Метрика схожести              | 'euclidean', 'precomputed'          |

## ◆ 4. Алгоритм работы

1. **Инициализация:** построить матрицу схожести  $S(i,k)$  между точками
  2. **Responsibility:**  $R(i,k)$  — насколько  $k$  подходит быть exemplar для  $i$
  3. **Availability:**  $A(i,k)$  — насколько  $i$  должна выбрать  $k$  как exemplar
  4. **Итерации:** обновлять  $R$  и  $A$  с демпфированием
  5. **Выбор exemplars:** точки где  $R(k,k) + A(k,k) > 0$
  6. **Назначение кластеров:** каждая точка к ближайшему exemplar
- Формулы обновления:**
- $R(i,k) \leftarrow S(i,k) - \max\{A(i,k') + S(i,k')\}$  для  $k' \neq k$
  - $A(i,k) \leftarrow \min\{0, R(k,k) + \sum \max\{0, R(i',k)\}\}$  для  $i' \notin \{i,k\}$

## ◆ 5. Параметр Preference

**Что это:** начальное значение  $S(i,i)$  — самоподобие точки

**Влияние:**

- Больше preference → больше кластеров
- Меньше preference → меньше кластеров

**Выбор значения:**

```
# Автоматический (медиана)
preference = np.median(S)

# Или квантили для контроля
preference = np.percentile(S, 10) # меньше
                                   # кластеров
preference = np.percentile(S, 50) # средне
preference = np.percentile(S, 90) # больше
                                   # кластеров

model = AffinityPropagation(
    preference=preference
)
```

**Подбор через поиск:**

```
from sklearn.metrics import silhouette_score

S = -pairwise_distances(X, metric='euclidean')
prefs = np.percentile(S, [10, 30, 50, 70, 90])

best_score = -1
for pref in prefs:
    ap = AffinityPropagation(preference=pref)
    labels = ap.fit_predict(S)
    if len(np.unique(labels)) > 1:
        score = silhouette_score(X, labels)
        if score > best_score:
            best_score = score
            best_pref = pref
```

## ◆ 6. Матрица схожести

**Отрицательное квадратичное расстояние:**

```
# По умолчанию в sklearn
S(i,k) = -||x_i - x_k||^2

# Вручную
from sklearn.metrics import pairwise_distances
S = -pairwise_distances(X, metric='euclidean')**2
```

**Другие метрики:**

```
# Косинусная схожесть
from sklearn.metrics.pairwise import cosine_similarity
S = cosine_similarity(X)

# Корреляция
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler
X_scaled = StandardScaler().fit_transform(X)
S = cosine_similarity(X_scaled)

# Пользовательская
def custom_similarity(X):
    # Ваша логика
    return S

S = custom_similarity(X)
ap = AffinityPropagation(affinity='precomputed')
labels = ap.fit_predict(S)
```

## ◆ 7. Предобработка данных

### ✓ Масштабирование важно

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

### ✓ Снижение размерности

- AP работает с матрицей  $N \times N$  — память  $O(N^2)$
- PCA для больших данных перед AP
- Или использовать выборку точек

### ✓ Обработка выбросов

- Выбросы могут стать отдельными exemplars
- Предварительная фильтрация рекомендуется

## ◆ 8. Когда использовать

### ✓ Хорошо

- ✓ Неизвестное число кластеров
- ✓ Нужны представители (exemplars)
- ✓ Малые данные (<5K точек)
- ✓ Нестандартные метрики схожести
- ✓ Важна стабильность результатов

### ✗ Плохо

- ✗ Большие данные (>10K точек)
- ✗ Ограничения памяти ( $O(N^2)$ )
- ✗ Нужна быстрая работа
- ✗ Высокая размерность без PCA
- ✗ Много шума в данных

## ◆ 9. Проблемы и решения

| Проблема                | Решение                                        |
|-------------------------|------------------------------------------------|
| Не сходится             | Увеличить damping (0.5 → 0.9), max_iter        |
| Слишком много кластеров | Уменьшить preference                           |
| Слишком мало кластеров  | Увеличить preference                           |
| Медленная работа        | Уменьшить данные, PCA, sparse matrix           |
| Много памяти            | Выборка данных, использовать mini-batch подход |
| Осцилляции              | Увеличить damping, convergence_iter            |

## ◆ 10. Метрики оценки

```
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score
)

labels = model.fit_predict(X)
n_clusters = len(model.cluster_centers_indices_)

if n_clusters > 1:
    sil = silhouette_score(X, labels)
    db = davies_bouldin_score(X, labels)
    ch = calinski_harabasz_score(X, labels)

    print(f"Кластеров: {n_clusters}")
    print(f"Silhouette: {sil:.3f}")
    print(f"Davies-Bouldin: {db:.3f}")
    print(f"Calinski-Harabasz: {ch:.3f}")

# Exemplar quality
exemplar_indices = model.cluster_centers_indices_
exemplars = X[exemplar_indices]
print(f"Exemplars: {exemplar_indices}")
```

## ◆ 11. Визуализация

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 2D проекция
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)

# Точки
plt.scatter(X_2d[:, 0], X_2d[:, 1],
            c=labels, cmap='viridis',
            alpha=0.6, s=50)

# Exemplars
exemplar_indices = model.cluster_centers_indices_
exemplars_2d = X_2d[exemplar_indices]
plt.scatter(exemplars_2d[:, 0], exemplars_2d[:, 1],
            c='red', marker='*', s=300,
            edgecolors='black', linewidths=2,
            label='Exemplars')

# Связи с exemplars
for i, label in enumerate(labels):
    exemplar_idx = exemplar_indices[label]
    plt.plot([X_2d[i, 0], X_2d[exemplar_idx, 0]],
             [X_2d[i, 1], X_2d[exemplar_idx, 1]],
             'k-', alpha=0.1, linewidth=0.5)

plt.legend()
plt.title(f'AP: {n_clusters} кластеров')
plt.show()
```

## ◆ 12. Сравнение с другими методами

| Метод        | Преимущества AP               | Недостатки AP            |
|--------------|-------------------------------|--------------------------|
| K-means      | Авто K, exemplars             | Медленнее, больше памяти |
| Mean Shift   | Детерминированный, стабильнее | Медленнее                |
| DBSCAN       | Все точки в кластере          | Нужно знать примерное K  |
| Hierarchical | Плоская структура             | Медленнее                |

## ◆ 13. Применения

### Биоинформатика:

- Кластеризация генов по экспрессии
- Идентификация представительных последовательностей

### Computer Vision:

- Распознавание лиц (exemplar faces)
- Кластеризация изображений

### Тексты и NLP:

- Кластеризация документов
- Выбор представительных текстов

### Социальные сети:

- Обнаружение сообществ
- Выявление лидеров мнений (exemplars)

## ◆ 14. Продвинутые техники

### Sparse Affinity Propagation:

```
# Для больших данных
from sklearn.metrics import pairwise_distances
from scipy.sparse import csr_matrix

# Sparse similarity matrix
distances = pairwise_distances(X,
metric='euclidean')
# Обнулить далекие точки
threshold = np.percentile(distances, 90)
distances[distances > threshold] = 0
S_sparse = csr_matrix(-distances)

# Использовать как обычно
# (sklearn не поддерживает, нужна своя реализация)
```

### Hierarchical Affinity Propagation:

- Применить AP на exemplars для мета-кластеризации
- Создает иерархию кластеров

```
# Уровень 1
ap1 = AffinityPropagation()
labels1 = ap1.fit_predict(X)
exemplars1 = X[ap1.cluster_centers_indices_]

# Уровень 2 (мета)
ap2 = AffinityPropagation()
labels2 = ap2.fit_predict(exemplars1)
```

## ◆ 15. Чек-лист

- [ ] Масштабировать данные
- [ ] Проверить размер данных (<10K)
- [ ] Выбрать метрику схожести
- [ ] Подобрать preference (grid search)
- [ ] Установить damping (0.5-0.9)
- [ ] Проверить сходимость (увеличить max\_iter если нужно)
- [ ] Оценить качество кластеризации
- [ ] Проанализировать exemplars

### Объяснение заказчику:

«Affinity Propagation выбирает наиболее типичных представителей (exemplars) из ваших данных и группирует остальные точки вокруг них. Метод сам определяет оптимальное число групп, анализируя связи между всеми точками данных».

## ◆ 16. Полный пример

```
from sklearn.cluster import AffinityPropagation
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import numpy as np

# Подготовка
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Подбор preference
from sklearn.metrics import pairwise_distances
S = -pairwise_distances(X_scaled,
metric='euclidean')**2
preferences = np.percentile(S, [10, 30, 50, 70])

results = []
for pref in preferences:
    ap = AffinityPropagation(
        preference=pref,
        damping=0.7,
        max_iter=300,
        convergence_iter=15
    )
    labels = ap.fit_predict(X_scaled)
    n_clusters = len(ap.cluster_centers_indices_)

    if n_clusters > 1:
        score = silhouette_score(X_scaled, labels)
        results.append({
            'pref': pref,
            'n_clusters': n_clusters,
            'score': score,
            'model': ap
        })
# Лучший результат
best = max(results, key=lambda x: x['score'])
print(f"Preference: {best['pref']:.2f}")
print(f"Кластеров: {best['n_clusters']}")
print(f"Silhouette: {best['score']:.3f}")

# Exemplars
exemplar_idx =
best['model'].cluster_centers_indices_
print(f"Exemplar indices: {exemplar_idx}")
```



# Алгоритмический трейдинг

17 Январь 2026

## ◆ 1. Суть алгоритмического трейдинга

### Автоматизированная торговля на основе алгоритмов и ML

- **Ключевая концепция:** детали и примеры для суть алгоритмического трейдинга
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

Суть алгоритмического трейдинга — важный аспект для понимания темы

## ◆ 2. Типы стратегий

### Trend following, Mean reversion, Statistical arbitrage, Market making

- **Ключевая концепция:** детали и примеры для типы стратегий
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

Типы стратегий — важный аспект для понимания темы

## ◆ 3. Признаки (features)

**Price, Volume, Technical indicators, Sentiment, Macroeconomic**

- **Ключевая концепция:** детали и примеры для признаки (features)
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Признаки (features) — важный аспект для понимания темы*

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import
RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Алгоритмический трейдинг Cheatsheet — 3 колонки

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 4. Технические индикаторы

**Moving averages, RSI, MACD, Bollinger Bands, ATR**

- **Ключевая концепция:** детали и примеры для технические индикаторы
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Технические индикаторы — важный аспект для понимания темы*

## ◆ 5. ML модели

**Linear Regression, Random Forest, XGBoost, Neural Networks, Reinforcement Learning**

- **Ключевая концепция:** детали и примеры для ml модели
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *ML модели — важный аспект для понимания темы*

## ◆ 6. Прогнозирование

**Price prediction, Direction prediction, Volatility forecasting**

- **Ключевая концепция:** детали и примеры для прогнозирования
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 Прогнозирование — важный аспект для понимания темы

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import
RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Алгоритмический трейдинг Cheatsheet — 3 колонки

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 7. Reinforcement Learning

**Agent-based trading, Q-learning, Policy gradients**

- **Ключевая концепция:** детали и примеры для reinforcement learning
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 Reinforcement Learning — важный аспект для понимания темы

## ◆ 8. Backtesting

**Исторические данные, метрики (Sharpe ratio, max drawdown)**

- **Ключевая концепция:** детали и примеры для backtesting
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 Backtesting — важный аспект для понимания темы

## ◆ 9. Риск-менеджмент

### Position sizing, stop-loss, portfolio optimization

- Ключевая концепция:** детали и примеры для риск-менеджмент
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Риск-менеджмент — важный аспект для понимания темы

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import
RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

### Алгоритмический трейдинг Cheatsheet — 3 колонки

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 10. Проблемы

### Overfitting, look-ahead bias, transaction costs, slippage

- Ключевая концепция:** детали и примеры для проблемы
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Проблемы — важный аспект для понимания темы

## ◆ 11. High-frequency trading

### Latency, order book dynamics

- Ключевая концепция:** детали и примеры для high-frequency trading
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 High-frequency trading — важный аспект для понимания темы

## ◆ 12. Практический пример

### Simple momentum strategy с ML

- Ключевая концепция:** детали и примеры для практический пример
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Практический пример — важный аспект для понимания темы

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']
```

```
# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Обучение модели
from sklearn.ensemble import
RandomForestClassifier
```

```
model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)
```

```
# Оценка
from sklearn.metrics import accuracy_score,
classification_report
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

# ⚡ ALS (Alternating Least Squares)

17 Январь 2026

## ◆ 1. Основы ALS

- **Цель:** матричная факторизация для рекомендаций
- **Идея:** чередующаяся оптимизация двух матриц
- **Преимущество:** параллелизация, хорошо для больших данных
- **Применение:** колаборативная фильтрация
- **R ≈ U × V^T:** user factors × item factors

## ◆ 2. Алгоритм

### Итеративный процесс:

1. Инициализировать U и V случайно
2. Зафиксировать V, оптимизировать U
3. Зафиксировать U, оптимизировать V
4. Повторять шаги 2-3 до сходимости

### Функция потерь:

$$L = \sum (r_{ij} - u_i^T v_j)^2 + \lambda (\|U\|^2 + \|V\|^2)$$

### ◆ 3. Реализация с нуля

```

import numpy as np

class ALS:
    def __init__(self, n_factors=10,
                 n_iterations=10,
                 reg_param=0.01):
        self.n_factors = n_factors
        self.n_iterations = n_iterations
        self.reg_param = reg_param

    def fit(self, R):
        self.n_users, self.n_items = R.shape

        # Инициализация
        self.U = np.random.rand(self.n_users,
                               self.n_factors)
        self.V = np.random.rand(self.n_items,
                               self.n_factors)

        # Мaska известных рейтингов
        self.mask = (R > 0).astype(float)

        for iteration in range(self.n_iterations):
            # Фиксируем V, обновляем U
            for i in range(self.n_users):
                # Известные рейтинги пользователя
                items_rated = np.where(R[i] > 0)

                if len(items_rated) == 0:
                    continue

                V_i = self.V[items_rated]
                R_i = R[i, items_rated]

                # Closed-form solution
                A = V_i.T @ V_i + \
                    self.reg_param *
                    np.eye(self.n_factors)
                b = V_i.T @ R_i
                self.U[i] = np.linalg.solve(A, b)

            # Фиксируем U, обновляем V
            for j in range(self.n_items):
                # Пользователи, оценившие товар j
                users_rated = np.where(R[:, j] >
                                      0)[0]

                if len(users_rated) == 0:
                    continue

                U_j = self.U[users_rated]
                R_j = R[users_rated, j]

                A = U_j.T @ U_j + \
                    self.reg_param *

```

## ALS (Alternating Least Squares) Cheatsheet — 3 колонки

```

np.eye(self.n_factors)
    b = U_j.T @ R_j
    self.V[j] = np.linalg.solve(A, b)

    # RMSE
    if iteration % 2 == 0:
        rmse = self.compute_rmse(R)
        print(f"Iteration {iteration}: {rmse:.4f}")

    return self

def predict(self, user, item):
    return np.dot(self.U[user], self.V[item])

def compute_rmse(self, R):
    predictions = self.U @ self.V.T
    errors = (R - predictions) * self.mask
    return np.sqrt(np.sum(errors ** 2) /
                  np.sum(self.mask))

# Использование
R = np.array([
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [1, 0, 0, 4],
    [0, 1, 5, 4],
])

als = ALS(n_factors=2, n_iterations=10,
          reg_param=0.1)
als.fit(R)

# Предсказание
print(f"Predicted rating: {als.predict(0, 2):.2f}")

```

### ◆ 4. Implicit ratings

```

import implicit
from scipy.sparse import csr_matrix

# Данные: user-item interactions
# 0 = не взаимодействовал, положительное =
# взаимодействовал
data = csr_matrix([
    [1, 1, 0, 0, 1],
    [1, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 1, 1, 1]
])

# ALS модель
model = implicit.als.AlternatingLeastSquares(
    factors=50,
    iterations=10,
    regularization=0.01,
    use_gpu=False
)

# Обучение (транспонируем для item-user)
model.fit(data.T)

# Рекомендации для пользователя 0
user_id = 0
recommendations = model.recommend(
    user_id,
    data[user_id],
    N=10
)

print("Recommendations:")
for item_id, score in recommendations:
    print(f"Item {item_id}: {score:.4f}")

```

## ◆ 5. PySpark ALS

```
from pyspark.ml.recommendation import ALS
from pyspark.sql import SparkSession

# Spark session
spark =
SparkSession.builder.appName("ALS").getOrCreate()

# Данные
data = spark.createDataFrame([
    (0, 0, 5.0),
    (0, 1, 3.0),
    (1, 0, 4.0),
    (1, 3, 1.0),
], ["userId", "itemId", "rating"])

# ALS модель
als = ALS(
    maxIter=10,
    regParam=0.01,
    userCol="userId",
    itemCol="itemId",
    ratingCol="rating",
    coldStartStrategy="drop",
    nonnegative=True
)

# Обучение
model = als.fit(data)

# Предсказания
predictions = model.transform(data)
predictions.show()

# Рекомендации для всех пользователей
user_recs = model.recommendForAllUsers(10)
user_recs.show(truncate=False)

# Рекомендации для всех товаров
item_recs = model.recommendForAllItems(10)
item_recs.show(truncate=False)
```

## ◆ 6. Weighted ALS

Для неявных отзывов с весами уверенности:

```
# Weighted ALS
def weighted_als(R, C, n_factors=10, n_iter=10,
                 lambda_reg=0.01):
    """
    R: матрица предпочтений (0/1)
    C: матрица уверенности (веса)
    """
    n_users, n_items = R.shape

    # Инициализация
    U = np.random.rand(n_users, n_factors) * 0.01
    V = np.random.rand(n_items, n_factors) * 0.01

    for _ in range(n_iter):
        # Update U
        for u in range(n_users):
            Cu = np.diag(C[u])
            A = V.T @ Cu @ V + \
                lambda_reg * np.eye(n_factors)
            b = V.T @ Cu @ R[u]
            U[u] = np.linalg.solve(A, b)

        # Update V
        for i in range(n_items):
            Ci = np.diag(C[:, i])
            A = U.T @ Ci @ U + \
                lambda_reg * np.eye(n_factors)
            b = U.T @ Ci @ R[:, i]
            V[i] = np.linalg.solve(A, b)

    return U, V

# Пример
R_implicit = (R > 0).astype(float) # бинарные
C_implicit = 1 + 40 * R_implicit # веса
уверенности

U, V = weighted_als(R_implicit, C_implicit)
predictions = U @ V.T
```

## ◆ 7. Параметры

| Параметр       | Описание                   | Рекомендации |
|----------------|----------------------------|--------------|
| factors        | Число латентных факторов   | 10-100       |
| iterations     | Число итераций             | 10-20        |
| regularization | L2 регуляризация           | 0.01-0.1     |
| alpha          | Вес уверенности (implicit) | 40           |

## ◆ 8. Оценка качества

```
from sklearn.model_selection import
train_test_split

# Разбивка данных
train_data, test_data = train_test_split(
    ratings_df, test_size=0.2, random_state=42
)

# Обучение
als = ALS(n_factors=50)
als.fit(train_data)

# Оценка на тесте
def evaluate(model, test_data):
    predictions = []
    actuals = []

    for _, row in test_data.iterrows():
        user = row['userId']
        item = row['itemId']
        rating = row['rating']

        pred = model.predict(user, item)
        predictions.append(pred)
        actuals.append(rating)

    rmse = np.sqrt(np.mean(
        (np.array(predictions) -
        np.array(actuals))**2
    ))
    return rmse

rmse = evaluate(als, test_data)
print(f"Test RMSE: {rmse:.4f}")
```

## ◆ 9. Bias terms

```
# ALS с bias
class BiasedALS(ALS):
    def fit(self, R):
        # Глобальное среднее
        self.global_mean = np.mean(R[R > 0])

        # User biases
        self.user_bias = np.zeros(self.n_users)
        for i in range(self.n_users):
            ratings = R[i][R[i] > 0]
            if len(ratings) > 0:
                self.user_bias[i] =
                    np.mean(ratings) - \
                    self.global_mean

        # Item biases
        self.item_bias = np.zeros(self.n_items)
        for j in range(self.n_items):
            ratings = R[:, j][R[:, j] > 0]
            if len(ratings) > 0:
                self.item_bias[j] =
                    np.mean(ratings) - \
                    self.global_mean

        # Вычитаем biases из рейтингов
        R_centered = R.copy()
        for i in range(self.n_users):
            for j in range(self.n_items):
                if R[i, j] > 0:
                    R_centered[i, j] -=
                        (self.global_mean +
                         self.user_bias[i] +
                         self.item_bias[j])

        # Стандартный ALS на центрированных данных
        super().fit(R_centered)

        return self

    def predict(self, user, item):
        base_pred = super().predict(user, item)
        return (base_pred + self.global_mean +
               self.user_bias[user] +
               self.item_bias[item])
```

## ◆ 10. Параллелизация

```
from joblib import Parallel, delayed

class ParallelALS(ALS):
    def __init__(self, n_jobs=-1, **kwargs):
        super().__init__(**kwargs)
        self.n_jobs = n_jobs

    def _update_user(self, u, R):
        items_rated = np.where(R[u] > 0)[0]
        if len(items_rated) == 0:
            return self.U[u]

        V_u = self.V[items_rated]
        R_u = R[u, items_rated]

        A = V_u.T @ V_u + \
            self.reg_param * \
            np.eye(self.n_factors)
        b = V_u.T @ R_u
        return np.linalg.solve(A, b)

    def fit(self, R):
        # ... инициализация ...

        for iteration in range(self.n_iterations):
            # Параллельное обновление U
            self.U = np.array(
                Parallel(n_jobs=self.n_jobs)(
                    delayed(self._update_user)(u,
  R)
                    for u in range(self.n_users)
                )
            )

            # Параллельное обновление V
            # (аналогично)
            # ...
```

## ◆ 11. Сравнение: ALS vs SGD

| Аспект         | ALS             | SGD              |
|----------------|-----------------|------------------|
| Скорость       | Параллелизуется | Последовательный |
| Память         | Больше          | Меньше           |
| Сходимость     | Медленнее       | Быстрее          |
| Implicit data  | Отлично         | Хорошо           |
| Большие данные | Отлично (Spark) | Сложнее          |

## ◆ 12. Когда использовать

### ✓ Хорошо

- ✓ Большие датасеты
- ✓ Неявные отзывы (implicit feedback)
- ✓ Нужна параллелизация
- ✓ Spark/распределенные вычисления

### ✗ Плохо

- ✗ Малые данные (SGD быстрее)
- ✗ Онлайн-обучение (нужна перетренировка)
- ✗ Нужны bias terms (сложнее в ALS)

### ◆ 13. Чек-лист

- [ ] Подготовить user-item матрицу
- [ ] Выбрать число факторов
- [ ] Настроить регуляризацию
- [ ] Определить implicit/explicit
- [ ] Установить веса уверенности (implicit)
- [ ] Выбрать число итераций
- [ ] Оценить RMSE на валидации
- [ ] Рассмотреть параллелизацию

### 💡 Объяснение заказчику:

«ALS — это эффективный алгоритм для рекомендаций, который чередует оптимизацию факторов пользователей и товаров. Его главное преимущество — возможность быстро обрабатывать миллионы пользователей и товаров на кластере компьютеров. Идеально подходит для крупных e-commerce платформ».



# Дисперсионный анализ (ANOVA)

 Январь 2026

## ◆ 1. Основы ANOVA

- **Цель:** сравнение средних значений между группами
- **Гипотезы:**  $H_0$ : все средние равны vs  $H_1$ : хотя бы одно отличается
- **Идея:** сравнение внутригрупповой и межгрупповой вариации
- **F-статистика:** отношение дисперсий
- **Применение:** A/B/C тестирование, эксперименты

**Простыми словами:** ANOVA проверяет, отличаются ли средние значения в разных группах больше, чем можно объяснить случайным разбросом.

## ◆ 2. One-Way ANOVA

Сравнение средних для одного фактора с несколькими уровнями:

```
import numpy as np
from scipy import stats

# Данные трех групп
group1 = np.array([23, 25, 27, 29, 31])
group2 = np.array([33, 35, 37, 39, 41])
group3 = np.array([43, 45, 47, 49, 51])

# One-way ANOVA
f_stat, p_value = stats.f_oneway(group1, group2,
group3)

print(f"F-статистика: {f_stat:.4f}")
print(f"p-value: {p_value:.4f}")

if p_value < 0.05:
    print("Отвергаем  $H_0$ : группы отличаются")
else:
    print("Не отвергаем  $H_0$ : группы не отличаются")
```

### ◆ 3. Ручной расчет ANOVA

```
def one_way_anova_manual(groups):
    """Ручной расчет One-Way ANOVA"""
    k = len(groups) # число групп
    n = sum(len(g) for g in groups) # общее число наблюдений

    # Общее среднее
    grand_mean = np.mean([x for g in groups for x in g])

    # SST (Total Sum of Squares)
    sst = sum((x - grand_mean)**2
              for g in groups for x in g)

    # SSB (Between-group Sum of Squares)
    ssb = sum(len(g) * (np.mean(g) -
                         grand_mean)**2
              for g in groups)

    # SSW (Within-group Sum of Squares)
    ssw = sum((x - np.mean(g))**2
              for g in groups for x in g)

    # Degrees of freedom
    df_between = k - 1
    df_within = n - k

    # Mean squares
    msb = ssb / df_between
    msw = ssw / df_within

    # F-statistic
    f_stat = msb / msw

    # p-value
    p_value = 1 - stats.f.cdf(f_stat, df_between,
                               df_within)

    return {
        'F': f_stat,
        'p_value': p_value,
        'SSB': ssb,
        'SSW': ssw,
        'SST': sst,
        'df_between': df_between,
        'df_within': df_within
    }

# Использование
groups = [group1, group2, group3]
results = one_way_anova_manual(groups)
print(f"F = {results['F']:.4f}, p = {results['p_value']:.4f}")
```

### ◆ 4. ANOVA таблица

| Источник       | SS  | df  | MS              | F           |
|----------------|-----|-----|-----------------|-------------|
| Между группами | SSB | k-1 | MSB = SSB/(k-1) | F = MSB/MSW |
| Внутри групп   | SSW | n-k | MSW = SSW/(n-k) | -           |
| Всего          | SST | n-1 | -               | -           |

```
import pandas as pd

def anova_table(groups):
    results = one_way_anova_manual(groups)

    table = pd.DataFrame({
        'Source': ['Between', 'Within', 'Total'],
        'SS': [results['SSB'], results['SSW'],
               results['SST']],
        'df': [results['df_between'],
               results['df_within'],
               results['df_between'] +
               results['df_within']],
        'MS':
            [results['SSB']/results['df_between'],
             results['SSW']/results['df_within'],
             np.nan],
        'F': [results['F'], np.nan, np.nan],
        'p-value': [results['p_value'], np.nan,
                    np.nan]
    })
    return table

print(anova_table(groups))
```

### ◆ 5. Предположения ANOVA

- Нормальность:** данные в каждой группе нормально распределены
- Гомоскедастичность:** равные дисперсии в группах
- Независимость:** наблюдения независимы

```
# Проверка нормальности (Shapiro-Wilk)
for i, group in enumerate(groups, 1):
    stat, p = stats.shapiro(group)
    print(f"Group {i}: p={p:.4f}",
          "\u2295 Normal" if p > 0.05 else "x Not normal")

# Проверка равенства дисперсий (Levene's test)
stat, p = stats.levene(*groups)
print(f"\nLevene's test: p={p:.4f}",
      "\u2295 Equal var" if p > 0.05 else "x Unequal var")

# Если предположения нарушены, используйте:
# - Welch's ANOVA (неравные дисперсии)
# - Kruskal-Wallis (непараметрический)
```

## ◆ 6. Post-hoc тесты

После значимого ANOVA — попарные сравнения:

```
from scipy.stats import tukey_hsd

# Tukey HSD (Honestly Significant Difference)
result = tukey_hsd(*groups)
print("Tukey HSD:")
print(result)

# Bonferroni correction (ручной)
from itertools import combinations

def bonferroni_test(groups, alpha=0.05):
    k = len(groups)
    n_comparisons = k * (k - 1) // 2
    adjusted_alpha = alpha / n_comparisons

    print(f"Adjusted alpha:
{adjusted_alpha:.4f}\n")

    for (i, g1), (j, g2) in
combinations(enumerate(groups), 2):
        t_stat, p = stats.ttest_ind(g1, g2)
        sig = "/" if p < adjusted_alpha else "x"
        print(f"Group {i} vs {j}: p={p:.4f}
{sig}")

bonferroni_test(groups)
```

## ◆ 7. Two-Way ANOVA

Два фактора и их взаимодействие:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Данные
data = pd.DataFrame({
    'score': [23, 25, 27, 33, 35, 37,
              43, 45, 47, 53, 55, 57],
    'treatment': ['A', 'A', 'A', 'B', 'B', 'B',
                  'A', 'A', 'A', 'B', 'B', 'B'],
    'gender': ['M', 'M', 'M', 'M', 'M', 'M',
               'F', 'F', 'F', 'F', 'F', 'F']
})

# Two-way ANOVA
model = ols('score ~ C(treatment) + C(gender) +
            C(treatment):C(gender)', data=data).fit()

anova_results = sm.stats.anova_lm(model, typ=2)
print(anova_results)
```

## ◆ 9. Эффект размера

```
def eta_squared(groups):
    """Eta-squared: доля объясненной дисперсии"""
    results = one_way_anova_manual(groups)
    return results['SSB'] / results['SST']

def omega_squared(groups):
    """Omega-squared: несмещенная оценка"""
    results = one_way_anova_manual(groups)
    k = len(groups)
    n = sum(len(g) for g in groups)

    numerator = results['SSB'] - (k - 1) *
    results['SSW'] / (n - k)
    denominator = results['SST'] + results['SSW'] /
    (n - k)

    return numerator / denominator

eta2 = eta_squared(groups)
omega2 = omega_squared(groups)

print(f"\u03b7\u00b2 = {eta2:.4f}")
print(f"\u03c9\u00b2 = {omega2:.4f}")

# Интерпретация: 0.01=small, 0.06=medium,
0.14=large
```

## ◆ 8. Repeated Measures ANOVA

Для зависимых выборок (одни и те же субъекты):

```
from statsmodels.stats.anova import AnovaRM

# Данные: несколько измерений одних субъектов
data = pd.DataFrame({
    'subject': [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4],
    'time': ['T1', 'T2', 'T3'] * 4,
    'score': [23, 25, 27, 33, 35, 37,
              43, 45, 47, 53, 55, 57]
})

# Repeated measures ANOVA
aovrm = AnovaRM(data, 'score', 'subject',
                 within=['time'])
res = aovrm.fit()
print(res)
```

## ◆ 10. Непараметрические альтернативы

```
# Kruskal-Wallis (непараметрический аналог ANOVA)
h_stat, p_value = stats.kruskal(*groups)
print(f"Kruskal-Wallis H: {h_stat:.4f}, p=
{p_value:.4f}")

# Mann-Whitney U для попарных сравнений
from scipy.stats import mannwhitneyu

for i in range(len(groups)):
    for j in range(i+1, len(groups)):
        u_stat, p = mannwhitneyu(groups[i],
        groups[j])
        print(f"Group {i} vs {j}: U={u_stat:.0f},
p={p:.4f}")
```

## ◆ 11. Визуализация

```
import matplotlib.pyplot as plt
import seaborn as sns

# Boxplot
data_long = pd.DataFrame({
    'value': np.concatenate(groups),
    'group': np.repeat(['Group 1', 'Group 2',
    'Group 3'], [len(g) for g in groups])
})

plt.figure(figsize=(10, 6))
sns.boxplot(x='group', y='value', data=data_long)
plt.title('Сравнение групп')
plt.ylabel('значение')
plt.show()

# Violin plot с точками
plt.figure(figsize=(10, 6))
sns.violinplot(x='group', y='value',
data=data_long)
sns.swarmplot(x='group', y='value',
data=data_long,
color='black', alpha=0.5)
plt.title('Распределение по группам')
plt.show()
```

## ◆ 12. Когда использовать

### ✓ Хорошо

- ✓ Сравнение 3+ групп
- ✓ Нормально распределенные данные
- ✓ Равные дисперсии
- ✓ A/B/C тестирование
- ✓ Экспериментальные данные

### ✗ Плохо

- ✗ Две группы (используйте t-test)
- ✗ Нарушение предположений (используйте Kruskal-Wallis)
- ✗ Неравные дисперсии (используйте Welch ANOVA)
- ✗ Сильные выбросы

## ◆ 13. Чек-лист

- [ ] Проверить нормальность (Shapiro-Wilk)
- [ ] Проверить равенство дисперсий (Levene)
- [ ] Провести ANOVA
- [ ] Если значимо — post-hoc тесты
- [ ] Рассчитать размер эффекта
- [ ] Визуализировать результаты
- [ ] Интерпретировать в контексте задачи
- [ ] Проверить на выбросы

### 💡 Объяснение заказчику:

«ANOVA помогает понять, действительно ли разные группы (например, разные версии продукта, маркетинговые кампании) дают разные результаты, или наблюдаемые различия можно объяснить случайностью. Это статистически строгий способ сравнения нескольких вариантов одновременно».



# Модели для распознавания речи (ASR)

4 января 2026

## ◆ 1. Суть ASR

- **Цель:** преобразовать аудио в текст
- **Входные данные:** wav-файлы или спектрограммы
- **Выходные данные:** последовательность слов/символов
- **Метрики:** WER (Word Error Rate), CER (Character Error Rate)

## ◆ 2. Классические подходы

- **HMM-GMM:** скрытые марковские модели + гауссовые смеси
- **Kaldi:** open-source toolkit для классического ASR
- **Проблемы:** требуют feature engineering, фонетических словарей

## ◆ 3. Deep Learning подходы

- **DeepSpeech:** end-to-end RNN + CTC loss
- **Listen, Attend and Spell:** encoder-decoder с attention
- **Conformer:** convolution + transformer
- **Wav2Vec 2.0:** self-supervised предобучение

## ◆ 4. Препроцессинг данных

- Нормализация амплитуды
- Ресэмплирование (обычно 16кГц или 22кГц)
- Удаление тишины в начале/конце
- Аугментация: pitch shift, time stretch, добавление шума

## ◆ 5. Базовый код (библиотека)

- `import librosa` — обработка аудио
- `torchaudio` — PyTorch для аудио
- `tensorflow_io` — TensorFlow аудио
- Работа с форматами: WAV, MP3, FLAC

## ◆ 6. Метрики качества

- **WER** (Word Error Rate) — процент ошибок слов
- **CER** (Character Error Rate) — процент ошибок символов
- **BLEU** — для оценки точности транскрипции
- **Real-time factor** — скорость распознавания

## ◆ 7. Когда использовать

- Транскрипция речи в текст
- Голосовые ассистенты
- Субтитры для видео
- Нужна генерация речи (→ TTS)

## ◆ 8. Популярные датасеты

- **LibriSpeech** — аудиокниги, чистая речь
- **Common Voice** — краудсорсинг, многоязычный
- **TIMIT** — фонетически размеченная речь
- **Switchboard** — разговорная телефонная речь



# Астрофизика и космология с ML

17 Январь 2026

## ◆ 1. Введение

### Основные концепции и применения

- Темная материя и темная энергия
- Космологические параметры
- Large scale structure
- ML для теоретической астрофизики
- Simulation-based inference

## ◆ 2. Темная материя

### Основные концепции и применения

- N-body симуляции
- Weak gravitational lensing
- Субструктуры гало
- CNN для поиска DM сигналов
- Indirect detection через ML

## ◆ 3. Гравитационное линзирование

### Основные концепции и применения

- Strong lensing детекция
- Weak lensing shear maps
- Измерение массы скоплений
- ResNet для классификации линз
- Cosmological constraints

## ◆ 4. Космологические параметры

### Основные концепции и применения

- $\Omega_m$ ,  $\Omega_\Lambda$ ,  $H_0$ ,  $\sigma_8$
- Байесовский инференс
- Simulation-based inference (SBI)
- Neural posterior estimation
- MCMC vs нейросети

## ◆ 5. Реионизация

### Основные концепции и применения

- 21cm космология
- Первые звезды и галактики
- 3D CNN для 21cm кубов
- Epoch of Reionization
- SKA и будущие наблюдения

## ◆ 6. Cosmic Microwave Background

### Основные концепции и применения

- Температурные флуктуации CMB
- Polarization E и B modes
- Foreground cleaning с U-Net
- Космологические параметры из CMB
- Planck и будущие миссии

## ◆ 7. Барионные акустические осцилляции

### Основные концепции и применения

- BAO как стандартная линейка
- Измерение  $H(z)$  и  $D_A(z)$
- Power spectrum анализ
- ML для BAO scale determination
- Dark energy constraints

## ◆ 8. Supernovae Type Ia

### Основные концепции и применения

- Стандартные свечи
- Классификация типов SN
- RNN для кривых блеска
- Distance modulus измерение
- Dark energy equation of state

## ◆ 9. Формирование структур

### Основные концепции и применения

- Гравитационный коллапс
- Halo finding с GNN
- Galaxy formation physics
- Эмуляторы симулаций
- Large scale structure evolution

## ◆ 10. Будущее

### Основные концепции и применения

- LSST/Rubin Observatory
- Euclid mission
- JWST ранняя Вселенная
- ML для real-time processing
- Petabyte-scale data challenges

# 🎯 Attention Mechanism

17 Январь 2026

## ◆ 1. Суть Attention

- **Проблема RNN:** сложно запоминать длинные последовательности
- **Идея:** модель "обращает внимание" на важные части входа
- **Механизм:** взвешенная сумма входов
- **Веса:** вычисляются динамически
- **Применение:** NLP, CV, Speech
- **Основа:** Transformers (BERT, GPT)

## ◆ 2. Математика (упрощённо)

### Query, Key, Value:

- **Query (Q):** что ищем
- **Key (K):** с чем сравниваем
- **Value (V):** что извлекаем

### Формула:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{dk}) \cdot V$$

## ◆ 3. Базовая реализация

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ScaledDotProductAttention(nn.Module):
    def __init__(self, d_k):
        super().__init__()
        self.d_k = d_k

    def forward(self, Q, K, V,
mask=None):
        # Q, K, V: (batch, seq_len, d_k)

        # 1. Dot product
        scores = torch.matmul(Q,
K.transpose(-2, -1))

        # 2. Scale
        scores = scores / (self.d_k ** 0.5)

        # 3. Mask (optional)
        if mask is not None:
            scores =
scores.masked_fill(mask == 0, -1e9)

        # 4. Softmax
        attention_weights =
F.softmax(scores, dim=-1)

        # 5. Multiply by V
        output =
torch.matmul(attention_weights, V)

        return output, attention_weights
```

## ◆ 4. Self-Attention

### Каждый элемент внимания к остальным

```
class SelfAttention(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.d_model = d_model

        # Проекции для Q, K, V
        self.W_q = nn.Linear(d_model,
d_model)
        self.W_k = nn.Linear(d_model,
d_model)
        self.W_v = nn.Linear(d_model,
d_model)

        self.attention =
ScaledDotProductAttention(d_model)

    def forward(self, x):
        # x: (batch, seq_len, d_model)

        Q = self.W_q(x)
        K = self.W_k(x)
        V = self.W_v(x)

        output, weights =
self.attention(Q, K, V)

        return output, weights
```

## ◆ 5. Multi-Head Attention

### Несколько attention параллельно

```
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model,
                 num_heads):
        super().__init__()
        assert d_model % num_heads == 0

        self.d_model = d_model
        self.num_heads = num_heads
        self.d_k = d_model // num_heads

        self.W_q = nn.Linear(d_model,
                             d_model)
        self.W_k = nn.Linear(d_model,
                             d_model)
        self.W_v = nn.Linear(d_model,
                             d_model)
        self.W_o = nn.Linear(d_model,
                             d_model)

    def split_heads(self, x):
        batch_size = x.size(0)
        x = x.view(batch_size, -1,
                   self.num_heads, self.d_k)
        return x.transpose(1, 2)

    def forward(self, Q, K, V,
               mask=None):
        Q =
        self.split_heads(self.W_q(Q))
        K =
        self.split_heads(self.W_k(K))
        V =
        self.split_heads(self.W_v(V))

        # Attention для каждой головы
        scores = torch.matmul(Q,
                              K.transpose(-2, -1)) / (self.d_k ** 0.5)

        if mask is not None:
            scores =
            scores.masked_fill(mask == 0, -1e9)

        attention = F.softmax(scores,
                              dim=-1)
        output = torch.matmul(attention,
```

V)

```
# Concat heads
output = output.transpose(1,
                         2).contiguous()
output =
output.view(output.size(0), -1,
            self.d_model)

return self.W_o(output)
```

## ◆ 7. Masked Attention (GPT)

```
# Маска для causal attention
def create_causal_mask(seq_len):
    mask =
    torch.triu(torch.ones(seq_len, seq_len),
               diagonal=1)
    mask = mask.masked_fill(mask == 1, -
                           float('inf'))
    return mask

# Применение
mask = create_causal_mask(seq_len)
scores = scores + mask # Broadcasting
```

## ◆ 6. Типы Attention

| Тип                    | Описание                      | Применение                   |
|------------------------|-------------------------------|------------------------------|
| <b>Self-Attention</b>  | Элементы внимают друг к другу | Transformers, BERT           |
| <b>Cross-Attention</b> | Две разные последовательности | Encoder-Decoder, Translation |
| <b>Causal/Masked</b>   | Внимание только на прошлое    | GPT, языковые модели         |
| <b>Global</b>          | Внимание ко всем элементам    | Большинство Transformers     |
| <b>Local</b>           | Только к ближайшим            | Эффективные Transformers     |

## ◆ 8. Позиционное кодирование

**Attention не учитывает порядок — добавляем позиции**

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model,
                 max_len=5000):
        super().__init__()

        pe = torch.zeros(max_len,
                         d_model)
        position = torch.arange(0,
                               max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) *
            -(math.log(10000.0) /
              d_model)
        )

        pe[:, 0::2] = torch.sin(position *
                                div_term)
        pe[:, 1::2] = torch.cos(position *
                                div_term)

        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(1), :]
        return x
```

## ◆ 9. Преимущества Attention

- **Параллелизация:** в отличие от RNN
- **Длинные зависимости:** любая позиция к любой
- **Интерпретируемость:** веса attention показывают важность
- **Гибкость:** работает для разных модальностей
- **Масштабируемость:** эффективные варианты для длинных последовательностей

## ◆ 10. Применения

- **NLP:** BERT, GPT, T5 — все на Transformers
- **Machine Translation:** оригинальное применение
- **Computer Vision:** Vision Transformers (ViT)
- **Speech Recognition:** Whisper, Wav2Vec2
- **Multimodal:** CLIP (изображения + текст)

## ◆ 11. Cross-Attention в Encoder-Decoder

```
# Encoder output → Keys & Values
# Decoder state → Query

class EncoderDecoderAttention(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.mha =
            MultiHeadAttention(d_model, num_heads=8)

    def forward(self, decoder_state,
               encoder_output):
        # Query from decoder
        Q = decoder_state

        # Keys and Values from encoder
        K = encoder_output
        V = encoder_output

        output = self.mha(Q, K, V)
        return output
```

## ◆ 12. Эффективные варианты

- **Linear Attention:**  $O(n)$  вместо  $O(n^2)$
- **Sparse Attention:** внимание к подмножеству
- **Longformer:** sliding window + global
- **Reformer:** LSH attention
- **Performer:** kernel approximation

## ◆ 13. Лучшие практики

- **Масштабирование:** делить на  $\sqrt{d_k}$  обязательно!
- **Multi-head:** 8 или 16 голов стандарт
- **Dropout:** на attention weights (0.1)
- **Residual:** Add & Norm после attention
- **Позиции:** добавлять positional encoding

## ◆ 14. Чек-лист

- [ ] Реализовать Q, K, V проекции
- [ ] Масштабировать scores на  $\sqrt{d_k}$
- [ ] Применить softmax
- [ ] Для decoder — использовать маску
- [ ] Multi-head для лучшего качества
- [ ] Добавить positional encoding
- [ ] Residual connections + LayerNorm
- [ ] Dropout на attention weights

«Attention — это механизм выборочного внимания: модель смотрит на весь текст и решает, какие слова важны для понимания текущего слова. Как когда мы читаем — фокусируемся на ключевых словах, а не на каждом одинаково».

# Классификация звуков (Audio Classification)

 4 января 2026

## ◆ 1. Суть

- **Цель:** классифицировать аудио по категориям
- **Примеры задач:** распознавание музыки, детекция событий, эмоции в речи
- **Входные данные:** raw audio или признаки (MFCC, спектrogramma)
- **Метрики:** accuracy, F1, confusion matrix

## ◆ 2. Признаки (Features)

- **MFCC:** Mel-Frequency Cepstral Coefficients
- **Спектrogramma:** визуализация частот во времени
- **Mel-спектrogramma:** логарифмическая шкала частот
- **Chroma:** представление высоты тона

## ◆ 3. Модели

- **Классические:** SVM, Random Forest на признаках
- **CNN:** 2D свертки на спектrogramмах
- **RNN/LSTM:** для временных зависимостей
- **Transformer:** Audio Spectrogram Transformer (AST)

## ◆ 4. Препроцессинг данных

- Нормализация амплитуды
- Ресэмплирование (обычно 16кГц или 22кГц)
- Удаление тишины в начале/конце
- Аугментация: pitch shift, time stretch, добавление шума

## ◆ 5. Базовый код (библиотека)

- `import librosa` — обработка аудио
- `torchaudio` — PyTorch для аудио
- `tensorflow_io` — TensorFlow аудио
- Работа с форматами: WAV, MP3, FLAC

## ◆ 6. Метрики качества

- Precision, Recall, F1-score
- Confusion Matrix
- ROC-AUC для бинарной классификации
- Top-k accuracy для множественных классов

## ◆ 7. Когда использовать

-  Есть размеченные аудиоданные
-  Задача классификации звуков
-  Требуется генерация (→ TTS, audio generation)
-  Нужно понимание содержания речи (→ ASR + NLP)

## ◆ 8. Популярные датасеты

- AudioSet — 2M клипов, 527 классов
- ESC-50 — environmental sounds
- UrbanSound8K — городские звуки
- Speech Commands — команды для голосового управления

# 🎵 Audio Generation

17 Январь 2026

## ◆ 1. Суть

- **Задача:** создание аудио с нуля или на основе входных данных
- **Типы генерации:** музыка, речь, звуковые эффекты
- **Подходы:** autoregressive, diffusion, GAN-based
- **Представление:** raw waveform или спектrogramма

## ◆ 2. Основные подходы

| Подход            | Описание                   | Примеры                    |
|-------------------|----------------------------|----------------------------|
| Autoregressive    | Генерация сэмпл за сэмплом | WaveNet, SampleRNN         |
| VAE-based         | Вариационные автоэнкодеры  | MusicVAE, Jukebox          |
| GAN-based         | Состязательное обучение    | WaveGAN, MelGAN            |
| Diffusion         | Диффузионные модели        | DiffWave, WaveGrad         |
| Transformer-based | Attention механизмы        | Music Transformer, MuseNet |

## ◆ 3. WaveNet — базовый пример

```
import torch
import torch.nn as nn

class WaveNetLayer(nn.Module):
    def __init__(self, in_channels, out_channels,
                 kernel_size, dilation):
        super().__init__()
        self.conv = nn.Conv1d(
            in_channels, out_channels,
            kernel_size, dilation=dilation,
            padding=dilation*(kernel_size-1)//2
        )
        self.gate_conv = nn.Conv1d(
            in_channels, out_channels,
            kernel_size, dilation=dilation,
            padding=dilation*(kernel_size-1)//2
        )

    def forward(self, x):
        tanh_out = torch.tanh(self.conv(x))
        sigmoid_out =
            torch.sigmoid(self.gate_conv(x))
        return tanh_out * sigmoid_out
```

## ◆ 4. Представления аудио

### Raw Waveform

- Прямое представление сигнала (16-48 кГц)
- Высокое разрешение, но медленная генерация
- Используется в WaveNet, WaveGlow

### Mel-спектрограмма

- Частотно-временное представление
- Быстрее генерировать, требует вокодера
- Используется в Tacotron 2, FastSpeech

```
import librosa
# Преобразование в mel-спектрограмму
mel_spec = librosa.feature.melspectrogram(
    y=audio, sr=22050, n_mels=80
)
log_mel_spec = librosa.power_to_db(mel_spec)
```

## ◆ 5. Генерация музыки

| Модель   | Особенности                         |
|----------|-------------------------------------|
| MusicVAE | Интерполяция мелодий, вариации      |
| Jukebox  | Генерация музыки с вокалом (OpenAI) |
| MuseNet  | Multi-instrument, разные стили      |
| MusicGen | Text-to-music (Meta)                |
| AudioLDM | Diffusion для аудио из текста       |

```
# Пример с MusicGen (Hugging Face)
from transformers import AutoProcessor,
MusicgenForConditionalGeneration

processor =
AutoProcessor.from_pretrained("facebook/musicgen-small")
model =
MusicgenForConditionalGeneration.from_pretrained(
    "facebook/musicgen-small"
)

inputs = processor(
    text=["upbeat electronic music"],
    padding=True,
    return_tensors="pt"
)
audio_values = model.generate(**inputs,
max_new_tokens=256)
```

## ◆ 6. Вокодеры

**Роль:** преобразование mel-спектрограмм в waveform

| Вокодер     | Скорость     | Качество |
|-------------|--------------|----------|
| Griffin-Lim | Быстро       | Среднее  |
| WaveGlow    | Средне       | Высокое  |
| HiFi-GAN    | Быстро       | Отличное |
| MelGAN      | Очень быстро | Хорошее  |

```
# Пример использования HiFi-GAN
from hifigan import Generator

vocoder = Generator()
vocoder.load_state_dict(torch.load('hifigan.pth'))
vocoder.eval()

with torch.no_grad():
    audio = vocoder(mel_spectrogram)
```

## ◆ 8. Условная генерация

| Тип условия | Применение                     |
|-------------|--------------------------------|
| Текст       | Text-to-audio, text-to-music   |
| Класс       | Жанр музыки, тип звука         |
| Мелодия     | Гармонизация, аранжировка      |
| Ритм        | Drum patterns, beat generation |
| Эмоция      | Эмоциональная музыка           |

```
# Conditioning пример
class ConditionalGenerator(nn.Module):
    def __init__(self, condition_dim, audio_dim):
        super().__init__()
        self.cond_proj = nn.Linear(condition_dim,
256)
        self.audio_net = WaveNetModel()

    def forward(self, noise, condition):
        cond_embed = self.cond_proj(condition)
        # Добавляем условие на каждом слое
        return self.audio_net(noise, cond_embed)
```

## ◆ 7. Diffusion модели для аудио

**Принцип:** постепенное удаление шума

- **DiffWave:** диффузия на raw waveform
- **WaveGrad:** градиентная диффузия
- **Riffusion:** Stable Diffusion для спектрограмм
- **AudioLDM:** latent diffusion для аудио

**Преимущества:**

- Высокое качество
- Стабильное обучение
- Контроль через conditioning

## ◆ 9. Оценка качества

| Метрика | Что измеряет                                  |
|---------|-----------------------------------------------|
| FAD     | Fréchet Audio Distance (схожесть с реальными) |
| IS      | Inception Score (разнообразие и качество)     |
| MOS     | Mean Opinion Score (субъективная оценка)      |
| PESQ    | Качество речи                                 |
| STOI    | Разборчивость речи                            |

## ◆ 10. Практические советы

### ✓ Рекомендуется

- ✓ Использовать pre-trained модели
- ✓ Mel-спектrogramмы для быстрой генерации
- ✓ HiFi-GAN для качественного вокодинга
- ✓ Diffusion для высокого качества
- ✓ Data augmentation (pitch shift, time stretch)

### ✗ Избегать

- ✗ Генерация raw waveform без GPU
- ✗ Обучение с нуля без большого датасета
- ✗ Игнорирование нормализации аудио
- ✗ Слишком низкий sampling rate (<16 кГц)

## ◆ 12. Примеры применения

- Создание фоновой музыки** для видео, игр
- Генерация звуковых эффектов** по описанию
- Музыкальное сопровождение** рекламы
- Персонализированная музыка** для пользователей
- Аудио для виртуальных миров**
- Помощь музыкантам** в создании черновиков

### 💡 Объяснение заказчику:

*«Нейронная сеть учится на тысячах музыкальных композиций, чтобы затем создавать новую музыку по вашему текстовому описанию или заданному стилю. Как художник, который изучил разные стили и теперь может создать картину в любом из них».*

## ◆ 13. Чек-лист

- [ ] Выбрать тип генерации (музыка/звуки/речь)
- [ ] Определить представление (waveform/mel-spec)
- [ ] Подобрать модель под задачу
- [ ] Подготовить датасет (качественный аудио)
- [ ] Настроить preprocessing (нормализация, sample rate)
- [ ] Выбрать вокодер (если работаем с mel-spec)
- [ ] Определить метрики качества
- [ ] Провести A/B тестирование с пользователями

## ◆ 11. Библиотеки и инструменты

```
# Установка основных библиотек
pip install torch torchaudio
pip install librosa
pip install transformers
pip install diffusers

# Для работы с MIDI
pip install pretty_midi mido

# Для обработки аудио
pip install soundfile scipy
```

### Популярные фреймворки:

- Magenta:** музыкальная генерация (Google)
- AudioCraft:** MusicGen, AudioGen (Meta)
- Stability Audio:** Stable Audio

# 🎵 Audio Processing: Spectrograms & MFCC

 Январь 2026

## ◆ 1. Основы аудио

- **Звуковая волна:** изменение давления во времени
- **Sample Rate (SR):** количество отсчётов в секунду (Hz)
- **Nyquist частота:** SR/2 (максимальная частота)
- **Обычные SR:** 16 kHz (речь), 44.1 kHz (музыка)
- **Amplitude:** громкость сигнала
- **Frequency:** высота звука (Hz)

## ◆ 2. Загрузка аудио

```
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Загрузка аудиофайла
audio, sr = librosa.load('audio.wav', sr=22050)

# Информация
print(f"Sample rate: {sr} Hz")
print(f"Длительность: {len(audio)/sr:.2f} сек")
print(f"Shape: {audio.shape}")

# Визуализация waveform
plt.figure(figsize=(12, 4))
librosa.display.waveshow(audio, sr=sr)
plt.title('Waveform')
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.show()
```

## ◆ 3. Spectrogram (Спектрограмма)

**Идея:** визуализация частотного содержания во времени

- **Ось X:** время
- **Ось Y:** частота
- **Цвет/Яркость:** интенсивность (амплитуда/ мощность)
- **Метод:** STFT (Short-Time Fourier Transform)

```
# Вычисление спектрограммы
D = librosa.stft(audio, n_fft=2048,
hop_length=512)
magnitude = np.abs(D)
power = magnitude**2

# Преобразование в дБ
spectrogram_db =
librosa.amplitude_to_db(magnitude, ref=np.max)

# Визуализация
plt.figure(figsize=(12, 4))
librosa.display.specshow(
    spectrogram_db,
    sr=sr,
    hop_length=512,
    x_axis='time',
    y_axis='hz'
)
plt.colorbar(format='%.2f dB')
plt.title('Спектрограмма')
plt.show()
```

## ◆ 4. Mel-Spectrogram

**Mel-шкала:** шкала частот, близкая к восприятию человека

- Нелинейная шкала частот
- Больше деталей на низких частотах
- Меньше на высоких (где слух менее чувствителен)
- Стандарт для распознавания речи

```
# Mel-спектрограмма
mel_spec = librosa.feature.melspectrogram(
    y=audio,
    sr=sr,
    n_fft=2048,
    hop_length=512,
    n_mels=128, # количество mel-полос
    fmax=8000 # максимальная частота
)

# В дБ
mel_spec_db = librosa.power_to_db(mel_spec,
ref=np.max)

# Визуализация
plt.figure(figsize=(12, 4))
librosa.display.specshow(
    mel_spec_db,
    sr=sr,
    hop_length=512,
    x_axis='time',
    y_axis='mel'
)
plt.colorbar(format='%.2f dB')
plt.title('Mel-Спектрограмма')
plt.show()
```

## ◆ 5. MFCC (Mel-Frequency Cepstral Coefficients)

**Цель:** компактное представление спектра звука

- **Шаги:**

1. Вычислить mel-спектрограмму
2. Взять логарифм
3. Применить DCT (Discrete Cosine Transform)
4. Взять первые N коэффициентов

- **Обычно:** 13-40 коэффициентов

- **Первый коэффициент:** средняя энергия (часто удаляют)

```
# Вычисление MFCC
mfccs = librosa.feature.mfcc(
    y=audio,
    sr=sr,
    n_mfcc=13,      # количество коэффициентов
    n_fft=2048,
    hop_length=512
)

# Визуализация
plt.figure(figsize=(12, 4))
librosa.display.specshow(
    mfccs,
    sr=sr,
    hop_length=512,
    x_axis='time'
)
plt.colorbar()
plt.title('MFCC')
plt.ylabel('MFCC коэффициенты')
plt.show()

# Статистики для ML
mfcc_mean = np.mean(mfccs, axis=1)
mfcc_std = np.std(mfccs, axis=1)
features = np.concatenate([mfcc_mean, mfcc_std])
```

## ◆ 6. Дополнительные признаки

| Признак                   | Описание                                                                              | Применение         |
|---------------------------|---------------------------------------------------------------------------------------|--------------------|
| <b>Zero Crossing Rate</b> | Частота смены знака                                                                   | Шумы vs музыка     |
| <b>Spectral Centroid</b>  | Центр масс спектра                                                                    | Яркость звука      |
| <b>Spectral Rolloff</b>   | Частота ниже которой 85% энергии                                                      | Тембр              |
| <b>Chroma</b>             | 12 pitch classes                                                                      | Музикальный анализ |
| <b>Tempo</b>              | Темп в BPM                                                                            | Ритм               |
|                           | # Zero Crossing Rate<br>zcr = librosa.feature.zero_crossing_rate(audio)               |                    |
|                           | # Spectral Centroid<br>spec_cent = librosa.feature.spectral_centroid(y=audio, sr=sr)  |                    |
|                           | # Spectral Rolloff<br>spec_rolloff = librosa.feature.spectral_rolloff(y=audio, sr=sr) |                    |
|                           | # Chroma features<br>chroma = librosa.feature.chroma_stft(y=audio, sr=sr)             |                    |
|                           | # Tempo<br>tempo, beats = librosa.beat.beat_track(y=audio, sr=sr)                     |                    |
|                           | print(f"Tempo: {tempo:.2f} BPM")                                                      |                    |

## ◆ 7. Предобработка аудио

```
# Удаление тишины
audio_trimmed, index = librosa.effects.trim(
    audio,
    top_db=20 # порог в дБ
)

# Нормализация
audio_norm = librosa.util.normalize(audio)

# Изменение sample rate
audio_resampled = librosa.resample(
    audio,
    orig_sr=sr,
    target_sr=16000
)

# Добавление шума (augmentation)
noise = np.random.randn(len(audio))
audio_noisy = audio + 0.005 * noise

# Изменение высоты тона
audio_pitched = librosa.effects.pitch_shift(
    audio,
    sr=sr,
    n_steps=2 # полутона
)

# Изменение скорости
audio_stretched = librosa.effects.time_stretch(
    audio,
    rate=1.2 # в 1.2 раза быстрее
)
```

## ◆ 8. Извлечение признаков для ML

```
def extract_features(audio_path):
    """Извлечь все признаки из аудио"""
    # Загрузка
    audio, sr = librosa.load(audio_path, sr=22050)

    # MFCC
    mfccs = librosa.feature.mfcc(y=audio, sr=sr,
                                 n_mfcc=13)
    mfcc_mean = np.mean(mfccs, axis=1)
    mfcc_std = np.std(mfccs, axis=1)

    # Spectral features
    spec_cent =
        librosa.feature.spectral_centroid(y=audio, sr=sr)
    spec_cent_mean = np.mean(spec_cent)

    zcr =
        librosa.feature.zero_crossing_rate(audio)
    zcr_mean = np.mean(zcr)

    # Chroma
    chroma = librosa.feature.chroma_stft(y=audio,
   sr=sr)
    chroma_mean = np.mean(chroma, axis=1)

    # Объединение всех признаков
    features = np.concatenate([
        mfcc_mean,
        mfcc_std,
        [spec_cent_mean],
        [zcr_mean],
        chroma_mean
    ])

    return features

# Использование
features = extract_features('audio.wav')
print(f"Feature vector shape: {features.shape}")
```

## ◆ 9. Data Augmentation

- **Добавление шума:** white noise, background noise
- **Pitch shifting:** изменение высоты тона
- **Time stretching:** изменение скорости
- **Time shifting:** сдвиг во времени
- **Dynamic range compression:** нормализация громкости

```
import audiomentations as A

# Библиотека для аугментации
augmenter = A.Compose([
    A.AddGaussianNoise(min_amplitude=0.001,
                        max_amplitude=0.015, p=0.5),
    A.TimeStretch(min_rate=0.8, max_rate=1.2,
                  p=0.5),
    A.PitchShift(min_semitones=-4,
                  max_semitones=4, p=0.5),
    A.Shift(min_fraction=-0.5, max_fraction=0.5,
            p=0.5),
])

# Применение
augmented_audio = augmenter(samples=audio,
                             sample_rate=sr)
```

## ◆ 10. Применения

### ✓ Распознавание речи (ASR)

- ✓ MFCC + RNN/Transformer
- ✓ Mel-спектограмма + CNN
- ✓ wav2vec 2.0, Whisper

### ✓ Классификация звуков

- ✓ Музыкальные жанры
- ✓ Эмоции в речи
- ✓ Детекция событий (сирена, выстрел)

### ✓ Синтез речи (TTS)

- ✓ Mel-спектограмма → Vocoder
- ✓ Tacotron 2, FastSpeech

### ✓ Музыкальный анализ

- ✓ Определение темпа, тональности
- ✓ Source separation

## ◆ 11. Параметры STFT

| Параметр   | Описание        | Типичные значения  |
|------------|-----------------|--------------------|
| n_fft      | Размер окна FFT | 512, 1024, 2048    |
| hop_length | Шаг окна        | n_fft/4 (512)      |
| win_length | Длина окна      | = n_fft            |
| window     | Оконная функция | 'hann' (по умолч.) |

**Частотное разрешение:** sr / n\_fft

**Временное разрешение:** hop\_length / sr

**Компромисс:** большое n\_fft → лучшее частотное разрешение, но хуже временное

## ◆ 12. PyTorch/TensorFlow пайплайн

```
import torch
import torchaudio
import torchaudio.transforms as T

# Загрузка
waveform, sample_rate =
torchaudio.load('audio.wav')

# Mel-спектrogramма трансформация
mel_transform = T.MelSpectrogram(
    sample_rate=sample_rate,
    n_fft=2048,
    hop_length=512,
    n_mels=128
)

# Применение
mel_spec = mel_transform(waveform)

# В дБ
power_to_db = T.AmplitudeToDB()
mel_spec_db = power_to_db(mel_spec)

# MFCC трансформация
mfcc_transform = T.MFCC(
    sample_rate=sample_rate,
    n_mfcc=13,
    melkwargs={
        'n_fft': 2048,
        'hop_length': 512,
        'n_mels': 128
    }
)

mfcc = mfcc_transform(waveform)

print(f"Mel-spec shape: {mel_spec.shape}")
print(f"MFCC shape: {mfcc.shape}")
```

## ◆ 13. Чек-лист

- [ ] Выбрать sample rate: 16 kHz для речи, 22-44 kHz для музыки
- [ ] Предобработка: удалить тишину, нормализовать
- [ ] Выбрать представление: MFCC (компактно), Mel-spec (для DL)
- [ ] Настроить параметры: n\_fft, hop\_length
- [ ] Аугментация: шум, pitch shift, time stretch
- [ ] Извлечь статистики: mean, std для MFCC
- [ ] Визуализировать: проверить качество данных

### 💡 Объяснение заказчику:

«Спектrogramма — это как "фотография" звука, показывающая какие частоты присутствуют в каждый момент времени. MFCC — это компактный набор чисел, описывающих характеристики звука, который хорошо подходит для распознавания речи и музыки».

# 🎯 Автоэнкодеры (Autoencoders)

 Январь 2026

## ◆ 1. Суть

- **Автоэнкодер** — нейросеть для обучения представлений
- **Архитектура:** Encoder → Bottleneck → Decoder
- **Цель:** воссоздать вход на выходе
- **Обучение:** без учителя (unsupervised)
- **Bottleneck:** сжатое представление данных

## ◆ 2. Архитектура

Input → Encoder → Latent Space (z) → Decoder → Output

| Компонент        | Функция                        |
|------------------|--------------------------------|
| Encoder          | Сжатие данных в latent space   |
| Latent space (z) | Компактное представление       |
| Decoder          | Восстановление из latent space |

Размерность latent space << размерность входа

## ◆ 3. Простой автоэнкодер (Keras)

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Параметры
input_dim = 784 # например, MNIST 28x28
encoding_dim = 32 # размер латентного пространства

# Encoder
encoder_input = layers.Input(shape=(input_dim,))
encoded = layers.Dense(128, activation='relu')(encoder_input)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

encoder = models.Model(encoder_input, encoded, name='encoder')

# Decoder
decoder_input = layers.Input(shape=(encoding_dim,))
decoded = layers.Dense(128, activation='relu')(decoder_input)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)

decoder = models.Model(decoder_input, decoded, name='decoder')

# Autoencoder
autoencoder_input = layers.Input(shape=(input_dim,))
encoded_output = encoder(autoencoder_input)
decoded_output = decoder(encoded_output)

autoencoder = models.Model(
    autoencoder_input,
    decoded_output,
    name='autoencoder'
)

autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')
autoencoder.summary()
```

## ◆ 4. Простой автоэнкодер (PyTorch)

```
import torch
import torch.nn as nn

class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, encoding_dim),
            nn.ReLU()
        )

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 128),
            nn.ReLU(),
            nn.Linear(128, input_dim),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    def encode(self, x):
        return self.encoder(x)

    def decode(self, z):
        return self.decoder(z)

    # Создание модели
model = Autoencoder(input_dim=784,
encoding_dim=32)
```

## ◆ 5. Convolutional Autoencoder

```
# Для изображений
from tensorflow.keras import layers, models

def build_conv_autoencoder(input_shape):
    # Encoder
    encoder_input =
    layers.Input(shape=input_shape)
    x = layers.Conv2D(32, (3, 3),
activation='relu', padding='same')(encoder_input)
    x = layers.MaxPooling2D((2, 2),
padding='same')(x)
    x = layers.Conv2D(64, (3, 3),
activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2),
padding='same')(x)
    encoded = layers.Conv2D(128, (3, 3),
activation='relu', padding='same')(x)

    encoder = models.Model(encoder_input, encoded,
name='encoder')

    # Decoder
    decoder_input =
    layers.Input(shape=encoded.shape[1:])
    x = layers.Conv2D(128, (3, 3),
activation='relu', padding='same')(decoder_input)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3),
activation='relu', padding='same')(x)
    x = layers.UpSampling2D((2, 2))(x)
    decoded = layers.Conv2D(input_shape[-1], (3,
3),
activation='sigmoid',
padding='same')(x)

    decoder = models.Model(decoder_input, decoded,
name='decoder')

    # Autoencoder
    autoencoder_input =
    layers.Input(shape=input_shape)
    autoencoder = models.Model(
        autoencoder_input,
        decoder(encoder(autoencoder_input)))
    )

    return encoder, decoder, autoencoder

encoder, decoder, autoencoder =
build_conv_autoencoder((28, 28, 1))
autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')
```

## ◆ 6. Обучение автоэнкодера

```
# Keras
from tensorflow.keras.callbacks import
EarlyStopping

# Подготовка данных (например, MNIST)
from tensorflow.keras.datasets import mnist
(x_train, _), (x_test, _) = mnist.load_data()

# Нормализация
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten для обычного АЕ
x_train = x_train.reshape((len(x_train), -1))
x_test = x_test.reshape((len(x_test), -1))

# Обучение
history = autoencoder.fit(
    x_train, x_train, # вход = выход!
    epochs=50,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test, x_test),
    callbacks=[EarlyStopping(patience=5)])
)

# Предсказание
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

## ◆ 7. Типы автоэнкодеров

| Тип                          | Особенность              | Применение       |
|------------------------------|--------------------------|------------------|
| <b>Vanilla</b>               | Простой MLP              | Базовое сжатие   |
| <b>Convolutional</b>         | CNN слои                 | Изображения      |
| <b>Sparse</b>                | Разреженная активация    | Feature learning |
| <b>Denoising</b>             | Шум на входе             | Шумоподавление   |
| <b>Variational<br/>(VAE)</b> | Вероятностный            | Генерация        |
| <b>Contractive</b>           | Регуляризация градиентов | Робастность      |

## ◆ 8. Sparse Autoencoder

```
# Добавление L1 регуляризации для разреженности
from tensorflow.keras import regularizers

encoded = layers.Dense(
    encoding_dim,
    activation='relu',
    activity_regularizer=regularizers.l1(1e-5)
)(encoder_input)

# Или KL-дивергенция для sparsity
from tensorflow.keras import backend as K

def kl_divergence_regularizer(rho=0.05):
    def loss(y_true, y_pred):
        rho_hat = K.mean(y_pred, axis=0)
        kl = rho * K.log(rho / rho_hat) + \
            (1 - rho) * K.log((1 - rho) / (1 -
rho_hat))
        return K.sum(kl)
    return loss

# rho — желаемая средняя активация (обычно 0.05)
```

## ◆ 9. Denoising Autoencoder

```
import numpy as np

# Добавление шума к данным
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(
    loc=0.0, scale=1.0, size=x_train.shape
)
x_test_noisy = x_test + noise_factor * np.random.normal(
    loc=0.0, scale=1.0, size=x_test.shape
)

# Clip значения в [0, 1]
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Обучение: вход — зашумлённые данные, выход — чистые
autoencoder.fit(
    x_train_noisy, x_train,
    epochs=50,
    batch_size=256,
    validation_data=(x_test_noisy, x_test)
)

# Теперь модель умеет убирать шум!
```

## ◆ 10. Variational Autoencoder (VAE)

```
from tensorflow.keras import backend as K

class Sampling(layers.Layer):
    """Reparameterization trick"""
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = K.shape(z_mean)[0]
        dim = K.shape(z_mean)[1]
        epsilon = K.random_normal(shape=(batch, dim))
        return z_mean + K.exp(0.5 * z_log_var) * epsilon

    # Encoder
    encoder_input = layers.Input(shape=(input_dim,))
    x = layers.Dense(128, activation='relu')(encoder_input)
    z_mean = layers.Dense(latent_dim)(x)
    z_log_var = layers.Dense(latent_dim)(x)
    z = Sampling()([z_mean, z_log_var])

    encoder = models.Model(encoder_input, [z_mean, z_log_var, z])

    # Decoder
    decoder_input = layers.Input(shape=(latent_dim,))
    x = layers.Dense(128, activation='relu')(decoder_input)
    decoder_output = layers.Dense(input_dim,
                                  activation='sigmoid')(x)

    decoder = models.Model(decoder_input, decoder_output)

    # VAE loss
    def vae_loss(x, x_decoded):
        # Reconstruction loss
        reconstruction_loss = K.binary_crossentropy(x, x_decoded)
        reconstruction_loss *= input_dim

        # KL divergence
        kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
        kl_loss = K.sum(kl_loss, axis=-1)
        kl_loss *= -0.5

        return K.mean(reconstruction_loss + kl_loss)
```

## ◆ 11. Применения автоэнкодеров

- Снижение размерности** — альтернатива PCA
- Шумоподавление** — denoising autoencoder
- Обнаружение аномалий** — высокая ошибка реконструкции
- Генерация данных** — VAE
- Предобучение** — transfer learning
- Feature extraction** — использовать encoder
- Сжатие данных** — lossy compression

## ◆ 12. Обнаружение аномалий

```
# Обучить на нормальных данных
autoencoder.fit(x_train_normal, x_train_normal,
epochs=50)

# Вычислить ошибку реконструкции
reconstructed = autoencoder.predict(x_test)
mse = np.mean(np.square(x_test - reconstructed),
axis=1)

# Установить порог
threshold = np.percentile(mse, 95)

# Аномалии — это данные с высокой ошибкой
anomalies = mse > threshold

print(f"Обнаружено аномалий: {anomalies.sum()}")

# Визуализация
import matplotlib.pyplot as plt
plt.hist(mse, bins=50)
plt.axvline(threshold, color='r', linestyle='--',
label='Threshold')
plt.xlabel('Reconstruction Error')
plt.ylabel('Count')
plt.legend()
plt.show()
```

## ◆ 13. Визуализация латентного пространства

```
import matplotlib.pyplot as plt

# Для 2D латентного пространства
encoding_dim = 2
# ... обучить autoencoder ...

# Закодировать тестовые данные
encoded_imgs = encoder.predict(x_test)

# Визуализация
plt.figure(figsize=(10, 8))
plt.scatter(encoded_imgs[:, 0], encoded_imgs[:, 1],
           c=y_test, cmap='viridis', alpha=0.5)
plt.colorbar()
plt.xlabel('Latent dimension 1')
plt.ylabel('Latent dimension 2')
plt.title('Latent Space Visualization')
plt.show()

# Для размерности > 2 использовать t-SNE или UMAP
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=42)
encoded_2d = tsne.fit_transform(encoded_imgs)

plt.scatter(encoded_2d[:, 0], encoded_2d[:, 1],
           c=y_test, cmap='viridis', alpha=0.5)
plt.colorbar()
plt.show()
```

## ◆ 14. Генерация новых данных (VAE)

```
# Сэмплировать из латентного пространства
import numpy as np

# Случайные точки из стандартного нормального распределения
n_samples = 10
random_latent_vectors = np.random.normal(size=(n_samples, latent_dim))

# Декодировать
generated_images =
decoder.predict(random_latent_vectors)

# Визуализация
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(20, 2))
for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.imshow(generated_images[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()

# Интерполяция между двумя точками
def interpolate(encoder, decoder, img1, img2,
steps=10):
    z1 = encoder.predict(img1.reshape(1, -1))[0]
    # mean для VAE
    z2 = encoder.predict(img2.reshape(1, -1))[0]

    interpolated = []
    for alpha in np.linspace(0, 1, steps):
        z = (1 - alpha) * z1 + alpha * z2
        interpolated.append(decoder.predict(z.reshape(1, -1)))

interpolated
```

## ◆ 15. Когда использовать

### ✓ Хорошо

- ✓ Снижение размерности для визуализации
- ✓ Обнаружение аномалий
- ✓ Шумоподавление
- ✓ Feature extraction
- ✓ Генерация данных (VAE)
- ✓ Предобучение для других задач

### ✗ Плохо / Есть лучше

- ✗ Простое сжатие → используйте PCA
- ✗ Классификация → используйте supervised learning
- ✗ Реалистичная генерация → используйте GAN

## ◆ 16. Оптимизация автоэнкодера

```
# 1. Архитектура
# - Симметричный encoder/decoder
# - Постепенное уменьшение/увеличение размерности

# 2. Функция потерь
# Binary crossentropy для [0,1]
autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')

# MSE для других
autoencoder.compile(optimizer='adam', loss='mse')

# 3. Регуляризация
from tensorflow.keras import regularizers
layers.Dense(128, activation='relu',
            kernel_regularizer=regularizers.l2(1e-4))

# 4. Batch Normalization
from tensorflow.keras.layers import
BatchNormalization
x = layers.Dense(128)(x)
x = BatchNormalization()(x)
x = layers.Activation('relu')(x)

# 5. Learning rate scheduling
from tensorflow.keras.callbacks import
ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.5, patience=5)

history = autoencoder.fit(
    x_train, x_train,
    epochs=100,
    callbacks=[reduce_lr]
)
```

## ◆ 17. Метрики качества

```
# 1. Reconstruction Loss
from sklearn.metrics import mean_squared_error

reconstructed = autoencoder.predict(x_test)
mse = mean_squared_error(x_test.flatten(),
                           reconstructed.flatten())
print(f'MSE: {mse:.4f}')

# 2. SSIM (для изображений)
from skimage.metrics import structural_similarity
as ssim

ssim_scores = []
for i in range(len(x_test)):
    score = ssim(x_test[i], reconstructed[i])
    ssim_scores.append(score)

print(f'Mean SSIM: {np.mean(ssim_scores):.4f}')

# 3. Visualization
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Оригинал
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28),
cmap='gray')
    plt.axis('off')

    # Реконструкция
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(reconstructed[i].reshape(28, 28),
cmap='gray')
    plt.axis('off')
plt.show()
```

## ◆ 18. Чек-лист

- [ ] Нормализовать входные данные (0-1 или стандартизация)
- [ ] Выбрать правильный тип АЕ для задачи
- [ ] Подобрать размер латентного пространства
- [ ] Симметричная архитектура encoder/decoder
- [ ] Использовать appropriate activation functions
- [ ] Правильная функция потерь (BCE vs MSE)
- [ ] Регуляризация для предотвращения переобучения
- [ ] Визуализировать реконструкции
- [ ] Для VAE: балансировать reconstruction и KL loss

### Объяснение заказчику:

«Автоэнкодер — это нейросеть, которая учится сжимать данные до самого важного, а потом восстанавливать их обратно. Как ZIP-архив, но "умный" — он понимает, что важно в данных, а что можно отбросить».



# AutoML: Автоматизация машинного обучения

17 Январь 2026

## ◆ 1. Что такое AutoML?

- **Определение:** автоматизация процесса машинного обучения
- **Цель:** снизить барьер входа в ML, ускорить разработку
- **Автоматизирует:** выбор моделей, гиперпараметры, feature engineering
- **Для кого:** data scientists и разработчики
- **Преимущества:** экономия времени, базовые решения, эксперименты

## ◆ 2. Что автоматизируется?

| Этап                | Описание                                   |
|---------------------|--------------------------------------------|
| Предобработка       | Обработка пропусков, кодирование категорий |
| Feature Engineering | Создание новых признаков                   |
| Выбор модели        | Автоматический перебор алгоритмов          |
| Подбор параметров   | Оптимизация гиперпараметров                |
| Архитектура         | Поиск архитектуры нейросетей (NAS)         |
| Ансамбли            | Создание ансамблей моделей                 |
| Валидация           | Кросс-валидация и оценка                   |

## ◆ 3. Auto-sklearn

Автоматизация на основе scikit-learn:

```
# Установка
# pip install auto-sklearn

from autosklearn.classification import
AutoSklearnClassifier
from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score

# Разделение данных
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# AutoML модель
automl = AutoSklearnClassifier(
    time_left_for_this_task=3600, # 1 час
    per_run_time_limit=300, # 5 минут на
    модель
    n_jobs=-1,
    memory_limit=8192, # МБ
    ensemble_size=50
)

# Обучение
automl.fit(X_train, y_train)

# Предсказание
y_pred = automl.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test,
y_pred):.3f}")

# Статистика
print(automl.sprint_statistics())
print(automl.show_models())
```

## ◆ 4. TPOT (Tree-based Pipeline Optimization)

```
# Установка
# pip install tpot

from tpot import TPOTClassifier
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
)

# TPOT использует генетические алгоритмы
tpot = TPOTClassifier(
    generations=5, # поколения эволюции
    population_size=50, # размер популяции
    cv=5, # кросс-валидация
    random_state=42,
    verbosity=2,
    n_jobs=-1,
    max_time_mins=60, # максимальное время
    max_eval_time_mins=5 # время на одну
    модель
)

# Обучение
tpot.fit(X_train, y_train)

# Оценка
print(f"Score: {tpot.score(X_test, y_test):.3f}")

# Экспорт лучшего пайплайна
tpot.export('tpot_pipeline.py')
```

## ◆ 5. H2O AutoML

```
# Установка
# pip install h2o

import h2o
from h2o.automl import H2OAutoML

# Инициализация H2O
h2o.init()

# Загрузка данных в H2O
train = h2o.H2OFrame(df_train)
test = h2o.H2OFrame(df_test)

# Определение колонок
x = train.columns
y = 'target'
x.remove(y)

# AutoML
aml = H2OAutoML(
    max_runtime_secs=3600,           # 1 час
    max_models=20,                 # макс. моделей
    seed=42,
    nfolds=5,                      # кросс-валидация
    sort_metric='AUC'
)

# Обучение
aml.train(x=x, y=y, training_frame=train)

# Leaderboard - рейтинг моделей
lb = aml.leaderboard
print(lb)

# Лучшая модель
best_model = aml.leader

# Предсказание
preds = best_model.predict(test)
```

## ◆ 6. PyCaret

```
# Установка
# pip install pycaret

from pycaret.classification import *

# Инициализация
clf_setup = setup(
    data=df,
    target='target',
    session_id=42,
    train_size=0.8,
    normalize=True,
    transformation=True,
    ignore_features=['id'],
    silent=True,
    verbose=False
)

# Сравнение всех моделей
best_models = compare_models(n_select=3)

# Создание модели
best = create_model('xgboost')

# Тюнинг
tuned = tune_model(best, n_iter=50)

# Ансамбль
ensemble = ensemble_model(tuned, method='Bagging')

# Оценка
evaluate_model(ensemble)

# Финализация и сохранение
final = finalize_model(ensemble)
save_model(final, 'my_model')

# Предсказание
predictions = predict_model(final, data=test_df)
```

## ◆ 7. MLBox

```
# Установка
# pip install mlbox

from mlbox.preprocessing import Reader, Drift_thresholder
from mlbox.optimisation import Optimiser
from mlbox.prediction import Predictor

# Загрузка данных
paths = ["train.csv", "test.csv"]
target_name = "target"

rd = Reader(sep=",")
df = rd.train_test_split(paths, target_name)

# Удаление признаков с дрейфом
dft = Drift_thresholder()
df = dft.fit_transform(df)

# Оптимизация (AutoML)
opt = Optimiser(
    scoring='accuracy',
    n_folds=5
)

# Поиск лучших параметров
best_params = opt.optimize(
    space={'ne_numerical_strategy': {"search": "choice",
                                      "space": [0, 'mean']}},
    df=df,
    max_evals=40
)

# Предсказание
prd = Predictor()
prd.fit_predict(best_params, df)
```

## ◆ 8. AutoKeras (для нейросетей)

```
# Установка
# pip install autokeras

import autokeras as ak

# Классификация изображений
clf = ak.ImageClassifier(
    max_trials=10, # количество попыток
    overwrite=True,
    directory='autokeras',
    project_name='image_classification'
)

# Обучение
clf.fit(x_train, y_train, epochs=10)

# Оценка
clf.evaluate(x_test, y_test)

# Экспорт модели
model = clf.export_model()

# Классификация текста
text_clf = ak.TextClassifier(max_trials=10)
text_clf.fit(x_train, y_train)

# Табличные данные
structured_clf =
ak.StructuredDataClassifier(max_trials=10)
structured_clf.fit(x_train, y_train)

# Регрессия
reg = ak.StructuredDataRegressor(max_trials=10)
reg.fit(x_train, y_train)
```

## ◆ 9. Google AutoML Tables

Облачный сервис от Google (платный):

```
# Установка Google Cloud SDK и библиотеки
# pip install google-cloud-automl

from google.cloud import automl_v1beta1 as automl

# Создание клиента
client = automl.TablesClient(project='project-id',
region='us-central1')

# Создание датасета
dataset = client.create_dataset(
    dataset_display_name='my_dataset'
)

# Импорт данных
client.import_data(
    dataset=dataset,
    gcs_input_uris='gs://bucket/data.csv'
)

# Обучение модели
model = client.create_model(
    'my_model',
    dataset=dataset,
    train_budget_milli_node_hours=1000, # ~1 час
    optimization_objective='MAXIMIZE_AU_ROC'
)

# Предсказание
response = client.predict(
    model_name=model.name,
    inputs={'feature1': value1, 'feature2': value2}
)
```

## ◆ 10. Сравнение фреймворков

| Фреймворк           | Плюсы                                | Минусы                       |
|---------------------|--------------------------------------|------------------------------|
| <b>Auto-sklearn</b> | Мощный, научный подход               | Медленный, сложная установка |
| <b>TPOT</b>         | Генетические алгоритмы, экспорт кода | Долгое обучение              |
| <b>H2O AutoML</b>   | Быстрый, масштабируемый              | Свой формат данных           |
| <b>PyCaret</b>      | Простой, интерактивный               | Меньше контроля              |
| <b>AutoKeras</b>    | Для нейросетей, Keras API            | Только DL                    |
| <b>MLBox</b>        | Дрейф данных, feature engineering    | Меньше популярен             |

## ◆ 11. Байесовская оптимизация

Современный метод подбора гиперпараметров:

```
# Установка
# pip install scikit-optimize

from skopt import BayesSearchCV
from skopt.space import Real, Integer, Categorical
from sklearn.ensemble import RandomForestClassifier

# Пространство поиска
search_spaces = {
    'n_estimators': Integer(10, 200),
    'max_depth': Integer(3, 15),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 10),
    'max_features': Categorical(['sqrt', 'log2',
        None])
}

# Байесовская оптимизация
opt = BayesSearchCV(
    RandomForestClassifier(random_state=42),
    search_spaces,
    n_iter=50,           # количество итераций
    cv=5,
    n_jobs=-1,
    random_state=42
)

opt.fit(X_train, y_train)
print(f"Best score: {opt.best_score_:.3f}")
print(f"Best params: {opt.best_params_}")
```

## ◆ 12. Optuna

Продвинутый фреймворк для оптимизации:

```
# Установка
# pip install optuna

import optuna
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

def objective(trial):
    # Параметры для оптимизации
    n_estimators =
    trial.suggest_int('n_estimators', 10, 200)
    max_depth = trial.suggest_int('max_depth', 2,
32)
    min_samples_split =
    trial.suggest_int('min_samples_split', 2, 20)

    # Модель
    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        random_state=42
    )

    # Оценка
    score = cross_val_score(
        model, X_train, y_train,
        cv=5, scoring='f1_weighted'
    ).mean()

    return score

# Создание исследования
study = optuna.create_study(
    direction='maximize',
    sampler=optuna.samplers.TPESampler(seed=42)
)

# Оптимизация
study.optimize(objective, n_trials=100)

# Лучшие параметры
print(f"Best value: {study.best_value:.3f}")
print(f"Best params: {study.best_params}")

# Визуализация
```

```
optuna.visualization.plot_optimization_history(study)
optuna.visualization.plot_param_importances(study)
```

## ◆ 13. Hyperopt

```
# Установка
# pip install hyperopt

from hyperopt import hp, fmin, tpe, Trials,
STATUS_OK
from sklearn.ensemble import
GradientBoostingClassifier
from sklearn.model_selection import
cross_val_score

# Пространство поиска
space = {
    'n_estimators': hp.choice('n_estimators',
range(50, 200)),
    'max_depth': hp.choice('max_depth', range(3,
15)),
    'learning_rate':
hp.loguniform('learning_rate', -5, 0),
    'subsample': hp.uniform('subsample', 0.5, 1.0)
}

def objective(params):
    model = GradientBoostingClassifier(
        n_estimators=int(params['n_estimators']),
        max_depth=int(params['max_depth']),
        learning_rate=params['learning_rate'],
        subsample=params['subsample'],
        random_state=42
    )

    score = cross_val_score(
        model, X_train, y_train, cv=5
    ).mean()

    # Hyperopt минимизирует, поэтому возвращаем -
score
    return {'loss': -score, 'status': STATUS_OK}

# Оптимизация
trials = Trials()
best = fmin(
    fn=objective,
    space=space,
    algo=tpe.suggest,    # Tree-structured Parzen
Estimator
    max_evals=100,
    trials=trials
)
print(f"Best params: {best}")
```

## ◆ 14. Neural Architecture Search (NAS)

Автоматический поиск архитектуры нейросетей:

```
# AutoKeras для NAS
import autokeras as ak

# Поиск архитектуры
input_node = ak.ImageInput()
output_node = ak.ImageBlock(
    block_type='resnet',
    normalize=True,
    augment=True
)(input_node)
output_node = ak.ClassificationHead()(output_node)

clf = ak.AutoModel(
    inputs=input_node,
    outputs=output_node,
    max_trials=10
)

clf.fit(x_train, y_train, epochs=10)

# Или Keras Tuner
from kerastuner import HyperModel, RandomSearch
import tensorflow as tf

class MyHyperModel(HyperModel):
    def build(self, hp):
        model = tf.keras.Sequential()

        # Подбираем количество слоев и нейронов
        for i in range(hp.Int('num_layers', 1,
5)):
            model.add(tf.keras.layers.Dense(
                units=hp.Int(f'units_{i}', 32,
512, step=32),
                activation='relu'
            ))

            model.add(tf.keras.layers.Dense(10,
activation='softmax'))

        model.compile(
            optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )
        return model

tuner = RandomSearch(
    MyHyperModel(),
    objective='val_accuracy',
    max_trials=10
)
```

```
tuner.search(x_train, y_train, epochs=5,
validation_split=0.2)
```

## ◆ 15. Feature Engineering автоматизация

```
# Featuretools - автоматическое создание признаков
# pip install featuretools

import featuretools as ft
import pandas as pd

# Создание EntitySet
es = ft.EntitySet(id='data')

# Добавление датафрейма
es = es.add_dataframe(
    dataframe_name='transactions',
    dataframetransactions_df,
    index='transaction_id',
    time_index='timestamp'
)

es = es.add_dataframe(
    dataframe_name='customers',
    dataframetcustomers_df,
    index='customer_id'
)

# Связь между таблицами
es = es.add_relationship('customers',
'customer_id',
'transactions',
'customer_id')

# Автоматическая генерация признаков
feature_matrix, feature_defs = ft.dfs(
    entityset=es,
    target_dataframe_name='customers',
    max_depth=2,
    verbose=True
)

print(f"Generated {len(feature_defs)} features")
print(feature_matrix.head())
```

## ◆ 16. Автоматическая обработка данных

```
# DataPrep - автоматическая предобработка
# pip install dataprep

from dataprep.eda import create_report, plot,
plot_correlation

# Автоматический EDA отчет
create_report(df).show_browser()

# Визуализация признака
plot(df, 'column_name')

# Корреляции
plot_correlation(df)

# Sklearn ColumnTransformer - автоматическая обработка
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler,
OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Определение типов колонок
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns
categorical_features = df.select_dtypes(include=['object']).columns

# Пайпайн для числовых
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Пайпайн для категориальных
categorical_transformer = Pipeline(steps=[
    ('imputer',
     SimpleImputer(strategy='most_frequent')),
    ('onehot',
     OneHotEncoder(handle_unknown='ignore'))
])

# Объединение
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer,
         numeric_features),
        ('cat', categorical_transformer,
         categorical_features)
    ]
)
```

## ◆ 17. Когда использовать AutoML

### ✓ Хорошо использовать

- ✓ Быстрое прототипирование
- ✓ Базовая модель (baseline)
- ✓ Малый опыт в ML
- ✓ Ограничено время
- ✓ Стандартные задачи (классификация, регрессия)
- ✓ Табличные данные
- ✓ Исследование возможностей данных

### ✗ Не подходит

- ✗ Нестандартные задачи
- ✗ Специфичные требования
- ✗ Необходим глубокий контроль
- ✗ Очень большие данные (если не H2O)
- ✗ Критичная производительность
- ✗ Кастомные метрики
- ✗ Интерпретируемость важнее точности

## ◆ 18. Лучшие практики AutoML

- **Предобработка:** очистить данные перед AutoML
- **Время:** дать достаточно времени для поиска
- **Валидация:** проверить результаты на hold-out set
- **Интерпретация:** понять, что делает модель
- **Baseline:** использовать как отправную точку
- **Итерация:** дорабатывать вручную лучшие модели
- **Ресурсы:** контролировать вычислительные затраты
- **Документация:** сохранять параметры и результаты

## ◆ 19. Ограничения AutoML

- **Черный ящик:** не всегда понятно, что происходит
- **Вычислительные ресурсы:** может быть дорого
- **Время:** иногда долгое обучение
- **Переобучение:** может переобучиться на валидации
- **Специфика домена:** не учитывает доменные знания
- **Кастомизация:** сложно добавить свои компоненты
- **Воспроизведимость:** не всегда легко воспроизвести

## ◆ 20. Workflow с AutoML

- 1. Анализ данных:** EDA, понимание данных
- 2. Очистка:** обработка пропусков, выбросов
- 3. Разделение:** train/val/test split
- 4. AutoML:** запуск автоматического обучения
- 5. Анализ результатов:** изучение лучших моделей
- 6. Валидация:** проверка на test set
- 7. Доработка:** ручная оптимизация при необходимости
- 8. Feature engineering:** добавление доменных признаков
- 9. Повторный запуск:** AutoML с новыми признаками
- 10. Финализация:** выбор и сохранение модели

## ◆ 21. Метрики и мониторинг

```
# Большинство AutoML фреймворков поддерживают
# кастомные метрики

# PyCaret - кастомная метрика
from pycaret.classification import setup,
compare_models

def custom_metric(y_true, y_pred):
    # Ваша метрика
    return score

setup(data=df, target='target',
      custom_metric=custom_metric)

# Optuna - несколько метрик
def multi_objective(trial):
    # ... параметры ...
    model = ...

    accuracy = ...
    f1 = ...

    return accuracy, f1 # несколько целей

study = optuna.create_study(directions=
['maximize', 'maximize'])
study.optimize(multi_objective, n_trials=100)

# Мониторинг через Weights & Biases
import wandb
from wandb.keras import WandbCallback

wandb.init(project="automl-experiment")
# логирование метрик
wandb.log({"accuracy": accuracy, "loss": loss})
```

## ◆ 22. Продакшн и деплой

```
# Экспорт модели из AutoML
# ТР0Т
tpot.export('best_pipeline.py')

# PyCaret
save_model(model, 'my_automl_model')

# H2O
h2o.save_model(model, path='./models/')

# Загрузка и использование
# PyCaret
from pycaret.classification import load_model,
predict_model

model = load_model('my_automl_model')
predictions = predict_model(model, data=new_data)

# H2O
loaded_model =
h2o.load_model("./models/model_name")
predictions = loaded_model.predict(h2o_test_frame)

# Упаковка в Docker
# Dockerfile
FROM python:3.9
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY model.pkl .
COPY app.py .
CMD ["python", "app.py"]
```

## ◆ 23. Выбор AutoML фреймворка

**Для начинающих:** PyCaret (простой интерфейс)

**Для табличных данных:** H2O AutoML или Auto-sklearn

**Для нейросетей:** AutoKeras или Keras Tuner

**Для гибкости:** Optuna или Hyperopt

**Для больших данных:** H2O AutoML

**Для research:** Auto-sklearn или TPOT

**Для продакшна:** H2O или PyCaret

## ◆ 24. Чек-лист AutoML проекта

- [ ] Провести EDA и понять данные
- [ ] Очистить данные от явных ошибок
- [ ] Разделить на train/val/test
- [ ] Выбрать AutoML фреймворк
- [ ] Определить метрику оптимизации
- [ ] Установить время/ресурсы для поиска
- [ ] Запустить AutoML
- [ ] Проанализировать найденные модели
- [ ] Проверить на test set
- [ ] Интерпретировать результаты
- [ ] Добавить доменные знания
- [ ] Повторить с улучшениями
- [ ] Сохранить и задокументировать

### Объяснение заказчику:

«AutoML — это как опытный ассистент, который автоматически тестирует сотни комбинаций алгоритмов и настроек, находя оптимальное решение для ваших данных. Это экономит недели ручной работы и дает качественный результат быстро».



# AutoML для нейросетей

 17 Январь 2026

## ◆ 1. Суть

- **Автоматизация:** поиск архитектуры и гиперпараметров
- **NAS:** Neural Architecture Search
- **HPO:** Hyperparameter Optimization
- **Цель:** найти лучшую модель

## ◆ 2. Neural Architecture Search

- Search space: возможные архитектуры
- Search strategy: как искать
- Performance estimation: оценка кандидатов

## ◆ 3. Search Strategies

- Random search
- Bayesian optimization
- Evolutionary algorithms
- Reinforcement learning
- Gradient-based (DARTS)

## ◆ 4. DARTS

- Differentiable Architecture Search
- Relaxed search space
- Градиентная оптимизация
- Быстрее чем RL/Evolution

## ◆ 5. AutoKeras

- High-level API
- Поиск архитектуры автоматически
- Простое использование

## ◆ 6. Optuna для HP

- Hyperparameter optimization
- Tree-structured Parzen Estimator
- Pruning неперспективных trial'ов

## ◆ 7. Population Based Training

- Параллельное обучение
- Mutation лучших моделей
- Онлайн HP optimization

## ◆ 8. Meta-learning

- Warm start from similar tasks
- Transfer learned architectures
- Few-shot architecture search

## ◆ 9. Стоимость

- NAS: очень дорого (1000s GPU hours)
- HPO: умеренно (10-100 GPU hours)
- Transfer NAS: дешевле

## ◆ 10. Практическое применение

- Начать с готовых архитектур
- HPO для fine-tuning
- NAS только если есть ресурсы
- Использовать transfer learning

## ◆ 11. Weight Sharing в NAS

Ускорение поиска архитектуры

- One-shot NAS: одна супер-сеть
- Подсети наследуют веса
- Быстрее чем обучать каждую отдельно
- ENAS, DARTS используют это

## ◆ 12. Пример: Optuna HPO

Код для поиска гиперпараметров

```
import optuna

def objective(trial):
    lr = trial.suggest_float('lr', 1e-5,
    hidden = trial.suggest_int('hidden',
    layers = trial.suggest_int('layers',
        model = create_model(hidden, layers)
    val_loss = train(model, lr)
    return val_loss

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)
```

## ◆ 14. Стратегии ускорения

print(f'Best params: {study.best\_params}')

Как сделать NAS быстрее

- Early stopping для плохих кандидатов
- Transfer learning from similar tasks
- Low-fidelity estimates (меньше epochs)
- Parallel evaluation

## ◆ 13. AutoKeras пример

Простой AutoML

```
import autokeras as ak

clf = ak.StructuredDataClassifier(
    max_trials=10
)2, 512)
1, 5)
clf.fit(x_train, y_train, epochs=10)
preds = clf.predict(x_test)
```

## ◆ 15. Transfer NAS

Переиспользование архитектур

- Начать с архитектуры для похожей задачи
- Fine-tune architecture
- Дешевле чем поиск с нуля
- Работает для CV, NLP

## ◆ 16. Чек-лист

Практическое применение

- [ ] Начать с известной архитектуры
- [ ] HPO для fine-tuning
- [ ] Optuna для гиперпараметров
- [ ] Использовать early stopping
- [ ] Параллелизовать trials
- [ ] Рассмотреть transfer learning
- [ ] NAS только если нужно и есть ресурсы
- [ ] Сохранить лучшую модель



# Обратное распространение ошибки

Январь 2026

## ◆ 1. Суть

- **Цель:** эффективно вычислить градиенты в нейросети
- **Метод:** распространение ошибки от выхода к входу
- **Основа:** правило дифференцирования сложных функций (chain rule)
- **Применение:** обучение всех нейронных сетей

*Backpropagation — это способ узнать, насколько каждый нейрон виноват в ошибке, чтобы его исправить.*

## ◆ 2. Два этапа обучения

### 1. Forward pass (прямой проход):

- Данные идут от входа к выходу
- Вычисляются активации всех слоёв
- Получается предсказание
- Вычисляется функция потерь

### 2. Backward pass (обратный проход):

- Ошибка идёт от выхода к входу
- Вычисляются градиенты для каждого веса
- Веса обновляются градиентным спуском

## ◆ 3. Chain Rule (правило цепочки)

### Математическая основа backpropagation:

Если  $z = f(y)$  и  $y = g(x)$ , то:

$$\frac{\partial z}{\partial x} = \left(\frac{\partial z}{\partial y}\right) * \left(\frac{\partial y}{\partial x}\right)$$

для нейросети:

$$\frac{\partial \text{Loss}}{\partial w_1} = \left(\frac{\partial \text{Loss}}{\partial \text{output}}\right) * \left(\frac{\partial \text{output}}{\partial h}\right) * \left(\frac{\partial h}{\partial w_1}\right)$$

где  $h$  — скрытый слой

## ◆ 4. Простой пример: 1 слой

```
import numpy as np

# Forward pass
x = np.array([1.0, 2.0, 3.0]) # вход
w = np.array([0.5, -0.2, 0.1]) # веса
b = 0.1 # bias

# Линейная комбинация
z = np.dot(x, w) + b # z = 1.2

# Активация (sigmoid)
a = 1 / (1 + np.exp(-z)) # a ≈ 0.77

# Функция потерь (MSE)
y_true = 1.0
loss = (a - y_true)**2 # loss ≈ 0.053

# Backward pass
# Градиент loss по a
dL_da = 2 * (a - y_true) # ≈ -0.46

# Градиент a по z (производная sigmoid)
da_dz = a * (1 - a) # ≈ 0.177

# Градиент loss по z (chain rule)
dL_dz = dL_da * da_dz # ≈ -0.081

# Градиенты по весам
dL_dw = dL_dz * x # [-0.081, -0.163, -0.244]
dL_db = dL_dz # -0.081

# Обновление весов
learning_rate = 0.1
w = w - learning_rate * dL_dw
b = b - learning_rate * dL_db
```

## ◆ 5. Многослойная сеть

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size,
                 output_size):
        # Инициализация весов
        self.W1 = np.random.randn(input_size,
                                  hidden_size) * 0.01
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size,
                                  output_size) * 0.01
        self.b2 = np.zeros((1, output_size))

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def sigmoid_derivative(self, a):
        return a * (1 - a)

    def forward(self, X):
        # Слой 1
        self.z1 = X.dot(self.W1) + self.b1
        self.a1 = self.sigmoid(self.z1)

        # Слой 2
        self.z2 = self.a1.dot(self.W2) + self.b2
        self.a2 = self.sigmoid(self.z2)

        return self.a2

    def backward(self, X, y, output):
        m = X.shape[0]

        # Градиенты слоя 2
        dL_da2 = output - y # для MSE loss
        da2_dz2 = self.sigmoid_derivative(self.a2)
        dL_dz2 = dL_da2 * da2_dz2

        dL_dW2 = (1/m) * self.a1.T.dot(dL_dz2)
        dL_db2 = (1/m) * np.sum(dL_dz2, axis=0,
                                keepdims=True)

        # Градиенты слоя 1
        dL_da1 = dL_dz2.dot(self.W2.T)
        da1_dz1 = self.sigmoid_derivative(self.a1)
        dL_dz1 = dL_da1 * da1_dz1

        dL_dW1 = (1/m) * X.T.dot(dL_dz1)
        dL_db1 = (1/m) * np.sum(dL_dz1, axis=0,
                                keepdims=True)

        return dL_dW1, dL_db1, dL_dW2, dL_db2

    def train(self, X, y, epochs=1000, lr=0.1):
        for epoch in range(epochs):
            # Forward
            output = self.forward(X)
```

## Обратное распространение ошибки Cheatsheet — 3 колонки

```
# Backward
dW1, db1, dW2, db2 = self.backward(X,
                                     y, output)

# Обновление весов
self.W1 -= lr * dW1
self.b1 -= lr * db1
self.W2 -= lr * dW2
self.b2 -= lr * db2

if epoch % 100 == 0:
    loss = np.mean((output - y)**2)
    print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

## ◆ 6. Производные активаций

| Активация  | Функция                    | Производная                |
|------------|----------------------------|----------------------------|
| Sigmoid    | $\sigma(x) = 1/(1+e^{-x})$ | $\sigma(x)(1-\sigma(x))$   |
| Tanh       | $\tanh(x)$                 | $1 - \tanh^2(x)$           |
| ReLU       | $\max(0, x)$               | 1 if $x>0$ , else 0        |
| Leaky ReLU | $\max(\alpha x, x)$        | 1 if $x>0$ , else $\alpha$ |
| Softmax    | $e^{x_i}/\sum e^{x_j}$     | $\sigma_i(1-\sigma_j)$     |

```
def relu_derivative(z):
    return (z > 0).astype(float)

def tanh_derivative(a):
    return 1 - np.tanh(a)**2

def leaky_relu_derivative(z, alpha=0.01):
    dz = np.ones_like(z)
    dz[z < 0] = alpha
    return dz
```

## ◆ 7. Производные функций потерь

| Loss | Формула                                   | Производная $\partial L/\partial \hat{y}$ |
|------|-------------------------------------------|-------------------------------------------|
| MSE  | $(y - \hat{y})^2$                         | $2(\hat{y} - y)$                          |
| MAE  | $ y - \hat{y} $                           | $\text{sign}(\hat{y} - y)$                |
| BCE  | $-y \log(\hat{y}) - (1-y)\log(1-\hat{y})$ | $(\hat{y} - y)/(\hat{y}(1-\hat{y}))$      |
| CCE  | $-\sum y_i \log(\hat{y}_i)$               | $-y/\hat{y}$                              |

## ◆ 8. Computational Graph

## Визуализация вычислений:

Пример:  $z = (x + y) * w$ 

Forward:

 $x=2, y=3 \rightarrow a=5$  (сложение)  
 $a=5, w=4 \rightarrow z=20$  (умножение)

Backward:

$$\begin{aligned}\partial z/\partial z &= 1 \\ \partial z/\partial a &= w = 4 \\ \partial z/\partial w &= a = 5 \\ \partial z/\partial x &= \partial z/\partial a * \partial a/\partial x = 4 * 1 = 4 \\ \partial z/\partial y &= \partial z/\partial a * \partial a/\partial y = 4 * 1 = 4\end{aligned}$$

Каждая операция хранит:

- Входы (для backward pass)
- Выход (для forward pass)
- Локальные градиенты

## ◆ 9. Автоматическое дифференцирование

**Современные фреймворки делают это автоматически:**

```
# PyTorch
import torch

x = torch.tensor([1.0, 2.0, 3.0], requires_grad=True)
w = torch.tensor([0.5, -0.2, 0.1], requires_grad=True)
b = torch.tensor(0.1, requires_grad=True)

# Forward
z = torch.dot(x, w) + b
a = torch.sigmoid(z)
y_true = torch.tensor(1.0)
loss = (a - y_true)**2

# Backward (автоматически!)
loss.backward()

print(f"Градиент w: {w.grad}")
print(f"Градиент b: {b.grad}")

# TensorFlow
import tensorflow as tf

x = tf.constant([1.0, 2.0, 3.0])
w = tf.Variable([0.5, -0.2, 0.1])
b = tf.Variable(0.1)

with tf.GradientTape() as tape:
    z = tf.reduce_sum(x * w) + b
    a = tf.sigmoid(z)
    loss = (a - 1.0)**2

# Градиенты автоматически
gradients = tape.gradient(loss, [w, b])
print(f"Градиенты: {gradients}")
```

## ◆ 10. Проблемы backpropagation

| Проблема                    | Причина                     | Решение                              |
|-----------------------------|-----------------------------|--------------------------------------|
| <b>Vanishing gradient</b>   | Производные близки к 0      | ReLU, Batch Norm, ResNet             |
| <b>Exploding gradient</b>   | Производные слишком большие | Gradient clipping, Normalization     |
| <b>Dead ReLU</b>            | Нейроны всегда выдают 0     | Leaky ReLU, правильная инициализация |
| <b>Медленная сходимость</b> | Плохая инициализация        | Xavier, He initialization            |

## ◆ 11. Vanishing Gradient

**Проблема глубоких сетей:**

- Градиенты умножаются на каждом слое
- Если производная  $< 1$ , градиент уменьшается
- В глубоких сетях градиенты  $\rightarrow 0$
- Ранние слои почти не обучаются

```
# Пример: sigmoid имеет max производную 0.25
# В 10-слойной сети:  $(0.25)^{10} \approx 0.000001$ 

# Решения:
# 1. ReLU вместо sigmoid
def relu(x):
    return np.maximum(0, x)

# 2. Batch Normalization
from tensorflow.keras.layers import
BatchNormalization

# 3. Residual connections (ResNet)
output = input + F(input) # skip connection

# 4. LSTM/GRU для RNN
```

## ◆ 12. Exploding Gradient

**Противоположная проблема:**

- Градиенты становятся очень большими
- Веса обновляются слишком сильно
- Модель расходится

# Решения:

```
# 1. Gradient Clipping
import torch.nn as nn

max_grad_norm = 1.0
torch.nn.utils.clip_grad_norm_(model.parameters(),
max_grad_norm)

# 2. Правильная инициализация
# Xavier/Glorot для sigmoid/tanh
w = np.random.randn(n_in, n_out) *
np.sqrt(1.0/n_in)

# He initialization для ReLU
w = np.random.randn(n_in, n_out) *
np.sqrt(2.0/n_in)

# 3. Меньший learning rate
optimizer = torch.optim.Adam(model.parameters(),
lr=0.0001)

# 4. Batch Normalization
# Нормализует активации, стабилизирует градиенты
```

## ◆ 13. Проверка градиентов

```
def gradient_check(f, x, analytic_grad,
epsilon=1e-5):
    """
    Проверяет правильность аналитических
    градиентов
    численным методом
    """
    numeric_grad = np.zeros_like(x)

    for i in range(x.size):
        x_plus = x.copy()
        x_minus = x.copy()

        x_plus.flat[i] += epsilon
        x_minus.flat[i] -= epsilon

        # Численная производная
        numeric_grad.flat[i] = (f(x_plus) -
                               f(x_minus)) / (2*epsilon)

    # Относительная ошибка
    numerator = np.linalg.norm(numeric_grad -
                               analytic_grad)
    denominator = np.linalg.norm(numeric_grad) +
    np.linalg.norm(analytic_grad)
    relative_error = numerator / denominator

    if relative_error < 1e-7:
        print("✅ Градиенты корректны")
    else:
        print(f"❌ Ошибка: {relative_error}")
        print(f"Численный: {numeric_grad}")
        print(f"Аналитический: {analytic_grad}")

    return relative_error
```

## ◆ 14. Чек-лист

- [ ] Правильная инициализация весов (Xavier/He)
- [ ] Выбор подходящей активации (ReLU для глубоких сетей)
- [ ] Мониторинг градиентов (tensorboard, wandb)
- [ ] Gradient clipping для RNN
- [ ] Batch Normalization для глубоких сетей
- [ ] Проверка градиентов при разработке
- [ ] Использование автоматического дифференцирования
- [ ] Регулярная проверка на vanishing/exploding

## 💡 Объяснение заказчику:

«Backpropagation — это способ, которым нейросеть понимает, какие её части работают хорошо, а какие плохо, чтобы улучшить каждую часть в правильном направлении».

## ◆ 15. Оптимизация производительности

- Vectorization:** избегайте циклов, используйте матричные операции
- GPU acceleration:** используйте PyTorch/TensorFlow с CUDA
- Mixed precision:** FP16 для ускорения
- Gradient accumulation:** для больших батчей
- Checkpointing:** сохраняйте промежуточные активации

```
# PyTorch automatic mixed precision
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()

for data, target in dataloader:
    optimizer.zero_grad()

    with autocast():
        output = model(data)
        loss = criterion(output, target)

    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
```



# Бэггинг (Bootstrap Aggregating)

17 Январь 2026

## ◆ 1. Суть метода

**Bagging** (Bootstrap AGGREGATING) — метод ансамблирования, который создаёт несколько версий обучающей выборки с помощью bootstrap-сэмплирования.

- **Идея:** обучить много моделей на разных подвыборках
- **Bootstrap:** случайная выборка с возвращением
- **Агрегация:** усреднение (регрессия) или голосование (классификация)
- **Цель:** снизить дисперсию и переобучение
- **Эффект:** более стабильные и точные предсказания

Бэггинг особенно эффективен для нестабильных моделей (деревья решений), где небольшое изменение данных сильно меняет результат.

## ◆ 2. Алгоритм

1. Создать  $B$  bootstrap-выборок из исходных данных
2. Обучить базовый алгоритм на каждой выборке
3. Для регрессии: усреднить предсказания всех моделей
4. Для классификации: выбрать класс по мажоритарному голосованию

**Bootstrap-выборка:** выбираем  $N$  объектов из  $N$  с возвращением. В среднем 63.2% уникальных объектов попадают в каждую выборку.

## ◆ 3. Базовый код (Scikit-learn)

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Разделение данных
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Бэггинг с деревьями решений
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,           # Количество
    # Моделей
    max_samples=1.0,            # Размер каждой
    # Выборки
    max_features=1.0,          # Доля признаков
    bootstrap=True,             # Использовать
    bootstrap=True,
    bootstrap_features=False,   # Не сэмплировать
    # ПРИЗНАКИ
    oob_score=True,             # Out-of-bag оценка
    random_state=42,            # Параллельность
    n_jobs=-1
)

# Обучение
bagging.fit(X_train, y_train)

# Предсказание
y_pred = bagging.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")

# Out-of-bag оценка
print(f"OOB Score: {bagging.oob_score:.4f}")
```

## ◆ 4. Ключевые параметры

| Параметр     | Описание                         | Рекомендации                                 |
|--------------|----------------------------------|----------------------------------------------|
| n_estimators | Количество базовых моделей       | 50-500 (чем больше, тем лучше, но медленнее) |
| max_samples  | Размер bootstrap-выборки         | 0.5-1.0 (1.0 по умолчанию)                   |
| max_features | Доля признаков для каждой модели | 1.0 (все признаки)                           |
| bootstrap    | Использовать ли bootstrap        | True (суть метода)                           |
| oob_score    | Вычислять OOB-оценку             | True (бесплатная валидация)                  |
| n_jobs       | Число ядер CPU                   | -1 (все доступные)                           |

## ◆ 5. Out-of-Bag (OOB) оценка

**OOB** — это объекты, которые НЕ попали в bootstrap-выборку для конкретной модели (примерно 36.8%).

- Каждый объект оценивается моделями, которые не видели его при обучении
- Получаем бесплатную кросс-валидацию без отдельной валидационной выборки
- OOB-оценка близка к оценке на тестовой выборке

```
# OOB оценка автоматически
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,
    oob_score=True
)
bagging.fit(X_train, y_train)

print(f"OOB Score: {bagging.oob_score_:.4f}")

# OOB предсказания для каждого объекта
oob_predictions = bagging.oob_decision_function_
```

## ◆ 6. Регрессия с Bagging

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error,
r2_score

# Бэггинг для регрессии
bagging_reg = BaggingRegressor(
    estimator=DecisionTreeRegressor(max_depth=10),
    n_estimators=100,
    max_samples=0.8,
    oob_score=True,
    random_state=42,
    n_jobs=-1
)

# Обучение
bagging_reg.fit(X_train, y_train)

# Предсказание
y_pred = bagging_reg.predict(X_test)

# Метрики
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.4f}")
print(f"R2: {r2:.4f}")
print(f"OOB Score (R2): {bagging_reg.oob_score_:.4f}")
```

## ◆ 7. Преимущества и недостатки

### ✓ Преимущества

- ✓ Снижает дисперсию (переобучение)
- ✓ Работает с любыми базовыми алгоритмами
- ✓ Простой в реализации и понимании
- ✓ Параллелизуется легко
- ✓ ОВ-оценка как бесплатная валидация
- ✓ Стабилизирует нестабильные модели

### ✗ Недостатки

- ✗ Не снижает смещение (bias)
- ✗ Увеличивает вычислительные затраты
- ✗ Теряет интерпретируемость
- ✗ Может быть менее эффективен для стабильных моделей
- ✗ Требует больше памяти

## ◆ 8. Выбор базовой модели

Бэггинг эффективен для **нестабильных** (high variance) моделей:

| Модель             | Эффективность | Комментарий                                   |
|--------------------|---------------|-----------------------------------------------|
| Деревья решений    | ★★★★★         | Идеально! (Random Forest = Bagging + деревья) |
| Нейронные сети     | ★★★★★         | Хорошо снижает дисперсию                      |
| kNN                | ★★★           | Средний эффект                                |
| SVM                | ★★            | Небольшой прирост                             |
| Линейная регрессия | ★             | Почти нет эффекта (стабильная модель)         |

```
# Пример с разными базовыми моделями
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Bagging с kNN
bagging_knn = BaggingClassifier(
    estimator=KNeighborsClassifier(n_neighbors=5),
    n_estimators=50
)

# Bagging с SVM
bagging_svm = BaggingClassifier(
    estimator=SVC(probability=True),
    n_estimators=50
)
```

## ◆ 9. Feature Bagging (Случайные подпространства)

Вариант бэггинга, где мы также сэмплируем признаки:

```
# Бэггинг по признакам
feature_bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,
    max_samples=1.0, # Все объекты
    max_features=0.5, # Только 50%
    bootstrap=True,
    bootstrap_features=True, # Сэмплировать признаки!
    random_state=42
)

feature_bagging.fit(X_train, y_train)

# Полезно при большом числе признаков
# или когда признаки коррелированы
```

## ◆ 10. Настройка гиперпараметров

```
from sklearn.model_selection import GridSearchCV

# Сетка параметров
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0],
    'estimator__max_depth': [5, 10, 15, None] # Параметры базовой модели
}

# Grid Search
grid_search = GridSearchCV(
    BaggingClassifier(
        estimator=DecisionTreeClassifier(),
        oob_score=True,
        random_state=42
    ),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

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

## ◆ 11. Сравнение с Random Forest

**Random Forest** = Bagging + деревья + случайные признаки на каждом split

| Аспект             | Bagging                   | Random Forest   |
|--------------------|---------------------------|-----------------|
| Базовая модель     | Любая                     | Только деревья  |
| Выбор признаков    | При инициализации         | На каждом split |
| Декорреляция       | Средняя                   | Высокая         |
| Производительность | Зависит от базовой модели | Оптимизирована  |
| Гибкость           | Высокая                   | Только деревья  |

```
# Сравнение
from sklearn.ensemble import RandomForestClassifier

# Bagging с деревьями
bagging = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators=100
)

# Random Forest
rf = RandomForestClassifier(
    n_estimators=100,
    max_features='sqrt' # Дополнительная рандомизация
)

# Random Forest обычно лучше для деревьев
```

## ◆ 12. Когда использовать Bagging

### ✓ Используйте Bagging

- ✓ Модель сильно переобучается
- ✓ Высокая дисперсия предсказаний
- ✓ Нестабильная базовая модель (деревья)
- ✓ Небольшой датасет
- ✓ Нужна простая интерпретация ансамбля
- ✓ Хотите использовать не деревья (kNN, SVM, нейросети)

### ✗ Не используйте Bagging

- ✗ Высокое смещение (underfitting) — используйте Boosting
- ✗ Очень большой датасет — может быть избыточно
- ✗ Стабильная базовая модель (линейная регрессия)
- ✗ Критична скорость inference
- ✗ Ограничены ресурсы памяти

## ◆ 13. Практический пример: Wine Dataset

```
from sklearn.datasets import load_wine
from sklearn.model_selection import cross_val_score
import numpy as np

# Загрузка данных
wine = load_wine()
X, y = wine.data, wine.target

# Одно дерево
single_tree =
DecisionTreeClassifier(random_state=42)
scores_single = cross_val_score(
    single_tree, X, y, cv=5, scoring='accuracy'
)

# Bagging с деревьями
bagging = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators=100,
    random_state=42
)
scores_bagging = cross_val_score(
    bagging, X, y, cv=5, scoring='accuracy'
)

print(f"Одно дерево: {scores_single.mean():.4f} ± {scores_single.std():.4f}")
print(f"Bagging (100 деревьев): {scores_bagging.mean():.4f} ± {scores_bagging.std():.4f}")

# Результат: Bagging обычно стабильнее и точнее!
```

## ◆ 14. Важность признаков

```
# Feature importance для деревьев в бэггинге
import numpy as np

# Обучаем бэггинг с деревьями
bagging = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators=100,
    random_state=42
)
bagging.fit(X_train, y_train)

# Усредняем важность по всем деревьям
feature_importances = np.zeros(X_train.shape[1])
for estimator in bagging.estimators_:
    feature_importances += estimator.feature_importances_
feature_importances /= len(bagging.estimators_)

# Визуализация
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.bar(range(len(feature_importances)), feature_importances)
plt.xlabel('Признак')
plt.ylabel('Важность')
plt.title('Feature Importance в Bagging')
plt.show()
```

## ◆ 15. Чек-лист

- [ ] Выбрать нестабильную базовую модель (деревья, нейросети)
- [ ] Установить достаточное количество оценщиков (50-200)
- [ ] Включить `oob_score=True` для валидации
- [ ] Использовать `n_jobs=-1` для параллельности
- [ ] Настроить `max_samples` (обычно 0.5-1.0)
- [ ] Рассмотреть `max_features` для признаков
- [ ] Сравнить с одиночной моделью
- [ ] Проверить стабильность через кросс-валидацию
- [ ] Для деревьев: сравнить с Random Forest
- [ ] Настроить параметры базовой модели
- [ ] Оценить trade-off: точность vs скорость

### Объяснение заказчику:

«Бэггинг — это как опрос нескольких экспертов: каждый эксперт обучался на слегка разных данных, и мы усредняем их мнения. Это даёт более надёжные и стабильные предсказания, чем мнение одного эксперта».

# Batch Normalization

 17 Январь 2026

## 1. Суть

- Проблема:** Internal Covariate Shift — распределение активаций меняется при обучении
- Решение:** нормализация по батчу
- Где:** между слоями нейросети
- Эффект:** ускорение обучения, стабилизация, регуляризация
- Автор:** Sergey Ioffe, Christian Szegedy (2015)

## 2. Математика (упрощённо)

Для батча размером  $m$ :

- Среднее по батчу:**  $\mu = (1/m) \sum x_i$
- Дисперсия по батчу:**  $\sigma^2 = (1/m) \sum (x_i - \mu)^2$
- Нормализация:**  $\hat{x}_i = (x_i - \mu) / \sqrt{(\sigma^2 + \epsilon)}$
- Масштабирование и сдвиг:**  $y_i = \gamma \hat{x}_i + \beta$

$\gamma, \beta$  — обучаемые параметры

$\epsilon$  — маленькая константа для численной стабильности (обычно  $1e-5$ )

## 3. PyTorch (базовый)

```
import torch.nn as nn

# Для полносвязных слоёв
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.BatchNorm1d(256), # BatchNorm ПОСЛЕ
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.BatchNorm1d(128),
    nn.ReLU(),
    nn.Linear(128, 10)
)

# Для свёрточных слоёв
cnn_model = nn.Sequential(
    nn.Conv2d(3, 64, 3, padding=1),
    nn.BatchNorm2d(64), # BatchNorm2d для 2D
    nn.ReLU(),
    nn.Conv2d(64, 128, 3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU()
)
```

## 4. Keras/TensorFlow

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Dense(256, input_shape=(784,)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dense(128),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dense(10, activation='softmax')
])

# Для CNN
cnn_model = models.Sequential([
    layers.Conv2D(64, 3, padding='same',
    input_shape=(28, 28, 1)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Conv2D(128, 3, padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu')
])
```

## ◆ 5. Порядок слоёв (дебаты!)

**Классический порядок (оригинальная статья):**

```
Linear → BatchNorm → Activation
```

**Альтернативный порядок (иногда работает лучше):**

```
Linear → Activation → BatchNorm
```

**Рекомендация:** используйте классический порядок, если не уверены

## ◆ 6. Train vs Eval режимы

**КРИТИЧЕСКИ ВАЖНО!**

| Режим | Статистики                        | Параметры                    |
|-------|-----------------------------------|------------------------------|
| Train | $\mu, \sigma^2$ по текущему батчу | Обновляются running mean/var |
| Eval  | running mean/var                  | Фиксированы                  |

```
# PyTorch
model.train()    # Для обучения
model.eval()     # Для инференса

# TensorFlow/Keras
model.fit(...)   # Автоматически training=True
model.predict(...) # Автоматически training=False

# Явно:
output = model(x, training=True) # Train mode
output = model(x, training=False) # Eval mode
```

## ◆ 7. Преимущества

- Ускорение обучения:** можно использовать больший learning rate
- Стабильность:** меньше чувствительность к инициализации
- Регуляризация:** добавляет шум через батч-статистики
- Меньше dropout:** может частично заменить dropout
- Градиенты:** уменьшает проблему исчезающих градиентов

## ◆ 9. Проблемы с малыми батчами

**Проблема:** при малом batch size статистики неточны

**Альтернативы:**

- Layer Normalization:** нормализация по признакам, не по батчу
- Group Normalization:** нормализация по группам каналов
- Instance Normalization:** для style transfer

```
# PyTorch
nn.LayerNorm(normalized_shape)
nn.GroupNorm(num_groups, num_channels)
nn.InstanceNorm2d(num_features)
```

## ◆ 8. Важные параметры

| Параметр            | Описание                     | Значение по умолчанию    |
|---------------------|------------------------------|--------------------------|
| momentum            | Для running mean/var         | 0.1 (PyTorch), 0.99 (TF) |
| eps                 | Для численной стабильности   | 1e-5                     |
| affine              | Обучать $\gamma$ и $\beta$ ? | True                     |
| track_running_stats | Отслеживать running stats?   | True                     |

```
# PyTorch с параметрами
bn = nn.BatchNorm1d(
    num_features=256,
    eps=1e-5,
    momentum=0.1,
    affine=True
)
```

## ◆ 10. Layer Normalization

```
# Layer Norm – для RNN, Transformers
import torch.nn as nn
```

```
# Нормализация по признакам (не по батчу!)
layer_norm = nn.LayerNorm(hidden_size)
```

```
# Пример в Transformer
class TransformerBlock(nn.Module):
    def __init__(self, d_model):
        super().__init__()
        self.attention =
            MultiHeadAttention(d_model)
        self.norm1 = nn.LayerNorm(d_model)
        self.ffn = FeedForward(d_model)
        self.norm2 = nn.LayerNorm(d_model)

    def forward(self, x):
        # Residual + Norm
        x = x + self.attention(self.norm1(x))
        x = x + self.ffn(self.norm2(x))
        return x
```

## ◆ 11. Сравнение нормализаций

| Тип           | По чому нормализует | Применение          |
|---------------|---------------------|---------------------|
| Batch Norm    | По батчу и пикселям | CNN (большие батчи) |
| Layer Norm    | По признакам        | RNN, Transformers   |
| Instance Norm | По каждому примеру  | Style transfer      |
| Group Norm    | По группам каналов  | Малые батчи         |

## ◆ 12. Когда использовать

### ✓ Используйте BatchNorm

- ✓ CNN с достаточным batch size ( $\geq 16$ )
- ✓ Глубокие сети
- ✓ Классификация изображений
- ✓ Детекция объектов
- ✓ Нужно ускорить обучение

### ✗ Не используйте BatchNorm

- ✗ Малый batch size ( $< 8$ )
- ✗ RNN (используйте LayerNorm)
- ✗ Online learning (по одному примеру)
- ✗ Сильная зависимость от батча нежелательна

## ◆ 13. Лучшие практики

- Обязательно:** правильно переключайте train/eval режим
- Dropout:** можно уменьшить вероятность dropout с BatchNorm
- Learning rate:** можно увеличить в 2-10 раз
- Batch size:** минимум 8-16 для стабильности
- Инициализация:** менее критична с BatchNorm
- Не использовать bias:** в слое перед BatchNorm (он всё равно вычитается)

## ◆ 14. Отключение bias

```
# PyTorch: отключаем bias перед BatchNorm
nn.Linear(784, 256, bias=False) # Нет bias
nn.BatchNorm1d(256)             # β заменяет bias

nn.Conv2d(3, 64, 3, bias=False) # Нет bias
nn.BatchNorm2d(64)             # β заменяет bias

# Keras/TensorFlow
layers.Dense(256, use_bias=False)
layers.BatchNormalization()

layers.Conv2D(64, 3, use_bias=False)
layers.BatchNormalization()
```

## ◆ 15. Чек-лист

- [ ] Добавить BatchNorm после линейных/ свёрточных слоёв
- [ ] Убедиться в правильном порядке: Linear → BN → Activation
- [ ] Отключить bias в слое перед BatchNorm
- [ ] Правильно переключать train/eval режимы
- [ ] Использовать достаточный batch size ( $\geq 8-16$ )
- [ ] Попробовать увеличить learning rate
- [ ] Для малых батчей — рассмотреть LayerNorm/GroupNorm
- [ ] Для RNN/Transformers — использовать LayerNorm

## 💡 Объяснение заказчику:

«Batch Normalization — это как если бы мы на каждом этапе обработки данных приводили их к единому стандартному виду. Это позволяет нейросети обучаться быстрее и стабильнее, так как она не тратит время на адаптацию к постоянно меняющемуся распределению данных».



# Байесовский подход в Machine Learning

Январь 2026

## ◆ 1. Основы байесовского подхода

**Байесовский подход** рассматривает параметры модели как случайные величины с распределениями вероятностей.

- **Ключевая идея:** обновление убеждений на основе данных
- **Отличие от частотного подхода:** параметры — случайные величины
- **Преимущества:**
  - Квантификация неопределенности
  - Естественная регуляризация через априорные распределения
  - Работа с малым количеством данных

"Частотный подход спрашивает: какова вероятность данных при фиксированных параметрах? Байесовский: какова вероятность параметров при данных?"

## ◆ 2. Теорема Байеса

**Фундаментальная формула:**

$$P(\theta|D) = P(D|\theta) \times P(\theta) / P(D)$$

| Термин        | Название                           | Интерпретация                                  |
|---------------|------------------------------------|------------------------------------------------|
| $P(\theta D)$ | <b>Posterior</b><br>Апостериорное  | Убеждение о $\theta$ после наблюдения данных D |
| $P(D \theta)$ | <b>Likelihood</b><br>Правдоподобие | Вероятность данных при параметрах $\theta$     |
| $P(\theta)$   | <b>Prior</b><br>Априорное          | Начальное убеждение о $\theta$ до данных       |
| $P(D)$        | <b>Evidence</b><br>Свидетельство   | Нормализующая константа                        |

**Интуиция:**

$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$

"Новое убеждение = Данные  $\times$  Старое убеждение"

### ◆ 3. Пример: монета

**Задача:** оценить вероятность выпадения орла  $\theta$

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Данные: 7 орлов из 10 бросков
heads = 7
tosses = 10

# Prior: Beta(2, 2) - слегка информативный
prior_alpha, prior_beta = 2, 2

# Likelihood: Binomial(heads | θ, tosses)
# Posterior: Beta(alpha + heads, beta + tails)
posterior_alpha = prior_alpha + heads
posterior_beta = prior_beta + (tosses - heads)

# Posterior распределение
theta = np.linspace(0, 1, 1000)
prior = stats.beta.pdf(theta, prior_alpha,
prior_beta)
posterior = stats.beta.pdf(theta, posterior_alpha,
posterior_beta)

# Визуализация
plt.figure(figsize=(10, 5))
plt.plot(theta, prior, label='Prior', linestyle='--')
plt.plot(theta, posterior, label='Posterior')
plt.axvline(heads/tosses, color='red',
linestyle=':', label='MLE')
plt.xlabel('θ (вероятность орла)')
plt.ylabel('Плотность вероятности')
plt.legend()
plt.title('Байесовская оценка параметра монеты')
plt.show()

# Posterior среднее и интервал
post_mean = posterior_alpha / (posterior_alpha +
posterior_beta)
post_std = np.sqrt(posterior_alpha *
posterior_beta /
((posterior_alpha +
posterior_beta)**2 *
(posterior_alpha +
posterior_beta + 1)))

print(f"Posterior среднее: {post_mean:.3f}")
print(f"Posterior std: {post_std:.3f}")

# 95% credible interval
ci_low, ci_high = stats.beta.ppf([0.025, 0.975],
posterior_alpha,
```

```
posterior_beta)
print(f"95% CI: [{ci_low:.3f}, {ci_high:.3f}]")
```

### ◆ 4. Выбор априорного распределения (Prior)

**Типы приоров:**

| Тип                                       | Описание                                       | Когда использовать         |
|-------------------------------------------|------------------------------------------------|----------------------------|
| <b>Uninformative</b><br>(неинформативный) | Равномерное или широкое распределение          | Нет предварительных знаний |
| <b>Informative</b><br>(информативный)     | Узкое распределение вокруг ожидаемого значения | Есть сильные убеждения     |
| <b>Conjugate</b><br>(сопряженный)         | Prior и posterior одного семейства             | Для аналитического вывода  |
| <b>Empirical</b><br>(эмпирический)        | Оценен из данных                               | Иерархические модели       |

**Популярные сопряженные пары:**

- Binomial likelihood → Beta prior
- Normal likelihood (известная  $\sigma$ ) → Normal prior
- Poisson likelihood → Gamma prior
- Categorical likelihood → Dirichlet prior

### ◆ 5. Байесовская линейная регрессия

**Модель:**  $y = Xw + \epsilon$ , где  $w \sim N(\mu_0, \Sigma_0)$

```
import numpy as np
from scipy import stats

class BayesianLinearRegression:
    def __init__(self, alpha=1.0, beta=1.0):
        """
        alpha: precision of prior (1/variance)
        beta: precision of noise (1/noise_variance)
        """
        self.alpha = alpha
        self.beta = beta
        self.w_mean = None
        self.w_cov = None

    def fit(self, X, y):
        # Prior: w ~ N(0, (1/alpha)I)
        n_features = X.shape[1]

        # Posterior parameters
        S_N_inv = self.alpha * np.eye(n_features) +
        self.beta * X.T @ X
        self.w_cov = np.linalg.inv(S_N_inv)
        self.w_mean = self.beta * self.w_cov @ X.T @ y

        return self

    def predict(self, X, return_std=False):
        y_mean = X @ self.w_mean

        if return_std:
            # Predictive variance
            y_var = 1/self.beta + np.sum(X @
            self.w_cov * X, axis=1)
            y_std = np.sqrt(y_var)
            return y_mean, y_std

        return y_mean

    # Использование
    from sklearn.datasets import make_regression
    from sklearn.model_selection import train_test_split

    X, y = make_regression(n_samples=100,
    n_features=5, noise=0.5)
    X_train, X_test, y_train, y_test =
    train_test_split(X, y)

    # Обучение
    model = BayesianLinearRegression(alpha=1.0,
```

```

beta=25.0)
model.fit(X_train, y_train)

# Предсказание с неопределенностью
y_pred, y_std = model.predict(X_test,
return_std=True)

print(f"Posterior mean weights:
{model.w_mean}")
print(f"Posterior covariance:
{model.w_cov}")

# Визуализация неопределенности
plt.figure(figsize=(10, 5))
plt.errorbar(range(len(y_test)), y_pred,
yerr=2*y_std,
fmt='o', capsize=5, alpha=0.7,
label='Predictions')
plt.plot(y_test, 'r*', label='True values')
plt.xlabel('Sample')
plt.ylabel('Value')
plt.legend()
plt.title('Байесовская регрессия с
неопределенностью')
plt.show()

```

## ◆ 6. Наивный байесовский классификатор

**Предположение:** признаки условно независимы при данном классе

$$P(y|x_1, \dots, x_n) \propto P(y) \times \prod P(x_i|y)$$

```

from sklearn.naive_bayes import GaussianNB,
MultinomialNB

# Gaussian Naive Bayes (для числовых признаков)
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Предсказания
y_pred = gnb.predict(X_test)

# Вероятности классов
y_proba = gnb.predict_proba(X_test)

# Multinomial NB (для текста, счетов)
from sklearn.feature_extraction.text import
CountVectorizer

vectorizer = CountVectorizer()
X_train_counts =
vectorizer.fit_transform(texts_train)

mnb = MultinomialNB(alpha=1.0) # alpha -
# слаживание Лапласа
mnb.fit(X_train_counts, y_train)

# Bernoulli NB (для бинарных признаков)
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(X_train_binary, y_train)

```

### Когда использовать:

- Классификация текстов (очень эффективно)
- Базовый baseline
- Когда нужна быстрая модель
- Малое количество данных
- Сильная корреляция между признаками

## ◆ 7. Байесовская оптимизация гиперпараметров

Использование байесовского подхода для эффективного поиска гиперпараметров:

```

from sklearn.gaussian_process import
GaussianProcessRegressor
from sklearn.gaussian_process.kernels import
Matern
from scipy.stats import norm
from scipy.optimize import minimize

def bayesian_optimization(func, bounds,
n_iter=20):
    """
    Байесовская оптимизация
    func: целевая функция (например, валидационная
точность)
    bounds: границы параметров
    """
    # Начальные случайные точки
    X_sample = np.random.uniform(bounds[:, 0],
bounds[:, 1],
size=(5,
bounds.shape[0]))
    Y_sample = np.array([func(x) for x in
X_sample])

    # Gaussian Process
    gp = GaussianProcessRegressor(
        kernel=Matern(nu=2.5),
        alpha=1e-6,
        n_restarts_optimizer=10
    )

    for i in range(n_iter):
        # Обновить GP
        gp.fit(X_sample, Y_sample)

        # Acquisition function (Expected
Improvement)
        def acquisition(x):
            mu, sigma = gp.predict(x.reshape(1,
-1),
return_std=True)
            best_y = np.max(Y_sample)

            # Expected Improvement
            with np.errstate(divide='ignore'):
                Z = (mu - best_y) / sigma
                ei = (mu - best_y) * norm.cdf(Z) +
\

                sigma * norm.pdf(Z)

```

```

        return -ei # минимизируем
отрицательное EI

# Найти следующую точку
x_next = None
best_acq = np.inf

for x0 in np.random.uniform(bounds[:, 0],
bounds[:, 1],
size=(10,
bounds.shape[0])):
    result = minimize(acquisition, x0,
bounds=bounds,
method='L-BFGS-B')
    if result.fun < best_acq:
        best_acq = result.fun
        x_next = result.x

# Оценить функцию в новой точке
y_next = func(x_next)

# Добавить к сэмплам
X_sample = np.vstack([X_sample, x_next])
y_sample = np.append(y_sample, y_next)

# Лучшая найденная точка
best_idx = np.argmax(y_sample)
return X_sample[best_idx], y_sample[best_idx]

# Пример использования
def objective(params):
    """Оценка модели с данными гиперпараметрами"""
    from sklearn.ensemble import
RandomForestClassifier
    model = RandomForestClassifier(
        n_estimators=int(params[0]),
        max_depth=int(params[1]),
        min_samples_split=int(params[2]))
    )
    model.fit(X_train, y_train)
    return model.score(X_val, y_val)

bounds = np.array([
    [50, 200], # n_estimators
    [3, 20], # max_depth
    [2, 20] # min_samples_split
])
best_params, best_score = bayesian_optimization(
    objective, bounds, n_iter=30
)

print(f"Лучшие параметры: {best_params}")
print(f"Лучший score: {best_score:.4f}")

```

```

mean = predictions.mean(dim=0)
std = predictions.std(dim=0)

```

## 8. Байесовские нейронные сети

Вместо точечных оценок весов используем распределения:

```

# PyTorch + Blitz (Bayesian NN library)
import torch
import torch.nn as nn
from blitz.modules import BayesianLinear
from blitz.utils import variational_estimator

@variational_estimator
class BayesianNN(nn.Module):
    def __init__(self, input_dim, hidden_dim,
output_dim):
        super().__init__()
        self.blinear1 = BayesianLinear(input_dim,
hidden_dim)
        self.blinear2 = BayesianLinear(hidden_dim,
output_dim)

    def forward(self, x):
        x = torch.relu(self.blinear1(x))
        x = self.blinear2(x)
        return x

# Обучение
model = BayesianNN(input_dim=10, hidden_dim=50,
output_dim=1)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(),
lr=0.01)

for epoch in range(100):
    loss = model.sample_elbo(
        inputs=X_train,
        labels=y_train,
        criterion=criterion,
        sample_nbr=3 # количество сэмплов весов
    )

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# Предсказание с неопределенностью
predictions = []
for _ in range(100): # множественные сэмплы
    with torch.no_grad():
        pred = model(X_test)
        predictions.append(pred)

predictions = torch.stack(predictions)

```

## ◆ 9. Марковские цепи Монте-Карло (MCMC)

Для сложных posterior распределений используем сэмплирование:

```
# PyMC для байесовского моделирования
import pymc as pm

with pm.Model() as model:
    # Prior
    alpha = pm.Normal('alpha', mu=0, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=10)
    sigma = pm.HalfNormal('sigma', sigma=1)

    # Likelihood
    mu = alpha + beta * X
    y_obs = pm.Normal('y_obs', mu=mu, sigma=sigma,
                      observed=y)

    # MCMC сэмплирование
    trace = pm.sample(
        2000, # количество сэмплов
        tune=1000, # burn-in период
        return_inferencedata=True,
        chains=4 # параллельные цепи
    )

# Анализ результатов
import arviz as az

# Summary статистики
print(az.summary(trace))

# Визуализация posterior
az.plot_posterior(trace)

# Trace plots
az.plot_trace(trace)

# Диагностика сходимости
print(f"R-hat: {az.rhat(trace)}") # должно быть ~1.0
```

**Популярные MCMC алгоритмы:**

- **Metropolis-Hastings:** базовый алгоритм
- **Gibbs Sampling:** для известных условных распределений
- **Hamiltonian Monte Carlo (HMC):** эффективный для высоких размерностей

- **NUTS:** автоматическая настройка HMC

## ◆ 10. Вариационный байесовский вывод

Аппроксимация posterior простым распределением:

Минимизируем  $\text{KL}(q(\theta) \parallel p(\theta|D))$

Эквивалентно максимизации ELBO (Evidence Lower Bound)

```
# Variational Inference с PyMC
with pm.Model() as model:
    # Prior и likelihood как раньше
    alpha = pm.Normal('alpha', mu=0, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=10,
                     shape=X.shape[1])
    sigma = pm.HalfNormal('sigma', sigma=1)

    mu = alpha + pm.math.dot(X, beta)
    y_obs = pm.Normal('y_obs', mu=mu, sigma=sigma,
                      observed=y)

    # Variational inference (быстрее MCMC)
    approx = pm.fit(
        n=10000, # итераций
        method='advi' # Automatic Differentiation
    )

    # Получить posterior сэмплы
    trace = approx.sample(2000)

# Преимущества VI:
# ✓ Быстрее MCMC
# ✓ Масштабируется на большие данные
# ✓ Удобно для нейросетей (VAE)
# ✗ Аппроксимация (не точный posterior)
```

## ◆ 11. Гауссовские процессы

Байесовский подход к регрессии с бесконечномерными приорами:

```
from sklearn.gaussian_process import
GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF,
WhiteKernel

# Ядро (kernel) определяет prior
kernel = RBF(length_scale=1.0) +
WhiteKernel(noise_level=0.1)

gp = GaussianProcessRegressor(
    kernel=kernel,
    n_restarts_optimizer=10,
    alpha=1e-10
)

# Обучение
gp.fit(X_train, y_train)

# Предсказание с неопределенностью
y_pred, y_std = gp.predict(X_test,
                           return_std=True)

# Визуализация
plt.figure(figsize=(12, 5))
plt.plot(X_test, y_test, 'r.', label='True')
plt.plot(X_test, y_pred, 'b-', label='Prediction')
plt.fill_between(
    X_test.ravel(),
    y_pred - 2*y_std,
    y_pred + 2*y_std,
    alpha=0.3,
    label='95% confidence'
)
plt.legend()
plt.title('Gaussian Process Regression')
plt.show()

# Оптимизированные гиперпараметры ядра
print(f"Оптимизированное ядро:
{gp.kernel_}")
```

## ◆ 12. Байесовский А/В тест

Сравнение двух вариантов с учетом неопределенности:

```
import pymc as pm

# Данные А/В теста
n_A, conversions_A = 1000, 100 # вариант A
n_B, conversions_B = 1000, 120 # вариант B

with pm.Model() as model:
    # Prior (Beta распределение)
    p_A = pm.Beta('p_A', alpha=1, beta=1)
    p_B = pm.Beta('p_B', alpha=1, beta=1)

    # Likelihood (Binomial)
    obs_A = pm.Binomial('obs_A', n=n_A, p=p_A,
                         observed=conversions_A)
    obs_B = pm.Binomial('obs_B', n=n_B, p=p_B,
                         observed=conversions_B)

    # Разница между вариантами
    delta = pm.Deterministic('delta', p_B - p_A)

    # Сэмплирование
    trace = pm.sample(5000, tune=1000,
                      return_inferencedata=True)

# Анализ
import arviz as az

print(az.summary(trace, var_names=['p_A', 'p_B',
                                     'delta']))

# Вероятность, что B лучше A
prob_B_better = (trace.posterior['delta'] >
                  0).mean()
print(f"P(B > A) = {prob_B_better:.1%}")

# Визуализация
az.plot_posterior(trace, var_names=['delta'])

# Expected loss (ожидаемые потери при выборе неправильного варианта)
delta_samples =
    trace.posterior['delta'].values.flatten()
loss_choose_A = np.maximum(delta_samples,
                           0).mean()
loss_choose_B = np.maximum(-delta_samples,
                           0).mean()

print(f"Expected loss если выбрать A:
{loss_choose_A:.4f}")
```

```
print(f"Expected loss если выбрать B:
{loss_choose_B:.4f}")
```

## ◆ 13. Иерархические байесовские модели

Модели с несколькими уровнями параметров:

```
# Пример: студенты в разных школах
import pymc as pm

# Данные
n_schools = 8
n_students_per_school = 20

with pm.Model() as hierarchical_model:
    # Гиперпараметры (популяционные)
    mu_global = pm.Normal('mu_global', mu=0,
                           sigma=10)
    sigma_global = pm.HalfNormal('sigma_global',
                                  sigma=5)

    # Параметры школ (групповые)
    mu_school = pm.Normal('mu_school',
                           mu=mu_global,
                           sigma=sigma_global,
                           shape=n_schools)

    sigma_school = pm.HalfNormal('sigma_school',
                                 sigma=2,
                                 shape=n_schools)

    # Наблюдения студентов
    for i in range(n_schools):
        pm.Normal(f'obs_school_{i}', mu=mu_school[i],
                  sigma=sigma_school[i],
                  observed=student_scores[i])

    # Inference
    trace = pm.sample(2000, tune=1000,
                      return_inferencedata=True)

# Shrinkage effect: школы с малым количеством данных
# "притягиваются" к глобальному среднему
```

### Преимущества:

- Частичное pooling данных между группами
- Работа с малым количеством данных в группах
- Естественная регуляризация

## ◆ 14. Bayesian Model Averaging (BMA)

Комбинирование предсказаний нескольких моделей:

```
# BMA: взвешивание моделей по posterior вероятности
def bayesian_model_averaging(models, X_train, y_train, X_test):
    """
        models: список моделей [(model1, prior1),
        (model2, prior2), ...]
    """
    from scipy.special import logsumexp

    # Вычислить evidence для каждой модели
    log_evidences = []
    predictions = []

    for model, prior in models:
        # Обучить модель
        model.fit(X_train, y_train)

        # Log likelihood (приближение)
        y_pred = model.predict(X_train)
        log_likelihood = -0.5 * np.sum((y_train - y_pred)**2)

        # Log evidence = log likelihood + log prior
        log_evidence = log_likelihood +
        np.log(prior)
        log_evidences.append(log_evidence)

        # Предсказание
        predictions.append(model.predict(X_test))

    # Posterior model probabilities
    log_evidences = np.array(log_evidences)
    log_posterior_probs = log_evidences -
    logsumexp(log_evidences)
    posterior_probs = np.exp(log_posterior_probs)

    # Взвешенное предсказание
    predictions = np.array(predictions)
    bma_prediction = np.average(predictions,
    axis=0,
    weights=posterior_probs)

    return bma_prediction, posterior_probs

# Пример
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
```

Байесовский подход в Machine Learning Cheatsheet — 3 колонки

```
models = [
    (Ridge(alpha=1.0), 0.3),
    (Ridge(alpha=10.0), 0.3),
    (RandomForestRegressor(n_estimators=100), 0.4)
]

bma_pred, model_probs = bayesian_model_averaging(
    models, X_train, y_train, X_test
)

print("Posterior model probabilities:")
for i, prob in enumerate(model_probs):
    print(f"Model {i+1}: {prob:.3f}")
```

## ◆ 15. Best Practices

### ✓ Делать

- ✓ Визуализировать prior и posterior
- ✓ Проверять сходимость MCMC (R-hat, trace plots)
- ✓ Использовать информативные приоры когда есть знания
- ✓ Квантфицировать неопределенность
- ✓ Проводить posterior predictive checks

### ✗ Избегать

- ✗ Игнорировать выбор приора
- ✗ Использовать MCMC без проверки сходимости
- ✗ Забывать про масштабирование признаков
- ✗ Переоценивать точность с малым количеством сэмплов

## ◆ 16. Библиотеки Python

| Библиотека             | Особенности         | Применение                      |
|------------------------|---------------------|---------------------------------|
| PyMC                   | Мощная, современная | Общее байесовское моделирование |
| Stan (PyStan)          | HMC/NUTS, быстрый   | Сложные иерархические модели    |
| TensorFlow Probability | Интеграция с TF     | Байесовские нейросети           |
| Pyro                   | Основан на PyTorch  | Глубокие вероятностные модели   |
| scikit-learn           | Простые методы      | Naive Bayes, GP                 |

## ◆ 17. Частотный vs Байесовский

| Аспект           | Частотный                  | Байесовский        |
|------------------|----------------------------|--------------------|
| Параметры        | Фиксированные, неизвестные | Случайные величины |
| Вероятность      | Частота событий            | Степень убеждения  |
| Prior knowledge  | Не используется            | Через prior        |
| Неопределенность | Confidence intervals       | Credible intervals |
| Интерпретация    | Сложнее                    | Интуитивнее        |
| Вычисления       | Обычно быстрее             | Может быть дорого  |

## ◆ 18. Когда использовать байесовский подход

- **Малое количество данных:** prior помогает регуляризации
- **Нужна неопределенность:** получаем credible intervals
- **Есть экспертные знания:** можно включить через prior
- **Иерархические структуры:** естественное моделирование
- **Онлайн обучение:** легко обновлять posterior
- **A/B тестирование:** точная квантификация рисков
- **Очень большие данные + нужна скорость:** частотный быстрее
- **Простые задачи:** может быть overkill



# Байесовская оптимизация гиперпараметров

July  
17 4 января 2026

## 1. Суть

- Цель:** найти оптимальные гиперпараметры
- Подход:** умный перебор вместо случайного
- Основа:** Bayesian inference + Gaussian Processes
- Преимущество:** меньше итераций чем Grid/Random Search
- Применение:** любая ML модель с гиперпараметрами

## 2. Сравнение методов

| Метод         | Итерации    | Качество | Когда использовать |
|---------------|-------------|----------|--------------------|
| Grid Search   | Очень много | Хорошее  | Мало параметров    |
| Random Search | Много       | Среднее  | Много параметров   |
| Bayesian Opt  | Мало        | Отличное | Дорогие вычисления |
| Hyperband     | Средне      | Хорошее  | Быстрые модели     |

## 3. Как работает

### Алгоритм:

- Начать с нескольких случайных точек
- Построить Gaussian Process (суррогатная модель)
- Acquisition function выбирает следующую точку
- Оценить модель в этой точке
- Обновить GP, повторить 3-4

### Баланс exploration-exploitation:

- Exploration: пробовать новые области
- Exploitation: улучшать известные хорошие области

## 4. Код с scikit-optimize

```
from skopt import BayesSearchCV
from sklearn.ensemble import RandomForestClassifier

# Пространство поиска
search_spaces = {
    'n_estimators': (10, 300),
    'max_depth': (3, 20),
    'min_samples_split': (2, 20),
    'min_samples_leaf': (1, 10)
}

# Bayesian Search
opt = BayesSearchCV(
    RandomForestClassifier(),
    search_spaces,
    n_iter=50, # итераций
    cv=5,
    n_jobs=-1,
    random_state=42
)

opt.fit(X_train, y_train)
print(f"Best params: {opt.best_params_}")
print(f"Best score: {opt.best_score_.4f}")
```

## ◆ 5. Код с Optuna

```
import optuna
from sklearn.model_selection import cross_val_score

def objective(trial):
    params = {
        'n_estimators':
            trial.suggest_int('n_estimators', 10,
            300),
        'max_depth':
            trial.suggest_int('max_depth', 3, 20),
        'min_samples_split':
            trial.suggest_int('min_samples_split',
            2, 20),
        'learning_rate':
            trial.suggest_float('learning_rate',
            0.01, 1.0, log=True)
    }

    model =
RandomForestClassifier(**params)
    score = cross_val_score(model,
X_train, y_train, cv=5).mean()
    return score

study =
optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

print(f"Best params:
{study.best_params}")
print(f"Best value:
{study.best_value:.4f}")
```

## ◆ 6. Acquisition Functions

| Функция                         | Описание                | Когда использовать   |
|---------------------------------|-------------------------|----------------------|
| Expected Improvement (EI)       | Ожидаемое улучшение     | По умолчанию, баланс |
| Probability of Improvement (PI) | Вероятность улучшения   | Более консервативно  |
| Upper Confidence Bound (UCB)    | Верхняя граница доверия | Больше exploration   |
| Thompson Sampling               | Случайная выборка       | Стохастичность       |

## ◆ 7. Gaussian Process

**Суррогатная модель:** приближает целевую функцию

- Дает среднее и дисперсию предсказания
- Дисперсия = неопределенность
- Высокая дисперсия → нужно исследовать

**Ядро (kernel):**

- Matern: универсальное
- RBF: гладкие функции
- Выбор влияет на результат

## ◆ 8. Продвинутые техники

**Многофидельная оптимизация:**

```
# HyperBand + Bayesian
from ray import tune
from ray.tune.suggest.bayesopt import
BayesOptSearch
from ray.tune.schedulers import
HyperBandScheduler

config = {
    "learning_rate": tune.loguniform(1e-4, 1e-1),
    "batch_size": tune.choice([16, 32, 64, 128])
}

scheduler =
HyperBandScheduler(max_t=100)
search_alg = BayesOptSearch()

analysis = tune.run(
    train_model,
    config=config,
    scheduler=scheduler,
    search_alg=search_alg,
    num_samples=50
)
```

## ◆ 9. Параллелизация

```
# Optuna с параллелизацией
import optuna
from optuna.samplers import TPESampler

study = optuna.create_study(
    direction='maximize',
    sampler=TPESampler(n_startup_trials=10)
)

# Запуск параллельно
from joblib import Parallel, delayed

def run_trial(_):
    study.optimize(objective,
    n_trials=1)

Parallel(n_jobs=4)(delayed(run_trial)(i)
for i in range(100))
```

## ◆ 10. Визуализация процесса

```
import matplotlib.pyplot as plt

# История оптимизации
fig =
optuna.visualization.plot_optimization_history(
fig.show()

# Важность параметров
fig =
optuna.visualization.plot_param_importance(
fig.show()

# Slice plot (1D)
fig =
optuna.visualization.plot_slice(study)
fig.show()

# Contour plot (2D)
fig =
optuna.visualization.plot_contour(study,
    params=['learning_rate',
    'n_estimators'])
fig.show()
```

## ◆ 11. Warm Start

### Использование предыдущих результатов:

```
# Сохранение study
import joblib
joblib.dump(study, 'study.pkl')

# Загрузка и продолжение
study = joblib.load('study.pkl')
study.optimize(objective, n_trials=50)
# продолжить

# Transfer learning между датасетами
# Использовать результаты от похожей
задачи
```

## ◆ 12. Лучшие практики

- **Логарифмическая шкала:** для learning\_rate
- **Categorical:** для дискретных опций
- **Conditional:** зависимые параметры
- **Pruning:** останавливать плохие trials рано
- **Cross-validation:** обязательно использовать
- **Time budget:** ограничить время, а не итерации

## ◆ 13. Pruning (ранняя остановка)

```
import optuna

def objective(trial):
    params = {...}
    model =
RandomForestClassifier(**params)

    # Incremental training with pruning
    for epoch in range(100):
        score = model.score(X_val,
y_val)

        # Report intermediate value
        trial.report(score, epoch)

        # Prune if not promising
        if trial.should_prune():
            raise optuna.TrialPruned()

    return score

study = optuna.create_study(
    pruner=optuna.pruners.MedianPruner()
)
study.optimize(objective, n_trials=100)
```

## ◆ 14. Когда использовать

| Ситуация                   | Метод                        |
|----------------------------|------------------------------|
| Быстрая модель (<1 мин)    | Grid Search / Random Search  |
| Медленная модель (>10 мин) | <b>Bayesian Optimization</b> |
| Очень медленная (>1 час)   | Bayesian + Pruning           |
| Много параметров (>10)     | Random Search / Bayesian     |
| Мало ресурсов              | Hyperband                    |

## ◆ 15. Чек-лист

- [ ] Определить пространство поиска
- [ ] Выбрать метод (Bayesian если модель медленная)
- [ ] Настроить cross-validation
- [ ] Запустить с небольшим n\_trials
- [ ] Визуализировать процесс
- [ ] Использовать pruning если возможно
- [ ] Сохранить результаты
- [ ] Проверить на test set

«Байесовская оптимизация — это умный поиск гиперпараметров. Вместо слепого перебора, алгоритм учится на предыдущих попытках и направляет поиск в перспективные области. Это экономит время и вычислительные ресурсы, особенно для медленных моделей».

## ◆ 1. Основы байесовского вывода

**Байесовский вывод:** обновление вероятностных представлений на основе наблюдений

- **Теорема Байеса:**  $P(\theta|D) = P(D|\theta) \times P(\theta) / P(D)$
- **Prior**  $P(\theta)$ : априорное распределение до наблюдений
- **Likelihood**  $P(D|\theta)$ : правдоподобие данных при параметрах
- **Posterior**  $P(\theta|D)$ : апостериорное распределение после наблюдений
- **Evidence**  $P(D)$ : нормировочная константа

 *Байесовский вывод предоставляет полное распределение параметров, а не точечную оценку*

## ◆ 2. Формула Байеса

**Теорема Байеса:**

$$P(\theta|D) = P(D|\theta) \times P(\theta) / P(D)$$

где:

- $\theta$  (theta) - параметры модели
- $D$  - наблюдаемые данные
- $P(\theta)$  - априорное распределение
- $P(D|\theta)$  - функция правдоподобия
- $P(\theta|D)$  - апостериорное распределение
- $P(D) = \int P(D|\theta) P(\theta) d\theta$  - evidence

**Упрощенная форма:**

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

## ◆ 3. Выбор априорного распределения

| Тип                | Когда использовать    | Пример                                       |
|--------------------|-----------------------|----------------------------------------------|
| Uninformative      | Нет знаний            | Uniform, Jeffreys                            |
| Weakly informative | Общие знания          | Normal(0, 10)                                |
| Informative        | Сильные знания        | Normal( $\mu_{\text{expert}}$ , $\sigma^2$ ) |
| Conjugate          | Аналитическое решение | Beta для Bernoulli                           |
| Hierarchical       | Групповые данные      | Multi-level priors                           |

## ◆ 4. Сопряженные распределения

| Likelihood            | Conjugate Prior               | Posterior                                           |
|-----------------------|-------------------------------|-----------------------------------------------------|
| Bernoulli             | Beta( $\alpha, \beta$ )       | Beta( $\alpha+k, \beta+n-k$ )                       |
| Binomial              | Beta( $\alpha, \beta$ )       | Beta( $\alpha+k, \beta+n-k$ )                       |
| Multinomial           | Dirichlet( $\alpha$ )         | Dirichlet( $\alpha+counts$ )                        |
| Poisson               | Gamma( $\alpha, \beta$ )      | Gamma( $\alpha+\Sigma x, \beta+n$ )                 |
| Normal ( $\mu$ )      | Normal( $\mu_0, \sigma_0^2$ ) | Normal( $\mu_{\text{new}}, \sigma_{\text{new}}^2$ ) |
| Normal ( $\sigma^2$ ) | Inverse-Gamma                 | Inverse-Gamma                                       |

## ◆ 5. Байесовская линейная регрессия

```

import numpy as np
from scipy import stats

class BayesianLinearRegression:
    def __init__(self, alpha=1.0, beta=1.0):
        """
        alpha: точность prior на веса ( $1/\sigma^2$ )
        beta: точность noise ( $1/\sigma^2_{\text{noise}}$ )
        """
        self.alpha = alpha
        self.beta = beta
        self.m_N = None # Posterior mean
        self.S_N = None # Posterior covariance

    def fit(self, X, y):
        N, M = X.shape

        # Prior:  $w \sim N(0, \alpha^{-1}I)$ 
        S_0 = (1/self.alpha) * np.eye(M)
        m_0 = np.zeros(M)

        # Posterior:  $w \sim N(m_N, S_N)$ 
        S_N_inv = self.alpha * np.eye(M) +
        self.beta * X.T @ X
        self.S_N = np.linalg.inv(S_N_inv)
        self.m_N = self.beta * self.S_N @ X.T @ y

        return self

    def predict(self, X, return_std=False):
        # Предсказание:  $y \sim N(m_N^T x, \sigma^2)$ 
        y_pred = X @ self.m_N

        if return_std:
            # Uncertainty:  $\sigma^2 = 1/\beta + x^T S_N x$ 
            y_var = 1/self.beta + np.sum(X @
        self.S_N * X, axis=1)
            y_std = np.sqrt(y_var)
            return y_pred, y_std

        return y_pred

    def sample_weights(self, n_samples=100):
        """Сэмплирование весов из posterior"""
        return np.random.multivariate_normal(
            self.m_N, self.S_N, size=n_samples
        )

    # Использование
    model = BayesianLinearRegression(alpha=1.0,
    beta=25.0)
    model.fit(X_train, y_train)
    y_pred, y_std = model.predict(X_test,
    return_std=True)

```

```

# Confidence intervals
y_lower = y_pred - 1.96 * y_std
y_upper = y_pred + 1.96 * y_std

```

## ◆ 6. Байесовская классификация

```

import numpy as np
from scipy import stats

class BayesianLogisticRegression:
    """Байесовская логистическая регрессия с Laplace approximation"""

    def __init__(self, alpha=1.0, max_iter=100):
        self.alpha = alpha # Prior precision
        self.max_iter = max_iter
        self.w_map = None # MAP estimate
        self.H_inv = None # Inverse Hessian

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        N, M = X.shape

        # Initialize weights
        w = np.zeros(M)

        # Newton-Raphson optimization for MAP
        for _ in range(self.max_iter):
            # Predictions
            pred = self.sigmoid(X @ w)

            # Gradient
            grad = X.T @ (pred - y) + self.alpha *
        w

            # Hessian
            R = np.diag(pred * (1 - pred))
            H = X.T @ R @ X + self.alpha *
        np.eye(M)

            # Update
            w = w - np.linalg.solve(H, grad)

        self.w_map = w
        self.H_inv = np.linalg.inv(H)

        return self

    def predict_proba(self, X):
        # MAP prediction
        z = X @ self.w_map
        p = self.sigmoid(z)

        # Uncertainty (approximate)
        var = np.sum(X @ self.H_inv * X, axis=1)

        return p, var

    def predict(self, X, threshold=0.5):

```

```
p, _ = self.predict_proba(X)
return (p >= threshold).astype(int)
```

```
# Использование
model = BayesianLogisticRegression(alpha=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_proba, uncertainty = model.predict_proba(X_test)
```

## 7. Bayesian Model Selection

```
import numpy as np
from scipy import stats

def bayesian_model_comparison(models, X, y):
    """
    Сравнение моделей через Bayes Factor
    """
    log_evidences = []

    for model in models:
        # Вычисление log evidence (marginal likelihood)
        #  $P(D|M) = \int P(D|\theta, M) P(\theta|M) d\theta$ 

        # Laplace approximation
        model.fit(X, y)

        # Log likelihood at MAP
        if hasattr(model, 'log_likelihood'):
            log_like = model.log_likelihood(X, y)
        else:
            # Approximate for sklearn models
            y_pred = model.predict(X)
            log_like = -0.5 * np.sum((y -
y_pred)**2)

        # Penalty for complexity (BIC approximation)
        n_params = len(model.w_map) if
hasattr(model, 'w_map') else X.shape[1]
        n_samples = len(y)
        log_evidence = log_like - 0.5 * n_params *
np.log(n_samples)

        log_evidences.append(log_evidence)

    # Bayes Factors
    log_evidences = np.array(log_evidences)
    bayes_factors = np.exp(log_evidences -
log_evidences[0])

    # Posterior probabilities
    posterior_probs = bayes_factors /
np.sum(bayes_factors)

    return {
        'log_evidences': log_evidences,
        'bayes_factors': bayes_factors,
        'posterior_probs': posterior_probs
    }

# Использование
results = bayesian_model_comparison([model1,
model2, model3], X, y)
```

```
print(f"Posterior probabilities:
{results['posterior_probs']})")
```

## ◆ 8. Bayesian A/B Testing

```
import numpy as np
from scipy import stats

def bayesian_ab_test(conversions_A, trials_A,
                     conversions_B, trials_B,
                     prior_alpha=1, prior_beta=1):
    """
    Байесовское A/B тестирование с Beta-Binomial
    моделью
    """
    # Posterior distributions
    posterior_A = stats.beta(
        prior_alpha + conversions_A,
        prior_beta + trials_A - conversions_A
    )
    posterior_B = stats.beta(
        prior_alpha + conversions_B,
        prior_beta + trials_B - conversions_B
    )

    # Monte Carlo для P(B > A)
    samples_A = posterior_A.rvs(size=10000)
    samples_B = posterior_B.rvs(size=10000)

    prob_B_better = np.mean(samples_B > samples_A)

    # Expected loss
    expected_loss_A = np.mean(np.maximum(samples_B
- samples_A, 0))
    expected_loss_B = np.mean(np.maximum(samples_A
- samples_B, 0))

    # Posterior means and credible intervals
    mean_A = posterior_A.mean()
    mean_B = posterior_B.mean()
    ci_A = posterior_A.interval(0.95)
    ci_B = posterior_B.interval(0.95)

    return {
        'prob_B_better': prob_B_better,
        'expected_loss_A': expected_loss_A,
        'expected_loss_B': expected_loss_B,
        'mean_A': mean_A,
        'mean_B': mean_B,
        'ci_A': ci_A,
        'ci_B': ci_B
    }

# Пример
results = bayesian_ab_test(
    conversions_A=100, trials_A=1000,
    conversions_B=120, trials_B=1000
)
print(f"P(B > A) =
```

## Байесовский вывод Cheatsheet — 3 колонки

```
{results['prob_B_better']:.3f}")
print(f"Expected loss if choose A:
{results['expected_loss_A']:.4f}")
print(f"Expected loss if choose B:
{results['expected_loss_B']:.4f}")
```

## ◆ 9. Преимущества и недостатки

## Преимущества:

- ✓ Полная неопределенность (uncertainty quantification)
- ✓ Естественная регуляризация через prior
- ✓ Работает с малыми данными
- ✓ Incorporates prior knowledge
- ✓ Последовательное обновление
- ✓ Автоматический model selection

## Недостатки:

- ✗ Вычислительно затратно
- ✗ Требует выбора prior
- ✗ Сложная интерпретация
- ✗ Может быть чувствителен к prior

## ◆ 10. Библиотеки для байесовского вывода

```
# PyMC3
import pymc3 as pm

with pm.Model() as model:
    # Priors
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10,
shape=X.shape[1])
    sigma = pm.HalfNormal('sigma', sd=1)

    # Linear model
    mu = alpha + pm.math.dot(X, beta)

    # Likelihood
    y_obs = pm.Normal('y_obs', mu=mu, sd=sigma,
observed=y)

    # Inference
    trace = pm.sample(2000, tune=1000)

# Arviz для анализа
import arviz as az
az.plot_trace(trace)
az.summary(trace)

# Stan через PyStan
import pystan

stan_code = """
data {
    int N;
    vector[N] x;
    vector[N] y;
}
parameters {
    real alpha;
    real beta;
    real sigma;
}
model {
    y ~ normal(alpha + beta * x, sigma);
}"""
sm = pystan.StanModel(model_code=stan_code)
fit = sm.sampling(data={'N': len(y), 'x': X, 'y': y})
```

## ◆ 11. Лучшие практики

- **Выбор prior:** начните с weakly informative priors
- **Prior predictive checks:** симулируйте данные из prior
- **Posterior predictive checks:** проверяйте согласованность
- **Sensitivity analysis:** тестируйте разные priors
- **Convergence diagnostics:** R-hat, effective sample size
- **Визуализация:** используйте trace plots, posterior plots
- **Документация:** описывайте выбор priors

## ◆ 12. Применение

**Когда использовать байесовский вывод:**

- ⚪ Малое количество данных
- ⚪ Нужна оценка неопределенности
- ⚪ Есть prior knowledge
- ⚪ Последовательное обновление данных
- ⚪ А/В тестирование с малыми выборками
- ⚪ Иерархические модели
- ⚪ Robustness к outliers
- ⚪ Causal inference

 Байесовский подход особенно ценен когда неопределенность критична для принятия решений



# Байесовские нейронные сети

17 4 января 2026

## 1. Суть

- Цель:** неопределенность в предсказаниях нейросети
- Идея:** веса — не числа, а распределения
- Отличие от обычных NN:** выдают mean + uncertainty
- Применение:** медицина, финансы, autonomous vehicles
- Типы uncertainty:** aleatoric (шум данных) и epistemic (неуверенность модели)

## 2. Обычные NN vs Байесовские

| Аспект      | Обычные NN          | Байесовские NN        |
|-------------|---------------------|-----------------------|
| Веса        | Фиксированные числа | Распределения $P(w)$  |
| Выход       | Одно значение       | Распределение         |
| Uncertainty | Нет                 | Да                    |
| Обучение    | Градиентный спуск   | Variational Inference |
| Скорость    | Быстро              | Медленнее             |
| Robustness  | Средняя             | Выше                  |

## 3. Математическая основа

### Байесовский подход:

$$P(w|D) = P(D|w) * P(w) / P(D)$$

где:

- $P(w|D)$ : posterior (веса после данных)
- $P(D|w)$ : likelihood (вероятность данных)
- $P(w)$ : prior (априорное распределение)
- $P(D)$ : evidence (нормализация)

### Предсказание:

$$P(y|x, D) = \int P(y|x, w) P(w|D) dw$$

Интегрируем по всем возможным весам

## 4. Variational Inference

**Проблема:** точный posterior  $P(w|D)$  вычислить невозможно

**Решение:** приблизить его  $q(w|\theta)$

```
# Минимизируем KL-дивергенцию
KL(q(w|\theta) || P(w|D))

# Эквивалентно
максимизации ELBO:
ELBO = E_q[log P(D|w)] - 
      KL(q(w|\theta) || P(w))
      ↑ data fit
      ↑ regularization
```

## 5. Код с TensorFlow Probability

```
import tensorflow as tf
import tensorflow_probability as tfp
tfd = tfp.distributions

# Bayesian Dense Layer
model = tf.keras.Sequential([
    tfp.layers.DenseVariational(
        units=64,
        make_prior_fn=lambda
        *args: tfd.Normal(0., 1.),
        make_posterior_fn=tfp.layers
        kl_weight=1/len(X_train)
    ),
    tf.keras.layers.ReLU(),
    tfp.layers.DenseVariational(
        units=10,
        make_prior_fn=lambda
        *args: tfd.Normal(0., 1.),
        make_posterior_fn=tfp.layers
        kl_weight=1/len(X_train)
    )
])

# Компиляция
model.compile(
    optimizer='adam',
    loss=lambda y, p_y: -p_y.log_prob(y)
)

model.fit(X_train,
          y_train, epochs=10)
```

## ◆ 6. MC Dropout (простая альтернатива)

**Идея:** Dropout во время inference = приближение Bayesian NN

```
import tensorflow as tf

# Обычная модель с Dropout
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64,
        activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10)
])

# Предсказание с
# uncertainty
def predict_with_uncertainty(X,
    n_samples=100):
    predictions = []
    for _ in
    range(n_samples):
        # training=True
        чтобы Dropout работал
        pred = model(X,
            training=True)

    predictions.append(pred)

    predictions =
    tf.stack(predictions)
    mean =
    tf.reduce_mean(predictions,
        axis=0)
    std =
    tf.math.reduce_std(predictions,
        axis=0)

    return mean, std

y_mean, y_std =
predict_with_uncertainty(X_t
print(f"Prediction:
{y_mean[0]:.2f} ±
{y_std[0]:.2f}")
```

## ◆ 7. Bayes by Backprop

**Алгоритм:**

1. Веса  $w \sim N(\mu, \sigma^2)$
2. Sample  $w$  из  $q(w|\theta)$
3. Forward pass с этими весами
4. Backprop для  $\mu$  и  $\sigma$
5. Повторить

```
# PyTorch код
import torch
import torch.nn as nn

class
BayesianLinear(nn.Module):
    def __init__(self,
        in_features,
        out_features):
        super().__init__()
        self.w_mu =
        nn.Parameter(torch.randn(out
        in_features))
        self.w_rho =
        nn.Parameter(torch.randn(out
        in_features))

    def forward(self, x):
        w_sigma =
        torch.log(1 +
        torch.exp(self.w_rho))
        w = self.w_mu +
        w_sigma *
        torch.randn_like(w_sigma)
        return F.linear(x,
        w)
```

## ◆ 9. Ensemble как приближение

```
# Обучить несколько
моделей
models = []
for i in range(5):
    model = create_model()
    model.fit(X_train,
    y_train, epochs=10)
    models.append(model)

# Предсказание
predictions =
[m.predict(X_test) for m
in models]
mean_pred =
np.mean(predictions,
axis=0)
std_pred =
np.std(predictions,
axis=0)

# Это приближение Bayesian
posterior
# Но не учитывает
корреляции между весами
```

## ◆ 8. Типы Uncertainty

| Тип       | Описание                | Источник                      | Как<br>снизить                        |
|-----------|-------------------------|-------------------------------|---------------------------------------|
| Aleatoric | Шум в данных            | Измерения,<br>стохастичность  | Невозможно<br>(природа<br>данных)     |
| Epistemic | Неуверенность<br>модели | Мало данных,<br>плохая модель | Больше<br>данных,<br>лучшая<br>модель |

```
# Разделение uncertainty
# Aleatoric: из выходного
распределения
# Epistemic: из вариации
весов (MC Dropout)
```

## ◆ 10. Калибровка uncertainty

**Проверка:** если модель говорит 90% уверенность, должна быть права в 90% случаев

```
from sklearn.calibration
import calibration_curve

# Получить вероятности и uncertainty
probs, uncertainties =
predict_with_uncertainty(X_t)

# Calibration curve
prob_true, prob_pred =
calibration_curve(
    y_test, probs,
    n_bins=10
)

plt.plot(prob_pred,
prob_true, marker='o')
plt.plot([0, 1], [0, 1],
linestyle='--')
plt.xlabel('Predicted probability')
plt.ylabel('True probability')
plt.title('Calibration Plot')
plt.show()
```

## ◆ 11. Применения

- Медицина:** не давать диагноз если не уверен
- Autonomous driving:** передать управление человеку
- Финансы:** risk-aware торговля
- Active learning:** запросить метку для неуверенных примеров
- Out-of-distribution detection:** обнаружить странные входы

## ◆ 12. Продвинутые методы

**SWAG (Stochastic Weight Averaging Gaussian):**

```
# Аппроксимация posterior
через SWA
# Простой и быстрый метод
```

**Laplace Approximation:**

```
# Гауссовское приближение posterior
# Вокруг MAP estimate
```

**Hamiltonian Monte Carlo:**

```
# Точное семплирование
# Очень медленно
```

## ◆ 14. Когда использовать

| Ситуация                     | Рекомендация               |
|------------------------------|----------------------------|
| Критичные решения (медицина) | Bayesian NN обязательно    |
| Autonomous systems           | Bayesian NN                |
| Active learning              | Bayesian NN или MC Dropout |
| Обычная классификация        | Не обязательно             |
| Мало вычислений              | MC Dropout или Ensemble    |

## ◆ 13. Практические советы

- Начните с MC Dropout:** проще всего
- Используйте calibration:** проверить uncertainty
- Prior имеет значение:** выбирайте разумный
- KL weight:**  $1/N_{train}$  для баланса
- Visualization:** показывать uncertainty на графиках
- Decision threshold:** не предсказывать если uncertainty высокая

## ◆ 15. Чек-лист

- [ ] Выбрать метод (MC Dropout для начала)
- [ ] Реализовать uncertainty quantification
- [ ] Проверить calibration
- [ ] Визуализировать uncertainty
- [ ] Определить threshold для отказа от предсказания
- [ ] Тестировать на OOD данных
- [ ] Документировать limitations

«Байесовские нейронные сети не просто предсказывают, но и говорят насколько уверены в предсказании. Это критически важно в задачах где цена ошибки высока. Модель должна "знать что она не знает" и уметь сказать "я не уверена" вместо выдачи ложных предсказаний».



# Bayesian Optimization

3 января 2026

## ◆ 1. Суть

- **Для дорогих функций:** когда каждая оценка дорогая
- **Суррогатная модель:** Gaussian Process (GP) для аппроксимации
- **Acquisition function:** баланс exploration/exploitation
- **Итеративный процесс:** выбор следующей точки для оценки

Макс:  $f(x)$ , где оценка  $f(x)$  дорогая

## ◆ 2. Базовый код (scikit-optimize)

```
from skopt import gp_minimize
from skopt.space import Real, Integer

# Определить пространство поиска
space = [
    Real(1e-6, 1e-1, prior='log-uniform',
         name='learning_rate'),
    Integer(10, 100, name='n_estimators'),
    Integer(3, 10, name='max_depth')
]

# Целевая функция (минимизируем, например, -accuracy)
def objective(params):
    lr, n_est, depth = params

    model = RandomForestClassifier(
        n_estimators=n_est,
        max_depth=depth,
        random_state=42
    )
    model.fit(X_train, y_train)
    score = model.score(X_val, y_val)

    return -score # минимизируем, поэтому отрицательный

# Bayesian Optimization
result = gp_minimize(
    objective,
    space,
    n_calls=50,          # число итераций
    random_state=42,
    verbose=True
)

print(f"Best params: {result.x}")
print(f"Best score: {-result.fun}")
```

## ◆ 3. Базовый код (Optuna)

```
import optuna

def objective(trial):
    # Определяем гиперпараметры
    lr = trial.suggest_loguniform('learning_rate',
        1e-6, 1e-1)
    n_est = trial.suggest_int('n_estimators', 10,
        100)
    depth = trial.suggest_int('max_depth', 3, 10)

    # Обучаем модель
    model = RandomForestClassifier(
        n_estimators=n_est,
        max_depth=depth,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Возвращаем метрику (максимизируем)
    return model.score(X_val, y_val)

# Создать study
study = optuna.create_study(
    direction='maximize',
    sampler=optuna.samplers.TPESampler()
)

# Оптимизация
study.optimize(objective, n_trials=50)

print(f"Best params: {study.best_params}")
print(f"Best score: {study.best_value}")
```

## ◆ 4. Как работает

1. **Инициализация:** несколько случайных точек
  2. **Surrogate Model:** обучить GP на текущих точках
  3. **Acquisition Function:** найти следующую точку
  4. **Оценка:** вычислить  $f(x)$  в новой точке
  5. **Повторить:** обновить GP и повторить
- GP (Gaussian Process):** дает предсказание + uncertainty
- Acquisition Function:** использует оба для выбора

## ◆ 6. Параметры оптимизации

| Параметр         | Описание                | Совет                      |
|------------------|-------------------------|----------------------------|
| n_calls          | Число итераций          | 50-200, зависит от бюджета |
| n_initial_points | Случайных точек вначале | 10-20                      |
| acq_func         | Acquisition function    | 'EI', 'PI', 'gp_hedge'     |
| kappa            | Exploration для UCB     | 1.96 по умолчанию          |
| xi               | Exploration для EI/PI   | 0.01 по умолчанию          |

## ◆ 5. Acquisition Functions

| Функция                         | Описание                | Когда использовать   |
|---------------------------------|-------------------------|----------------------|
| EI (Expected Improvement)       | Ожидаемое улучшение     | По умолчанию, баланс |
| PI (Probability of Improvement) | Вероятность улучшения   | Консервативный       |
| UCB (Upper Confidence Bound)    | Верхняя граница доверия | Больше exploration   |
| LCB (Lower Confidence Bound)    | Нижняя граница          | Минимизация          |

## ◆ 7. Типы пространств поиска

```
from skopt.space import Real, Integer, Categorical
space = [
    # Непрерывные параметры
    Real(0.001, 0.1, prior='log-uniform',
         name='lr'),
    Real(0.1, 0.9, name='dropout'),

    # Целочисленные параметры
    Integer(10, 200, name='n_estimators'),
    Integer(2, 10, name='max_depth'),

    # Категориальные параметры
    Categorical(['adam', 'sgd', 'rmsprop'],
                name='optimizer'),
    Categorical([32, 64, 128, 256],
                name='batch_size')
]
```

## ◆ 8. Продвинутый пример с CV

```
from skopt import gp_minimize
from sklearn.model_selection import cross_val_score

def objective(params):
    lr, n_est, depth = params

    model = RandomForestClassifier(
        n_estimators=n_est,
        max_depth=depth,
        random_state=42
    )

    # Кросс-валидация
    scores = cross_val_score(
        model, X_train, y_train,
        cv=5, scoring='accuracy'
    )

    # Минимизируем отрицательный score
    return -scores.mean()

result = gp_minimize(
    objective,
    space,
    n_calls=50,
    n_initial_points=10,
    acq_func='EI',
    random_state=42
)

print(f"Best: {result.x}, Score: {-result.fun:.4f}")
```

## ◆ 9. Визуализация результатов

```
from skopt.plots import plot_convergence,
plot_objective
import matplotlib.pyplot as plt

# Convergence plot
plot_convergence(result)
plt.title('Convergence Plot')
plt.show()

# Objective function plot
plot_objective(result)
plt.tight_layout()
plt.show()

# История поиска
import pandas as pd
history = pd.DataFrame({
    'iteration': range(len(result.func_vals)),
    'score': -result.func_vals
})
history['best_so_far'] = history['score'].cummax()

plt.figure(figsize=(10, 6))
plt.plot(history['iteration'], history['score'],
'o', label='Score')
plt.plot(history['iteration'],
history['best_so_far'], '--', label='Best so far')
plt.xlabel('Iteration')
plt.ylabel('Score')
plt.legend()
plt.title('Optimization Progress')
plt.show()
```

## ◆ 10. Преимущества и недостатки

### ✓ Преимущества

- ✓ Эффективнее Grid/Random Search
- ✓ Учитывает прошлые оценки
- ✓ Хорошо для дорогих функций
- ✓ Баланс exploration/exploitation
- ✓ Меньше итераций для хорошего результата

### ✗ Недостатки

- ✗ Медленнее на одну итерацию
- ✗ Плохо масштабируется (>20 параметров)
- ✗ Требует больше памяти
- ✗ GP плохо работает с категориями
- ✗ Не параллелизуется легко

## ◆ 11. Когда использовать

### ✓ Хорошо подходит

- ✓ Дорогие вычисления (обучение модели)
- ✓ Малый бюджет итераций (<200)
- ✓ Непрерывное пространство
- ✓ 2-20 параметров
- ✓ Нужен оптимум, не просто "хорошо"

### ✗ Плохо подходит

- ✗ Дешевые вычисления (используйте Grid Search)
- ✗ Много параметров (>20)
- ✗ Нужна параллелизация
- ✗ Дискретное пространство с большим числом значений

## ◆ 12. BO vs Grid vs Random Search

| Метод         | Итераций | Эффективность | Когда использовать             |
|---------------|----------|---------------|--------------------------------|
| Grid Search   | Много    | Низкая        | Малое пространство, дешево     |
| Random Search | Средне   | Средняя       | Baseline, высокие размерности  |
| Bayesian Opt  | Мало     | Высокая       | Дорого, малое число параметров |

## ◆ 13. Использование с XGBoost

```
import xgboost as xgb
from skopt import gp_minimize
from skopt.space import Real, Integer

def objective(params):
    max_depth, learning_rate, n_estimators,
    subsample = params

    model = xgb.XGBClassifier(
        max_depth=max_depth,
        learning_rate=learning_rate,
        n_estimators=n_estimators,
        subsample=subsample,
        random_state=42,
        use_label_encoder=False,
        eval_metric='logloss'
    )

    model.fit(X_train, y_train)
    score = model.score(X_val, y_val)

    return -score

space = [
    Integer(3, 10, name='max_depth'),
    Real(0.01, 0.3, prior='log-uniform',
         name='learning_rate'),
    Integer(50, 300, name='n_estimators'),
    Real(0.5, 1.0, name='subsample')
]

result = gp_minimize(objective, space, n_calls=50)
print(f"Best params: {result.x}")
```

## ◆ 14. Параллелизация

```
# Optuna поддерживает параллелизацию
import optuna
from joblib import Parallel, delayed

def objective(trial):
    lr = trial.suggest_loguniform('lr', 1e-6, 1e-1)
    # ... обучение модели
    return score

# Параллельная оптимизация
study = optuna.create_study(direction='maximize')

# n_jobs=-1 использует все ядра
study.optimize(objective, n_trials=100, n_jobs=-1)

# Или ручная параллелизация (для skopt)
from skopt import Optimizer

opt = Optimizer(space, base_estimator='GP',
                 acq_func='EI')

# Запросить несколько точек параллельно
x = opt.ask(n_points=4)

# Оценить параллельно
y = Parallel(n_jobs=4)(delayed(objective)(xi) for
xi in x)

# Обновить optimizer
opt.tell(x, y)
```

## ◆ 15. Практические советы

- **Начните с малого:** 50-100 итераций обычно достаточно
- **Log-scale для LR:** используйте prior='log-uniform'
- **Мониторьте convergence:** постройте график
- **Сравните с Random:** baseline для оценки пользы
- **Warmstart:** используйте прошлые результаты
- **Optuna для продакшена:** лучше интеграция и DB

## ◆ 16. Чек-лист

- [ ] Определить целевую функцию (objective)
- [ ] Задать пространство поиска
- [ ] Выбрать библиотеку (skopt/Optuna)
- [ ] Определить бюджет итераций
- [ ] Запустить оптимизацию
- [ ] Визуализировать convergence
- [ ] Оценить лучшие параметры на test
- [ ] Сравнить с Random Search

«*Bayesian Optimization — золотой стандарт для подбора гиперпараметров когда вычисления дороги. Находит оптимум в 5-10 раз быстрее чем Random Search*».

 Полезные ссылки

-  [scikit-optimize Documentation](#)
-  [Optuna Documentation](#)
-  [Practical Bayesian Optimization Paper](#)
-  [Visual Guide to Bayesian Optimization](#)



# BERT Fine-tuning

17 Январь 2026

## ◆ 1. Суть

- **BERT**: Bidirectional Encoder Representations from Transformers
- **Fine-tuning**: дообучение на своих данных
- **Применение**: классификация текста, NER, Q&A
- **Библиотека**: Hugging Face Transformers

## ◆ 2. Установка

```
# pip
pip install transformers torch

# Дополнительно
pip install datasets accelerate
```

## ◆ 3. Базовый код

```
from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    Trainer,
    TrainingArguments
)

# Загрузка предобученной модели
model =
BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=2
)

# Токенизатор
tokenizer = BertTokenizer.from_pretrained('bert-
base-uncased')

# Токенизация данных
def tokenize(texts):
    return tokenizer(
        texts,
        padding=True,
        truncation=True,
        max_length=512,
        return_tensors="pt"
    )

inputs = tokenize(train_texts)
```

## ◆ 4. Fine-tuning с Trainer

```
# Параметры обучения
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    learning_rate=2e-5,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True
)

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset
)

# Обучение
trainer.train()

# Оценка
trainer.evaluate()
```

## ◆ 5. Предсказание

```
# Новый текст
text = "This is a test sentence"

# Токенизация
inputs = tokenizer(text, return_tensors="pt")

# Предсказание
outputs = model(**inputs)
predictions = outputs.logits.argmax(-1)

print(f"Predicted class: {predictions.item()}")
```

## ◆ 6. Параметры обучения

| Параметр         | Значение         |
|------------------|------------------|
| learning_rate    | 2e-5, 3e-5, 5e-5 |
| num_train_epochs | 2-4              |
| batch_size       | 8-32             |
| max_length       | 128-512          |
| warmup_steps     | 500-1000         |

## ◆ 7. Когда использовать

### ✓ Хорошо

- ✓ Классификация текста
- ✓ Sentiment analysis
- ✓ Named Entity Recognition
- ✓ Question Answering
- ✓ Есть GPU

### ✗ Плохо

- ✗ Малоданных (<1000)
- ✗ Нет GPU (очень медленно)
- ✗ Real-time на CPU
- ✗ Очень длинные тексты (>512)

## ◆ 8. Чек-лист

- [ ] Выбрать базовую модель (bert-base-uncased)
- [ ] Подготовить данные
- [ ] Токенизировать тексты
- [ ] Настроить TrainingArguments
- [ ] Fine-tune модель
- [ ] Оценить на test
- [ ] Сохранить модель

«BERT революционизировал NLP. Fine-tuning позволяет адаптировать мощную предобученную модель под вашу задачу за несколько часов обучения».

## 🔗 Полезные ссылки

- 📚 Transformers документация
- 🤗 BERT на Hugging Face
- 📄 BERT научная статья
- 🎥 BERT explained

# BERT и языковые модели

 Январь 2026

## 1. BERT: основы

**BERT** (Bidirectional Encoder Representations from Transformers) — революционная языковая модель от Google (2018)

- **Двунаправленная:** учитывает контекст слева и справа
- **Предобучение:** на огромных корпусах текстов
- **Fine-tuning:** дообучение на конкретной задаче
- **Transfer learning:** переносит знания между задачами

## 2. Архитектура BERT

Основана на **Transformer Encoder**

- **BERT-Base:** 12 слоев, 768 hidden units, 12 attention heads (110M параметров)
- **BERT-Large:** 24 слоя, 1024 hidden units, 16 attention heads (340M параметров)
- **Входные токены:** [CLS] + текст + [SEP]
- **Позиционные эмбеддинги:** до 512 токенов

## 3. Предобучение BERT

Две задачи для предобучения:

| Задача                         | Описание                                  |
|--------------------------------|-------------------------------------------|
| MLM (Masked Language Model)    | Предсказать 15% замаскированных токенов   |
| NSP (Next Sentence Prediction) | Определить, идет ли предложение B после A |

*MLM позволяет модели учиться двунаправленно, в отличие от классических LM*

## 4. Базовый код с BERT

```
from transformers import BertTokenizer, BertModel
import torch

# Загрузка модели и токенизатора
tokenizer = BertTokenizer.from_pretrained(
    'bert-base-uncased'
)
model = BertModel.from_pretrained(
    'bert-base-uncased'
)

# Токенизация
text = "Hello, how are you?"
inputs = tokenizer(
    text,
    return_tensors="pt",
    padding=True,
    truncation=True
)

# Получение эмбеддингов
with torch.no_grad():
    outputs = model(**inputs)

# Эмбеддинг [CLS] токена (представление всей последовательности)
cls_embedding = outputs.last_hidden_state[:, 0, :]

# Все токенные эмбеддинги
token_embeddings = outputs.last_hidden_state
```

## ◆ 5. Fine-tuning BERT для классификации

```
from transformers import
BertForSequenceClassification
from transformers import Trainer,
TrainingArguments

# Модель для классификации
model =
BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=2 # бинарная классификация
)

# Параметры обучения
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_steps=500,
    weight_decay=0.01,
    logging_steps=10
)

# Обучение
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset
)

trainer.train()
```

## ◆ 6. Варианты BERT

| Модель     | Особенность             | Применение                |
|------------|-------------------------|---------------------------|
| RoBERTa    | Без NSP, больше данных  | Лучше BERT                |
| ALBERT     | Факторизация параметров | Меньше параметров         |
| DistilBERT | Knowledge distillation  | 60% быстрее, 97% качества |
| ELECTRA    | Discriminator подход    | Эффективнее обучение      |
| DeBERTa    | Disentangled attention  | SOTA результаты           |

## ◆ 7. Многоязычные модели

- **mBERT**: multilingual BERT на 104 языках
- **XLM-RoBERTa**: cross-lingual, 100 языков
- **mT5**: multilingual T5
- **ruBERT**: специально для русского языка

```
# Загрузка ruBERT
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained(
    'DeepPavlov/rubert-base-cased'
)
model = BertModel.from_pretrained(
    'DeepPavlov/rubert-base-cased'
)
```

## ◆ 8. Задачи с BERT

| Задача                         | Модель HuggingFace            |
|--------------------------------|-------------------------------|
| Классификация текста           | BertForSequenceClassification |
| NER (Named Entity Recognition) | BertForTokenClassification    |
| Question Answering             | BertForQuestionAnswering      |
| Multiple Choice                | BertForMultipleChoice         |
| Sentence Similarity            | Sentence-BERT                 |

## ◆ 9. Оптимизация производительности

- **DistilBERT**: 40% меньше, 60% быстрее
- **Quantization**: квантование весов (int8)
- **ONNX**: оптимизированный runtime
- **TensorRT**: ускорение на GPU
- **Pruning**: удаление неважных весов

```
# Квантование модели
from transformers import
BertForSequenceClassification
from torch.quantization import quantize_dynamic

model =
BertForSequenceClassification.from_pretrained(
    'bert-base-uncased'
)

quantized_model = quantize_dynamic(
    model, {torch.nn.Linear}, dtype=torch.qint8
)
```

## ◆ 10. GPT vs BERT

| Аспект          | BERT             | GPT                                  |
|-----------------|------------------|--------------------------------------|
| Архитектура     | Encoder-only     | Decoder-only                         |
| Направление     | Двунаправленная  | Однонаправленная (авторегрессионная) |
| Задача обучения | MLM + NSP        | Next token prediction                |
| Лучше для       | Понимание текста | Генерация текста                     |
| Fine-tuning     | Требуется        | Можно zero-shot                      |

## ◆ 11. История языковых моделей

1. **Word2Vec, GloVe** (2013): статические эмбеддинги
2. **ELMo** (2018): контекстные эмбеддинги
3. **GPT** (2018): трансформер decoder
4. **BERT** (2018): двунаправленный трансформер
5. **GPT-2** (2019): масштабирование GPT
6. **T5, BART** (2019): encoder-decoder
7. **GPT-3** (2020): 175B параметров
8. **ChatGPT, GPT-4** (2022-2023): RLHF

## ◆ 12. Sentence-BERT

Модификация BERT для создания **sentence embeddings**

```
from sentence_transformers import
SentenceTransformer
from sklearn.metrics.pairwise import
cosine_similarity

# Загрузка модели
model = SentenceTransformer(
    'paraphrase-multilingual-MiniLM-L12-v2'
)

# Получение эмбеддингов
sentences = [
    "Машинное обучение – это круто",
    "Deep Learning очень интересен",
    "Я люблю пиццу"
]

embeddings = model.encode(sentences)

# Сходство
similarity = cosine_similarity(
    [embeddings[0]], [embeddings[1]]
)
print(f"Similarity: {similarity[0][0]:.3f}")
```

## ◆ 13. Применения BERT

- **Sentiment Analysis**: анализ тональности
- **Text Classification**: категоризация
- **NER**: извлечение сущностей
- **Question Answering**: SQuAD
- **Semantic Search**: семантический поиск
- **Summarization**: с encoder-decoder
- **Chatbots**: понимание запросов
- **Recommendation**: контентная фильтрация

## ◆ 14. Best Practices

1. **Выбор размера**: Base для начала, Large для лучшего качества
2. **Learning rate**: 2e-5, 3e-5, 5e-5
3. **Batch size**: 16 или 32 (зависит от GPU)
4. **Epochs**: 3-4 обычно достаточно
5. **Warmup**: 10% от общих шагов
6. **Max length**: обрезать до 128-512 токенов
7. **Gradient clipping**: max\_grad\_norm=1.0

## ◆ 15. Преимущества и ограничения

### Преимущества

- ✓ SOTA результаты на многих задачах NLP
- ✓ Transfer learning экономит время
- ✓ Понимает контекст
- ✓ Много готовых моделей

### Ограничения

- ✗ Требует много вычислительных ресурсов
- ✗ Ограничение 512 токенов
- ✗ Не генерирует текст (нужен GPT)
- ✗ Fine-tuning требует размеченных данных

## ◆ 16. Чек-лист использования

1. ✓ Выбрать подходящую предобученную модель
2. ✓ Подготовить данные в нужном формате
3. ✓ Настроить токенизатор с padding/truncation
4. ✓ Выбрать гиперпараметры (lr, batch\_size, epochs)
5. ✓ Добавить warmup для стабильности
6. ✓ Мониторить overfitting на валидации
7. ✓ Оценить на тестовой выборке
8. ✓ Оптимизировать для production (quantization, ONNX)

# Bias в нейронных сетях

 17 Январь 2026

## ◆ 1. Что такое Bias

- **Algorithm Bias:** систематические ошибки в предсказаниях
- **Причины:** данные, дизайн модели, процесс обучения
- **Последствия:** дискриминация, неравенство, несправедливость
- **Важность:** этические, правовые и социальные риски

*Bias в ML может усиливать существующие социальные предрассудки и создавать новые формы дискриминации.*

## ◆ 2. Типы Bias

| Тип              | Описание                               |
|------------------|----------------------------------------|
| Selection Bias   | Непредставительная выборка данных      |
| Historical Bias  | Отражение исторических предрассудков   |
| Measurement Bias | Неточность в измерениях признаков      |
| Aggregation Bias | Упрощение разнородных групп            |
| Evaluation Bias  | Несбалансированные тестовые данные     |
| Deployment Bias  | Использование в неподходящем контексте |

## ◆ 3. Источники Bias в данных

**Проблемы с данными:**

- **Underrepresentation:** недостаток данных для групп
- **Label bias:** предвзятость в метках
- **Sampling bias:** неравномерная выборка
- **Proxy variables:** косвенные признаки дискриминации

```
# Пример: гендерный bias в данных
train_data = {
    'male_doctors': 900,
    'female_doctors': 100,
    'male_nurses': 100,
    'female_nurses': 900
}
# Модель может усилить стереотипы
```

## ◆ 4. Protected Attributes

**Защищённые признаки** (не должны влиять на решения):

- Расса и этническая принадлежность
- Пол и гендерная идентичность
- Возраст
- Религия
- Сексуальная ориентация
- Инвалидность
- Национальность

*Даже без явного использования protected attributes, bias может проявляться через корреляции.*

## ◆ 5. Метрики Fairness

### Demographic Parity:

```
# P(ŷ=1|A=0) = P(ŷ=1|A=1)
# Равная вероятность положительного предсказания

from aif360.metrics import
BinaryLabelDatasetMetric

metric = BinaryLabelDatasetMetric(
    dataset,
    privileged_groups=[{'sex': 1}],
    unprivileged_groups=[{'sex': 0}]
)
print(metric.mean_difference())
```

### Equal Opportunity:

```
# P(ŷ=1|y=1,A=0) = P(ŷ=1|y=1,A=1)
# Равная true positive rate
```

## ◆ 6. Обнаружение Bias

### Инструменты анализа:

```
# AI Fairness 360 (IBM)
from aif360.datasets import AdultDataset
from aif360.metrics import ClassificationMetric

dataset = AdultDataset()
privileged = [{'race': 1}]
unprivileged = [{'race': 0}]

metric = ClassificationMetric(
    dataset_true, dataset_pred,
    privileged_groups=privileged,
    unprivileged_groups=unprivileged
)

print(f"Disparate impact:\n{metric.disparate_impact()}")
print(f"Equal opportunity:\n{metric.equal_opportunity_difference()}")
```

## ◆ 7. Pre-processing методы

### Устранение bias перед обучением:

- **Reweighting:** изменение весов примеров
- **Resampling:** балансировка групп
- **Learning fair representations:** удаление bias из признаков

```
# Reweighting
from aif360.algorithms.preprocessing import
Reweighting

RW = Reweighting(
    unprivileged_groups=unprivileged,
    privileged_groups=privileged
)
dataset_transformed = RW.fit_transform(dataset)
```

## ◆ 8. In-processing методы

### Модификация процесса обучения:

- **Adversarial debiasing:** adversarial сеть для удаления bias
- **Prejudice remover:** регуляризация для fairness
- **Fair constraints:** ограничения в оптимизации

```
# Adversarial debiasing
from aif360.algorithms.inprocessing import
AdversarialDebiasing

debiaser = AdversarialDebiasing(
    privileged_groups=privileged,
    unprivileged_groups=unprivileged,
    scope_name='debiaser',
    debias=True
)

debiaser.fit(dataset_train)
dataset_pred = debiaser.predict(dataset_test)
```

## ◆ 9. Post-processing методы

### Коррекция после обучения:

- **Equalized odds:** коррекция порогов
- **Calibrated equalized odds:** калибровка предсказаний
- **Reject option classification:** изменение решений вблизи границы

```
# Reject Option Classification
from aif360.algorithms.postprocessing import
RejectOptionClassification

ROC = RejectOptionClassification(
    unprivileged_groups=unprivileged,
    privileged_groups=privileged,
    low_class_thresh=0.01,
    high_class_thresh=0.99,
    num_class_thresh=100,
    num_ROC_margin=50,
    metric_name="Statistical parity difference",
    metric_ub=0.05,
    metric_lb=-0.05
)

dataset_transformed = ROC.fit_predict(
    dataset_valid, dataset_pred
)
```

## ◆ 10. Fairness-Accuracy Trade-off

| Подход                | Accuracy | Fairness |
|-----------------------|----------|----------|
| Baseline model        | 0.85     | 0.60     |
| Reweighting           | 0.83     | 0.75     |
| Adversarial debiasing | 0.82     | 0.82     |
| Post-processing       | 0.84     | 0.78     |

Часто приходится жертвовать небольшой точностью ради справедливости.

## ◆ 11. Практические примеры

### Кредитный scoring:

- Проблема: дискриминация по race/полу
- Решение: равные approval rates для групп
- Метрика: demographic parity

### Найм сотрудников:

- Проблема: bias против женщин в tech
- Решение: blind recruiting, fair representations
- Метрика: equal opportunity

### Медицинская диагностика:

- Проблема: underrepresentation меньшинств
- Решение: balanced data collection
- Метрика: equalized odds

## ◆ 12. Библиотеки для Fairness

```
# AI Fairness 360 (IBM)
pip install aif360

# Fairlearn (Microsoft)
pip install fairlearn

# What-If Tool (Google)
pip install witwidget

# FairML
pip install fairml
```

### Пример с Fairlearn:

```
from fairlearn.metrics import MetricFrame
from sklearn.metrics import accuracy_score

# Анализ по группам
metric_frame = MetricFrame(
    metrics=accuracy_score,
    y_true=y_test,
    y_pred=y_pred,
    sensitive_features=sensitive_features
)

print(metric_frame.by_group)
print(metric_frame.difference())
```

## ◆ 13. Best Practices

### ✓ Рекомендации

- ✓ Собирать разнообразные данные
- ✓ Регулярно проверять fairness метрики
- ✓ Документировать решения о bias
- ✓ Включать stakeholders в процесс
- ✓ Использовать multiple fairness definitions
- ✓ Мониторить модель в production

### ✗ Избегать

- ✗ Игнорирование protected attributes
- ✗ Использование biased proxy variables
- ✗ Тестирование только на accuracy
- ✗ Отсутствие diversity в команде
- ✗ Скрытие информации о bias

## ◆ 14. Правовые аспекты

- **GDPR** (EU): право на объяснение решений
- **Fair Credit Reporting Act** (US): защита от дискриминации
- **AI Act** (EU): регуляция высокорискованных AI систем
- **Равноправие**: национальные законы о недискриминации

*Compliance с законами требует активного мониторинга и устранения bias.*

# Bias-Variance Trade-off

 3 января 2026

## 1. Суть

- **Ошибка модели** = Bias<sup>2</sup> + Variance + Irreducible Error
- **Bias (смещение)**: ошибка из-за упрощений модели
- **Variance (дисперсия)**: чувствительность к обучающим данным
- **Trade-off**: уменьшение одного увеличивает другое

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \sigma^2$$

## 2. Bias (Смещение)

- **Определение**: насколько модель систематически ошибается
- **Высокий bias**: модель слишком простая (недообучение)
- **Примеры**: линейная модель для нелинейных данных
- **Симптомы**: плохая точность на train и test

```
# Пример: высокий bias
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
# Ошибка на train высокая → высокий bias
```

## 3. Variance (Дисперсия)

- **Определение**: насколько предсказания меняются при разных train данных
- **Высокая variance**: модель слишком сложная (переобучение)
- **Примеры**: глубокое дерево без ограничений
- **Симптомы**: отличная точность на train, плохая на test

```
# Пример: высокая variance
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor() # без ограничений
model.fit(X_train, y_train)
# Train отлично, test плохо → высокая variance
```

## 4. Визуальное сравнение

| Модель               | Bias    | Variance | Описание       |
|----------------------|---------|----------|----------------|
| Линейная регрессия   | Высокий | Низкая   | Недообучение   |
| Дерево (depth=3)     | Средний | Средняя  | Баланс ✓       |
| Дерево (без лимитов) | Низкий  | Высокая  | Переобучение   |
| Random Forest        | Низкий  | Средняя  | Хороший баланс |

## 5. Кривые обучения

```
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np

train_sizes, train_scores, test_scores =
learning_curve(
    model, X, y,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5, scoring='r2'
)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores.mean(axis=1),
label='Train')
plt.plot(train_sizes, test_scores.mean(axis=1),
label='Test')
plt.xlabel('Training examples')
plt.ylabel('Score')
plt.legend()
plt.title('Learning Curves')
plt.show()
```

## 6. Интерпретация кривых обучения

- **Высокий bias**: train и test ошибки близки, но высоки
- **Высокая variance**: большой разрыв между train и test
- **Хороший баланс**: train и test близки, обе низкие

 **Высокий bias**: train ≈ test, обе плохие

 **Высокая variance**: train отлично, test плохо

 **Баланс**: train ≈ test, обе хорошие

## ◆ 7. Способы уменьшить Bias

- ✓ Усложнить модель (больше параметров)
- ✓ Добавить больше признаков
- ✓ Полиномиальные признаки
- ✓ Уменьшить регуляризацию
- ✓ Использовать более сложный алгоритм

```
# Уменьшить bias
from sklearn.preprocessing import
PolynomialFeatures

# Добавить полиномиальные признаки
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

model = LinearRegression()
model.fit(X_poly, y)
```

## ◆ 8. Способы уменьшить Variance

- ✓ Упростить модель (меньше параметров)
- ✓ Больше обучающих данных
- ✓ Увеличить регуляризацию
- ✓ Кросс-валидация
- ✓ Ансамбли (Random Forest, Bagging)
- ✓ Early stopping
- ✓ Dropout (для нейросетей)

```
# Уменьшить variance
from sklearn.ensemble import RandomForestRegressor

# Ансамбль уменьшает variance
model = RandomForestRegressor(
    n_estimators=100,
    max_depth=5 # ограничить сложность
)
model.fit(X_train, y_train)
```

## ◆ 9. Регуляризация

```
from sklearn.linear_model import Ridge, Lasso

# Ridge (L2) регуляризация
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

# Lasso (L1) регуляризация
lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)

# alpha ↑ → bias ↑, variance ↓
# alpha ↓ → bias ↓, variance ↑
```

## ◆ 10. Validation Curves

```
from sklearn.model_selection import
validation_curve

param_range = [1, 2, 3, 5, 7, 10, 15, 20]
train_scores, test_scores = validation_curve(
    DecisionTreeRegressor(random_state=42),
    X, y,
    param_name='max_depth',
    param_range=param_range,
    cv=5, scoring='r2'
)

plt.figure(figsize=(10, 6))
plt.plot(param_range, train_scores.mean(axis=1),
label='Train')
plt.plot(param_range, test_scores.mean(axis=1),
label='Test')
plt.xlabel('max_depth')
plt.ylabel('R2 Score')
plt.legend()
plt.title('Validation Curve')
plt.show()
```

## ◆ 11. Диагностика проблемы

| Симптомы        | Проблема         | Решение                     |
|-----------------|------------------|-----------------------------|
| Train ↓, Test ↓ | Высокий bias     | Усложнить модель            |
| Train ↑, Test ↓ | Высокая variance | Упростить или больше данных |
| Train ↑, Test ↑ | Хороший баланс   | Продолжить оптимизацию      |

## ◆ 12. Сложность модели

### ✗ Слишком простая (High Bias)

- ✗ Линейная модель для нелинейных данных
- ✗ Малое число признаков
- ✗ Сильная регуляризация
- ✗ Мелкое дерево (max\_depth=1)

### ✓ Оптимальная сложность

- ✓ Баланс bias-variance
- ✓ Хорошо на train и test
- ✓ Кросс-валидация
- ✓ Подобранныя регуляризация

## ◆ 13. Практический пример

```
import numpy as np
from sklearn.model_selection import cross_val_score

# Простая модель (высокий bias)
simple = LinearRegression()
simple_scores = cross_val_score(simple, X, y,
cv=5, scoring='r2')

# Сложная модель (высокая variance)
complex = DecisionTreeRegressor()
complex_scores = cross_val_score(complex, X, y,
cv=5, scoring='r2')

# Сбалансированная модель
balanced = DecisionTreeRegressor(max_depth=5,
min_samples_split=20)
balanced_scores = cross_val_score(balanced, X, y,
cv=5, scoring='r2')

print(f"Simple: {simple_scores.mean():.3f}")
print(f"Complex: {complex_scores.mean():.3f}")
print(f"Balanced: {balanced_scores.mean():.3f}")
```

## ◆ 14. Роль количества данных

- **Мало данных:** высокая variance (переобучение легко)
- **Много данных:** variance ↓, но bias остается
- **Bias не уменьшается** с увеличением данных
- **Variance уменьшается** с увеличением данных

«Больше данных помогает только при высокой variance. При высоком bias нужна более сложная модель»

## ◆ 16. Практические советы

- **Начните с простой модели:** постепенно усложняйте
- **Всегда используйте кросс-валидацию**
- **Визуализируйте кривые обучения**
- **Не гонитесь за 100% на train:** это переобучение
- **Баланс важнее максимума:** test важнее train

«Bias-variance trade-off — фундаментальная концепция ML. Понимание этого баланса критично для построения хороших моделей».

## ◆ 15. Чек-лист диагностики

- [ ] Оценить ошибку на train и test
- [ ] Построить learning curves
- [ ] Построить validation curves
- [ ] Определить проблему (bias/variance)
- [ ] Применить соответствующие решения
- [ ] Повторить оценку

## Полезные ссылки

-  [Scikit-learn: Underfitting vs Overfitting](#)
-  [Wikipedia: Bias-variance tradeoff](#)
-  [Understanding Bias-Variane](#)

# Bias-Variance Decomposition

17 4 января 2026

## 1. Суть

- Цель:** понять источники ошибки модели
- Разложение:**  $\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$
- Bias (смещение):** систематическая ошибка
- Variance (дисперсия):** чувствительность к данным
- Trade-off:** уменьшая bias, увеличиваем variance

## 2. Математическая формула

Ожидаемая ошибка предсказания:

$$E[(y - \hat{f}(x))^2] = \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x)) + \sigma^2$$

где:

- $y$ : истинное значение
- $\hat{f}(x)$ : предсказание модели
- Bias<sup>2</sup>:  $E[\hat{f}(x)] - f(x)^2$
- Variance:  $E[(\hat{f}(x) - E[\hat{f}(x)])^2]$
- $\sigma^2$ : irreducible error (шум)

## 3. Bias (Смещение)

**Определение:** ошибка от неправильных предположений в алгоритме

**Причины высокого bias:**

- Слишком простая модель
- Недостаточная выразительность
- Мало признаков
- Линейная модель для нелинейных данных

**Признаки:**

- Низкая точность на train
- Низкая точность на test
- Недобучение (underfitting)

## 4. Variance (Дисперсия)

**Определение:** чувствительность модели к флуктуациям в обучающих данных

**Причины высокой variance:**

- Слишком сложная модель
- Много параметров
- Малый размер обучающей выборки
- Нет регуляризации

**Признаки:**

- Высокая точность на train
- Низкая точность на test
- Переобучение (overfitting)

## 5. Bias-Variance Trade-off

| Сложность модели | Bias      | Variance  | Итоговая ошибка        |
|------------------|-----------|-----------|------------------------|
| Очень низкая     | Высокий ↑ | Низкая ↓  | Высокая (underfitting) |
| Оптимальная      | Средний   | Средняя   | Минимальная ✓          |
| Очень высокая    | Низкий ↓  | Высокая ↑ | Высокая (overfitting)  |

## 6. Визуализация

```
import matplotlib.pyplot as plt
import numpy as np

# Симуляция bias-variance trade-off
complexity = np.linspace(0, 10, 100)
bias = 10 / (1 + complexity) # уменьшается
variance = complexity / 2 # растет
total_error = bias**2 + variance

plt.figure(figsize=(10, 6))
plt.plot(complexity, bias**2, label='Bias2', linewidth=2)
plt.plot(complexity, variance, label='Variance', linewidth=2)
plt.plot(complexity, total_error, label='Total Error',
         linewidth=3, linestyle='--')
plt.axvline(complexity[np.argmin(total_error)],
            color='r', linestyle=':', label='Optimal')
plt.xlabel('Model Complexity')
plt.ylabel('Error')
plt.legend()
plt.title('Bias-Variance Trade-off')
plt.grid(True, alpha=0.3)
plt.show()
```

## ◆ 7. Практическое измерение

```
from mlxtend.evaluate import bias_variance_decomp
from sklearn.tree import DecisionTreeRegressor

# Вычисление bias и variance
mse, bias, var = bias_variance_decomp(
    DecisionTreeRegressor(max_depth=3),
    X_train, y_train,
    X_test, y_test,
    loss='mse',
    num_rounds=100,
    random_state=42
)

print(f"MSE: {mse:.4f}")
print(f"Bias2: {bias:.4f}")
print(f"Variance: {var:.4f}")
print(f"Bias2 + Variance: {bias + var:.4f}")
```

## ◆ 8. Снижение Bias

### Методы:

- Усложнить модель (больше слоев, параметров)
- Добавить признаки, полиномиальные признаки
- Уменьшить регуляризацию
- Использовать нелинейные модели
- Увеличить время обучения
- Ансамбли (boosting)

```
# Пример: увеличение глубины дерева
shallow = DecisionTreeRegressor(max_depth=2) # high bias
deep = DecisionTreeRegressor(max_depth=10) # low bias
```

## ◆ 9. Снижение Variance

### Методы:

- Упростить модель (меньше параметров)
- Больше данных для обучения
- Регуляризация (L1, L2, dropout)
- Feature selection
- Early stopping
- Ансамбли (bagging, random forest)
- Cross-validation

```
# Пример: регуляризация
from sklearn.linear_model import Ridge

# Высокая регуляризация -> низкая variance
model = Ridge(alpha=10.0)
```

## ◆ 11. Диагностика с помощью кривых обучения

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, val_scores =
learning_curve(
    model, X, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10)
)

train_mean = train_scores.mean(axis=1)
val_mean = val_scores.mean(axis=1)

# Анализ
gap = train_mean - val_mean

if gap[-1] > 0.1:
    print("High variance (overfitting)")
    print("Solution: регуляризация, больше данных")
elif train_mean[-1] < 0.8:
    print("High bias (underfitting)")
    print("Solution: усложнить модель")
```

## ◆ 10. Примеры для разных моделей

| Модель                       | Bias    | Variance | Комментарий              |
|------------------------------|---------|----------|--------------------------|
| Linear Regression            | Высокий | Низкая   | Простая модель           |
| Polynomial (degree=10)       | Низкий  | Высокая  | Переобучение             |
| Decision Tree (max_depth=20) | Низкий  | Высокая  | Гибкая, но нестабильная  |
| Random Forest                | Средний | Низкая   | Bagging снижает variance |
| Gradient Boosting            | Низкий  | Средняя  | Boosting снижает bias    |
| k-NN (k=1)                   | Низкий  | Высокая  | Очень гибкая             |
| k-NN (k=100)                 | Высокий | Низкая   | Усреднение               |

## ◆ 12. Ансамбли и Bias-Variance

**Bagging** (Random Forest):

- Снижает variance
- Не меняет bias
- Усредняет независимые модели

**Boosting** (XGBoost, AdaBoost):

- Снижает bias
- Может увеличить variance
- Последовательно улучшает модели

```
# Bagging снижает variance
from sklearn.ensemble import BaggingRegressor
bagging = BaggingRegressor(n_estimators=50)

# Boosting снижает bias
from sklearn.ensemble import GradientBoostingRegressor
boosting =
GradientBoostingRegressor(n_estimators=100)
```

## ◆ 13. Практический workflow

1. **Baseline:** простая модель (high bias)
2. **Диагностика:** learning curves
3. **Если underfitting:** усложнить модель
4. **Если overfitting:** регуляризация, больше данных
5. **Повторить** до оптимального баланса

## ◆ 14. Чек-лист

- [ ] Построить learning curves
- [ ] Определить: high bias или high variance?
- [ ] Для high bias: усложнить модель, добавить признаки
- [ ] Для high variance: регуляризация, больше данных
- [ ] Попробовать разные модели
- [ ] Использовать cross-validation
- [ ] Измерить bias и variance численно
- [ ] Найти оптимальную сложность модели

## ◆ 15. Заключение

«Bias-variance trade-off — это фундаментальная концепция в ML. Простая модель делает слишком сильные предположения (high bias), сложная — слишком чувствительна к данным (high variance). Искусство ML — найти золотую середину. Это как настройка микроскопа: слишком грубо — не видно деталей, слишком точно — видно только шум».

# RNN (Двунаправленные нейросети)

 Январь 2026

## ◆ 1. Суть

- **Специализация:** последовательные данные
- **Память:** использует информацию из прошлого
- **Двунаправленность:** выход возвращается на вход
- **Применение:** текст, временные ряды, аудио

## ◆ 2. Структура RNN

- **x\_t:** входные данные в момент t
- **h\_t:** скрытое состояние (память)
- **y\_t:** выходные данные

### Формула:

$$\begin{aligned} h_t &= \tanh(w_{hh} * h_{t-1} + w_{xh} * x_t + b_h) \\ y_t &= w_{hy} * h_t + b_y \end{aligned}$$

## ◆ 3. Простая RNN (PyTorch)

```
import torch
import torch.nn as nn

class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size,
                 output_size):
        super().__init__()
        self.hidden_size = hidden_size

        self.rnn = nn.RNN(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=1,
            batch_first=True
        )

        self.fc = nn.Linear(hidden_size,
                           output_size)

    def forward(self, x):
        # x: (batch_size, seq_len, input_size)
        out, hidden = self.rnn(x)
        # out: (batch_size, seq_len, hidden_size)

        # Берём последний выход
        out = self.fc(out[:, -1, :])
        return out

model = SimpleRNN(input_size=10, hidden_size=128,
                   output_size=1)
```

## ◆ 4. LSTM (Long Short-Term Memory)

Решение проблемы исчезающего градиента:

```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size,
                 output_size):
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=2,
            batch_first=True,
            dropout=0.2
        )

        self.fc = nn.Linear(hidden_size,
                           output_size)

    def forward(self, x):
        out, (hidden, cell) = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out
```

## ◆ 5. GRU (Gated Recurrent Unit)

Упрощённая версия LSTM:

```
class GRUModel(nn.Module):
    def __init__(self, input_size, hidden_size,
                 output_size):
        super().__init__()

        self.gru = nn.GRU(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=2,
            batch_first=True,
            dropout=0.2
        )

        self.fc = nn.Linear(hidden_size,
                           output_size)

    def forward(self, x):
        out, hidden = self.gru(x)
        out = self.fc(out[:, -1, :])
        return out

# GRU быстрее LSTM, но немного менее мощный
```

## ◆ 6. Сравнение RNN, LSTM, GRU

| Модель | Скорость | Память   | Градиент   |
|--------|----------|----------|------------|
| RNN    | ⚡⚡⚡      | Короткая | Исчезает   |
| LSTM   | ⚡        | Длинная  | Стабильный |
| GRU    | ⚡⚡       | Длинная  | Стабильный |

## ◆ 7. Двунаправленные RNN

```
# Bidirectional RNN
class BiRNN(nn.Module):
    def __init__(self, input_size, hidden_size,
                 output_size):
        super().__init__()

        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=2,
            batch_first=True,
            bidirectional=True, # Ключевой
        параметр
            dropout=0.2
        )

        # hidden_size * 2 из-за bidirectional
        self.fc = nn.Linear(hidden_size * 2,
                           output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.fc(out[:, -1, :])
        return out

# Обрабатывает последовательность в обоих
направлениях
```

## ◆ 8. Типы задач

| Тип          | Описание                                | Пример              |
|--------------|-----------------------------------------|---------------------|
| Many-to-One  | Последовательность → одно значение      | Sentiment analysis  |
| One-to-Many  | Одно значение → последовательность      | Image captioning    |
| Many-to-Many | Последовательность → последовательность | Machine translation |
| Seq2Seq      | Encoder-Decoder                         | Перевод текста      |

## ◆ 9. Подготовка данных

```
import torch
from torch.nn.utils.rnn import pad_sequence

# Паддинг последовательностей разной длины
sequences = [torch.tensor([1, 2, 3]),
             torch.tensor([4, 5]),
             torch.tensor([6, 7, 8, 9])]

padded = pad_sequence(sequences, batch_first=True,
                      padding_value=0)
# Результат: [[1, 2, 3, 0],
#               [4, 5, 0, 0],
#               [6, 7, 8, 9]]

# Создание батчей для RNN
# Формат: (batch_size, seq_len, input_size)
X = torch.randn(32, 50, 128) # 32 сэмпла, 50
шагов, 128 признаков
```

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Временные ряды
- ✓ Анализ текста
- ✓ Распознавание речи
- ✓ Машинный перевод
- ✓ Предсказание следующего элемента

### ✗ Плохо

- ✗ Очень длинные последовательности (используйте Transformers)
- ✗ Нужна параллелизация (RNN последовательны)
- ✗ Табличные данные без временной зависимости

## ◆ 11. Проблемы и решения

| Проблема                   | Решение                   |
|----------------------------|---------------------------|
| Исчезающий градиент        | Использовать LSTM/GRU     |
| Взрывающийся градиент      | Gradient clipping         |
| Медленное обучение         | Batch processing, GPU     |
| Переобучение               | Dropout, регуляризация    |
| Длинные последовательности | Truncated BPTT, Attention |

```
# Gradient clipping
torch.nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)
```

## ◆ 12. Чек-лист

- [ ] Выбрать тип RNN (LSTM/GRU для большинства задач)
- [ ] Нормализовать входные данные
- [ ] Правильно подготовить последовательности (padding)
- [ ] Использовать batch\_first=True для удобства
- [ ] Добавить dropout между слоями
- [ ] Применить gradient clipping
- [ ] Рассмотреть bidirectional для контекста
- [ ] Для длинных последовательностей — Transformers

### 💡 Объяснение заказчику:

«RNN работает как наша память: читая предложение слово за словом, мы помним предыдущие слова, чтобы понять смысл. Так же RNN запоминает предыдущие элементы последовательности для анализа текущего».



## Биннинг и дискретизация

Январь 2026

### ◆ 1. Зачем нужна дискретизация

- **Простота:** упрощает модель и интерпретацию
- **Нелинейность:** помогает линейным моделям захватывать нелинейные зависимости
- **Выбросы:** снижает влияние выбросов
- **Производительность:** ускоряет некоторые алгоритмы
- **Категоризация:** создает осмысленные группы (возрастные, ценовые)
- **Деревья:** иногда улучшает работу деревьев решений

Биннинг превращает возраст 25 лет в категорию "молодой взрослый" (18-30), делая паттерны более явными.

### ◆ 2. Equal Width Binning

Деление на интервалы одинаковой ширины:

```
import pandas as pd
import numpy as np

# Пример данных
ages = np.array([18, 25, 35, 45, 55, 65, 75, 85])

# Equal width с pandas
age_bins = pd.cut(ages, bins=4)
print(age_bins)
# [(17.933, 35.75], (17.933, 35.75], (35.75, 53.5], ...]

# С метками
age_bins = pd.cut(ages,
                   bins=4,
                   labels=['Young', 'Adult', 'Middle', 'Senior']
                  )

# Получить границы
age_bins = pd.cut(ages, bins=4, retbins=True)
print(age_bins[1]) # границы бинов

# С numpy
bins = np.linspace(ages.min(), ages.max(), 5) # 4 бина
binned = np.digitize(ages, bins)
```

**Плюсы:** Простота, интуитивность

**Минусы:** Может быть неравномерное распределение точек

### ◆ 3. Equal Frequency Binning (Quantile)

Бины с одинаковым количеством наблюдений:

```
# Quantile binning с pandas
age_bins = pd.qcut(ages, q=4) # 4 квартиля
print(age_bins)

# С метками
age_bins = pd.qcut(ages,
                   q=4,
                   labels=['Q1', 'Q2', 'Q3', 'Q4']
                  )

# Получить границы
age_bins, bins = pd.qcut(ages, q=4, retbins=True)
print(bins)

# Или указать квантили вручную
age_bins = pd.qcut(ages,
                   q=[0, 0.25, 0.5, 0.75, 1.0],
                   labels=['25%', '50%', '75%', '100%']
                  )

# С sklearn
from sklearn.preprocessing import KBinsDiscretizer

kbd = KBinsDiscretizer(
    n_bins=4,
    encode='ordinal',
    strategy='quantile'
)
binned = kbd.fit_transform(ages.reshape(-1, 1))
```

**Плюсы:** Равномерное распределение

**Минусы:** Может группировать разные значения в один бин

## ◆ 4. Custom Binning

Определение границ вручную на основе доменных знаний:

```
# Возрастные группы по демографическим стандартам
age_bins = [0, 18, 30, 45, 60, 100]
age_labels = ['Ребенок', 'Молодой', 'Взрослый',
              'Средний', 'Пожилой']

df['age_group'] = pd.cut(
    df['age'],
    bins=age_bins,
    labels=age_labels,
    include_lowest=True
)

# Зарплатные категории
salary_bins = [0, 30000, 60000, 100000, np.inf]
salary_labels = ['Low', 'Medium', 'High', 'Very
High']

df['salary_cat'] = pd.cut(
    df['salary'],
    bins=salary_bins,
    labels=salary_labels
)

# BMI категории (стандарт ВОЗ)
bmi_bins = [0, 18.5, 25, 30, np.inf]
bmi_labels = ['Underweight', 'Normal',
               'Overweight', 'Obese']

df['bmi_cat'] = pd.cut(
    df['bmi'],
    bins=bmi_bins,
    labels=bmi_labels
)
```

## ◆ 5. KBinsDiscretizer (sklearn)

Универсальный инструмент с разными стратегиями:

```
from sklearn.preprocessing import KBinsDiscretizer

# Uniform (equal width)
kbd_uniform = KBinsDiscretizer(
    n_bins=5,
    encode='ordinal', # ordinal, onehot, onehot-
    dense
    strategy='uniform'
)
X_binned = kbd_uniform.fit_transform(X)

# Quantile (equal frequency)
kbd_quantile = KBinsDiscretizer(
    n_bins=5,
    encode='ordinal',
    strategy='quantile'
)
X_binned = kbd_quantile.fit_transform(X)

# K-means (оптимальные границы)
kbd_kmeans = KBinsDiscretizer(
    n_bins=5,
    encode='ordinal',
    strategy='kmeans'
)
X_binned = kbd_kmeans.fit_transform(X)

# Получение границ
print(kbd_uniform.bin_edges_)

# One-hot encoding бинов
kbd_onehot = KBinsDiscretizer(
    n_bins=5,
    encode='onehot-dense'
)
X_onehot = kbd_onehot.fit_transform(X)
```

## ◆ 6. K-means Binning

Использование K-means для оптимального разбиения:

```
from sklearn.cluster import KMeans

# K-means clustering для биннинга
def kmeans_binning(data, n_bins):
    """Биннинг на основе K-means"""
    kmeans = KMeans(n_clusters=n_bins,
                    random_state=42)
    labels = kmeans.fit_predict(data.reshape(-1,
  1))
    centers = kmeans.cluster_centers_.flatten()

    # Сортировка центров для получения порядка
    sorted_indices = np.argsort(centers)
    label_mapping = {old: new for new, old in
                     enumerate(sorted_indices)}
    mapped_labels = np.array([label_mapping[l] for
                             l in labels])

    return mapped_labels, centers[sorted_indices]

# Использование
ages = df['age'].values
binned_ages, centers = kmeans_binning(ages,
                                       n_bins=4)

# Или через KBinsDiscretizer
kbd = KBinsDiscretizer(n_bins=4, encode='ordinal',
                      strategy='kmeans')
binned = kbd.fit_transform(ages.reshape(-1, 1))
```

**Плюсы:** Адаптивные границы, минимизация дисперсии внутри бинов

## ◆ 7. Сравнение стратегий

| Стратегия       | Когда использовать   | Плюсы                | Минусы                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|----------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Equal Width     | Равномерные данные   | Простота             | # После биннинга нужно закодировать категории<br># 1. Ordinal Encoding (порядковые бины)<br>Неравномерное распределение числа: 0, 1, 2, 3<br><code>X_ordinal = binned # от KBinsDiscretizer с encode='ordinal'</code>                                                                                                                                                                                                                                                                                                                                                  |
| Equal Frequency | Скошенные данные     | Равные размеры бинов | Разные ширины интервалов<br># 2. One-Hot Encoding<br><code>from sklearn.preprocessing import OneHotEncoder</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| K-means         | Неравномерные данные | Оптимальность        | Вычислительная сложность<br><code>onehot = OneHotEncoder(sparse=False)</code><br><code>X_onehot = onehot.fit_transform(binned)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Custom          | Есть доменные знания | Интерпретируемость   | Требует экспертизы<br># Или с pandas<br><code>df['age_binned'] = pd.cut(df['age'], bins=4)</code><br><code>df_encoded = pd.get_dummies(df, columns=['age_binned'], prefix='age')</code><br><br># 3. Target Encoding (с осторожностью)<br><code>bin_target_means = df.groupby('age_binned')[['target']].mean()</code><br><code>df['age_target_enc'] = df['age_binned'].map(bin_target_means)</code><br><br># 4. Frequency Encoding<br><code>bin_freq = df['age_binned'].value_counts() / len(df)</code><br><code>df['age_freq'] = df['age_binned'].map(bin_freq)</code> |

```
# Визуальное сравнение
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

for ax, strategy in zip(axes, ['uniform', 'quantile', 'kmeans']):
    kbd = KBinsDiscretizer(n_bins=4, encode='ordinal', strategy=strategy)
    binned = kbd.fit_transform(ages.reshape(-1, 1))
    ax.hist(binned, bins=4)
    ax.set_title(f'{strategy.capitalize()} Binning')
plt.show()
```

## ◆ 8. Encoding после биннинга

## ◆ 9. Binning для деревьев решений

Специальные подходы для tree-based моделей:

```
# Decision tree binning - использует дерево для нахождения оптимальных разбиений
from sklearn.tree import DecisionTreeRegressor

def decision_tree_binning(X, y, max_leaf_nodes=5):
    """Биннинг на основе дерева решений"""
    dt =
DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes,
random_state=42)
    dt.fit(X.reshape(-1, 1), y)

    # Предсказания дерева - это индексы листьев
    binned = dt.apply(X.reshape(-1, 1))

    # Получить границы
    tree = dt.tree_
    thresholds = tree.threshold[tree.feature != -2]

    return binned, np.sort(thresholds)

# Использование
X = df['age'].values
y = df['target'].values
binned, boundaries = decision_tree_binning(X, y,
max_leaf_nodes=4)

# Для XGBoost/LightGBM часто биннинг не нужен!
# Они делают это автоматически и оптимально
```

## ◆ 10. Monotonic Binning

Биннинг с сохранением монотонности:

```
# Для кредитного скоринга важна монотонность
from sklearn.tree import DecisionTreeClassifier

def monotonic_binning(X, y, n_bins=5):
    """
    Биннинг с гарантией монотонности
    (для кредитного скоринга и финансовых задач)
    """
    # Сортировка по X
    sorted_idx = np.argsort(X)
    X_sorted = X[sorted_idx]
    y_sorted = y[sorted_idx]

    # Разбиение на квантили
    bin_edges = np.percentile(
        X_sorted,
        np.linspace(0, 100, n_bins + 1)
    )

    # Проверка монотонности средних в бинах
    bins = pd.cut(X, bins=bin_edges,
                  include_lowest=True, duplicates='drop')
    bin_means = pd.Series(y).groupby(bins).mean()

    # Если не монотонные, объединить соседние
    while not bin_means.is_monotonic_increasing:
        # Найти нарушение и объединить
        diff = bin_means.diff()
        if (diff < 0).any():
            # Слияние бинов
            pass # упрощено

    return bins

# Проверка монотонности
def check_monotonicity(binned_feature, target):
    bin_means =
    pd.Series(target).groupby(binned_feature).mean()
    return bin_means.is_monotonic_increasing or
    bin_means.is_monotonic_decreasing
```

## ◆ 11. Weight of Evidence (WoE)

Специальное кодирование для кредитного скоринга:

```
def calculate_woe(df, feature, target, bins=5):
    """
    Weight of Evidence для бинарной классификации
    WoE = ln(% Good / % Bad)
    """
    # Биннинг
    df['binned'] = pd.qcut(df[feature], q=bins,
                           duplicates='drop')

    # Группировка
    grouped = df.groupby('binned')[target].agg(['sum', 'count'])
    grouped['bad'] = grouped['sum']
    grouped['good'] = grouped['count'] - grouped['sum']

    # Процентные доли
    grouped['bad_pct'] = grouped['bad'] / grouped['bad'].sum()
    grouped['good_pct'] = grouped['good'] / grouped['good'].sum()

    # WoE
    grouped['woe'] = np.log(grouped['good_pct'] / grouped['bad_pct'])

    # IV (Information Value)
    grouped['iv'] = (grouped['good_pct'] - grouped['bad_pct']) * grouped['woe']

    return grouped

# Использование
woe_table = calculate_woe(df, 'age', 'default',
                           bins=5)
print(woe_table[['woe', 'iv']])
print(f"Total IV: {woe_table['iv'].sum():.4f}")

# Применение WoE
woe_map = dict(zip(woe_table.index,
                   woe_table['woe']))
df['age_woe'] = df['binned'].map(woe_map)
```

## ◆ 12. Adaptive Binning

```
# Адаптивный биннинг на основе целевой переменной
from sklearn.tree import DecisionTreeClassifier

def supervised_binning(X, y, max_bins=5,
                      min_samples_bin=50):
    """
    Биннинг под конкретную целевую
    переменную"""
    # Используем дерево решений
    dt = DecisionTreeClassifier(
        max_leaf_nodes=max_bins,
        min_samples_leaf=min_samples_bin,
        random_state=42
    )
    dt.fit(X.reshape(-1, 1), y)

    # Получаем границы из дерева
    tree = dt.tree_
    thresholds = []

    def get_thresholds(node=0):
        if tree.feature[node] != -2: # не лист
            thresholds.append(tree.threshold[node])
            get_thresholds(tree.children_left[node])
            get_thresholds(tree.children_right[node])

        get_thresholds()

    thresholds = sorted(set(thresholds))

    # Создаем бины
    bins = [-np.inf] + thresholds + [np.inf]
    return pd.cut(X, bins=bins)

# Использование
binned = supervised_binning(df['age'].values,
                            df['target'].values)
```

## ◆ 13. Binning в Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import KBinsDiscretizer, StandardScaler
from sklearn.linear_model import LogisticRegression

# Pipeline с биннингом
pipeline = Pipeline([
    ('binning', KBinsDiscretizer(
        n_bins=5,
        encode='onehot-dense',
        strategy='quantile'
    )),
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression())
])

pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)

# ColumnTransformer для разных признаков
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer([
    ('age_binning', KBinsDiscretizer(n_bins=5),
     ['age']),
    ('salary_binning', KBinsDiscretizer(n_bins=4),
     ['salary']),
    ('other', 'passthrough', ['city', 'gender'])
])

X_transformed = ct.fit_transform(X)
```

## ◆ 14. Визуализация биннинга

```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_binning_comparison(data, n_bins=4):
    """Сравнение разных стратегий биннинга"""
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    strategies = ['uniform', 'quantile', 'kmeans']

    # Original distribution
    axes[0, 0].hist(data, bins=30, alpha=0.7)
    axes[0, 0].set_title('Original Distribution')

    # Different binning strategies
    for ax, strategy in zip(axes.flat[1:], strategies):
        kbd = KBinsDiscretizer(
            n_bins=n_bins,
            encode='ordinal',
            strategy=strategy
        )
        binned = kbd.fit_transform(data.reshape(-1, 1))

        ax.hist(binned, bins=n_bins, alpha=0.7)
        ax.set_title(f'{strategy.capitalize()} Binning')

        # Показать границы
        for edge in kbd.bin_edges_[0][1:-1]:
            ax.axvline(edge, color='red',
                       linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

# Использование
plot_binning_comparison(df['age'].values,
                        n_bins=5)
```

## ◆ 15. Best Practices

### ✓ Делать

- ✓ Использовать доменные знания для custom binning
- ✓ Проверять распределение после биннинга
- ✓ Использовать quantile для скошенных данных
- ✓ Тестиовать разные стратегии
- ✓ Сохранять границы бинов для test set
- ✓ Визуализировать результаты

### ✗ Не делать

- ✗ Слишком много бинов (теряется обобщение)
- ✗ Слишком мало бинов (теряется информация)
- ✗ Биннинг для tree-based без причины
- ✗ Игнорировать выбросы при биннинге
- ✗ Забывать про монотонность в финансах

## ◆ 16. Чек-лист

- [ ] Определена цель биннинга (интерпретация, производительность, нелинейность)
- [ ] Выбрана подходящая стратегия (uniform, quantile, kmeans, custom)
- [ ] Определено оптимальное количество бинов
- [ ] Визуализировано распределение до и после
- [ ] Проверена монотонность (если важно)
- [ ] Выбран метод кодирования (ordinal, one-hot, WoE)
- [ ] Границы бинов сохранены для test set
- [ ] Сравнены результаты с/без биннинга
- [ ] Документированы метки бинов

### Объяснение заказчику:

«Биннинг — это группировка похожих значений. Вместо точного возраста (25, 26, 27 лет) мы используем категории (молодые, взрослые, пожилые). Это упрощает анализ и делает модель более устойчивой к шуму в данных».



# Биоинформатика и Computational Biology

Январь 2026

## ◆ 1. Введение

### Основные концепции и применения

- Геномика и секвенирование
- Протеомика и структура белков
- Транскриптомика и экспрессия генов
- ML для анализа биологических данных
- Применение в медицине и фармацевтике

## ◆ 2. Секвенирование ДНК

### Основные концепции и применения

- NGS технологии и base calling
- Выравнивание (alignment) на референсный геном
- Variant calling и поиск мутаций
- DeepVariant от Google для variant calling
- Quality control и фильтрация данных

## ◆ 3. AlphaFold и структура белков

### Основные концепции и применения

- Предсказание 3D структуры белка
- Attention-based архитектура
- Точность близка к экспериментальной
- Революция в structural biology
- Применения в drug discovery

## ◆ 4. RNA-seq анализ

### Основные концепции и применения

- Single-cell RNA-seq
- Differential gene expression
- Clustering клеточных типов
- Trajectory inference
- Scanpy и Seurat для анализа

## ◆ 5. Drug discovery

### Основные концепции и применения

- Virtual screening кандидатов
- QSAR модели
- Молекулярная генерация
- ADMET предсказание
- DeepChem библиотека

## ◆ 6. Геномные варианты

### Основные концепции и применения

- Missense и nonsense мутации
- Regulatory variants
- Предсказание патогенности
- DeepSEA для регуляторных эффектов
- Clinical interpretation

## ◆ 7. Protein-ligand взаимодействие

### Основные концепции и применения

- Docking и binding affinity
- 3D CNN для комплексов
- Virtual screening
- Structure-based drug design
- Оптимизация лидерных соединений

## ◆ 8. Медицинская геномика

### Основные концепции и применения

- Cancer genomics
- Rare disease diagnosis
- Pharmacogenomics
- Персонализированная медицина
- Biomarker discovery

## ◆ 9. Микробиом

### Основные концепции и применения

- Метагеномный анализ
- Taxonomic classification
- Functional profiling
- Связь с заболеваниями
- 16S rRNA sequencing

## ◆ 10. CRISPR и gene editing

### Основные концепции и применения

- Предсказание off-target эффектов
- Guide RNA design
- Оптимизация эффективности
- Therapeutic applications
- Safety assessment

## ◆ 11. Инструменты

### Основные концепции и применения

- Biopython для биоинформатики
- Scikit-bio для анализа
- DeepChem для ML
- PyMOL для визуализации
- RDKit для хемоинформатики



# Биологически вдохновленные архитектуры

17 Январь 2026

## ◆ 1. Введение

**Биологически вдохновленные архитектуры** — нейронные сети, основанные на принципах работы мозга.

- **Мотивация:** мозг — самая эффективная вычислительная система
- **Цель:** заимствовать принципы для улучшения AI
- **Подходы:** от точных моделей до вдохновленных идей
- **Применение:** новые архитектуры, обучение, эффективность

"Мозг использует 20W энергии, современные модели — киловатты!"

## ◆ 2. Биологический нейрон vs Искусственный

| Аспект               | Биологический          | Искусственный             |
|----------------------|------------------------|---------------------------|
| <b>Сигнал</b>        | Спайки (импульсы)      | Непрерывное число         |
| <b>Время</b>         | Важно                  | Обычно игнорируется       |
| <b>Связи</b>         | Пластичные, изменяемые | Фиксированная архитектура |
| <b>Энергия</b>       | Очень эффективный      | Энергозатратный           |
| <b>Обучение</b>      | Локальное, онлайн      | Глобальное (backprop)     |
| <b>Асинхронность</b> | Да                     | Нет (синхронные слои)     |

### ◆ 3. Spiking Neural Networks (SNN)

SNN — нейросети с дискретными спайками вместо непрерывных значений.

**Модель нейрона (LIF - Leaky Integrate-and-Fire):**

- Мембранный потенциал  $v(t)$  аккумулирует входы
- При  $v(t) > \theta$  генерируется спайк
- После спайка  $v(t)$  сбрасывается
- "Leaky" — потенциал постепенно уменьшается

```
# Упрощенная модель LIF нейрона
class LIFNeuron:
    def __init__(self, threshold=1.0, decay=0.9):
        self.threshold = threshold
        self.decay = decay
        self.v = 0.0 # мембранный потенциал

    def step(self, input_current):
        # Интеграция входа
        self.v += input_current

        # Leaky (утечка)
        self.v *= self.decay

        # Генерация спайка
        if self.v >= self.threshold:
            spike = 1.0
            self.v = 0.0 # reset
        else:
            spike = 0.0

        return spike
```

**Преимущества SNN:**

- Энергоэффективность (событийно-ориентированные)
- Временная динамика (естественная обработка последовательностей)
- Биологическая правдоподобность

Биологически вдохновленные архитектуры Cheatsheet — 3 колонки

- Возможность работы на нейроморфном железе

### ◆ 4. Spike-Timing-Dependent Plasticity (STDP)

**STDP** — биологическое правило обучения: "нейроны которые вместе активируются, связываются".

**Правило:**

- Если пре-синаптический спайк перед пост-синаптическим: усиление связи (LTP)
- Если пре-синаптический спайк после пост-синаптического: ослабление связи (LTD)
- Зависит от временной разницы  $\Delta t$

```
# Упрощенная реализация STDP
def stdp_update(pre_spike_time, post_spike_time,
                 weight, A_plus=0.01, A_minus=0.01,
                 tau_plus=20, tau_minus=20):
    dt = post_spike_time - pre_spike_time

    if dt > 0: # LTP (усиление)
        dw = A_plus * np.exp(-dt / tau_plus)
    else: # LTD (ослабление)
        dw = -A_minus * np.exp(dt / tau_minus)

    return weight + dw
```



STDP — один из основных механизмов обучения в мозге

### ◆ 5. Attention механизм и мозг

Attention в трансформерах вдохновлен механизмами внимания в мозге:

**Биологическое внимание:**

- Фокусировка на важной информации
- Подавление нерелевантного
- Топ-даун и боттом-ап процессы
- Связано с активностью в префронтальной коре

**Параллели с Attention:**

- Query-Key-Value ≈ Поиск релевантных воспоминаний
- Softmax ≈ Конкуренция между нейронами
- Multi-head ≈ Разные аспекты внимания

**Отличия от мозга:**

- Мозг использует разреженное внимание
- Внимание в мозге иерархическое
- Временная динамика внимания

## ◆ 6. Иерархическая обработка (визуальная кора)

Визуальная система мозга обрабатывает информацию иерархически — основа для CNN.

### Уровни визуальной коры:

- **V1:** простые признаки (края, ориентация)
- **V2:** комбинации простых признаков
- **V4:** сложные формы, цвет
- **IT:** объекты, лица

### Соответствие в CNN:

- Ранние слои → V1 (края, текстуры)
- Средние слои → V2-V4 (части объектов)
- Поздние слои → IT (целевые объекты)

 Нейроны в V1 реагируют на края под определенными углами — как фильтры в CNN!

## ◆ 7. Рекуррентные связи

В мозге ~80% связей рекуррентные (обратные связи), не только прямые.

### Типы рекуррентности в мозге:

- Локальная: внутри области
- Межобластная: между областями коры
- Топ-даун: от высших областей к низшим

### Функции:

- Контекстная модуляция
- Предсказания (predictive coding)
- Рабочая память
- Усиление сигнала

### В нейросетях:

- RNN, LSTM — упрощенная рекуррентность
- ResNet — skip connections (аналог обратных связей)
- Attention — неявная рекуррентность

## ◆ 8. Predictive Coding

**Predictive Coding** — теория работы мозга: мозг постоянно предсказывает входы и учится на ошибках предсказаний.

### Принцип:

- Высшие уровни предсказывают активность низших
- Низшие уровни передают вверх только ошибки предсказания
- Обучение минимизирует ошибку предсказания

```
# Упрощенная модель predictive coding
class PredictiveCodingLayer:
    def __init__(self):
        self.prediction = None
        self.error = None

    def forward(self, input_signal,
               top_down_pred):
        # Ошибка предсказания
        self.error = input_signal - top_down_pred

        # Текущая активность
        self.prediction = input_signal

        # Передаем ошибку наверх
        return self.error

    def update_weights(self, learning_rate):
        # Обновление для минимизации ошибки
        # (упрощенная версия)
        pass
```

### Связь с ML:

- Variational Autoencoders (VAE)
- Generative models
- Self-supervised learning

## ◆ 9. Neurogenesis и структурная пластиность

Мозг может создавать новые нейроны и изменять структуру связей.

### Neurogenesis:

- Рождение новых нейронов (гиппокамп)
- Происходит в течение всей жизни
- Важен для обучения и памяти

### Структурная пластиность:

- Появление новых синапсов
- Удаление неиспользуемых связей (pruning)
- Реорганизация связей

### Аналоги в ML:

- **Neural Architecture Search (NAS):** автоматический поиск архитектуры
- **Network pruning:** удаление неважных связей
- **Progressive networks:** добавление новых модулей
- **Lottery ticket hypothesis:** нахождение эффективных подсетей

## ◆ 10. Capsule Networks

Вдохновлены обработкой визуальной информации в коре.

### Идея:

- Группы нейронов (капсулы) представляют объекты
- Вектор активации кодирует свойства (поза, масштаб, и т.д.)
- Длина вектора = вероятность присутствия
- Направление вектора = параметры объекта

```
# Упрощенная Capsule
class Capsule(nn.Module):
    def __init__(self, in_caps, out_caps,
                 in_dim, out_dim):
        super().__init__()
        self.W = nn.Parameter(
            torch.randn(out_caps, in_caps,
                       out_dim, in_dim)
        )

    def squash(self, s):
        # Нелинейность для векторов
        norm = torch.norm(s, dim=-1, keepdim=True)
        return (norm ** 2 / (1 + norm ** 2)) * (s
   / norm)

    def forward(self, x):
        # Dynamic routing (упрощено)
        u = torch.einsum('...ji,kjiz->...kz', x,
                        self.W)
        return self.squash(u)
```

### Преимущества:

- Экви-вариантность к трансформациям
- Лучше моделируют иерархии объектов
- Более интерпретируемы

## ◆ 11. Глиальные клетки

Глия составляет ~50% клеток мозга и играет важную роль.

### Функции глии:

- Модуляция синаптической передачи
- Метаболическая поддержка нейронов
- Регуляция внеклеточной среды
- Участие в обучении

### Аналоги в нейросетях:

- **Batch Normalization:** регуляция активаций
- **Neuromodulation layers:** контекстная модуляция
- **Auxiliary networks:** вспомогательные модули

 Глия только недавно признана важной для вычислений

## ◆ 12. Гомеостатическая пластичность

Механизмы стабилизации активности нейронов.

**Принципы:**

- Поддержание целевого уровня активности
- Scaling синаптических весов
- Изменение порогов возбуждения
- Баланс возбуждения/торможения

**В нейросетях:**

- Batch Normalization:** нормализация активаций
- Layer Normalization:** стабилизация обучения
- Weight Normalization:** нормализация весов
- Adaptive learning rates:** адаптация под уровень активности

```
# Synaptic scaling (упрощенно)
def homeostatic_scaling(weights, target_activity,
                        current_activity,
                        tau=0.01):
    # Масштабирование весов для достижения
    # целевой активности
    scaling_factor = target_activity /
    current_activity
    return weights * (1 + tau * (scaling_factor -
    1))
```

## ◆ 13. Oscillations и ритмы мозга

Мозг работает с различными ритмами (theta, alpha, beta, gamma).

**Функции oscillations:**

- Синхронизация удаленных областей
- Временное связывание информации
- Гейтинг информационных потоков
- Координация различных процессов

**Применение в ML:**

- Temporal binding в последовательностях
- Циклические learning rate schedules
- Oscillatory RNNs
- Phase synchronization в многозадачном обучении

## ◆ 14. Working memory (рабочая память)

Временное хранение и манипуляция информацией.

**Свойства:**

- Ограниченнная емкость ( $\sim 7 \pm 2$  элементов)
- Поддерживается рекуррентной активностью
- Предфронтальная кора
- Взаимодействие с долговременной памятью

**В нейросетях:**

- LSTM, GRU:** cell state как рабочая память
- Memory networks:** явная память
- Transformers:** attention как обращение к памяти
- Neural Turing Machines:** внешняя память

```
# Упрощенная модель working memory
class WorkingMemoryCell:
    def __init__(self, capacity=7):
        self.memory = []
        self.capacity = capacity

    def store(self, item, importance):
        self.memory.append((item, importance))
        # Ограничение емкости
        if len(self.memory) > self.capacity:
            # Удаляем наименее важные
            self.memory.sort(key=lambda x: x[1])
            self.memory = self.memory[-self.capacity:]

    def retrieve(self, query):
        # Поиск наиболее релевантного
        scores = [similarity(query, item)
                  for item, _ in self.memory]
        best_idx = np.argmax(scores)
        return self.memory[best_idx][0]
```

## ◆ 15. Sleep и консолидация памяти

Сон критически важен для обучения и памяти.

**Процессы во сне:**

- Replay нейронной активности
- Перенос из гиппокампа в кору
- Synaptic downscaling
- Удаление ненужных связей

**Аналоги в ML:**

- **Experience replay:** повторное обучение на старых данных
- **Knowledge distillation:** перенос знаний
- **Pruning:** удаление ненужных связей
- **Regularization:** downscaling весов

```
# Experience replay (из DQN)
class ReplayBuffer:
    def __init__(self, capacity):
        self.buffer = []
        self.capacity = capacity

    def store(self, experience):
        self.buffer.append(experience)
        if len(self.buffer) > self.capacity:
            self.buffer.pop(0)

    def sample(self, batch_size):
        # "Replay" во время обучения
        return random.sample(self.buffer,
batch_size)

# Offline training = "сон"
for epoch in range(offline_epochs):
    batch = replay_buffer.sample(batch_size)
    train_on_batch(model, batch)
```

## ◆ 16. Локальное обучение

Backpropagation биологически неправдоподобен — нужны локальные правила.

**Проблемы backprop:**

- Требует глобальной информации
- Симметричные прямые/обратные веса
- Разделение фаз (forward/backward)

**Альтернативы:**

- **Feedback Alignment:** случайные обратные веса
- **Target Propagation:** обучение через целевые значения
- **Contrastive Hebbian Learning:** два прохода с разными условиями
- **Equilibrium Propagation:** физический подход

```
# Feedback Alignment (упрощенно)
class FALayer(nn.Module):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.W_forward = nn.Parameter(
            torch.randn(out_features, in_features)
        )
        # Случайные обратные веса (не обучаются!)
        self.W_backward = torch.randn(
            in_features, out_features
        )

    def forward(self, x):
        return torch.matmul(x, self.W_forward.t())

    def backward_pass(self, grad_output):
        # Используем случайные веса для обратного
        # прохода
        return torch.matmul(grad_output,
self.W_backward.t())
```

## ◆ 17. Будущие направления

**Активные области исследований:**

- **Биологически правдоподобное обучение:** альтернативы backprop
- **Энергоэффективность:** спайковые сети на нейроморфном железе
- **Континуальное обучение:** обучение без забывания
- **Embodied AI:** связь с телом и окружением
- **Consciousness models:** моделирование осознания

 Нейронаука и AI взаимно обогащают друг друга



17 Январь 2026

## ◆ 1. Суть метода

- Иерархическая структура:** строит CF-дерево (Clustering Feature Tree)
- Один проход:** обрабатывает данные инкрементально, не требует всех данных в памяти
- Масштабируемость:** работает с миллионами точек
- Компактное представление:** кластеры хранятся как CF тройки ( $N$ ,  $LS$ ,  $SS$ )
- Память эффективно:**  $O(\text{память})$  вместо  $O(N^2)$

## ◆ 2. Базовый код

```
from sklearn.cluster import Birch
import numpy as np

# Базовое использование
model = Birch(
    n_clusters=3,
    threshold=0.5,
    branching_factor=50
)
labels = model.fit_predict(X)

# Инкрементальное обучение
model = Birch(n_clusters=None) # без финального
# кластеризатора
model.partial_fit(X_batch1)
model.partial_fit(X_batch2)
# ...
labels = model.predict(X_test)

# Получить subclusters
subclusters = model.subcluster_centers_
print(f"Subclusters: {len(subclusters)}")
print(f"Final clusters: {len(np.unique(labels))}")
```

## ◆ 3. Ключевые параметры

| Параметр         | Описание                  | Совет                                          |
|------------------|---------------------------|------------------------------------------------|
| n_clusters       | Число финальных кластеров | None для авто, или int/AgglomerativeClustering |
| threshold        | Радиус subcluster         | 0.1-1.0, меньше = больше subclusters           |
| branching_factor | Макс детей в CF узле      | 50 по умолчанию, больше = быстрее              |
| compute_labels   | Вычислять метки при fit   | True по умолчанию                              |
| copy             | Копировать данные         | False для экономии памяти                      |

## ◆ 4. CF дерево и CF тройки

### Clustering Feature (CF):

- N:** число точек в кластере
- LS:** линейная сумма =  $\sum x_i$
- SS:** квадратичная сумма =  $\sum (x_i)^2$

### Свойства CF:

- Аддитивность:  $CF_1 + CF_2 = (N_1+N_2, LS_1+LS_2, SS_1+SS_2)$
- Центроид:  $c = LS/N$
- Радиус:  $R = \sqrt{(SS/N - (LS/N)^2)}$
- Диаметр:  $D = \sqrt{(2\cdot SS/N - 2\cdot (LS/N)^2)}$

### CF дерево:

- Листья содержат subclusters (CF записи)
- Внутренние узлы агрегируют информацию
- Параметр B (branching\_factor) = макс детей
- Параметр T (threshold) = макс радиус subcluster

## ◆ 5. Алгоритм работы

**1. Инициализация:** создать пустое CF дерево

**2. Вставка точки:**

- Найти ближайший leaf entry
- Проверить threshold: можно ли добавить
- Если да — обновить CF, если нет — создать новый entry

**3. Разбиение узла:** если превышен branching\_factor

**4. Финальная кластеризация:** применить агломеративную кластеризацию к subcluster centers

**Фазы BIRCH:**

1. Построение начального CF дерева (один проход)
2. Опционально: сжатие дерева (меньше threshold)
3. Глобальная кластеризация subclusters
4. Опционально: рефайнинг кластеров

## ◆ 6. Выбор threshold

**Влияние threshold:**

- Маленький T → много subclusters → точнее, но медленнее
- Большой T → мало subclusters → быстрее, но грубее

**Подбор через эксперименты:**

```
from sklearn.metrics import silhouette_score

thresholds = [0.1, 0.3, 0.5, 0.7, 1.0]
results = []

for t in thresholds:
    birch = Birch(
        n_clusters=3,
        threshold=t,
        branching_factor=50
    )
    labels = birch.fit_predict(X)
    n_subclusters = len(birch.subcluster_centers_)

    score = silhouette_score(X, labels)
    results.append({
        'threshold': t,
        'subclusters': n_subclusters,
        'score': score
    })
    print(f"T={t}: subclusters={n_subclusters}, score={score:.3f}")

# Выбрать лучший
best = max(results, key=lambda x: x['score'])
```

## ◆ 7. Предобработка данных

**✓ Масштабирование обязательно**

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# BIRCH чувствителен к масштабу
# threshold зависит от единиц измерения
```

**✓ Обработка выбросов**

- Выбросы могут создавать много subclusters
- Рассмотреть предварительную фильтрацию

**✓ Категориальные признаки**

- BIRCH работает только с числовыми данными
- Необходимо кодирование (one-hot, target encoding)

## ◆ 8. Когда использовать

### ✓ Хорошо

- ✓ Большие данные (>100K точек)
- ✓ Ограниченная память
- ✓ Потоковые данные (incremental)
- ✓ Сферические кластеры
- ✓ Нужна быстрая работа

### ✗ Плохо

- ✗ Кластеры произвольной формы
- ✗ Разные плотности кластеров
- ✗ Высокая размерность (>50)
- ✗ Категориальные данные
- ✗ Нужна высокая точность

## ◆ 9. Проблемы и решения

| Проблема                   | Решение                                         |
|----------------------------|-------------------------------------------------|
| Слишком много subclusters  | Увеличить threshold                             |
| Слишком мало subclusters   | Уменьшить threshold                             |
| Плохое качество            | Масштабировать данные, подобрать threshold      |
| Медленная работа           | Увеличить threshold, branching_factor           |
| Переполнение памяти        | Уменьшить branching_factor, увеличить threshold |
| Чувствительность к порядку | Фаза 2: пересборка дерева с меньшим T           |

## ◆ 10. Инкрементальное обучение

```
# Для потоковых данных
from sklearn.cluster import Birch

birch = Birch(
    n_clusters=None, # без финального
    # кластеризатора
    threshold=0.5,
    branching_factor=50
)

# Обрабатывать батчами
batch_size = 1000
for i in range(0, len(X), batch_size):
    X_batch = X[i:i+batch_size]
    birch.partial_fit(X_batch)

    # Можно получить subclusters на любом этапе
    n_sub = len(birch.subcluster_centers_)
    print(f"Batch {i//batch_size}: {n_sub} subclusters")

# Финальная кластеризация
from sklearn.cluster import AgglomerativeClustering
final_clusterer =
AgglomerativeClustering(n_clusters=3)
subclusters = birch.subcluster_centers_
final_labels =
final_clusterer.fit_predict(subclusters)

# Предсказание для новых данных
# (нужно вручную назначить к ближайшему
# subcluster,
# затем к финальному кластеру)
```

## ◆ 11. Метрики оценки

```
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score
)

labels = model.fit_predict(X)
n_clusters = len(np.unique(labels))
n_subclusters = len(model.subcluster_centers_)

print(f"Subclusters: {n_subclusters}")
print(f"Final clusters: {n_clusters}")

if n_clusters > 1:
    sil = silhouette_score(X, labels)
    db = davies_bouldin_score(X, labels)
    ch = calinski_harabasz_score(X, labels)

    print(f"Silhouette: {sil:.3f}")
    print(f"Davies-Bouldin: {db:.3f}")
    print(f"Calinski-Harabasz: {ch:.3f}")
```

## ◆ 12. Сравнение с другими методами

| Метод             | Преимущества BIRCH                         | Недостатки BIRCH              |
|-------------------|--------------------------------------------|-------------------------------|
| K-means           | Быстрее на больших данных, инкрементальный | Только сферические кластеры   |
| DBSCAN            | Линейная сложность, память эффективно      | Не находит произвольные формы |
| Hierarchical      | Быстрее, O(N) vs O(N <sup>2</sup> )        | Менее точная иерархия         |
| MiniBatch K-means | Более точный, иерархия                     | Медленнее                     |

## ◆ 13. Комбинация с другими алгоритмами

### BIRCH + AgglomerativeClustering:

```
from sklearn.cluster import Birch, AgglomerativeClustering

# Сжатие данных с BIRCH
birch = Birch(n_clusters=None, threshold=0.5)
birch.fit(X)
subclusters = birch.subcluster_centers_

# Финальная кластеризация
agg = AgglomerativeClustering(
    n_clusters=3,
    linkage='ward'
)
cluster_labels = agg.fit_predict(subclusters)

# Назначить финальные метки
labels = birch.predict(X)
final_labels = cluster_labels[labels]
```

### BIRCH + DBSCAN:

```
# Для данных с произвольной формой
from sklearn.cluster import DBSCAN

birch = Birch(n_clusters=None, threshold=0.3)
birch.fit(X)
subclusters = birch.subcluster_centers_

# DBSCAN на subclusters
dbSCAN = DBSCAN(eps=0.5, min_samples=2)
cluster_labels = dbSCAN.fit_predict(subclusters)
```

## ◆ 14. Визуализация CF дерева

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 2D проекция
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)

# Точки данных
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_2d[:, 0], X_2d[:, 1],
            c=labels, cmap='viridis', alpha=0.5,
            s=20)
plt.title(f'Final clusters: {len(np.unique(labels))}')

# Subclusters
subclusters_2d =
pca.transform(model.subcluster_centers_)
plt.subplot(1, 2, 2)
plt.scatter(X_2d[:, 0], X_2d[:, 1],
            c='lightgray', alpha=0.3, s=10)
plt.scatter(subclusters_2d[:, 0], subclusters_2d[:, 1],
            c='red', marker='o', s=100,
            edgecolors='black', linewidths=1)
plt.title(f'Subclusters: {len(model.subcluster_centers_)}')

plt.tight_layout()
plt.show()
```

## ◆ 15. Чек-лист

- [ ] Масштабировать данные обязательно
- [ ] Подобрать threshold (grid search)
- [ ] Выбрать branching\_factor (50-100)
- [ ] Определить n\_clusters или использовать None
- [ ] Проверить число subclusters (не слишком много/мало)
- [ ] Оценить качество (silhouette, visual)
- [ ] Для больших данных: использовать partial\_fit
- [ ] Сравнить с MiniBatch K-means

### Объяснение заказчику:

«BIRCH эффективно работает с большими объёмами данных, создавая компактное дерево-представление. Метод обрабатывает данные за один проход, что делает его идеальным для потоковых данных и ситуаций с ограниченной памятью».

## ◆ 16. Полный пример

```
from sklearn.cluster import Birch
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import numpy as np

# Подготовка
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Подбор параметров
best_score = -1
best_params = None

for threshold in [0.3, 0.5, 0.7, 1.0]:
    for n_clust in [3, 5, 7, None]:
        birch = Birch(
            n_clusters=n_clust,
            threshold=threshold,
            branching_factor=50
        )
        labels = birch.fit_predict(X_scaled)

        if len(np.unique(labels)) > 1:
            score = silhouette_score(X_scaled,
                                      labels)
            if score > best_score:
                best_score = score
                best_params = (threshold, n_clust)
                best_model = birch

print(f"Best: T={best_params[0]}, n={best_params[1]}")
print(f"Score: {best_score:.3f}")
print(f"Subclusters: {len(best_model.subcluster_centers_)}")

# Инкрементальное применение
birch_inc = Birch(
    n_clusters=best_params[1],
    threshold=best_params[0]
)
batch_size = 10000
for i in range(0, len(X_scaled), batch_size):

    birch_inc.partial_fit(X_scaled[i:i+batch_size])

    labels_inc = birch_inc.predict(X_scaled)
    print(f"Incremental score:
{silhouette_score(X_scaled, labels_inc):.3f}")
```

# Блендинг (Blending)

 Январь 2026

## ◆ 1. Суть

- **Blending:** комбинирование предсказаний нескольких моделей
- **Holdout set:** отдельный набор для обучения мета-модели
- **Простота:** проще чем стекинг, нет кросс-валидации
- **Эффективность:** часто побеждает в Kaggle
- **Гибкость:** можно комбинировать любые модели

## ◆ 2. Блендинг vs Стекинг

| Характеристика      | Блендинг           | Стекинг         |
|---------------------|--------------------|-----------------|
| <b>Данные</b>       | Train/Holdout/Test | Train/Test + CV |
| <b>Мета-модель</b>  | На holdout         | На out-of-fold  |
| <b>Сложность</b>    | Простая            | Сложная         |
| <b>Переобучение</b> | Меньше риск        | Больше риск     |
| <b>Данные</b>       | Теряем holdout     | Используем все  |

### ◆ 3. Базовый пример

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Разделение данных: train (50%) / holdout (25%) / test (25%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
X_train, X_holdout, y_train, y_holdout =
train_test_split(
    X_temp, y_temp, test_size=0.33,
    random_state=42
)

# Базовые модели
model1 = RandomForestClassifier(n_estimators=100,
random_state=42)
model2 =
GradientBoostingClassifier(n_estimators=100,
random_state=42)

# Обучение на train
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# Предсказания на holdout (для обучения мета-модели)
holdout_pred1 = model1.predict_proba(X_holdout)[:, 1]
holdout_pred2 = model2.predict_proba(X_holdout)[:, 1]
holdout_features = np.column_stack([holdout_pred1,
holdout_pred2])

# Мета-модель (блендер)
meta_model = LogisticRegression()
meta_model.fit(holdout_features, y_holdout)

# Предсказания на test
test_pred1 = model1.predict_proba(X_test)[:, 1]
test_pred2 = model2.predict_proba(X_test)[:, 1]
test_features = np.column_stack([test_pred1,
test_pred2])

# Финальные предсказания
final_predictions =
meta_model.predict(test_features)
accuracy = accuracy_score(y_test,
```

```
final_predictions)
print(f"Blending Accuracy: {accuracy:.4f}")
```

### ◆ 4. Простое усреднение

```
# Простое среднее (простейший блендинг)
def simple_average_blend(predictions_list):
    return np.mean(predictions_list, axis=0)

# Пример
pred1 = model1.predict_proba(X_test)[:, 1]
pred2 = model2.predict_proba(X_test)[:, 1]
pred3 = model3.predict_proba(X_test)[:, 1]

final_pred = simple_average_blend([pred1, pred2,
pred3])
final_class = (final_pred > 0.5).astype(int)

# Взвешенное среднее
def weighted_average_blend(predictions_list,
weights):
    weighted_sum =
np.zeros_like(predictions_list[0])
    for pred, weight in zip(predictions_list,
weights):
        weighted_sum += pred * weight
    return weighted_sum / sum(weights)

# Веса подобраны на holdout
weights = [0.5, 0.3, 0.2]
final_pred = weighted_average_blend([pred1, pred2,
pred3], weights)
```

### ◆ 5. Оптимизация весов

```
from scipy.optimize import minimize

# Функция для оптимизации весов
def optimize_weights(predictions_list, y_true):
    def loss_function(weights):
        # Нормализуем веса
        weights = np.array(weights) /
np.sum(weights)

        # Взвешенное среднее
        blended =
np.zeros_like(predictions_list[0])
        for pred, weight in zip(predictions_list,
weights):
            blended += pred * weight

        # Лог-лосс
        from sklearn.metrics import log_loss
        return log_loss(y_true, blended)

    # Начальные веса (равномерные)
    initial_weights = [1.0] *
len(predictions_list)

    # Ограничения: все веса положительные
    bounds = [(0.0, 1.0)] * len(predictions_list)

    # Оптимизация
    result = minimize(
        loss_function,
        initial_weights,
        method='SLSQP',
        bounds=bounds
    )

    # Нормализованные веса
    optimal_weights = result.x / np.sum(result.x)
    return optimal_weights

# Получение предсказаний на holdout
holdout_preds =
    model1.predict_proba(X_holdout)[:, 1],
    model2.predict_proba(X_holdout)[:, 1],
    model3.predict_proba(X_holdout)[:, 1]
]

# Оптимизация весов
optimal_weights = optimize_weights(holdout_preds,
y_holdout)
print(f"Optimal weights: {optimal_weights}")

# Применение на test
test_preds =
    model1.predict_proba(X_test)[:, 1],
    model2.predict_proba(X_test)[:, 1],
```

```
model3.predict_proba(X_test)[:, 1]
]
final_pred = weighted_average_blend(test_preds,
optimal_weights)
```

## ◆ 6. Rank averaging

```
from scipy.stats import rankdata

# Rank averaging - робастный к выбросам
def rank_average_blend(predictions_list):
    # Ранжируем каждый набор предсказаний
    ranked_predictions = []
    for pred in predictions_list:
        ranked = rankdata(pred) / len(pred)
        ranked_predictions.append(ranked)

    # Усредняем ранги
    return np.mean(ranked_predictions, axis=0)

# Пример
pred1 = model1.predict_proba(X_test)[:, 1]
pred2 = model2.predict_proba(X_test)[:, 1]
pred3 = model3.predict_proba(X_test)[:, 1]

final_pred = rank_average_blend([pred1, pred2, pred3])

# Преобразование обратно в вероятности
# (опционально)
# можно оставить ранги для ranking задач
```

## ◆ 7. Геометрическое среднее

```
# Геометрическое среднее
def geometric_mean_blend(predictions_list):
    product = np.ones_like(predictions_list[0])
    for pred in predictions_list:
        # Избегаем log(0)
        pred_safe = np.clip(pred, 1e-15, 1 - 1e-15)
        product *= pred_safe
    return np.power(product, 1.0 / len(predictions_list))

# Пример
pred1 = model1.predict_proba(X_test)[:, 1]
pred2 = model2.predict_proba(X_test)[:, 1]
pred3 = model3.predict_proba(X_test)[:, 1]

final_pred = geometric_mean_blend([pred1, pred2, pred3])

# Альтернатива через логарифмы (численно
# стабильнее)
def geometric_mean_blend_log(predictions_list):
    log_sum = np.zeros_like(predictions_list[0])
    for pred in predictions_list:
        pred_safe = np.clip(pred, 1e-15, 1 - 1e-15)
        log_sum += np.log(pred_safe)
    return np.exp(log_sum / len(predictions_list))
```

## ◆ 8. Мета-признаки

```
# Создание расширенных мета-признаков
def create_meta_features(base_predictions,
X_original):
    meta_features = []

    # 1. Базовые предсказания
    for pred in base_predictions:
        meta_features.append(pred)

    # 2. Взаимодействия предсказаний
    for i in range(len(base_predictions)):
        for j in range(i + 1, len(base_predictions)):
            # Произведение
            meta_features.append(base_predictions[i] *
base_predictions[j])
            # Разность
            meta_features.append(np.abs(base_predictions[i] -
base_predictions[j]))

    # 3. Статистики
    meta_features.append(np.mean(base_predictions,
axis=0))
    meta_features.append(np.std(base_predictions,
axis=0))
    meta_features.append(np.min(base_predictions,
axis=0))
    meta_features.append(np.max(base_predictions,
axis=0))

    # 4. Оригинальные признаки (опционально)
    # Можно добавить важные исходные признаки

    return np.column_stack(meta_features)

# Использование
holdout_preds = [
    model1.predict_proba(X_holdout)[:, 1],
    model2.predict_proba(X_holdout)[:, 1],
    model3.predict_proba(X_holdout)[:, 1]
]

meta_features_holdout =
create_meta_features(holdout_preds, X_holdout)

# Обучение мета-модели на расширенных признаках
from sklearn.ensemble import
GradientBoostingClassifier
meta_model =
GradientBoostingClassifier(n_estimators=100)
meta_model.fit(meta_features_holdout, y_holdout)
```

## ◆ 9. Многоуровневый блендинг

```
# Двухуровневый блендинг
# Уровень 1: несколько групп моделей
group1_models = [RandomForestClassifier(),
GradientBoostingClassifier()]
group2_models = [LogisticRegression(),
SVC(probability=True)]

# Обучение и предсказания для группы 1
group1_preds_holdout = []
group1_preds_test = []
for model in group1_models:
    model.fit(X_train, y_train)

group1_preds_holdout.append(model.predict_proba(X_holdout[:, 1]))

group1_preds_test.append(model.predict_proba(X_test[:, 1]))

# Блендинг группы 1
group1_blend_holdout =
np.mean(group1_preds_holdout, axis=0)
group1_blend_test = np.mean(group1_preds_test,
axis=0)

# То же для группы 2
group2_preds_holdout = []
group2_preds_test = []
for model in group2_models:
    model.fit(X_train, y_train)

group2_preds_holdout.append(model.predict_proba(X_holdout[:, 1]))

group2_preds_test.append(model.predict_proba(X_test[:, 1]))

group2_blend_holdout =
np.mean(group2_preds_holdout, axis=0)
group2_blend_test = np.mean(group2_preds_test,
axis=0)

# Уровень 2: блендинг групп
meta_features_holdout =
np.column_stack([group1_blend_holdout,
group2_blend_holdout])
meta_features_test =
np.column_stack([group1_blend_test,
group2_blend_test])

meta_model = LogisticRegression()
meta_model.fit(meta_features_holdout, y_holdout)
```

```
final_pred =
meta_model.predict(meta_features_test)
```

## ◆ 10. Блендинг для регрессии

```
from sklearn.ensemble import
RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Базовые модели
model1 = RandomForestRegressor(n_estimators=100,
random_state=42)
model2 =
GradientBoostingRegressor(n_estimators=100,
random_state=42)
model3 = Ridge(alpha=1.0)

# Обучение
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)

# Предсказания на holdout
holdout_pred1 = model1.predict(X_holdout)
holdout_pred2 = model2.predict(X_holdout)
holdout_pred3 = model3.predict(X_holdout)
holdout_features = np.column_stack([holdout_pred1,
holdout_pred2, holdout_pred3])

# Мета-модель (Ridge регрессия для блендинга)
meta_model = Ridge(alpha=0.1)
meta_model.fit(holdout_features, y_holdout)

# Предсказания на test
test_pred1 = model1.predict(X_test)
test_pred2 = model2.predict(X_test)
test_pred3 = model3.predict(X_test)
test_features = np.column_stack([test_pred1,
test_pred2, test_pred3])

# Финальные предсказания
final_predictions =
meta_model.predict(test_features)
rmse = np.sqrt(mean_squared_error(y_test,
final_predictions))
print(f"Blending RMSE: {rmse:.4f}")
```

## ◆ 11. Практические советы

- Размер holdout:** 15-33% от train данных
- Разнообразие моделей:** используйте разные типы (tree, linear, neural)
- Корреляция:** модели должны ошибаться по-разному
- Начните с простого:** простое среднее часто работает хорошо
- Мета-модель:** начните с линейной (Logistic/Ridge)
- Проверка:** валидируйте на отдельном test set
- Время:** блендинг быстрее стекинга

## ◆ 12. Когда использовать

### ✓ Хорошо

- ✓ Соревнования (Kaggle)
- ✓ Множество готовых моделей
- ✓ Ограничено время
- ✓ Простота важнее точности
- ✓ Большой dataset

### ✗ Плохо

- ✗ Очень малый dataset
- ✗ Нужна интерпретируемость
- ✗ Production с жесткими требованиями
- ✗ Нельзя терять данные на holdout

## ◆ 13. Типичные ошибки

- **Утечка данных:** holdout должен быть полностью изолирован
- **Слишком маленький holdout:** < 10% = ненадежные веса
- **Переобучение мета-модели:** используйте регуляризацию
- **Игнорирование корреляции:** похожие модели не улучшают результат
- **Сложная мета-модель:** начните с простой

## ◆ 14. Чек-лист

- [ ] Разделить данные на train/holdout/test
- [ ] Обучить разнообразные базовые модели на train
- [ ] Проверить корреляцию предсказаний моделей
- [ ] Получить предсказания на holdout
- [ ] Обучить мета-модель на holdout
- [ ] Проверить на test set
- [ ] Сравнить с отдельными моделями
- [ ] Попробовать разные методы блендинга
- [ ] Оптимизировать веса если нужно

### 💡 Объяснение заказчику:

«Блендинг — это как консилиум врачей: каждый специалист дает свой диагноз, а главврач (мета-модель) принимает финальное решение, учитывая мнения всех. Мы используем отдельную группу пациентов (holdout), чтобы понять, кому из врачей стоит больше доверять».



17 декабря 2025

## ◆ 1. Суть бустинга

- Последовательное обучение:** каждая новая модель исправляет ошибки предыдущих
- Фокус на сложных примерах:** плохо предсказанные объекты получают больший вес
- Композиция слабых моделей:** в итоге получаем одну сильную модель
- Минимизация функции потерь:** градиентный спуск в пространстве функций

[Данные] → [Модель 1] → Ошибки → [Вес сложных ↑] → [Модель 2] → Ошибки → ... → Финальная композиция

## ◆ 2. Основные алгоритмы

| Алгоритм                 | Особенность                                          | Использование             |
|--------------------------|------------------------------------------------------|---------------------------|
| <b>AdaBoost</b>          | Адаптивное увеличение веса ошибочных объектов        | Бинарная классификация    |
| <b>Gradient Boosting</b> | Оптимизация любой дифференцируемой функции потерь    | Классификация и регрессия |
| <b>XGBoost</b>           | Регуляризованный градиентный бустинг с оптимизациями | Соревнования, production  |
| <b>LightGBM</b>          | Гистограммный метод, скорость и эффективность памяти | Большие данные            |
| <b>CatBoost</b>          | Эффективная работа с категориальными признаками      | Данные с категориями      |

## ◆ 3. Базовый код

```
# XGBoost
import xgboost as xgb
model = xgb.XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6
)
model.fit(X_train, y_train)

# LightGBM
import lightgbm as lgb
model = lgb.LGBMClassifier(
    n_estimators=100,
    learning_rate=0.05,
    num_leaves=31
)

# CatBoost
from catboost import CatBoostClassifier
model = CatBoostClassifier(
    iterations=100,
    learning_rate=0.05,
    depth=6,
    silent=True
)
```

## ◆ 4. Ключевые параметры

| Параметр              | Описание                          | Совет                                  |
|-----------------------|-----------------------------------|----------------------------------------|
| n_estimators          | Число деревьев (итераций)         | 100-1000, с early stopping             |
| learning_rate         | Темп обучения (шаг)               | 0.01-0.3<br>(меньше → больше деревьев) |
| max_depth             | Макс. глубина деревьев            | 3-10 (глубже → сложнее модель)         |
| subsample             | Доля данных для каждого дерева    | 0.7-1.0<br>(Stochastic GB)             |
| colsample_bytree      | Доля признаков для каждого дерева | 0.7-1.0                                |
| reg_alpha, reg_lambda | L1 и L2 регуляризация             | Защита от переобучения                 |
| min_child_weight      | Мин. сумма весов в листе          | Больше → консервативнее                |

## ◆ 6. Проблемы и решения

| Проблема                  | Решение                                                           |
|---------------------------|-------------------------------------------------------------------|
| Переобучение              | Уменьшить max_depth, увеличить регуляризацию, ранняя остановка    |
| Долгое обучение           | Уменьшить n_estimators, увеличить learning_rate, использовать GPU |
| Чувствительность к шуму   | Увеличить min_child_weight, уменьшить learning_rate               |
| Память                    | Использовать LightGBM, уменьшить глубину                          |
| Подбор параметров         | Bayesian optimization, Hyperopt, Optuna                           |
| Несбалансированные классы | Параметр scale_pos_weight, балансировка весов                     |

## ◆ 7. Ранняя остановка (Early Stopping)

- Принцип:** Останавливаем обучение, когда качество на валидации перестает расти
- Параметры:** early\_stopping\_rounds , eval\_set , eval\_metric
- Преимущества:** Автоматический выбор числа деревьев, защита от переобучения

```
# XGBoost с ранней остановкой
model = xgb.XGBClassifier(n_estimators=1000,
                           learning_rate=0.05)
model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    early_stopping_rounds=50,
    verbose=False
)

# LightGBM
model = lgb.LGBMClassifier(n_estimators=1000)
model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    callbacks=[lgb.early_stopping(50)]
)
```

## ◆ 5. Преимущества бустинга

- Высокая точность:** часто лучший результат на табличных данных
- Гибкость:** разные функции потерь, обработка пропусков
- Автоматический отбор признаков:** важность признаков встроена
- Регуляризация:** встроенная защита от переобучения
- Работа с категориями:** особенно CatBoost
- Интерпретируемость:** важность признаков, SHAP значения

## ◆ 8. Сравнение алгоритмов

| Критерий                 | XGBoost | LightGBM     | CatBoost             |
|--------------------------|---------|--------------|----------------------|
| <b>Скорость обучения</b> | Быстро  | Очень быстро | Средне/быстро        |
| <b>Память</b>            | Средне  | Мало         | Средне               |
| <b>Категории</b>         | One-Hot | Гистограммы  | Встроенная обработка |
| <b>Точность</b>          | Высокая | Высокая      | Высокая              |
| <b>Подбор параметров</b> | Сложный | Проще        | Проще                |
| <b>Производство</b>      | Отлично | Хорошо       | Хорошо               |

## ◆ 9. Когда использовать бустинг

### ✓ Хорошо

- ✓ Табличные данные
- ✓ Нужна максимальная точность
- ✓ Соревнования (Kaggle)
- ✓ Смешанные типы признаков
- ✓ Достаточно данных (>10К строк)
- ✓ Ресурсы для настройки параметров

### ✗ Плохо

- ✗ Мало данных (< 1К строк)
- ✗ Требуется мгновенное предсказание
- ✗ Ограниченные вычислительные ресурсы
- ✗ Высокий уровень шума в данных
- ✗ Требуется простая интерпретируемость
- ✗ Онлайн-обучение

## ◆ 10. Важность признаков и интерпретация

- **Gain:** среднее уменьшение impurity при использовании признака
- **Cover:** сколько примеров касается признака
- **Frequency:** как часто признак используется в деревьях
- **SHAP значения:** вклад каждого признака в предсказание для каждого объекта

```
# Важность признаков (XGBoost)
importance = model.feature_importances_
for name, importance in zip(feature_names,
                           importance):
    print(f"{name}: {importance}")

# SHAP значения
import shap
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

## ◆ 11. Чек-лист работы с бустингом

- [ ] Закодировать категории (кроме CatBoost)
- [ ] Масштабировать числовые признаки (опционально, но может помочь)
- [ ] Разделить на train/val/test
- [ ] Настроить early stopping
- [ ] Начать с дефолтных параметров
- [ ] Подобрать learning\_rate и n\_estimators
- [ ] Оптимизировать глубину деревьев
- [ ] Добавить регуляризацию при переобучении
- [ ] Использовать кросс-валидацию для оценки
- [ ] Проанализировать важность признаков
- [ ] Проверить качество на отложенной выборке

## ◆ 12. Типичные ошибки

- **Слишком большой learning\_rate:** модель расходится
- **Слишком много деревьев без early stopping:** переобучение
- **Слишком глубокие деревья:** переобучение, медленная работа
- **Отсутствие валидационной выборки:** нет контроля переобучения
- **Игнорирование категориальных признаков:** потеря информации
- **Неправильная обработка пропусков:** бустинг может сам, но нужно проверять

«Бустинг — это мощный инструмент, который требует аккуратного обращения. Ранняя остановка, правильный learning\_rate и регуляризация — ключ к успеху.»



# Bootstrap методы

17 Январь 2026

## ◆ 1. Основы Bootstrap

- **Цель:** оценка статистик и доверительных интервалов
- **Идея:** ресемплинг (повторная выборка) из данных
- **Преимущество:** не требует предположений о распределении
- **Применение:** оценка uncertainty, валидация моделей
- **Создатель:** Bradley Efron (1979)

**Простыми словами:** представьте, что у вас есть мешок с шариками (ваша выборка). Bootstrap — это многократное вытаскивание шариков с возвратом, чтобы понять разброс результатов.

## ◆ 2. Алгоритм Bootstrap

1. Имеется выборка  $X = \{x_1, x_2, \dots, x_n\}$
2. Создать  $B$  bootstrap-выборок размера  $n$  с возвратом
3. Для каждой выборки вычислить статистику  $\theta^*$
4. Получить распределение  $\{\theta^*_1, \theta^*_2, \dots, \theta^*_B\}$
5. Оценить доверительный интервал, стандартную ошибку

**Важно:** выборка с возвратом — каждый элемент может быть выбран несколько раз.

## ◆ 3. Простой пример

```
import numpy as np
import matplotlib.pyplot as plt

# Исходные данные
data = np.array([23, 25, 27, 29, 31, 33, 35, 37, 39, 41])

# Параметры bootstrap
n_bootstrap = 10000
bootstrap_means = []

# Bootstrap ресемплинг
np.random.seed(42)
for i in range(n_bootstrap):
    # Выборка с возвратом
    sample = np.random.choice(data,
                               size=len(data),
                               replace=True)
    bootstrap_means.append(np.mean(sample))

# Результаты
bootstrap_means = np.array(bootstrap_means)
print(f"Среднее оригинальной выборки: {np.mean(data):.2f}")
print(f"Bootstrap SE: {np.std(bootstrap_means):.2f}")

# 95% доверительный интервал
ci_lower = np.percentile(bootstrap_means, 2.5)
ci_upper = np.percentile(bootstrap_means, 97.5)
print(f"95% CI: [{ci_lower:.2f}, {ci_upper:.2f}]")
```

## ◆ 4. Доверительные интервалы

**Percentile метод:** самый простой

```
def bootstrap_ci_percentile(data, statistic_func,
                             n_bootstrap=10000,
                             confidence=0.95):
    bootstrap_stats = []

    for _ in range(n_bootstrap):
        sample = np.random.choice(data,
                                   size=len(data),
                                   replace=True)
        stat = statistic_func(sample)
        bootstrap_stats.append(stat)

    bootstrap_stats = np.array(bootstrap_stats)
    alpha = 1 - confidence

    ci_lower = np.percentile(bootstrap_stats,
                            100 * alpha / 2)
    ci_upper = np.percentile(bootstrap_stats,
                            100 * (1 - alpha / 2))

    return ci_lower, ci_upper

# Использование
data = np.random.normal(100, 15, 100)
ci = bootstrap_ci_percentile(data, np.mean,
                             n_bootstrap=10000)
print(f"95% CI: [{ci[0]:.2f}, {ci[1]:.2f}]")
```

## ◆ 5. BCa (Bias-Corrected Accelerated)

Более точный метод, учитывающий смещение и асимметрию:

```
from scipy import stats

def bootstrap_bca(data, statistic_func,
                  n_bootstrap=10000,
                  confidence=0.95):
    n = len(data)

    # Bootstrap replicates
    bootstrap_stats = []
    for _ in range(n_bootstrap):
        sample = np.random.choice(data, size=n,
                                   replace=True)
        bootstrap_stats.append(statistic_func(sample))

    bootstrap_stats = np.array(bootstrap_stats)
    theta_hat = statistic_func(data)

    # Bias correction
    z0 = stats.norm.ppf(
        np.mean(bootstrap_stats) < theta_hat)
    )

    # Acceleration (jackknife)
    jackknife_stats = []
    for i in range(n):
        jack_sample = np.delete(data, i)
        jackknife_stats.append(statistic_func(jack_sample))

    jackknife_stats = np.array(jackknife_stats)
    jack_mean = np.mean(jackknife_stats)

    numerator = np.sum((jack_mean -
jackknife_stats)**3)
    denominator = 6 * (np.sum((jack_mean -
jackknife_stats)**2)**1.5)
    a = numerator / denominator if denominator != 0 else 0

    # Adjusted percentiles
    alpha = 1 - confidence
    z_alpha = stats.norm.ppf(alpha / 2)
    z_1_alpha = stats.norm.ppf(1 - alpha / 2)

    p1 = stats.norm.cdf(z0 + (z0 + z_alpha) / (1 -
a * (z0 + z_alpha)))
    p2 = stats.norm.cdf(z0 + (z0 + z_1_alpha) / (1 -
a * (z0 + z_1_alpha)))

    ci_lower = np.percentile(bootstrap_stats, 100 *
* p1)
    ci_upper = np.percentile(bootstrap_stats, 100 *
* p2)

    return ci_lower, ci_upper

# Использование
ci_bca = bootstrap_bca(data, np.median)
print(f"BCa 95% CI: [{ci_bca[0]:.2f}, {ci_bca[1]:.2f}]")
```

## Bootstrap методы Cheatsheet — 3 колонки

```
* p1)
    ci_upper = np.percentile(bootstrap_stats, 100
* p2)

    return ci_lower, ci_upper

# Использование
ci_bca = bootstrap_bca(data, np.median)
print(f"BCa 95% CI: [{ci_bca[0]:.2f}, {ci_bca[1]:.2f}]")
```

## ◆ 6. Стандартная ошибка

```
def bootstrap_se(data, statistic_func,
                 n_bootstrap=10000):
    """Оценка стандартной ошибки через
    bootstrap"""
    bootstrap_stats = []

    for _ in range(n_bootstrap):
        sample = np.random.choice(data,
                                   size=len(data),
                                   replace=True)
        bootstrap_stats.append(statistic_func(sample))

    bootstrap_stats = np.array(bootstrap_stats)
    return np.std(bootstrap_stats)

# Примеры для разных статистик
data = np.random.normal(50, 10, 100)

# SE для среднего
se_mean = bootstrap_se(data, np.mean)
print(f"SE(mean): {se_mean:.3f}")

# SE для медианы
se_median = bootstrap_se(data, np.median)
print(f"SE(median): {se_median:.3f}")

# SE для стандартного отклонения
se_std = bootstrap_se(data, np.std)
print(f"SE(std): {se_std:.3f}")

# SE для 90-го перцентиля
se_p90 = bootstrap_se(data, lambda x:
np.percentile(x, 90))
print(f"SE(p90): {se_p90:.3f}")
```

## ◆ 7. Параметрический Bootstrap

Когда известно распределение данных:

```
from scipy import stats as sp_stats

def parametric_bootstrap(data,
                         distribution='norm',
                         n_bootstrap=10000):
    """
    Параметрический bootstrap:
    подгоняем распределение, затем сэмплируем
    """

    if distribution == 'norm':
        # Оценка параметров
        mu, sigma = sp_stats.norm.fit(data)

        bootstrap_means = []
        for _ in range(n_bootstrap):
            # Генерация из подогнанного
            # распределения
            sample = sp_stats.norm.rvs(mu, sigma,
                                       size=len(data))
            bootstrap_means.append(np.mean(sample))

    return np.array(bootstrap_means)

    elif distribution == 'expon':
        loc, scale = sp_stats.expon.fit(data)

        bootstrap_means = []
        for _ in range(n_bootstrap):
            sample = sp_stats.expon.rvs(loc,
   scale,
   size=len(data))
            bootstrap_means.append(np.mean(sample))

    return np.array(bootstrap_means)

# Использование
data = np.random.normal(100, 15, 50)
boot_stats = parametric_bootstrap(data, 'norm')

ci = np.percentile(boot_stats, [2.5, 97.5])
print(f"Параметрический 95% CI: [{ci[0]:.2f}, {ci[1]:.2f}]")
```

## ◆ 8. Bootstrap для регрессии

```
from sklearn.linear_model import LinearRegression
from sklearn.utils import resample

def bootstrap_regression(X, y, n_bootstrap=1000):
    """Bootstrap для оценки коэффициентов
    регрессии"""
    coef_bootstrap = []

    for _ in range(n_bootstrap):
        # Ресэмплинг наблюдений (пар X, y)
        X_boot, y_boot = resample(X, y,
                                   replace=True,
                                   n_samples=len(X))

        # Обучение модели
        model = LinearRegression()
        model.fit(X_boot, y_boot)

        coef_bootstrap.append(model.coef_)

    coef_bootstrap = np.array(coef_bootstrap)

    # Доверительные интервалы для коэффициентов
    ci_lower = np.percentile(coef_bootstrap, 2.5,
                             axis=0)
    ci_upper = np.percentile(coef_bootstrap, 97.5,
                             axis=0)

    return coef_bootstrap, ci_lower, ci_upper

# Пример
X = np.random.randn(100, 3)
y = 2*X[:, 0] + 3*X[:, 1] - X[:, 2] +
np.random.randn(100)

coefs, ci_low, ci_high = bootstrap_regression(X,
  y)

print("Коэффициенты регрессии (95% CI):")
for i in range(X.shape[1]):
    print(f"  β{i}: [{ci_low[i]:.3f}, {ci_high[i]:.3f}]")
```

## ◆ 9. Валидация моделей ML

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

def bootstrap_model_performance(X, y, model,
                               n_bootstrap=1000):
    """Оценка производительности модели"""
    scores = []

    for _ in range(n_bootstrap):
        # Bootstrap выборка
        indices = np.random.choice(len(X),
                                    size=len(X),
                                    replace=True)
        X_boot = X[indices]
        y_boot = y[indices]

        # OOB (Out-of-Bag) индексы
        oob_indices = np.array(list(
            set(range(len(X))) - set(indices)
        ))

        if len(oob_indices) == 0:
            continue

        # Обучение и оценка
        model.fit(X_boot, y_boot)
        y_pred = model.predict(X[oob_indices])
        score = accuracy_score(y[oob_indices],
                               y_pred)
        scores.append(score)

    scores = np.array(scores)

    return {
        'mean': np.mean(scores),
        'std': np.std(scores),
        'ci_95': np.percentile(scores, [2.5,
                                       97.5])
    }

# Использование
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200,
                           n_features=10,
                           random_state=42)
model = RandomForestClassifier(n_estimators=50)

results = bootstrap_model_performance(X, y, model)
print(f"Accuracy: {results['mean']:.3f} ± {results['std']:.3f}")
```

```
print(f"95% CI: [{results['ci_95'][0]:.3f}, {results['ci_95'][1]:.3f}]")
```

## ◆ 10. Сравнение моделей

```
def bootstrap_model_comparison(X, y, model1,
model2,
                               n_bootstrap=1000):
    """Сравнение двух моделей через bootstrap"""
    diff_scores = []

    for _ in range(n_bootstrap):
        # Bootstrap выборка
        indices = np.random.choice(len(X),
size=len(X),
                               replace=True)
        X_boot = X[indices]
        y_boot = y[indices]

        # OOB оценка
        oob_indices = np.array(list(
            set(range(len(X))) - set(indices)
        ))

        if len(oob_indices) < 10:
            continue

        # Модель 1
        model1.fit(X_boot, y_boot)
        score1 = accuracy_score(y[oob_indices],
model1.predict(X[oob_indices]))

        # Модель 2
        model2.fit(X_boot, y_boot)
        score2 = accuracy_score(y[oob_indices],
model2.predict(X[oob_indices]))

        diff_scores.append(score1 - score2)

    diff_scores = np.array(diff_scores)

    # p-value (двусторонний тест)
    p_value = 2 * min(
        np.mean(diff_scores >= 0),
        np.mean(diff_scores <= 0)
    )

    return {
        'mean_diff': np.mean(diff_scores),
        'ci_95': np.percentile(diff_scores, [2.5,
97.5]),
        'p_value': p_value
    }

# Использование
from sklearn.linear_model import
LogisticRegression
```

## Bootstrap методы Cheatsheet — 3 колонки

```
model1 = RandomForestClassifier(n_estimators=50)
model2 = LogisticRegression()

results = bootstrap_model_comparison(X, y, model1,
model2)
print(f"Разница: {results['mean_diff']:.4f}")
print(f"95% CI: {results['ci_95']}")
```

## ◆ 11. Библиотека scipy.stats

```
from scipy.stats import bootstrap

# Простой способ с scipy
data_tuple = (data,) # данные как tuple

# Функция статистики
def mean_func(sample):
    return np.mean(sample)

# Bootstrap
res = bootstrap(data_tuple, mean_func,
                 n_resamples=10000,
                 confidence_level=0.95,
                 method='percentile',
                 random_state=42)

print(f"95% CI: {res.confidence_interval}")
print(f"SE: {res.standard_error}")
```

## ◆ 12. Когда использовать

### ✓ Хорошо

- ✓ Малые выборки ( $n < 30-50$ )
- ✓ Неизвестное распределение
- ✓ Сложные статистики (медиана, квантили)
- ✓ Оценка uncertainty в ML
- ✓ Сравнение моделей

### ✗ Плохо

- ✗ Очень малые выборки ( $n < 10$ )
- ✗ Сильная зависимость данных
- ✗ Экстремальные значения (используйте другие методы)
- ✗ Когда есть теоретическое распределение

## ◆ 13. Чек-лист

- [ ] Определить статистику для оценки
- [ ] Выбрать число бутстреп-итераций (обычно 1000-10000)
- [ ] Выбрать метод (непараметрический/параметрический)
- [ ] Проверить размер выборки ( $n > 20$ )
- [ ] Выбрать метод CI (percentile/BCa)
- [ ] Визуализировать bootstrap распределение
- [ ] Проверить стабильность результатов
- [ ] Использовать random seed для воспроизводимости

### Объяснение заказчику:

«Bootstrap — это метод оценки надежности наших результатов. Представьте, что мы многократно пересемплируем наши данные (как будто проводим эксперимент снова и снова) и смотрим, насколько стабильны наши выводы. Это помогает понять, можем ли мы доверять нашим результатам или они случайны».



# Теория временных рядов Box-Jenkins

Январь 2026

## ◆ 1. Суть методологии

- **Авторы:** George Box и Gwilym Jenkins (1970)
- **Цель:** систематический подход к построению ARIMA моделей
- **Итеративный процесс:** идентификация → оценка → диагностика
- **ARIMA:** AutoRegressive Integrated Moving Average
- **Применение:** прогнозирование временных рядов

## ◆ 2. Этапы методологии

### 1. Идентификация модели

- Проверка стационарности
- Определение порядков p, d, q
- Анализ ACF и PACF

### 2. Оценка параметров

- Подбор коэффициентов
- Максимизация правдоподобия

### 3. Диагностика

- Анализ остатков
- Тесты адекватности

### 4. Прогнозирование

- Генерация прогнозов
- Доверительные интервалы

## ◆ 3. Модель ARIMA(p,d,q)

ARIMA(p, d, q):

p = порядок авторегрессии (AR)  
d = порядок дифференцирования (I)  
q = порядок скользящего среднего (MA)

Формула:

$$\Phi(B)(1-B)^d y_t = \theta(B)\varepsilon_t$$

где:

$\Phi(B)$  = AR полином  
 $\theta(B)$  = MA полином  
 $B$  = оператор сдвига назад  
 $\varepsilon_t$  = белый шум

## ◆ 4. Проверка стационарности

```
from statsmodels.tsa.stattools import adfuller,
kpss
import pandas as pd

# Augmented Dickey-Fuller тест
def check_stationarity(series):
    # ADF тест (H0: нестационарен)
    adf_result = adfuller(series)
    print(f'ADF Statistic: {adf_result[0]:.4f}')
    print(f'p-value: {adf_result[1]:.4f}')

    # KPSS тест (H0: стационарен)
    kpss_result = kpss(series)
    print(f'KPSS Statistic: {kpss_result[0]:.4f}')
    print(f'p-value: {kpss_result[1]:.4f}')

    if adf_result[1] < 0.05:
        print("✅ Ряд стационарен (ADF)")
    else:
        print("❌ Ряд нестационарен (ADF)")

check_stationarity(df['value'])
```

### Признаки нестационарности:

- Тренд в данных
- Сезонность
- Изменяющаяся дисперсия
- p-value ADF > 0.05

## ◆ 5. Дифференцирование

```
import numpy as np

# Первое дифференцирование (d=1)
df['diff1'] = df['value'].diff()

# Второе дифференцирование (d=2)
df['diff2'] = df['diff1'].diff()

# Сезонное дифференцирование (для SARIMA)
df['seasonal_diff'] = df['value'].diff(12) # для
месячных

# Комбинированное
df['combined'] = df['value'].diff(12).diff()

# Визуализация
import matplotlib.pyplot as plt
fig, axes = plt.subplots(3, 1, figsize=(10, 8))
df['value'].plot(ax=axes[0], title='Original')
df['diff1'].plot(ax=axes[1], title='First
Difference')
df['diff2'].plot(ax=axes[2], title='Second
Difference')
plt.tight_layout()
plt.show()

# Проверка стационарности после дифференцирования
check_stationarity(df['diff1'].dropna())
```

## ◆ 6. ACF и PACF анализ

```
from statsmodels.graphics.tsaplots import
plot_acf, plot_pacf

# Построение ACF и PACF
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# ACF - автокорреляционная функция
plot_acf(df['diff1'].dropna(), lags=40,
ax=axes[0])
axes[0].set_title('ACF')

# PACF - частная автокорреляционная функция
plot_pacf(df['diff1'].dropna(), lags=40,
ax=axes[1])
axes[1].set_title('PACF')

plt.tight_layout()
plt.show()
```

### Правила определения p и q:

| Модель    | ACF                       | PACF                      |
|-----------|---------------------------|---------------------------|
| AR(p)     | Затухает                  | Обрывается после<br>lag p |
| MA(q)     | Обрывается после<br>lag q | Затухает                  |
| ARMA(p,q) | Затухает                  | Затухает                  |

## ◆ 7. Подбор модели ARIMA

```
from statsmodels.tsa.arima.model import ARIMA

# Ручной подбор
model = ARIMA(df['value'], order=(1, 1, 1))
fitted_model = model.fit()

# Вывод результатов
print(fitted_model.summary())

# Автоматический подбор
from pmdarima import auto_arima

auto_model = auto_arima(
    df['value'],
    start_p=0, start_q=0,
    max_p=5, max_q=5,
    d=None, # автоматический выбор d
    seasonal=False,
    stepwise=True,
    suppress_warnings=True,
    error_action='ignore',
    trace=True
)

print(auto_model.summary())
print(f"Best model: ARIMA{auto_model.order}")
```

## ◆ 8. Критерии выбора модели

```
# Сравнение моделей
models_to_test = [
    (1, 1, 1),
    (2, 1, 1),
    (1, 1, 2),
    (2, 1, 2)
]

results = []
for order in models_to_test:
    try:
        model = ARIMA(df['value'], order=order)
        fitted = model.fit()
        results.append({
            'order': order,
            'AIC': fitted.aic,
            'BIC': fitted.bic,
            'HQIC': fitted.hqic
        })
    except:
        pass

# Вывод в DataFrame
results_df = pd.DataFrame(results)
print(results_df.sort_values('AIC'))
```

### Информационные критерии:

- **AIC:**  $-2\log(L) + 2k$  (меньше = лучше)
- **BIC:**  $-2\log(L) + k*\log(n)$  (штраф за параметры)
- **HQIC:** Hannan-Quinn
- Выбирайте модель с минимальным значением

## ◆ 9. Диагностика остатков

```
import scipy.stats as stats

# Получение остатков
residuals = fitted_model.resid

# 1. Тест Ljung-Box (автокорреляция остатков)
from statsmodels.stats.diagnostic import acorr_ljungbox
lb_test = acorr_ljungbox(residuals, lags=10)
print("Ljung-Box test:\n", lb_test)

# 2. Тест на нормальность (Jarque-Bera)
jb_stat, jb_pvalue = stats.jarque_bera(residuals)
print(f"Jarque-Bera: stat={jb_stat:.4f}, p-value={jb_pvalue:.4f}")

# 3. Визуализация
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Остатки во времени
residuals.plot(ax=axes[0, 0], title='Residuals over time')

# ACF остатков
plot_acf(residuals, lags=40, ax=axes[0, 1])

# Гистограмма
axes[1, 0].hist(residuals, bins=30,
edgecolor='black')
axes[1, 0].set_title('Residuals Distribution')

# Q-Q plot
stats.probplot(residuals, dist="norm",
plot=axes[1, 1])

plt.tight_layout()
plt.show()
```

## ◆ 10. Прогнозирование

```
# Прогноз на 12 шагов вперед
forecast_steps = 12
forecast =
fitted_model.forecast(steps=forecast_steps)

# С доверительными интервалами
forecast_result =
fitted_model.get_forecast(steps=forecast_steps)
forecast_df = forecast_result.summary_frame()

# Визуализация
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['value'],
label='Historical')
plt.plot(forecast_df.index, forecast_df['mean'],
label='Forecast', color='red')
plt.fill_between(forecast_df.index,
forecast_df['mean_ci_lower'],
forecast_df['mean_ci_upper'],
alpha=0.3, color='red')
plt.legend()
plt.title('ARIMA Forecast')
plt.show()

# Метрики точности (на тестовой выборке)
from sklearn.metrics import mean_squared_error,
mean_absolute_error
rmse = np.sqrt(mean_squared_error(test['value'],
forecast))
mae = mean_absolute_error(test['value'], forecast)
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

## ◆ 11. SARIMA для сезонности

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# SARIMA(p,d,q)(P,D,Q)m
# m = период сезонности (12 для месячных данных)
model = SARIMAX(
    df['value'],
    order=(1, 1, 1),           # несезонная часть
    seasonal_order=(1, 1, 12),  # сезонная
    # часть
    enforce_stationarity=False,
    enforce_invertibility=False
)

fitted = model.fit()
print(fitted.summary())

# Прогноз
forecast = fitted.forecast(steps=12)

# Автоматический выбор SARIMA
auto_sarima = auto_arima(
    df['value'],
    start_p=0, start_q=0,
    max_p=3, max_q=3,
    seasonal=True,
    m=12,  # период сезонности
    start_P=0, start_Q=0,
    max_P=2, max_Q=2,
    d=None, D=None,
    trace=True,
    stepwise=True
)
```

## ◆ 12. Практические советы

- **d обычно 0, 1 или 2:** больше не требуется
- **Начните с простых моделей:** ARIMA(1,1,1)
- **Используйте auto\_arima** для первичного анализа
- **Проверяйте остатки:** должны быть белым шумом
- **Длина данных:** минимум 50-100 наблюдений
- **Сезонность:** используйте SARIMA
- **Валидация:** out-of-sample тестирование

## ◆ 13. Когда использовать

### ✓ Хорошо

- ✓ Одномерные временные ряды
- ✓ Линейные зависимости
- ✓ Стационарные или приводимые к стационарности
- ✓ Экономические и финансовые данные
- ✓ Короткосрочные прогнозы

### ✗ Плохо

- ✗ Нелинейные зависимости
- ✗ Множество внешних переменных
- ✗ Структурные сдвиги
- ✗ Малый объем данных (< 50)
- ✗ Очень долгосрочные прогнозы

## ◆ 14. Чек-лист Box-Jenkins

- [ ] Визуализировать временной ряд
- [ ] Проверить стационарность (ADF, KPSS)
- [ ] Применить дифференцирование при необходимости
- [ ] Построить ACF и PACF графики
- [ ] Определить порядки p, d, q
- [ ] Подобрать несколько моделей
- [ ] Сравнить по AIC/BIC
- [ ] Проверить остатки (Ljung-Box, нормальность)
- [ ] Выбрать финальную модель
- [ ] Сделать прогноз с доверительными интервалами
- [ ] Валидировать на out-of-sample данных

### 💡 Объяснение заказчику:

«Методология Box-Jenkins — это как систематический рецепт для прогнозирования: сначала готовим данные (делаем стационарными), затем выбираем правильную модель (анализируем графики), проверяем качество (смотрим на ошибки) и только потом делаем прогноз. Это золотой стандарт для временных рядов с 1970-х годов».



## Связь нейросетей и мозга

 Январь 2026

### ◆ 1. Биологический нейрон vs Искусственный нейрон

#### Биологический нейрон:

- Дендриты:** получают сигналы от других нейронов
- Тело клетки:** суммирует входные сигналы
- Аксон:** передает сигнал другим нейронам
- Синапсы:** точки контакта между нейронами
- Потенциал действия:** бинарный сигнал (есть/нет)

#### Искусственный нейрон:

- Входы:** аналог дендритов ( $x_1, x_2, \dots$ )
- Веса:** аналог синаптической силы ( $w_1, w_2, \dots$ )
- Суммирование:**  $z = \sum(w_i x_i) + b$
- Функция активации:** аналог потенциала действия
- Выход:**  $y = f(z)$

### ◆ 2. Сходства между мозгом и нейросетями

- Параллельная обработка:** оба используют распределенные вычисления
- Обучение через опыт:** изменение весов/синапсов
- Иерархическая структура:** от простых к сложным признакам
- Распределенное представление:** информация хранится в паттернах активаций
- Робастность:** устойчивость к шуму и повреждениям
- Обобщение:** способность работать с новыми данными

### ◆ 3. Ключевые различия

| Параметр                   | Биологический мозг | Искусственные нейросети   |
|----------------------------|--------------------|---------------------------|
| <b>Количество нейронов</b> | ~86 млрд           | Миллионы (крупные модели) |
| <b>Синапсы/связи</b>       | ~100 трлн          | Милиарды параметров       |
| <b>Скорость</b>            | ~100 Гц            | ГГц (но последовательно)  |
| <b>Энергопотребление</b>   | ~20 Вт             | кВт-МВт (обучение)        |
| <b>Архитектура</b>         | 3D, рекуррентная   | Слоистая, feed-forward    |
| <b>Сигналы</b>             | Импульсы (спайки)  | Непрерывные значения      |
| <b>Обучение</b>            | Постоянное, online | Batch, offline            |
| <b>Пластичность</b>        | Всегда активна     | Только во время обучения  |

## ◆ 4. Биологически вдохновленные механизмы

### Механизмы из нейронауки в ИИ:

- **Backpropagation:** не биологически правдоподобен, но эффективен
- **Hebbian Learning:** "нейроны, которые активируются вместе, связываются вместе"
- **Spike-Timing-Dependent Plasticity (STDP):** изменение весов зависит от времени спайков
- **Attention:** аналог внимания в мозге
- **Memory networks:** вдохновлены гиппокампом
- **Convolutional layers:** зрительная кора V1

## ◆ 5. Зрительная система мозга и CNN

### Иерархия зрительной коры:

- **V1 (первичная):** простые признаки (линии, края) → Conv слои
- **V2:** комбинации простых признаков → Deeper Conv
- **V4:** формы, текстуры → Mid-level features
- **IT (inferotemporal):** объекты → High-level features

### Принципы CNN из нейронауки:

- **Локальные рецептивные поля:** нейроны реагируют на локальные участки
- **Разделяемые веса:** одинаковые детекторы по всему полю зрения
- **Pooling:** инвариантность к небольшим смещениям

## ◆ 6. Рабочая память и Attention

**Префронтальная кора:** удерживает информацию "в уме"

### Attention механизмы:

- **Selective attention:** фокус на релевантной информации
- **Working memory:** временное хранение и обработка
- **Self-attention в Transformers:** аналог распределенного внимания
- **Визуальное внимание:** saliency maps в нейросетях

**Memory networks:** внешняя память для нейросетей

## ◆ 7. Обучение: мозг vs нейросети

### Мозг:

- **Unsupervised learning:** основной режим обучения
- **Few-shot learning:** обучение с малым числом примеров
- **Continual learning:** постоянное обучение без забывания
- **Transfer learning:** перенос знаний между задачами
- **Meta-learning:** "обучение обучению"

### Современные нейросети:

- **Supervised learning:** основной подход
- **Many-shot learning:** требуют много данных
- **Catastrophic forgetting:** забывают старые задачи
- **Limited transfer:** сложный перенос знаний

## ◆ 8. Spiking Neural Networks (SNN)

### Биологически правдоподобная модель

#### Особенности:

- **Импульсное кодирование:** информация в частоте и времени спайков
- **Асинхронная обработка:** события происходят в разное время
- **Энергоэффективность:** активность только при спайках
- **Temporal coding:** время имеет значение

#### Модели нейронов:

- Leaky Integrate-and-Fire (LIF)
- Izhikevich model
- Hodgkin-Huxley model

```
# Пример LIF нейрона
class LIFNeuron:
    def __init__(self, tau=10.0, threshold=1.0):
        self.tau = tau
        self.threshold = threshold
        self.membrane_potential = 0.0

    def step(self, input_current, dt=1.0):
        # Leaky integration
        self.membrane_potential += (-
            self.membrane_potential + input_current) * dt / self.tau

        # Spike generation
        if self.membrane_potential >=
            self.threshold:
            self.membrane_potential = 0.0
            return True # Spike!
        return False
```

## ◆ 9. Нейроморфные вычисления

### Hardware вдохновленный мозгом

#### Примеры чипов:

- **IBM TrueNorth:** 1M нейронов, 256M синапсов, 70mW
- **Intel Loihi:** 128K нейронов, асинхронный
- **BrainScaleS:** аналоговый нейроморфный
- **SpiNNaker:** 1M ARM cores, real-time

#### Преимущества:

- Крайне низкое энергопотребление
- Параллельная асинхронная обработка
- Event-driven вычисления
- Подходит для edge devices

## ◆ 10. Пластиность и адаптация

### Синаптическая пластиность:

- **Long-Term Potentiation (LTP):** усиление синапсов
- **Long-Term Depression (LTD):** ослабление синапсов
- **Homeostatic plasticity:** стабилизация активности
- **Structural plasticity:** рост новых связей

#### В нейросетях:

- **Weight updates:** аналог LTP/LTD
- **Batch normalization:** аналог homeostatic
- **Dropout:** forced plasticity
- **Architecture search:** структурные изменения

## ◆ 11. Предиктивное кодирование

### Теория работы мозга: мозг постоянно предсказывает входные сигналы

#### Принципы:

- **Top-down predictions:** высокоуровневые предсказания
- **Bottom-up errors:** разница между предсказанием и реальностью
- **Hierarchical processing:** на каждом уровне
- **Минимизация ошибки предсказания:** цель обучения

#### B Deep Learning:

- **Autoencoders:** предсказание входа
- **Predictive coding networks**
- **Self-supervised learning:** предсказание частей входа

## ◆ 12. Что мы еще не понимаем

### Открытые вопросы:

- **Backpropagation в мозге:** как мозг обучается?
- **Consciousness:** откуда берется сознание?
- **Continual learning:** как избежать забывания?
- **Few-shot learning:** как обучаться с малым числом примеров?
- **Common sense reasoning:** как мозг использует здравый смысл?
- **Compositional understanding:** как строить сложные концепции?
- **Causal reasoning:** понимание причинно-следственных связей

## ◆ 13. Будущие направления

- **Hybrid systems:** комбинация символьного и нейронного ИИ
- **Neuromorphic computing:** hardware как мозг
- **Brain-computer interfaces:** прямое взаимодействие
- **Continual learning:** обучение без забывания
- **Meta-learning:** обучение обучению
- **Causal models:** понимание причинности
- **Biological plausibility:** более реалистичные модели

## ◆ 14. Практические применения

Где используются биологически вдохновленные подходы:

- **Computer vision:** CNN из зрительной коры
- **Attention mechanisms:** из механизмов внимания мозга
- **Reinforcement learning:** из дофаминовой системы
- **Memory networks:** из гиппокампа
- **Neuromorphic sensors:** event cameras
- **Edge AI:** энергоэффективные чипы

## ◆ 15. Ключевые выводы

- **Вдохновение ≠ копирование:** ИИ берет идеи, но не копирует точно
- **Разная эффективность:** мозг энергоэффективнее, ИИ быстрее на специфических задачах
- **Дополняющие подходы:** изучение мозга помогает ИИ, ИИ помогает нейронауке
- **Еще много неизвестного:** мозг гораздо сложнее современных нейросетей
- **Будущее в гибридах:** комбинация биологических принципов и инженерных решений

«Мозг — это не компьютер, а нейросеть — не мозг. Но понимание принципов работы мозга вдохновляет создание более эффективного и гибкого ИИ».

 **Canonical Correlation Analysis (CCA)**

## ◆ 1. Основная идея

Находит линейные комбинации двух наборов переменных с максимальной корреляцией.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Deep CCA с PyTorch
import torch
import torch.nn as nn

class DeepCCAModel(nn.Module):
    def __init__(self, input_dim1, input_dim2, latent_dim):
        super().__init__()
        self.encoder1 = nn.Sequential(
            nn.Linear(input_dim1, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )
        self.encoder2 = nn.Sequential(
            nn.Linear(input_dim2, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )

    def forward(self, x1, x2):
        return self.encoder1(x1), self.encoder2(x2)

model = DeepCCAModel(50, 30, 10)
optimizer = torch.optim.Adam(model.parameters())

for epoch in range(100):
    z1, z2 = model(X1_batch, X2_batch)
    # Maximize correlation between z1 and z2
    loss = -torch.trace(z1.T @ z2) / len(z1)
```

## Canonical Correlation Analysis (CCA) — Cheatsheet — 3 колонки

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

### 💡 Когда использовать:

*Этот метод особенно эффективен когда нужно находить линейные комбинации двух наборов переменных с максимальной корреляцией....*

## ◆ 2. Математическая формулировка

$\max \text{corr}(X^*a, Y^*b)$  где  $a$  и  $b$  — канонические веса.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Regularized CCA
from sklearn.cross_decomposition import CCA
import numpy as np

# Ridge CCA для высокоразмерных данных
class RidgeCCA:
    def __init__(self, n_components, reg=1e-05):
        self.n_components = n_components
        self.reg = reg

    def fit(self, X, Y):
        # Добавляем регуляризацию к ковариациям
        C_xx = X.T @ X / len(X) + self.reg * np.eye(len(X))
        C_yy = Y.T @ Y / len(Y) + self.reg * np.eye(len(Y))
        C_xy = X.T @ Y / len(X)

        # Решаем обобщённую задачу на собственные векторы
        # (упрощённая версия)
        self.cca = CCA(n_components=self.n_components)
        self.cca.fit(X, Y)
        return self

rcca = RidgeCCA(n_components=5)
rcca.fit(X_train, Y_train)
```

### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно  $\max \text{corr}(x^*a, y^*b)$  где  $a$  и  $b$  — канонические веса....

### ◆ 3. Canonical Variables

Канонические переменные  $U = Xa$  и  $V = Yb$  максимально коррелируют.

#### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# CCA с sklearn
from sklearn.cross_decomposition import CCA
import numpy as np

X = np.random.randn(100, 10)
Y = np.random.randn(100, 8)

cca = CCA(n_components=3)
cca.fit(X, Y)

X_c, Y_c = cca.transform(X, Y)

# Вычисляем канонические корреляции
correlations = [np.corrcoef(X_c[:, i], Y_c[i])[0, 1] for i in range(3)]
print(f"Корреляции: {correlations}")
```

#### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно канонические переменные  $u = xa$  и  $v = yb$  максимально коррелируют....

### ◆ 4. Applications

Мультимодальное обучение, связь между различными представлениями данных.

#### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Deep CCA с PyTorch
import torch
import torch.nn as nn

class DeepCCAModel(nn.Module):
    def __init__(self, input_dim1, input_dim2, latent_dim):
        super().__init__()
        self.encoder1 = nn.Sequential(
            nn.Linear(input_dim1, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )
        self.encoder2 = nn.Sequential(
            nn.Linear(input_dim2, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )

    def forward(self, x1, x2):
        return self.encoder1(x1), self.encoder2(x2)

model = DeepCCAModel(50, 30, 10)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for epoch in range(100):
    z1, z2 = model(X1_batch, X2_batch)
    # Maximize correlation between z1 and z2
    loss = -torch.trace(z1.T @ z2) / len(z1)
```

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

### Когда использовать:

Этот метод особенно эффективен когда нужно мультимодальное обучение, связь между различными представлениями данных....

## ◆ 5. CCA vs PCA

PCA: максимизирует variance. CCA:  
максимизирует correlation между двумя наборами.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Regularized CCA
from sklearn.cross_decomposition import CCA
import numpy as np

# Ridge CCA для высокоразмерных данных
class RidgeCCA:
    def __init__(self, n_components, reg=1e-05):
        self.n_components = n_components
        self.reg = reg

    def fit(self, X, Y):
        # Добавляем регуляризацию к ковариантной матрице
        C_xx = X.T @ X / len(X) + self.reg * np.eye(len(X))
        C_yy = Y.T @ Y / len(Y) + self.reg * np.eye(len(Y))
        C_xy = X.T @ Y / len(X)

        # Решаем обобщённую задачу на собственные векторы
        # (упрощённая версия)
        self.cca = CCA(n_components=self.n_components)
        self.cca.fit(X, Y)
        return self

rcca = RidgeCCA(n_components=5)
rcca.fit(X_train, Y_train)
```

### Когда использовать:

Этот метод особенно эффективен когда нужно rcca: максимизирует variance. cca: максимизирует correlation между двумя наборами....

## ◆ 6. Kernel CCA

Нелинейное расширение CCA с использованием kernel trick.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# CCA с sklearn
from sklearn.cross_decomposition import CCA
import numpy as np

X = np.random.randn(100, 10)
Y = np.random.randn(100, 8)

cca = CCA(n_components=3)
cca.fit(X, Y)

X_c, Y_c = cca.transform(X, Y)

# Вычисляем канонические корреляции
correlations = [np.corrcoef(X_c[:, i], Y_c[:, i])[0, 1] for i in range(3)]
print(f"Корреляции: {correlations}")
```

### Когда использовать:

*Этот метод особенно эффективен когда нужно нелинейное расширение cca с использованием kernel trick....*

## Canonical Correlation Analysis (CCA) — Cheatsheet — 3 колонки

## ◆ 7. Deep CCA

Нейросетевой подход: две сети учат представления с максимальной корреляцией.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Deep CCA с PyTorch
import torch
import torch.nn as nn

class DeepCCAModel(nn.Module):
    def __init__(self, input_dim1, input_dim2, latent_dim):
        super().__init__()
        self.encoder1 = nn.Sequential(
            nn.Linear(input_dim1, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )
        self.encoder2 = nn.Sequential(
            nn.Linear(input_dim2, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )

    def forward(self, x1, x2):
        return self.encoder1(x1), self.encoder2(x2)

model = DeepCCAModel(50, 30, 10)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for epoch in range(100):
    X1_batch, X2_batch = get_batches()
    z1, z2 = model(X1_batch, X2_batch)
    # Maximize correlation between z1 and z2
    loss = -torch.trace(z1.T @ z2) / len(z1)
```

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

### 💡 Когда использовать:

*Этот метод особенно эффективен когда нужно нейросетевой подход: две сети учат представления с максимальной корреляцией....*

## ◆ 8. Regularization

Ridge CCA для стабильности при  $p > n$  (больше признаков чем объектов).

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Regularized CCA
from sklearn.cross_decomposition import CCA
import numpy as np

# Ridge CCA для высокоразмерных данных
class RidgeCCA:
    def __init__(self, n_components, reg=1e-05):
        self.n_components = n_components
        self.reg = reg

    def fit(self, X, Y):
        # Добавляем регуляризацию к ковариационной матрице
        C_xx = X.T @ X / len(X) + self.reg * np.eye(len(X))
        C_yy = Y.T @ Y / len(Y) + self.reg * np.eye(len(Y))
        C_xy = X.T @ Y / len(X)

        # Решаем обобщённую задачу на собственные векторы
        # (упрощённая версия)
        self.cca = CCA(n_components=self.n_components)
        self.cca.fit(X, Y)
        return self

rcca = RidgeCCA(n_components=5)
rcca.fit(X_train, Y_train)
```

### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно ridge cca для стабильности при  $p > n$  (больше признаков чем объектов)....

## ◆ 9. Interpretation

Канонические веса показывают важность исходных переменных.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# CCA с sklearn
from sklearn.cross_decomposition import CCA
import numpy as np

X = np.random.randn(100, 10)
Y = np.random.randn(100, 8)

cca = CCA(n_components=3)
cca.fit(X, Y)

X_c, Y_c = cca.transform(X, Y)

# Вычисляем канонические корреляции
correlations = [np.corrcoef(X_c[:, i], Y_c[:, i])[0, 1] for i in range(3)]
print(f"Корреляции: {correlations}")
```

### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно канонические веса показывают важность исходных переменных....

## ◆ 10. Assumptions

Линейность связей, нормальность распределений (для статистических тестов).

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Deep CCA с PyTorch
import torch
import torch.nn as nn

class DeepCCAModel(nn.Module):
    def __init__(self, input_dim1, input_dim2, latent_dim):
        super().__init__()
        self.encoder1 = nn.Sequential(
            nn.Linear(input_dim1, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )
        self.encoder2 = nn.Sequential(
            nn.Linear(input_dim2, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim)
        )

    def forward(self, x1, x2):
        return self.encoder1(x1), self.encoder2(x2)

model = DeepCCAModel(50, 30, 10)
optimizer = torch.optim.Adam(model.parameters())

for epoch in range(100):
    z1, z2 = model(X1_batch, X2_batch)
    # Maximize correlation between z1 and z2
    loss = -torch.trace(z1.T @ z2) / len(z1)
```

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

### 💡 Когда использовать:

*Этот метод особенно эффективен когда нужно линейность связей, нормальность распределений (для статистических тестов)....*

## ◆ 11. Use Cases

Связь генетики и фенотипов, текст-изображение matching, мультимодальный retrieval.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# Regularized CCA
from sklearn.cross_decomposition import CCA
import numpy as np

# Ridge CCA для высокоразмерных данных
class RidgeCCA:
    def __init__(self, n_components, reg=1e-05):
        self.n_components = n_components
        self.reg = reg

    def fit(self, X, Y):
        # Добавляем регуляризацию к ковариантным матрицам
        C_xx = X.T @ X / len(X) + self.reg * np.eye(len(X))
        C_yy = Y.T @ Y / len(Y) + self.reg * np.eye(len(Y))
        C_xy = X.T @ Y / len(X)

        # Решаем обобщённую задачу на собственные векторы
        # (упрощённая версия)
        self.cca = CCA(n_components=self.n_components)
        self.cca.fit(X, Y)
        return self

rcca = RidgeCCA(n_components=5)
rcca.fit(X_train, Y_train)
```

### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно связь генетики и фенотипов, текст-изображение matching, мультимодальный retrieval....

## ◆ 12. Implementation

sklearn.cross\_decomposition.CCA, rcca package, custom PyTorch implementations.

### Детали реализации:

- Математические основы и формулы
- Алгоритмы и оптимизация
- Параметры и их влияние на результат
- Типичные проблемы и решения
- Сравнение с альтернативными подходами

```
# CCA с sklearn
from sklearn.cross_decomposition import CCA
import numpy as np

X = np.random.randn(100, 10)
Y = np.random.randn(100, 8)

cca = CCA(n_components=3)
cca.fit(X, Y)

X_c, Y_c = cca.transform(X, Y)

# Вычисляем канонические корреляции
correlations = [np.corrcoef(X_c[:, i], Y_c[...]) for i in range(3)]
print(f"Корреляции: {correlations}")
```

### 💡 Когда использовать:

Этот метод особенно эффективен когда нужно sklearn.cross\_decomposition.cca, rcca package, custom pytorch implementations....



# Канонический корреляционный анализ (CCA)

4 января 2026

## ◆ 1. Суть

- **Цель:** найти связи между двумя наборами признаков ( $X$  и  $Y$ )
- **Как работает:** ищет линейные комбинации  $X$  и  $Y$  с максимальной корреляцией
- **Результат:** канонические переменные (компоненты) для обоих наборов
- **Пример:** связать демографию (возраст, доход) с покупками (категории товаров)

## ◆ 2. Базовый код

```
from sklearn.cross_decomposition import CCA
cca = CCA(n_components=2)
cca.fit(X, Y)

# Трансформировать оба набора
X_c, Y_c = cca.transform(X, Y)

# Канонические корреляции
import numpy as np
correlations = [np.corrcoef(X_c[:, i], Y_c[:, i])[0, 1]
                 for i in range(2)]
print(correlations) # [0.85, 0.62]
```

## ◆ 3. Параметры

| Параметр     | Описание                           |
|--------------|------------------------------------|
| n_components | Число канонических компонент       |
| scale        | Масштабировать данные (True/False) |
| max_iter     | Макс. итераций (по умолчанию 500)  |
| tol          | Критерий сходимости (1e-6)         |

## ◆ 4. CCA vs PCA vs LDA

| Метод | Входные данные | Цель                    |
|-------|----------------|-------------------------|
| PCA   | $X$            | Максимальная дисперсия  |
| LDA   | $X, labels$    | Разделение классов      |
| CCA   | $X, Y$         | Максимальная корреляция |

## ◆ 5. Применения

- **Мультимодальные данные:** связь текста и изображений
- **Нейронаука:** связь активности мозга и поведения
- **Генетика:** связь генотипа и фенотипа
- **Маркетинг:** связь характеристик клиента и покупок
- **Финансы:** связь экономических показателей
- **Feature extraction:** создание совместных признаков

## ◆ 6. Предобработка

```
from sklearn.preprocessing import StandardScaler
# Масштабировать оба набора
scaler_X = StandardScaler()
scaler_Y = StandardScaler()

X_scaled = scaler_X.fit_transform(X)
Y_scaled = scaler_Y.fit_transform(Y)

cca = CCA(n_components=3)
X_c, Y_c = cca.fit_transform(X_scaled, Y_scaled)
```

⚠️ Масштабирование критично для CCA!

## ◆ 7. Интерпретация результатов

```
# Веса (loadings) для первой компоненты
print("X weights:", cca.x_weights_[:, 0])
print("Y weights:", cca.y_weights_[:, 0])

# Корреляции исходных признаков с компонентами
X_c, Y_c = cca.transform(X, Y)
X_loadings = np.corrcoef(X.T, X_c[:, 0].T)[-1, :-1]
Y_loadings = np.corrcoef(Y.T, Y_c[:, 0].T)[-1, :-1]
```

## ◆ 8. Выбор числа компонент

### Метод 1: По порогу корреляции

```
# Вычислить канонические корреляции
cca_full = CCA(n_components=min(X.shape[1],
Y.shape[1]))
X_c, Y_c = cca_full.fit_transform(X, Y)

correlations = [np.corrcoef(X_c[:, i], Y_c[:, i])
[0, 1]
for i in range(X_c.shape[1])]

# Выбрать компоненты с корреляцией > 0.3
n_comp = sum(c > 0.3 for c in correlations)
```

## ◆ 9. Kernel CCA

### Нелинейное расширение CCA

```
# Нет прямой реализации в sklearn
# Используйте библиотеки:
# - mvlearn.embed.KCCA
# - pyrccca (Regularized CCA)

# Пример концепта:
from sklearn.kernel_approximation import
RBFSampler

rbf = RBFSampler(gamma=1, n_components=100)
X_rbf = rbf.fit_transform(X)
Y_rbf = rbf.fit_transform(Y)

cca = CCA(n_components=5)
X_c, Y_c = cca.fit_transform(X_rbf, Y_rbf)
```

## ◆ 10. Regularized CCA

Для случаев: мало данных или много признаков

```
# Использовать PLSCanonical с регуляризацией
from sklearn.cross_decomposition import
PLSCanonical

pls = PLSCanonical(n_components=2)
X_c, Y_c = pls.fit_transform(X, Y)

# Или использовать pyrcca библиотеку
# from pyrcca import CCA
# cca = CCA(regularization=0.1)
# cca.train([X, Y])
```

## ◆ 11. Визуализация

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# График первой канонической компоненты
axes[0].scatter(X_c[:, 0], Y_c[:, 0], alpha=0.5)
axes[0].set_xlabel('X canonical 1')
axes[0].set_ylabel('Y canonical 1')
axes[0].set_title(f'Correlation: {correlations[0]:.3f}')

# График канонических корреляций
axes[1].bar(range(len(correlations)),
correlations)
axes[1].set_xlabel('Component')
axes[1].set_ylabel('Correlation')
axes[1].set_title('Canonical Correlations')
plt.tight_layout()
plt.show()
```

## ◆ 12. Когда использовать

### ✓ Хорошо

- ✓ Два набора связанных признаков
- ✓ Изучение взаимосвязей
- ✓ Достаточно данных ( $n > p$ )
- ✓ Линейные зависимости

### ✗ Плохо

- ✗ Один набор признаков ( $\rightarrow$  PCA)
- ✗ Мало данных, много признаков ( $\rightarrow$  RCCA)
- ✗ Нелинейные связи ( $\rightarrow$  Kernel CCA)
- ✗ Задача классификации ( $\rightarrow$  LDA)

## ◆ 13. Статистическая значимость

```
# Проверка значимости канонических корреляций
# Используйте статистические тесты

from scipy import stats

# Для каждой компоненты
n = len(X)
p, q = X.shape[1], Y.shape[1]
r = correlations[0]

# Приближенный тест
chi2 = -(n - 1 - 0.5*(p + q + 1)) * np.log(1 -
r**2)
df = p * q
p_value = 1 - stats.chi2.cdf(chi2, df)
print(f'p-value: {p_value}')
```

## ◆ 14. Частые ошибки

- **✗ Забыли масштабировать** → неправильные веса
- **✗  $n < p$  или  $n < q$**  → переобучение, используйте регуляризацию
- **✗ Слишком много компонент** → шум
- **✗ Игнорирование мультиколлинеарности** → нестабильность
- **✓ Правильно:** масштабирование → выбор компонент → проверка значимости

## ◆ 15. Связанные методы

| Метод        | Особенность                           |
|--------------|---------------------------------------|
| PLS          | Partial Least Squares, предсказание Y |
| PLSCanonical | Каноническая версия PLS               |
| Kernel CCA   | Нелинейное расширение                 |
| RCCA         | Regularized CCA                       |
| Deep CCA     | Нейросетевая версия                   |

## ◆ 17. Объяснение заказчику

«CCA находит скрытые связи между двумя группами данных. Например, связывает демографические данные клиентов (возраст, доход, образование) с их покупательским поведением (категории, частота, суммы). Показывает, какие комбинации демографии наиболее связаны с какими паттернами покупок».

## ◆ 16. Чек-лист

- [ ] Убедиться, что есть два набора признаков
- [ ] Проверить размерности:  $n > \max(p, q)$
- [ ] Масштабировать оба набора
- [ ] Выбрать число компонент
- [ ] Проверить канонические корреляции
- [ ] Проверить статистическую значимость
- [ ] Интерпретировать веса компонент

# Capsule Networks (CapsNets)

## 1. Мотивация

### Проблемы CNN:

- Pooling теряет информацию:** где именно находится feature
- Нет понимания части-целое:** не моделирует иерархические отношения
- View-point variation:** плохо работает при изменении ракурса
- Adversarial vulnerability:** легко обмануть

**Идея Hinton:** использовать группы нейронов (capsules) для представления объектов и их свойств

**Capsule:** вектор активаций, где длина = вероятность существования entity, направление = properties (поза, освещение и т.д.)

## 2. Capsule — базовая единица

**Определение:** капсула = группа нейронов, выход которых — вектор активаций

### Компоненты вектора:

- Length ( $\|v\|$ ):** вероятность существования entity [0,1]
- Direction:** instantiation parameters (позиция, ориентация, размер и т.д.)

**Squashing function:** нелинейность для нормализации длины

$$v_j = (\|s_j\|^2 / (1 + \|s_j\|^2)) \cdot (s_j / \|s_j\|)$$

где  $s_j$  — взвешенная сумма входов

**Свойства:** короткие векторы  $\rightarrow 0$ , длинные  $\rightarrow 1$ , направление сохраняется

## 3. Dynamic Routing

**Проблема:** как капсулы нижнего уровня посылают выход капсулам верхнего?

**Routing by agreement:** итеративный процесс определения coupling coefficients

### Алгоритм routing:

Для каждой капсулы  $i$  на уровне  $l$ :  
Вычислить prediction vectors:  
 $\hat{u}_{j|i} = W_{ij} \cdot u_i$

Инициализировать  $b_{ij} = 0$

Для  $r$  итераций routing:  
 $c_{ij} = \text{softmax}(b_{ij})$  # coupling coeffs  
 $s_j = \sum_i c_{ij} \hat{u}_{j|i}$  # взвеш. сумма  
 $v_j = \text{squash}(s_j)$  # выход  
 капсулы  
 $b_{ij} += \hat{u}_{j|i} \cdot v_j$  # обновить routing

**Интуиция:** если  $u_i$  "согласен" с  $v_j$  (большое скалярное произведение), увеличить  $c_{ij}$

## 4. Архитектура CapsNet

**Слои (оригинальная CapsNet для MNIST):**

1. **Conv1:** обычная свёртка  $9 \times 9$ , 256 каналов, ReLU
2. **PrimaryCaps:**  $32 \text{ channels} \times 8\text{D capsules} = 256 \text{ capsules}$
3. **DigitCaps:**  $10 \text{ классов} \times 16\text{D capsules} = 10 \text{ capsules}$
4. **Reconstruction:** декодер для регуляризации

**PrimaryCaps:** применить convolution для создания векторных выходов

**DigitCaps:** routing между PrimaryCaps и DigitCaps для классификации

## 5. Loss функция

**Margin loss для классификации:**

$$L_k = T_k \cdot \max(0, m^+ - \|v_k\|)^2 + \lambda \cdot (1-T_k) \cdot \max(0, \|v_k\| - m^-)^2$$

где:

- $T_k = 1$  если класс  $k$  присутствует
- $m^+ = 0.9$ ,  $m^- = 0.1$  (пороги)
- $\lambda = 0.5$  (down-weighting)

**Цель:** длина капсулы класса  $k$  должна быть близка к 1 если  $k$  присутствует, к 0 иначе

**Reconstruction loss:** дополнительная регуляризация через decoder

$$L_{recon} = \|image - reconstruction\|^2$$

**Total loss:**  $L = L_{margin} + \alpha \cdot L_{recon}$ , где  $\alpha = 0.0005$

## 6. Decoder для регуляризации

**Цель:** заставить капсулы выучить значимые representations

**Архитектура:**

- Вход: 16D вектор правильного класса
- Fully connected:  $512 \rightarrow 1024 \rightarrow 784 (28 \times 28)$
- Выход: реконструкция входного изображения

**Обучение:** минимизировать MSE между исходным и реконструированным изображением

**Эффект:** капсулы учатся кодировать позицию, размер, ориентацию — информацию, необходимую для реконструкции

## 7. Преимущества CapsNets

**Equivariance к трансформациям:** если объект повернуть, изменяется направление вектора, не вероятность

**View-point invariance:** лучше работает при изменении ракурса

**Part-whole relationships:** иерархическое представление

**Меньше параметров:** по сравнению с глубокими CNN для аналогичной задачи

**Лучше на sparse data:** эффективнее учится на малых датасетах

**Robustness:** более устойчив к adversarial attacks

## 8. Недостатки и ограничения

**Вычислительная сложность:** routing требует многократных forward passes

**Медленное обучение:** значительно медленнее CNN

**Не работает на сложных датасетах:** MNIST, smallNORB OK; ImageNet, COCO — плохо

**Routing нестабилен:** может требовать тщательной инициализации

**Ограниченные ресурсы:** высокие требования к памяти

**Мало исследований:** относительно новая технология, мало best practices

## 9. Реализация на PyTorch

```

import torch
import torch.nn as nn
import torch.nn.functional as F

class PrimaryCapsule(nn.Module):
    def __init__(self, in_channels,
out_channels,
dim_caps, kernel_size,
stride):
        super().__init__()
        self.dim_caps = dim_caps
        self.conv = nn.Conv2d(
            in_channels,
            out_channels * dim_caps,
            kernel_size=kernel_size,
            stride=stride
        )

    def forward(self, x):
        # x: [batch, in_channels, H, W]
        out = self.conv(x)
        batch, _, H, W = out.shape
        # Reshape to [batch, num_caps,
dim_caps, H, W]
        out = out.view(
            batch, -1, self.dim_caps, H,
W
        )
        # Squash
        return
    self.squash(out.view(batch, -1,
self.dim_caps))

    def squash(self, s):
        s_norm = torch.norm(s, dim=2,
keepdim=True)
        s_norm_sq = s_norm ** 2
        v = (s_norm_sq / (1 +
s_norm_sq)) * (s / s_norm)
        return v

class DigitCapsule(nn.Module):
    def __init__(self, in_caps,
out_caps,
in_dim, out_dim,
num_routing=3):
        super().__init__()

```

## Capsule Networks Cheatsheet — 3 колонки

```

self.in_caps = in_caps
self.out_caps = out_caps
self.num_routing = num_routing

# Transformation matrix
self.W = nn.Parameter(
    torch.randn(out_caps,
in_caps,
out_dim, in_dim)
)

def forward(self, u):
    # u: [batch, in_caps, in_dim]
    batch = u.size(0)

    # Compute  $\hat{u}_j|i$  predictions
    u_hat = torch.einsum('bnid,oidn->bnoi',
u, self.W)

    # Initialize routing logits
    b = torch.zeros(batch,
self.in_caps,
self.out_caps,
1).to(u.device)

    # Routing iterations
    for iteration in
range(self.num_routing):
        c = F.softmax(b, dim=2)  # coupling coefficients
        s = torch.sum(c * u_hat,
dim=1)  # weighted sum
        v = self.squash(s)  # squash

        if iteration <
self.num_routing - 1:
            # Update routing logits
            b = b + torch.sum(
                u_hat *
v.unsqueeze(1),
dim=-1, keepdim=True
            )

    return v

    def squash(self, s):
        s_norm = torch.norm(s, dim=2,
keepdim=True)
        s_norm_sq = s_norm ** 2
        v = (s_norm_sq / (1 +

```

```

s_norm_sq)) * (s / s_norm)
return v

```

## 10. Варианты и улучшения

**EM Routing:** использовать EM algorithm вместо dynamic routing

**Self-attention routing:** attention mechanism для routing

**Inverted Dot-Product Attention Routing:** более эффективный routing

**Matrix Capsules:** представлять капсулы матрицами вместо векторов

**Stacked Capsules:** несколько capsule слоёв для глубоких сетей

**Efficient CapsNets:** оптимизации для снижения вычислительной сложности

## 11. Применения

**Image Classification:** MNIST, CIFAR-10, smallNORB

**Object Detection:** локализация и классификация объектов

**Segmentation:** сегментация с понижением spatial info loss

**Medical Imaging:** анализ медицинских изображений

**NLP:** text classification, sentiment analysis

**Video Analysis:** action recognition, video segmentation

**3D Vision:** point cloud classification

## 12. Best Practices

**Начинать с простых датасетов:** MNIST или Fashion-MNIST для прототипирования

**Настройка routing:**

- Начать с 3 routing iterations
- Больше итераций не всегда лучше
- Экспериментировать с инициализацией  $b$

**Learning rate:** меньшие LR (0.001-0.0001), использовать decay

**Regularization:** decoder помогает, но  $\alpha$  нужно тщательно подбирать

**Когда использовать:**

- Задачи с иерархическими структурами
- Малые датасеты
- Когда важна интерпретируемость

**Когда не использовать:** большие сложные датасеты, real-time inference



17 Январь 2026

## ◆ 1. Суть

- **Алгоритм:** градиентный бустинг от Yandex
- **Особенность:** автоматическая работа с категориями
- **Преимущества:** не требует препроцессинга категорий
- **Применение:** табличные данные, рекомендательные системы

## ◆ 2. Установка

```
# pip
pip install catboost

# conda
conda install -c conda-forge catboost

# C GPU (если доступен CUDA)
# GPU поддержка включена автоматически
```

## ◆ 3. Базовый код — Классификация

```
from catboost import CatBoostClassifier
from sklearn.model_selection import
train_test_split

# Разделение данных
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Указываем категориальные признаки
cat_features = ['city', 'gender', 'device']

# Создание модели
model = CatBoostClassifier(
    iterations=1000,
    learning_rate=0.03,
    depth=6,
    verbose=100, # печать каждые 100 итераций
    random_state=42
)

# Обучение
model.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_test, y_test),
    early_stopping_rounds=50,
    plot=True # визуализация обучения
)

# Предсказание
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)
```

## ◆ 4. Регрессия

```
from catboost import CatBoostRegressor

model = CatBoostRegressor(
    iterations=500,
    learning_rate=0.05,
    depth=6,
    loss_function='RMSE', # или 'MAE', 'MAPE'
    random_state=42
)

model.fit(
    X_train, y_train,
    cat_features=cat_features,
    eval_set=(X_test, y_test),
    verbose=False
)

y_pred = model.predict(X_test)
```

## ◆ 5. Работа с категориями — просто!

```
import pandas as pd

# данные с категориями (строки — это ОК!)
data = pd.DataFrame({
    'age': [25, 30, 35, 40],
    'city': ['Moscow', 'SPB', 'Moscow', 'Kazan'],
    'color': ['red', 'blue', 'red', 'green'],
    'price': [100, 200, 150, 250]
})

# CatBoost работает со строками напрямую!
# Просто указываем имена или индексы

# Вариант 1: имена колонок
cat_features = ['city', 'color']

# Вариант 2: индексы
cat_features = [1, 2]

# Вариант 3: автоопределение (только для
# DataFrame)
# CatBoost сам найдет object/category типы
model.fit(
    X_train, y_train,
    cat_features='auto' # автоматически!
)
```

## ◆ 6. Ключевые параметры

| Параметр            | Описание                      | Значения  |
|---------------------|-------------------------------|-----------|
| iterations          | Количество деревьев           | 100-10000 |
| learning_rate       | Скорость обучения             | 0.01-0.3  |
| depth               | Глубина деревьев              | 4-10      |
| l2_leaf_reg         | L2 регуляризация              | 1-10      |
| border_count        | Разбиений для числ. признаков | 32-255    |
| bagging_temperature | Интенсивность bagging         | 0-1       |
| random_strength     | Случайность при разбиении     | 0-10      |

## ◆ 7. Loss Functions

| Задача                 | Loss Function |
|------------------------|---------------|
| Бинарная классификация | LogLoss       |
| Мультиклассовая        | MultiClass    |
| Регрессия              | RMSE          |
| Регрессия L1           | MAE           |
| Регрессия %            | MAPE          |
| Ranking                | YetiRank      |
| Пуассон                | Poisson       |

## ◆ 8. Важность признаков

```
# Получить важность
feature_importance = model.get_feature_importance()

# Для DataFrame с именами
feature_names = X_train.columns
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importance
}).sort_values('importance', ascending=False)

print(importance_df.head(10))

# Визуализация встроенная
model.get_feature_importance(
    prettified=True,
    type='FeatureImportance' # или 'ShapValues'
)

# Типы важности:
# - FeatureImportance (default)
# - ShapValues (медленнее, но точнее)
# - Interaction (важность взаимодействий)
# - PredictionDiff
```

## ◆ 9. Автоматический подбор параметров

```
# Встроенный grid search
model = CatBoostClassifier()

grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'depth': [4, 6, 8],
    'l2_leaf_reg': [1, 3, 5, 7]
}

# Grid Search
grid_search_result = model.grid_search(
    grid,
    X_train, y_train,
    cv=5,
    plot=True,
    verbose=False
)

# Лучшие параметры
print(grid_search_result['params'])

# Randomized Search
randomized_search_result =
model.randomized_search(
    grid,
    X_train, y_train,
    n_iter=20,
    cv=5
)
```

## ◆ 10. Pool — продвинутый датасет

```
from catboost import Pool

# Создание Pool с метаданными
train_pool = Pool(
    data=X_train,
    label=y_train,
    cat_features=cat_features,
    weight=sample_weights, # веса примеров
    baseline=baseline, # начальные предсказания
    feature_names=feature_names
)

test_pool = Pool(
    data=X_test,
    label=y_test,
    cat_features=cat_features
)

# Обучение
model = CatBoostClassifier(iterations=500)
model.fit(train_pool, eval_set=test_pool)

# Преимущества Pool:
# - Быстрее (предобработка 1 раз)
# - Удобно для экспериментов
# - Поддержка весов и baseline
```

## ◆ 11. Cross-Validation

```
from catboost import cv

# Подготовка данных
pool = Pool(X, y, cat_features=cat_features)

# Параметры модели
params = {
    'iterations': 1000,
    'learning_rate': 0.03,
    'depth': 6,
    'loss_function': 'Logloss'
}

# CV
cv_results = cv(
    pool=pool,
    params=params,
    fold_count=5,
    shuffle=True,
    stratified=True,
    early_stopping_rounds=50,
    verbose=100
)

# Результаты
print(f"Best iteration: {cv_results.shape[0]}")
print(f"Best score: {cv_results['test-Logloss-mean'].min():.4f}")

# DataFrame с результатами
print(cv_results.head())
```

## ◆ 12. GPU ускорение

```
# CatBoost автоматически использует GPU если
доступен!
model = CatBoostClassifier(
    iterations=1000,
    task_type='GPU', # явное указание
    devices='0', # номер GPU (или '0:1' для
нескольких)
)

# Параметры GPU
model = CatBoostClassifier(
    task_type='GPU',
    gpu_ram_part=0.9, # использовать 90% GPU
памяти
    pinned_memory_size=2_000_000_000, # 2GB
)

# Для CPU (если нужно принудительно)
model = CatBoostClassifier(task_type='CPU')
```

## ◆ 13. Работа с текстом

```
# CatBoost может работать с текстом!
from catboost import CatBoostClassifier

# данные с текстовыми колонками
data = pd.DataFrame({
    'text_feature': ['hello world', 'machine
learning', ...],
    'category': ['A', 'B', ...],
    'target': [0, 1, ...]
})

# Указываем текстовые признаки
model = CatBoostClassifier(
    iterations=500,
    text_features=['text_feature'], # текстовые
признаки
    cat_features=['category'], # категориальные
)

model.fit(X_train, y_train)

# CatBoost автоматически:
# - Токенизирует текст
# - Создает эмбеддинги
# - Использует их в обучении
```

## ◆ 14. Визуализация обучения

```
# Встроенная визуализация
model = CatBoostClassifier(iterations=1000)

model.fit(
    X_train, y_train,
    eval_set=(X_test, y_test),
    plot=True # интерактивный график в Jupyter
)

# Сохранение метрик
model.fit(
    X_train, y_train,
    eval_set=(X_test, y_test),
    use_best_model=True,
    train_dir='catboost_info' # сохранить логи
)

# Визуализация через TensorBoard
# catboost_info содержит файлы для TensorBoard
# tensorboard --logdir=catboost_info

# Получить историю
evals_result = model.get_evals_result()
print(evals_result.keys()) # метрики
print(evals_result['learn']['Logloss']) # на
train
print(evals_result['validation']['Logloss']) # на
test
```

## ◆ 15. SHAP values (объяснимость)

```
# CatBoost имеет встроенный SHAP!
import shap

# Получить SHAP values
shap_values = model.get_feature_importance(
    data=Pool(X_test, cat_features=cat_features),
    type='ShapValues'
)

# Визуализация
shap.initjs()
shap.force_plot(
    model.get_all_params()['baseline'],
    shap_values[0, :-1], # для первого примера
    X_test.iloc[0]
)

# Summary plot
shap.summary_plot(
    shap_values[:, :-1],
    X_test
)
```

## ◆ 16. Сохранение и загрузка

```
# Сохранение в разных форматах
# 1. CatBoost формат (лучший)
model.save_model('model.cbm')
model = CatBoostClassifier()
model.load_model('model.cbm')

# 2. JSON (человеко-читаемый)
model.save_model('model.json', format='json')

# 3. ONNX (для production)
model.save_model('model.onnx', format='onnx')

# 4. Python object (для sklearn Pipeline)
import pickle
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)

# 5. C++ код (для встраивания)
model.save_model('model.cpp', format='cpp')
```

## ◆ 17. Когда использовать

### ✓ Хорошо

- ✓ Много категориальных признаков
- ✓ Табличные данные
- ✓ Нужна интерпретируемость (SHAP)
- ✓ Нужна стабильность результатов
- ✓ Есть текстовые признаки

### ✗ Плохо

- ✗ Очень малые данные (<500 строк)
- ✗ Нужна максимальная скорость обучения
- ✗ Изображения (используйте CNN)
- ✗ Последовательности (используйте RNN/Transformer)

## ◆ 18. CatBoost vs XGBoost vs LightGBM

| Критерий     | CatBoost    | LightGBM        | XGBoost      |
|--------------|-------------|-----------------|--------------|
| Категории    | 🥇 Нативно   | ✓ Нужно указать | ✗ Кодировать |
| Скорость     | Средняя     | 🥇 Быстрая       | Медленная    |
| Точность     | 🥇 Высокая   | Высокая         | Высокая      |
| Настройка    | 🥇 Простая   | Сложная         | Средняя      |
| Переобучение | 🥇 Устойчив  | Склонен         | Средне       |
| Текст        | 🥇 Поддержка | ✗               | ✗            |

## ◆ 19. Типичные ошибки

### ✗ Неправильно

- ✗ Кодировать категории вручную
- ✗ Не указывать `cat_features`
- ✗ Использовать слишком большой `depth`
- ✗忽орировать `use_best_model=True`

### ✓ Правильно

- ✓ Передавать категории как строки
- ✓ Всегда указывать `cat_features`
- ✓ Начинать с `depth=6`
- ✓ Использовать `early stopping` и `best model`

## ◆ 20. Чек-лист

- [ ] Определить категориальные признаки
- [ ] Указать `cat_features` при обучении
- [ ] Использовать `early_stopping_rounds`
- [ ] Установить `use_best_model=True`
- [ ] Начать с дефолтных параметров
- [ ] Настроить `depth` и `learning_rate`
- [ ] Использовать Pool для оптимизации
- [ ] Проверить переобучение на валидации
- [ ] Использовать SHAP для интерпретации

### 💡 Объяснение заказчику:

«CatBoost — это умный алгоритм, который понимает категории "как есть". Если у вас есть признаки типа "город" или "цвет", не нужно их преобразовывать — CatBoost сам разберется и найдет закономерности».

## 🔗 Полезные ссылки

- 🌐 Официальный сайт
- 📚 Документация
- 💻 GitHub репозиторий
- ⚙ Гайд по тюнингу
- 📖 API Reference
- 📝 Tutorials и примеры
- 📄 Научная статья



# Кодирование категориальных признаков

17 Январь 2026

## ◆ 1. Зачем кодировать?

- **ML-алгоритмы** работают только с числами
- **Категориальные данные:** текст, категории, группы
- **Примеры:** цвет, город, тип продукта
- **Правильное кодирование** критично для качества модели

### Типы категориальных признаков:

- **Номинальные:** нет порядка (цвет, город)
- **Порядковые:** есть порядок (low, medium, high)
- **Бинарные:** два значения (да/нет)

## ◆ 2. Label Encoding

Простое присвоение чисел категориям:

```
from sklearn.preprocessing import LabelEncoder

# Пример данных
colors = ['red', 'blue', 'green', 'red', 'green']

# Label Encoding
le = LabelEncoder()
encoded = le.fit_transform(colors)
print(encoded) # [2 0 1 2 1]

# Обратное преобразование
decoded = le.inverse_transform(encoded)
print(decoded) # ['red' 'blue' 'green' 'red'
'green']

# Получить соответствие
print(dict(zip(le.classes_,
le.transform(le.classes_))))
# {'blue': 0, 'green': 1, 'red': 2}

# Для pandas DataFrame
import pandas as pd
df = pd.DataFrame({'color': colors})
df['color_encoded'] =
le.fit_transform(df['color'])
```

**Проблема:** создаёт искусственный порядок!

## ◆ 3. One-Hot Encoding

Создание бинарных колонок для каждой категории:

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Pandas способ (проще)
df = pd.DataFrame({'color': ['red', 'blue',
'green', 'red']})
df_encoded = pd.get_dummies(df, columns=['color'],
prefix='color')
print(df_encoded)
#   color_blue  color_green  color_red
# 0          0          0          1
# 1          1          0          0
# 2          0          1          0
# 3          0          0          1

# Sklearn способ
ohe = OneHotEncoder(sparse_output=False,
drop=None)
encoded = ohe.fit_transform(df[['color']])
feature_names =
ohe.get_feature_names_out(['color'])

# С удалением первой категории (избежать
мультиколлинеарности)
ohe = OneHotEncoder(sparse_output=False,
drop='first')
encoded = ohe.fit_transform(df[['color']])

# Pandas с drop_first
df_encoded = pd.get_dummies(df, columns=['color'],
drop_first=True)
```

## ◆ 4. Ordinal Encoding

Для категорий с естественным порядком:

```
from sklearn.preprocessing import OrdinalEncoder

# Данные с порядком
sizes = [['small'], ['medium'], ['large'],
          ['small'], ['large']]

# Определяем порядок
encoder = OrdinalEncoder(
    categories=[['small', 'medium', 'large']])
)

encoded = encoder.fit_transform(sizes)
print(encoded) # [[0.] [1.] [2.] [0.] [2.]]

# Pandas способ
df = pd.DataFrame({'size': ['small', 'medium',
                             'large', 'small']})

size_mapping = {'small': 0, 'medium': 1, 'large': 2}
df['size_encoded'] = df['size'].map(size_mapping)

# Или более гибко
from pandas.api.types import CategoricalDtype

size_order = CategoricalDtype(
    categories=['small', 'medium', 'large'],
    ordered=True)
)
df['size_cat'] = df['size'].astype(size_order)
df['size_encoded'] = df['size_cat'].cat.codes
```

## ◆ 5. Frequency Encoding

Замена на частоту появления:

```
import pandas as pd

df = pd.DataFrame({
    'city': ['Moscow', 'SPB', 'Moscow', 'Kazan',
             'SPB', 'Moscow']
})

# Подсчёт частот
freq = df['city'].value_counts(normalize=True)
print(freq)
# Moscow      0.50
# SPB        0.33
# Kazan      0.17

# Кодирование
df['city_freq'] = df['city'].map(freq)

# Или подсчёт количества
counts = df['city'].value_counts()
df['city_count'] = df['city'].map(counts)

# Функция для использования
def frequency_encoding(df, column):
    freq = df[column].value_counts(normalize=True)
    return df[column].map(freq)
```

## ◆ 6. Target Encoding (Mean Encoding)

Замена на среднее значение целевой переменной:

```
import pandas as pd
import numpy as np

# Данные
df = pd.DataFrame({
    'city': ['Moscow', 'SPB', 'Moscow', 'Kazan',
             'SPB'],
    'target': [100, 80, 120, 90, 85]
})

# Простой target encoding (ТОЛЬКО на train!)
target_mean = df.groupby('city')['target'].mean()
df['city_target_enc'] =
df['city'].map(target_mean)

# С регуляризацией (smoothing)
def target_encode_smooth(df, cat_col, target_col,
alpha=5):
    global_mean = df[target_col].mean()
    agg = df.groupby(cat_col)
    [target_col].agg(['mean', 'count'])

    # Smoothed mean
    smooth = (agg['count'] * agg['mean'] + alpha *
    global_mean) / (agg['count'] + alpha)

    return df[cat_col].map(smooth)

df['city_smooth'] = target_encode_smooth(df,
'city', 'target', alpha=5)

# Category Encoders библиотека
from category_encoders import TargetEncoder

te = TargetEncoder(cols=['city'])
# Fit только на train!
X_train_enc = te.fit_transform(X_train, y_train)
X_test_enc = te.transform(X_test)
```

**⚠ Риск переобучения!** Используйте cross-validation

## ◆ 7. Binary Encoding

Комбинация label и one-hot encoding:

```
from category_encoders import BinaryEncoder

# Данные
df = pd.DataFrame({
    'city': ['Moscow', 'SPB', 'Kazan', 'Omsk',
    'Moscow']
})

# Binary encoding
be = BinaryEncoder(cols=['city'])
df_encoded = be.fit_transform(df)

print(df_encoded)
#   city_0  city_1  city_2
# 0      0      0      1
# 1      0      1      0
# 2      0      1      1
# 3      1      0      0
# 4      0      0      1

# Плюсы: меньше колонок чем one-hot
# Для N категорий создаёт log2(N) колонок
```

## ◆ 8. Hash Encoding

Для высококардинальных признаков:

```
from sklearn.feature_extraction import
FeatureHasher
from category_encoders import HashingEncoder

# Category Encoders способ
he = HashingEncoder(cols=['city'], n_components=8)
df_encoded = he.fit_transform(df)

# Sklearn способ
hasher = FeatureHasher(n_features=8,
input_type='string')
hashed = hasher.fit_transform(df['city'].values)

# Плюсы:
# - Фиксированное число колонок
# - Не нужно запоминать все категории
# - Работает с новыми категориями

# Минусы:
# - Возможны коллизии (разные категории → один
хеш)
# - Сложнее интерпретировать
```

## ◆ 9. Leave-One-Out Encoding

Target encoding без утечки данных:

```
from category_encoders import LeaveOneOutEncoder

# Для каждой строки вычисляет mean target
# по всем остальным строкам с той же категорией

df = pd.DataFrame({
    'city': ['Moscow', 'SPB', 'Moscow', 'SPB',
    'Moscow'],
    'target': [100, 80, 120, 85, 110]
})

loo = LeaveOneOutEncoder(cols=['city'])
df_encoded = loo.fit_transform(df['city'],
df['target'])

print(df_encoded)
# Для 1-й Moscow: среднее по 2-й и 3-й Moscow
# Для 2-й Moscow: среднее по 1-й и 3-й Moscow

# Преимущество: меньше переобучения чем обычный
target encoding
```

## ◆ 10. Обработка редких категорий

```
import pandas as pd

df = pd.DataFrame({
    'city': ['Moscow', 'SPB', 'Moscow', 'Kazan',
    'Omsk',
    'Moscow', 'Tver', 'SPB', 'Moscow']
})

# Метод 1: Группировка редких в "Other"
def group_rare_categories(df, column,
threshold=0.05):
    # Частота каждой категории
    freq = df[column].value_counts(normalize=True)

    # Редкие категории
    rare = freq[freq < threshold].index

    # Замена
    df[column + '_grouped'] = df[column].where(
        ~df[column].isin(rare),
        'Other'
    )

    return df

df = group_rare_categories(df, 'city',
threshold=0.15)

# Метод 2: Топ-N категорий + Other
def keep_top_categories(df, column, n=5):
    top_n =
    df[column].value_counts().head(n).index
    df[column + '_top'] = df[column].where(
        df[column].isin(top_n),
        'Other'
    )
    return df

df = keep_top_categories(df, 'city', n=3)
```

## ◆ 11. Обработка новых категорий

```
# Проблема: в test появляются новые категории

# Решение 1: handle_unknown в OneHotEncoder
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(
    handle_unknown='ignore', # Игнорировать новые
    sparse_output=False
)

X_train = [['Moscow'], ['SPB'], ['Kazan']]
X_test = [['Moscow'], ['NewCity']] # Новая
категория

ohe.fit(X_train)
encoded_test = ohe.transform(X_test)
# NewCity будет закодирован как все нули

# Решение 2: Заранее добавить категорию "Unknown"
df['city_with_unknown'] =
df['city'].cat.add_categories(['Unknown'])

# Решение 3: Заменить неизвестные на частую
категорию
train_categories = set(X_train['city'])
X_test['city_safe'] = X_test['city'].where(
    X_test['city'].isin(train_categories),
    'Moscow' # Самая частая из train
)
```

## ◆ 12. Выбор метода кодирования

| Метод            | Когда использовать                                | Плюсы/Минусы                                       |
|------------------|---------------------------------------------------|----------------------------------------------------|
| <b>Label</b>     | Порядковые<br>признаки, таргет в<br>классификации | ✓ Простой<br>✗ Создаёт<br>ложный порядок           |
| <b>One-Hot</b>   | Мало категорий (<10),<br>линейные модели          | ✓ Нет ложных<br>связей<br>✗ Много<br>колонок       |
| <b>Ordinal</b>   | Есть естественный<br>порядок                      | ✓ Сохраняет<br>порядок<br>✗ Нужно знать<br>порядок |
| <b>Target</b>    | Высокая<br>кардинальность, tree-<br>based модели  | ✓ Мощный<br>✗ Риск<br>переобучения                 |
| <b>Frequency</b> | Когда частота<br>категории важна                  | ✓ Простой<br>✗ Теряет<br>идентичность              |
| <b>Binary</b>    | Средняя<br>кардинальность (10-<br>100)            | ✓ Компактный<br>✗ Менее<br>интерпретируем          |
| <b>Hashing</b>   | Очень высокая<br>кардинальность<br>(>1000)        | ✓ Фикс. размер<br>✗ Коллизии                       |

## ◆ 13. Лучшие практики

### ✓ Делать

- ✓ Fit encoding только на train данных
- ✓ One-hot для линейных моделей
- ✓ Target encoding для gradient boosting
- ✓ Обрабатывать редкие категории
- ✓ Продумать стратегию для новых категорий
- ✓ Использовать Pipeline для автоматизации

### ✗ Не делать

- ✗ Fit на всех данных (train + test)
- ✗ Label encoding для номинальных признаков
- ✗ Target encoding без cross-validation
- ✗ One-hot для высокой кардинальности
- ✗ Игнорировать новые категории в test

## ◆ 14. Pipeline для кодирования

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier

# Определяем типы колонок
cat_features = ['city', 'color']
num_features = ['price', 'age']

# Трансформеры
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_features),
        ('cat',
        OneHotEncoder(handle_unknown='ignore'),
        cat_features)
    ]
)

# Pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])

# Обучение и предсказание в один шаг
pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)

# Сохранение pipeline
import joblib
joblib.dump(pipeline, 'model_pipeline.pkl')
```

## ◆ 15. Чек-лист

- [ ] Определить тип категориальных признаков
- [ ] Проверить кардинальность (количество уникальных)
- [ ] Для низкой (<10) → One-Hot Encoding
- [ ] Для средней (10-100) → Binary/Target Encoding
- [ ] Для высокой (>100) → Target/Hash Encoding
- [ ] Обработать редкие категории
- [ ] Продумать стратегию для новых категорий
- [ ] Использовать Pipeline
- [ ] Проверить метрики с разными методами
- [ ] Документировать выбранный метод

### Объяснение заказчику:

«Кодирование категорий — это преобразование текстовых значений в числа, понятные для алгоритмов. Например, цвета "красный", "синий", "зелёный" можно закодировать как 0, 1, 2, или создать отдельный признак для каждого цвета».

1  
2  
3  
4

# Выбор числа кластеров

17 Январь 2026

## ◆ 1. Проблема выбора K

- **Нет правильного ответа:** оптимальное K зависит от задачи
- **Компромисс:** слишком мало или слишком много кластеров
- **Множество методов:** каждый со своими плюсами
- **Бизнес-контекст:** иногда важнее статистики

## ◆ 2. Метод локтя (Elbow Method)

Самый популярный метод:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Вычисление inertia для разных K
inertias = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(K_range, inertias, 'bx-')
plt.xlabel('Количество кластеров K')
plt.ylabel('Inertia (WCSS)')
plt.title('Метод локтя')
plt.grid(True)
plt.show()

# Ищем "локоть" – точку перегиба
```

**Как найти локоть:** Точка, где уменьшение inertia замедляется

## ◆ 3. Silhouette Score

Измеряет качество кластеризации:

```
from sklearn.metrics import silhouette_score

silhouette_scores = []
K_range = range(2, 11) # Минимум 2 кластера

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    silhouette_scores.append(score)
    print(f"K={k}: Silhouette Score = {score:.3f}")

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(K_range, silhouette_scores, 'bx-')
plt.xlabel('Количество кластеров K')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score для разных K')
plt.grid(True)
plt.show()

# Выбор K с максимальным silhouette score
optimal_k = K_range[np.argmax(silhouette_scores)]
print(f"Оптимальное K: {optimal_k}")
```

**Диапазон:** [-1, 1], ближе к 1 = лучше

## ◆ 4. Silhouette Plot

```
from sklearn.metrics import silhouette_samples
import numpy as np

def plot_silhouette(X, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters,
                    random_state=42)
    labels = kmeans.fit_predict(X)

    silhouette_avg = silhouette_score(X, labels)
    sample_silhouette_values = silhouette_samples(X, labels)

    fig, ax = plt.subplots(figsize=(10, 6))
    y_lower = 10

    for i in range(n_clusters):
        ith_cluster_values = sample_silhouette_values[labels == i]
        ith_cluster_values.sort()

        size_cluster_i = ith_cluster_values.shape[0]
        y_upper = y_lower + size_cluster_i

        ax.fill_betweenx(np.arange(y_lower, y_upper),
                        0, ith_cluster_values,
                        alpha=0.7,
                        label=f'Cluster {i}')

        y_lower = y_upper + 10

    ax.axvline(x=silhouette_avg, color="red",
               linestyle="--",
               label=f'Average: {silhouette_avg:.3f}')
    ax.set_xlabel("Silhouette Coefficient")
    ax.set_ylabel("Cluster")
    ax.set_title(f"Silhouette Plot (K={n_clusters})")
    ax.legend()
    plt.show()

# Использование
plot_silhouette(X_scaled, n_clusters=3)
```

## ◆ 5. Davies-Bouldin Index

Чем меньше, тем лучше:

```
from sklearn.metrics import davies_bouldin_score

db_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    score = davies_bouldin_score(X_scaled, labels)
    db_scores.append(score)
    print(f"K={k}: DB Index = {score:.3f}")

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(K_range, db_scores, 'bx-')
plt.xlabel('Количество кластеров K')
plt.ylabel('Davies-Bouldin Index')
plt.title('Davies-Bouldin Index (меньше = лучше)')
plt.grid(True)
plt.show()

# Оптимальное K – минимум
optimal_k = K_range[np.argmin(db_scores)]
print(f"Оптимальное K: {optimal_k}")
```

## ◆ 6. Calinski-Harabasz Index

Чем больше, тем лучше:

```
from sklearn.metrics import calinski_harabasz_score

ch_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)
    score = calinski_harabasz_score(X_scaled,
                                    labels)
    ch_scores.append(score)
    print(f"K={k}: CH Index = {score:.3f}")

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(K_range, ch_scores, 'bx-')
plt.xlabel('Количество кластеров K')
plt.ylabel('Calinski-Harabasz Index')
plt.title('Calinski-Harabasz Index (больше = лучше)')
plt.grid(True)
plt.show()

# Оптимальное K – максимум
optimal_k = K_range[np.argmax(ch_scores)]
print(f"Оптимальное K: {optimal_k}")
```

## ◆ 7. Gap Statistic

Сравнение с случайными данными:

```
from sklearn.cluster import KMeans
import numpy as np

def gap_statistic(X, K_range, n_refs=10):
    gaps = []

    for k in K_range:
        # Реальные данные
        kmeans = KMeans(n_clusters=k,
                        random_state=42)
        kmeans.fit(X)
        ref_dispersion = kmeans.inertia_

        # Случайные данные
        ref_dispersions = []
        for _ in range(n_refs):
            # Генерируем случайные данные в том же
            # диапазоне
            random_data = np.random.uniform(
                X.min(axis=0),
                X.max(axis=0),
                size=X.shape
            )
            kmeans_ref = KMeans(n_clusters=k,
                                random_state=42)
            kmeans_ref.fit(random_data)

        ref_dispersions.append(kmeans_ref.inertia_)

        # Gap = log(ref) - log(real)
        gap = np.log(np.mean(ref_dispersions)) -
              np.log(ref_dispersion)
        gaps.append(gap)

    return gaps

# Вычисление
K_range = range(1, 11)
gaps = gap_statistic(X_scaled, K_range)

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(K_range, gaps, 'bx-')
plt.xlabel('Количество кластеров K')
plt.ylabel('Gap Statistic')
plt.title('Gap Statistic')
plt.grid(True)
plt.show()

# Оптимальное K – максимум gap
```

```
optimal_k = K_range[np.argmax(gaps)]
print(f"Оптимальное K: {optimal_k}")
```

## 8. Сравнение всех методов

```
import numpy as np
import matplotlib.pyplot as plt

def compare_methods(X, K_range):
    metrics = {
        'Inertia': [],
        'Silhouette': [],
        'Davies-Bouldin': [],
        'Calinski-Harabasz': []
    }

    for k in K_range:
        kmeans = KMeans(n_clusters=k,
random_state=42)
        labels = kmeans.fit_predict(X)

        metrics['Inertia'].append(kmeans.inertia_)
        if k > 1:

    metrics['Silhouette'].append(silhouette_score(X,
labels))
        metrics['Davies-
Bouldin'].append(davies_bouldin_score(X, labels))
        metrics['Calinski-
Harabasz'].append(calinski_harabasz_score(X,
labels))

    # Визуализация
    fig, axes = plt.subplots(2, 2, figsize=(15,
10))

    axes[0, 0].plot(K_range, metrics['Inertia'],
'bx-')
    axes[0, 0].set_title('Elbow Method (Inertia)')
    axes[0, 0].set_xlabel('K')
    axes[0, 0].grid(True)

    axes[0, 1].plot(K_range[1:], metrics['Silhouette'],
'bx-')
    axes[0, 1].set_title('Silhouette Score')
    axes[0, 1].set_xlabel('K')
    axes[0, 1].grid(True)

    axes[1, 0].plot(K_range[1:], metrics['Davies-
Bouldin'], 'bx-')
    axes[1, 0].set_title('Davies-Bouldin (меньше =
лучше)')
    axes[1, 0].set_xlabel('K')
    axes[1, 0].grid(True)

    axes[1, 1].plot(K_range[1:], metrics['Calinski-Harabasz'],
'bx-')
    axes[1, 1].set_title('Calinski-Harabasz
(больше = лучше)')
    axes[1, 1].set_xlabel('K')
```

```
axes[1, 1].grid(True)

plt.tight_layout()
plt.show()

compare_methods(X_scaled, range(1, 11))
```

## 9. Визуальный анализ

```
# Визуализация для разных K
from sklearn.decomposition import PCA

# Снижение размерности для визуализации
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

for idx, k in enumerate([2, 3, 4, 5, 6, 7]):
    ax = axes[idx // 3, idx % 3]

    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled)

    scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1],
c=labels, cmap='viridis',
alpha=0.6)
    ax.set_title(f'K = {k}')
    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
    plt.colorbar(scatter, ax=ax)

plt.tight_layout()
plt.show()
```

## ◆ 10. Сравнение методов

| Метод             | Критерий   | Плюсы                   | Минусы                         |
|-------------------|------------|-------------------------|--------------------------------|
| Elbow             | Визуальный | Интуитивный, популярный | Субъективный, не всегда четкий |
| Silhouette        | max        | Четкий критерий         | Медленный на больших данных    |
| Davies-Bouldin    | min        | Быстрый                 | Менее популярный               |
| Calinski-Harabasz | max        | Быстрый                 | Чувствителен к плотности       |
| Gap Statistic     | max        | Статистически обоснован | Медленный, сложный             |

## ◆ 11. Бизнес-контекст

Иногда K определяется задачей:

- **Сегментация клиентов:** бюджет на маркетинг (3-5 сегментов)
- **Товарные категории:** структура магазина
- **Региональное деление:** логистические центры

```
# Пример: выбор между статистикой и бизнесом
optimal_k_stats = 5 # По silhouette score
optimal_k_business = 3 # Бюджет на 3 кампании
```

```
print(f"Статистический оптимум: K={optimal_k_stats}")
print(f"Бизнес-требование: K={optimal_k_business}")
print("Решение: Используем K=3 с пояснением для стейкholderов")
```

## ◆ 12. Чек-лист выбора K

- [ ] Начать с метода локтя (Elbow)
- [ ] Проверить Silhouette Score
- [ ] Сравнить Davies-Bouldin и Calinski-Harabasz
- [ ] Визуализировать кластеры для разных K
- [ ] Учесть бизнес-контекст
- [ ] Проверить размеры кластеров
- [ ] Убедиться, что кластеры интерпретируемы
- [ ] Документировать выбор
- [ ] Валидировать на отложенных данных

### ❖❖ Объяснение заказчику:

«Выбор числа кластеров — это баланс между детализацией и простотой. Слишком мало кластеров теряют важные различия, слишком много — усложняют понимание. Мы используем несколько методов, чтобы найти оптимальное количество для вашей задачи».

 CI/CD для ML Январь 2026

## ◆ 1. Основы CI/CD для ML

**CI/CD для ML:** автоматизация жизненного цикла ML моделей

- **Continuous Integration:** автоматическое тестирование кода и моделей
- **Continuous Delivery:** автоматический deploy в staging
- **Continuous Deployment:** автоматический deploy в production
- **Continuous Training:** автоматическое переобучение
- **Model Monitoring:** отслеживание performance

 *ML CI/CD отличается от традиционного CI/CD наличием данных и моделей*

## ◆ 2. GitHub Actions для ML

```
# .github/workflows/ml-pipeline.yml
name: ML Pipeline

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.9

      - name: Install dependencies
        run: |
          pip install -r requirements.txt

      - name: Run tests
        run: |
          pytest tests/

      - name: Data validation
        run: |
          python scripts/validate_data.py

      - name: Train model
        run: |
          python train.py

      - name: Evaluate model
        run: |
          python evaluate.py

      - name: Check model metrics
        run: |
          python scripts/check_metrics.py

      - name: Register model
        if: github.ref == 'refs/heads/main'
        run: |
          python scripts/register_model.py
```

## ◆ 3. DVC для версионирования данных

```
# Инициализация DVC
dvc init

# Добавление данных под контроль DVC
dvc add data/raw/train.csv
dvc add models/model.pkl

# Git commit
git add data/raw/train.csv.dvc
models/model.pkl.dvc
git commit -m "Add data and model"

# Настройка remote storage
dvc remote add -d myremote s3://mybucket/dvcstore
dvc push

# Pipeline с DVC
# dvc.yaml
stages:
  prepare:
    cmd: python prepare.py
    deps:
      - data/raw/train.csv
    outs:
      - data/processed/train.csv

  train:
    cmd: python train.py
    deps:
      - data/processed/train.csv
      - train.py
    params:
      - train.learning_rate
      - train.epochs
    metrics:
      - metrics.json:
          cache: false
    outs:
      - models/model.pkl

# Запуск pipeline
dvc repro

# Сравнение экспериментов
dvc metrics show
dvc plots show
```

## ◆ 4. MLflow для трекинга

```
# mlflow_project/MLproject
name: my_ml_project

conda_env: conda.yaml

entry_points:
  main:
    parameters:
      learning_rate: {type: float, default: 0.01}
      epochs: {type: int, default: 100}
      command: "python train.py {learning_rate} {epochs}"

# Запуск через MLflow
mlflow run . -P learning_rate=0.001 -P epochs=200

# CI/CD integration
# .github/workflows/mlflow.yml
- name: Run MLflow project
  run: |
    mlflow run . --experiment-name ci-cd-experiment

    # Get best run
    BEST_RUN=$(mlflow runs list --experiment-name ci-cd-experiment \
      --order-by "metrics.accuracy DESC" --max-results 1 \
      --format json | jq -r '.[0].run_id')

    # Register if metrics are good
    python register_if_good.py $BEST_RUN
```

## ◆ 5. Automated Testing для ML

```
# tests/test_model.py
import pytest
from model import MyModel

def test_model_training():
    """Тест обучения модели"""
    model = MyModel()
    model.train(X_train, y_train)
    assert model.is_fitted()

def test_model_inference():
    """Тест предсказаний"""
    model = MyModel.load("model.pkl")
    predictions = model.predict(X_test)
    assert len(predictions) == len(X_test)
    assert all(0 <= p <= 1 for p in predictions)

def test_model_performance():
    """Тест метрик"""
    model = MyModel.load("model.pkl")
    accuracy = model.evaluate(X_test, y_test)
    assert accuracy >= 0.85, f"Accuracy {accuracy} below threshold"

def test_data_schema():
    """Тест схемы данных"""
    import pandas as pd
    df = pd.read_csv("data/train.csv")

    required_columns = ["feature1", "feature2",
    "target"]
    assert all(col in df.columns for col in
    required_columns)
    assert df["target"].isin([0, 1]).all()

def test_inference_time():
    """Тест скорости инференса"""
    import time
    model = MyModel.load("model.pkl")

    start = time.time()
    model.predict(X_test[:1000])
    duration = time.time() - start

    assert duration < 1.0, f"Inference too slow:
{duration}s"
```

## ◆ 6. Model Deployment

```
# Deploy с GitHub Actions
- name: Deploy to production
  if: github.ref == 'refs/heads/main'
  run: |
    # Build Docker image
    docker build -t mymodel:${{ github.sha }} .

    # Push to registry
    docker push myregistry/mymodel:${{ github.sha }}

# Deploy to Kubernetes
kubectl set image deployment/mymodel \
  mymodel=myregistry/mymodel:${{ github.sha }}

# Wait for rollout
kubectl rollout status deployment/mymodel

# Dockerfile для модели
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY model.pkl .
COPY serve.py .

EXPOSE 8000

CMD ["python", "serve.py"]
```

## ◆ 7. Continuous Training

```
# Автоматическое переобучение
# .github/workflows/retrain.yml
name: Retrain Model

on:
  schedule:
    - cron: '0 0 * * 0' # Weekly
  workflow_dispatch: # Manual trigger

jobs:
  retrain:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Fetch latest data
        run: python scripts/fetch_data.py

      - name: Validate data
        run: python scripts/validate_data.py

      - name: Train model
        run: python train.py

      - name: Evaluate model
        run: python evaluate.py

      - name: Compare with production
        run: |
          python scripts/compare_models.py \
            --new-model models/model.pkl \
            --prod-model production/model.pkl \
            --test-data data/test.csv

      - name: Deploy if better
        if: success()
        run: python scripts/deploy_if_better.py
```

## ◆ 8. Model Monitoring в CI/CD

```
# Мониторинг после deploy
# .github/workflows/monitor.yml
name: Monitor Deployed Model

on:
  schedule:
    - cron: '*/30 * * * *' # Every 30 min

jobs:
  monitor:
    runs-on: ubuntu-latest
    steps:
      - name: Check model health
        run: |
          curl -f http://model-api/health || exit 1
      - name: Check metrics
        run: |
          python scripts/check_production_metrics.py
      - name: Check for drift
        run: |
          python scripts/detect_drift.py
      - name: Alert if issues
        if: failure()
        run: |
          python scripts/send_alert.py
```

## ◆ 9. Infrastructure as Code

```
# Terraform для ML infrastructure
# main.tf
resource "aws_sagemaker_model" "my_model" {
  name           = "my-model"
  execution_role_arn = aws_iam_role.sagemaker.arn

  primary_container {
    image          = var.model_image
    model_data_url = var.model_artifact_url
  }
}

resource "aws_sagemaker_endpoint_configuration" "config" {
  name = "my-model-config"

  production_variants {
    variant_name      = "variant-1"
    model_name        =
    aws_sagemaker_model.my_model.name
    initial_instance_count = 1
    instance_type     = "ml.m5.large"
  }
}

resource "aws_sagemaker_endpoint" "endpoint" {
  name           = "my-model-endpoint"
  endpoint_config_name =
  aws_sagemaker_endpoint_configuration.config.name
}

# Apply c GitHub Actions
- name: Deploy infrastructure
  run: |
    terraform init
    terraform plan
    terraform apply -auto-approve
```

## ◆ 10. Best Practices

- **✓ Version everything:** code, data, models, config
- **✓ Automated testing:** unit, integration, performance
- **✓ Gradual rollout:** canary или blue-green deployment
- **✓ Monitoring:** метрики, drift, errors
- **✓ Rollback plan:** быстрый откат при проблемах
- **✓ Documentation:** документируйте pipeline
- **✓ Reproducibility:** фиксируйте окружение
- **✓ Security:** сканируйте на уязвимости

## ◆ 11. Инструменты

| Категория           | Инструменты                        |
|---------------------|------------------------------------|
| CI/CD               | GitHub Actions, GitLab CI, Jenkins |
| Data versioning     | DVC, Pachyderm                     |
| Experiment tracking | MLflow, Weights & Biases           |
| Model registry      | MLflow, Neptune.ai                 |
| Orchestration       | Airflow, Kubeflow, Prefect         |
| Deployment          | Docker, Kubernetes, SageMaker      |
| Monitoring          | Prometheus, Grafana, Evidently     |



17 Январь 2026

## ◆ 1. Суть метода

- **Randomized search:** улучшение PAM через случайный поиск
- **Медоиды:** центры кластеров — реальные точки из данных
- **Локальный минимум:** ищет через случайные соседи
- **Эффективность:** быстрее PAM на больших данных
- **Параметры:** numlocal (перезапусков), maxneighbor (соседей)

## ◆ 2. Базовый код

```
# Нет в sklearn, используем pyclustering
from pyclustering.cluster.clarans import clarans

# Инициализация
clarans_instance = clarans(
    data=X.tolist(),
    number_clusters=3,
    numlocal=2,           # число локальных
    минимумов
    maxneighbor=3         # макс соседей для
    проверки
)

# Кластеризация
clarans_instance.process()
clusters = clarans_instance.get_clusters()
medoids = clarans_instance.get_medoids()

print(f"Найдено кластеров: {len(clusters)}")
print(f"Медоиды: {medoids}")
```

## ◆ 3. Ключевые параметры

| Параметр        | Описание           | Совет                |
|-----------------|--------------------|----------------------|
| number_clusters | Число кластеров    | Определить заранее   |
| numlocal        | Число перезапусков | 2-5 для стабильности |
| maxneighbor     | Макс соседей       | max(250, 1.25% N)    |

## ◆ 4. Алгоритм работы

1. **Инициализация:** случайно выбрать k медоидов
  2. **Локальный поиск:**
    - Сгенерировать случайного соседа (заменить медоид)
    - Вычислить стоимость (total dissimilarity)
    - Если лучше — принять, иначе — попробовать другого соседа
  3. **Остановка:** если проверили maxneighbor соседей без улучшения
  4. **Перезапуск:** повторить numlocal раз, выбрать лучший
- Стоимость:**  $TC = \sum dist(point, nearest\_medoid)$

## ◆ 5. Выбор параметров

```
# Подбор через качество
from sklearn.metrics import silhouette_score
import numpy as np

best_score = -1
for nl in [2, 3, 5]:
    for mn in [3, 5, 10]:
        clr = clarans(
            data=X.tolist(),
            number_clusters=3,
            numlocal=nl,
            maxneighbor=mn
        )
        clr.process()
        clusters = clr.get_clusters()

        labels = np.zeros(len(X))
        for i, c in enumerate(clusters):
            labels[c] = i

        if len(np.unique(labels)) > 1:
            score = silhouette_score(X, labels)
            if score > best_score:
                best_score = score
                best_params = (nl, mn)

print(f"Best: numlocal={best_params[0]}, maxneighbor={best_params[1]}")
```

## ◆ 6. Когда использовать

### ✓ Хорошо

- ✓ Средние данные (10K-100K)
- ✓ Нужны медоиды (реальные точки)
- ✓ Сферические кластеры
- ✓ Быстрее чем PAM

### ✗ Плохо

- ✗ Очень большие данные (>500K)
- ✗ Произвольные формы кластеров
- ✗ Высокая размерность
- ✗ Неизвестное K

## ◆ 7. Предобработка

### ✓ Масштабирование

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

**✓ Обработка выбросов:** медоиды устойчивы, но крайние выбросы лучше удалить

## ◆ 8. Проблемы и решения

| Проблема                | Решение                         |
|-------------------------|---------------------------------|
| Медленная работа        | Уменьшить maxneighbor, numlocal |
| Плохое качество         | Увеличить numlocal, maxneighbor |
| Нестабильные результаты | Увеличить numlocal              |
| Локальный минимум       | Больше перезапусков             |

## ◆ 9. Сравнение с PAM и K-means

| Метод   | Скорость     | Качество | Медоиды |
|---------|--------------|----------|---------|
| K-means | Очень быстро | Хорошо   | Нет     |
| PAM     | Медленно     | Отлично  | Да      |
| CLARANS | Средне       | Хорошо   | Да      |

CLARANS = компромисс между скоростью K-means и качеством PAM

## ◆ 10. Чек-лист

- [ ] Установить pyclustering
- [ ] Масштабировать данные
- [ ] Определить число кластеров
- [ ] Выбрать numlocal (2-5)
- [ ] Выбрать maxneighbor (зависит от N)
- [ ] Оценить качество
- [ ] Проверить медоиды
- [ ] Сравнить с K-means

### 💡 Объяснение заказчику:

«CLARANS находит наиболее типичные объекты (медоиды) для представления каждой группы. Использует умный случайный поиск для баланса между скоростью и качеством кластеризации. Отлично подходит для данных, где важно выбрать реальные представители групп».

# Классическое RL (табличное, без нейросетей)

 17 Январь 2026

## ◆ 1. Основы

- **RL**: агент учится действовать в среде
- **Табличное**: Q-таблица для хранения значений
- **Discrete**: конечные состояния и действия
- **Model-free**: не требует модели среды

### Основные компоненты:

- States (S): состояния среды
- Actions (A): возможные действия
- Rewards (R): вознаграждения
- Policy ( $\pi$ ): стратегия агента

## ◆ 2. Формализация

### Markov Decision Process (MDP):

```
MDP = (S, A, P, R, γ)

S - множество состояний
A - множество действий
P(s'|s, a) - вероятность перехода
R(s, a) - функция вознаграждения
γ - коэффициент дисконтирования (0 < γ < 1)
```

### Value Function:

$$V(s) = E[\sum \gamma^t * R_t | s_0 = s]$$

$$Q(s, a) = E[\sum \gamma^t * R_t | s_0 = s, a_0 = a]$$

## ◆ 3. Value Iteration

Алгоритм для нахождения оптимальной value function:

```
import numpy as np

# Инициализация
V = np.zeros(n_states)

for iteration in range(max_iterations):
    V_new = np.zeros(n_states)

    for s in range(n_states):
        # Максимум по действиям
        q_values = []
        for a in range(n_actions):
            q = reward[s, a] + gamma * sum(
                transition_prob[s, a, s_next] *
                V[s_next]
            )
            q_values.append(q)
        V_new[s] = max(q_values)

    if np.max(np.abs(V - V_new)) < threshold:
        break
    V = V_new
```

## ◆ 4. Policy Iteration

Алгоритм итерации по стратегиям:

```
# 1. Policy Evaluation
def policy_eval(policy, V, gamma=0.9):
    while True:
        delta = 0
        for s in range(n_states):
            v = V[s]
            a = policy[s]
            V[s] = reward[s, a] + gamma * sum(
                transition_prob[s, a, s_next] *
                V[s_next]
            )
            for s_next in range(n_states)
            )
            delta = max(delta, abs(v - V[s]))
        if delta < threshold:
            break
    return V

# 2. Policy Improvement
def policy_improve(V, gamma=0.9):
    policy_stable = True
    for s in range(n_states):
        old_action = policy[s]
        # Жадная стратегия
        q_values = [reward[s, a] + gamma * sum(
            transition_prob[s, a, s_next] *
            V[s_next]
        )
        for s_next in range(n_states)
        ) for a in range(n_actions)]
        policy[s] = np.argmax(q_values)
        if old_action != policy[s]:
            policy_stable = False
    return policy, policy_stable
```

## ◆ 5. Табличные методы

| Метод            | Требует модели | On/Off-policy |
|------------------|----------------|---------------|
| Value Iteration  | Да             | -             |
| Policy Iteration | Да             | -             |
| Q-Learning       | Нет            | Off-policy    |
| SARSA            | Нет            | On-policy     |
| Monte Carlo      | Нет            | Both          |

## ◆ 6. GridWorld пример

```
import numpy as np

class GridWorld:
    def __init__(self, size=5):
        self.size = size
        self.state = (0, 0)
        self.goal = (size-1, size-1)

    def reset(self):
        self.state = (0, 0)
        return self.state

    def step(self, action):
        # actions: 0=up, 1=right, 2=down, 3=left
        x, y = self.state
        if action == 0 and x > 0:
            x -= 1
        elif action == 1 and y < self.size-1:
            y += 1
        elif action == 2 and y > 0:
            y -= 1
        else:
            x += 1

        self.state = (x, y)
        reward = 1.0 if self.state == self.goal
        else -0.1
        done = (self.state == self.goal)

        return self.state, reward, done
```

## ◆ 7. Exploration vs Exploitation

### ε-greedy:

```
def epsilon_greedy(Q, state, epsilon=0.1):
    if np.random.random() < epsilon:
        # Exploration: случайное действие
        return np.random.randint(n_actions)
    else:
        # Exploitation: лучшее действие
        return np.argmax(Q[state, :])
```

### Softmax (Boltzmann):

```
def softmax_policy(Q, state, temperature=1.0):
    q_values = Q[state, :]
    exp_q = np.exp(q_values / temperature)
    probs = exp_q / np.sum(exp_q)
    return np.random.choice(n_actions, p=probs)
```

## ◆ 8. Примеры применения

- **GridWorld:** навигация робота
- **Tic-Tac-Toe:** простые игры
- **Frozen Lake:** стохастическая среда
- **Taxi:** дискретные задачи
- **Cliff Walking:** on/off-policy сравнение

### OpenAI Gym пример:

```
import gym

env = gym.make('FrozenLake-v1')
n_states = env.observation_space.n
n_actions = env.action_space.n

Q = np.zeros((n_states, n_actions))
```

## ◆ 9. Ограничения табличных методов

- **Curse of dimensionality:** размер таблицы растёт экспоненциально
- **Discrete** только: не работает с непрерывными состояниями
- **Медленная сходимость:** требует много итераций
- **Память:** большие таблицы Q-значений
- **Generalization:** нет обобщения между похожими состояниями

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Малое количество состояний ( $< 10^6$ )
- ✓ Дискретные действия и состояния
- ✓ Простые задачи (игры, навигация)
- ✓ Нужна теоретическая гарантия сходимости
- ✓ Обучение новичков RL

### ✗ Плохо

- ✗ Большое пространство состояний
- ✗ Непрерывные состояния/действия
- ✗ Сложные задачи (Atari, робототехника)
- ✗ Нужна быстрая адаптация

## ◆ 11. Чек-лист

- [ ] Дискретное пространство состояний и действий
- [ ] Инициализировать Q-таблицу нулями или случайно
- [ ] Выбрать правильный  $\gamma$  (0.9-0.99)
- [ ] Настроить exploration ( $\epsilon = 0.1-0.3$ )
- [ ] Уменьшать  $\epsilon$  со временем (decay)
- [ ] Проверить сходимость алгоритма
- [ ] Визуализировать learned policy
- [ ] Тестировать на простых средах (GridWorld)

### Объяснение заказчику:

«Табличное RL — это как шпаргалка с ответами: агент запоминает, что делать в каждой ситуации. Это работает отлично для простых задач, но не подходит для сложных, где ситуаций слишком много».

# Метрики классификации

 17 Январь 2026

## 1. Основные понятия

- **TP** (True Positive): правильно определён положительный класс
- **TN** (True Negative): правильно определён отрицательный класс
- **FP** (False Positive): ложная тревога (Type I error)
- **FN** (False Negative): пропуск (Type II error)

## 2. Confusion Matrix

```
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=['Class 0', 'Class 1']
)
disp.plot()
plt.show()

# cm = [[TN, FP],
#        [FN, TP]]
```

## 3. Accuracy (Точность)

**Формула:**  $(TP + TN) / (TP + TN + FP + FN)$

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.3f}")
```

**Когда использовать:**

-  Сбалансированные классы
-  Несбалансированные классы (misleading!)

## 4. Precision (Точность положительного класса)

**Формула:**  $TP / (TP + FP)$

**Вопрос:** "Из всех предсказанных положительных, сколько действительно положительных?"

```
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.3f}")
```

**Когда важна:**

- Стоимость FP высока (спам-фильтр)
- Медицинская диагностика (не пугать здоровых)

## 5. Recall (Полнота, Sensitivity)

**Формула:**  $TP / (TP + FN)$

**Вопрос:** "Из всех реальных положительных, сколько мы нашли?"

```
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.3f}")
```

**Когда важен:**

- Стоимость FN высока (пропуск болезни)
- Fraud detection (найти все мошенничества)

## 6. F1-Score

**Формула:**  $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

**Гармоническое среднее** precision и recall

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.3f}")
```

**Когда использовать:**

- Несбалансированные классы
- Нужен баланс precision/recall
- Общая метрика качества

## ◆ 7. Выбор метрики

| Задача                  | Метрика       | Почему                     |
|-------------------------|---------------|----------------------------|
| Медицинская диагностика | Recall        | Нельзя пропустить больного |
| Спам-фильтр             | Precision     | Не терять важные письма    |
| Fraud detection         | F1 или Recall | Найти все случаи           |
| Balanced dataset        | Accuracy      | Простая интерпретация      |
| Imbalanced dataset      | F1, ROC-AUC   | Учёт дисбаланса            |

## ◆ 8. Classification Report

```
from sklearn.metrics import classification_report

report = classification_report(y_test, y_pred,
target_names=['Class 0', 'Class 1'])
print(report)

# Output:
#          precision    recall  f1-score
support
#   Class 0      0.85     0.90      0.87
100
#   Class 1      0.88     0.82      0.85
90
#   accuracy           0.86
190
#   macro avg      0.86     0.86      0.86
190
#weighted avg    0.86     0.86      0.86
190
```

## ◆ 9. ROC-AUC

**ROC:** Receiver Operating Characteristic

**AUC:** Area Under Curve (0.5-1.0)

```
from sklearn.metrics import roc_auc_score,
roc_curve

# Нужны вероятности, а не предсказания
y_proba = model.predict_proba(X_test)[:, 1]

auc = roc_auc_score(y_test, y_proba)
print(f"ROC-AUC: {auc:.3f}")

# Построение ROC кривой
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

## ◆ 11. Мультиклассовая классификация

```
# Macro average: усреднение по классам (равный вес)
f1_macro = f1_score(y_test, y_pred,
average='macro')

# Weighted average: взвешенное по количеству примеров
f1_weighted = f1_score(y_test, y_pred,
average='weighted')

# Micro average: глобальный подсчёт
f1_micro = f1_score(y_test, y_pred,
average='micro')

print(f"F1 Macro: {f1_macro:.3f}")
print(f"F1 Weighted: {f1_weighted:.3f}")
print(f"F1 Micro: {f1_micro:.3f}")

# ROC-AUC для мультикласса
from sklearn.preprocessing import label_binarize
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
auc = roc_auc_score(y_test_bin,
y_proba_multiclass, multi_class='ovr')
```

## ◆ 10. Precision-Recall Curve

```
from sklearn.metrics import
precision_recall_curve, average_precision_score

precision, recall, thresholds =
precision_recall_curve(y_test, y_proba)
ap = average_precision_score(y_test, y_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'AP = {ap:.2f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()

# Лучше для несбалансированных классов, чем ROC
```

## ◆ 12. Чек-лист

- [ ] Построить confusion matrix
- [ ] Вычислить classification report
- [ ] Для несбалансированных — F1, not Accuracy
- [ ] Выбрать метрику под бизнес-цель
- [ ] Построить ROC-AUC кривую
- [ ] Для сильного дисбаланса — PR curve
- [ ] Для мультикласса — macro/weighted avg
- [ ] Проверить метрики на каждом классе отдельно

### Объяснение заказчику:

«Метрики — это способ измерить качество модели. Accuracy говорит "сколько правильных", Precision — "насколько можно доверять положительным предсказаниям", Recall — "сколько мы нашли из всех случаев". Выбор метрики зависит от того, какие ошибки дороже».

# CLIP and Multimodal Models

 17 Январь 2026

## ◆ 1. Multimodal Learning

Объединение модальностей: текст, изображения, аудио

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 2. CLIP Architecture

Contrastive Language-Image Pre-training

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 3. Training CLIP

Contrastive loss, batch size, datasets

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 4. Zero-Shot Classification

Text prompts для классификации

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 5. CLIP Applications

Search, classification, generation

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 6. ALIGN

Google's multimodal model

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 7. BLIP

### Bootstrapping Language-Image Pre-training

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 8. Flamingo

### Visual language models

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 9. GPT-4V

### Multimodal GPT-4

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```

## ◆ 10. Practical Usage

OpenCLIP, Hugging Face

- **Ключевая идея:** подробное объяснение концепции
- **Преимущества:** основные плюсы подхода
- **Недостатки:** ограничения и сложности
- **Применение:** где используется на практике

```
# Пример кода
import torch
import torch.nn as nn

# Демонстрационный код
model = nn.Sequential(
    nn.Linear(10, 20),
    nn.ReLU(),
    nn.Linear(20, 10)
)

# Использование
x = torch.randn(32, 10)
output = model(x)
print(f"Output shape: {output.shape}")
```



# Визуализация кластеров

## 1. Scatter Plot кластеров

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,6))
scatter = plt.scatter(X[:,0], X[:,1],
                      c=labels,
                      cmap='viridis',
                      alpha=0.6)
plt.scatter(centroids[:,0],
            centroids[:,1],
            c='red', marker='X', s=200,
            edgecolors='black')
plt.colorbar(scatter)
plt.title('K-means Clustering')
plt.show()
```

## 2. PCA для визуализации

```
from sklearn.decomposition import PCA

# Для данных > 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:,0], X_pca[:,1],
            c=labels, cmap='tab10')
plt.title(f'PCA (explained var: {pca.explained_variance_ratio_.sum():.2%})')
plt.show()
```

## 3. t-SNE визуализация

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2,
            random_state=42)
X_tsne = tsne.fit_transform(X)

plt.scatter(X_tsne[:,0], X_tsne[:,1],
            c=labels)
plt.title('t-SNE Clustering Visualization')
plt.show()
```

## 4. Dendrogram для иерархической кластеризации

```
from scipy.cluster.hierarchy import dendrogram, linkage

linkage_matrix = linkage(X,
                         method='ward')

plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()
```

## 5. Silhouette Plot

```
from sklearn.metrics import silhouette_samples
import numpy as np

silhouette_vals = silhouette_samples(X,
                                      labels)

y_lower = 0
for i in range(n_clusters):
    cluster_silhouette_vals = silhouette_vals[labels == i]
    cluster_silhouette_vals.sort()

    size = cluster_silhouette_vals.shape[0]
    y_upper = y_lower + size

    plt.fill_betweenx(np.arange(y_lower,
                                y_upper),
                      0,
                      cluster_silhouette_vals,
                      alpha=0.7)
    y_lower = y_upper

plt.axvline(x=silhouette_vals.mean(),
            color="red", linestyle="--")
plt.title('Silhouette Plot')
plt.show()
```

## ◆ 6. 3D кластеры

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111,
projection='3d')

ax.scatter(X[:,0], X[:,1], X[:,2],
c=labels, cmap='viridis')
ax.set_title('3D Clustering')
plt.show()
```

## ◆ 8. Elbow Method визуализация

```
inertias = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

plt.plot(K_range, inertias, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

## ◆ 9. Размеры кластеров

```
import pandas as pd

cluster_sizes =
pd.Series(labels).value_counts().sort_index()

plt.bar(cluster_sizes.index,
cluster_sizes.values)
plt.xlabel('Cluster')
plt.ylabel('Size')
plt.title('Cluster Sizes')
plt.show()
```

## ◆ 7. Heatmap расстояний

```
from sklearn.metrics import
pairwise_distances

distances = pairwise_distances(X[:100])
# Subset for vis

plt.figure(figsize=(10, 8))
sns.heatmap(distances, cmap='viridis')
plt.title('Pairwise Distances Heatmap')
plt.show()
```

## ◆ 10. Чек-лист

- [ ] Использовать PCA/t-SNE для высоких размерностей
- [ ] Показать центроиды
- [ ] Визуализировать Silhouette Score
- [ ] Проверить размеры кластеров
- [ ] Использовать разные цветовые схемы
- [ ] Добавить легенду и подписи
- [ ] Сохранить в высоком разрешении
- [ ] Интерактивные графики (Plotly)



## Метрики оценки кластеризации

### 1. Типы метрик кластеризации

- Внутренние метрики:** оценка без истинных меток
  - Silhouette Score
  - Davies-Bouldin Index
  - Calinski-Harabasz Index
  - Inertia / Within-cluster sum of squares
- Внешние метрики:** требуют истинные метки
  - Adjusted Rand Index (ARI)
  - Adjusted Mutual Information (AMI)
  - Normalized Mutual Information (NMI)
  - Fowlkes-Mallows Index
  - Homogeneity, Completeness, V-measure

### 2. Silhouette Score

**Диапазон:**  $[-1, 1]$ , выше = лучше

**Формула:**  $s = (b - a) / \max(a, b)$

- $a$ : среднее расстояние до точек своего кластера
- $b$ : среднее расстояние до точек ближайшего другого кластера

```
from sklearn.metrics import
silhouette_score,
silhouette_samples
import matplotlib.pyplot as plt

# Общий score
score = silhouette_score(X, labels)
print(f"Silhouette Score:
{score:.3f}")

# Интерпретация:
# > 0.7: отлично, четкие кластеры
# 0.5-0.7: хорошее качество
# 0.25-0.5: средне, есть перекрытия
# < 0.25: плохо, неправильная
# кластеризация

# Silhouette для каждой точки
sample_scores =
silhouette_samples(X, labels)

# Визуализация silhouette plot
def plot_silhouette(X, labels):
    n_clusters = len(set(labels)) - 
(1 if -1 in labels else 0)

    fig, ax = plt.subplots(figsize=
(10, 6))
    y_lower = 10

    for i in range(n_clusters):
        cluster_scores =
sample_scores[labels == i]
        cluster_scores.sort()

        size =
cluster_scores.shape[0]
        y_upper = y_lower + size

        ax.fill_betweenx(np.arange(y_lower,
y_upper),
                     0,
                     cluster_scores,
                     alpha=0.7,
                     label=f'Cluster {i}')

        y_lower = y_upper + 10

        ax.set_xlabel('Silhouette
Coefficient')
        ax.set_ylabel('Cluster')
        ax.axvline(x=score,
color='red', linestyle='--',
label=f'Mean:
{score:.3f}')
        ax.legend()
        ax.set_title('Silhouette Plot')
        plt.show()

plot_silhouette(X, labels)
```

### 3. Davies-Bouldin Index

**Диапазон:**  $[0, \infty)$ , ниже = лучше

**Идея:** отношение внутри-кластерных расстояний к между-кластерным

```
from sklearn.metrics import
davies_bouldin_score

db = davies_bouldin_score(X,
labels)
print(f"Davies-Bouldin Index:
{db:.3f}")

# Интерпретация:
# < 0.5: отлично
# 0.5-1.0: хорошо
# 1.0-2.0: приемлемо
# > 2.0: плохо

# Сравнение для разного числа
# кластеров
from sklearn.cluster import KMeans

db_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k,
random_state=42)
    labels = kmeans.fit_predict(X)
    db = davies_bouldin_score(X,
labels)
    db_scores.append(db)

plt.figure(figsize=(10, 5))
plt.plot(K_range, db_scores,
marker='o', linewidth=2)
plt.xlabel('Number of Clusters')
plt.ylabel('Davies-Bouldin Index')
plt.title('Davies-Bouldin Index vs
Number of Clusters')
plt.grid(alpha=0.3)
plt.show()

optimal_k =
K_range[np.argmin(db_scores)]
print(f"Optimal K (lowest DB):
{optimal_k}")
```

## ◆ 4. Calinski-Harabasz Index

**Диапазон:**  $[0, \infty)$ , выше = лучше

**Также называется:** Variance Ratio Criterion

```
from sklearn.metrics import
calinski_harabasz_score

ch = calinski_harabasz_score(X,
labels)
print(f"Calinski-Harabasz:
{ch:.1f}")

# Интерпретация:
# Чем выше, тем плотнее и лучше
# разделены кластеры
# Зависит от размера данных и числа
# кластеров

# Сравнение разных алгоритмов
from sklearn.cluster import DBSCAN,
AgglomerativeClustering

algorithms = {
    'K-Means':
KMeans(n_clusters=3),
    'DBSCAN': DBSCAN(eps=0.5),
    'Hierarchical':
AgglomerativeClustering(n_clusters=3)
}

results = {}
for name, algo in
algorithms.items():
    labels = algo.fit_predict(X)
    if len(set(labels)) > 1: #
Больше 1 кластера
        ch =
calinski_harabasz_score(X, labels)
        results[name] = ch
    else:
        results[name] = 0

# Визуализация
plt.figure(figsize=(10, 5))
plt.bar(results.keys(),
results.values(), alpha=0.7)
plt.ylabel('Calinski-Harabasz
Score')
plt.title('Algorithm Comparison')
plt.xticks(rotation=15)
plt.grid(axis='y', alpha=0.3)
plt.show()
```

## ◆ 5. Adjusted Rand Index (ARI)

**Требует:** истинные метки. **Диапазон:**  $[-1, 1]$

```
from sklearn.metrics import
adjusted_rand_score

ari =
adjusted_rand_score(true_labels,
pred_labels)
print(f"ARI: {ari:.3f}")

# Интерпретация:
# 1.0: идеальное совпадение
# > 0.9: отличное
# 0.5-0.9: хорошее
# 0.0: случайное совпадение
# < 0.0: хуже случайного

# Сравнение с Rand Index
from sklearn.metrics import
rand_score

rand = rand_score(true_labels,
pred_labels)
print(f"Rand Index: {rand:.3f}")
print(f"ARI: {ari:.3f}")

# ARI корректирует на случайные
# совпадения
```

## ◆ 7. Homogeneity, Completeness, V-measure

```
from sklearn.metrics import (
homogeneity_score,
completeness_score,
v_measure_score,

homogeneity_completeness_v_measure
)

# Homogeneity: каждый кластер
# содержит только один класс
homogeneity =
homogeneity_score(true_labels,
pred_labels)

# Completeness: все члены класса в
# одном кластере
completeness =
completeness_score(true_labels,
pred_labels)

# V-measure: гармоническое среднее
# homogeneity и completeness
v_measure =
v_measure_score(true_labels,
pred_labels)

# Или все сразу:
h, c, v =
homogeneity_completeness_v_measure(
    true_labels, pred_labels
)

print(f"Homogeneity: {h:.3f}")
print(f"Completeness: {c:.3f}")
print(f"V-measure: {v:.3f}")

# Все в диапазоне [0, 1], выше =
лучше
```

## ◆ 6. Mutual Information метрики

```
from sklearn.metrics import (
adjusted_mutual_info_score,
normalized_mutual_info_score,
mutual_info_score
)

# Adjusted Mutual Information
# (корректированная)
ami =
adjusted_mutual_info_score(true_label
pred_labels)

# Normalized Mutual Information
nmi =
normalized_mutual_info_score(true_lab
pred_labels)

# Mutual Information (сырая)
mi = mutual_info_score(true_labels,
pred_labels)

print(f"AMI: {ami:.3f}")
print(f"NMI: {nmi:.3f}")
print(f"MI: {mi:.3f}")

# Все в диапазоне [0, 1] (или [-1,
1] для АМИ)
# Выше = лучше совпадение с
истинными метками
```

## ◆ 8. Elbow Method и Inertia

```
from sklearn.cluster import KMeans

# Inertia: сумма квадратов
# расстояний до центроидов
inertias = []
silhouettes = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k,
                     random_state=42, n_init=10)
    kmeans.fit(X)

    inertias.append(kmeans.inertia_)

silhouettes.append(silhouette_score(X,
                                     kmeans.labels_))

# Визуализация Elbow Method
fig, axes = plt.subplots(1, 2,
                        figsize=(14, 5))

# Inertia
axes[0].plot(K_range, inertias,
              marker='o', linewidth=2)
axes[0].set_xlabel('Number of Clusters (K)')
axes[0].set_ylabel('Inertia')
axes[0].set_title('Elbow Method')
axes[0].grid(alpha=0.3)

# Silhouette
axes[1].plot(K_range, silhouettes,
              marker='s',
              linewidth=2,
              color='green')
axes[1].set_xlabel('Number of Clusters (K)')
axes[1].set_ylabel('Silhouette Score')
axes[1].set_title('Silhouette Score vs K')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# Поиск оптимального K
# 1. По Elbow (визуально)
# 2. По максимальному Silhouette
optimal_k_silhouette =
K_range[np.argmax(silhouettes)]
print(f"Optimal K (max silhouette): {optimal_k_silhouette}")
```

## ◆ 9. Сравнительная таблица метрик

| Метрика           | Требует метки | Диапазон | Интерпретация | Когда используется                                                                                  |
|-------------------|---------------|----------|---------------|-----------------------------------------------------------------------------------------------------|
| Silhouette        | Нет           | [-1, 1]  | Выше = лучше  | Общая оценка качества<br>Index: {CH:.1f})<br>print(f" → Выше =")                                    |
| Davies-Bouldin    | Нет           | [0, ∞)   | Ниже = лучше  | Сравнение между K                                                                                   |
| Calinski-Harabasz | Нет           | [0, ∞)   | Выше = лучше  | # Внешние метрики (если есть метки)<br>if true_labels is not None:                                  |
| ARI               | Да            | [-1, 1]  | Выше = лучше  | Сравнение с группами:<br>Внешние метрики (с метками):")                                             |
| AMI/NMI           | Да            | [0, 1]   | Выше = лучше  | Информационное<br>сходство<br>ari =                                                                 |
| V-measure         | Да            | [0, 1]   | Выше = лучше  | Balanced adjusted_rand_score(true_labels,<br>homogeneity_completeness<br>print(f" ARI: {ari:.3f}")) |

## ◆ 10. Комплексная оценка кластеризации

```
def comprehensive_evaluation(X,
                            labels, true_labels=None):
    """Полная оценка качества
    кластеризации"""

    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = list(labels).count(-1)

    print("=" * 50)
    print("ОЦЕНКА КЛАСТЕРИЗАЦИИ")
    print("=" * 50)

    # Базовая информация
    print(f"Базовая информация:")
    print(f" Количество кластеров: {n_clusters}")
    print(f" Количество шумовых
    точек: {n_noise}")
    print(f" Размер каждого
    кластера:")
    for i in set(labels):
        if i != -1:
            count =
list(labels).count(i)
            print(f" Кластер
{i}: {count} точек")

    # Внутренние метрики
    if n_clusters > 1:
        print(f"Внутренние метрики (без меток):")

        sil = silhouette_score(X,
                               labels)
        print(f" Silhouette Score:
{sil:.3f}")
        if sil > 0.7:
            print(f" → Отлично!")
        elif sil > 0.5:
            print(f" → Хорошо")
        elif sil > 0.25:
            print(f" → Средне")
        else:
            print(f" → Плохо")

        db =
davies_bouldin_score(X, labels)
        print(f" Davies-Bouldin
Index: {db:.3f}")
        if db < 0.5:
            print(f" → Отлично!")
        elif db < 1.0:
            print(f" → Хорошо")
        else:
            print(f" → Нуждается
в улучшении")
```

```
print(f" AMI: {ami:.3f}")

nmi =
normalized_mutual_info_score(true_labels)
print(f" NMI: {nmi:.3f}")

h, c, v =
homogeneity_completeness_v_measure(
    true_labels, labels
)
print(f" Homogeneity:
{h:.3f}")
print(f" Completeness:
{c:.3f}")
print(f" V-measure:
{v:.3f}")

print("=" * 50)

# Использование
comprehensive_evaluation(X, labels,
true_labels)
```

## ◆ 11. Выбор оптимального числа кластеров

```
def find_optimal_clusters(X,
max_k=10):
    """Поиск оптимального К
несколькоими методами"""

    metrics = {
        'k': [],
        'inertia': [],
        'silhouette': [],
        'davies_bouldin': [],
        'calinski_harabasz': []
    }

    for k in range(2, max_k + 1):
        kmeans =
KMeans(n_clusters=k,
random_state=42, n_init=10)
        labels =
kmeans.fit_predict(X)

        metrics['k'].append(k)

    metrics['inertia'].append(kmeans.iner
metrics['silhouette'].append(silhouet
labels))

    metrics['davies_bouldin'].append(davi
labels))

    metrics['calinski_harabasz'].append(c
labels))

    # Визуализация
    fig, axes = plt.subplots(2, 2,
figsize=(14, 10))

    # Inertia
    axes[0, 0].plot(metrics['k'],
metrics['inertia'],
marker='o',
linewidth=2)
    axes[0, 0].set_title('Inertia
(Elbow Method)')
    axes[0, 0].set_xlabel('K')
    axes[0,
0].set_ylabel('Inertia')
    axes[0, 0].grid(alpha=0.3)

    # Silhouette
    axes[0, 1].plot(metrics['k'],
metrics['silhouette'],
marker='s',
linewidth=2, color='green')
    axes[0,
1].set_title('Silhouette Score')
    axes[0, 1].set_xlabel('K')
    axes[0, 1].set_ylabel('Score')
    axes[0, 1].grid(alpha=0.3)

    # Davies-Bouldin
    axes[1, 0].plot(metrics['k'],
metrics['davies_bouldin'],
marker='^',
linewidth=2, color='red')
    axes[1, 0].set_title('Davies-
Bouldin Index')
    axes[1, 0].set_xlabel('K')
    axes[1, 0].set_ylabel('Index
(lower=better)')
    axes[1, 0].grid(alpha=0.3)

    # Calinski-Harabasz
    axes[1, 1].plot(metrics['k'],
metrics['calinski_harabasz'],
marker='d',
linewidth=2, color='purple')
    axes[1, 1].set_title('Calinski-
Harabasz Index')
    axes[1, 1].set_xlabel('K')
    axes[1, 1].set_ylabel('Index
(higher=better)')
    axes[1, 1].grid(alpha=0.3)
```

```
plt.tight_layout()
plt.show()

# Рекомендации
k_silhouette = metrics['k']
[np.argmax(metrics['silhouette'])]
k_db = metrics['k']
[np.argmin(metrics['davies_bouldin'])]
k_ch = metrics['k']
[np.argmax(metrics['calinski_harabasz'])

print("Рекомендуемое K:")
print(f"  По Silhouette:
{k_silhouette}")
print(f"  По Davies-Bouldin:
{k_db}")
print(f"  По Calinski-Harabasz:
{k_ch}")

return metrics

# Использование
metrics = find_optimal_clusters(X,
max_k=10)
```

## ◆ 12. Чек-лист оценки кластеризации

1.  Проверить базовые характеристики  
(число кластеров, размеры)
2.  Вычислить Silhouette Score (>0.5  
хорошо)
3.  Проверить Davies-Bouldin Index  
(<1.0 хорошо)
4.  Рассчитать Calinski-Harabasz Index
5.  Использовать Elbow Method для  
выбора K
6.  Визуализировать Silhouette plot
7.  Если есть метки - вычислить ARI,  
AMI, V-measure
8.  Сравнить несколько значений K
9.  Визуализировать кластеры (PCA/t-
SNE)
10.  Проверить стабильность (разные  
random\_state)

# 🎯 Архитектуры CNN (LeNet, AlexNet, VGG)

 Январь 2026

## ◆ 1. LeNet-5 (1998)

**Пионер CNN:** одна из первых сверточных сетей, разработана Yann LeCun

- **Архитектура:** Conv(6) → Pool → Conv(16) → Pool → FC(120) → FC(84) → FC(10)
- **Применение:** распознавание рукописных цифр MNIST
- **Особенности:** использует tanh активацию, размер входа  $32 \times 32$
- **Параметры:** ~60K параметров

```
import torch.nn as nn

class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)  # 32x32 -
> 28x28
        self.pool = nn.AvgPool2d(2, 2)  # 28x28 -
> 14x14
        self.conv2 = nn.Conv2d(6, 16, 5) # 14x14 -
> 10x10
        # После pool: 10x10 -> 5x5
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(torch.tanh(self.conv1(x)))
        x = self.pool(torch.tanh(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```

## ◆ 2. AlexNet (2012)

**Революция ImageNet:** победитель ILSVRC 2012, снизил ошибку до 16.4%

- **Архитектура:** 5 conv + 3 FC слоя, всего ~60M параметров
- **Инновации:**
  - ReLU активация (вместо tanh/sigmoid)
  - Dropout для регуляризации
  - Data augmentation (перевороты, обрезки)
  - Local Response Normalization (LRN)
  - Обучение на GPU
- **Вход:** 224×224×3 RGB изображения

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11,
                     stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),

            nn.Conv2d(64, 192, kernel_size=5,
                     padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),

            nn.Conv2d(192, 384, kernel_size=3,
                     padding=1),
            nn.ReLU(inplace=True),

            nn.Conv2d(384, 256, kernel_size=3,
                     padding=1),
            nn.ReLU(inplace=True),

            nn.Conv2d(256, 256, kernel_size=3,
                     padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
```

```
)  
def forward(self, x):  
    x = self.features(x)  
    x = x.view(x.size(0), 256 * 6 * 6)  
    x = self.classifier(x)  
    return x
```

## ◆ 3. VGG (2014)

**Глубина и простота:** очень глубокие сети с простой архитектурой

- **Варианты:** VGG-16 (16 слоёв), VGG-19 (19 слоёв)
- **Принцип:** только 3×3 свёртки, удвоение числа фильтров после каждого pooling
- **Параметры:** VGG-16 ~138M, VGG-19 ~144M
- **Особенности:** несколько 3×3 свёрток дают такое же receptive field как одна большая

```
# VGG-16 архитектура
class VGG16(nn.Module):
    def __init__(self, num_classes=1000):
        super(VGG16, self).__init__()
        self.features = nn.Sequential(
            # Block 1
            nn.Conv2d(3, 64, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # Block 2
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # Block 3
            nn.Conv2d(128, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # Block 4
            nn.Conv2d(256, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),

            # Block 5
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
```

```

        nn.Conv2d(512, 512, 3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, 3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(2, 2),
    )
    self.classifier = nn.Sequential(
        nn.Linear(512 * 7 * 7, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(4096, num_classes),
    )
)

```

## ◆ 4. Сравнение архитектур

| Архитектура | Год  | Слои | Параметры | Top-5 Error (ImageNet) |
|-------------|------|------|-----------|------------------------|
| LeNet-5     | 1998 | 7    | 60K       | N/A (MNIST)            |
| AlexNet     | 2012 | 8    | 60M       | 16.4%                  |
| VGG-16      | 2014 | 16   | 138M      | 7.3%                   |
| VGG-19      | 2014 | 19   | 144M      | 7.3%                   |
| GoogLeNet   | 2014 | 22   | 6.8M      | 6.7%                   |
| ResNet-50   | 2015 | 50   | 25.5M     | 3.6%                   |

## ◆ 5. GoogLeNet / Inception (2014)

**Inception модули:** параллельные свёртки разных размеров

- **Идея:** вместо выбора размера свёртки, применить все одновременно
- **Модуль Inception:**  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  свёртки +  $3 \times 3$  pooling параллельно
- **1x1 свёртки:** снижение размерности перед дорогими операциями
- **Auxiliary classifiers:** дополнительные выходы для борьбы с vanishing gradient

```

class InceptionModule(nn.Module):
    def __init__(self, in_channels, out_1x1,
red_3x3, out_3x3,
red_5x5, out_5x5, out_pool):
        super(InceptionModule, self).__init__()

        # 1x1 conv
        self.branch1 = nn.Sequential(
            nn.Conv2d(in_channels, out_1x1, 1),
            nn.ReLU(inplace=True)
        )

        # 1x1 -> 3x3 conv
        self.branch2 = nn.Sequential(
            nn.Conv2d(in_channels, red_3x3, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(red_3x3, out_3x3, 3,
padding=1),
            nn.ReLU(inplace=True)
        )

        # 1x1 -> 5x5 conv
        self.branch3 = nn.Sequential(
            nn.Conv2d(in_channels, red_5x5, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(red_5x5, out_5x5, 5,
padding=2),
            nn.ReLU(inplace=True)
        )

        # 3x3 pool -> 1x1 conv
        self.branch4 = nn.Sequential(
            nn.MaxPool2d(3, stride=1, padding=1),
            nn.Conv2d(in_channels, out_pool, 1),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):

```

```

        return torch.cat([
            self.branch1(x),
            self.branch2(x),
            self.branch3(x),
            self.branch4(x)
        ], dim=1)

```

## ◆ 6. Использование предобученных моделей

```

# PyTorch
import torchvision.models as models

# Загрузка предобученных моделей
alexnet = models.alexnet(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
vgg19 = models.vgg19(pretrained=True)

# Перевод в режим inference
alexnet.eval()

# Предсказание
import torch
from PIL import Image
from torchvision import transforms

# Предобработка изображения
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

img = Image.open('image.jpg')
img_tensor = preprocess(img).unsqueeze(0)

with torch.no_grad():
    output = alexnet(img_tensor)
    probabilities =
        torch.nn.functional.softmax(output[0], dim=0)

# Топ-5 предсказания
top5_prob, top5_catid = torch.topk(probabilities, 5)
for i in range(5):
    print(f"top5_catid[{i}]: {top5_catid[i].item()}")
    print(f"top5_prob[{i}]: {top5_prob[i].item():.4f}")

```

## ◆ 7. Fine-tuning для своих данных

```
# Заморозка всех слоёв кроме последнего
vgg16 = models.vgg16(pretrained=True)

# Заморозить параметры
for param in vgg16.parameters():
    param.requires_grad = False

# Заменить последний слой
num_features = vgg16.classifier[6].in_features
vgg16.classifier[6] = nn.Linear(num_features, 10)
# 10 классов

# Обучение
criterion = nn.CrossEntropyLoss()
optimizer =
torch.optim.Adam(vgg16.classifier[6].parameters(),
lr=0.001)

# Обучение только последнего слоя
for epoch in range(10):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = vgg16(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Разморозка всех слоёв для fine-tuning
for param in vgg16.parameters():
    param.requires_grad = True

optimizer = torch.optim.Adam(vgg16.parameters(),
lr=0.0001)
```

## ◆ 8. Feature extraction

```
# Использование CNN как feature extractor
vgg16 = models.vgg16(pretrained=True)
vgg16.eval()

# Удаление classifier
feature_extractor =
nn.Sequential(*list(vgg16.children())[:-1])

# Извлечение признаков
features_list = []
with torch.no_grad():
    for inputs, _ in data_loader:
        features = feature_extractor(inputs)
        features = features.view(features.size(0),
-1)

        features_list.append(features.cpu().numpy())

    features = np.vstack(features_list)
    print(f"Размерность признаков: {features.shape}")

# Использование признаков для классификации
from sklearn.svm import SVC
from sklearn.model_selection import
train_test_split

X_train, X_test, y_train, y_test =
train_test_split(
    features, labels, test_size=0.2,
    random_state=42
)

svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)
accuracy = svm.score(X_test, y_test)
print(f"Accuracy: {accuracy:.3f}")
```

## ◆ 9. Keras/TensorFlow реализация

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16,
AlexNet
from tensorflow.keras import layers, models

# Загрузка предобученной модели
base_model = VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)

# Создание своей модели
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Заморозка базовой модели
base_model.trainable = False

# Компиляция
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Обучение
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=val_dataset
)
```

## ◆ 10. Эволюция идей

- **LeNet → AlexNet:** ReLU, dropout, GPU, data augmentation
- **AlexNet → VGG:** глубже, проще (только 3×3), систематичность
- **VGG → GoogLeNet:** параллельные свёртки, 1×1 для снижения вычислений
- **GoogLeNet → ResNet:** skip connections, сверхглубокие сети (152+ слоя)
- **Далее:** DenseNet, EfficientNet, Vision Transformers

## ◆ 11. Когда использовать каждую архитектуру

| Архитектура | Когда использовать                               |
|-------------|--------------------------------------------------|
| LeNet       | Простые задачи, малые изображения (28×28)        |
| AlexNet     | Базовый transfer learning, обучающие примеры     |
| VGG         | Feature extraction, визуализация, style transfer |
| GoogLeNet   | Ограниченнная память, нужна эффективность        |
| ResNet      | Современный выбор, высокая точность              |

## ◆ 12. Оптимизация памяти и скорости

```
# Mixed precision training (PyTorch)
from torch.cuda.amp import autocast, GradScaler
scaler = GradScaler()

for epoch in range(epochs):
    for inputs, labels in train_loader:
        optimizer.zero_grad()

        # Forward pass с autocast
        with autocast():
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        # Backward pass
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

# Gradient checkpointing для экономии памяти
import torch.utils.checkpoint as checkpoint

class VGGWithCheckpointing(nn.Module):
    def forward(self, x):
        x = checkpoint.checkpoint(self.block1, x)
        x = checkpoint.checkpoint(self.block2, x)
        # ...
        return x
```

## ◆ 13. Чек-лист использования

1.  Выбрать архитектуру под задачу и ресурсы
2.  Загрузить предобученную модель (ImageNet)
3.  Предобработать данные (resize, normalize)
4.  Заморозить слои для fine-tuning
5.  Заменить последний слой под свои классы
6.  Обучить только новые слои
7.  Разморозить и дообучить с малым lr
8.  Использовать data augmentation
9.  Мониторить overfitting
10.  Оценить на тестовых данных

# CNN (Сверточные нейросети)

 17 Январь 2026

## ◆ 1. Суть

- **Специализация:** обработка изображений
- **Свертка:** поиск паттернов (ребра, текстуры)
- **Локальность:** анализ окрестностей пикселей
- **Иерархия:** от простых к сложным признакам

## ◆ 2. Основные слои

| Слой       | Назначение              |
|------------|-------------------------|
| Conv2D     | Извлечение признаков    |
| MaxPooling | Уменьшение размерности  |
| Flatten    | Преобразование в вектор |
| Dense      | Классификация           |
| Dropout    | Регуляризация           |
| BatchNorm  | Стабилизация обучения   |

## ◆ 3. Простая CNN (PyTorch)

```
import torch
import torch.nn as nn

class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3,
padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 8 * 8, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

## ◆ 4. Простая CNN (TensorFlow)

```
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3),
activation='relu',
input_shape=(32, 32, 3),
padding='same'),
    keras.layers.MaxPooling2D((2, 2)),

    keras.layers.Conv2D(64, (3, 3),
activation='relu',
padding='same'),
    keras.layers.MaxPooling2D((2, 2)),

    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

## ◆ 5. Параметры свертки

| Параметр     | Описание               | Типичные значения |
|--------------|------------------------|-------------------|
| in_channels  | Входные каналы (RGB=3) | 1, 3              |
| out_channels | Количество фильтров    | 32, 64, 128, 256  |
| kernel_size  | Размер фильтра         | 3, 5, 7           |
| stride       | Шаг свертки            | 1, 2              |
| padding      | Дополнение границ      | 'same', 'valid'   |

## ◆ 6. Типы пулинга

**Max Pooling** (по умолчанию):

```
nn.MaxPool2d(kernel_size=2, stride=2)
```

**Average Pooling:**

```
nn.AvgPool2d(kernel_size=2, stride=2)
```

**Global Average Pooling:**

```
nn.AdaptiveAvgPool2d((1, 1)) # Любой размер -> 1x1
```

## ◆ 7. Data Augmentation

```
from torchvision import transforms
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomCrop(32, padding=4),
    transforms.ColorJitter(brightness=0.2,
    contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)))
])

# TensorFlow/Keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1
)
```

## ◆ 9. Когда использовать

### ✓ Хорошо

- ✓ Классификация изображений
- ✓ Детекция объектов
- ✓ Сегментация
- ✓ Распознавание лиц
- ✓ 1D-данные (временные ряды, текст)

### ✗ Плохо

- ✗ Табличные данные
- ✗ Малый датасет (используйте Transfer Learning)
- ✗ Нужна интерпретируемость
- ✗ Ограничены вычислительные ресурсы

## ◆ 10. Расчёт размеров

**Размер после свертки:**

```
output_size = (input_size - kernel_size +
2*padding) / stride + 1

# Пример: 32x32 изображение, kernel=3, padding=1,
stride=1
output = (32 - 3 + 2*1) / 1 + 1 = 32
# Размер не изменился благодаря padding='same'
```

**Размер после pooling:**

```
output_size = input_size / pool_size

# Пример: 32x32 изображение, MaxPool2d(2)
output = 32 / 2 = 16
```

## ◆ 11. Популярные архитектуры

| Архитектура  | Год  | Особенности              |
|--------------|------|--------------------------|
| LeNet-5      | 1998 | Первая CNN               |
| AlexNet      | 2012 | ReLU, Dropout            |
| VGG          | 2014 | Глубокая, 3x3 фильтры    |
| ResNet       | 2015 | Skip connections         |
| Inception    | 2015 | Многомасштабные фильтры  |
| EfficientNet | 2019 | Баланс точность/скорость |

## ◆ 12. Чек-лист

- [ ] Нормализовать изображения (0-1 или -1 до 1)
- [ ] Использовать data augmentation
- [ ] Начать с простой архитектуры
- [ ] Использовать BatchNormalization
- [ ] Добавить Dropout для регуляризации
- [ ] Попробовать Transfer Learning
- [ ] Использовать GPU для ускорения
- [ ] Мониторить train/val loss

### 💡 Объяснение заказчику:

«CNN работает как наши глаза: сначала видит простые детали (линии, углы), потом комбинирует их в сложные объекты (глаза, нос), и в итоге распознает целую картину (лицо человека)».

# CNN для временных рядов (TCN)

17 Январь 2026

## ◆ 1. Суть TCN

- **TCN:** Temporal Convolutional Networks
- **1D Conv:** свертки по временной оси
- **Dilated:** расширенные свертки
- **Causal:** только прошлое

### Преимущества:

- Параллельная обработка
- Длинные зависимости
- Быстрее RNN
- Стабильное обучение

## ◆ 2. Архитектура

Input → [Dilated Conv] → [Residual] → Output

Dilation rates: 1, 2, 4, 8, 16...  
Receptive field =  $2^n * \text{kernel\_size}$

### Пример:

Kernel=3, Dilations=[1,2,4,8]  
Receptive field = 31 timesteps

## ◆ 3. PyTorch реализация

```
import torch
import torch.nn as nn

class TCNBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, dilation):
        super().__init__()
        self.conv = nn.Conv1d(
            in_channels, out_channels,
            kernel_size, padding='same',
            dilation=dilation
        )
        self.bn = nn.BatchNorm1d(out_channels)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        out = self.conv(x)
        out = self.bn(out)
        out = self.relu(out)
        out = self.dropout(out)
        return out

class TCN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super().__init__()
        layers = []
        for i in range(num_layers):
            dilation = 2 ** i
            layers.append(TCNBlock(
                input_size if i == 0 else
                hidden_size,
                hidden_size, kernel_size=3,
                dilation=dilation
            ))
        self.network = nn.Sequential(*layers)
        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        out = self.network(x)
        out = out[:, :, -1] # last timestep
        return self.fc(out)
```

## ◆ 4. Dilated Convolutions

### Формула:

$$\text{output}[t] = \sum \text{weight}[k] * \text{input}[t - k * \text{dilation}]$$

Dilation=1: обычная свертка  
Dilation=2: пропускаем каждый второй  
Dilation=4: пропускаем каждый четвертый

## ◆ 5. Causal Padding

```
# Только прошлое, не будущее
padding = (kernel_size - 1) * dilation

conv = nn.Conv1d(in_ch, out_ch, kernel_size,
                padding=padding,
                dilation=dilation)

# Обрезать будущее
def causal_forward(x):
    out = conv(x)
    return out[:, :, :-padding]
```

## ◆ 6. Residual Connections

```
class ResidualTCN(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.tcn = TCNBlock(channels, channels)
        self.residual = nn.Conv1d(channels, channels, 1)

    def forward(self, x):
        out = self.tcn(x)
        residual = self.residual(x)
        return out + residual
```

## ◆ 7. Обучение TCN

```
model = TCN(input_size=1, hidden_size=64,
             num_layers=4)
optimizer = torch.optim.Adam(model.parameters(),
                             lr=0.001)
criterion = nn.MSELoss()

for epoch in range(100):
    for X_batch, y_batch in dataloader:
        # X: (batch, channels, seq_len)
        X_batch = X_batch.transpose(1, 2)

        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

## ◆ 8. TCN vs RNN

| Критерий      | TCN        | RNN/LSTM  |
|---------------|------------|-----------|
| Скорость      | Быстрее    | Медленнее |
| Параллелизм   | Да         | Нет       |
| Память        | Больше     | Меньше    |
| Gradient flow | Стабильный | Проблемы  |

## ◆ 9. Примеры применения

- **Финансы:** прогноз цен акций
- **Энергетика:** прогноз нагрузки
- **Погода:** прогноз температуры
- **Аномалии:** детекция в сенсорах

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Длинные последовательности
- ✓ Нужна скорость inference
- ✓ Много вычислительных ресурсов
- ✓ Стабильное обучение важно

### ✗ Плохо

- ✗ Ограничена память GPU
- ✗ Очень короткие последовательности
- ✗ Нужна малая латентность

## ◆ 11. Чек-лист

- [ ] Подготовить данные в формате (batch, channels, seq\_len)
- [ ] Выбрать количество слоев (4-8)
- [ ] Настроить dilation rates
- [ ] Добавить residual connections
- [ ] Использовать Dropout (0.1-0.2)
- [ ] Применить causal padding
- [ ] Сравнить с LSTM baseline
- [ ] Визуализировать receptive field

## Объяснение заказчику:

«TCN — это как смотреть на временной ряд через увеличительное стекло с разным увеличением одновременно: близкие точки видим детально, дальние — общий тренд. И всё это быстрее, чем RNN».

# CNN Visualization Techniques

 17 Январь 2026

## ◆ 1. Зачем визуализировать CNN?

- **Интерпретируемость:** понять, что видит сеть
- **Отладка:** найти проблемы в обучении
- **Доверие:** объяснить предсказания
- **Улучшение:** идеи для архитектуры
- **Научное понимание:** как работает deep learning

## ◆ 2. Категории методов

| Метод         | Что показывает              | Сложность |
|---------------|-----------------------------|-----------|
| Фильтры       | Что детектируют первые слои | Низкая    |
| Feature Maps  | Активации на разных слоях   | Низкая    |
| Grad-CAM      | Важные области для класса   | Средняя   |
| Saliency Maps | Влияние каждого пикселя     | Средняя   |
| DeepDream     | Что сеть "видит" в шуме     | Высокая   |
| Neural Style  | Комбинация стиля и контента | Высокая   |

## ◆ 3. Визуализация фильтров (весов)

```
import torch
import matplotlib.pyplot as plt
import torchvision.models as models

# Загрузка модели
model = models.vgg16(pretrained=True)

# Получить веса первого слоя
first_layer = model.features[0]
filters = first_layer.weight.data.cpu()

# filters: (64, 3, 3, 3) = (out_ch, in_ch, H, W)
print(f"Filter shape: {filters.shape}")

# Визуализация
fig, axes = plt.subplots(8, 8, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    if i < filters.shape[0]:
        # Нормализация для визуализации
        f = filters[i].permute(1, 2, 0) # (3, 3, 3) -> (3, 3, 3)
        f = (f - f.min()) / (f.max() - f.min())
        ax.imshow(f)
        ax.axis('off')
plt.tight_layout()
plt.show()
```

## ◆ 4. Визуализация Feature Maps

```
import torch
import torchvision.transforms as transforms
from PIL import Image

# Загрузка изображения
img = Image.open('cat.jpg')
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])
img_tensor = transform(img).unsqueeze(0)

# Hook для захвата активаций
activations = {}
def get_activation(name):
    def hook(model, input, output):
        activations[name] = output.detach()
        return hook

    model.features[0].register_forward_hook(get_activation)
    model.features[5].register_forward_hook(get_activation)

# Регистрация hook
model.eval()
with torch.no_grad():
    output = model(img_tensor)

# Визуализация
conv1_act = activations['conv1'].squeeze()
fig, axes = plt.subplots(8, 8, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
    if i < conv1_act.shape[0]:
        ax.imshow(conv1_act[i], cmap='viridis')
        ax.axis('off')
plt.show()
```

## ◆ 5. Saliency Maps (Vanilla Gradient)

**Идея:** вычислить градиент выхода по входу

```
def compute_saliency_map(model, image,
target_class):
    """
        Вычислить saliency map для изображения
    """
    # Включить градиенты для входа
    image.requires_grad = True

    # Forward pass
    model.eval()
    output = model(image)

    # Backward от целевого класса
    model.zero_grad()
    target = output[0, target_class]
    target.backward()

    # Градиент по входу
    saliency = image.grad.data.abs()
    saliency = saliency.max(dim=1)[0] # max по
    каналам

    return saliency.squeeze().cpu()

# Использование
img_tensor.requires_grad = True
saliency = compute_saliency_map(model, img_tensor,
target_class=281)

plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.imshow(img)
plt.title('Original')
plt.subplot(122)
plt.imshow(saliency, cmap='hot')
plt.title('Saliency Map')
plt.show()
```

## ◆ 6. Grad-CAM (Class Activation Mapping)

**Преимущество:** показывает важные области для конкретного класса

```
import torch.nn.functional as F

def grad_cam(model, image, target_class,
target_layer):
    """
        Вычислить Grad-CAM
    """
    # Захват градиентов и активаций
    gradients = []
    activations = []

    def backward_hook(module, grad_input,
grad_output):
        gradients.append(grad_output[0])

    def forward_hook(module, input, output):
        activations.append(output)

    # Регистрация hooks
    handle_fw =
target_layer.register_forward_hook(forward_hook)
    handle_bw =
target_layer.register_full_backward_hook(backward_ho

    # Forward
    model.eval()
    output = model(image)

    # Backward от целевого класса
    model.zero_grad()
    target = output[0, target_class]
    target.backward()

    # Вычисление Grad-CAM
    grads = gradients[0].cpu().data
    acts = activations[0].cpu().data

    # Веса = среднее по spatial dims
    weights = grads.mean(dim=(2, 3), keepdim=True)

    # Взвешенная сумма активаций
    cam = (weights * acts).sum(dim=1,
keepdim=True)
    cam = F.relu(cam) # ReLU

    # Нормализация и resize
    cam = F.interpolate(cam, size=(224, 224),
mode='bilinear')
    cam = cam - cam.min()
    cam = cam / cam.max()
```

```
handle_fw.remove()
handle_bw.remove()

return cam.squeeze().numpy()

# Использование
target_layer = model.features[-1] # последний
conv слой
cam = grad_cam(model, img_tensor,
target_class=281,
target_layer=target_layer)

# Наложение на изображение
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.imshow(img)
plt.subplot(122)
plt.imshow(img)
plt.imshow(cam, cmap='jet', alpha=0.5)
plt.title('Grad-CAM')
plt.show()
```

## ◆ 7. Guided Grad-CAM

**Комбинация:** Grad-CAM × Guided Backpropagation

- Grad-CAM: где важно (coarse)
- Guided Backprop: что важно (fine)
- Комбинация: четкая визуализация важных деталей

```
def guided_backprop(model, image, target_class):
    """
    Guided Backpropagation
    """
    # Заменить ReLU на Guided ReLU
    class GuidedReLU(torch.autograd.Function):
        @staticmethod
        def forward(ctx, input):
            return input.clamp(min=0)

        @staticmethod
        def backward(ctx, grad_output):
            # Пропускать только положительные
            # градиенты
            return grad_output.clamp(min=0)

    # Модифицировать модель (упрощенно)
    image.requires_grad = True
    output = model(image)

    model.zero_grad()
    target = output[0, target_class]
    target.backward()

    # Градиент с guided backprop
    guided_grads = image.grad.data

    return guided_grads

# Guided Grad-CAM = Grad-CAM * Guided Backprop
guided_grads = guided_backprop(model, img_tensor,
                                281)
guided_cam = cam * guided_grads.cpu().numpy()

plt.imshow(guided_cam.transpose(1, 2, 0))
plt.title('Guided Grad-CAM')
plt.show()
```

## ◆ 8. Occlusion Sensitivity

**Идея:** закрывать части изображения и смотреть на изменение предсказания

```
def occlusion_sensitivity(model, image,
                           target_class,
                           patch_size=32,
                           stride=8):
    """
    Вычислить Occlusion Sensitivity
    """
    model.eval()
    H, W = image.shape[2:]

    # Базовый score
    with torch.no_grad():
        base_output = model(image)
        base_score = base_output[0,
                               target_class].item()

    heatmap = torch.zeros((H // stride, W // stride))

    # Закрывать патчи
    for i in range(0, H - patch_size, stride):
        for j in range(0, W - patch_size, stride):
            # Копия изображения
            occluded = image.clone()
            # Закрыть патч (серым или черным)
            occluded[:, :, i:i+patch_size,
                      j:j+patch_size] = 0.5

            # Предсказание
            with torch.no_grad():
                output = model(occluded)
                score = output[0,
                               target_class].item()

            # Разница
            heatmap[i//stride, j//stride] = base_score - score

    return heatmap.numpy()

# Использование
occlusion_map = occlusion_sensitivity(model,
   img_tensor,
   281)
plt.imshow(occlusion_map, cmap='hot')
plt.title('Occlusion Sensitivity')
plt.show()
```

## ◆ 9. DeepDream

**Идея:** максимизировать активации на определенном слое

```
def deep_dream(model, image, layer, iterations=20,
               lr=0.01):
    """
    DeepDream visualization
    """
    image = image.clone().requires_grad_(True)

    # Hook для захвата активаций
    activations = []
    def hook(module, input, output):
        activations.append(output)
    handle = layer.register_forward_hook(hook)

    for i in range(iterations):
        activations.clear()
        model.zero_grad()

        # Forward
        output = model(image)

        # Оптимизировать сумму активаций
        loss = activations[0].norm()
        loss.backward()

        # Gradient ascent
        with torch.no_grad():
            image += lr * image.grad
            image.grad.zero_()

    handle.remove()

    return image.detach()

# Использование
dreamed = deep_dream(model, img_tensor,
                      model.features[10])

# Визуализация
dreamed_img = dreamed.squeeze().permute(1, 2, 0).cpu()
dreamed_img = (dreamed_img - dreamed_img.min()) /
\ (dreamed_img.max() -
dreamed_img.min())
plt.imshow(dreamed_img)
plt.title('DeepDream')
plt.show()
```

## ◆ 10. Библиотеки для визуализации

| Библиотека       | Особенности               | Установка                |
|------------------|---------------------------|--------------------------|
| Captum           | От PyTorch, много методов | pip install captum       |
| pytorch-grad-cam | Простая в использовании   | pip install grad-cam     |
| torchcam         | CAM методы                | pip install torchcam     |
| tf-keras-vis     | Для TensorFlow/Keras      | pip install tf-keras-vis |

## ◆ 11. Captum пример

```
from captum.attr import (
    IntegratedGradients,
    GradientShap,
    Occlusion,
    LayerGradCam
)

# Integrated Gradients
ig = IntegratedGradients(model)
attributions = ig.attribute(img_tensor,
                           target=281, n_steps=50)

# Gradient SHAP
gs = GradientShap(model)
baseline = torch.zeros_like(img_tensor)
attributions = gs.attribute(img_tensor,
                            baselines=baseline, target=281)

# Layer Grad-CAM
layer_gc = LayerGradCam(model, model.features[-1])
attributions = layer_gc.attribute(img_tensor,
                                  target=281)

# Визуализация
from captum.attr import visualization as viz

viz.visualize_image_attr(
    attributions.squeeze().cpu().permute(1, 2,
0).numpy(),
    original_image=img,
    method='blended_heat_map',
    sign='positive',
    show_colorbar=True
)
```

## ◆ 12. pytorch-grad-cam пример

```
from pytorch_grad_cam import GradCAM, HiResCAM,
ScoreCAM
from pytorch_grad_cam.utils.image import
show_cam_on_image
from pytorch_grad_cam.utils.model_targets import
ClassifierOutputTarget

# Выбор метода
cam = GradCAM(model=model, target_layers=
[model.features[-1]])
# Альтернативы:
# cam = HiResCAM(...)
# cam = ScoreCAM(...)

# Вычисление CAM
targets = [ClassifierOutputTarget(281)]
grayscale_cam = cam(input_tensor=img_tensor,
targets=targets)

# Наложение на изображение
grayscale_cam = grayscale_cam[0, :]
rgb_img = np.array(img) / 255.0
visualization = show_cam_on_image(rgb_img,
grayscale_cam,
use_rgb=True)

plt.imshow(visualization)
plt.title('Grad-CAM visualization')
plt.show()
```

## ◆ 13. Сравнение методов

| Метод                | Скорость | Детализация | Class-specific |
|----------------------|----------|-------------|----------------|
| Vanilla Gradient     | Быстро   | Высокая     | Да             |
| Grad-CAM             | Быстро   | Низкая      | Да             |
| Guided Grad-CAM      | Средне   | Высокая     | Да             |
| Occlusion            | Медленно | Средняя     | Да             |
| Integrated Gradients | Медленно | Высокая     | Да             |
| DeepDream            | Медленно | N/A         | Нет            |

## ◆ 14. Чек-лист

- [ ] Начать с простого: фильтры и feature maps
- [ ] Grad-CAM для понимания важных областей
- [ ] Saliency maps для детализации
- [ ] Использовать библиотеки (Captum, grad-cam)
- [ ] Визуализировать разные слои: ранние vs поздние
- [ ] Проверить на разных классах
- [ ] Сравнить методы на одном изображении
- [ ] Документировать находки для улучшения модели

### 💡 Объяснение заказчику:

«Визуализация CNN — это как рентген для нейронной сети: мы можем увидеть, на какие части изображения модель обращает внимание при принятии решения. Это помогает убедиться, что модель фокусируется на правильных вещах, а не на артефактах».



# Коллаборативная фильтрация

17 Январь 2026

## ◆ 1. Что такое коллаборативная фильтрация?

- Определение:** метод создания рекомендаций на основе поведения пользователей
- Идея:** похожие пользователи любят похожие вещи
- Данные:** матрица пользователь-товар с рейтингами
- Типы:** User-based, Item-based, Model-based
- Применение:** Netflix, Amazon, Spotify

## ◆ 2. Типы коллаборативной фильтрации

| Тип          | Описание                          | Когда использовать                |
|--------------|-----------------------------------|-----------------------------------|
| User-based   | Находит похожих пользователей     | Мало пользователей, много товаров |
| Item-based   | Находит похожие товары            | Много пользователей, мало товаров |
| Model-based  | Матричная факторизация (SVD, ALS) | Большие данные, масштабируемость  |
| Memory-based | Использует всю матрицу            | Малые данные, простота            |

## ◆ 3. User-based CF

Находит похожих пользователей и рекомендует то, что понравилось им:

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Матрица user-item (пользователи x товары)
# Строки - пользователи, столбцы - товары
ratings = np.array([
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [0, 1, 5, 4]
])

# Вычисляем similarity между пользователями
user_similarity = cosine_similarity(ratings)

# Для user 0 находим похожих
user_id = 0
similar_users =
np.argsort(user_similarity[user_id])[::-1][1:]

# Предсказание рейтинга для item 2
item_id = 2
numerator = 0
denominator = 0

for similar_user in similar_users:
    if ratings[similar_user, item_id] > 0:
        sim = user_similarity[user_id, similar_user]
        numerator += sim * ratings[similar_user, item_id]
        denominator += sim

predicted_rating = numerator / denominator if denominator > 0 else 0
print(f"Predicted rating: {predicted_rating:.2f}")
```

## ◆ 4. Item-based CF

Находит похожие товары на те, что понравились пользователю:

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Транспонируем для item similarity
# Строки теперь товары, столбцы - пользователи
ratings_T = ratings.T

# Similarity между товарами
item_similarity = cosine_similarity(ratings_T)

def predict_item_based(user_id, item_id, ratings,
item_sim):
    # Находим товары, которые user уже оценил
    rated_items = np.where(ratings[user_id] > 0)
    [0]

    numerator = 0
    denominator = 0

    for rated_item in rated_items:
        if rated_item != item_id:
            sim = item_sim[item_id, rated_item]
            numerator += sim * ratings[user_id, rated_item]
            denominator += abs(sim)

    return numerator / denominator if denominator > 0 else 0

# Предсказание
prediction = predict_item_based(0, 2, ratings,
item_similarity)
print(f"Predicted rating: {prediction:.2f}")
```

## ◆ 5. Метрики сходства

| Метрика   | Формула/<br>Описание                                                           | Когда использовать         |
|-----------|--------------------------------------------------------------------------------|----------------------------|
| Cosine    | $\cos(\theta) = \mathbf{A} \cdot \mathbf{B} / (\ \mathbf{A}\  \ \mathbf{B}\ )$ | Самая популярная           |
| Pearson   | Корреляция Пирсона                                                             | Учитывает средние значения |
| Jaccard   | $ A \cap B  /  A \cup B $                                                      | Бинарные данные            |
| Euclidean | $\sqrt{\sum(a-b)^2}$                                                           | Расстояние в пространстве  |

```
from sklearn.metrics.pairwise import (
    cosine_similarity,
    euclidean_distances
)
from scipy.stats import pearsonr

# Cosine similarity
cos_sim = cosine_similarity(ratings)

# Pearson correlation
def pearson_similarity(ratings):
    n_users = ratings.shape[0]
    sim_matrix = np.zeros((n_users, n_users))

    for i in range(n_users):
        for j in range(n_users):
            if i != j:
                # Общие оцененные товары
                mask = (ratings[i] > 0) &
(ratings[j] > 0)
                if mask.sum() > 0:
                    corr, _ = pearsonr(ratings[i]
[mask],
[ratings[j]
[mask]])
                    sim_matrix[i,j] = corr

    return sim_matrix
```

## ◆ 6. Библиотека Surprise

Специализированная библиотека для рекомендаций:

```
# Установка
# pip install scikit-surprise

from surprise import Dataset, Reader
from surprise import KNNBasic, SVD
from surprise.model_selection import cross_validate
import pandas as pd

# Загрузка данных
# Формат: user_id, item_id, rating
df = pd.DataFrame({
    'user_id': [1, 1, 1, 2, 2, 3, 3, 3],
    'item_id': [1, 2, 3, 1, 3, 2, 3, 4],
    'rating': [5, 3, 4, 4, 5, 2, 4, 3]
})

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['user_id',
'item_id', 'rating']],
reader)

# User-based CF
sim_options = {
    'name': 'cosine',
    'user_based': True
}
algo = KNNBasic(sim_options=sim_options)

# Кросс-валидация
cross_validate(algo, data, measures=['RMSE',
'MAE'], cv=5)

# Обучение на всех данных
trainset = data.build_full_trainset()
algo.fit(trainset)

# Предсказание
user_id = 1
item_id = 4
prediction = algo.predict(user_id, item_id)
print(f"Predicted rating: {prediction.est:.2f}")
```

## ◆ 7. Item-based с Surprise

```
from surprise import KNNBasic

# Item-based CF
sim_options = {
    'name': 'cosine',
    'user_based': False # Item-based
}

algo = KNNBasic(k=40, sim_options=sim_options)
algo.fit(trainset)

# Получение похожих товаров
inner_id = trainset.to_inner_iid(item_id)
neighbors = algo.get_neighbors(inner_id, k=10)

# Конвертация обратно в внешние ID
similar_items = [trainset.to_raw_iid(inner_id)
for inner_id in neighbors]
print(f"Similar items to {item_id}:
{similar_items}")

# KNNWithMeans - учитывает средние рейтинги
from surprise import KNNWithMeans

algo = KNNWithMeans(k=40, sim_options=sim_options)
algo.fit(trainset)

# KNNBaseline - с baseline estimates
from surprise import KNNBaseline

algo = KNNBaseline(k=40, sim_options=sim_options)
algo.fit(trainset)
```

## ◆ 8. Матричная факторизация (SVD)

Model-based подход, разлагает матрицу на латентные факторы:

```
from surprise import SVD
from surprise.model_selection import GridSearchCV

# SVD (Singular Value Decomposition)
algo = SVD(
    n_factors=100,      # количество латентных
факторов
    n_epochs=20,        # эпох обучения
    lr_all=0.005,       # learning rate
    reg_all=0.02        # регуляризация
)

algo.fit(trainset)
prediction = algo.predict(user_id, item_id)

# Grid Search для подбора параметров
param_grid = {
    'n_factors': [50, 100, 150],
    'n_epochs': [20, 30],
    'lr_all': [0.002, 0.005],
    'reg_all': [0.02, 0.1]
}

gs = GridSearchCV(SVD, param_grid, measures=
['rmse'], cv=3)
gs.fit(data)

print(f"Best RMSE: {gs.best_score['rmse']:.3f}")
print(f"Best params: {gs.best_params['rmse']}")

# Лучшая модель
best_algo = gs.best_estimator['rmse']
best_algo.fit(trainset)
```

## ◆ 9. SVD++ и NMF

```
from surprise import SVDpp, NMF

# SVD++ - учитывает implicit feedback
svdpp = SVDpp(
    n_factors=20,
    n_epochs=20,
    lr_all=0.007,
    reg_all=0.02
)
svdpp.fit(trainset)

# NMF (Non-negative Matrix Factorization)
# Все факторы неотрицательные - лучше
интерпретация
nmf = NMF(
    n_factors=15,
    n_epochs=50,
    biased=True
)
nmf.fit(trainset)

# Получение латентных факторов
user_factors = nmf.pu  # user factors
item_factors = nmf.qi  # item factors

# Предсказание вручную
user_inner_id = trainset.to_inner_uid(user_id)
item_inner_id = trainset.to_inner_iid(item_id)

predicted = (user_factors[user_inner_id] @
            item_factors[item_inner_id].T)
print(f"Manual prediction: {predicted:.2f}")
```

## ◆ 10. Implicit Feedback

Когда есть только факт взаимодействия (клики, просмотры), без явных рейтингов:

```
# pip install implicit

import implicit
from scipy.sparse import csr_matrix

# Матрица user-item (разреженная)
# Значения - количество взаимодействий
user_items = csr_matrix(ratings)

# ALS (Alternating Least Squares)
model = implicit.als.AlternatingLeastSquares(
    factors=50,
    regularization=0.01,
    iterations=15
)

# Обучение (транспонируем для implicit)
model.fit(user_items.T)

# Рекомендации для пользователя
user_id = 0
recommendations = model.recommend(
    user_id,
    user_items[user_id],
    N=10,  # топ-10
    filter_already_liked_items=True
)

for item_id, score in recommendations:
    print(f"Item {item_id}: score {score:.3f}")

# Похожие товары
similar_items = model.similar_items(item_id, N=10)

# BM25 для текстовых данных
model = implicit.bm25.BM25Recommender(K1=100,
B=0.8)
model.fit(user_items.T)
```

## ◆ 11. LightFM - гибридные рекомендации

```
# pip install lightfm

from lightfm import LightFM
from lightfm.data import Dataset
from lightfm.evaluation import precision_at_k

# Создание датасета
dataset = Dataset()
dataset.fit(users=user_ids, items=item_ids)

# Построение матрицы взаимодействий
(interactions, weights) =
dataset.build_interactions(
    [(user, item, rating) for user, item, rating
     in data]
)

# Модель (комбинирует CF и content-based)
model = LightFM(
    loss='warp', # WARP loss для implicit
    no_components=30
)

# Обучение
model.fit(interactions, epochs=10, num_threads=4)

# Предсказания
scores = model.predict(user_ids, item_ids)

# С features (гибридная фильтрация)
# User features
user_features_matrix =
dataset.build_user_features([
    (user_id, ['age:25-34', 'gender:M'])
    for user_id in user_ids
])

# Item features
item_features_matrix =
dataset.build_item_features([
    (item_id, ['category:electronics',
    'brand:samsung'])
    for item_id in item_ids
])

model.fit(
    interactions,
    user_features=user_features_matrix,
    item_features=item_features_matrix,
    epochs=10
)
```

## ◆ 12. Проблема холодного старта

Новые пользователи или товары без истории:

- **Новый пользователь:**

- Запросить первичные предпочтения
- Использовать демографические данные
- Показывать популярные товары
- Гибридный подход (content-based)

- **Новый товар:**

- Content-based фильтрация
- Показывать похожим пользователям
- A/B тестирование
- Использовать метаданные товара

```
# Гибридный подход
def hybrid_recommendation(user_id, n=10):
    # Если пользователь новый
    if user_history[user_id].sum() < 3:
        # Показываем популярные
        popular = ratings.sum(axis=0).argsort()[-n:]
        return popular[:n]
    else:
        # Коллаборативная фильтрация
        return collaborative_filter(user_id, n)

# Популярные товары как baseline
def popular_items(ratings, n=10):
    item_counts = (ratings > 0).sum(axis=0)
    popular = item_counts.argsort()[-n:]
    return popular[:n]
```

## ◆ 13. Метрики оценки

```
from surprise import accuracy
from surprise.model_selection import train_test_split

# Разделение данных
trainset, testset = train_test_split(data,
test_size=0.2)

algo.fit(trainset)
predictions = algo.test(testset)

# RMSE и MAE
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)

# Precision@K и Recall@K
from collections import defaultdict

def precision_recall_at_k(predictions, k=10,
threshold=3.5):
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = {}
    recalls = {}

    for uid, user_ratings in user_est_true.items():
        # Сортировка по предсказанному рейтингу
        user_ratings.sort(key=lambda x: x[0],
reverse=True)

        # Топ-K предсказаний
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])
        n_rel_and_rec_k = sum(((true_r >=
threshold) and (est >= threshold)))
            for (est, true_r) in user_ratings[:k])

        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 0

    return precisions, recalls

precisions, recalls =
precision_recall_at_k(predictions, k=10)
print(f"Precision@10:
{sum(precisions.values())/len(precisions):.3f}")
```

```
print(f"Recall@10:  
{sum(recalls.values())/len(recalls):.3f}")
```

## ◆ 14. Diversity и Serendipity

Разнообразие рекомендаций:

```
import numpy as np

def diversity(recommendations, item_similarity):
    """Среднее расстояние между рекомендованными товарами"""
    n = len(recommendations)
    if n <= 1:
        return 0

    total_dissim = 0
    count = 0

    for i in range(n):
        for j in range(i+1, n):
            # 1 - similarity = dissimilarity
            dissim = 1 -
item_similarity[recommendations[i],
recommendations[j]]
            total_dissim += dissim
            count += 1

    return total_dissim / count

def serendipity(recommendations, user_profile,
item_similarity):
    """Неожиданность рекомендаций"""
    serendipity_score = 0

    for rec_item in recommendations:
        # Насколько товар непохож на то, что user уже видел
        max_sim = max([item_similarity[rec_item,
profile_item]
for profile_item in user_profile])
        serendipity_score += (1 - max_sim)

    return serendipity_score /
len(recommendations)

# Балансировка accuracy и diversity
def rerank_for_diversity(recommendations, scores,
item_sim, lambda_=0.5):
    """MMR (Maximal Marginal Relevance)
reranking"""
    reranked = []
    candidates = list(recommendations)

    # Первый - самый релевантный
    best_idx = np.argmax([scores[i] for i in
candidates])
    reranked.append(candidates.pop(best_idx))
```

```
while candidates:
    mmr_scores = []
    for item in candidates:
        # Relevance - lambda * max_similarity_to_selected
        relevance = scores[item]
        max_sim = max([item_sim[item,
selected]
for selected in
reranked])
        mmr = lambda_ * relevance - (1 -
lambda_) * max_sim
        mmr_scores.append(mmr)

    best_idx = np.argmax(mmr_scores)
    reranked.append(candidates.pop(best_idx))

return reranked
```

## ◆ 15. Масштабирование и оптимизация

- Разреженные матрицы:** используйте `scipy.sparse`
- Предвычисления:** кэшируйте `similarity matrices`
- Approximate NN:** Annoy, FAISS для быстрого поиска
- Батчинг:** обрабатывайте пользователей батчами
- Sampling:** negative sampling для implicit feedback
- Incremental updates:** обновление без полного переобучения

```
# FAISS для быстрого поиска похожих
# pip install faiss-cpu

import faiss
import numpy as np

# Создание индекса
dimension = item_embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(item_embeddings.astype('float32'))

# Быстрый поиск k ближайших
k = 10
distances, indices = index.search(
    query_embedding.reshape(1, -1).astype('float32'),
    k
)

# Approximate search (быстрее на больших данных)
index = faiss.IndexIVFFlat(
    faiss.IndexFlatL2(dimension),
    dimension,
    100 # количество кластеров
)
index.train(item_embeddings.astype('float32'))
index.add(item_embeddings.astype('float32'))
index.nprobe = 10 # количество кластеров для поиска

distances, indices = index.search(query_embedding, k)
```

## ◆ 16. Онлайн обучение

```
# Инкрементальное обновление
class IncrementalCF:
    def __init__(self, n_factors=20, learning_rate=0.01):
        self.n_factors = n_factors
        self.lr = learning_rate
        self.user_factors = {}
        self.item_factors = {}

    def update(self, user_id, item_id, rating):
        # Инициализация если новый
        if user_id not in self.user_factors:
            self.user_factors[user_id] = np.random.randn(self.n_factors) * 0.01
        if item_id not in self.item_factors:
            self.item_factors[item_id] = np.random.randn(self.n_factors) * 0.01

        # Предсказание
        pred = self.user_factors[user_id] @ self.item_factors[item_id]
        error = rating - pred

        # SGD update
        user_update = error * self.item_factors[item_id]
        item_update = error * self.user_factors[user_id]

        self.user_factors[user_id] += self.lr * user_update
        self.item_factors[item_id] += self.lr * item_update

    def predict(self, user_id, item_id):
        if user_id not in self.user_factors or item_id not in self.item_factors:
            return 0 # или средний рейтинг
        return self.user_factors[user_id] @ self.item_factors[item_id]

    # Использование
model = IncrementalCF()
for user, item, rating in new_interactions:
    model.update(user, item, rating)
```

## ◆ 17. Лучшие практики

- Нормализация:** учитывайте bias пользователей и товаров
- Временной фактор:** свежие взаимодействия важнее
- Implicit vs Explicit:** выбирайте правильный подход
- Валидация:** временная валидация (не случайная)
- Холодный старт:** гибридный подход
- Diversity:** избегайте filter bubble
- A/B тесты:** проверяйте метрики бизнеса
- Мониторинг:** отслеживайте качество в production

## ◆ 18. Когда использовать

### ✓ Хорошо подходит

- ✓ Много пользователей и истории взаимодействий
- ✓ Товары/контент похожи по природе
- ✓ Важна персонализация
- ✓ E-commerce, стриминг, новости
- ✓ Есть explicit или implicit feedback

### ✗ Плохо подходит

- ✗ Новая платформа (холодный старт)
- ✗ Очень разнородный контент
- ✗ Мало данных о взаимодействиях
- ✗ Нужна объяснимость рекомендаций
- ✗ Критична актуальность (trending)

## ◆ 19. Чек-лист реализации

- [ ] Собрать данные о взаимодействиях
- [ ] Выбрать тип CF (user/item/model-based)
- [ ] Обработать разреженность данных
- [ ] Выбрать метрику similarity
- [ ] Реализовать baseline (популярные товары)
- [ ] Обучить модель CF
- [ ] Настроить гиперпараметры
- [ ] Решить проблему холодного старта
- [ ] Добавить diversity в рекомендации
- [ ] Валидация на исторических данных
- [ ] A/B тест в production
- [ ] Настроить мониторинг метрик

### Объяснение заказчику:

«Коллаборативная фильтрация — это как сарафанное радио в цифровом мире: система анализирует, что нравится похожим на вас пользователям, и рекомендует вам то, что с высокой вероятностью понравится. Именно так работают рекомендации Netflix и Amazon».

# Коллаборативная фильтрация: User-based & Item-based

 Январь 2026

## ◆ 1. Суть коллаборативной фильтрации

**Collaborative Filtering (CF):** рекомендации на основе поведения схожих пользователей или объектов

- **Идея:** "Люди, похожие на вас, выбирают это"
- **Входные данные:** матрица взаимодействий пользователь-объект (рейтинги, клики, покупки)
- **Два подхода:** User-based CF и Item-based CF
- **Преимущества:** не требует знаний о контенте, адаптируется к предпочтениям
- **Недостатки:** cold start, проблема разреженности данных

 CF использует "мудрость толпы" для персонализации рекомендаций

## ◆ 2. User-based Collaborative Filtering

**Подход:** найти похожих пользователей и рекомендовать то, что нравится им

### Алгоритм:

1. Вычислить схожесть между пользователями
2. Найти k наиболее похожих пользователей (соседей)
3. Агрегировать их рейтинги для получения рекомендаций
4. Рекомендовать топ-N объектов

### Формула предсказания рейтинга:

$$\hat{r}_{ui} = \bar{r}_u + \sum (\text{sim}(u, v) * (r_{vi} - \bar{r}_v)) / \sum |\text{sim}(u, v)|$$

где:

$\hat{r}_{ui}$  - предсказанный рейтинг пользователя  $u$  для объекта  $i$   
 $\bar{r}_u$  - средний рейтинг пользователя  $u$   
 $\text{sim}(u, v)$  - схожесть между пользователями  $u$  и  $v$   
 $r_{vi}$  - рейтинг пользователя  $v$  для объекта  $i$

## ◆ 3. Item-based Collaborative Filtering

**Подход:** рекомендовать объекты, похожие на те, что нравятся пользователю

### Алгоритм:

1. Вычислить схожесть между объектами
2. Для каждого объекта, который понравился пользователю, найти k наиболее похожих объектов
3. Агрегировать рейтинги для получения рекомендаций
4. Рекомендовать топ-N объектов

### Формула предсказания:

$$\hat{r}_{ui} = \sum (\text{sim}(i, j) * r_{uj}) / \sum |\text{sim}(i, j)|$$

где:

$\text{sim}(i, j)$  - схожесть между объектами  $i$  и  $j$   
 $r_{uj}$  - рейтинг пользователя  $u$  для объекта  $j$

 Item-based CF часто работает лучше из-за стабильности схожести объектов

## ◆ 4. Метрики схожести

| Метрика                    | Формула                                                                                          | Особенности          |
|----------------------------|--------------------------------------------------------------------------------------------------|----------------------|
| <b>Косинусное сходство</b> | $\cos(u,v) = (u \cdot v) / (\ u\  \cdot \ v\ )$                                                  | Игнорирует масштаб   |
| <b>Корреляция Пирсона</b>  | $\rho = \frac{\sum((x-\bar{x})(y-\bar{y}))}{\sqrt{(\sum(x-\bar{x})^2 \cdot \sum(y-\bar{y})^2)}}$ | Учитывает отклонения |
| <b>Adjusted Cosine</b>     | Косинус с нормализацией по средним                                                               | Для item-based CF    |
| <b>Jaccard</b>             | $ A \cap B  /  A \cup B $                                                                        | Для бинарных данных  |

## ◆ 5. User-based CF: Python реализация

```

import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Матрица рейтингов (пользователи x объекты)
# 0 означает отсутствие рейтинга
ratings = np.array([
    [5, 3, 0, 1],
    [4, 0, 0, 1],
    [1, 1, 0, 5],
    [1, 0, 0, 4],
    [0, 1, 5, 4],
])

# Вычисляем схожесть между пользователями
# Заменяем 0 на NaN для правильного подсчета
ratings_for_sim = ratings.copy().astype(float)
ratings_for_sim[ratings_for_sim == 0] = np.nan

# Косинусное сходство (замена NaN на 0)
user_sim = cosine_similarity(
    np.nan_to_num(ratings_for_sim)
)

# Предсказание рейтинга для пользователя 0,
# объекта 2
def predict_rating(user_id, item_id, ratings,
user_sim, k=2):
    # Находим k наиболее похожих пользователей
    sim_scores = user_sim[user_id].copy()
    sim_scores[user_id] = -1 # Исключаем самого себя

    # Находим пользователей, оценивших данный
    # объект
    rated_mask = ratings[:, item_id] > 0
    sim_scores[~rated_mask] = -1

    # Топ-k похожих пользователей
    top_k_users = np.argsort(sim_scores)[-k:]
    [::-1]

    # Взвешенное среднее рейтингов
    weights = user_sim[user_id, top_k_users]
    ratings_by_neighbors = ratings[top_k_users,
item_id]

    if weights.sum() == 0:
        return ratings[user_id][ratings[user_id] >
0].mean()

    pred = np.dot(weights, ratings_by_neighbors) /
weights.sum()
    return pred

```

```

# Предсказываем рейтинг
pred = predict_rating(0, 2, ratings, user_sim,
k=2)
print(f"Предсказанный рейтинг: {pred:.2f}")

```

## ◆ 6. Item-based CF: Python реализация

```

import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Вычисляем схожесть между объектами
# Транспонируем матрицу (объекты x пользователи)
item_sim = cosine_similarity(
    np.nan_to_num(ratings_for_sim.T)
)

def predict_rating_item_based(user_id, item_id,
                             ratings,
                             item_sim, k=2):
    # Находим объекты, оцененные пользователем
    user_ratings = ratings[user_id]
    rated_items = np.where(user_ratings > 0)[0]

    if len(rated_items) == 0:
        return ratings[:, item_id][ratings[:, item_id] > 0].mean()

    # Схожесть с оцененными объектами
    sim_scores = item_sim[item_id, rated_items]

    # Топ-k похожих объектов
    top_k_idx = np.argsort(sim_scores)[-k:][::-1]
    top_k_items = rated_items[top_k_idx]

    # Взвешенное среднее
    weights = item_sim[item_id, top_k_items]
    ratings_by_user = user_ratings[top_k_items]

    if weights.sum() == 0:
        return user_ratings[user_ratings > 0].mean()

    pred = np.dot(weights, ratings_by_user) / weights.sum()
    return pred

# Предсказываем рейтинг
pred_item = predict_rating_item_based(0, 2,
                                      ratings, item_sim, k=2)
print(f"Предсказанный рейтинг (item-based): {pred_item:.2f}")

```

## ◆ 7. Использование библиотеки Surprise

```

from surprise import KNNBasic, Dataset, Reader
from surprise.model_selection import cross_validate

# Подготовка данных
data = [
    ('user1', 'item1', 5),
    ('user1', 'item2', 3),
    ('user2', 'item1', 4),
    ('user2', 'item4', 1),
    ('user3', 'item1', 1),
    # ... больше данных
]

reader = Reader(rating_scale=(1, 5))
dataset = Dataset.load_from_df(
    pd.DataFrame(data, columns=['user', 'item',
                                'rating']),
    reader
)

# User-based CF
user_based = KNNBasic(
    k=40,
    sim_options={
        'name': 'cosine',
        'user_based': True
    }
)

# Item-based CF
item_based = KNNBasic(
    k=40,
    sim_options={
        'name': 'cosine',
        'user_based': False
    }
)

# Кросс-валидация
cv_results_user = cross_validate(user_based,
                                   dataset,
                                   measures=[
                                       'RMSE', 'MAE'],
                                   cv=5,
                                   verbose=True)

cv_results_item = cross_validate(item_based,
                                   dataset,
                                   measures=[
                                       'RMSE', 'MAE'],
                                   cv=5,
                                   verbose=True)

# Обучение и предсказание

```

```

trainset = dataset.build_full_trainset()
user_based.fit(trainset)

```

```

# Предсказание для пользователя и объекта
prediction = user_based.predict('user1', 'item3')
print(f"Predicted rating: {prediction.est:.2f}")

```

## ◆ 8. Сравнение User-based vs Item-based

| Аспект                  | User-based                       | Item-based                    |
|-------------------------|----------------------------------|-------------------------------|
| <b>Схожесть</b>         | Между пользователями             | Между объектами               |
| <b>Вычисления</b>       | Дороже (больше пользователей)    | Дешевле (меньше объектов)     |
| <b>Стабильность</b>     | Меняется часто                   | Более стабильна               |
| <b>Объяснимость</b>     | "Люди как вы купили..."          | "Похоже на то, что вы любите" |
| <b>Масштабируемость</b> | Хуже                             | Лучше                         |
| <b>Cold start</b>       | Проблема с новыми пользователями | Проблема с новыми объектами   |

## ◆ 9. Проблемы и решения

### 1. Cold Start (холодный старт)

- **Новый пользователь:** популярные объекты, опрос предпочтений
- **Новый объект:** гибридные методы, content-based подход

### 2. Разреженность данных (Sparsity)

- Матричная факторизация (SVD, NMF)
- Уменьшение размерности
- Использование неявных сигналов (просмотры, клики)

### 3. Масштабируемость

- Approximate Nearest Neighbors (ANN)
- Кластеризация пользователей/объектов
- Локально-чувствительное хеширование (LSH)

### 4. Разнообразие рекомендаций

- Добавление случайности
- Диверсификация топ-N списка
- Exploration-exploitation баланс

## ◆ 10. Оптимизации и улучшения

### 1. Нормализация рейтингов

```
# Центрирование по среднему пользователю
centered = ratings - ratings.mean(axis=1,
keepdims=True)

# Z-score нормализация
from scipy import stats
normalized = stats.zscore(ratings, axis=1,
nan_policy='omit')
```

### 2. Учет временного фактора

```
# Экспоненциальное затухание старых взаимодействий
import datetime as dt

def time_decay_weight(timestamp, current_time,
half_life=30):
    """half_life в днях"""
    days_diff = (current_time - timestamp).days
    return 0.5 ** (days_diff / half_life)
```

### 3. Негативные примеры

- Учет отрицательных рейтингов
- Implicit negative feedback (пропущенные объекты)

## ◆ 11. Метрики качества

### Метрики точности

- **RMSE:**  $\sqrt{(\sum(r - \hat{r})^2) / n}$
- **MAE:**  $\sum|r - \hat{r}| / n$
- **Precision@K:** доля релевантных в топ-K
- **Recall@K:** доля найденных релевантных
- **NDCG:** normalized discounted cumulative gain

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error

# RMSE
rmse = np.sqrt(mean_squared_error(true_ratings,
predicted_ratings))

# MAE
mae = mean_absolute_error(true_ratings,
predicted_ratings)

# Precision@K
def precision_at_k(recommended, relevant, k):
    recommended_k = recommended[:k]
    return len(set(recommended_k) & set(relevant)) / k

# Coverage (покрытие каталога)
def catalog_coverage(recommendations,
catalog_size):
    unique_recommended = set()
    for recs in recommendations:
        unique_recommended.update(recs)
    return len(unique_recommended) / catalog_size
```

## ◆ 12. Практические советы

- **Выбор k:** обычно 20-50 соседей, подбирать по валидации
- **Предвычисление:** схожести можно вычислить заранее
- **Порог схожести:** отсекать слабо похожих соседей
- **Бизнес-правила:** фильтрация недоступных объектов
- **A/B тестирование:** всегда проверять на реальных пользователях
- **Гибридные подходы:** комбинировать с content-based
- **Инкрементальное обновление:** не пересчитывать все с нуля
- **Контекст:** учитывать время, устройство, местоположение

 *Item-based CF часто выбор по умолчанию для production систем*



# Community Detection

17 Январь 2026

## ◆ 1. Суть

- **Задача:** найти плотно связанные группы узлов в графе
- **Сообщество:** группа узлов с большим числом внутренних связей
- **Применение:** социальные сети, биология, интернет
- **Сложность:** NP-полная задача в общем случае
- **Подходы:** модулярность, спектральные методы, label propagation

*Community detection - задача разбиения графа на кластеры узлов с плотными внутренними связями и разреженными внешними*

## ◆ 2. Модулярность (Modularity)

### Метрика качества разбиения:

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - k_i k_j / 2m] \delta(c_i, c_j)$$

где:

- $A_{ij}$  - матрица смежности
- $k_i$  - степень узла  $i$
- $m$  - число ребер
- $c_i$  - сообщество узла  $i$
- $\delta(c_i, c_j) = 1$  если  $c_i = c_j$

### Значения:

- $Q = 0$ : случайное разбиение
- $Q > 0.3$ : хорошая структура сообществ
- $Q > 0.7$ : очень сильная структура

## ◆ 3. Louvain алгоритм

```
import networkx as nx
from community import community_louvain

# Создание графа
G = nx.karate_club_graph()

# Обнаружение сообществ
communities = community_louvain.best_partition(G)

# Модулярность
modularity =
    community_louvain.modularity(communities, G)
print(f"Modularity: {modularity:.3f}")

# Визуализация
import matplotlib.pyplot as plt

pos = nx.spring_layout(G)
colors = [communities[node] for node in G.nodes()]

nx.draw(G, pos, node_color=colors,
        with_labels=True, cmap=plt.cm.rainbow)
plt.show()
```

## ◆ 4. Label Propagation

```
def label_propagation(G, max_iter=100):
    """
    Label Propagation алгоритм
    Быстрый и простой метод
    """
    import random

    # Инициализация уникальными метками
    labels = {node: node for node in G.nodes()}

    for iteration in range(max_iter):
        # Случайный порядок узлов
        nodes = list(G.nodes())
        random.shuffle(nodes)

        changed = False
        for node in nodes:
            # Подсчет меток соседей
            neighbor_labels = [labels[neighbor]
                               for neighbor in
                               G.neighbors(node)]

            if not neighbor_labels:
                continue

            # Выбор самой частой метки
            from collections import Counter
            most_common =
                Counter(neighbor_labels).most_common(1)[0][0]

            if labels[node] != most_common:
                labels[node] = most_common
                changed = True

        if not changed:
            break

    return labels

# Использование
communities = label_propagation(G)

# или через NetworkX
from networkx.algorithms import community
communities_nx =
    community.label_propagation_communities(G)
```

## ◆ 5. Girvan-Newman

### Делительный (divisive) подход:

```
from networkx.algorithms.community import
girvan_newman

# Последовательное удаление ребер с максимальной
betweenness
communities_generator = girvan_newman(G)

# Получить k сообществ
k = 3
communities = []
for _ in range(k - 1):
    communities = next(communities_generator)

# Результат
for i, community in enumerate(communities):
    print(f"Community {i}: {community}")
```

### Алгоритм:

1. Вычислить betweenness для всех ребер
2. Удалить ребро с максимальной betweenness
3. Пересчитать betweenness
4. Повторять до достижения k сообществ

## ◆ 6. Spectral Clustering

```
from sklearn.cluster import SpectralClustering
import numpy as np

# Матрица смежности
A = nx.to_numpy_array(G)

# Spectral clustering
sc = SpectralClustering(n_clusters=3,
affinity='precomputed')
labels = sc.fit_predict(A)

# Создание словаря сообществ
communities = {node: label for node, label in
enumerate(labels)}

# Вручную
def spectral_clustering_manual(A, k):
"""
Спектральная кластеризация
"""
from scipy.sparse.linalg import eigsh

# Степенная матрица
D = np.diag(A.sum(axis=1))

# Лапласиан
L = D - A

# Нормализованный лапласиан
D_inv_sqrt = np.diag(1.0 /
np.sqrt(D.diagonal()))
L_norm = D_inv_sqrt @ L @ D_inv_sqrt

# k наименьших собственных векторов
eigenvalues, eigenvectors = eigsh(L_norm, k=k,
which='SM')

# K-means на собственных векторах
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=k)
labels = kmeans.fit_predict(eigenvectors)

return labels
```

## ◆ 7. Infomap

```
# Использует random walks и теорию информации
# Минимизирует длину описания случайных блужданий

try:
    import infomap

    # Создание Infomap объекта
    im = infomap.Infomap("--two-level")

    # Добавление ребер
    for u, v in G.edges():
        im.add_link(u, v)

    # Запуск
    im.run()

    # Результаты
    communities = {}
    for node in im.tree:
        if node.is_leaf:
            communities[node.node_id] =
node.module_id

    print(f"Found {im.num_top_modules} communities")
    print(f"Codelength: {im.codelength}")

except ImportError:
    print("Install infomap: pip install infomap")
```

## ◆ 8. Сравнение алгоритмов

| Алгоритм      | Сложность           | Качество | Скорость      |
|---------------|---------------------|----------|---------------|
| Louvain       | O(n log n)          | ★★★★★    | Быстрый       |
| Label Prop.   | O(m + n)            | ★★★★     | Очень быстрый |
| Girvan-Newman | O(m <sup>2</sup> n) | ★★★★★    | Медленный     |
| Spectral      | O(n <sup>3</sup> )  | ★★★★★    | Средний       |
| Infomap       | O(n log n)          | ★★★★★★   | Быстрый       |

## ◆ 9. Метрики оценки

```
from sklearn.metrics import adjusted_rand_score,
normalized_mutual_info_score

def evaluate_communities(true_labels,
pred_labels):
    """
    Оценка качества обнаружения сообществ
    """
    # ARI (Adjusted Rand Index)
    ari = adjusted_rand_score(true_labels,
pred_labels)

    # NMI (Normalized Mutual Information)
    nmi =
normalized_mutual_info_score(true_labels,
pred_labels)

    print(f"ARI: {ari:.3f}")
    print(f"NMI: {nmi:.3f}")

    return ari, nmi

# Внутренние метрики
def internal_metrics(G, communities):
    """
    Метрики без ground truth
    """
    # Modularity
    mod = nx.algorithms.community.modularity(G,
communities)

    # Coverage
    cov = nx.algorithms.community.coverage(G,
communities)

    # Performance
    perf = nx.algorithms.community.performance(G,
communities)

    print(f"Modularity: {mod:.3f}")
    print(f"Coverage: {cov:.3f}")
    print(f"Performance: {perf:.3f})
```

## ◆ 10. Overlapping Communities

### Перекрывающиеся сообщества:

```
from networkx.algorithms.community import
k_clique_communities

# k-clique communities
# Узлы в одном сообществе если соединены k-кликой
communities = list(k_clique_communities(G, k=3))

for i, comm in enumerate(communities):
    print(f"Community {i}: {comm}")

# DEMON алгоритм (ego-network based)
def demon_algorithm(G, epsilon=0.25):
    """
    DEMON - обнаружение перекрывающихся сообществ
    """
    communities = []

    for node in G.nodes():
        # Ego-сеть узла
        ego_net = nx.ego_graph(G, node)

        # Label propagation на ego-сети
        labels = label_propagation(ego_net)

        # Извлечение сообществ
        for label in set(labels.values()):
            community = {n for n, l in
labels.items() if l == label}
            if len(community) >= 3:
                communities.append(community)

    # Слияние похожих сообществ
    merged =
merge_similar_communities(communities, epsilon)
    return merged
```

## ◆ 11. Иерархические сообщества

```
from scipy.cluster.hierarchy import dendrogram,
linkage
import matplotlib.pyplot as plt

def hierarchical_communities(G):
    """
    Иерархическая кластеризация для графов
    """
    # Матрица кратчайших путей
    path_length =
dict(nx.all_pairs_shortest_path_length(G))

    n = len(G.nodes())
    distance_matrix = np.zeros((n, n))

    nodes = list(G.nodes())
    for i, u in enumerate(nodes):
        for j, v in enumerate(nodes):
            if u in path_length and v in
path_length[u]:
                distance_matrix[i][j] =
path_length[u][v]
            else:
                distance_matrix[i][j] = n #
Disconnected

    # Hierarchical clustering
    Z = linkage(distance_matrix, method='ward')

    # Dendrogram
    plt.figure(figsize=(10, 5))
    dendrogram(Z, labels=nodes)
    plt.title('Hierarchical Community Structure')
    plt.show()

    return Z
```

## ◆ 12. Лучшие практики

- **Выбор алгоритма:** Louvain для больших графов
- **Валидация:** используйте несколько метрик
- **Параметры:** resolution для Louvain
- **Визуализация:** проверяйте результаты глазами
- **Масштабируемость:** Label Prop. для огромных графов
- **Качество:** Infomap для максимального качества

### ✓ Когда использовать

- Социальные сети
- Биологические сети
- Анализ веб-страниц
- Рекомендательные системы

### ✗ Проблемы

- Resolution limit
- Нестабильность Label Prop.
- Вычислительная сложность

# ⚡ Нейросети vs Классические методы

 Январь 2026

## ◆ 1. Основные различия

- **Нейросети:** автоматическое извлечение признаков из сырых данных
- **Классические методы:** требуют ручного feature engineering
- **Данные:** нейросети нужно много данных, классические работают с меньшим количеством
- **Интерпретируемость:** классические методы более прозрачны
- **Производительность:** нейросети превосходят на сложных задачах (изображения, текст)

*Выбор между нейросетями и классическими методами зависит от задачи, объёма данных и требований к интерпретируемости.*

## ◆ 2. Установка

```
# Установка через pip
pip install fastai

# Или через conda
conda install -c fastai -c pytorch fastai

# Проверка
python -c "import fastai;
print(fastai.__version__)"

# GPU support (CUDA)
pip install fastai torch torchvision --extra-index-url https://download.pytorch.org/whl/cu118
```

## ◆ 3. Computer Vision

### Классификация изображений:

```
from fastai.vision.all import *

# Загрузка данных
path = untar_data(URLs.PETS) / 'images'

# DataLoader
dls = ImageDataLoaders.from_name_re(
    path,
    get_image_files(path),
    pat=r'(.+)\.jpg$',
    item_tfms=Resize(460),
    batch_tfms=aug_transforms(size=224)
)

# Создание модели
learn = vision_learner(
    dls,
    resnet34,
    metrics=error_rate
)

# Обучение
learn.fine_tune(3)
```

### Доступные архитектуры:

- ResNet (18, 34, 50, 101, 152)
- EfficientNet (b0-b7)
- DenseNet, VGG, SqueezeNet
- Vision Transformer (ViT)

## ◆ 4. Transfer Learning

- **Discriminative learning rates:** разные LR для слоёв
- **Gradual unfreezing:** постепенное размораживание
- **Fine-tuning:** автоматическая настройка

```
# Freeze backbone
learn.freeze()
learn.fit_one_cycle(2)

# Unfreeze и train с разными LR
learn.unfreeze()
learn.fit_one_cycle(5, lr_max=slice(1e-6, 1e-4))

# Learning rate finder
learn.lr_find()
```

| Метод           | Описание                  |
|-----------------|---------------------------|
| fine_tune()     | Freeze → Train → Unfreeze |
| fit_one_cycle() | One-cycle policy обучение |
| lr_find()       | Поиск оптимального LR     |

## ◆ 5. NLP с Нейросети vs Классические методы

### Text Classification:

```
from fastai.text.all import *

# Загрузка данных
path = untar_data(URLs.IMDB)

# DataLoader
dls = TextDataLoaders.from_folder(
    path,
    valid='test',
    text_vocab=None
)

# Модель
learn = text_classifier_learner(
    dls,
    AWD_LSTM,
    drop_mult=0.5,
    metrics=accuracy
)

# Fine-tuning
learn.fine_tune(4, 1e-2)
```

### Language Model:

```
# Обучение language model
learn_lm = language_model_learner(
    dls_lm,
    AWD_LSTM,
    drop_mult=0.3,
    metrics=[accuracy, Perplexity()]
)

learn_lm.fit_one_cycle(1, 2e-2)
```

## ◆ 6. Tabular Data

```
from fastai.tabular.all import *

# Подготовка данных
path = untar_data(URLs.ADULT_SAMPLE)
df = pd.read_csv(path/'adult.csv')

# DataLoader
dls = TabularDataLoaders.from_csv(
    path='adult.csv',
    path=path,
    y_names="salary",
    cat_names=['workclass', 'education', 'marital-status'],
    cont_names=['age', 'fnlwgt', 'education-num'],
    procs=[Categorify, FillMissing, Normalize]
)

# Модель
learn = tabular_learner(
    dls,
    metrics=accuracy
)

learn.fit_one_cycle(3)
```

## ◆ 8. Learning Rate Policies

### 1cycle Policy:

- Warm-up: постепенное увеличение LR
- Annealing: постепенное уменьшение
- Momentum: обратно пропорционально LR

```
# 1cycle training
learn.fit_one_cycle(
    n_epoch=10,
    lr_max=1e-3,
    moms=(0.95, 0.85), # momentum range
    div=25.0,           # initial LR division
    pct_start=0.3       # warmup percentage
)

# Flat + cosine annealing
learn.fit_flat_cos(
    n_epoch=10,
    lr=1e-3,
    pct_start=0.75
)
```

## ◆ 7. Data Augmentation

### Встроенные трансформации:

```
# Image augmentation
aug_transforms(
    mult=1.0,           # multiplier
    do_flip=True,        # horizontal flip
    flip_vert=False,     # vertical flip
    max_rotate=10.0,      # rotation degrees
    max_zoom=1.1,         # zoom factor
    max_lighting=0.2,      # lighting change
    max_warp=0.2,          # perspective warp
    p_affine=0.75,        # probability
    p_lighting=0.75
)

# Custom transforms
item_tfms = [Resize(460)]
batch_tfms = [
    *aug_transforms(size=224),
    Normalize.from_stats(*imagenet_stats)
]
```

## ◆ 9. Callbacks

| Callback              | Назначение             |
|-----------------------|------------------------|
| EarlyStoppingCallback | Остановка при plateau  |
| SaveModelCallback     | Сохранение best модели |
| CSVLogger             | Логирование в CSV      |
| ReduceLROnPlateau     | Уменьшение LR          |
| MixedPrecision        | FP16 training          |
| GradientClip          | Gradient clipping      |

```
# Использование callbacks
learn.fit_one_cycle(
    10,
    cbs=[
        EarlyStoppingCallback(patience=3),
        SaveModelCallback(),
        CSVLogger()
    ]
)
```

## ◆ 10. Inference и Export

```
# Inference на одном примере
img = PILImage.create('test.jpg')
pred_class, pred_idx, probs = learn.predict(img)

# Batch inference
dl = learn.dls.test_dl(test_items)
preds, _ = learn.get_preds(dl=dl)

# Export модели
learn.export('model.pkl')

# Load для inference
learn_inf = load_learner('model.pkl')
pred = learn_inf.predict(img)
```

## ◆ 11. Interpretation

```
# Classification interpretation
interp =
ClassificationInterpretation.from_learner(learn)

# Confusion matrix
interp.plot_confusion_matrix(figsize=(12,12))

# Top losses
interp.plot_top_losses(9, figsize=(15,11))

# Most confused
interp.most_confused(min_val=2)
```

## ◆ 12. Best Practices

### ✓ Рекомендации

- ✓ Использовать lr\_find()
- ✓ Применять fit\_one\_cycle()
- ✓ Начинать с small models
- ✓ Использовать mixed precision
- ✓ Progressive resizing
- ✓ Test time augmentation

### ✗ Избегать

- ✗ Случайный выбор LR
- ✗ Игнорирование validation
- ✗ Переобучение с первого раза
- ✗ Слишком большой batch size
- ✗ Отсутствие augmentation



# Computational Chemistry и ML

Январь 2026

## 1. ML в вычислительной химии: обзор

**Революция:** ML ускоряет химические расчеты в тысячи раз

- **Традиционные методы:** DFT, молекулярная динамика (часы-дни)
- **ML подход:** предсказание за миллисекунды
- **Применения:** drug discovery, материалы, катализ

## 2. Представление молекул

**Способы кодирования:**

- **SMILES:** текстовое представление (CC(=O)O для уксусной кислоты)
- **Molecular graphs:** атомы = узлы, связи = рёбра
- **3D coordinates:** xyz координаты атомов
- **Fingerprints:** Morgan, MACCS, бинарные векторы
- **Descriptors:** молекулярный вес, logP, TPSA

## 3. Graph Neural Networks для молекул

**Архитектуры:**

- **MPNN** (Message Passing NN): информация между атомами
- **SchNet**: continuous-filter convolutions
- **DimeNet**: directional message passing
- **GemNet**: geometric message passing

```
# Пример MPNN
for layer in range(num_layers):
    # Message passing
    messages =
    aggregate_neighbors(node_features,
    edges)
    # Update
    node_features =
    update_function(node_features, messages)
```

## 4. Предсказание свойств молекул

**Задачи:**

- **Solubility:** растворимость в воде
- **Lipophilicity:** logP, проницаемость мембран
- **Toxicity:** LD50, AMES test
- **Binding affinity:** связывание с белком
- **ADMET:** Absorption, Distribution, Metabolism, Excretion, Toxicity

## 5. Квантовая химия и ML

**Предсказание энергий:**

- **Potential Energy Surfaces (PES)**
- **Atomization energies**
- **Molecular forces**

**Модели:**

- **ANI** (Accurate Neural Network Engine for Molecular Energies)
- **SchNet**: rotation-equivariant
- **PhysNet**: physical constraints

## 6. Drug Discovery с ML

**Pipeline:**

- **Virtual screening:** фильтрация больших библиотек
- **De novo design:** генерация новых молекул
- **Lead optimization:** улучшение кандидатов
- **Retrosynthesis:** планирование синтеза

**Успехи:**

- DSI-poised-Fragment (AstraZeneca)
- Antibiotics discovery (MIT, 2020)

## ◆ 7. Генеративные модели молекул

**Подходы:**

- **VAE:** Junction Tree VAE, Grammar VAE
- **GAN:** MolGAN
- **Reinforcement Learning:** REINVENT
- **Flow models:** MoFlow
- **Diffusion:** E(n) Equivariant Diffusion

## ◆ 8. AlphaFold и структура белков

**AlphaFold 2 (DeepMind, 2020):**

- Предсказание 3D структуры белка из последовательности
- Accuracy ~90% (atomic level)
- Революция в структурной биологии

**Архитектура:**

- Evoformer: attention + pair representation
- Structure module: итеративное уточнение

## ◆ 9. Reaction Prediction

**Задача:** предсказать продукты химической реакции

- **Template-based:** использование известных шаблонов
- **Template-free:** seq2seq модели
- **Graph-based:** изменения в молекулярном графе

**Модели:**

- Molecular Transformer
- Graph2Graph
- LocalRetro

## ◆ 10. Молекулярная динамика

**ML-accelerated MD:**

- **Force fields:** обученные на квантовых расчетах
- **Coarse-graining:** упрощение системы
- **Enhanced sampling:** редкие события

## ◆ 11. Датасеты и бенчмарки

| Dataset | Размер | Задача                |
|---------|--------|-----------------------|
| QM9     | 134K   | Квантовые свойства    |
| ZINC    | 250M+  | Drug-like молекулы    |
| PubChem | 100M+  | Химические соединения |
| ChEMBL  | 2M+    | Bioactivity data      |

## ◆ 12. Инструменты и библиотеки

- **RDKit:** хемоинформатика, fingerprints
- **DeepChem:** ML для химии
- **PyTorch Geometric:** GNN для молекул
- **SchNetPack:** neural networks для химии
- **OpenMM:** молекулярная динамика

## ◆ 13. Вызовы

- **Data scarcity:** мало экспериментальных данных
- **Out-of-distribution:** обобщение на новые химические пространства
- **Interpretability:** понимание предсказаний
- **Физическая правдоподобность:** соблюдение законов физики

## ◆ 14. Будущее

- **Autonomous labs:** самоуправляемые эксперименты
- **Foundation models:** универсальные модели для химии
- **Quantum + ML:** гибридные подходы
- **Materials design:** новые материалы с нужными свойствами

«ML трансформирует химию из экспериментальной науки в предиктивную, ускоряя открытие новых лекарств и материалов в десятки раз».

# Computational Social Science

 Январь 2026

## ◆ 1. Введение в CSS

**Computational Social Science** — применение вычислительных методов для исследования социальных феноменов

- **Большие данные:** социальные сети, мобильные данные, цифровые следы
- **ML методы:** обработка естественного языка, сетевой анализ, прогнозирование
- **Цели:** понимание социального поведения, прогнозирование трендов, разработка политик
- **Междисциплинарность:** социология + информатика + статистика

## ◆ 2. Источники данных

**Типы социальных данных:**

- **Социальные сети:** Twitter, Facebook, LinkedIn, Instagram
- **Мобильные данные:** GPS треки, звонки, SMS
- **Опросы:** традиционные и онлайн-опросы
- **Административные данные:** переписи, налоговые данные, образовательные записи
- **Web scraping:** форумы, новостные сайты, блоги
- **Сенсорные данные:** носимые устройства, умные дома

*Важно: соблюдение этических норм и приватности при работе с персональными данными*

## ◆ 3. Анализ социальных сетей

**Network Science** подходы:

- **Центральность:** degree, betweenness, closeness, eigenvector
- **Community detection:** Louvain, Girvan-Newman, Label Propagation
- **Influence analysis:** PageRank, HITS, k-shell decomposition
- **Temporal networks:** эволюция связей, динамические графы
- **Multilayer networks:** несколько типов связей одновременно

```
# NetworkX пример
import networkx as nx
G = nx.Graph()
G.add_edges_from(edges)
centrality =
nx.betweenness_centrality(G)
communities =
nx.community.louvain_communities(G)
```

## ◆ 4. Анализ текстов и мнений

NLP для социальных данных:

- **Sentiment analysis:** определение тональности текстов (позитив/негатив)
- **Topic modeling:** LDA, NMF для выявления скрытых тем
- **Named Entity Recognition:** извлечение имён, организаций, локаций
- **Hate speech detection:** выявление токсичных комментариев
- **Stance detection:** определение позиции автора по вопросу
- **Fact-checking:** автоматическая проверка фактов

## ◆ 5. Прогнозирование социальных явлений

Предсказательные модели:

- **Распространение информации:** модели диффузии, каскады ретвитов
- **Выборы и референдумы:** прогнозирование результатов голосования
- **Социальные движения:** предсказание протестов и активности
- **Экономические индикаторы:** безработица, потребительская активность
- **Здоровье:** распространение эпидемий на основе мобильности
- **Преступность:** предсказание криминальных инцидентов

## ◆ 6. Обработка временных рядов

Анализ социальных трендов:

- **Time series forecasting:** ARIMA, Prophet, LSTM для социальных показателей
- **Anomaly detection:** выявление необычных паттернов активности
- **Event detection:** обнаружение важных событий в потоке данных
- **Seasonal patterns:** выявление сезонности в социальном поведении
- **Causality analysis:** Granger causality для связей между явлениями

```
# Prophet для социальных трендов
from prophet import Prophet
df = pd.DataFrame({'ds': dates, 'y': values})
model = Prophet()
model.fit(df)
forecast = model.predict(future)
```

## ◆ 7. Мобильность и пространственный анализ

Геопространственные данные:

- **Траектории движения:** анализ GPS данных, паттерны мобильности
- **Point of Interest (POI):** популярные места, категоризация локаций
- **Flow analysis:** миграционные потоки, маятниковая миграция
- **Urban segregation:** изучение пространственного неравенства
- **Accessibility:** доступность услуг и инфраструктуры
- **Activity inference:** определение активности по GPS (дом, работа, шоппинг)

## ◆ 8. Компьютерное зрение для социальных исследований

Image & Video Analysis:

- **Face recognition:** анализ демографии в публичных пространствах
- **Scene understanding:** классификация городских пространств
- **Crowd analysis:** подсчёт людей, анализ плотности толпы
- **Emotion detection:** распознавание эмоций по фото
- **Fashion & style:** анализ трендов в одежде
- **Protest analysis:** автоматическое выявление протестов на фото

## ◆ 9. Моделирование агентов (АВМ)

### Agent-Based Modeling:

- **Индивидуальные агенты:** моделирование поведения отдельных людей
- **Взаимодействия:** правила взаимодействия между агентами
- **Emergent behavior:** коллективные эффекты из индивидуальных действий
- **Mesa framework:** Python библиотека для АВМ
- **NetLogo:** популярная платформа для агентного моделирования
- **Calibration:** подгонка параметров модели к реальным данным

## ◆ 10. Культурная аналитика

### Digital Humanities методы:

- **Cultural trends:** анализ изменений культурных предпочтений
- **Meme analysis:** распространение интернет-мемов
- **Music & art trends:** эволюция музыкальных и художественных стилей
- **Language evolution:** изменения в языке со временем
- **Historical text analysis:** применение NLP к историческим документам
- **Cross-cultural comparison:** сравнение культур через данные

## ◆ 11. Этика и вызовы

### Этические соображения:

- **Privacy:** защита персональных данных, анонимизация
- **Informed consent:** согласие участников исследования
- **Bias & fairness:** избежание дискриминации в моделях
- **Transparency:** объяснимость решений алгоритмов
- **Dual use:** потенциальное негативное использование результатов
- **Data quality:** репрезентативность и погрешности в данных

*Критически важно: соблюдение IRB (Institutional Review Board) требований*

## ◆ 13. Инструменты и библиотеки

### Python экосистема:

| Библиотека   | Назначение                    |
|--------------|-------------------------------|
| NetworkX     | Анализ графов и сетей         |
| igraph       | Быстрый сетевой анализ        |
| NLTK, spaCy  | Обработка естественного языка |
| scikit-learn | Машинное обучение             |
| pandas       | Обработка табличных данных    |
| geopandas    | Геопространственный анализ    |
| Mesa         | Agent-based modeling          |
| Tweepy       | Twitter API доступ            |

## ◆ 12. Методы машинного обучения

### Популярные ML подходы:

- **Supervised learning:** классификация социальных групп, регрессия
- **Unsupervised learning:** кластеризация сообществ, выявление паттернов
- **Deep learning:** CNN для изображений, RNN/Transformers для текстов
- **Graph neural networks:** анализ социальных сетей
- **Transfer learning:** использование предобученных моделей
- **Ensemble methods:** комбинирование моделей для лучшей точности

## ◆ 14. Исследовательские направления

### Актуальные темы 2024-2026:

- **Misinformation:** борьба с дезинфекцией и фейковыми новостями
- **Polarization:** изучение политической и социальной поляризации
- **COVID-19 impact:** анализ социальных последствий пандемии
- **Climate change:** общественное мнение и поведение по климату
- **Social bots:** выявление и изучение ботов в соцсетях
- **Gig economy:** изучение новых форм занятости
- **Digital inequality:** цифровой разрыв и неравенство доступа
- **Mental health:** онлайн-индикаторы психологического состояния

## ◆ 15. Валидация и оценка

### Проверка качества исследований:

- **External validity:** обобщаемость результатов на другие популяции
- **Robustness checks:** проверка устойчивости к вариациям
- **Ground truth comparison:** сравнение с реальными данными
- **Cross-validation:** предотвращение переобучения моделей
- **Sensitivity analysis:** влияние параметров на результаты
- **Reproducibility:** воспроизводимость исследований

## ◆ 16. Кейс-стадии и применения

### Реальные примеры:

- **Arab Spring:** роль Twitter в революциях 2011 года
- **Brexit prediction:** прогнозирование результатов референдума
- **Flu tracking:** Google Flu Trends, мониторинг эпидемий
- **Urban planning:** оптимизация транспорта через мобильные данные
- **Job market:** анализ LinkedIn для понимания рынка труда
- **Disaster response:** Twitter для координации при катастрофах



# Concept Activation Vectors

 4 января 2026

## ◆ 1. Суть CAV

- Проблема:** нейросети оперируют низкоуровневыми признаками
- Решение:** связать внутренние представления с человеко-понятными концепциями
- CAV (Concept Activation Vector):** направление в пространстве активаций, соответствующее концепции
- TCAV:** Testing with Concept Activation Vectors - метод количественной оценки
- Цель:** понять, использует ли модель определенные концепции для предсказаний

## ◆ 2. Как работает TCAV

- Выбор концепции:** например, "полосатый", "пушистый"
- Сбор примеров:** позитивные (с концепцией) и негативные (без)
- Извлечение активаций:** получить активации слоя для примеров
- Обучение классификатора:** линейный классификатор разделяет концепцию
- CAV = нормаль к гиперплоскости:** направление концепции
- Вычисление TCAV score:** чувствительность предсказаний к концепции

## ◆ 3. Математика TCAV

### CAV вычисление:

Обучить линейный классификатор:  $w \cdot h + b = 0$

$CAV = w / \|w\|$  (нормализованный вектор весов)

### Directional derivative:

$$S_{C,k,l}(x) = \nabla_{h_l} f_k(x) \cdot v_C^l$$

где:

- $f_k$  - предсказание класса  $k$
- $h_l$  - активации слоя  $l$
- $v_C^l$  - CAV для концепции  $C$  в слое  $l$

### TCAV score:

$$TCAV_{C,k,l} = |\{x \in X_k : S_{C,k,l}(x) > 0\}| / |X_k|$$

## ◆ 4. Базовый код

```
import torch
import torch.nn as nn
from sklearn.linear_model import LogisticRegression

# 1. Извлечение активаций
def get_activations(model, images, layer_name):
    activations = []
    def hook(module, input, output):
        activations.append(output.detach())
    handle = getattribute(model,
    layer_name).register_forward_hook(hook)
    model(images)
    handle.remove()
    return activations[0]

# 2. Обучение CAV
conceptActs = get_activations(model,
concept_imgs, 'layer4')
randomActs = get_activations(model, random_imgs,
'layer4')

x = torch.cat([conceptActs,
randomActs]).cpu().numpy()
y = [1]*len(conceptActs) + [0]*len(randomActs)

clf = LogisticRegression().fit(x, y)
cav = clf.coef_[0] / np.linalg.norm(clf.coef_[0])
```

## ◆ 5. Вычисление TCAV score

```
def compute_tcav_score(model, test_images, cav,
layer_name):
    tcav_scores = []

    for img in test_images:
        img.requires_grad = True

        # Forward pass
        activations = get_activations(model,
img.unsqueeze(0), layer_name)
        output = model(img.unsqueeze(0))

        # Gradient
        output[0, target_class].backward()
        grad = activations.grad

        # Directional derivative
        directional_deriv = torch.sum(grad *
torch.tensor(cav))

        tcav_scores.append(directional_deriv.item() > 0)

    return sum(tcav_scores) / len(tcav_scores)
```

## ◆ 6. Полный пример TCAV

```
import numpy as np
from sklearn.linear_model import SGDClassifier

class TCAVInterpreter:
    def __init__(self, model, layer_names):
        self.model = model
        self.layer_names = layer_names
        self.cavs = {}

    def train_cav(self, concept_images,
random_images, layer):
        # Получить активации
        concept_acts =
self.getActs(concept_images, layer)
        random_acts = self.getActs(random_images,
layer)

        # Обучить классификатор
        X = np.vstack([concept_acts, random_acts])
        y = np.array([1]*len(concept_acts) +
[0]*len(random_acts))

        clf = SGDClassifier(loss='log',
max_iter=1000)
        clf.fit(X, y)

        # CAV = нормализованные веса
        cav = clf.coef_[0] /
np.linalg.norm(clf.coef_[0])
        self.cavs[(concept, layer)] = cav

        return cav

    def compute_tcav(self, test_images, concept,
target_class, layer):
        cav = self.cavs[(concept, layer)]
        scores = []

        for img in test_images:
            score = self.directional_derivative(
                img, cav, target_class, layer
            )
            scores.append(score > 0)

        return np.mean(scores)
```

## ◆ 7. Выбор концепций

### Хорошие концепции:

- Визуально различимые (полосы, текстуры, цвета)
- Семантически значимые для задачи
- Достаточно примеров (20-50 изображений)

### Примеры концепций:

| Задача                  | Концепции                      |
|-------------------------|--------------------------------|
| Медицинская диагностика | Текстура, плотность, симметрия |
| Классификация животных  | Полосы, пятна, мех, перья      |
| Распознавание объектов  | Углы, края, формы              |
| Анализ настроения       | Улыбка, слезы, гнев            |

## ◆ 8. Статистическая значимость

```
from scipy import stats

def tcav_statistical_test(model, concept,
n_runs=10):
    tcav_scores = []

    for _ in range(n_runs):
        # Случайные негативные примеры каждый раз
        random_imgs = sample_random_images()

        # Обучить CAV
        cav = train_cav(concept_imgs, random_imgs)

        # Вычислить TCAV
        score = compute_tcav_score(test_imgs, cav)
        tcav_scores.append(score)

    # t-test: отличается ли от 0.5 (случайность)?
    t_stat, p_value =
stats.ttest_1samp(tcav_scores, 0.5)

    mean_score = np.mean(tcav_scores)
    return mean_score, p_value < 0.05 # значимо?
```

## ◆ 9. Визуализация результатов

```
import matplotlib.pyplot as plt

def visualize_tcav_results(concepts, tcav_scores,
layers):
    fig, ax = plt.subplots(figsize=(10, 6))

    x = np.arange(len(concepts))
    width = 0.2

    for i, layer in enumerate(layers):
        scores = [tcav_scores[c][layer] for c in
concepts]
        ax.bar(x + i*width, scores, width,
label=layer)

    ax.set_ylabel('TCAV Score')
    ax.set_xlabel('Concept')
    ax.set_title('Concept Importance by Layer')
    ax.set_xticks(x + width)
    ax.set_xticklabels(concepts)
    ax.legend()
    ax.axhline(0.5, color='red', linestyle='--',
label='Random')

    plt.tight_layout()
    plt.show()
```

## ◆ 11. Библиотека tcav

```
# Установка
pip install tcav

# Использование
from tcav import tcav

# Подготовка
bottlenecks = ['conv4', 'conv5'] # слои
concepts = ['striped', 'furry']
target_class = 'zebra'

# Запуск TCAV
tcav_results = tcav.get_tcav_scores(
    model,
    target_class,
    concepts,
    bottlenecks,
    activation_dir='./activations',
    cav_dir='./cavs',
    num_random_exp=10
)

# Результаты
for concept in concepts:
    for layer in bottlenecks:
        score = tcav_results[concept][layer]
        print(f'{concept} @ {layer}: {score:.3f}")
```

## ◆ 10. Интерпретация TCAV scores

- **TCAV ≈ 0.5:** концепция не влияет на предсказания
- **TCAV > 0.5:** концепция положительно влияет
- **TCAV < 0.5:** концепция отрицательно влияет
- **TCAV близко к 1:** очень важная концепция

### По слоям:

- Ранние слои: простые концепции (края, цвета)
- Средние слои: текстуры, паттерны
- Поздние слои: сложные семантические концепции

## ◆ 12. Практические советы

- **Число примеров:** 20-50 для концепции, 100-500 случайных
- **Качество CAV:** проверьте точность классификатора (должна быть >0.8)
- **Слои:** тестируйте несколько слоев, не только последний
- **Случайные концепции:** используйте для baseline
- **Статистика:** запускайте несколько раз с разными random
- **Нормализация:** активации должны быть нормализованы

## ◆ 13. Преимущества и ограничения

### ✓ Преимущества

- ✓ Человеко-понятные концепции
- ✓ Количественная оценка важности
- ✓ Работает без переобучения модели
- ✓ Применимо к разным архитектурам
- ✓ Помогает найти bias

### ✗ Ограничения

- ✗ Требует подготовки примеров концепций
- ✗ Вычислительно затратно
- ✗ Ограничено линейной разделимостью
- ✗ Субъективный выбор концепций
- ✗ Не подходит для очень глубоких слоев

## ◆ 14. Применения TCAV

| Область   | Применение                                                  |
|-----------|-------------------------------------------------------------|
| Медицина  | Проверка, использует ли модель клинически значимые признаки |
| Fairness  | Обнаружение использования защищенных атрибутов (раса, пол)  |
| Debugging | Понимание ошибок модели через концепции                     |
| Доверие   | Объяснение решений экспертам                                |

## ◆ 15. Расширения TCAV

- ACE: Automatic Concept Extraction** - автоматический поиск концепций
- Multi-modal TCAV**: для мультимодальных моделей
- Adversarial TCAV**: устойчивость к adversarial attacks
- Compositional TCAV**: комбинации концепций

## ◆ 16. Чек-лист использования

- [ ] Определить интересующие концепции
- [ ] Собрать примеры концепций (20-50 изображений)
- [ ] Собрать случайные примеры (100-500 изображений)
- [ ] Выбрать слои для анализа
- [ ] Обучить CAV для каждой концепции и слоя
- [ ] Проверить качество CAV (accuracy классификатора)
- [ ] Вычислить TCAV scores
- [ ] Провести статистическое тестирование
- [ ] Визуализировать и интерпретировать результаты

### 💡 Объяснение заказчику:

«TCAV позволяет проверять, использует ли нейросеть понятные нам концепции для принятия решений. Например, при распознавании зебр мы можем проверить, действительно ли модель обращает внимание на полосы, а не на фон или другие случайные признаки. Это как спросить: "Ты смотришь на то, на что нужно?"».



# Concept Drift Detection

\*July\* 17 Январь 2026

## 1. Что такое Drift?

- **Data Drift:** изменение распределения X
- **Concept Drift:** изменение  $P(Y|X)$
- **Label Drift:** изменение Y

## 2. Детекция дрейфа

```
from scipy.stats import ks_2samp

# Kolmogorov-Smirnov test
train_data = [...]
production_data = [...]

stat, p_value = ks_2samp(train_data, production_data)
if p_value < 0.05:
```

## 3. Population Stability Index

```
import numpy as np

def calculate_psi(expected, actual, bins=10):
    breakpoints = np.quantile(expected,
                              np.linspace(0, 1, bins+1))

    expected_percents = np.histogram(expected, breakpoints)[0] / len(expected)
    actual_percents = np.histogram(actual, breakpoints)[0] / len(actual)

    psi = np.sum((actual_percents - expected_percents) *
                 np.log(actual_percents / expected_percents))

    return psi

psi = calculate_psi(train_data, prod_data)
# PSI < 0.1: no drift
# PSI 0.1-0.2: moderate
# PSI > 0.2: significant drift
```

## 4. ADWIN

```
from river import drift

detector = drift.ADWIN()

for i, value in enumerate(stream):
    detector.update(value)
    if detector.drift_detected:
        print(f"Drift at index {i}")
```

## 5. DDM

```
from river.drift import DDM

ddm = DDM()

for pred, true in zip(predictions, true_labels):
    error = int(pred != true)
    ddm.update(error)

    if ddm.drift_detected:
```

## 6. Мониторинг метрик

```
# Отслеживание accuracy/F1 во времени
import pandas as pd

metrics_log = []
for batch in data_stream:
    predictions = model.predict(batch)
    accuracy = compute_accuracy(predictions, batch.labels)

    metrics_log.append({
        'timestamp': batch.timestamp,
        'accuracy': accuracy
    })

df = pd.DataFrame(metrics_log)
```

## 7. Стратегии реакции

- **Periodic Retraining**: регулярное переобучение
- **Triggered Retraining**: при детекции drift
- **Incremental Learning**: онлайн-обучение
- **Ensemble**: несколько моделей разных периодов

## 8. Методы детекции

| Метод      | Применение              |
|------------|-------------------------|
| KS-test    | Распределения признаков |
| PSI        | Сравнение распределений |
| ADWIN      | Потоковые данные        |
| DDM        | Ошибки классификации    |
| Monitoring | Метрики модели          |

## 9. Инструменты

```
# Evidently AI
from evidently.test_suite import TestSuite
from evidently.test_preset import DataDriftTestPreset

test_suite = TestSuite(tests=[DataDriftTestPreset()])
test_suite.run(reference_data=train,
               current_data=prod)
test_suite
```

## 10. Чек-лист

- Мониторить распределения признаков
- Отслеживать метрики модели
- Настроить алерты на drift
- Иметь план переобучения
- Логировать предсказания

## 11. Обнаружение Drift с помощью KS-test

```
from scipy.stats import ks_2samp
import pandas as pd

def detect_drift_ks_test(reference_data, current_data, features, threshold=0.05):
    """Обнаружение дрейфа с помощью Kolmogorov-Smirnov теста"""
    drift_detected = {}

    for feature in features:
        # KS-test для каждого признака
        statistic, p_value = ks_2samp(
            reference_data[feature],
            current_data[feature]
        )

        drift_detected[feature] = {
            'p_value': p_value,
            'drift': p_value < threshold
        }

        if p_value < threshold:
            print(f"⚠️ Drift detected in {feature}: p={p_value:.4f}")

    return drift_detected

# Использование
drift_results = detect_drift_ks_test(X_train_df, X_new_df,
                                      X_train_df.columns,
                                      threshold=0.01)
```

## 12. Мониторинг в Production

```
# Continuous monitoring setup
class DriftMonitor:
    def __init__(self, reference_data, window_size=1000):
        self.reference_data = reference_data
        self.window_size = window_size
        self.current_window = []
        self.alerts = []
```

```

def update(self, new_sample):
    self.current_window.append(new_sample)

    if len(self.current_window) >= self.window_size:
        # Проверка на drift
        current_df = pd.DataFrame(self.current_window)
        drift = self.check_drift(current_df)

        if drift:
            self.alerts.append({
                'timestamp': datetime.now(),
                'type': 'data_drift'
            })

    # Скользящее окно
    self.current_window = self.current_window[-self.window_size//2:]

def check_drift(self, current_data):
    # Проверка KS-test для каждого признака
    for col in self.reference_data.columns:
        _, p_value = ks_2samp(self.reference_data[col],
                              current_data[col])
        if p_value < 0.01:
            return True
    return False

# Production usage
monitor = DriftMonitor(X_train)
for sample in incoming_stream:
    monitor.update(sample)
    if monitor.alerts:
        # Trigger retraining or alert
        trigger_retraining_pipeline()

```

## 13. Стратегии реагирования на Drift

- **Переобучение модели:** на новых данных
- **Адаптация модели:** инкрементное обучение
- **Ансамбль моделей:** комбинация старых и новых
- **Feature engineering:** добавление новых признаков
- **Переключение моделей:** на pre-trained альтернативу
- **Мониторинг и алERTы:** уведомление команды

## 14. Чек-лист мониторинга Drift

1.  Установить baseline метрики на training data
2.  Настроить мониторинг входных признаков
3.  Мониторить предсказания модели
4.  Отслеживать performance метрики
5.  Использовать статистические тесты (KS, Chi-square)
6.  Визуализировать распределения со временем
7.  Настроить автоматические alerts
8.  Иметь план реагирования на drift
9.  Регулярно переобучать модели
10.  Версионировать данные и модели



# Матрица ошибок (Confusion Matrix)

Январь 2026

## ◆ 1. Суть

- **Цель:** детальный анализ ошибок классификации
- **Показывает:** какие классы путает модель
- **Применение:** бинарная и мультиклассовая классификация
- **Основа:** для вычисления всех метрик

*Confusion Matrix — это таблица, показывающая, как часто модель правильно и неправильно классифицирует объекты.*

## ◆ 2. Структура (бинарная)

|                  | Predicted:<br>Positive | Predicted:<br>Negative |
|------------------|------------------------|------------------------|
| Actual: Positive | TP (True Positive)     | FN (False Negative)    |
| Actual: Negative | FP (False Positive)    | TN (True Negative)     |

### Расшифровка:

- **TP:** правильно предсказали положительный класс
- **TN:** правильно предсказали отрицательный класс
- **FP:** ошибочно предсказали положительный (Type I error)
- **FN:** ошибочно предсказали отрицательный (Type II error)

## ◆ 3. Базовый код

```
from sklearn.metrics import confusion_matrix
import numpy as np

# Реальные и предсказанные значения
y_true = [0, 1, 0, 1, 0, 1, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1, 1, 1]

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print(cm)
# [[3 1]
#  [1 3]]

# Извлечение компонентов
tp, fp, fn, tn = cm.ravel()
print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")
```

## ◆ 4. Визуализация

```
import matplotlib.pyplot as plt
import seaborn as sns

# Создание confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Визуализация
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()

# С процентами
cm_normalized = cm.astype('float') / cm.sum(axis=1)
[:, np.newaxis]
sns.heatmap(cm_normalized, annot=True, fmt='.%2f',
            cmap='Blues')
plt.title('Normalized Confusion Matrix')
plt.show()
```

## ◆ 5. Метрики из матрицы

| Метрика              | Формула                                             | Интерпретация          |
|----------------------|-----------------------------------------------------|------------------------|
| Accuracy             | $(TP+TN)/(TP+TN+FP+FN)$                             | Общая точность         |
| Precision            | $TP/(TP+FP)$                                        | Точность положительных |
| Recall (Sensitivity) | $TP/(TP+FN)$                                        | Полнота                |
| Specificity          | $TN/(TN+FP)$                                        | Полнота отрицательных  |
| F1-Score             | $2 * \frac{Precision * Recall}{Precision + Recall}$ | Гармоническое среднее  |

## ◆ 6. Вычисление метрик

```
from sklearn.metrics import (
    accuracy_score, precision_score,
    recall_score, f1_score
)

# Из confusion matrix
tn, fp, fn, tp = confusion_matrix(y_true,
y_pred).ravel()

accuracy = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
specificity = tn / (tn + fp)
f1 = 2 * (precision * recall) / (precision + recall)

print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"F1-Score: {f1:.3f}")

# Или напрямую из sklearn
print(f"Accuracy: {accuracy_score(y_true,
y_pred):.3f}")
print(f"Precision: {precision_score(y_true,
y_pred):.3f}")
print(f"Recall: {recall_score(y_true, y_pred):.3f}")
print(f"F1-Score: {f1_score(y_true, y_pred):.3f}")
```

## ◆ 7. Мультиклассовая классификация

```
# Мультиклассовая confusion matrix
y_true_multi = [0, 1, 2, 0, 1, 2, 1, 2]
y_pred_multi = [0, 2, 1, 0, 1, 1, 1, 2]

cm_multi = confusion_matrix(y_true_multi,
y_pred_multi)
print(cm_multi)

# Визуализация
plt.figure(figsize=(10, 8))
sns.heatmap(cm_multi, annot=True, fmt='d',
cmap='Blues',
xticklabels=['Class 0', 'Class 1', 'Class
2'],
yticklabels=['Class 0', 'Class 1', 'Class
2'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Multiclass Confusion Matrix')
plt.show()

# Метрики для каждого класса
from sklearn.metrics import classification_report
print(classification_report(y_true_multi,
y_pred_multi))
```

## ◆ 8. Интерпретация ошибок

### Type I Error (False Positive)

- ✗ Ложная тревога
- ✗ Предсказали положительный, на самом деле отрицательный
- ✗ Пример: здоровый человек диагностирован больным

### Type II Error (False Negative)

- ✗ Пропуск события
- ✗ Предсказали отрицательный, на самом деле положительный
- ✗ Пример: больной человек не диагностирован

## ◆ 9. Classification Report

```
from sklearn.metrics import classification_report

# Полный отчёт
print(classification_report(y_true, y_pred,
target_names=['Negative',
'Positive']))

# Вывод:
#             precision    recall   f1-score
# support
# #           Negative      0.75     0.75     0.75
# 4           Positive      0.75     0.75     0.75
# 4
# #
# accuracy                           0.75
# 8
# macro avg       0.75     0.75     0.75
# 8
# weighted avg    0.75     0.75     0.75
# 8
```

## ◆ 10. Чек-лист

- [ ] Построить confusion matrix
- [ ] Визуализировать с цветовой картой
- [ ] Вычислить Precision, Recall, F1
- [ ] Проанализировать ошибки (FP vs FN)
- [ ] Для мультикласса: смотреть на перепутанные классы
- [ ] Нормализовать по строкам для интерпретации
- [ ] Использовать classification\_report

### 💡 Объяснение заказчику:

«Confusion Matrix — это детальный отчёт об ошибках модели: мы видим не только общую точность, но и конкретно, какие типы ошибок делает модель и как часто».

# Constrained Optimization в ML

17 5 января 2026

## 1. Суть методов

**Constrained optimization** — минимизация функции потерь с ограничениями

```
minimize f(x)
subject to g(x) ≤ 0 (неравенства)
          h(x) = 0 (равенства)
```

- **f(x)** — функция потерь (loss function)
- **g(x), h(x)** — ограничения (constraints)
- **x** — параметры модели или решения

В ML часто нужно не просто минимизировать ошибку, но и соблюдать определённые условия

## 2. Зачем нужны ограничения в ML

- **Fairness:** равные ошибки для разных групп
- **Privacy:** differential privacy bounds
- **Monotonicity:** монотонность по определённым признакам
- **Shape constraints:** выпуклость, гладкость функции
- **Sparsity:** ограничение на число ненулевых параметров
- **Resource limits:** ограничения памяти, времени
- **Domain knowledge:** физические/бизнес ограничения

## 3. Типы ограничений

| Тип               | Пример                      |
|-------------------|-----------------------------|
| Box constraints   | $0 \leq w_i \leq 1$         |
| Linear equality   | $\sum w_i = 1$              |
| Linear inequality | $Aw \leq b$                 |
| Quadratic         | $\ w\ ^2 \leq C$            |
| Convex            | $f(w) \leq 0$ , f выпуклая  |
| Nonlinear         | $g(w) = 0$ , g произвольная |

## 4. Метод множителей Лагранжа

**Основа теории:** превращаем ограничения в штрафы

$$L(x, \lambda, \mu) = f(x) + \sum \lambda_i \cdot g_i(x) + \sum \mu_j \cdot h_j(x)$$

- $\lambda, \mu$  — множители Лагранжа (Lagrange multipliers)
- **KKT условия:** необходимые условия оптимальности

**Dual problem:** максимизация по  $\lambda, \mu$  (часто проще решить)

SVM использует dual формулировку для эффективного решения

## 5. Методы решения

### 1. Penalty methods (штрафные функции):

# Превращаем constraint в penalty  
 $\text{minimize } f(x) + \rho \cdot \sum \max(0, g_i(x))^2$

### 2. Barrier methods (барьерные функции):

# Барьер внутри допустимой области  
 $\text{minimize } f(x) - \mu \cdot \sum \log(-g_i(x))$

### 3. Projected gradient descent:

# Градиентный шаг + проекция на допустимую область  
 $x_{t+1} = \text{project}(x_t - \alpha \cdot \nabla f(x_t))$

### 4. Augmented Lagrangian (ADMM):

- Комбинирует Lagrange multipliers и penalty

## ◆ 6. Projected Gradient Descent

```
import numpy as np

def project_simplex(x):
    """Проекция на симплекс: x >= 0, sum(x) = 1"""
    n = len(x)
    x_sorted = np.sort(x)[::-1]
    cumsum = np.cumsum(x_sorted)
    idx = np.arange(n) + 1
    rho = np.where(x_sorted > (cumsum - 1) / idx)[0][-1]
    theta = (cumsum[rho] - 1) / (rho + 1)
    return np.maximum(x - theta, 0)

def pgd(f, grad_f, x0, n_iter=1000, lr=0.01):
    """Projected Gradient Descent"""
    x = x0.copy()
    for i in range(n_iter):
        # Gradient step
        x = x - lr * grad_f(x)
        # Projection
        x = project_simplex(x)
    return x
```

## ◆ 7. Практика: scipy.optimize

```
from scipy.optimize import minimize

# Функция и ограничения
def objective(x):
    return x[0]**2 + x[1]**2

def constraint_eq(x):
    return x[0] + x[1] - 1

def constraint_ineq(x):
    return x[0] - x[1]

# Определяем ограничения
constraints = [
    {'type': 'eq', 'fun': constraint_eq},
    {'type': 'ineq', 'fun': constraint_ineq}
]

# Bounds: 0 <= x_i <= 1
bounds = [(0, 1), (0, 1)]

# Решение
result = minimize(
    objective,
    x0=[0.5, 0.5],
    method='SLSQP', # Sequential Least Squares
    Programming
    bounds=bounds,
    constraints=constraints
)
print(result.x, result.fun)
```

## ◆ 8. Constrained ML: примеры

### Пример 1: Portfolio Optimization

```
# min risk, subject to: sum(w) = 1, w >= 0
from scipy.optimize import minimize
import numpy as np

def portfolio_variance(w, Sigma):
    return w @ Sigma @ w

def optimize_portfolio(returns, Sigma):
    n = len(returns)
    # Ограничения
    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    ]
    bounds = [(0, 1)] * n

    result = minimize(
        lambda w: portfolio_variance(w, Sigma),
        x0=np.ones(n)/n,
        method='SLSQP',
        bounds=bounds,
        constraints=constraints
    )
    return result.x
```

## ◆ 9. Fairness Constraints

**Задача:** модель должна иметь равную точность для разных групп

```
# Constraint: |TPR_group1 - TPR_group2| <= ε
# TPR = True Positive Rate

from sklearn.linear_model import LogisticRegression

# Fairness-aware обучение
def fair_logistic_regression(X, y, sensitive_attr, epsilon=0.05):
    # Разделяем на группы
    group1 = sensitive_attr == 0
    group2 = sensitive_attr == 1

    # Стандартная модель
    model = LogisticRegression()

    # Добавляем fairness constraint через
    # post-processing или regularization
    # (упрощённый пример)

    return model
```

**Библиотеки:**

- `fairlearn` — fairness constraints в `sklearn`
- `AIF360` — IBM AI Fairness 360

## ◆ 10. ADMM (Alternating Direction Method of Multipliers)

**Мощный метод для больших задач:**

```
# Решаем: min f(x) + g(z), subject to Ax + Bz = c
# ADMM итерации:
x_{k+1} = argmin_x L_ρ(x, z_k, u_k)
z_{k+1} = argmin_z L_ρ(x_{k+1}, z, u_k)
u_{k+1} = u_k + ρ(Ax_{k+1} + Bz_{k+1} - c)
```

**Применение:**

- Lasso регрессия с дополнительными ограничениями
- Распределённая оптимизация (каждая машина решает свою часть)
- Consensus optimization

## ◆ 11. Constrained Neural Networks

**Способы добавления ограничений:**

- **Penalty в loss:** `loss = mse + λ · penalty`
- **Projection layers:** проекция после каждого слоя
- **Constrained architectures:** архитектура гарантирует constraint
- **Augmented Lagrangian:** добавляем Lagrange multipliers

```
# PyTorch: monotonicity constraint
class MonotonicNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        # Используем положительные веса для
        # монотонности
        with torch.no_grad():
            self.fc1.weight.clamp_(min=0)
            self.fc2.weight.clamp_(min=0)

        h = F.relu(self.fc1(x))
        return self.fc2(h)
```

## ◆ 12. Методы оптимизации

| Метод          | Библиотека      | Применение                    |
|----------------|-----------------|-------------------------------|
| SLSQP          | scipy.optimize  | Общего назначения             |
| Interior Point | scipy, cvxpy    | Выпуклые задачи               |
| Active Set     | scipy, quadprog | QP задачи                     |
| ADMM           | cvxpy           | Большие, распределённые       |
| Projected GD   | Custom          | Простые constraints           |
| Frank-Wolfe    | Custom          | Структурированные constraints |

## ◆ 13. Выпуклая оптимизация (CVXPY)

```
import cvxpy as cp

# Переменные
x = cp.Variable(n)

# Objective
objective = cp.Minimize(cp.sum_squares(A @ x - b))

# Constraints
constraints = [
    x >= 0,           # Non-negative
    cp.sum(x) == 1,   # Sum to 1
    cp.norm(x, 1) <= c # L1 constraint
]

# Решение
problem = cp.Problem(objective, constraints)
problem.solve()

print("Optimal x:", x.value)
print("Optimal value:", problem.value)
```

CVXPY автоматически выбирает solver (ECOS, SCS, OSQP)

## ◆ 14. Типичные ошибки

- ✗ **Несовместные ограничения** — проверьте feasibility
- ✗ **Слишком жёсткие ограничения** — модель не может обучиться
- ✗ **Игнорирование выпуклости** — невыпуклые задачи сложнее
- ✗ **Неправильная нормализация** constraints и objective
- ✗ **Забыть про численную стабильность**
- ✓ Проверять ККТ условия для оптимальности
- ✓ Начинать с feasible point
- ✓ Использовать готовые библиотеки (cvxpy, scipy)

## ◆ 15. Когда использовать

### ✓ Constrained optimization когда:

- ✓ Есть чёткие бизнес/физические ограничения
- ✓ Нужна fairness или privacy
- ✓ Требуется интерпретируемость (monotonicity)
- ✓ Ограничения критичны для применения
- ✓ Задача выпуклая

### ✗ Не нужно если:

- ✗ Нет чётких ограничений
- ✗ Ограничения можно добавить в loss
- ✗ Задача слишком сложная (deep NN)
- ✗ Constraints меняются часто

## ◆ 16. Чек-лист

- [ ] Формализовать ограничения математически
- [ ] Проверить feasibility (есть ли допустимые точки)
- [ ] Определить тип задачи (выпуклая/ невыпуклая)
- [ ] Выбрать метод решения
- [ ] Нормализовать objective и constraints
- [ ] Реализовать с библиотекой (cvxpy, scipy)
- [ ] Проверить ККТ условия для решения
- [ ] Валидировать constraints на test set
- [ ] Сравнить с unconstrained baseline

**Золотое правило:** если задача выпуклая с линейными/квадратичными ограничениями — используйте готовые солверы (cvxpy), иначе — projected gradient или ADMM.

# Контентная фильтрация

 17 Январь 2026

## ◆ 1. Суть метода

- **Идея:** рекомендовать элементы на основе их характеристик (контента)
- **Принцип:** если понравился товар с определенными свойствами, рекомендуем похожие
- **Основа:** профиль пользователя + профиль товара
- **Не требует:** данных о других пользователях
- **Решает проблему:** холодного старта для новых товаров

## ◆ 3. Архитектура системы

1. **Профиль товара:** вектор признаков товара (жанр, цена, бренд и т.д.)
2. **Профиль пользователя:** вектор предпочтений (усредненный или взвешенный)
3. **Мера похожести:** косинусное сходство, евклидово расстояние
4. **Ранжирование:** сортировка по сходству
5. **Фильтрация:** удаление уже просмотренных

## ◆ 2. Vs коллаборативная фильтрация

| Аспект            | Контентная             | Коллаборативная         |
|-------------------|------------------------|-------------------------|
| Основа            | Свойства товаров       | Поведение пользователей |
| Холодный старт    | Только пользователи    | Пользователи и товары   |
| Разнообразие      | Низкое (filter bubble) | Высокое (serendipity)   |
| Объяснимость      | Высокая                | Низкая                  |
| Требует признаков | Да                     | Нет                     |

## ◆ 4. Базовый пример: фильмы

```

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Данные о фильмах
movies = pd.DataFrame({
    'title': ['Terminator', 'Titanic', 'Avatar', 'True Lies'],
    'genres': ['Action Sci-Fi', 'Drama Romance', 'Action Sci-Fi Adventure', 'Action Comedy'],
    'director': ['Cameron', 'Cameron', 'Cameron', 'Cameron'],
    'year': [1984, 1997, 2009, 1994]
})

# Создать признаки из текста
movies['features'] = (movies['genres'] + ' ' +
                      movies['director'])

# TF-IDF векторизация
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['features'])

# Вычислить схожесть
cosine_sim = cosine_similarity(tfidf_matrix,
                                tfidf_matrix)

def get_recommendations(title, top_n=3):
    # Найти индекс фильма
    idx = movies[movies['title']] ==
          title].index[0]

    # Получить оценки схожести
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Отсортировать по схожести
    sim_scores = sorted(sim_scores,
                        key=lambda x: x[1],
                        reverse=True)

    # Получить top N (исключая сам фильм)
    sim_scores = sim_scores[1:top_n+1]

    # Индексы похожих фильмов
    movie_indices = [i[0] for i in sim_scores]

    return movies['title'].iloc[movie_indices]

# Рекомендации

```

## Контентная фильтрация Cheatsheet — 3 колонки

```

print(get_recommendations('Terminator'))
# Output: Avatar, True Lies

```

## ◆ 5. Создание профиля пользователя

### Метод 1: Усреднение

```

# Профиль = среднее векторов понравившихся товаров
user_likes = [0, 2] # индексы фильмов
user_profile =
tfidf_matrix[user_likes].mean(axis=0)

# Найти похожие
similarities = cosine_similarity(user_profile,
tfidf_matrix)
recommendations = np.argsort(similarities[0])
[::-1]

```

### Метод 2: Взвешивание по рейтингу

```

# Если есть рейтинги (1-5)
ratings = np.array([5, 4]) # рейтинги
пользователя
weights = ratings / ratings.sum()

# Взвешенный профиль
user_profile = (tfidf_matrix[user_likes].T *
weights).T.sum(axis=0)

```

## ◆ 6. Извлечение признаков

### Текстовые признаки:

```

from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF для текста
tfidf = TfidfVectorizer(
    max_features=100,
    stop_words='english',
    ngram_range=(1, 2) # униграммы и биграммы
)

text_features =
tfidf.fit_transform(movies['description'])

```

### Категориальные признаки:

```

from sklearn.preprocessing import MultiLabelBinarizer

# Жанры как множество
genres = movies['genres'].str.split()
mlb = MultiLabelBinarizer()
genre_features = mlb.fit_transform(genres)

```

### Числовые признаки:

```

from sklearn.preprocessing import StandardScaler

# Нормализация числовых признаков
scaler = StandardScaler()
numeric_features =
scaler.fit_transform(movies[['year', 'budget']])

```

## ◆ 7. Объединение признаков

```
from scipy.sparse import hstack
import numpy as np

# Объединить разные типы признаков
combined_features = hstack([
    text_features,          # TF-IDF текста
    genre_features,         # One-hot жанров
    numeric_features         # Нормализованные
    числа
])

# Если нужно взвесить разные типы
text_weight = 0.7
genre_weight = 0.2
numeric_weight = 0.1

weighted_features = hstack([
    text_features * text_weight,
    genre_features * genre_weight,
    numeric_features * numeric_weight
])
```

## ◆ 8. Меры похожести

### Косинусное сходство (самое популярное):

```
from sklearn.metrics.pairwise import
cosine_similarity

sim = cosine_similarity(user_profile,
item_features)
```

### Евклидово расстояние:

```
from sklearn.metrics.pairwise import
euclidean_distances

dist = euclidean_distances(user_profile,
item_features)
# Преобразовать в сходство
sim = 1 / (1 + dist)
```

### Корреляция Пирсона:

```
from scipy.stats import pearsonr

def pearson_similarity(v1, v2):
    corr, _ = pearsonr(v1.flatten(), v2.flatten())
    return corr
```

## ◆ 9. Класс рекомендательной системы

```
class ContentBasedRecommender:
    def __init__(self, items_df, feature_cols):
        self.items = items_df
        self.feature_cols = feature_cols
        self.tfidf = TfidfVectorizer()
        self.item_profiles = None

    def fit(self):
        # Объединить признаки в текст
        features_text =
        self.items[self.feature_cols].apply(
            lambda x: ' '.join(x.astype(str)),
            axis=1
        )

        # Создать матрицу признаков
        self.item_profiles =
        self.tfidf.fit_transform(features_text)
        return self

    def build_user_profile(self, user_items,
ratings=None):
        # Профиль на основе взаимодействий
        if ratings is None:
            # Простое усреднение
            return
        self.item_profiles[user_items].mean(axis=0)
        else:
            # Взвешенное по рейтингам
            weights = np.array(ratings) /
            sum(ratings)
            return
        (self.item_profiles[user_items].T *
        weights).T.sum(axis=0)

    def recommend(self, user_profile, top_n=10,
exclude_items=None):
        # Вычислить сходство
        similarities =
        cosine_similarity(user_profile,
        self.item_profiles)

        # Отсортировать
        indices = np.argsort(similarities[0])
        [::-1]

        # Исключить уже просмотренные
        if exclude_items:
            indices = [i for i in indices if i not
            in exclude_items]

        return indices[:top_n]

    # Использование
    recommender = ContentBasedRecommender(movies,
```

```
['genres', 'director'])
recommender.fit()

user_items = [0, 2] # Пользователю понравились фильмы 0 и 2
user_profile =
recommender.build_user_profile(user_items)
recommendations =
recommender.recommend(user_profile, top_n=5,
exclude_items=user_items)
```

## ◆ 10. Обработка текста для контента

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Предобработка текста
def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Удалить пунктуацию
    text = ''.join([c for c in text if c.isalnum() or c.isspace()])

    # Токенизация
    tokens = text.split()

    # Удалить стоп-слова
    stop_words = set(stopwords.words('english'))
    tokens = [t for t in tokens if t not in stop_words]

    # Стемминг
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(t) for t in tokens]

    return ' '.join(tokens)

# Применить
movies['processed_desc'] =
movies['description'].apply(preprocess_text)
```

## ◆ 11. Feature engineering для контента

- Текстовые:** TF-IDF, Word2Vec, BERT embeddings
- Категориальные:** One-hot, multi-label бинаризация
- Числовые:** нормализация, биннинг
- Темпоральные:** возраст товара, сезонность
- Графовые:** связи между товарами
- Визуальные:** CNN embeddings для изображений

```
# Word2Vec для семантических признаков
from gensim.models import Word2Vec

texts = [desc.split() for desc in movies['description']]
model = Word2Vec(texts, vector_size=100, window=5,
min_count=1)

# Усреднить векторы слов
def text_to_vec(text):
    words = text.split()
    vectors = [model.wv[w] for w in words if w in model.wv]
    return np.mean(vectors, axis=0) if vectors
else np.zeros(100)

movie_vecs = np.array([text_to_vec(desc) for desc
in movies['description']])
```

## ◆ 12. Проблема filter bubble

Контентная фильтрация может создавать "пузырь фильтров" - рекомендовать только похожие товары

### Решения:

- Добавить элемент случайности (exploration)
- Комбинировать с коллаборативной фильтрацией
- Использовать serendipity score
- Diversity-aware ранжирование

```
# Добавить exploration (epsilon-greedy)
import random

epsilon = 0.1 # 10% случайных рекомендаций

if random.random() < epsilon:
    # Случайная рекомендация
    recommendations =
random.sample(range(len(movies)), top_n)
else:
    # Контентная рекомендация
    recommendations =
recommender.recommend(user_profile, top_n)
```

## ◆ 13. Diversity в рекомендациях

```
def diversify_recommendations(items, similarities,
diversity_weight=0.3):
    """Максимизировать релевантность и
разнообразие"""
    selected = []
    candidates = list(range(len(items)))

    # Выбрать самый релевантный
    best_idx = np.argmax(similarities)
    selected.append(best_idx)
    candidates.remove(best_idx)

    # Итеративно добавлять разнообразные
    while len(selected) < top_n and candidates:
        scores = []
        for c in candidates:
            # Релевантность
            relevance = similarities[c]

            # Разнообразие (минимальное сходство с
            # выбранными)
            diversity = min([1 -
cosine_similarity(
                item_profiles[c], item_profiles[s]
            )[0][0] for s in selected])

            # Комбинированный score
            score = relevance * (1 -
diversity_weight) + diversity * diversity_weight
            scores.append((c, score))

        # Выбрать лучший
        best = max(scores, key=lambda x: x[1])
        selected.append(best[0])
        candidates.remove(best[0])

    return selected
```

## ◆ 14. Холодный старт

### Новый пользователь:

- Запросить предпочтения (анкета)
- Показать популярные товары
- Использовать демографические данные

```
# Демографический профиль
def demographic_profile(user_age, user_gender):
    # Найти похожих пользователей
    similar_users = users[
        (users['age'] == user_age) &
        (users['gender'] == user_gender)
    ]

    # Усреднить их профили
    return
    item_profiles[similar_users['liked_items']].mean(axis=1)
```

**Новый товар:** не проблема! Есть признаки контента

## ◆ 15. Оценка качества

```
from sklearn.model_selection import
train_test_split
from sklearn.metrics import precision_score,
recall_score

# Разделить взаимодействия пользователя
train_items, test_items = train_test_split(
    user_interactions, test_size=0.2
)

# Обучить на train
user_profile =
recommender.build_user_profile(train_items)
recommendations =
recommender.recommend(user_profile, top_n=10)

# Оценить на test
hits = len(set(recommendations) & set(test_items))
precision = hits / len(recommendations)
recall = hits / len(test_items)

print(f"Precision@10: {precision:.3f}")
print(f"Recall@10: {recall:.3f}")

# F1-score
f1 = 2 * (precision * recall) / (precision +
recall) if (precision + recall) > 0 else 0
print(f"F1@10: {f1:.3f}")
```

## ◆ 16. Гибридные системы

Комбинировать контентную и коллаборативную фильтрацию

```
# Weighted hybrid
def hybrid_recommend(user_id, item_id, alpha=0.5):
    # Контентная оценка
    content_score = cosine_similarity(
        user_profile[user_id],
        item_profile[item_id]
    )

    # Коллаборативная оценка
    collab_score =
collaborative_model.predict(user_id, item_id)

    # Взвешенная комбинация
    hybrid_score = alpha * content_score + (1 -
alpha) * collab_score

    return hybrid_score

# Switching hybrid
def switching_hybrid(user_id, item_id):
    # Если товар новый -> контентная
    if is_new_item(item_id):
        return content_based_recommend(user_id,
item_id)
    # Иначе -> коллаборативная
    else:
        return collaborative_recommend(user_id,
item_id)
```

## ◆ 17. Обновление профиля пользователя

```
# Инкрементальное обновление
class IncrementalUserProfile:
    def __init__(self, decay=0.9):
        self.profile = None
        self.decay = decay # Затухание старых
взаимодействий

    def update(self, new_item_vector, rating=1.0):
        if self.profile is None:
            self.profile = new_item_vector *
rating
        else:
            # Применить затухание к старому
профилю
            self.profile = self.profile * *
self.decay + new_item_vector * rating
            # Нормализовать
            self.profile = self.profile /
np.linalg.norm(self.profile)

    def get_profile(self):
        return self.profile

# Использование
user_prof = IncrementalUserProfile(decay=0.95)

for item_idx, rating in user_interactions:
    user_prof.update(item_profiles[item_idx],
rating)
```

## ◆ 18. Когда использовать

### ✓ Хорошо

- ✓ Есть богатые метаданные о товарах
- ✓ Новые товары появляются часто
- ✓ Нужна объяснимость рекомендаций
- ✓ Мало данных о взаимодействиях
- ✓ Текстовый контент (статьи, новости)
- ✓ Нужна персонализация с первого дня

### ✗ Плохо

- ✗ Мало признаков товаров
- ✗ Сложно извлечь качественные признаки
- ✗ Нужны неожиданные рекомендации
- ✗ Товары очень разнообразны
- ✗ Признаки не коррелируют с предпочтениями

## ◆ 19. Практические советы

- Используйте TF-IDF для текста, не bag-of-words
- Нормализуйте все признаки перед объединением
- Экспериментируйте с весами разных типов признаков
- Добавляйте временные признаки (свежесть контента)
- Фильтруйте очевидные рекомендации (тот же автор/жанр)
- Логируйте клики для A/B тестов
- Периодически обновляйте профили пользователей
- Комбинируйте с popularity-based для холодного старта

## ◆ 20. Чек-лист

- [ ] Определить, какие признаки товаров доступны
- [ ] Выбрать методы извлечения признаков (TF-IDF, embeddings)
- [ ] Нормализовать и объединить разные типы признаков
- [ ] Выбрать меру похожести (обычно косинусная)
- [ ] Построить профиль пользователя из истории
- [ ] Реализовать ранжирование по сходству
- [ ] Добавить фильтрацию просмотренных товаров
- [ ] Решить проблему filter bubble (diversity)
- [ ] Обработать холодный старт пользователей
- [ ] Оценить качество (precision, recall, diversity)
- [ ] Рассмотреть гибридный подход

### 💡 Объяснение заказчику:

«Контентная фильтрация работает как личный продавец, который знает ваши предпочтения. Если вам понравились боевики с Арнольдом Шварценеггером, система порекомендует похожие фильмы: тот же жанр, актер или режиссер. Это как если бы продавец сказал: "Раз вам понравилась эта книга, вот похожие по стилю и теме"».



# Context-aware рекомендации

 Январь 2026

## ◆ 1. Суть контекстных рекомендаций

**Учет контекста (время, место, устройство, социальный контекст)**

- **Ключевая концепция:** детали и примеры для суть контекстных рекомендаций
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 Суть контекстных рекомендаций — важный аспект для понимания темы

## ◆ 2. Типы контекста

**Temporal, Spatial, Social, Device, Mood, Weather**

- **Ключевая концепция:** детали и примеры для типы контекста
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 Типы контекста — важный аспект для понимания темы

## ◆ 3. Контекстуальное pre-filtering

### Фильтрация данных по контексту до построения модели

- Ключевая концепция:** детали и примеры для контекстуальное pre-filtering
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Контекстуальное pre-filtering — важный аспект для понимания темы

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import
RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

### Context-aware рекомендации Cheatsheet — 3 колонки

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 4. Контекстуальное post-filtering

### Фильтрация рекомендаций по контексту после построения

- Ключевая концепция:** детали и примеры для контекстуальное post-filtering
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Контекстуальное post-filtering — важный аспект для понимания темы

## ◆ 5. Контекстуальное моделирование

### Включение контекста в модель (Tensor Factorization, Factorization Machines)

- Ключевая концепция:** детали и примеры для контекстуальное моделирование
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Контекстуальное моделирование — важный аспект для понимания темы

## ◆ 6. Tensor Factorization

**Разложение тензора (пользователь × объект × контекст)**

- **Ключевая концепция:** детали и примеры для tensor factorization
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Tensor Factorization — важный аспект для понимания темы*

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import
RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Context-aware рекомендации Cheatsheet — 3 колонки

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 7. Factorization Machines

**Универсальная модель для учета признаков и контекста**

- **Ключевая концепция:** детали и примеры для factorization machines
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Factorization Machines — важный аспект для понимания темы*

## ◆ 8. Временной контекст

**Time-of-day, день недели, сезон, праздники**

- **Ключевая концепция:** детали и примеры для временной контекст
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Временной контекст — важный аспект для понимания темы*

## ◆ 9. Пространственный контекст

**Локация, расстояние, POI**

- **Ключевая концепция:** детали и примеры для пространственный контекст
- **Применение:** практическое использование в реальных задачах
- **Инструменты:** библиотеки и фреймворки
- **Best practices:** рекомендации экспертов
- **Типичные ошибки:** чего следует избегать
- **Метрики:** как измерять эффективность

 *Пространственный контекст — важный аспект для понимания темы*

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import
train_test_split
```

```
# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']
```

```
# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Обучение модели
from sklearn.ensemble import
RandomForestClassifier
```

```
model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)
```

```
# Оценка
from sklearn.metrics import accuracy_score,
classification_report
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

```
print(f"Accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))
```

## ◆ 10. Социальный контекст

### Один, с семьей, с друзьями, на работе

- Ключевая концепция:** детали и примеры для социальный контекст
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Социальный контекст — важный аспект для понимания темы

## ◆ 11. Практические примеры

### Рекомендации ресторанов, музыки, фильмов

- Ключевая концепция:** детали и примеры для практические примеры
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Практические примеры — важный аспект для понимания темы

## ◆ 12. Сбор контекстных данных

### Implicit vs explicit, privacy concerns

- Ключевая концепция:** детали и примеры для сбор контекстных данных
- Применение:** практическое использование в реальных задачах
- Инструменты:** библиотеки и фреймворки
- Best practices:** рекомендации экспертов
- Типичные ошибки:** чего следует избегать
- Метрики:** как измерять эффективность

 Сбор контекстных данных — важный аспект для понимания темы

```
# Пример кода
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

# Загрузка данных
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# Разделение на train/test
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Обучение модели
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)

# Оценка
from sklearn.metrics import accuracy_score,
classification_report

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

# Contrastive Learning

17 Январь 2026

## ◆ 1. Основы Contrastive Learning

- **Цель:** научить модель различать похожие и непохожие объекты
- **Идея:** положительные пары близко, отрицательные далеко в пространстве признаков
- **Без меток:** self-supervised обучение
- **Применение:** CV, NLP, audio, multimodal
- **Результат:** качественные embedding'и

## ◆ 2. Принцип работы

### Положительные пары:

- Augmented версии одного изображения
- Разные ракурсы одного объекта
- Предложение и его перефразировка

### Отрицательные пары:

- Разные изображения из датасета
- Другие объекты в батче
- Случайные примеры

### Функция потерь:

- NT-Xent (Normalized Temperature-scaled Cross Entropy)
- InfoNCE Loss
- Triplet Loss

## ◆ 3. SimCLR Framework

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms

class SimCLR(nn.Module):
    def __init__(self, base_encoder, projection_dim=128):
        super().__init__()
        self.encoder = base_encoder

        # Projection head
        # ResNet-50 выход: 2048
        self.projection = nn.Sequential(
            nn.Linear(2048, 2048),
            nn.ReLU(),
            nn.Linear(2048, projection_dim)
        )

    def forward(self, x):
        h = self.encoder(x)
        z = self.projection(h)
        return F.normalize(z, dim=1)

# Data augmentation - КЛЮЧЕВОЙ компонент
def get_simclr_augmentation():
    return transforms.Compose([
        transforms.RandomResizedCrop(224, scale=(0.2, 1.0)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomApply([
            transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)
        ], p=0.8),
        transforms.RandomGrayscale(p=0.2),
        transforms.GaussianBlur(kernel_size=23),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
```

dtype=torch.long)

```
loss = F.cross_entropy(logits, labels)
return loss
```

## ◆ 4. NT-Xent Loss

```
class NTXentLoss(nn.Module):
    def __init__(self, temperature=0.5):
        super().__init__()
        self.temperature = temperature

    def forward(self, z_i, z_j):
        """
        z_i, z_j: (batch_size, projection_dim)
        Положительные пары: (z_i[k], z_j[k])
        """
        batch_size = z_i.shape[0]

        # Объединяем оба представления
        z = torch.cat([z_i, z_j], dim=0) #
        (2*batch, dim)

        # Вычисляем косинусное сходство
        sim = F.cosine_similarity(
            z.unsqueeze(1), z.unsqueeze(0), dim=2
        ) # (2*batch, 2*batch)

        # Применяем температуру
        sim = sim / self.temperature

        # Мaska для положительных пар
        # Для каждого i положительный - i+batch
        или i-batch
        positive_mask = torch.zeros_like(sim,
   dtype=torch.bool)
        for i in range(batch_size):
            positive_mask[i, i + batch_size] =
True
            positive_mask[i + batch_size, i] =
True

        # Мaska для самих себя (диагональ)
        identity_mask = torch.eye(2 * batch_size,
device=z.device,
   dtype=torch.bool)

        # Логиты для всех отрицательных пар
        negatives = sim.masked_fill(identity_mask,
-9e15)

        # Логиты для положительных пар
        positives = sim[positive_mask].view(2 *
batch_size, 1)

        # NT-Xent loss
        logit = torch.cat([positives, negatives],
dim=1)
        labels = torch.zeros(2 * batch_size,
device=z.device,
```

## ◆ 5. Обучение SimCLR

```
from torchvision import models

# Создание модели
base_encoder = models.resnet50(pretrained=False)
base_encoder.fc = nn.Identity() # Удаляем
последний FC слой

model = SimCLR(base_encoder, projection_dim=128)
model = model.cuda()

# Оптимизатор
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.0003,
    weight_decay=1e-6
)

# Loss
criterion = NTXentLoss(temperature=0.5)

# Augmentation
augment = get_simclr_augmentation()

# Training loop
model.train()
for epoch in range(100):
    for images, _ in dataloader:
        # Создаем две augmented версии
        images_i = torch.stack([augment(img) for
img in images])
        images_j = torch.stack([augment(img) for
img in images])

        images_i = images_i.cuda()
        images_j = images_j.cuda()

        # Forward pass
        z_i = model(images_i)
        z_j = model(images_j)

        # Loss
        loss = criterion(z_i, z_j)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        print(f'Epoch {epoch}, Loss:
{loss.item():.4f}')
```

## ◆ 6. MoCo (Momentum Contrast)

```
class MoCo(nn.Module):
    def __init__(self, base_encoder, dim=128,
                 K=65536, m=0.999, T=0.07):
        super().__init__()
        self.K = K # Размер очереди
        self.m = m # Momentum для обновления
        self.T = T # Temperature

        # Encoder query
        self.encoder_q = base_encoder
        self.encoder_q.fc = nn.Sequential(
            nn.Linear(2048, 2048),
            nn.ReLU(),
            nn.Linear(2048, dim)
        )

        # Encoder key (momentum encoder)
        self.encoder_k = base_encoder
        self.encoder_k.fc = nn.Sequential(
            nn.Linear(2048, 2048),
            nn.ReLU(),
            nn.Linear(2048, dim)
        )

        # Копируем параметры
        for param_q, param_k in zip(
            self.encoder_q.parameters(),
            self.encoder_k.parameters()
        ):
            param_k.data.copy_(param_q.data)
            param_k.requires_grad = False

        # Очередь отрицательных примеров
        self.register_buffer(
            "queue",
            torch.randn(dim, K)
        )
        self.queue = F.normalize(self.queue,
                                dim=0)
        self.register_buffer("queue_ptr",
                            torch.zeros(1, dtype=torch.long))

    @torch.no_grad()
    def _momentum_update_key_encoder(self):
        """Momentum update для key encoder"""
        for param_q, param_k in zip(
            self.encoder_q.parameters(),
            self.encoder_k.parameters()
        ):
            param_k.data = param_k.data * self.m +
                           param_q.data * (1. -
   self.m)

    @torch.no_grad()
```

## Contrastive Learning Cheatsheet — 3 колонки

```
def _dequeue_and_enqueue(self, keys):
    """Обновление очереди"""
    batch_size = keys.shape[0]
    ptr = int(self.queue_ptr)

    # Добавляем в очередь
    self.queue[:, ptr:ptr + batch_size] =
    keys.T
    ptr = (ptr + batch_size) % self.K

    self.queue_ptr[0] = ptr

    def forward(self, im_q, im_k):
        # Query
        q = self.encoder_q(im_q)
        q = F.normalize(q, dim=1)

        # Key
        with torch.no_grad():
            self._momentum_update_key_encoder()
            k = self.encoder_k(im_k)
            k = F.normalize(k, dim=1)

        # Positive logits
        l_pos = torch.einsum('nc,nc->n', [q,
  k]).unsqueeze(-1)

        # Negative logits
        l_neg = torch.einsum('nc,ck->nk', [q,
  self.queue.clone().detach()])

        # Logits: (N, 1+K)
        logits = torch.cat([l_pos, l_neg], dim=1)
        / self.T

        # Labels
        labels = torch.zeros(logits.shape[0],
                             dtype=torch.long).cuda()

        # Update queue
        self._dequeue_and_enqueue(k)

        return logits, labels
```

## ◆ 7. BYOL (Bootstrap Your Own Latent)

Без отрицательных примеров!

```
class BYOL(nn.Module):
    def __init__(self, base_encoder,
                 projection_dim=256,
                 hidden_dim=4096,
                 moving_average_decay=0.996):
        super().__init__()
        self.moving_average_decay =
        moving_average_decay

        # Online network
        self.online_encoder = base_encoder
        self.online_projector = nn.Sequential(
            nn.Linear(2048, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, projection_dim)
        )
        self.online_predictor = nn.Sequential(
            nn.Linear(projection_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, projection_dim)
        )

        # Target network (EMA of online)
        self.target_encoder = base_encoder
        self.target_projector = nn.Sequential(
            nn.Linear(2048, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, projection_dim)
        )

        # Копируем параметры
        self.target_encoder.load_state_dict(
            self.online_encoder.state_dict()
        )
        self.target_projector.load_state_dict(
            self.online_predictor.state_dict()
        )

        # Замораживаем target
        for param in
        self.target_encoder.parameters():
            param.requires_grad = False
        for param in
        self.target_projector.parameters():
            param.requires_grad = False

    @torch.no_grad()
    def update_moving_average(self):
        """EMA update для target network"""
        for online, target in zip(
```

```

        self.online_encoder.parameters(),
        self.target_encoder.parameters()
    ):
        target.data =
self.moving_average_decay * target.data + \
            (1 -
self.moving_average_decay) * online.data

        for online, target in zip(
            self.online_projector.parameters(),
            self.target_projector.parameters()
        ):
            target.data =
self.moving_average_decay * target.data + \
                (1 -
self.moving_average_decay) * online.data

    def forward(self, x1, x2):
        # Online network
        online_proj_1 = self.online_projector(
            self.online_encoder(x1)
        )
        online_proj_2 = self.online_projector(
            self.online_encoder(x2)
        )

        online_pred_1 =
self.online_predictor(online_proj_1)
        online_pred_2 =
self.online_predictor(online_proj_2)

        # Target network
        with torch.no_grad():
            target_proj_1 = self.target_projector(
                self.target_encoder(x1)
            )
            target_proj_2 = self.target_projector(
                self.target_encoder(x2)
            )

        # Loss: mean squared error + normalize
        loss_1 = 2 - 2 * F.cosine_similarity(
            online_pred_1, target_proj_2.detach(),
dim=-1
        ).mean()
        loss_2 = 2 - 2 * F.cosine_similarity(
            online_pred_2, target_proj_1.detach(),
dim=-1
        ).mean()

        loss = (loss_1 + loss_2) / 2
        return loss

```

## 8. Сравнение методов

| Метод        | Отриц. примеры | Momentum | Особенность           |
|--------------|----------------|----------|-----------------------|
| SimCLR       | ✓ Batch        | ✗        | Большой batch size    |
| MoCo         | ✓ Queue        | ✓        | Очередь примеров      |
| BYOL         | ✗              | ✓        | Только positive pairs |
| SwAV         | ✗              | ✗        | Clustering online     |
| Barlow Twins | ✗              | ✗        | Redundancy reduction  |

## 9. Linear Evaluation

Оценка качества learned representations:

```

# Замораживаем encoder
for param in model.encoder.parameters():
    param.requires_grad = False

# Добавляем linear classifier
classifier = nn.Linear(2048, num_classes).cuda()

# Обучаем только classifier
optimizer = torch.optim.SGD(
    classifier.parameters(),
    lr=0.1,
    momentum=0.9
)

# Обучение
for epoch in range(100):
    for images, labels in train_loader:
        images = images.cuda()
        labels = labels.cuda()

        # Получаем features (frozen encoder)
        with torch.no_grad():
            features = model.encoder(images)

        # Классификация
        outputs = classifier(features)
        loss = F.cross_entropy(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# Оценка точности
model.eval()
classifier.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images = images.cuda()
        labels = labels.cuda()

        features = model.encoder(images)
        outputs = classifier(features)
        _, predicted = outputs.max(1)

        total += labels.size(0)
        correct += (predicted ==
labels).sum().item()

accuracy = 100. * correct / total

```

```
print(f'Linear Evaluation Accuracy:  
{accuracy:.2f}%)
```

## ◆ 10. Data Augmentation стратегии

Критичны для качества!

```
# Сильные augmentations для contrastive learning
strong_augment = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.08,
    1.0)),
    transforms.RandomHorizontalFlip(p=0.5),

    # Color jittering
    transforms.RandomApply([
        transforms.ColorJitter(
            brightness=0.4,
            contrast=0.4,
            saturation=0.4,
            hue=0.1
        ),
        p=0.8),
    # Grayscale
    transforms.RandomGrayscale(p=0.2),

    # Gaussian blur
    transforms.RandomApply([
        transforms.GaussianBlur(
            kernel_size=23,
            sigma=(0.1, 2.0)
        ),
        p=0.5),
    # Solarization (для SwAV, BYOL)
    transforms.RandomSolarize(threshold=128,
    p=0.2),

    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

# Композиция двух разных augmentations
def get_two_augmentations(image):
    return strong_augment(image),
    strong_augment(image)
```

## ◆ 11. Применения

### ✓ Хорошо работает

- ✓ Предобучение на unlabeled data
- ✓ Few-shot learning
- ✓ Transfer learning
- ✓ Image retrieval
- ✓ Обнаружение аномалий
- ✓ Multimodal learning (CLIP)

### ✗ Сложности

- ✗ Требует много GPU памяти
- ✗ Большой batch size критичен
- ✗ Долгое обучение (100+ эпох)
- ✗ Чувствительность к augmentations

## ◆ 12. Практические советы

- **Batch size:** SimCLR нужен большой (256-4096)
- **Temperature:** 0.1-0.5, влияет на hard negatives
- **Optimizer:** LARS или AdamW с weight decay
- **Learning rate:** Linear warmup + cosine decay
- **Augmentations:** критически важны, экспериментируйте
- **Projection head:** не сохраняем после обучения
- **Эпохи:** 100-1000 для хороших результатов

## ◆ 13. Чек-лист

- [ ] Выбрать подходящий метод (SimCLR, MoCo, BYOL)
- [ ] Настроить сильные augmentations
- [ ] Использовать большой batch size (если возможно)
- [ ] Выбрать правильную температуру
- [ ] Настроить learning rate schedule
- [ ] Обучать достаточно долго (100+ эпох)
- [ ] Провести linear evaluation
- [ ] Сравнить с supervised baseline
- [ ] Fine-tune на downstream задаче
- [ ] Мониторить качество embeddings

### 💡 Объяснение заказчику:

«Contrastive Learning позволяет обучить нейросеть находить хорошие представления данных без размеченных примеров — модель учится понимать, что два разных ракурса одного объекта похожи, а разные объекты различны».



# Cost-Sensitive Learning и веса классов

## ◆ 1. Основная идея

**Cost-Sensitive Learning** — подход, где разные типы ошибок имеют разную стоимость.

### Мотивация:

- Пропустить больного раком ( $FN \neq$  Ложная тревога ( $FP$ ))
- Не заблокировать мошенническую транзакцию  $\neq$  Заблокировать легитимную
- Несбалансированные классы: 99% негативных, 1% позитивных

### Матрица стоимости (Cost Matrix):

|            | Предсказано 0                 | Предсказано 1                 |
|------------|-------------------------------|-------------------------------|
| Истинный 0 | 0 (TN)                        | $C(0 \rightarrow 1)$ ( $FP$ ) |
| Истинный 1 | $C(1 \rightarrow 0)$ ( $FN$ ) | 0 (TP)                        |

Обычно:  $C(1 \rightarrow 0) \gg C(0 \rightarrow 1)$

## ◆ 2. Веса классов в sklearn

### Автоматический расчёт весов:

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Вычисляем веса для балансировки
classes = np.unique(y_train)
weights = compute_class_weight(
    class_weight='balanced',
    classes=classes,
    y=y_train
)

# weights = n_samples / (n_classes * n_samples)
print(f"Веса классов: {dict(zip(classes, weights))}")

# Пример: 900 объектов класса 0, 100 класса 1
# Веса: {0: 0.56, 1: 5.0}
```

### Применение к моделям:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Вариант 1: Автоматические веса
lr = LogisticRegression(class_weight='balanced')
rf = RandomForestClassifier(class_weight='balanced')
svc = SVC(class_weight='balanced')

# Вариант 2: Ручные веса
lr = LogisticRegression(class_weight={0: 1, 1: 5})
rf = RandomForestClassifier(class_weight={0: 1, 1: 5})
```

## ◆ 3. Веса для sample\_weight

Некоторые модели не поддерживают `class_weight`, но принимают `sample_weight`.

```
from sklearn.utils.class_weight import compute_sample_weight

# Вычисляем веса для каждого объекта
sample_weights = compute_sample_weight(
    class_weight='balanced',
    y=y_train
)

# XGBoost
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train, sample_weight=sample_weights)

# Neural networks
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
mlp.fit(X_train, y_train, sample_weight=sample_weights)

# AdaBoost
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(X_train, y_train, sample_weight=sample_weights)
```

## ◆ 4. Кастомная матрица стоимости

### Определение стоимости ошибок:

```
import numpy as np

# Матрица стоимости
# cost[i, j] = стоимость предсказания j для i
cost_matrix = np.array([
    [0, 1],      # TN, FP: стоимость FP = 1
    [10, 0]       # FN, TP: стоимость FN = 10
])

def cost_sensitive_predict(model, X, cost_matrix):
    """
    Предсказания с учётом стоимости
    """
    # Получаем вероятности
    proba = model.predict_proba(X)

    # Ожидаемая стоимость для каждого класса
    expected_costs = proba @ cost_matrix

    # Выбираем класс с минимальной стоимостью
    predictions = np.argmin(expected_costs, axis=1)

    return predictions

# Использование
lr = LogisticRegression()
lr.fit(X_train, y_train)

y_pred = cost_sensitive_predict(lr, X_test, cost_matrix)
```

## ◆ 5. Подбор оптимальных весов

### Метод 1: Grid Search

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score

# Перебор весов для minority класса
param_grid = {
    'class_weight': [
        {0: 1, 1: 1},
        {0: 1, 1: 3},
        {0: 1, 1: 5},
        {0: 1, 1: 10},
        {0: 1, 1: 20},
        'balanced'
    ]
}

grid = GridSearchCV(
    LogisticRegression(),
    param_grid,
    scoring='f1', # или кастомная метрика
    cv=5
)

grid.fit(X_train, y_train)
print(f"Лучшие веса: {grid.best_params_['cl
```

### Метод 2: Оптимизация

```
from scipy.optimize import minimize
from sklearn.model_selection import cross_val_score

def objective(weight):
    """Минимизируем валидационную ошибку"""
    model = LogisticRegression(
        class_weight={0: 1, 1: weight[0]}
    )
    score = -cross_val_score(
        model, X_train, y_train,
```

```
cv=5, scoring='f1'
).mean()
return score

result = minimize(
    objective,
    x0=[5.0], # начальное значение
    bounds=[(1, 50)], # диапазон поиска
    method='L-BFGS-B'
)

optimal_weight = result.x[0]
print(f"Оптимальный вес: {optimal_weight:.2f}")
```

## ◆ 6. Threshold moving

Альтернатива весам — изменение порога принятия решения.

### Поиск оптимального порога:

```
from sklearn.metrics import f1_score, precision_score

# Обучаем модель без весов
model = LogisticRegression()
model.fit(X_train, y_train)

# Получаем вероятности
y_proba = model.predict_proba(X_val)[:, 1]

# Перебираем пороги
thresholds = np.arange(0.05, 0.95, 0.05)
scores = []

for threshold in thresholds:
    y_pred = (y_proba >= threshold).astype(int)
    score = f1_score(y_val, y_pred)
    scores.append((threshold, score))

# Находим лучший порог
best_threshold, best_score = max(scores, key=lambda x: x[1])
print(f"Лучший порог: {best_threshold:.2f}")

# Финальные предсказания
y_pred = (y_proba >= best_threshold).astype(int)
```

## ◆ 7. Focal Loss для несбалансированных данных

**Focal Loss** — функция потерь, фокусирующаяся на сложных примерах.

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

### Реализация в Keras/TensorFlow:

```
import tensorflow as tf

def focal_loss(gamma=2., alpha=0.25):
    def focal_loss_fixed(y_true, y_pred):
        epsilon = tf.keras.backend.epsilon()
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)

        # Focal loss
        pt = tf.where(
            tf.equal(y_true, 1),
            y_pred,
            1 - y_pred
        )

        loss = -alpha * tf.pow(1. - pt, gamma) * tf.math.log(pt)
        return tf.reduce_mean(loss)

    return focal_loss_fixed

# Использование
model = tf.keras.Sequential([...])
model.compile(
    optimizer='adam',
    loss=focal_loss(gamma=2, alpha=0.75),
    metrics=['accuracy']
)
```

## ◆ 8. Взвешенная функция потерь

### Custom loss для PyTorch:

```
import torch
import torch.nn as nn

class WeightedBCELoss(nn.Module):
    def __init__(self, pos_weight=1.0):
        super().__init__()
        self.pos_weight = pos_weight

    def forward(self, y_pred, y_true):
        # Веса для каждого класса
        weights = torch.where(
            y_true == 1,
            self.pos_weight,
            1.0
        )

        # Взвешенная binary cross-entropy
        loss = nn.functional.binary_cross_entropy(
            y_pred,
            y_true,
            weight=weights
        )
        return loss

    # Использование
    criterion = WeightedBCELoss(pos_weight=10.0)
    loss = criterion(predictions, targets)
```

### sklearn с кастомной метрикой:

```
from sklearn.metrics import make_scorer

def custom_cost_score(y_true, y_pred):
    """Кастомная метрика с учётом стоимости"""
    tn = ((y_true == 0) & (y_pred == 0)).sum()
    fp = ((y_true == 0) & (y_pred == 1)).sum()
    fn = ((y_true == 1) & (y_pred == 0)).sum()
    tp = ((y_true == 1) & (y_pred == 1)).sum()

    # Стоимость ошибок
    cost = fn * 10 + fp * 1

    # Минимизируем стоимость (возвращаем -cost)
    return -cost
```

```
scorer = make_scorer(custom_cost_score)

# GridSearchCV с кастомной метрикой
grid = GridSearchCV(model, param_grid, scorer)
```

## ◆ 9. Метапридикат CostSensitiveClassifier

```
from sklearn.base import BaseEstimator, Clas

class CostSensitiveClassifier(BaseEstimator):
    def __init__(self, estimator, cost_matrix):
        self.estimator = estimator
        self.cost_matrix = cost_matrix

    def fit(self, X, y):
        self.estimator.fit(X, y)
        return self

    def predict_proba(self, X):
        return self.estimator.predict_proba(X)

    def predict(self, X):
        proba = self.predict_proba(X)

        # Ожидаемая стоимость для каждого к
        n_samples = X.shape[0]
        n_classes = len(self.cost_matrix)

        expected_costs = np.zeros((n_samples, n_classes))
        for i in range(n_classes):
            for j in range(n_classes):
                expected_costs[:, j] += (proba[:, i] * self.cost_matrix[i, j])

        # Предсказываем класс с минимальной
        return np.argmin(expected_costs, axis=1)

    # Использование
    cost_matrix = np.array([[0, 1], [10, 0]])
    clf = CostSensitiveClassifier(
        LogisticRegression(),
        cost_matrix
    )
```

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

## ◆ 10. Стратегии для многоклассовой классификации

### Веса для 3+ классов:

```
# Автоматические веса
from sklearn.utils.class_weight import compute_class_weight

classes = np.unique(y_train)
weights = compute_class_weight('balanced', classes)
class_weight_dict = dict(zip(classes, weights))

# Пример: {0: 0.5, 1: 2.0, 2: 4.0}

# Применение
model = RandomForestClassifier(class_weight=class_weight_dict)
model.fit(X_train, y_train)
```

### Матрица стоимости для многих классов:

```
# 3 класса: здоров (0), лёгкая болезнь (1),
cost_matrix = np.array([
    [0, 1, 5],   # 0: TN, ошибка как 1 (0)
    [1, 0, 3],   # 1: ошибка как 0 (1), Т
    [10, 5, 0]   # 2: ошибка как 0 (10),
])

# Самая дорогая ошибка: пропустить тяжёлую
```

## ◆ 11. Комбинирование со SMOTE

Веса классов + синтетические данные = двойной  
эффект!

```
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline

# Сначала SMOTE, потом model с весами
pipeline = Pipeline([
    ('smote', SMOTE(sampling_strategy=0.5)),
    ('classifier', LogisticRegression(
        class_weight={0: 1, 1: 3} # дополнительные веса
    ))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
```

### Когда использовать:

- **Только SMOTE:** мало данных minority класса (< 100)
- **Только веса:** достаточно данных, но разный дисбаланс
- **Оба:** сильный дисбаланс (1:100+) и важна точность

## ◆ 12. Мониторинг качества

### Метрики для несбалансированных данных:

| Метрика           | Когда использовать               | Оптимум |
|-------------------|----------------------------------|---------|
| F1-score          | Баланс precision/recall          | → 1     |
| F-beta            | Больший вес recall или precision | → 1     |
| Balanced Accuracy | Среднее между классами           | → 1     |
| MCC               | Универсальная метрика            | → 1     |
| ROC-AUC           | Общее качество ранжирования      | → 1     |

```
from sklearn.metrics import (
    f1_score, fbeta_score, balanced_accuracy_score,
    matthews_corrcoef, roc_auc_score
)

y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

print(f"F1: {f1_score(y_test, y_pred)}")
print(f"F2 (recall): {fbeta_score(y_test, y_proba, beta=2)}")
print(f"F0.5 (prec): {fbeta_score(y_test, y_proba, beta=0.5)}")
print(f"Bal Acc: {balanced_accuracy_score(y_test, y_pred)}")
print(f"MCC: {matthews_corrcoef(y_test, y_proba[:, 1])}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba[:, 1])}")
```

## ◆ 13. Сравнение подходов

| Подход           | Плюсы              | Минусы                  |
|------------------|--------------------|-------------------------|
| class_weight     | Просто, быстро     | Не всегда оптимально    |
| sample_weight    | Гибкость           | Нужно вычислять вручную |
| Threshold moving | Не меняет модель   | Требует валидации       |
| SMOTE            | Увеличивает данные | Может создать шум       |
| Focal Loss       | Фокус на сложных   | Только для DL           |
| Cost Matrix      | Точный контроль    | Сложнее настроить       |

## ◆ 14. Практические советы

### Для слабого дисбаланса (1:5 до 1:20):

- Используйте `class_weight='balanced'`
- Мониторьте F1-score
- SMOTE не обязателен

### Для сильного дисбаланса (1:20 до 1:100):

- Комбинируйте SMOTE + веса классов
- Используйте threshold moving
- Попробуйте Focal Loss для DL
- Мониторьте Balanced Accuracy и MCC

### Для экстремального дисбаланса (1:100+):

- Используйте anomaly detection подходы
- Cost-sensitive learning с матрицей стоимости
- Ensemble methods с разными весами
- Few-shot learning techniques

## ◆ 15. Чек-лист

- [ ] Проверить баланс классов  
(`value_counts()`)
- [ ] Определить стоимость ошибок (FN vs FP)
- [ ] Попробовать `class_weight='balanced'`
- [ ] Если не работает — подобрать веса вручную
- [ ] Рассмотреть threshold moving
- [ ] Для сильного дисбаланса — добавить SMOTE
- [ ] Использовать правильные метрики (F1, MCC, не Accuracy!)
- [ ] Проверить quality на обоих классах отдельно
- [ ] Визуализировать Precision-Recall кривую
- [ ] Валидировать на отложенной выборке

### 💡 Объяснение заказчику:

«Представьте аэропорт: пропустить террориста (ложноотрицательное решение) намного опаснее, чем лишний раз проверить невинного пассажира (ложноположительное). Cost-sensitive learning учит модель учитывать разную важность ошибок, делая систему безопаснее там, где это критично».



17 Январь 2026

## ◆ 1. Суть

- Полуконтролируемое обучение:** использует размеченные и неразмеченные данные
- Несколько представлений (views):** данные описаны разными наборами признаков
- Взаимное обучение:** модели обучаются друг друга на неразмеченных данных
- Цель:** улучшить качество при малом количестве размеченных примеров
- Ключевая идея:** разные представления дополняют друг друга

## ◆ 2. Основной алгоритм

### Шаги co-training:

1. Разделить признаки на два независимых набора (view1, view2)
2. Обучить две модели на размеченных данных
3. Каждая модель предсказывает метки для неразмеченных данных
4. Выбрать наиболее уверенные предсказания
5. Добавить их в обучающую выборку другой модели
6. Повторять до сходимости или исчерпания данных

## ◆ 3. Требования к представлениям

- Достаточность (Sufficiency):** каждое представление должно быть достаточным для обучения
- Условная независимость:** представления независимы при заданной метке класса
- Избыточность (Redundancy):** содержит перекрывающуюся информацию
- Баланс:** примерно одинаковая предсказательная способность

### Примеры разделения:

| Задача       | View 1         | View 2       |
|--------------|----------------|--------------|
| Веб-страницы | Текст страницы | Текст ссылок |
| Видео        | Изображения    | Аудио        |
| Email        | Тело письма    | Заголовок    |
| Документы    | Содержание     | Метаданные   |

## ◆ 4. Базовый код

```

import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

class CoTraining:
    def __init__(self, clf1, clf2, k=5,
max_iter=10):
        self.clf1 = clf1 # Модель для view 1
        self.clf2 = clf2 # Модель для view 2
        self.k = k # Примеров для
для добавления на итерацию
        self.max_iter = max_iter

    def fit(self, X1_labeled, X2_labeled,
y_labeled,
X1_unlabeled, X2_unlabeled):

        # Инициализация
        X1_l, X2_l, y_l = X1_labeled.copy(),
X2_labeled.copy(), y_labeled.copy()
        X1_u, X2_u = X1_unlabeled.copy(),
X2_unlabeled.copy()

        for iteration in range(self.max_iter):
            # Обучение на текущих размеченных
данных
            self.clf1.fit(X1_l, y_l)
            self.clf2.fit(X2_l, y_l)

            # Предсказания на неразмеченных
            proba1 = self.clf1.predict_proba(X1_u)
            proba2 = self.clf2.predict_proba(X2_u)

            # Выбор наиболее уверенных
            conf1 = np.max(proba1, axis=1)
            conf2 = np.max(proba2, axis=1)

            # Топ-k от каждой модели
            idx1 = np.argsort(conf1)[-self.k:]
            idx2 = np.argsort(conf2)[-self.k:]

            # Добавление в обучающую выборку
            X1_l = np.vstack([X1_l, X1_u[idx2]])
            X2_l = np.vstack([X2_l, X2_u[idx2]])
            y_l = np.hstack([y_l,
self.clf2.predict(X2_u[idx2])])

            X1_l = np.vstack([X1_l, X1_u[idx1]])
            X2_l = np.vstack([X2_l, X2_u[idx1]])
            y_l = np.hstack([y_l,
self.clf1.predict(X1_u[idx1])])

            # Удаление использованных
            неразмеченных

```

## Co-training Cheatsheet — 3 колонки

```

mask = np.ones(len(X1_u), dtype=bool)
mask[np.concatenate([idx1, idx2])] =
False

X1_u = X1_u[mask]
X2_u = X2_u[mask]

if len(X1_u) < self.k:
    break

# Финальное обучение
self.clf1.fit(X1_l, y_l)
self.clf2.fit(X2_l, y_l)

def predict(self, X1, X2):
    # Комбинирование предсказаний
    pred1 = self.clf1.predict_proba(X1)
    pred2 = self.clf2.predict_proba(X2)
    return np.argmax((pred1 + pred2) / 2,
axis=1)

```

## ◆ 5. Пример использования

```

# Генерация данных
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000,
n_features=20,
n_informative=15,
random_state=42)

# Разделение на два представления
X_view1 = X[:, :10]
X_view2 = X[:, 10:]

# Малая часть размеченных, остальное -
неразмеченное
labeled_size = 50
X1_labeled = X_view1[:labeled_size]
X2_labeled = X_view2[:labeled_size]
y_labeled = y[:labeled_size]

X1_unlabeled = X_view1[labeled_size:]
X2_unlabeled = X_view2[labeled_size:]

# Co-training
clf1 = GaussianNB()
clf2 = GaussianNB()

cotrainer = CoTraining(clf1, clf2, k=10,
max_iter=20)
cotrainer.fit(X1_labeled, X2_labeled, y_labeled,
X1_unlabeled, X2_unlabeled)

# Тестирование
# Используем оставшиеся данные для теста
from sklearn.metrics import accuracy_score
y_pred = cotrainer.predict(X_view1[labeled_size:],
X_view2[labeled_size:])
print(f"Accuracy:
{accuracy_score(y[labeled_size:], y_pred)}")

```

## ◆ 6. Варианты и расширения

### Democratic Co-Learning:

- Несколько классификаторов голосуют
- Добавляются примеры с консенсусом

### Tri-training:

- Три классификатора
- Два согласных обучаются третьего

### Co-EM:

- Вероятностные метки вместо жёстких
- EM-алгоритм для уточнения

### Self-training с разделением:

- Одна модель, разные входы
- Упрощённый вариант co-training

## ◆ 7. Проблемы и решения

| Проблема                    | Решение                        |
|-----------------------------|--------------------------------|
| Ошибки накапливаются        | Порог уверенности, валидация   |
| Представления не независимы | PCA, случайное разбиение       |
| Дисбаланс классов           | Балансированный выбор примеров |
| Модели расходятся           | Ограничить количество итераций |
| Нет естественных view       | Случайное разбиение признаков  |

## ◆ 8. Стратегии выбора примеров

- **Максимальная уверенность:** выбирать с max  $P(y|x)$
- **Margin sampling:** разница между топ-2 классами
- **Entropy:** наименьшая энтропия предсказаний
- **Сбалансированный выбор:** равное число из каждого класса
- **Динамический порог:** адаптивный выбор по итерациям

```
# Пример с margin sampling
def select_by_margin(proba, k):
    """Выбрать k примеров с наибольшим margin"""
    sorted_proba = np.sort(proba, axis=1)
    margins = sorted_proba[:, -1] - sorted_proba[:, -2]
    return np.argsort(margins)[-k:]

# Пример с entropy
def select_by_entropy(proba, k):
    """Выбрать k примеров с наименьшей
    энтропией"""
    entropy = -np.sum(proba * np.log(proba + 1e-10), axis=1)
    return np.argsort(entropy)[:k]
```

## ◆ 9. Метрики и валидация

- **Качество на тесте:** основная метрика
- **Согласованность моделей:** % совпадений предсказаний
- **Динамика обучения:** качество по итерациям
- **Размер псевдо-размечённых:** сколько добавлено примеров

```
def evaluate_cotraining(cotrainer, X1_test,
X2_test, y_test):
    # Предсказания каждой модели
    pred1 = cotrainer.clf1.predict(X1_test)
    pred2 = cotrainer.clf2.predict(X2_test)
    pred_combined = cotrainer.predict(X1_test,
X2_test)

    # Метрики
    from sklearn.metrics import accuracy_score
    acc1 = accuracy_score(y_test, pred1)
    acc2 = accuracy_score(y_test, pred2)
    acc_comb = accuracy_score(y_test,
pred_combined)

    # Согласованность
    agreement = np.mean(pred1 == pred2)

    return {
        'acc_view1': acc1,
        'acc_view2': acc2,
        'acc_combined': acc_comb,
        'agreement': agreement
    }
```

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Мало размеченных данных
- ✓ Много неразмеченных данных
- ✓ Естественные view (текст+метаданные)
- ✓ Разметка дорогая
- ✓ Мультимодальные данные

### ✗ Плохо

- ✗ Достаточно размеченных данных
- ✗ Невозможно разделить признаки
- ✗ Представления сильно коррелированы
- ✗ Нужна высокая точность

## ◆ 11. Продвинутые техники

### Co-training с глубоким обучением:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def create_view_network(input_dim):
    model = Sequential([
        Dense(64, activation='relu',
              input_dim=input_dim),
        Dense(32, activation='relu'),
        Dense(2, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Две нейросети для разных view
model1 = create_view_network(X_view1.shape[1])
model2 = create_view_network(X_view2.shape[1])

# Co-training цикл
for epoch in range(10):
    model1.fit(X1_labeled, y_labeled, epochs=1,
               verbose=0)
    model2.fit(X2_labeled, y_labeled, epochs=1,
               verbose=0)

    # Псевдо-метки с высокой уверенностью
    proba1 = model1.predict(X1_unlabeled)
    conf_mask = np.max(proba1, axis=1) > 0.95

    if conf_mask.sum() > 0:
        X2_labeled = np.vstack([X2_labeled,
                               X2_unlabeled[conf_mask]])
        y_labeled = np.hstack([y_labeled,
                              np.argmax(proba1[conf_mask], axis=1)])
```

## ◆ 12. Сравнение с другими методами

| Метод               | Особенности | Когда лучше       |
|---------------------|-------------|-------------------|
| Co-training         | Два view    | Естественные view |
| Self-training       | Один view   | Нет разделения    |
| Label propagation   | Граф        | Похожие объекты   |
| Semi-supervised GAN | Генератор   | Изображения       |

## ◆ 13. Чек-лист

- [ ] Проверить достаточность каждого view
- [ ] Убедиться в условной независимости
- [ ] Выбрать стратегию отбора примеров
- [ ] Установить порог уверенности
- [ ] Ограничить число итераций
- [ ] Мониторить качество на валидации
- [ ] Проверить согласованность моделей
- [ ] Сравнить с baseline (только размеченные данные)

### 💡 Объяснение заказчику:

«Co-training — это когда две модели обучаются друг друга. Представьте двух учителей: один знает математику, другой — физику. Они по очереди решают задачи и помогают друг другу. Так можно обучаться с меньшим количеством готовых решений, используя неразмеченные данные».

# Co-training (Со-обучение)

## 1. Основная идея

**Co-training** — метод полуконтролируемого обучения, использующий два различных "взгляда" (views) на данные для взаимного обучения классификаторов

### Предположения:

- **Sufficiency**: каждый view достаточен для классификации
- **Independence**: views условно независимы при классе
- **Compatibility**: предсказания согласуются на неразмеченных данных

**Пример:** классификация веб-страниц по содержимому текста (view 1) и гиперссылкам (view 2)

## 2. Алгоритм Co-training

Вход:  $L$  (labeled),  $U$  (unlabeled)

Инициализация: обучить  $h_1, h_2$  на  $L$

Повторять:

1.  $h_1$  предсказывает метки для  $U$   
Выбрать  $k$  самых уверенных примеров  
Добавить в  $L_1$
2.  $h_2$  предсказывает метки для  $U$   
Выбрать  $k$  самых уверенных примеров  
Добавить в  $L_2$
3. Переобучить  $h_1$  на  $L \cup L_2$   
Переобучить  $h_2$  на  $L \cup L_1$
4. Удалить  $L_1, L_2$  из  $U$

Пока не сойдётся или  $U$  пусто

**k** — гиперпараметр числа добавляемых примеров (обычно 1-10)

## 3. Multi-view данные

### Естественные views:

- **Текст + изображения**: статьи с фото
- **Audio + video**: видеозаписи
- **Текст + ссылки**: веб-страницы
- **Контент + метаданные**: документы

**Искусственные views:** если естественных нет, можно создать

- Разделить признаки случайно
- Использовать разные типы признаков
- Разные архитектуры моделей
- Разные preprocessing

**Требование:** views должны предоставлять дополнительную информацию друг другу

## 4. Теоретические гарантии

**PAC-learning bounds:** при выполнении условий sufficiency и independence, co-training сходится к оптимальному классификатору

**Ключевое свойство:** даже если один view делает ошибку, другой может её исправить

**Error reduction:**

$$\varepsilon(h_1, h_2) \leq \varepsilon(h_1) \cdot \varepsilon(h_2) / (1 - \varepsilon(h_1) - \varepsilon(h_2))$$

**Условия успеха:**

- Views действительно независимы
- Каждый view информативен
- Достаточно labeled данных для начала

## 5. Выбор confident примеров

**Стратегии выбора:**

- **Максимальная вероятность:**  $\max P(y|x)$
- **Margin:**  $P(y_1|x) - P(y_2|x)$  (разница топ-2)
- **Entropy:** низкая энтропия предсказания
- **Agreement:** оба view согласны

**Балансировка классов:** выбирать равное число примеров из каждого класса для избежания class imbalance

**Пороги confidence:** использовать адаптивные пороги вместо фиксированного k

## 6. Варианты Co-training

**Democratic Co-learning:** несколько ( $>2$ ) learners голосуют

**Co-EM:** использует EM-алгоритм вместо жёсткой маркировки

E-step: вычислить  $P(y|x)$  для каждого view

M-step: обновить параметры, взвешивая по  $P(y|x)$

**Tri-training:** три классификатора, два обучают третий

**Multi-view learning:** обобщение на  $>2$  views

**Self-training** — специальный случай с одним view

## 7. Применения в ML

**NLP:**

- Классификация текстов (content + metadata)
- Named Entity Recognition
- Sentiment analysis (текст разных частей документа)

**Computer Vision:**

- Классификация изображений (разные модальности)
- Object detection (appearance + context)
- Video analysis (spatial + temporal views)

**Web Mining:**

- Классификация веб-страниц
- Email classification
- Social network analysis

## 8. Реализация на Python

```

import numpy as np
from sklearn.naive_bayes import
MultinomialNB

class CoTraining:
    def __init__(self, clf1, clf2,
k=10):
        self.clf1 = clf1
        self.clf2 = clf2
        self.k = k

        def fit(self, x1_labeled,
x2_labeled, y_labeled,
X1_unlabeled, X2_unlabeled,
n_iter=100):

            for iteration in range(n_iter):
                # Обучить классификаторы
                self.clf1.fit(x1_labeled,
y_labeled)
                self.clf2.fit(x2_labeled,
y_labeled)

                # Предсказать для unlabeled
                proba1 =
self.clf1.predict_proba(X1_unlabeled)
                proba2 =
self.clf2.predict_proba(X2_unlabeled)

                # Выбрать самые уверенные
                conf1 = np.max(proba1,
axis=1)
                conf2 = np.max(proba2,
axis=1)

                # Топ-k от каждого
                classifier
                idx1 = np.argsort(conf1)[-self.k:]
                idx2 = np.argsort(conf2)[-self.k:]

                # Добавить в labeled
                X1_labeled =
np.vstack([x1_labeled,
X1_unlabeled[idx2]])
                X2_labeled =

```

## Co-training Cheatsheet — 3 колонки

```

np.vstack([X2_labeled,
X2_unlabeled[idx1]])

y1_new =
self.clf1.predict(X1_unlabeled[idx2])
y2_new =
self.clf2.predict(X2_unlabeled[idx1])

y_labeled =
np.concatenate([y_labeled, y2_new,
y1_new])

# Удалить из unlabeled
mask =
np.ones(len(X1_unlabeled), dtype=bool)
mask[np.concatenate([idx1,
idx2])] = False
X1_unlabeled =
X1_unlabeled[mask]
X2_unlabeled =
X2_unlabeled[mask]

if len(X1_unlabeled) == 0:
    break

return self

```

## 9. Проблемы и решения

**Error propagation:** неправильно размеченные примеры ухудшают качество

### Решения:

- Строгие пороги confidence
- Validation set для мониторинга
- Периодическое удаление шумных меток

**View agreement:** если views слишком похожи, co-training неэффективен

**Решение:** проверять независимость views, использовать regularization для их разделения

**Class imbalance:** один класс может доминировать в confident примерах

**Решение:** балансировка при выборе примеров

## 10. Сравнение с другими методами

**Self-training:** один view, может усиливать ошибки

**Co-training:** два views, взаимная коррекция ошибок

**Label propagation:** использует graph structure, не требует views

**Semi-supervised SVM:** прямая оптимизация на unlabeled

### Когда использовать co-training:

- Есть естественные multiple views
- Views условно независимы
- Мало labeled, много unlabeled данных

## 11. Deep Co-training

**Идея:** использовать нейронные сети в качестве views

**Подходы:**

- **Разные архитектуры:** CNN + RNN
- **Разные слои:** early layers + late layers
- **Разные модальности:** vision + text
- **Adversarial training:** генерация diverse views

**Deep Co-training алгоритм:**

1. Обучить две нейросети на labeled
2. Для unlabeled batch:
  - Получить предсказания обеих сетей
  - Выбрать согласованные high-confidence
  - Обучить каждую сеть на примерах другой
3. Повторять

## 12. Best Practices

**Начальное обучение:** убедиться, что оба view-specific классификатора работают разумно на labeled данных

**Размер k:**

- Малый k (1-5): консервативный, меньше ошибок
- Большой k (>10): быстрее использует unlabeled, но рискованнее

**Мониторинг:**

- Отслеживать agreement между views
- Validation accuracy обоих классификаторов
- Распределение confidence scores

**Остановка:** когда agreement падает или нет confident примеров

# Кредитный скринг

 5 января 2026

## ◆ 1. Суть кредитного скринга

- **Цель:** оценка вероятности дефолта (невозврата кредита)
- **Задача:** бинарная классификация (вернет/не вернет)
- **Скринговый балл:** числовая оценка кредитоспособности
- **Регуляторные требования:** прозрачность, объяснимость решений
- **Базель II/III:** международные стандарты оценки рисков

## ◆ 2. Основные типы скринга

| Тип                 | Назначение                | Момент применения               |
|---------------------|---------------------------|---------------------------------|
| Application scoring | Оценка заявки             | До выдачи кредита               |
| Behavioral scoring  | Поведение клиента         | В процессе кредита              |
| Collection scoring  | Работа с просрочкой       | При возникновении задолженности |
| Fraud scoring       | Обнаружение мошенничества | На всех этапах                  |

## ◆ 3. Основные признаки

### Демографические:

- Возраст, пол, семейное положение
- Образование, место работы
- Регион проживания

### Финансовые:

- Доход (подтвержденный/неподтвержденный)
- Наличие активов (недвижимость, авто)
- Долговая нагрузка (DTI - Debt-to-Income)
- История занятости

### Кредитная история:

- Количество действующих кредитов
- История платежей (DPD - Days Past Due)
- Количество запросов в БКИ
- Максимальная просрочка

## ◆ 4. Базовый код

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

# Логистическая регрессия (классика)
lr_model = LogisticRegression(
    penalty='l2',
    C=1.0,
    class_weight='balanced',
    random_state=42
)
lr_model.fit(X_train, y_train)

# Градиентный бустинг (современный подход)
from lightgbm import LGBMClassifier
gbm_model = LGBMClassifier(
    n_estimators=100,
    learning_rate=0.05,
    max_depth=5,
    scale_pos_weight=5 # для дисбаланса
)
gbm_model.fit(X_train, y_train)

# Предсказание вероятностей
y_proba = gbm_model.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_proba)
```

## ◆ 5. Популярные модели

| Модель                  | Плюсы                                       | Минусы                            |
|-------------------------|---------------------------------------------|-----------------------------------|
| Логистическая регрессия | Интерпретируемость, соответствие регуляциям | Только линейные зависимости       |
| Scorecard               | Максимальная прозрачность                   | Ручная настройка                  |
| Random Forest           | Нелинейные зависимости                      | Сложно объяснить                  |
| XGBoost/LightGBM        | Высокая точность                            | Требует объяснения для регулятора |
| Нейросети               | Максимальная гибкость                       | Черный ящик, редко используются   |

## ◆ 6. Создание скоркарты (Scorecard)

### Классический подход Weight of Evidence (WoE):

```
import scorecardpy as sc

# Разбиение на бины
bins = sc.woebin(df, y='default')

# Преобразование в WoE
train_woe = sc.woebin_ply(df_train, bins)

# Обучение логистической регрессии
lr = LogisticRegression()
lr.fit(train_woe, y_train)

# Создание скоркарты
card = sc.scorecard(bins, lr, X_train.columns)

# Расчет скорингового балла
scores = sc.scorecard_ply(df_test, card)
```

### WoE (Weight of Evidence):

- WoE =  $\ln(\% \text{ good} / \% \text{ bad})$
- Положительный WoE → меньше риска
- Отрицательный WoE → больше риска

## ◆ 7. Метрики качества

| Метрика                          | Описание                                                 | Типичное значение                     |
|----------------------------------|----------------------------------------------------------|---------------------------------------|
| ROC-AUC                          | Общее качество модели                                    | >0.7 (хорошо), >0.8 (отлично)         |
| Gini                             | $2 \cdot \text{AUC} - 1$                                 | >0.4 (хорошо), >0.6 (отлично)         |
| KS<br>(Kolmogorov-Smirnov)       | Максимальная разница между кумулятивными распределениями | >0.3 (хорошо), >0.4 (отлично)         |
| PSI (Population Stability Index) | Стабильность популяции                                   | <0.1 (стабильна), >0.25 (нестабильна) |
| Precision/Recall                 | Точность/полнота                                         | Зависит от бизнес-задачи              |

## ◆ 8. Обработка дисбаланса классов

**Проблема:** дефолтов обычно 3-10% от всех кредитов

### Решения:

- **Class weights:** `class_weight='balanced'`
- **Oversampling:** SMOTE, ADASYN
- **Undersampling:** RandomUnderSampler
- **Ансамбли:** BalancedRandomForest
- **Порог классификации:** подбор оптимального threshold

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled =
smote.fit_resample(X_train, y_train)
```

## ◆ 9. Feature Engineering

### Ключевые признаки:

- **DTI (Debt-to-Income):** платеж по кредиту / доход
- **LTV (Loan-to-Value):** сумма кредита / стоимость залога
- **Утилизация кредита:** использованный лимит / общий лимит
- **Возраст кредитной истории:** месяцев с первого кредита
- **Количество просрочек:** за последние 6/12/24 месяца

```
# Создание производных признаков
df['dti'] = df['monthly_payment'] / df['income']
df['credit_history_months'] = (
    pd.to_datetime('today') -
    df['first_credit_date']
).dt.days / 30
df['utilization'] = df['used_credit'] /
df['credit_limit']
```

## ◆ 10. Проблемы и решения

| Проблема                     | Решение                                        |
|------------------------------|------------------------------------------------|
| Дрейф данных                 | Регулярный мониторинг PSI, переобучение модели |
| Отсутствие кредитной истории | Альтернативные данные (телеом, соцсети)        |
| Мошенничество                | Отдельная модель fraud detection               |
| Регуляторные требования      | Использование интерпретируемых моделей         |
| Сезонность                   | Признаки времени года, праздников              |

## ◆ 11. Интерпретация модели

### Регуляторы требуют объяснения решений

#### Методы:

- **SHAP:** вклад каждого признака в решение
- **LIME:** локальное объяснение
- **Feature Importance:** важность признаков
- **Partial Dependence:** влияние признака на прогноз

```
import shap

# Объяснение модели SHAP
explainer = shap.TreeExplainer(gbm_model)
shap_values = explainer.shap_values(X_test)

# Визуализация
shap.summary_plot(shap_values, X_test)
```

## ◆ 12. Валидация модели

### Time-based split: важно для временных рядов

```
# Разделение по времени
train_end = '2024-01-01'
val_end = '2024-07-01'

train = df[df['date'] < train_end]
val = df[(df['date'] >= train_end) & (df['date'] <
val_end)]
test = df[df['date'] >= val_end]
```

### Out-of-time validation: проверка на будущих периодах

### Back-testing: ретроспективная проверка на исторических данных

## ◆ 13. Чек-лист для продакшна

- [ ] Модель соответствует регуляторным требованиям
- [ ] Документация всех признаков и их источников
- [ ] Мониторинг PSI для отслеживания дрейфа
- [ ] А/В тестирование новой модели
- [ ] План переобучения модели (обычно раз в квартал)
- [ ] Процедура объяснения отказов клиентам
- [ ] Защита от мошенничества
- [ ] Обработка edge cases (очень высокий/ низкий доход)

## ◆ 14. Когда использовать разные подходы

### ✓ Скоркарта (Scorecard)

- ✓ Требования регулятора к прозрачности
- ✓ Небольшие банки, простые продукты
- ✓ Нужно объяснять решения клиентам
- ✓ Ограниченные данные

### ✓ Gradient Boosting

- ✓ Крупные банки с большими данными
- ✓ Есть команда для поддержки сложных моделей
- ✓ Высокая конкуренция, нужна максимальная точность
- ✓ Есть инструменты для объяснения (SHAP)

## ◆ 15. Практические советы

- **Начните с простого:** логистическая регрессия как baseline
- **Бизнес-метрики важнее технических:** считайте прибыль и потери
- **Мониторинг критичен:** модель деградирует со временем
- **Альтернативные данные:** телеком, e-commerce, соцсети
- **Reject inference:** учитывайте отклоненных клиентов
- **Champion-Challenger:** постоянное тестирование новых моделей

 **Объяснение заказчику:** «Мы анализируем тысячи кредитных историй, чтобы понять, кто с большей вероятностью вернет кредит. Каждый фактор — возраст, доход, история — получает свой вес, и модель выдает финальную оценку кредитоспособности».



# CRF (Conditional Random Fields)

 Январь 2026

## ◆ 1. Суть

- **CRF:** Conditional Random Fields
- **Задача:** последовательная разметка (NER, POS tagging)
- **Подход:** условные вероятности с учетом контекста
- **Преимущество:** учитывает зависимости между метками
- **Применение:** NLP, биоинформатика, CV

CRF моделируют  $P(Y|X)$  для последовательностей, учитывая глобальный контекст

## ◆ 2. Формула

$$P(y|x) = (1/Z(x)) \exp(\sum_i \sum_k \lambda_k f_k(y_{i-1}, y_i, x, i))$$

где:

- $y$  - последовательность меток
- $x$  - входная последовательность
- $f_k$  - признаковые функции
- $\lambda_k$  - веса признаков
- $Z(x)$  - нормализующая константа

### Интуиция:

- Вычисляет вероятность всей последовательности меток
- Учитывает переходы между метками
- Использует признаки из всей последовательности

### ◆ 3. Sklearn-crfsuite

```

import sklearn_crfsuite
from sklearn_crfsuite import metrics

# Извлечение признаков для слова
def word2features(sent, i):
    word = sent[i][0]
    postag = sent[i][1]

    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word[-3:)': word[-3:],
        'word[-2:)': word[-2:],
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
        'postag': postag,
    }

    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:postag': postag1,
        })
    else:
        features['BOS'] = True

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:postag': postag1,
        })
    else:
        features['EOS'] = True

    return features

# Создание модели
crf = sklearn_crfsuite.CRF(
    algorithm='lbgfsg',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)

# Обучение
crf.fit(X_train, y_train)

```

```

# Предсказание
y_pred = crf.predict(X_test)

```

### ◆ 5. NER пример

```

# Данные в формате IOB
sent = [
    ('John', 'NNP', 'B-PER'),
    ('Smith', 'NNP', 'I-PER'),
    ('works', 'VBZ', 'O'),
    ('at', 'IN', 'O'),
    ('Google', 'NNP', 'B-ORG')
]

# Извлечение признаков
X = [[word2features(sent, i) for i in range(len(sent))]]

# Метки
y = [[label for _, _, label in sent]]

# Обучение
crf.fit(X, y)

# Предсказание для нового предложения
new_sent = [('Apple', 'NNP'), ('released', 'VBD'),
            ('iPhone', 'NNP')]
X_new = [[word2features(new_sent, i) for i in range(len(new_sent))]]
predictions = crf.predict(X_new)
print(predictions) # [['B-ORG', 'O', 'B-PRODUCT']]

```

### ◆ 4. Признаки

#### Типы признаков:

- Слово:** lowercase, суффиксы, префиксы
- Форма:** регистр, цифры, пунктуация
- Контекст:** окружающие слова (окно ±2)
- POS теги:** части речи
- Словари:** gazetteer признаки
- Кастомные:** domain-specific

```

# Пример признаков
features = {
    'word.lower': 'john',
    'word.isupper': False,
    'word.istitle': True,
    'word[-3:)': 'ohn',
    'postag': 'NNP',
    '-1:word.lower': 'mr',
    '+1:word.lower': 'smith'
}

```

## ◆ 6. Оценка

```
from sklearn_crfsuite import metrics

# Метрики
labels = list(crf.classes_)
labels.remove('0') # Исключаем класс '0'

# F1-score
f1 = metrics.flat_f1_score(y_test, y_pred,
                            average='weighted',
                            labels=labels)
print(f'F1: {f1:.3f}')

# Precision и Recall по классам
print(metrics.flat_classification_report(
    y_test, y_pred, labels=labels))

# Пример вывода:
#              precision    recall  f1-score
support
#
#          B-LOC      0.81      0.78      0.79
1084
#          I-LOC      0.76      0.73      0.74
325
#          B-MISC     0.73      0.49      0.59
339
#          I-MISC     0.70      0.53      0.60
557
#          B-ORG      0.79      0.82      0.80
1400
#          I-ORG      0.83      0.83      0.83
1104
#          B-PER      0.84      0.90      0.87
735
#          I-PER      0.90      0.93      0.92
634
```

## ◆ 7. CRF vs HMM

| Аспект     | CRF                    | HMM                       |
|------------|------------------------|---------------------------|
| Тип модели | Дискриминативная       | Генеративная              |
| Моделирует | $P(Y X)$               | $P(X,Y)$                  |
| Признаки   | Любые, перекрывающиеся | Ограниченные, независимые |
| Точность   | Выше                   | Ниже                      |
| Скорость   | Медленнее              | Быстрее                   |
| Label bias | Нет                    | Есть                      |

## ◆ 8. Гиперпараметры

- `c1` : L1 регуляризация (0.0-1.0)
  - Больше → sparser weights
- `c2` : L2 регуляризация (0.0-1.0)
  - Больше → smoother weights
- `algorithm` : 'lbgfs' (основной), 'l2sgd', 'ap'
- `max_iterations` : 100-500
- `all_possible_transitions` : True (по умолчанию)

```
# Grid search для оптимизации
from sklearn.model_selection import
RandomizedSearchCV

params_space = {
    'c1': [0.01, 0.05, 0.1, 0.5, 1.0],
    'c2': [0.01, 0.05, 0.1, 0.5, 1.0],
}

rs = RandomizedSearchCV(crf, params_space, cv=3,
n_iter=20)
rs.fit(X_train, y_train)
print(f'Best params: {rs.best_params_}')
```

## ◆ 9. Применения

- **Named Entity Recognition (NER)**
  - Извлечение имен, организаций, локаций
- **Part-of-Speech (POS) Tagging**
  - Определение частей речи
- **Chunking**
  - Выделение фраз в тексте
- **Gene Finding**
  - Биоинформатика: поиск генов в ДНК
- **OCR Post-processing**
  - Исправление ошибок распознавания
- **Information Extraction**
  - Извлечение структурированной информации

## ◆ 10. Лучшие практики

- **Контекстное окно:** используйте ±2 слова
- **POS теги:** добавляйте как признаки
- **Словари:** gazetteer lists для domain
- **Регуляризация:** начните с `c1=c2=0.1`
- **Cross-validation:** для выбора параметров
- **IOB формат:** используйте для multi-token entities
- **Баланс классов:** учитывайте дисбаланс

# Cross-modal Retrieval

 Январь 2026

## ◆ 1. Суть

- Задача:** поиск между разными модальностями
- Примеры:** текст → изображение, аудио → видео, скетч → фото
- Ключ:** общее embedding пространство
- Цель:** схожие объекты близки, разные — далеко

## ◆ 2. Типы задач

| Тип          | Описание                         | Пример          |
|--------------|----------------------------------|-----------------|
| Image-Text   | Поиск изображений по тексту      | Google Images   |
| Audio-Visual | Поиск видео по звуку             | Поиск по музыке |
| Sketch-Photo | Поиск фото по наброску           | Дизайн поиск    |
| Text-Video   | Поиск видео по описанию          | YouTube search  |
| 3D-2D        | Поиск 3D моделей по изображениям | CAD поиск       |

## ◆ 3. Архитектура

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class CrossModalRetrieval(nn.Module):
    def __init__(self, embedding_dim=512):
        super().__init__()
        # Encoder для модальности А
        self.encoder_a = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, embedding_dim)
        )
        # Encoder для модальности В
        self.encoder_b = nn.Sequential(
            nn.Linear(768, 512),
            nn.ReLU(),
            nn.Linear(512, embedding_dim)
        )

    def forward(self, modal_a, modal_b):
        emb_a =
F.normalize(self.encoder_a(modal_a))
        emb_b =
F.normalize(self.encoder_b(modal_b))
        return emb_a, emb_b

    def similarity(self, emb_a, emb_b):
        return emb_a @ emb_b.T # Cosine
similarity
```

## ◆ 4. Функции потерь

### Contrastive Loss

```
def contrastive_loss(emb_a, emb_b,
temperature=0.07):
    # Нормализация
    emb_a = F.normalize(emb_a, dim=-1)
    emb_b = F.normalize(emb_b, dim=-1)

    # Similarity matrix
    sim_matrix = (emb_a @ emb_b.T) / temperature

    # Labels (диагональ - позитивные пары)
    labels = torch.arange(len(emb_a))

    # Symmetric loss
    loss_a2b = F.cross_entropy(sim_matrix, labels)
    loss_b2a = F.cross_entropy(sim_matrix.T,
    labels)

    return (loss_a2b + loss_b2a) / 2
```

### Triplet Loss

```
def triplet_loss(anchor, positive, negative,
margin=0.2):
    pos_dist = F.pairwise_distance(anchor,
positive)
    neg_dist = F.pairwise_distance(anchor,
negative)
    loss = F.relu(pos_dist - neg_dist + margin)
    return loss.mean()
```

## ◆ 5. Hard Negative Mining

**Проблема:** не все негативные примеры полезны

**Решение:** выбирать сложные негативы

```
def hard_negative_mining(anchor, positives,
negatives, k=5):
    """Выбор k самых сложных негативов"""
    # Расстояния до всех негативов
    neg_dists = torch.cdist(anchor, negatives)

    # Берем k ближайших (сложных) негативов
    hard_neg_idx = neg_dists.topk(k,
largest=False).indices
    hard_negatives = negatives[hard_neg_idx]

    return hard_negatives

# Использование в обучении
for batch in dataloader:
    anchor, pos, all_negs = batch
    hard_negs = hard_negative_mining(anchor, pos,
all_negs)
    loss = triplet_loss(anchor, pos, hard_negs)
```

## ◆ 6. Метрики качества

| Метрика                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Описание                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| Recall@K                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Доля правильных в топ-K результатов               |
| MRR                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Mean Reciprocal Rank (средний обратный ранг)      |
| MAP                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Mean Average Precision                            |
| NDCG@K                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Normalized Discounted Cumulative Gain             |
| mAP@R                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Mean Average Precision at R (по всем релевантным) |
| <pre>def recall_at_k(similarities, k=5):     """Recall@K для batch"""     # similarities: [batch, batch] матрица     схожести     batch_size = similarities.size(0)     # Правильные ответы на диагонали     correct = torch.arange(batch_size)     # Top-k индексы     top_k = similarities.topk(k, dim=1).indices     # Проверка наличия правильного ответа     hits = (top_k == correct.unsqueeze(1)).any(dim=1)     return hits.float().mean().item()</pre> |                                                   |

## ◆ 7. CLIP-style retrieval

```
from transformers import CLIPModel, CLIPProcessor

# Загрузка CLIP
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Индексация базы изображений
def index_images(image_paths):
    embeddings = []
    for path in image_paths:
        image = Image.open(path)
        inputs = processor(images=image,
return_tensors="pt")
        with torch.no_grad():
            image_emb =
model.get_image_features(**inputs)
        embeddings.append(image_emb)
    return torch.cat(embeddings)

# Поиск по текстовому запросу
def search(query_text, image_embeddings, k=10):
    inputs = processor(text=[query_text],
return_tensors="pt")
    with torch.no_grad():
        text_emb =
model.get_text_features(**inputs)

    # Косинусная схожесть
    similarities = F.cosine_similarity(text_emb,
image_embeddings)
    top_k = similarities.topk(k)

    return top_k.indices, top_k.values
```

## ◆ 8. Масштабирование поиска

**Проблема:** поиск по миллионам объектов медленный

**Решения:**

- **FAISS:** библиотека от Facebook для быстрого поиска
- **Annoy:** approximate nearest neighbors
- **ScaNN:** от Google
- **HNSW:** hierarchical navigable small world

```
import faiss
import numpy as np

# Создание FAISS индекса
d = 512 # размерность эмбеддингов
index = faiss.IndexFlatIP(d) # Inner Product (cosine для normalized)

# Добавление эмбеддингов
embeddings_np = embeddings.cpu().numpy()
embeddings_np = embeddings_np / np.linalg.norm(embeddings_np, axis=1,
keepdims=True)
index.add(embeddings_np)

# Поиск
query_emb = query_emb.cpu().numpy()
query_emb = query_emb / np.linalg.norm(query_emb)
D, I = index.search(query_emb, k=10) # D - distances, I - indices
```

## ◆ 9. Multi-modal fusion

**Объединение нескольких модальностей для запроса**

```
class MultiModalRetrieval(nn.Module):
    def __init__(self):
        super().__init__()
        self.text_encoder = TextEncoder()
        self.image_encoder = ImageEncoder()
        self.audio_encoder = AudioEncoder()
        # Fusion layer
        self.fusion = nn.Linear(512*3, 512)

    def encode_query(self, text=None, image=None, audio=None):
        embeds = []
        if text is not None:
            embeds.append(self.text_encoder(text))
        if image is not None:
            embeds.append(self.image_encoder(image))
        if audio is not None:
            embeds.append(self.audio_encoder(audio))

        # Concatenate и проецируем
        fused = torch.cat(embeds, dim=-1)
        return self.fusion(fused)

    # Поиск с текстом и изображением
    query_emb = model.encode_query(
        text="red sports car",
        image=sketch_image
    )
```

## ◆ 10. Query expansion

**Улучшение запроса через pseudo-relevance feedback**

```
def query_expansion(query_emb, index, alpha=0.5, top_k=5):
    """
    Расширение запроса через топ-K результатов
    """
    # Первичный поиск
    D, I = index.search(query_emb, k=top_k)

    # Получаем эмбеддинги топ-K
    top_embeddings = index.reconstruct_batch(I[0])

    # Усредняем с исходным запросом
    expanded_query = (1-alpha) * query_emb + alpha * top_embeddings.mean(axis=0, keepdims=True)

    # Нормализуем
    expanded_query = expanded_query / np.linalg.norm(expanded_query)

    # Повторный поиск
    D, I = index.search(expanded_query, k=100)
    return I
```

## ◆ 11. Практические советы

### ✓ Рекомендуется

- ✓ Pre-trained энкодеры (CLIP, ALIGN)
- ✓ L2 нормализация эмбеддингов
- ✓ Hard negative mining
- ✓ Data augmentation для обеих модальностей
- ✓ FAISS для масштабирования
- ✓ Temperature scaling в contrastive loss

### ✗ Избегать

- ✗ Обучение без pre-training
- ✗ Маленький batch size (<128)
- ✗ Игнорирование качества пар данных
- ✗ Euclidean distance без нормализации
- ✗ Random negatives в triplet loss

## ◆ 12. Датасеты

| Датасет             | Модальности  | Размер           |
|---------------------|--------------|------------------|
| MSCOCO              | Image-Text   | 330К изображений |
| Flickr30K           | Image-Text   | 31К изображений  |
| Conceptual Captions | Image-Text   | 3.3М пар         |
| AudioCaps           | Audio-Text   | 50К аудио        |
| MSR-VTT             | Video-Text   | 10К видео        |
| Sketchy             | Sketch-Photo | 75К фото         |

## ◆ 13. Re-ranking стратегии

### Улучшение результатов после первичного поиска

- Reciprocal Rank Fusion:** объединение результатов нескольких методов
- Query expansion:** расширение запроса
- Cross-modal verification:** проверка соответствия
- Learned re-ranking:** обучаемая модель для пере-ранжирования

## ◆ 14. Применения

- E-commerce:** поиск товаров по изображению или описанию
- Медиа:** поиск видео, музыки, изображений
- Дизайн:** поиск референсов по наброскам
- Медицина:** поиск похожих случаев (изображения + текст)
- Образование:** поиск учебных материалов
- Безопасность:** поиск подозрительного контента

### 💡 Объяснение заказчику:

«Система позволяет искать, используя разные типы запросов. Например, найти фотографии по текстовому описанию, или найти видео по звуку. Модель учится понимать смысл в разных форматах и находить похожее, даже если запрос в одном формате, а результаты — в другом».

## ◆ 15. Чек-лист

- [ ] Определить пары модальностей
- [ ] Подготовить aligned датасет
- [ ] Выбрать pre-trained энкодеры
- [ ] Настроить contrastive/triplet loss
- [ ] Реализовать hard negative mining
- [ ] Настроить метрики (Recall@K, MRR)
- [ ] Создать FAISS индекс для масштабирования
- [ ] Оптимизировать inference (batch processing)
- [ ] Добавить re-ranking стратегии

 Кросс-валидация

 17 Январь 2026

## ◆ 1. Суть

- Цель:** надёжная оценка модели
- Метод:** многократное разбиение данных
- Преимущество:** использование всех данных
- Результат:** среднее качество по всем фолдам

## ◆ 2. K-Fold Cross-Validation

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

model = RandomForestClassifier(random_state=42)

# 5-fold кросс-валидация
scores = cross_val_score(
    model, X, y,
    cv=5,
    scoring='accuracy'
)

print(f"Scores: {scores}")
print(f"Mean: {scores.mean():.3f}")
print(f"Std: {scores.std():.3f}")
```

## ◆ 3. Типы кросс-валидации

| Метод             | Когда использовать         |
|-------------------|----------------------------|
| K-Fold            | Стандартный случай         |
| Stratified K-Fold | Несбалансированные классы  |
| Leave-One-Out     | Очень малая выборка        |
| Time Series Split | Временные ряды             |
| Group K-Fold      | Групповая структура данных |

## ◆ 4. Stratified K-Fold

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

# Сохраняет пропорции классов в каждом фолде
skf = StratifiedKFold(n_splits=5, shuffle=True,
                      random_state=42)

scores = cross_val_score(
    model, X, y,
    cv=skf,
    scoring='f1_weighted'
)

print(f"Stratified CV scores: {scores}")
print(f"Mean F1: {scores.mean():.3f}")
```

## ◆ 5. Разные метрики

```
from sklearn.model_selection import cross_validate

# Несколько метрик одновременно
scoring = ['accuracy', 'precision_weighted',
           'recall_weighted', 'f1_weighted']

cv_results = cross_validate(
    model, X, y,
    cv=5,
    scoring=scoring,
    return_train_score=True
)

for metric in scoring:
    test_score = cv_results[f'test_{metric}']
    print(f"{metric}: {test_score.mean():.3f} (+/- {test_score.std():.3f})")
```

## ◆ 6. Time Series Split

```
from sklearn.model_selection import TimeSeriesSplit

# Для временных рядов
tscv = TimeSeriesSplit(n_splits=5)

for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(f"Score: {score:.3f}")

# Не перемешивает данные
# Train всегда раньше test по времени
```

## ◆ 7. Кастомные фолды

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True,
random_state=42)

for fold, (train_idx, val_idx) in
enumerate(kf.split(X), 1):
    print(f"\nFold {fold}")

    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    model.fit(X_train, y_train)
    train_score = model.score(X_train, y_train)
    val_score = model.score(X_val, y_val)

    print(f"Train: {train_score:.3f}, Val:
{val_score:.3f}")
```

## ◆ 8. GridSearchCV с CV

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5, # Внутренняя кросс-валидация
    scoring='f1_weighted',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X, y)

print("Best params:", grid_search.best_params_)
print("Best CV score:", grid_search.best_score_)
```

## ◆ 9. Nested Cross-Validation

```
# Для честной оценки с подбором гиперпараметров

from sklearn.model_selection import
cross_val_score

# Внешний цикл CV
outer_cv = KFold(n_splits=5, shuffle=True,
random_state=42)

# GridSearch с внутренним CV
grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=3, # Внутренний CV
    scoring='accuracy'
)

# Оценка на внешнем CV
nested_scores = cross_val_score(
    grid_search, X, y,
    cv=outer_cv,
    scoring='accuracy'
)

print(f"Nested CV scores: {nested_scores}")
print(f"Mean: {nested_scores.mean():.3f}")
```

## ◆ 10. Leave-One-Out

```
from sklearn.model_selection import LeaveOneOut

# Только для малых датасетов (< 1000)
loo = LeaveOneOut()

scores = cross_val_score(model, X, y, cv=loo)

print(f"L0O CV score: {scores.mean():.3f}")
print(f"Total folds: {len(scores)}")

# Очень медленно, но максимально использует данные
```

## ◆ 11. Когда использовать

### ✓ Используйте CV

- ✓ Малый/средний датасет
- ✓ Нужна надёжная оценка
- ✓ Подбор гиперпараметров
- ✓ Сравнение моделей

### ✗ Не используйте CV

- ✗ Очень большой датасет (> 1M)
- ✗ Дорогое обучение модели
- ✗ Production inference
- ✗ Достаточно train/val/test split

## ◆ 12. Чек-лист

- [ ] Выбрать правильный тип CV для задачи
- [ ] Использовать StratifiedKFold для классификации
- [ ] Использовать TimeSeriesSplit для временных рядов
- [ ] Выбрать k=5 или k=10 для K-Fold
- [ ] Использовать shuffle=True (кроме временных рядов)
- [ ] Зафиксировать random\_state
- [ ] Оценить среднее и стандартное отклонение
- [ ] Для GridSearch — nested CV

### 💡 Объяснение заказчику:

«Кросс-валидация — это как проверка студента несколькими экзаменаторами. Вместо одного теста мы проводим несколько, каждый раз меняя, какие данные модель видела при обучении. Это даёт более объективную оценку».



17 Январь 2026

## ◆ 1. Суть метода

- **Multiple representatives:** кластер представлен несколькими точками, а не одним центроидом
- **Shrinking:** представители "усаживаются" к центру кластера (параметр  $\alpha$ )
- **Иерархический:** агломеративная кластеризация с особой метрикой расстояния
- **Устойчивость к выбросам:** случайная выборка + усадка точек
- **Произвольная форма:** не ограничен сферическими кластерами

## ◆ 2. Базовый код (Python)

```
# CURE нет в sklearn, используем pyclustering
from pyclustering.cluster.cure import cure
from pyclustering.utils import read_sample
from pyclustering.samplesdefinitions import FCPS_SAMPLES

# Загрузить данные
sample = read_sample(FCPS_SAMPLES.SAMPLE_LSUN)

# Создать CURE instance
cure_instance = cure(
    data=sample,
    number_cluster=3,           # число кластеров
    number_represent=5,         # представители на
    kластер                         # коэффи. усадки ( $\alpha$ )
    compression=0.5               # коэффи. усадки ( $\alpha$ )
)

# Кластеризация
cure_instance.process()
clusters = cure_instance.get_clusters()
representors = cure_instance.get_representors()

print(f"Найдено кластеров: {len(clusters)}")
print(f"Представителей: {len(representors)}")
```

## ◆ 3. Ключевые параметры

| Параметр         | Описание                        | Совет                           |
|------------------|---------------------------------|---------------------------------|
| number_cluster   | Число кластеров                 | 3-10 обычно                     |
| number_represent | Представителей на кластер       | 5-10 для хорошего покрытия      |
| compression      | $\alpha$ (0-1): усадка к центру | 0.3-0.7, меньше = меньше усадка |
| ccore            | Использовать C++ ядро           | True для скорости               |

## ◆ 4. Алгоритм работы

1. **Случайная выборка:** взять случайную подвыборку данных (для больших датасетов)
2. **Инициализация:** каждая точка — отдельный кластер
3. **Выбор представителей:** для каждого кластера выбрать с наиболье удалённых точек от центроида
4. **Усадка (shrinking):** сдвинуть представителей к центроиду на  $\alpha$ :  $p' = p + \alpha \cdot (\text{centroid} - p)$
5. **Слияние:** объединять ближайшие кластеры (расстояние = мин расстояние между представителями)
6. **Повторение 3-5:** пока не достигнуто число кластеров
7. **Назначение:** назначить все точки к ближайшему кластеру (по представителям)

## ◆ 5. Параметры детально

### **number\_represent (c):**

- Больше  $c \rightarrow$  лучше покрытие кластера  $\rightarrow$  точнее
- Меньше  $c \rightarrow$  быстрее, но хуже для сложных форм
- Рекомендация: 5-10 для большинства задач

### **compression ( $\alpha$ ):**

- $\alpha = 0$ : представители на границе кластера (устойчивость к выбросам++)
- $\alpha = 1$ : все представители в центре (как K-means)
- $\alpha = 0.3-0.5$ : хороший баланс

### **Расстояние между кластерами:**

- $\text{dist}(C_1, C_2) = \min(\text{dist}(p_1, p_2))$  для  $p_1 \in \text{Rep}(C_1), p_2 \in \text{Rep}(C_2)$
- Минимальное расстояние между представителями

## ◆ 6. Выбор параметров

### **Подбор через метрики:**

```
from sklearn.metrics import silhouette_score
import numpy as np

best_score = -1
best_params = None

for n_clust in [3, 4, 5, 6]:
    for n_rep in [5, 7, 10]:
        for alpha in [0.3, 0.5, 0.7]:
            cure_inst = cure(
                data=X.tolist(),
                number_cluster=n_clust,
                number_represent=n_rep,
                compression=alpha
            )
            cure_inst.process()
            clusters = cure_inst.get_clusters()

            # Создать метки
            labels = np.zeros(len(X))
            for i, cluster in enumerate(clusters):
                labels[cluster] = i

            if len(np.unique(labels)) > 1:
                score = silhouette_score(X,
   labels)
                if score > best_score:
                    best_score = score
                    best_params = (n_clust, n_rep,
                                   alpha)

print(f"Best: n={best_params[0]}, c={best_params[1]}, a={best_params[2]}")
print(f"Score: {best_score:.3f}")
```

## ◆ 7. Предобработка данных

### ✓ Масштабирование важно

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

### ✓ Обработка выбросов

- CURE устойчив к выбросам благодаря random sampling + shrinking
- Но крайние выбросы лучше удалить предварительно

### ✓ Снижение размерности

- PCA для >20 признаков ускоряет работу

## ◆ 8. Когда использовать

### ✓ Хорошо

- ✓ Кластеры произвольной формы
- ✓ Данные с выбросами
- ✓ Разные размеры кластеров
- ✓ Средний размер данных (10K-100K)
- ✓ Нелинейные границы

### ✗ Плохо

- ✗ Очень большие данные (>1M)
- ✗ Высокая размерность (>50)
- ✗ Нужна очень быстрая работа
- ✗ Неясное число кластеров
- ✗ Только сферические кластеры (используйте K-means)

## ◆ 9. Проблемы и решения

| Проблема             | Решение                                           |
|----------------------|---------------------------------------------------|
| Медленная работа     | Уменьшить number_represent, использовать sampling |
| Плохое качество      | Увеличить number_represent, подобрать compression |
| Выбросы влияют       | Уменьшить compression ( $\alpha$ ), предобработка |
| Не находит формы     | Увеличить number_represent, уменьшить $\alpha$    |
| Память заканчивается | Random sampling, уменьшить размер данных          |

## ◆ 10. Визуализация

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 2D проекция
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)

# Получить кластеры и представителей
cure_inst.process()
clusters = cure_inst.get_clusters()
representors = cure_inst.getRepresentors()

# Создать метки
labels = np.zeros(len(X))
for i, cluster in enumerate(clusters):
    for idx in cluster:
        labels[idx] = i

# Точки
plt.scatter(X_2d[:, 0], X_2d[:, 1],
            c=labels, cmap='viridis', alpha=0.6,
            s=30)

# Представители
for i, reps in enumerate(representors):
    rep_2d = X_2d[reps]
    plt.scatter(rep_2d[:, 0], rep_2d[:, 1],
                c='red', marker='*', s=200,
                edgecolors='black',
                linewidths=1.5)

    # Соединить представителей кластера
    for j in range(len(reps)):
        for k in range(j+1, len(reps)):
            plt.plot([rep_2d[j, 0], rep_2d[k, 0]],
                    [rep_2d[j, 1], rep_2d[k, 1]],
                    'r-', alpha=0.3, linewidth=1)

plt.title(f'CURE: {len(clusters)} кластеров')
plt.show()
```

## ◆ 11. Сравнение с другими методами

| Метод        | Преимущества CURE                                  | Недостатки CURE                    |
|--------------|----------------------------------------------------|------------------------------------|
| K-means      | Произвольные формы, устойчивость к выбросам        | Медленнее, нужно задать K          |
| DBSCAN       | Детерминированный, нет $\epsilon$ и $min\_samples$ | Нужно задать K                     |
| Hierarchical | Лучше для несферических кластеров                  | Сложнее настройка                  |
| BIRCH        | Лучше находит сложные формы                        | Медленнее, меньше масштабируемость |

## ◆ 12. Метрики оценки

```
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score
)

# Создать метки из кластеров
labels = np.zeros(len(X))
for i, cluster in enumerate(clusters):
    for idx in cluster:
        labels[idx] = i

n_clusters = len(np.unique(labels))

if n_clusters > 1:
    sil = silhouette_score(X, labels)
    db = davies_bouldin_score(X, labels)
    ch = calinski_harabasz_score(X, labels)

    print(f"Кластеров: {n_clusters}")
    print(f"Silhouette: {sil:.3f}")
    print(f"Davies-Bouldin: {db:.3f}")
    print(f"Calinski-Harabasz: {ch:.3f}")
```

## ◆ 13. Применения

### Spatial data:

- Географическая кластеризация (города, регионы)
- Астрономические данные

### Биология:

- Анализ генов с выбросами
- Таксономия

### Маркетинг:

- Сегментация клиентов
- Анализ поведения с аномалиями

## ◆ 15. Чек-лист

- [ ] Установить pyclustering: pip install pyclustering
- [ ] Масштабировать данные
- [ ] Определить число кластеров
- [ ] Выбрать `number_represent` (5-10)
- [ ] Установить `compression` (0.3-0.7)
- [ ] Обработать выбросы если нужно
- [ ] Оценить качество (silhouette)
- [ ] Визуализировать представителей

### 💡 Объяснение заказчику:

«CURE представляет каждую группу несколькими характерными точками, а не одним центром. Это позволяет находить группы любой формы и делает метод устойчивым к аномальным данным. Особенно хороши для данных со сложной геометрией».

## ◆ 14. Продвинутые техники

### Иерархический CURE:

- Можно остановить слияние на любом уровне
- Получить дендрограмму кластеров

### Параллельный CURE:

- Разделить данные на партиции
- Применить CURE к каждой партиции
- Объединить представителей
- Финальная кластеризация представителей

### Адаптивное число представителей:

```
# Больше представителей для больших кластеров
def adaptive_representatives(cluster_size):
    return max(5, min(15, cluster_size // 10))
```

## ◆ 16. Установка pyclustering

```
# Установка
pip install pyclustering

# Или с conda
conda install -c conda-forge pyclustering

# Проверка
from pyclustering.cluster.cure import cure
print("pyclustering установлен успешно!")
```

**Альтернатива:** Реализация CURE вручную

- Использовать `scipy.cluster.hierarchy` для агломеративной кластеризации
- Реализовать свою метрику расстояния с представителями
- Добавить логику shrinking



# Curriculum Learning

17 Январь 2026

## ◆ 1. Что такое Curriculum Learning

- **Цель:** обучать модель от простых примеров к сложным
- **Идея:** имитация человеческого обучения
- **Результат:** быстрая сходимость, лучшее качество
- **Применение:** CV, NLP, RL
- **Ключевые аспекты:** определение сложности, расписание

## ◆ 2. Основные стратегии

### 1. Predefined Curriculum:

- Заранее определенный порядок
- Основан на экспертных знаниях
- Простой в реализации

### 2. Self-Paced Learning:

- Модель сама выбирает примеры
- На основе loss или confidence
- Адаптивный подход

### 3. Transfer Teacher:

- Учительская модель выбирает примеры
- На основе предсказаний

### ◆ 3. Определение сложности

```

import torch
import numpy as np

class CurriculumDataset(torch.utils.data.Dataset):
    def __init__(self, X, y, difficulty_fn):
        self.X = X
        self.y = y

        # Вычисляем сложность для каждого примера
        self.difficulties = difficulty_fn(X, y)

        # Сортируем по сложности
        self.sorted_indices =
            np.argsort(self.difficulties)

        self.current_size = len(X) // 10 # Начинаем с 10%

    def set_curriculum_stage(self, stage):
        """Увеличиваем размер датасета"""
        max_size = len(self.X)
        self.current_size = min(
            int(max_size * (stage + 1) / 10),
            max_size
        )

    def __len__(self):
        return self.current_size

    def __getitem__(self, idx):
        real_idx = self.sorted_indices[idx]
        return self.X[real_idx], self.y[real_idx]

# Примеры функций сложности
def difficulty_by_label_noise(X, y):
    """Сложность на основе шума в метках"""
    # Больше шума = выше сложность
    return np.random.rand(len(X))

def difficulty_by_feature_variance(X, y):
    """Сложность на основе вариации признаков"""
    return np.var(X, axis=1)

def difficulty_by_class_rarity(X, y):
    """Сложность на основе редкости класса"""
    unique, counts = np.unique(y,
                                return_counts=True)
    class_difficulty = {
        cls: 1.0 / count
        for cls, count in zip(unique, counts)
    }
    return np.array([class_difficulty[label] for
label in y])

```

### ◆ 4. Baby Step Curriculum

```

class BabyStepCurriculum:
    def __init__(self, model, train_loader,
                 epochs_per_stage=10):
        self.model = model
        self.train_loader = train_loader
        self.epochs_per_stage = epochs_per_stage
        self.num_stages = 10

    def train(self):
        optimizer = torch.optim.Adam(
            self.model.parameters(),
            lr=0.001
        )
        criterion = torch.nn.CrossEntropyLoss()

        for stage in range(self.num_stages):
            # Обновляем размер датасета
            self.train_loader.dataset.set_curriculum_stage(stage)

            print(f"Stage {stage+1}/{self.num_stages}")
            print(f"Training on {len(self.train_loader.dataset)} samples")

            # Обучаем на текущем этапе
            for epoch in
range(self.epochs_per_stage):
                self.model.train()
                total_loss = 0

                for batch_X, batch_y in
self.train_loader:
                    optimizer.zero_grad()
                    outputs = self.model(batch_X)
                    loss = criterion(outputs,
batch_y)
                    loss.backward()
                    optimizer.step()

                    total_loss += loss.item()

                avg_loss = total_loss /
len(self.train_loader)
                print(f" Epoch {epoch+1}, Loss:
{avg_loss:.4f}")

    # Использование
dataset = CurriculumDataset(
    X_train, y_train,
    difficulty_fn=difficulty_by_feature_variance
)
dataloader = torch.utils.data.DataLoader(
    dataset, batch_size=32, shuffle=True
)

```

```

curriculum = BabyStepCurriculum(model, dataloader)
curriculum.train()

```

## ◆ 5. Self-Paced Learning

```

class SelfPacedLearning:
    def __init__(self, model, train_data,
lambda_init=1.0):
        self.model = model
        self.X, self.y = train_data
        self.lambda_param = lambda_init
        self.weights = np.ones(len(self.X))

    def update_weights(self, losses):
        """Обновляем веса примеров на основе
loss"""
        # Веса больше для простых примеров (малый
loss)
        self.weights = (losses <
self.lambda_param).astype(float)

        # Постепенно увеличиваем lambda
        # (включаем более сложные примеры)
        self.lambda_param *= 1.1

    def train_epoch(self):
        optimizer = torch.optim.Adam(
            self.model.parameters(),
            lr=0.001
        )
        criterion =
torch.nn.CrossEntropyLoss(reduction='none')

        self.model.train()
        losses = []

        for i in range(len(self.X)):
            if self.weights[i] > 0: # Используем
только выбранные
                x =
torch.FloatTensor(self.X[i]).unsqueeze(0)
                y = torch.LongTensor([self.y[i]])

                optimizer.zero_grad()
                output = self.model(x)
                loss = criterion(output, y)

                # Взвешенный loss
                weighted_loss = loss *
self.weights[i]
                weighted_loss.backward()
                optimizer.step()

                losses.append(loss.item())

            # Обновляем веса для следующей эпохи
            self.update_weights(np.array(losses))

        return np.mean(losses)

    def train(self, num_epochs=50):

```

## Curriculum Learning Cheatsheet — 3 колонки

```

for epoch in range(num_epochs):
    avg_loss = self.train_epoch()
    num_selected = np.sum(self.weights > 0)

    print(f"Epoch {epoch+1}: Loss=
{avg_loss:.4f}, "
          f"Selected=
{num_selected}/{len(self.X)}, "
          f"Lambda=
{self.lambda_param:.2f}")

```

## ◆ 6. Curriculum для NLP

```

# Сложность по длине предложения
def text_difficulty_by_length(texts):
    return np.array([len(text.split()) for text in
texts])

# Сложность по редкости слов
def text_difficulty_by_vocabulary(texts,
vocab_freq):
    difficulties = []
    for text in texts:
        words = text.split()
        # Среднее кол-во редких слов
        rare_count = sum(
            1 for w in words if vocab_freq.get(w,
0) < 100
        )
        difficulties.append(rare_count /
len(words))
    return np.array(difficulties)

# Сложность по синтаксической структуре
def text_difficulty_by_syntax(texts):
    """Используем spacy для анализа"""
    import spacy
    nlp = spacy.load('en_core_web_sm')

    difficulties = []
    for text in texts:
        doc = nlp(text)
        # Глубина дерева зависимостей
        max_depth = max(
            len(list(token.ancestors)) for token in
doc
        )
        difficulties.append(max_depth)

    return np.array(difficulties)

# Curriculum для текстовой классификации
class TextCurriculumDataset(torch.utils.data.Dataset):
    def __init__(self, texts, labels, tokenizer,
difficulty='length'):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer

        # Выбираем метрику сложности
        if difficulty == 'length':
            self.difficulties =
text_difficulty_by_length(texts)
            # ... другие метрики

            self.sorted_indices =
np.argsort(self.difficulties)

```

```

        self.current_size = len(texts) // 5

    def set_stage(self, stage):
        self.current_size = min(
            len(self.texts) * (stage + 1) // 5,
            len(self.texts)
        )

    def __getitem__(self, idx):
        real_idx = self.sorted_indices[idx]
        text = self.texts[real_idx]
        label = self.labels[real_idx]

        encoding = self.tokenizer(
            text,
            truncation=True,
            padding='max_length',
            max_length=128,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].squeeze(),
            'attention_mask': encoding['attention_mask'].squeeze(),
            'labels': torch.tensor(label)
        }

```

## 7. Curriculum для Computer Vision

```

from PIL import Image, ImageFilter

# Сложность по размытости
def image_difficulty_by_blur(images):
    difficulties = []
    for img in images:
        # Вариация лапласиана (размытость)
        gray = img.convert('L')
        laplacian_var = np.array(
            gray.filter(ImageFilter.FIND_EDGES)
        ).var()
        difficulties.append(1.0 / (laplacian_var +
1))
    return np.array(difficulties)

# Сложность по количеству объектов
def image_difficulty_by_object_count(images,
detector):
    """Используем object detector для подсчета
объектов"""
    difficulties = []
    for img in images:
        detections = detector(img)
        num_objects = len(detections)
        difficulties.append(num_objects)
    return np.array(difficulties)

# Сложность по размеру объектов
def image_difficulty_by_object_size(images,
bboxes):
    """Меньший объект = выше сложность"""
    difficulties = []
    for bbox_list in bboxes:
        if len(bbox_list) == 0:
            difficulties.append(0)
        else:
            min_size = min(
                (x2-x1) * (y2-y1)
                for x1, y1, x2, y2 in bbox_list
            )
            difficulties.append(1.0 / (min_size +
1))
    return np.array(difficulties)

# Progressive resizing - популярная техника
class ProgressiveResizeCurriculum:
    def __init__(self, model, start_size=64,
end_size=224, stages=4):
        self.model = model
        self.sizes = np.linspace(start_size,
end_size, stages).astype(int)
        self.current_stage = 0

    def get_transform(self):
        size = self.sizes[self.current_stage]

```

```

        return transforms.Compose([
            transforms.Resize(size),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            )
        ])

    def next_stage(self):
        self.current_stage = min(
            self.current_stage + 1,
            len(self.sizes) - 1
        )
        print(f'Moving to stage
{self.current_stage+1}, "
f"size=
{sizes[self.current_stage]}")
```

## ◆ 8. Curriculum для RL

```
class CurriculumRL:
    """Curriculum для Reinforcement Learning"""

    def __init__(self, env, agent, stages):
        self.env = env
        self.agent = agent
        self.stages = stages # Список настроек
        среды
        self.current_stage = 0

    def get_stage_env(self):
        """Возвращает среду текущей сложности"""
        stage_config =
        self.stages[self.current_stage]

        # Пример: изменение параметров среды

        self.env.set_difficulty(stage_config['difficulty'])

        self.env.set_reward_shaping(stage_config['reward_scale'])

        return self.env

    def should_advance(self, performance_history):
        """Проверка готовности к следующему
        этапу"""
        if len(performance_history) < 100:
            return False

        recent_performance =
        np.mean(performance_history[-100:])
        threshold = self.stages[self.current_stage]
        ['threshold']

        return recent_performance >= threshold

    def train(self, num_episodes=10000):
        episode_rewards = []

        for episode in range(num_episodes):
            env = self.get_stage_env()
            state = env.reset()
            episode_reward = 0
            done = False

            while not done:
                action =
            self.agent.select_action(state)
                next_state, reward, done, _ =
            env.step(action)

                self.agent.update(state, action,
            reward, next_state, done)

                state = next_state
                episode_reward += reward
```

```
episode_rewards.append(episode_reward)

# Проверяем готовность к следующему
этапу
if
self.should_advance(episode_rewards):
    if self.current_stage <
len(self.stages) - 1:
        self.current_stage += 1
        print(f"Advanced to stage
{self.current_stage+1}")

    if episode % 100 == 0:
        avg_reward =
np.mean(episode_rewards[-100:])
        print(f"Episode {episode}, "
f"Avg Reward:
{avg_reward:.2f}, "
f"Stage:
{self.current_stage+1}")

# Пример конфигурации для игры
stages = [
    {'difficulty': 0.2, 'reward_scale': 1.5,
'threshold': 100},
    {'difficulty': 0.4, 'reward_scale': 1.2,
'threshold': 200},
    {'difficulty': 0.6, 'reward_scale': 1.0,
'threshold': 300},
    {'difficulty': 0.8, 'reward_scale': 1.0,
'threshold': 400},
    {'difficulty': 1.0, 'reward_scale': 1.0,
'threshold': 500}
]
```

## ◆ 9. Teacher-Student Curriculum

```
class TeacherStudentCurriculum:
    """Учитель выбирает примеры для ученика"""

    def __init__(self, teacher_model,
student_model, train_data):
        self.teacher = teacher_model
        self.student = student_model
        self.X, self.y = train_data

        # Учитель предобучен на полных данных
        self.teacher.eval()

    def select_samples(self, batch_size,
difficulty_threshold):
        """Учитель выбирает подходящие примеры"""
        with torch.no_grad():
            # Получаем предсказания учителя
            X_tensor = torch.FloatTensor(self.X)
            teacher_logits = self.teacher(X_tensor)
            teacher_probs =
torch.softmax(teacher_logits, dim=1)

            # Confidence учителя
            confidences, _ =
teacher_probs.max(dim=1)

            # Выбираем примеры с подходящей
сложностью
            # Высокий confidence = простой пример
            mask = (confidences >=
difficulty_threshold) &
(confidences <= difficulty_threshold + 0.2)

            selected_indices = torch.where(mask)[0]

            if len(selected_indices) < batch_size:
                selected_indices =
torch.randperm(len(self.X))[:batch_size]
            else:
                selected_indices =
selected_indices[
torch.randperm(len(selected_indices))[:batch_size]
]

            return self.X[selected_indices],
self.y[selected_indices]

    def train_student(self, num_epochs=50):
        optimizer = torch.optim.Adam(
            self.student.parameters(),
            lr=0.001
        )
        criterion = torch.nn.CrossEntropyLoss()

        # Начинаем с высокого порога (простые
```

```
примеры)
    difficulty_threshold = 0.8

    for epoch in range(num_epochs):
        # Постепенно снижаем порог (добавляем
        сложные)
        difficulty_threshold = max(0.3, 0.8 -
epoch * 0.01)

        batch_X, batch_y = self.select_samples(
            batch_size=32,
            difficulty_threshold=difficulty_threshold
        )

        self.student.train()
        optimizer.zero_grad()

        outputs =
self.student(torch.FloatTensor(batch_X))
        loss = criterion(outputs,
torch.LongTensor(batch_y))

        loss.backward()
        optimizer.step()

        if epoch % 10 == 0:
            print(f"Epoch {epoch}, Loss:
{loss.item():.4f}, "
                  f"Difficulty:
{difficulty_threshold:.2f}")
```

## ◆ 10. Практические советы

- Определение сложности:** domain-specific метрики
- Скорость прогресса:** не слишком быстро/ медленно
- Мониторинг:** отслеживать качество на каждом этапе
- Адаптивность:** self-paced лучше fixed curriculum
- Transfer:** curriculum помогает transfer learning

## ◆ 11. Сравнение подходов

| Метод           | Автоматизация | Сложность | Эффективность |
|-----------------|---------------|-----------|---------------|
| Predefined      | ✗             | Низкая    | Средняя       |
| Self-Paced      | ✓             | Средняя   | Высокая       |
| Teacher-Student | ✓             | Высокая   | Очень высокая |
| RL Curriculum   | ⚠             | Высокая   | Высокая       |

## ◆ 12. Применения

### ✓ Хорошо работает

- ✓ Сложные задачи с четкой градацией
- ✓ Reinforcement Learning
- ✓ Few-shot learning
- ✓ Domain adaptation
- ✓ Noisy labels
- ✓ Imbalanced datasets

### ✗ Ограничения

- ✗ Сложно определить метрику сложности
- ✗ Дополнительные гиперпараметры
- ✗ Может замедлить обучение
- ✗ Требует экспериментов

## ◆ 13. Чек-лист

- [ ] Определить метрику сложности примеров
- [ ] Выбрать стратегию curriculum (predefined/self-paced)
- [ ] Настроить расписание изменения сложности
- [ ] Мониторить качество на каждом этапе
- [ ] Экспериментировать с скоростью прогресса
- [ ] Сравнить с baseline без curriculum
- [ ] Проверить на validation set
- [ ] Визуализировать прогресс обучения
- [ ] Документировать выбранную стратегию

## 💡 Объяснение заказчику:

«Curriculum Learning — это как обучение в школе: сначала изучаем простые примеры, потом переходим к более сложным. Это помогает модели быстрее учиться и достигать лучших результатов».

## ◆ 1. Суть

- Цель:** искусственно увеличить размер обучающей выборки
- Метод:** создание модифицированных копий данных
- Эффект:** улучшение обобщающей способности модели
- Борьба с переобучением:** модель видит больше вариаций
- Применение:** особенно критично при малом объёме данных

## ◆ 2. Аугментация изображений (основные техники)

| Трансформация          | Описание                     |
|------------------------|------------------------------|
| <b>Поворот</b>         | Вращение на случайный угол   |
| <b>Отражение</b>       | Горизонтальное/вертикальное  |
| <b>Масштабирование</b> | Увеличение/уменьшение        |
| <b>Сдвиг</b>           | Смещение по осям             |
| <b>Обрезка</b>         | Random crop                  |
| <b>Яркость</b>         | Изменение освещённости       |
| <b>Контраст</b>        | Изменение контраста          |
| <b>Шум</b>             | Добавление гауссовского шума |

## ◆ 3. Keras/TensorFlow (базовый)

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Создаём генератор с аугментацией
datagen = ImageDataGenerator(
    rotation_range=20,           # Поворот ±20°
    width_shift_range=0.2,        # Сдвиг по ширине
    height_shift_range=0.2,       # Сдвиг по высоте
    horizontal_flip=True,        # Отражение
    zoom_range=0.2,              # Масштабирование
    fill_mode='nearest'          # Заполнение
)

# Обучение с аугментацией
model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=50,
    validation_data=(X_val, y_val)
)
```

## ◆ 4. Albumentations (продвинутая библиотека)

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

# Композиция трансформаций
transform = A.Compose([
    A.Rotate(limit=30, p=0.5),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.GaussNoise(var_limit=(10, 50), p=0.2),
    A.RandomCrop(height=224, width=224),
    A.Normalize(mean=[0.485, 0.456, 0.406],
               std=[0.229, 0.224, 0.225]),
    ToTensorV2()
])

# Применение
augmented = transform(image=image)
aug_image = augmented['image']

# В PyTorch DataLoader
class AugmentedDataset(Dataset):
    def __init__(self, images, labels, transform):
        self.images = images
        self.labels = labels
        self.transform = transform

    def __getitem__(self, idx):
        image = self.images[idx]
        augmented = self.transform(image=image)
        return augmented['image'],
        self.labels[idx]
```

## ◆ 5. PyTorch (transforms)

```
from torchvision import transforms

# Трансформации для обучения
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2,
        saturation=0.2,
        hue=0.1
    ),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

# Для валидации - только нормализация!
val_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

## ◆ 7. Mixup (код)

```
import numpy as np

def mixup(x1, y1, x2, y2, alpha=0.2):
    """
    Смешивает два изображения и метки
    """
    lam = np.random.beta(alpha, alpha)

    # Смешиваем изображения
    x_mixed = lam * x1 + (1 - lam) * x2

    # Смешиваем метки
    y_mixed = lam * y1 + (1 - lam) * y2

    return x_mixed, y_mixed

# В training loop:
idx = np.random.permutation(len(X_batch))
X_mixed, y_mixed = mixup(
    X_batch, y_batch,
    X_batch[idx], y_batch[idx],
    alpha=0.2
)

loss = model.train_on_batch(X_mixed, y_mixed)
```

## ◆ 8. Аугментация текста

- **Синонимы:** замена слов синонимами
- **Обратный перевод:** перевести на другой язык и обратно
- **Вставка:** вставка случайных слов
- **Удаление:** удаление случайных слов
- **Перестановка:** изменение порядка слов
- **Контекстная замена:** BERT-based замены

```
# Пример с nlpaug
import nlpaug.augmenter.word as naw

# Синонимы
aug = naw.SynonymAug(aug_src='wordnet')
text = "Машинное обучение очень интересно"
augmented = aug.augment(text)

# Обратный перевод
aug = naw.BackTranslationAug(
    from_model_name='Helsinki-NLP/opus-mt-ru-en',
    to_model_name='Helsinki-NLP/opus-mt-en-ru'
)
augmented = aug.augment(text)
```

## ◆ 6. Продвинутые техники для изображений

- **Cutout:** случайное закрытие квадратных областей
- **Mixup:** смешивание двух изображений
- **CutMix:** вырезание и вставка частей изображений
- **AutoAugment:** автоматический поиск политик
- **RandAugment:** упрощённый AutoAugment
- **GridMask:** удаление регулярных сеток

## ◆ 9. Аугментация временных рядов

- **Jittering:** добавление шума
- **Scaling:** масштабирование амплитуды
- **Time Warping:** растяжение/сжатие времени
- **Window Slicing:** извлечение подокон
- **Rotation:** поворот в пространстве признаков

```
import numpy as np

# Jittering
def jitter(x, sigma=0.03):
    return x + np.random.normal(0, sigma, x.shape)

# Scaling
def scaling(x, sigma=0.1):
    factor = np.random.normal(1, sigma)
    return x * factor

# Window Slicing
def window_slice(x, reduce_ratio=0.9):
    target_len = int(len(x) * reduce_ratio)
    start = np.random.randint(0, len(x) - target_len)
    return x[start:start+target_len]
```

## ◆ 10. Аугментация аудио

- **Time Stretching:** изменение скорости
- **Pitch Shifting:** изменение тона
- **Background Noise:** добавление шума
- **Volume Control:** изменение громкости
- **Time Masking:** маскирование временных интервалов
- **Frequency Masking:** маскирование частот

```
# С использованием audiomentations
from audiomentations import Compose,
AddGaussianNoise, TimeStretch

augment = Compose([
    AddGaussianNoise(min_amplitude=0.001,
                      max_amplitude=0.015, p=0.5),
    TimeStretch(min_rate=0.8, max_rate=1.25,
                p=0.5),
])
augmented_audio = augment(samples=audio,
                           sample_rate=16000)
```

## ◆ 11. Когда применять

### ✓ Хорошо

- ✓ Малый размер обучающей выборки
- ✓ Модель переобучается
- ✓ Нужна инвариантность к трансформациям
- ✓ Несбалансированные классы (аугментировать миноритарный)
- ✓ Computer vision задачи

### ✗ Плохо

- ✗ Очень большой датасет (не нужно)
- ✗ Трансформации меняют семантику
- ✗ Медицинские изображения (осторожно!)
- ✗ Тест и валидация (НИКОГДА!)

## ◆ 12. Лучшие практики

- **Не аугментируйте валидацию/тест:** только train!
- **Реалистичность:** трансформации должны быть правдоподобными
- **Разумность:** не переусердствуйте (можно ухудшить качество)
- **Доменные знания:** учитывайте специфику задачи
- **Экспериментируйте:** найдите оптимальный набор
- **On-the-fly:** аугментация на лету экономит память

## ◆ 13. Offline vs Online аугментация

| Подход  | Описание                                | Плюсы/<br>Минусы                                                                                                                                                                                                                 |
|---------|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Offline | Заранее создать аугментированные данные | <ul style="list-style-type: none"> <li><span style="color: green;">✓</span> Быстрое обучение</li> <li><span style="color: red;">✗</span> Больше места на диске</li> </ul>                                                        |
| Online  | Аугментация во время обучения           | <ul style="list-style-type: none"> <li><span style="color: green;">✓</span> Экономия памяти</li> <li><span style="color: green;">✓</span> Больше вариаций</li> <li><span style="color: red;">✗</span> Медленнее эпоха</li> </ul> |

**Рекомендация:** Online для большинства случаев

## ◆ 14. Чек-лист

- [ ] Определить, нужна ли аугментация (малая выборка? переобучение?)
- [ ] Выбрать подходящие трансформации для домена
- [ ] Применять ТОЛЬКО к обучающей выборке
- [ ] Проверить, что аугментированные данные выглядят реалистично
- [ ] Начать с простых трансформаций
- [ ] Экспериментировать с интенсивностью
- [ ] Использовать on-the-fly аугментацию
- [ ] Мониторить метрики на валидации

### 💡 Объяснение заказчику:

«Аугментация данных — это как если бы мы показывали модели одно и то же изображение, но с разных углов, при разном освещении, в разных размерах. Модель учится распознавать объект в любых условиях, а не запоминать конкретные примеры».



# Предобработка данных: пропуски и кодирование

Январь 2026

## ◆ 1. Обзор

**Предобработка данных** — критический этап pipeline ML, включающий обработку пропущенных значений и преобразование категориальных переменных.

- **Цель:** подготовить данные для ML алгоритмов
- **Важность:** 80% времени в ML проектах
- **Влияние:** может улучшить точность на 10-30%
- **Ключевые задачи:**
  - Обработка пропусков (missing values)
  - Кодирование категориальных признаков
  - Масштабирование и нормализация

## ◆ 2. Типы пропущенных значений

| Тип                                           | Описание                                          | Пример                                 |
|-----------------------------------------------|---------------------------------------------------|----------------------------------------|
| <b>MCAR</b><br>(Missing Completely At Random) | Пропуски полностью случайны, не зависят от данных | Случайные сбои оборудования            |
| <b>MAR</b><br>(Missing At Random)             | Пропуски зависят от наблюдаемых данных            | Молодые люди реже указывают доход      |
| <b>MNAR</b><br>(Missing Not At Random)        | Пропуски зависят от самого пропущенного значения  | Люди с низким доходом не указывают его |

Понимание типа пропусков критично для выбора метода обработки!

## ◆ 3. Обнаружение пропусков

```
import pandas as pd
import numpy as np

# Проверка пропусков
df.isnull().sum()
df.isna().sum() # то же самое

# Процент пропусков
missing_pct = (df.isnull().sum() / len(df)) * 100
print(missing_pct.sort_values(ascending=False))

# Визуализация пропусков
import missingno as msno
msno.matrix(df)
msno.heatmap(df) # корреляция пропусков

# Паттерны пропусков
msno.dendrogram(df)
```

## ◆ 4. Удаление пропусков

**Когда использовать:** пропусков < 5%, данных достаточно

```
# Удалить строки с пропусками
df_clean = df.dropna()
```

```
# Удалить строки, где все NaN
df_clean = df.dropna(how='all')
```

```
# Удалить строки с NaN в конкретных столбцах
df_clean = df.dropna(subset=['age', 'income'])
```

```
# Удалить столбцы с > 50% пропусков
threshold = 0.5
df_clean = df.dropna(
    thresh=len(df) * threshold,
    axis=1
)
```

### ✓ Когда удалять

- ✓ Пропусков очень мало (< 5%)
- ✓ MCAR тип пропусков
- ✓ Большой датасет

### ✗ Когда не удалять

- ✗ Много пропусков (> 10%)
- ✗ Малый датасет
- ✗ MNAR или MAR типы

## ◆ 5. Простое заполнение (Imputation)

**Базовые методы** заполнения константами и статистиками:

```
from sklearn.impute import SimpleImputer

# Заполнение средним (для числовых)
imputer = SimpleImputer(strategy='mean')
df['age'] = imputer.fit_transform(df[['age']])

# Заполнение медианой (устойчиво к выбросам)
imputer = SimpleImputer(strategy='median')
df['income'] =
imputer.fit_transform(df[['income']])

# Заполнение модой (для категориальных)
imputer = SimpleImputer(strategy='most_frequent')
df['category'] =
imputer.fit_transform(df[['category']])

# Заполнение константой
imputer = SimpleImputer(strategy='constant',
fill_value=0)
df['score'] = imputer.fit_transform(df[['score']])

# Pandas методы
df['age'].fillna(df['age'].mean(), inplace=True)
df['city'].fillna('Unknown', inplace=True)

# Forward fill (использовать предыдущее значение)
df['price'].fillna(method='ffill', inplace=True)

# Backward fill
df['price'].fillna(method='bfill', inplace=True)
```

## ◆ 6. Продвинутое заполнение

**KNN Imputation:** использует K ближайших соседей

```
from sklearn.impute import KNNImputer

# KNN заполнение
imputer = KNNImputer(n_neighbors=5)
df_filled = pd.DataFrame(
    imputer.fit_transform(df),
    columns=df.columns
)
```

**Итеративное заполнение (MICE):**

```
from sklearn.experimental import
enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Итеративное заполнение
imputer = IterativeImputer(
    max_iter=10,
    random_state=42
)
df_filled = pd.DataFrame(
    imputer.fit_transform(df),
    columns=df.columns
)
```

**Заполнение на основе модели:**

```
from sklearn.ensemble import RandomForestRegressor

# Обучить модель для предсказания пропусков
def model_imputation(df, target_col):
    train = df[df[target_col].notna()]
    test = df[df[target_col].isna()]

    features = [c for c in df.columns if c !=
target_col]

    model = RandomForestRegressor()
    model.fit(train[features], train[target_col])

    df.loc[df[target_col].isna(), target_col] = \
model.predict(test[features])

    return df
```

## ◆ 7. Индикаторы пропусков

Создание дополнительных признаков для отслеживания пропусков:

```
# Создать бинарный признак для пропусков
df['age_was_missing'] =
df['age'].isnull().astype(int)

# Затем заполнить пропуски
df['age'].fillna(df['age'].median(), inplace=True)

# Sklearn метод
from sklearn.impute import MissingIndicator

indicator = MissingIndicator()
missing_mask = indicator.fit_transform(df)

# Добавить все индикаторы
for i, col in enumerate(df.columns):
    if indicator.features_[i]:
        df[f'{col}_missing'] = missing_mask[:, i]
```

 Индикаторы пропусков могут содержать важную информацию для модели!

## ◆ 8. Кодирование категориальных признаков

**Типы категориальных переменных:**

- **Номинальные:** нет порядка (цвет, город)
- **Порядковые:** есть порядок (образование, рейтинг)

**Label Encoding** — простое числовое кодирование:

```
from sklearn.preprocessing import LabelEncoder

# Label Encoding (для порядковых)
le = LabelEncoder()
df['education'] =
le.fit_transform(df['education'])

# Вручную для контроля порядка
education_map = {
    'High School': 1,
    'Bachelor': 2,
    'Master': 3,
    'PhD': 4
}
df['education'] =
df['education'].map(education_map)
```

**⚠ Label Encoding для номинальных создает ложный порядок!**

## ◆ 9. One-Hot Encoding

**Лучший выбор для номинальных переменных:**

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Pandas метод (простой)
df_encoded = pd.get_dummies(
    df,
    columns=['city', 'color'],
    drop_first=True # избежать мультиколлинеарности
)

# Sklearn метод (для pipeline)
encoder = OneHotEncoder(
    drop='first', # drop first category
    sparse=False,
    handle_unknown='ignore' # игнорировать новые категории
)

encoded = encoder.fit_transform(df[['city']])
encoded_df = pd.DataFrame(
    encoded,
    columns=encoder.get_feature_names_out(['city'])
)

# Объединить с исходным датафреймом
df = pd.concat([df, encoded_df], axis=1)
df.drop('city', axis=1, inplace=True)
```

**Проблемы One-Hot Encoding:**

- ❌ Большое количество категорий → много колонок
- ❌ Разреженные матрицы (sparse)
- ❌ Проклятие размерности

## ◆ 10. Частотное кодирование

Кодирование категорий по их частоте:

```
# Подсчет частоты категорий
frequency =
df['city'].value_counts(normalize=True)

# Замена на частоты
df['city_freq'] = df['city'].map(frequency)

# Альтернативно - счетчик
count = df['city'].value_counts()
df['city_count'] = df['city'].map(count)
```

**Преимущества:**

- ✓ Сохраняет размерность (1 колонка)
- ✓ Работает с высокой кардинальностью
- ✓ Быстро и эффективно

## ◆ 11. Target Encoding

Кодирование на основе целевой переменной:

```
from category_encoders import TargetEncoder

# Target Encoding
encoder = TargetEncoder()
df['city_encoded'] = encoder.fit_transform(
    df['city'],
    df['target']
)

# Вручную с регуляризацией
def target_encode(df, category, target, alpha=5):
    """
    alpha - параметр сглаживания
    """
    # Глобальное среднее
    global_mean = df[target].mean()

    # Среднее по категориям
    agg = df.groupby(category)
    [target].agg(['mean', 'count'])

    # Сглаживание
    encoded = (agg['mean'] * agg['count'] +
               global_mean * alpha) /
    (agg['count'] + alpha)

    return df[category].map(encoded)
```

**⚠ Риск переобучения!** Используйте кросс-валидацию!

## ◆ 12. Best Practices

### ✓ Делать

- ✓ Анализировать паттерны пропусков
- ✓ Использовать Pipeline для воспроизводимости
- ✓ Отдельно обрабатывать train/test
- ✓ Документировать решения
- ✓ Проверять результаты визуально
- ✓ Сохранять индикаторы пропусков

### ✗ Избегать

- ✗ Утечка данных из test в train
- ✗ Label Encoding для номинальных
- ✗忽орирование типа пропусков
- ✗ One-Hot для высокой кардинальности
- ✗ Target Encoding без CV

## ◆ 13. Полный Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler,
OneHotEncoder

# Определить типы признаков
numeric_features = ['age', 'income']
categorical_features = ['city', 'education']

# Pipeline для числовых
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Pipeline для категориальных
categorical_transformer = Pipeline(steps=[
    ('imputer',
     SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(drop='first',
                           handle_unknown='ignore'))
])

# Объединить
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer,
         numeric_features),
        ('cat', categorical_transformer,
         categorical_features)
    ]
)

# Использование
X_train_processed =
preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

## ◆ 14. Выбор метода заполнения

| Ситуация                           | Рекомендуемый метод      |
|------------------------------------|--------------------------|
| Числовые, нормальное распределение | Среднее                  |
| Числовые, с выбросами              | Медиана                  |
| Категориальные                     | Мода или константа       |
| Временные ряды                     | Forward/backward fill    |
| Сложные зависимости                | KNN или MICE             |
| Мало данных, MAR                   | Множественное заполнение |
| Много пропусков                    | Модельное заполнение     |

## ◆ 15. Выбор метода кодирования

| Ситуация                           | Метод                           |
|------------------------------------|---------------------------------|
| Порядковые (education, rating)     | Label Encoding                  |
| Номинальные, мало категорий (< 10) | One-Hot Encoding                |
| Номинальные, много категорий       | Frequency/Target Encoding       |
| Градиентный бустинг                | Label Encoding часто достаточно |
| Линейные модели                    | One-Hot обязательно             |
| Деревья                            | Label Encoding работает         |

## ◆ 16. Проверка результатов

```
# Проверить отсутствие пропусков
assert df.isnull().sum().sum() == 0

# Проверить типы данных
print(df.dtypes)

# Проверить диапазоны значений
print(df.describe())

# Проверить уникальные значения
for col in df.select_dtypes(include=['object']).columns:
    print(f"{col}: {df[col].nunique()} unique values")

# Визуализация распределений
import matplotlib.pyplot as plt
df.hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```

## ◆ 17. Частые ошибки

- ✗ **Data Leakage:** fit на всех данных до split
- ✗ **Неправильно:**

```
# Неправильно
imputer.fit(df)
X_train, X_test = train_test_split(df)

# Правильно
X_train, X_test = train_test_split(df)
imputer.fit(X_train)
X_train = imputer.transform(X_train)
X_test = imputer.transform(X_test)
```
- ✗ **Потеря информации:** удаление без анализа
- ✗ **Неправильное кодирование:** Label для номинальных
- ✗ **Игнорирование новых категорий** в test

## ◆ 18. Продвинутые техники

**Хэш-кодирование** для очень высокой кардинальности:

```
from sklearn.feature_extraction import FeatureHasher

hasher = FeatureHasher(
    n_features=10,
    input_type='string'
)
hashed =
hasher.transform(df['high_cardinality_col'])
```

**Эмбеддинги для категорий:**

```
# Обучить эмбеддинги (нейросетевой подход)
from tensorflow.keras.layers import Embedding

# Для категории с 1000 уникальных значений
embedding = Embedding(
    input_dim=1000,
    output_dim=50, # размерность эмбеддинга
    input_length=1
)
```

# 🎯 Визуализация данных в ML

 Январь 2026

## ◆ 1. Распределения признаков

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Гистограмма
sns.histplot(data=df, x='feature', bins=30,
kde=True)
plt.title('Распределение признака')
plt.show()

# Boxplot для нескольких категорий
sns.boxplot(data=df, x='category', y='value')
plt.title('Boxplot по категориям')
plt.show()

# Violin plot (комбинация box + density)
sns.violinplot(data=df, x='category', y='value')
plt.title('Violin plot')
plt.show()

# Несколько распределений на одном графике
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
for idx, col in enumerate(df.select_dtypes(include='number').columns
    sns.histplot(df[col], kde=True,
ax=axes[idx//2, idx%2])
    axes[idx//2, idx%2].set_title(f'Распределение
{col}')
plt.tight_layout()
plt.show()
```

## ◆ 2. Корреляционный анализ

```
# Heatmap корреляций
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True,
cmap='coolwarm',
center=0, vmin=-1, vmax=1,
square=True, linewidths=0.5)
plt.title('Матрица корреляций')
plt.show()

# Scatter matrix (pairplot)
sns.pairplot(df, hue='target', diag_kind='kde')
plt.show()

# Pandas scatter matrix
from pandas.plotting import scatter_matrix
scatter_matrix(df, alpha=0.5, figsize=(15, 15),
diagonal='kde')
plt.show()

# Топ коррелирующих признаков
def plot_top_correlations(df, target, n=10):
    corr = df.corr()
    [target].sort_values(ascending=False)
        top_corr = corr.head(n+1)[1:] # Исключаем сам
target

    plt.figure(figsize=(10, 6))
    top_corr.plot(kind='barh')
    plt.title(f'Топ {n} признаков по корреляции с
{target}')
    plt.xlabel('Корреляция')
    plt.tight_layout()
    plt.show()

plot_top_correlations(df, 'price', n=10)
```

### ◆ 3. Матрица ошибок и метрики классификации

```
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
from sklearn.metrics import classification_report
import numpy as np

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                               display_labels=['Class 0', 'Class 1'])
disp.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix')
plt.show()

# Нормализованная Confusion Matrix
cm_norm = confusion_matrix(y_test, y_pred,
normalize='true')
disp_norm =
ConfusionMatrixDisplay(confusion_matrix=cm_norm,
                       display_labels=['Class 0', 'Class 1'])
disp_norm.plot(cmap='Blues', values_format='%.2f')
plt.title('Normalized Confusion Matrix')
plt.show()

# Multiclass confusion matrix
from sklearn.datasets import load_iris
from sklearn.ensemble import
RandomForestClassifier
from sklearn.model_selection import
train_test_split

iris = load_iris()
X_train, X_test, y_train, y_test =
train_test_split(
    iris.data, iris.target, test_size=0.3,
    random_state=42
)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

cm_multi = confusion_matrix(y_test, y_pred)
disp_multi = ConfusionMatrixDisplay(
    confusion_matrix=cm_multi,
    display_labels=iris.target_names
)
disp_multi.plot(cmap='viridis')
plt.title('Multiclass Confusion Matrix')
plt.show()
```

### ◆ 4. ROC и PR кривые

```
from sklearn.metrics import roc_curve, auc,
precision_recall_curve
from sklearn.metrics import
average_precision_score

# ROC Curve
y_proba = clf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 5))

# ROC
plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2,
         linestyle='--', label='Random')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.grid(alpha=0.3)

# Precision-Recall Curve
precision, recall, _ =
precision_recall_curve(y_test, y_proba)
avg_precision = average_precision_score(y_test,
y_proba)

plt.subplot(1, 2, 2)
plt.plot(recall, precision, color='blue', lw=2,
         label=f'PR curve (AP = {avg_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

### ◆ 5. Важность признаков

```
# Feature importance для tree-based моделей
from sklearn.ensemble import
RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100,
random_state=42)
rf.fit(X_train, y_train)

# Получение важностей
importances = rf.feature_importances_
feature_names = X_train.columns # если DataFrame
indices = np.argsort(importances)[::-1]

# Визуализация
plt.figure(figsize=(12, 6))
plt.title('Feature Importances')
plt.bar(range(len(importances)), importances[indices])
plt.xticks(range(len(importances)),
[feature_names[i] for i in indices],
rotation=45, ha='right')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.tight_layout()
plt.show()

# Horizontal bar chart (для многих признаков)
top_n = 15
top_indices = indices[:top_n]
plt.figure(figsize=(10, 8))
plt.barh(range(top_n), importances[top_indices])
plt.yticks(range(top_n), [feature_names[i] for i in top_indices])
plt.xlabel('Importance')
plt.title(f'Top {top_n} Feature Importances')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

## ◆ 6. Кривые обучения

```
from sklearn.model_selection import learning_curve

# Вычисление learning curve
train_sizes, train_scores, val_scores =
learning_curve(
    estimator=clf,
    X=X_train,
    y=y_train,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Вычисление средних и std
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)

# Визуализация
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label='Training score',
         color='blue', marker='o')
plt.fill_between(train_sizes,
                 train_mean - train_std,
                 train_mean + train_std,
                 alpha=0.15, color='blue')

plt.plot(train_sizes, val_mean, label='Validation score',
         color='green', marker='s')
plt.fill_between(train_sizes,
                 val_mean - val_std,
                 val_mean + val_std,
                 alpha=0.15, color='green')

plt.xlabel('Training Set Size')
plt.ylabel('Accuracy Score')
plt.title('Learning Curves')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

## ◆ 7. Визуализация кластеризации

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Кластеризация
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)

# PCA для визуализации
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Scatter plot с кластерами
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
                      c=clusters, cmap='viridis',
                      alpha=0.6, edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=200,
            edgecolors='black', linewidths=2,
            label='Centroids')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('K-Means Clustering Visualization')
plt.colorbar(scatter, label='Cluster')
plt.legend()
plt.grid(alpha=0.3)
plt.show()

# Dendrogram для иерархической кластеризации
from scipy.cluster.hierarchy import dendrogram,
linkage

linkage_matrix = linkage(X, method='ward')

plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

## ◆ 8. Снижение размерности (PCA, t-SNE)

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Стандартизация
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# t-SNE
tsne = TSNE(n_components=2, random_state=42,
perplexity=30)
X_tsne = tsne.fit_transform(X_scaled)

# Визуализация
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# PCA
axes[0].scatter(X_pca[:, 0], X_pca[:, 1],
                 c=y, cmap='tab10', alpha=0.6)
axes[0].set_title(f'PCA (Explained Var: {pca.explained_variance_ratio_.sum():.2%})')
axes[0].set_xlabel('PC1')
axes[0].set_ylabel('PC2')
axes[0].grid(alpha=0.3)

# t-SNE
axes[1].scatter(X_tsne[:, 0], X_tsne[:, 1],
                 c=y, cmap='tab10', alpha=0.6)
axes[1].set_title('t-SNE')
axes[1].set_xlabel('Dimension 1')
axes[1].set_ylabel('Dimension 2')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# Scree plot (объясненная дисперсия)
pca_full = PCA()
pca_full.fit(X_scaled)

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(pca_full.explained_variance_ratio_)+1),
np.cumsum(pca_full.explained_variance_ratio_),
marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Scree Plot')
plt.grid(alpha=0.3)
plt.axhline(y=0.95, color='r', linestyle='--',
label='95% threshold')
```

```
plt.legend()
plt.show()
```

## ◆ 9. Residual plots для регрессии

```
from sklearn.linear_model import LinearRegression

# Обучение модели
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

# Residuals
residuals = y_test - y_pred

# Residual plot
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Residuals vs Predicted
axes[0, 0].scatter(y_pred, residuals, alpha=0.5)
axes[0, 0].axhline(y=0, color='r', linestyle='--')
axes[0, 0].set_xlabel('Predicted Values')
axes[0, 0].set_ylabel('Residuals')
axes[0, 0].set_title('Residuals vs Predicted')
axes[0, 0].grid(alpha=0.3)

# 2. Histogram of residuals
axes[0, 1].hist(residuals, bins=30,
edgecolor='black')
axes[0, 1].set_xlabel('Residuals')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].set_title('Distribution of Residuals')
axes[0, 1].grid(alpha=0.3)

# 3. Q-Q plot
from scipy import stats
stats.probplot(residuals, dist="norm",
plot=axes[1, 0])
axes[1, 0].set_title('Q-Q Plot')
axes[1, 0].grid(alpha=0.3)

# 4. Predicted vs Actual
axes[1, 1].scatter(y_test, y_pred, alpha=0.5)
axes[1, 1].plot([y_test.min(), y_test.max()],
[y_test.min(), y_test.max()],
'r--', lw=2)
axes[1, 1].set_xlabel('Actual Values')
axes[1, 1].set_ylabel('Predicted Values')
axes[1, 1].set_title('Predicted vs Actual')
axes[1, 1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

## ◆ 10. Сравнение моделей

```
from sklearn.model_selection import
cross_val_score
from sklearn.ensemble import
RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import
LogisticRegression

# Список моделей
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(probability=True)
}

# Cross-validation scores
results = {}
for name, model in models.items():
    scores = cross_val_score(model, X_train,
y_train,
cv=5,
scoring='accuracy')
    results[name] = scores

# Box plot сравнения
plt.figure(figsize=(12, 6))
plt.boxplot(results.values(),
labels=results.keys())
plt.ylabel('Accuracy Score')
plt.title('Model Comparison (5-Fold CV)')
plt.grid(axis='y', alpha=0.3)
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()

# Bar plot со средними значениями
means = {name: np.mean(scores) for name, scores in
results.items()}
stds = {name: np.std(scores) for name, scores in
results.items()}

plt.figure(figsize=(10, 6))
x = range(len(means))
plt.bar(x, means.values(), yerr=stds.values(),
capsize=5,
alpha=0.7, color='skyblue',
edgecolor='black')
plt.xticks(x, means.keys(), rotation=15)
plt.ylabel('Mean Accuracy')
plt.title('Mean Accuracy with Standard Deviation')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```

## ◆ 11. Временные ряды

```

import pandas as pd
import matplotlib.dates as mdates

# Временной ряд
dates = pd.date_range('2020-01-01', periods=365)
values = np.cumsum(np.random.randn(365)) + 100

ts_df = pd.DataFrame({'date': dates, 'value': values})
ts_df.set_index('date', inplace=True)

# Основной график
fig, axes = plt.subplots(3, 1, figsize=(14, 10))

# 1. Line plot
axes[0].plot(ts_df.index, ts_df['value'])
axes[0].set_title('Time Series Data')
axes[0].set_ylabel('Value')
axes[0].grid(alpha=0.3)

# 2. Скользящее среднее
ts_df['MA_7'] =
ts_df['value'].rolling(window=7).mean()
ts_df['MA_30'] =
ts_df['value'].rolling(window=30).mean()

axes[1].plot(ts_df.index, ts_df['value'],
             label='Original', alpha=0.5)
axes[1].plot(ts_df.index, ts_df['MA_7'],
             label='7-day MA', linewidth=2)
axes[1].plot(ts_df.index, ts_df['MA_30'],
             label='30-day MA', linewidth=2)
axes[1].set_title('Moving Averages')
axes[1].set_ylabel('Value')
axes[1].legend()
axes[1].grid(alpha=0.3)

# 3. Seasonal decomposition
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(ts_df['value'],
                                    model='additive',
                                    period=30)
axes[2].plot(ts_df.index, decomposition.trend)
axes[2].set_title('Trend Component')
axes[2].set_xlabel('Date')
axes[2].set_ylabel('Value')
axes[2].grid(alpha=0.3)

plt.tight_layout()
plt.show()

```

## ◆ 12. Интерактивная визуализация (Plotly)

```

import plotly.express as px
import plotly.graph_objects as go

# Scatter plot с hover
fig = px.scatter(df, x='feature1', y='feature2',
                  color='target', size='feature3',
                  hover_data=['feature4',
                  'feature5'],
                  title='Interactive Scatter Plot')
fig.show()

# Interactive confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

fig = go.Figure(data=go.Heatmap(
    z=cm,
    x=['Predicted 0', 'Predicted 1'],
    y=['Actual 0', 'Actual 1'],
    colorscale='Blues',
    text=cm,
    texttemplate=' %{text} ',
    textfont={"size": 16}
))
fig.update_layout(title='Interactive Confusion Matrix',
                  xaxis_title='Predicted',
                  yaxis_title='Actual')
fig.show()

# 3D scatter plot
fig = px.scatter_3d(df, x='feature1',
                     y='feature2', z='feature3',
                     color='target',
                     title='3D Scatter Plot')
fig.show()

# Интерактивная корреляционная матрица
corr = df.corr()
fig = go.Figure(data=go.Heatmap(
    z=corr.values,
    x=corr.columns,
    y=corr.columns,
    colorscale='RdBu',
    zmid=0
))
fig.update_layout(title='Interactive Correlation Matrix')
fig.show()

```

## ◆ 13. Best Practices

- Выбор цветов:** используйте colorblind-friendly палитры
- Размер графика:** достаточно большой для читаемости (10-14 inches)
- Подписи:** всегда добавляйте title, xlabel, ylabel
- Легенда:** размещайте так, чтобы не загораживала данные
- Gridlines:** используйте с alpha=0.3 для удобства
- DPI:** для сохранения используйте dpi=300
- Стиль:** используйте seaborn стили для согласованности

```

# Настройка стиля
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1.2)
sns.set_palette("husl")

```

```

# Сохранение с высоким разрешением
plt.savefig('plot.png', dpi=300,
bbox_inches='tight')

```

```

# Цветовые палитры
# Categorical: 'tab10', 'Set1', 'Set2', 'Set3'
# Sequential: 'Blues', 'Greens', 'Reds', 'viridis'
# Diverging: 'coolwarm', 'RdBu', 'RdYlGn'

```

## ◆ 14. Чек-лист визуализации

1.  Выбрать подходящий тип графика для данных
2.  Установить размер фигуры (figsize)
3.  Добавить заголовок (title)
4.  Подписать оси (xlabel, ylabel)
5.  Добавить легенду при необходимости
6.  Использовать подходящую цветовую схему
7.  Добавить grid для удобства чтения
8.  Проверить читаемость шрифтов
9.  Сохранить с высоким DPI
10.  Проверить на разных размерах экрана

July  
17

## Дата-специфичные признаки

17 Январь 2026

### ◆ 1. Суть

- Цель:** извлечь полезные паттерны из временных данных
- Зачем:** даты содержат скрытые закономерности (сезонность, тренды)
- Результат:** улучшение качества модели на 5-30%
- Применение:** продажи, логи, финансы, поведение пользователей

### ◆ 2. Базовое извлечение признаков

```
import pandas as pd

# Преобразование в datetime
df['date'] = pd.to_datetime(df['date'])

# Базовые признаки
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['hour'] = df['date'].dt.hour
df['minute'] = df['date'].dt.minute
df['dayofweek'] = df['date'].dt.dayofweek # 0=Пн, 6=Вс
df['dayofyear'] = df['date'].dt.dayofyear
df['quarter'] = df['date'].dt.quarter
df['weekofyear'] =
df['date'].dt.isocalendar().week

# День недели как название
df['day_name'] = df['date'].dt.day_name()

# Логические признаки
df['is_weekend'] = df['dayofweek'].isin([5, 6]).astype(int)
df['is_month_start'] =
df['date'].dt.is_month_start.astype(int)
df['is_month_end'] =
df['date'].dt.is_month_end.astype(int)
```

### ◆ 3. Циклические признаки

**Проблема:** месяц 12 и месяц 1 далеки числом, но близки по смыслу

**Решение:** синус и косинус для цикличности

```
import numpy as np

# Месяц (1-12)
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)

# День недели (0-6)
df['dow_sin'] = np.sin(2 * np.pi * df['dayofweek'] / 7)
df['dow_cos'] = np.cos(2 * np.pi * df['dayofweek'] / 7)

# Час (0-23)
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)

# День года (1-365)
df['doy_sin'] = np.sin(2 * np.pi * df['dayofyear'] / 365.25)
df['doy_cos'] = np.cos(2 * np.pi * df['dayofyear'] / 365.25)
```

### ◆ 4. Временные интервалы

```
# Разница между датами
df['days_since_start'] = (df['date'] - df['date'].min()).dt.days
df['days_until_end'] = (df['date'].max() - df['date']).dt.days
```

```
# Время с последнего события
df = df.sort_values('date')
df['days_since_last_event'] =
df['date'].diff().dt.days
```

```
# Скользящие окна
df['events_last_7d'] = df.rolling(window=7, on='date')[‘event’].sum()
df['events_last_30d'] = df.rolling(window=30, on='date')[‘event’].sum()
```

```
# Время до/после определенной даты
reference_date = pd.to_datetime('2024-01-01')
df['days_from_reference'] = (df['date'] - reference_date).dt.days
```

```
# Возраст данных
df['data_age_days'] = (pd.Timestamp.now() - df['date']).dt.days
```

## ◆ 5. Праздники и специальные дни

```
from datetime import date
import holidays

# Праздники России
ru_holidays = holidays.Russia()
df['is_holiday'] = df['date'].apply(
    lambda x: x.date() in ru_holidays
).astype(int)

# Расстояние до ближайшего праздника
def days_to_holiday(d):
    upcoming = [h for h in ru_holidays.keys() if h
    >= d.date()]
    if upcoming:
        return (min(upcoming) - d.date()).days
    return 365

df['days_to_holiday'] =
df['date'].apply(days_to_holiday)

# День до/после праздника
df['is_pre_holiday'] = (df['days_to_holiday'] ==
1).astype(int)
df['is_post_holiday'] =
df['is_holiday'].shift(1).fillna(0).astype(int)

# Пользовательские важные даты
special_dates = ['2024-01-01', '2024-12-31']
special_dates = pd.to_datetime(special_dates)
df['is_special'] =
df['date'].isin(special_dates).astype(int)
```

## ◆ 6. Бизнес-дни и рабочее время

```
from pandas.tseries.offsets import BDay

# Количество рабочих дней
df['business_days_from_start'] = df['date'].apply(
    lambda x: len(pd.bdate_range(df['date'].min(),
x))
)

# Это рабочий день?
df['is_business_day'] = df['date'].apply(
    lambda x: bool(len(pd.bdate_range(x, x)))
).astype(int)

# Рабочие часы
df['is_work_hours'] = (
    (df['hour'] >= 9) & (df['hour'] < 18) &
    (df['dayofweek'] < 5)
).astype(int)

# Части дня
def part_of_day(hour):
    if 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 22:
        return 'evening'
    else:
        return 'night'

df['part_of_day'] = df['hour'].apply(part_of_day)
```

## ◆ 7. Сезонность и тренды

```
# Сезон
def get_season(month):
    if month in [12, 1, 2]:
        return 'winter'
    elif month in [3, 4, 5]:
        return 'spring'
    elif month in [6, 7, 8]:
        return 'summer'
    else:
        return 'autumn'

df['season'] = df['month'].apply(get_season)

# Неделя месяца
df['week_of_month'] = (df['day'] - 1) // 7 + 1

# Декада месяца
df['decade_of_month'] = (df['day'] - 1) // 10 + 1

# Является ли середина месяца
df['is_mid_month'] = ((df['day'] >= 13) &
(df['day'] <= 17)).astype(int)

# Тренд (порядковый номер)
df['trend'] = range(len(df))

# Нормализованный тренд (0-1)
df['trend_normalized'] = df['trend'] /
df['trend'].max()
```

## ◆ 8. Агрегаты и статистики

```
# Группировка по периодам
# Среднее по дням недели
dow_mean = df.groupby('dayofweek')
['target'].transform('mean')
df['target_mean_by_dow'] = dow_mean

# Среднее по часам
hour_mean = df.groupby('hour')
['target'].transform('mean')
df['target_mean_by_hour'] = hour_mean

# Месячные агрегаты
df['year_month'] = df['date'].dt.to_period('M')
monthly_stats = df.groupby('year_month')
['target'].agg([
    'mean', 'std', 'min', 'max', 'count'
]).add_prefix('monthly_')
df = df.join(monthly_stats, on='year_month')

# Скользящие средние
df = df.sort_values('date')
df['ma_7d'] = df['target'].rolling(window=7,
min_periods=1).mean()
df['ma_30d'] = df['target'].rolling(window=30,
min_periods=1).mean()

# Экспоненциальное скользящее среднее
df['ema_7d'] = df['target'].ewm(span=7,
adjust=False).mean()
```

## ◆ 9. Взаимодействие признаков

```
# Комбинированные признаки
df['is_weekend_evening'] = (
    (df['is_weekend'] == 1) & (df['hour'] >= 18)
).astype(int)

df['is_workday_morning'] = (
    (df['is_weekend'] == 0) & (df['hour'] >= 6) &
(df['hour'] < 12)
).astype(int)

df['is_holiday_or_weekend'] = (
    (df['is_holiday'] == 1) | (df['is_weekend'] == 1)
).astype(int)

# Месяц + час (для seasonal patterns)
df['month_hour'] = df['month'].astype(str) + '_' +
df['hour'].astype(str)

# День недели + час
df['dow_hour'] = df['dayofweek'].astype(str) + '_' +
df['hour'].astype(str)

# Квартал + день недели
df['quarter_dow'] = df['quarter'].astype(str) + '_'
+ df['dayofweek'].astype(str)
```

## ◆ 10. Лаги и опережения

```
# Лаги (прошлые значения)
df = df.sort_values('date')
df['target_lag_1'] = df['target'].shift(1)
df['target_lag_7'] = df['target'].shift(7)
df['target_lag_30'] = df['target'].shift(30)

# Опережающие значения
df['target_lead_1'] = df['target'].shift(-1)

# Разность
df['target_diff_1'] = df['target'] -
df['target_lag_1']
df['target_diff_7'] = df['target'] -
df['target_lag_7']

# Процентное изменение
df['target_pct_change_1'] =
df['target'].pct_change(1)
df['target_pct_change_7'] =
df['target'].pct_change(7)

# Скользящая дисперсия
df['target_rolling_std_7'] =
df['target'].rolling(window=7).std()
df['target_rolling_std_30'] =
df['target'].rolling(window=30).std()
```

## ◆ 11. Специфичные паттерны

```
# Начало/конец периодов
df['is_year_start'] = (df['month'] == 1).astype(int)
df['is_year_end'] = (df['month'] == 12).astype(int)
df['is_quarter_start'] = df['date'].dt.is_quarter_start.astype(int)
df['is_quarter_end'] = df['date'].dt.is_quarter_end.astype(int)

# "Зарплатные" дни
df['is_payday'] = df['day'].isin([1, 2, 15, 16]).astype(int)

# Выходные перед/после праздников
df['is_long_weekend'] = (
    (df['is_weekend'] == 1) &
    ((df['days_to_holiday'] <= 2) | 
     (df['is_post_holiday'] == 1))
).astype(int)

# Дни распродаж (Black Friday, etc)
df['is_black_friday'] = (
    (df['month'] == 11) &
    (df['dayofweek'] == 4) &
    (df['week_of_month'] == 4)
).astype(int)

# Кастомные периоды (например, пандемия)
lockdown_start = pd.to_datetime('2020-03-01')
lockdown_end = pd.to_datetime('2020-06-01')
df['is_lockdown'] = (
    (df['date'] >= lockdown_start) & (df['date'] 
    <= lockdown_end)
).astype(int)
```

## ◆ 12. Лучшие практики

| Практика             | Описание                                               |
|----------------------|--------------------------------------------------------|
| <b>Циклические</b>   | Всегда используйте sin/cos для периодических признаков |
| <b>Масштаб</b>       | Нормализуйте тренды и временные интервалы              |
| <b>Лаги</b>          | Осторожно с утечкой данных (data leakage)              |
| <b>Валидация</b>     | Используйте временное разбиение (TimeSeriesSplit)      |
| <b>Null значения</b> | Заполняйте forward fill для временных рядов            |

### ⚠ Частые ошибки:

- Использование будущих данных в лагах
- Забыть про циклическую природу времени
- Игнорирование часовых поясов
- Не учитывать пропуски во времени

## ◆ 13. Полный пример

```
import pandas as pd
import numpy as np
from datetime import datetime

def create_date_features(df, date_col='date'):
    """Создает полный набор временных признаков"""
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col])

    # Базовые
    df['year'] = df[date_col].dt.year
    df['month'] = df[date_col].dt.month
    df['day'] = df[date_col].dt.day
    df['hour'] = df[date_col].dt.hour
    df['dayofweek'] = df[date_col].dt.dayofweek
    df['quarter'] = df[date_col].dt.quarter
    df['dayofyear'] = df[date_col].dt.dayofyear

    # Циклические
    df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
    df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)
    df['dow_sin'] = np.sin(2 * np.pi * df['dayofweek'] / 7)
    df['dow_cos'] = np.cos(2 * np.pi * df['dayofweek'] / 7)

    # Логические
    df['is_weekend'] = df['dayofweek'].isin([5, 6]).astype(int)
    df['is_month_start'] =
        df[date_col].dt.is_month_start.astype(int)
    df['is_month_end'] =
        df[date_col].dt.is_month_end.astype(int)

    # Интервалы
    df['days_since_start'] = (df[date_col] -
    df[date_col].min()).dt.days

    return df

# Использование
df_features = create_date_features(df,
    'transaction_date')
print(f"Создано {len(df_features.columns)} признаков")
```

## ◆ 14. Чек-лист

- [ ] Преобразовать в datetime формат
- [ ] Извлечь базовые признаки (год, месяц, день, час)
- [ ] Создать циклические признаки (sin/cos)
- [ ] Добавить is\_weekend, is\_holiday
- [ ] Вычислить временные интервалы
- [ ] Создать лаги для временных рядов
- [ ] Добавить агрегаты по периодам
- [ ] Проверить на утечку данных
- [ ] Использовать TimeSeriesSplit для валидации
- [ ] Визуализировать сезонность

### Объяснение заказчику:

«Даты — это не просто числа. Они содержат паттерны: люди покупают больше в выходные, продажи растут перед праздниками, трафик выше утром и вечером. Извлекая эти признаки, модель "понимает" время и делает точные прогнозы».



17 Январь 2026

## ◆ 1. Суть

- **DBSCAN** = Density-Based Spatial Clustering of Applications with Noise
- **Идея:** кластеры — это области высокой плотности
- **Не нужно** задавать число кластеров заранее
- **Находит** кластеры произвольной формы
- **Выделяет** выбросы (шум)

## ◆ 2. Ключевые понятия

| Понятие          | Описание                                           |
|------------------|----------------------------------------------------|
| $\epsilon$ (eps) | Радиус окрестности точки                           |
| MinPts           | Мин. кол-во точек в окрестности                    |
| Core point       | Точка с $\geq$ MinPts соседей в радиусе $\epsilon$ |
| Border point     | Не core, но в окрестности core                     |
| Noise point      | Ни core, ни border (выброс)                        |

## ◆ 3. Алгоритм

1. Для каждой непосещённой точки:
2. Найти все точки в радиусе  $\epsilon$
3. Если соседей  $<$  MinPts → шум (пока)
4. Если соседей  $\geq$  MinPts → новый кластер
5. Рекурсивно добавить все плотно-достижимые точки
6. Повторить для следующей непосещённой точки

## ◆ 4. Базовый код

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import numpy as np

# ОБЯЗАТЕЛЬНО масштабировать!
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# DBSCAN
dbscan = DBSCAN(
    eps=0.5,           # радиус окрестности
    min_samples=5,     # мин. точек для core point
    metric='euclidean',
    n_jobs=-1
)

# Обучение и предсказание
labels = dbscan.fit_predict(X_scaled)

# Количество кластеров (без шума)
n_clusters = len(set(labels)) - (1 if -1 in labels
else 0)
n_noise = list(labels).count(-1)

print(f"Кластеров: {n_clusters}")
print(f"Шум: {n_noise} точек
({100*n_noise/len(labels)}%)")

# -1 в labels означает шум
```

## ◆ 5. Ключевые параметры

| Параметр    | Описание           | Рекомендации                     |
|-------------|--------------------|----------------------------------|
| eps         | Радиус окрестности | Подбирать по k-distance graph    |
| min_samples | Мин. точек         | $2 \times \text{dim}$ или больше |
| metric      | Метрика расстояния | 'euclidean' (по умолчанию)       |
| algorithm   | Алгоритм поиска    | 'auto', 'ball_tree', 'kd_tree'   |

## ◆ 6. Подбор eps (k-distance graph)

```
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt

# Найти k ближайших соседей
k = 5 # обычно = min_samples
nbrs =
NearestNeighbors(n_neighbors=k).fit(X_scaled)
distances, indices = nbrs.kneighbors(X_scaled)

# Расстояния до k-го соседа
k_distances = distances[:, k-1]
k_distances = np.sort(k_distances)

# График
plt.figure(figsize=(10, 6))
plt.plot(k_distances)
plt.xlabel('Точки (отсортировано)')
plt.ylabel(f'Расстояние до {k}-го соседа')
plt.title('k-distance Graph')
plt.grid(True)
plt.show()

# Ищем "локоть" – это будет хороший eps
# Обычно eps ≈ расстояние в точке перегиба
```

## ◆ 7. Автоматический подбор eps

```
from kneed import KneeLocator

# Найти точку перегиба автоматически
k = 5
nbrs =
NearestNeighbors(n_neighbors=k).fit(X_scaled)
distances, _ = nbrs.kneighbors(X_scaled)
k_distances = np.sort(distances[:, k-1])

# Использовать KneeLocator
kneedle = KneeLocator(
    range(len(k_distances)),
    k_distances,
    curve='convex',
    direction='increasing'
)

# Оптимальный eps
optimal_eps = k_distances[kneedle.knee]
print(f"Рекомендуемый eps: {optimal_eps:.3f}")

# Применить
dbSCAN = DBSCAN(eps=optimal_eps, min_samples=k)
labels = dbSCAN.fit_predict(X_scaled)
```

## ◆ 8. Визуализация результатов

```
import matplotlib.pyplot as plt

# 2D визуализация
plt.figure(figsize=(12, 5))

# До кластеризации
plt.subplot(1, 2, 1)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1],
alpha=0.6)
plt.title('Исходные данные')

# После кластеризации
plt.subplot(1, 2, 2)

# Шум отдельно
noise = labels == -1
plt.scatter(X_scaled[noise, 0], X_scaled[noise, 1],
c='black', marker='x', s=50,
label='Шум')

# Кластеры
for cluster_id in set(labels):
    if cluster_id == -1:
        continue
    cluster_mask = labels == cluster_id
    plt.scatter(X_scaled[cluster_mask, 0],
    X_scaled[cluster_mask, 1],
    label=f'Кластер {cluster_id}',
    alpha=0.6)

plt.title(f'DBSCAN (кластеров: {n_clusters})')
plt.legend()
plt.show()
```

## ◆ 9. Анализ кластеров

```
# Размеры кластеров
unique, counts = np.unique(labels[labels != -1],
                           return_counts=True)
cluster_sizes = dict(zip(unique, counts))
print("Размеры кластеров:", cluster_sizes)

# Core samples (ядра кластеров)
core_samples_mask = np.zeros_like(labels,
                                   dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True

print(f"Core points: {core_samples_mask.sum()}")
print(f"Border points: {(labels != -1).sum() - core_samples_mask.sum()}")
print(f"Noise points: {(labels == -1).sum()}")

# Статистика по кластерам
import pandas as pd
for cluster_id in set(labels):
    if cluster_id == -1:
        continue
    cluster_points = X[labels == cluster_id]
    print(f"\nКластер {cluster_id}:")
    print(f"  Размер: {len(cluster_points)}")
    print(f"  Среднее:")
    print(f"    mean: {cluster_points.mean(axis=0)}")
    print(f"    Std: {cluster_points.std(axis=0)}")
```

## ◆ 10. Когда использовать

### ✓ Хорошо

- ✓ Кластеры произвольной формы
- ✓ Неизвестное число кластеров
- ✓ Данные с шумом и выбросами
- ✓ Кластеры разного размера
- ✓ Пространственные данные
- ✓ Обнаружение аномалий

### ✗ Плохо

- ✗ Кластеры сильно разной плотности
- ✗ Высокая размерность (curse of dimensionality)
- ✗ Нужна быстрая работа на больших данных
- ✗ Нет чёткого понятия плотности

## ◆ 11. Метрики качества

```
from sklearn.metrics import (
    silhouette_score,
    davies_bouldin_score,
    calinski_harabasz_score
)

# Только для кластеров (без шума)
mask = labels != -1
X_clustered = X_scaled[mask]
labels_clustered = labels[mask]

# Silhouette Score
if len(set(labels_clustered)) > 1:
    sil = silhouette_score(X_clustered,
                           labels_clustered)
    print(f"Silhouette Score: {sil:.3f}")

# Davies-Bouldin Index
db = davies_bouldin_score(X_clustered,
                           labels_clustered)
print(f"Davies-Bouldin Index: {db:.3f}")

# Calinski-Harabasz Score
ch = calinski_harabasz_score(X_clustered,
                             labels_clustered)
print(f"Calinski-Harabasz: {ch:.3f}")
```

## ◆ 12. Сравнение с K-means

| Аспект          | K-means             | DBSCAN                 |
|-----------------|---------------------|------------------------|
| Число кластеров | Нужно задать        | Находит сам            |
| Форма кластеров | Сферическая         | Произвольная           |
| Выбросы         | Включает в кластеры | Помечает как шум       |
| Скорость        | Быстро ( $O(n)$ )   | Медленнее ( $O(n^2)$ ) |
| Параметры       | K                   | eps, min_samples       |

## ◆ 13. Вариации DBSCAN

```
# HDBSCAN - иерархический DBSCAN
# Автоматически выбирает eps для разных плотностей
import hdbscan

clusterer = hdbscan.HDBSCAN(
    min_cluster_size=5,
    min_samples=5,
    metric='euclidean'
)

labels = clusterer.fit_predict(X_scaled)

# Вероятности принадлежности
probabilities = clusterer.probabilities_

# OPTICS - упорядоченная версия DBSCAN
from sklearn.cluster import OPTICS

optics = OPTICS(
    min_samples=5,
    max_eps=np.inf,
    metric='euclidean'
)

labels = optics.fit_predict(X_scaled)
```

## ◆ 14. Grid Search для параметров

```
from sklearn.metrics import silhouette_score

# Подбор параметров
best_score = -1
best_params = {}

eps_range = np.linspace(0.1, 2.0, 20)
min_samples_range = [3, 5, 7, 10]

for eps in eps_range:
    for min_samples in min_samples_range:
        dbSCAN = DBSCAN(eps=eps,
                        min_samples=min_samples)
        labels = dbSCAN.fit_predict(X_scaled)

        # Пропустить, если все точки - шум или
        # один кластер
        n_clusters = len(set(labels)) - (1 if -1
   in labels else 0)
        if n_clusters < 2:
            continue

        # Метрика только для кластеров
        mask = labels != -1
        if mask.sum() < 2:
            continue

        score = silhouette_score(X_scaled[mask],
                                  labels[mask])

        if score > best_score:
            best_score = score
            best_params = {'eps': eps,
                           'min_samples': min_samples}

print(f"Best params: {best_params}")
print(f"Best score: {best_score:.3f}")
```

## ◆ 15. Обработка шума

```
# Получить только кластеры (без шума)
clustered_mask = labels != -1
X_clustered = X[clustered_mask]
labels_clustered = labels[clustered_mask]

# Получить только шум
noise_mask = labels == -1
X_noise = X[negative_mask]

print(f"данных в кластерах: {len(X_clustered)}")
print(f"шумовых точек: {len(X_noise)}")

# Назначить шум к ближайшим кластерам
# (опционально)
from sklearn.neighbors import KNeighborsClassifier

if len(X_clustered) > 0 and len(X_noise) > 0:
    knn = KNeighborsClassifier(n_neighbors=1)
    knn.fit(X_clustered, labels_clustered)
    noise_labels = knn.predict(X_noise)

    # Объединить
    labels_all = labels.copy()
    labels_all[negative_mask] = noise_labels

print("Шум переназначен к кластерам")
```

## ◆ 16. Применение к реальным данным

```
# Пример: обнаружение географических кластеров
import pandas as pd

# Данные с координатами
data = pd.DataFrame({
    'latitude': [...],
    'longitude': [...]
})

# Масштабировать
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
coords_scaled =
scaler.fit_transform(data[['latitude',
'longitude']])

# DBSCAN с подобранными параметрами
# eps в радианах Земли (примерно)
# 0.01 ≈ 1.1 км
dbscan = DBSCAN(eps=0.01, min_samples=10)
data['cluster'] =
dbscan.fit_predict(coords_scaled)

# Визуализация на карте
import folium

m = folium.Map(location=[data['latitude'].mean(),
                         data['longitude'].mean()],
               zoom_start=10)

for cluster_id in set(data['cluster']):
    if cluster_id == -1:
        continue
    cluster_data = data[data['cluster'] ==
cluster_id]
    for _, row in cluster_data.iterrows():
        folium.CircleMarker(
            [row['latitude'], row['longitude']],
            radius=5,
            color=f"#{cluster_id*30%255:02x}5050"
        ).add_to(m)

m.save('clusters_map.html')
```

## ◆ 17. Чек-лист

- [ ] **ОБЯЗАТЕЛЬНО** масштабировать данные
- [ ] Использовать k-distance graph для подбора eps
- [ ]  $\text{min\_samples} \approx 2 \times \text{размерность данных}$
- [ ] Проверить долю шума (не должно быть > 50%)
- [ ] Визуализировать кластеры
- [ ] Проанализировать core/border/noise points
- [ ] Для разной плотности — использовать HDBSCAN
- [ ] Сравнить с K-means или другими методами
- [ ] Рассмотреть переназначение шума

### Объяснение заказчику:

«DBSCAN находит скопления точек в данных, автоматически определяя их количество и форму. Это как найти группы людей на площади — не важно, круглые они, вытянутые или причудливой формы. Одиночные люди считаются "шумом"».

## DCGAN и Conditional GAN

## ◆ 1. Основы GAN

GAN состоит из двух сетей: Generator (создаёт данные) и Discriminator (отличает реальные от поддельных).

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Пример кода для Основы GAN (vanilla GAN + Conditional GAN)
import torch
import torch.nn as nn
import torch.optim as optim

# Простые сети: полносвязный генератор и дискриминатор
latent_dim = 100
data_dim = 784 # пример: изображения 28x28

class Generator(nn.Module):
    def __init__(self, latent_dim, data_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, data_dim),
            nn.Tanh(), # данные в диапазоне [-1, 1]
        )
```

### DCGAN и Conditional GAN — Cheatsheet — 3 колонки

```
def forward(self, z):
    return self.net(z)

class Discriminator(nn.Module):
    def __init__(self, data_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(data_dim, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
        ) # без Sigmoid: используем BCEWithLogitsLoss()

    def forward(self, x):
        return self.net(x).view(-1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
G = Generator(latent_dim, data_dim).to(device)
D = Discriminator(data_dim).to(device)

criterion = nn.BCEWithLogitsLoss()
optimizer_G = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))

# dataloader должен отдавать real_images и fake_images
num_epochs = 5
for epoch in range(num_epochs):
    for real_images, _ in dataloader:
        real_images = real_images.view(real_images.size(0), -1)
        batch_size = real_images.size(0)

        # ----- Обучение дискриминатора -----
        z = torch.randn(batch_size, latent_dim)
        fake_images = G(z).detach() # оставляем в реальном диапазоне [-1, 1]
        real_labels = torch.ones(batch_size)
        fake_labels = torch.zeros(batch_size)

        D_real = D(real_images)
        D_fake = D(fake_images)
```

```
loss_D_real = criterion(D_real, real_labels)
loss_D_fake = criterion(D_fake, fake_labels)
loss_D = (loss_D_real + loss_D_fake) / 2

optimizer_D.zero_grad()
loss_D.backward()
optimizer_D.step()

# ----- Обучение генератора -----
z = torch.randn(batch_size, latent_dim)
fake_images = G(z)
G_labels = torch.ones(batch_size, dtype=torch.float32)
D_fake_for_G = D(fake_images)
loss_G = criterion(D_fake_for_G, G_labels)

optimizer_G.zero_grad()
loss_G.backward()
optimizer_G.step()

print(f"Epoch {epoch+1}: loss_D={loss_D}, loss_G={loss_G}")
```

## ◆ 2. DCGAN Architecture

Deep Convolutional GAN — использует свёрточные слои вместо fully-connected.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Wasserstein GAN (WGAN) Loss
import torch

def wasserstein_loss_d(real_output, fake_output):
    """Discriminator loss для WGAN"""
    return -(torch.mean(real_output)) - torch.mean(fake_output)

def wasserstein_loss_g(fake_output):
    """Generator loss для WGAN"""
    return -torch.mean(fake_output)

# Weight clipping для WGAN
def clip_weights(model, clip_value=0.01):
    for p in model.parameters():
        p.data.clamp_(-clip_value, clip_value)
```

## ◆ 3. Generator Network

Принимает случайный шум z, генерирует изображение через transpose convolutions.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Data Augmentation для GAN
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize(64),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])
```

## ◆ 4. Discriminator Network

Принимает изображение, выдаёт вероятность что оно реальное (0-1).

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Conditional GAN Implementation
import torch
import torch.nn as nn

class ConditionalGenerator(nn.Module):
    def __init__(self, latent_dim, num_classes):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, latent_dim)

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
            nn.Linear(1024, img_dim),
            nn.Tanh()
        )

    def forward(self, z, labels):
```

```

label_input = self.label_emb(labels)
x = torch.cat([z, label_input], dim=1)
return self.model(x)

# Conditional Discriminator
class ConditionalDiscriminator(nn.Module):
    def __init__(self):

        # Mode Collapse Detection and Prevention
        import numpy as np

        def check_modeCollapse(generated_samples, num_modes=10):
            """
            Проверяем разнообразие генерированных образцов
            """
            from sklearn.cluster import KMeans

            kmeans = KMeans(n_clusters=num_modes)
            kmeans.fit(generated_samples)

            # Считаем распределение по кластерам
            unique, counts = np.unique(kmeans.labels_, return_counts=True)
            distribution = counts / len(generated_samples)

            # Если больше 80% в одном кластере - вероятный mode
            if max(distribution) > 0.8:
                print("⚠ Warning: Possible mode collapse detected!")
                return True
            return False

        # Minibatch Discrimination для борьбы с mode collapse
        class MinibatchDiscrimination(nn.Module):
            def __init__(self, in_features, out_features, kernel_dim):
                super().__init__()
                self.T = nn.Parameter(torch.empty((in_features, out_features)))

            # GAN Evaluation Metrics
            import torch
            from scipy.linalg import sqrtm
            import numpy as np

            def calculate_fid(real_features, generated_features):
                """
                Fréchet Inception Distance (FID)
                Меньше = лучше (более похоже на реальные данные)
                """
                # Вычисляем mean и covariance
                mu1, sigma1 = real_features.mean(axis=0), np.cov(real_features, rowvar=False)
                mu2, sigma2 = generated_features.mean(axis=0), np.cov(generated_features, rowvar=False)

                # FID формула
                diff = mu1 - mu2
                covmean = sqrtm(sigma1.dot(sigma2))

                if np.iscomplexobj(covmean):
                    covmean = covmean.real

                fid = diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 * covmean)
                return fid

            def inception_score(generated_images, model, splits=10):
                """
                Inception Score (IS)
                Больше = лучше (более разнообразные и качественные)
                """
                pass

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, img_dim),
            nn.Tanh()
        )

        def forward(self, z, labels):
            label_input = self.label_emb(labels)
            x = torch.cat([z, label_input], dim=1)
            return self.model(x)

    # Conditional Discriminator
    class ConditionalDiscriminator(nn.Module):
        def __init__(self, latent_dim, num_classes, img_dim):
            super().__init__()
            self.label_emb = nn.Embedding(num_classes, num_classes)

            self.model = nn.Sequential(
                nn.Linear(latent_dim + num_classes, 128),
                nn.LeakyReLU(0.2),
                nn.Linear(128, 256),
                nn.BatchNorm1d(256),
                nn.LeakyReLU(0.2),
                nn.Linear(256, 512),
                nn.BatchNorm1d(512),
                nn.LeakyReLU(0.2),
                nn.Linear(512, img_dim),
                nn.Tanh()
            )

        def forward(self, z, labels):
            label_input = self.label_emb(labels)
            x = torch.cat([z, label_input], dim=1)
            return self.model(x)

    # Mode Collapse Detection and Prevention
    import numpy as np

    def check_modeCollapse(generated_samples, num_modes=10):
        """
        Проверяем разнообразие генерированных образцов
        """
        from sklearn.cluster import KMeans

        kmeans = KMeans(n_clusters=num_modes)
        kmeans.fit(generated_samples)

        # Считаем распределение по кластерам
        unique, counts = np.unique(kmeans.labels_, return_counts=True)
        distribution = counts / len(generated_samples)

        # Если больше 80% в одном кластере - вероятный mode
        if max(distribution) > 0.8:
            print("⚠ Warning: Possible mode collapse detected!")
            return True
        return False

    # Minibatch Discrimination для борьбы с mode collapse
    class MinibatchDiscrimination(nn.Module):
        def __init__(self, in_features, out_features, kernel_dim):
            super().__init__()
            self.T = nn.Parameter(torch.empty((in_features, out_features)))

        # GAN Evaluation Metrics
        import torch
        from scipy.linalg import sqrtm
        import numpy as np

        def calculate_fid(real_features, generated_features):
            """
            Fréchet Inception Distance (FID)
            Меньше = лучше (более похоже на реальные данные)
            """
            # Вычисляем mean и covariance
            mu1, sigma1 = real_features.mean(axis=0), np.cov(real_features, rowvar=False)
            mu2, sigma2 = generated_features.mean(axis=0), np.cov(generated_features, rowvar=False)

            # FID формула
            diff = mu1 - mu2
            covmean = sqrtm(sigma1.dot(sigma2))

            if np.iscomplexobj(covmean):
                covmean = covmean.real

            fid = diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 * covmean)
            return fid

        def inception_score(generated_images, model, splits=10):
            """
            Inception Score (IS)
            Больше = лучше (более разнообразные и качественные)
            """
            pass

```

## DCGAN и Conditional GAN — Cheatsheet — 3 колонки

```

# Получаем предска

# Conditional GAN Implementation
import torch
import torch.nn as nn

class ConditionalGenerator(nn.Module):
    def __init__(self, latent_dim, num_classes, img_dim):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, num_classes)

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, img_dim),
            nn.Tanh()
        )

    def forward(self, z, labels):
        label_input = self.label_emb(labels)
        x = torch.cat([z, label_input], dim=1)
        return self.model(x)

# Conditional Discriminator
class ConditionalDiscriminator(nn.Module):
    def __init__(self, latent_dim, num_classes, img_dim):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, num_classes)

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, img_dim),
            nn.Tanh()
        )

    def forward(self, z, labels):
        label_input = self.label_emb(labels)
        x = torch.cat([z, label_input], dim=1)
        return self.model(x)

# Mode Collapse Detection and Prevention
import numpy as np

def check_modeCollapse(generated_samples, num_modes=10):
    """
    Проверяем разнообразие генерированных образцов
    """
    from sklearn.cluster import KMeans

    kmeans = KMeans(n_clusters=num_modes)
    kmeans.fit(generated_samples)

    # Считаем распределение по кластерам
    unique, counts = np.unique(kmeans.labels_, return_counts=True)
    distribution = counts / len(generated_samples)

    # Если больше 80% в одном кластере - вероятный mode
    if max(distribution) > 0.8:
        print("⚠ Warning: Possible mode collapse detected!")
        return True
    return False

# Minibatch Discrimination для борьбы с mode collapse
class MinibatchDiscrimination(nn.Module):
    def __init__(self, in_features, out_features, kernel_dim):
        super().__init__()
        self.T = nn.Parameter(torch.empty((in_features, out_features)))

    # GAN Evaluation Metrics
    import torch
    from scipy.linalg import sqrtm
    import numpy as np

    def calculate_fid(real_features, generated_features):
        """
        Fréchet Inception Distance (FID)
        Меньше = лучше (более похоже на реальные данные)
        """
        # Вычисляем mean и covariance
        mu1, sigma1 = real_features.mean(axis=0), np.cov(real_features, rowvar=False)
        mu2, sigma2 = generated_features.mean(axis=0), np.cov(generated_features, rowvar=False)

        # FID формула
        diff = mu1 - mu2
        covmean = sqrtm(sigma1.dot(sigma2))

        if np.iscomplexobj(covmean):
            covmean = covmean.real

        fid = diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 * covmean)
        return fid

    def inception_score(generated_images, model, splits=10):
        """
        Inception Score (IS)
        Больше = лучше (более разнообразные и качественные)
        """
        pass

```

```

return fid

def inception_score(generated_images, model, splits=10):
    """
    Inception Score (IS)
    Больше = лучше (более разнообразные и качественные)
    """
    pass

# Получаем предскажи

# Conditional GAN Implementation
import torch
import torch.nn as nn

class ConditionalGenerator(nn.Module):
    def __init__(self, latent_dim, num_classes, 1
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, 1

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, img_dim),
            nn.Tanh()
        )

    def forward(self, z, labels):
        label_input = self.label_emb(labels)
        x = torch.cat([z, label_input], dim=1)
        return self.model(x)

# Conditional Discriminator
class ConditionalDiscriminator(nn.Module):
    def __init__(self, latent_dim, num_classes, img_dim):
        super().__init__()
        self.label_emb = nn.Embedding(num_classes, num_classes)

        self.model = nn.Sequential(
            nn.Linear(latent_dim + num_classes, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, img_dim),
            nn.Tanh()
        )

    def forward(self, z, labels):
        label_input = self.label_emb(labels)
        x = torch.cat([z, label_input], dim=1)
        return self.model(x)

# Mode Collapse Detection and Prevention
import numpy as np

def check_modeCollapse(generated_samples, num_modes=10):
    """
    Проверяем разнообразие генерированных образов
    """
    from sklearn.cluster import KMeans

    kmeans = KMeans(n_clusters=num_modes)
    kmeans.fit(generated_samples)

    # Считаем распределение по кластерам
    unique, counts = np.unique(kmeans.labels_, return_counts=True)
    distribution = counts / len(generated_samples)

    # Если больше 80% в одном кластере - вероятный mode
    if max(distribution) > 0.8:
        print("⚠ Warning: Possible mode collapse detected!")
        return True
    return False

# Minibatch Discrimination для борьбы с mode collapse
class MinibatchDiscrimination(nn.Module):
    def __init__(self, in_features, out_features, kernel_dim):
        super().__init__()
        self.T = nn.Parameter(torch.empty((in_features, out_features)))

    # GAN Evaluation Metrics
    import torch
    from scipy.linalg import sqrtm
    import numpy as np

    def calculate_fid(real_features, generated_features):
        """
        Fréchet Inception Distance (FID)
        Меньше = лучше (более похоже на реальные данные)
        """
        # Вычисляем mean и covariance
        mu1, sigma1 = real_features.mean(axis=0), np.cov(real_features, rowvar=False)
        mu2, sigma2 = generated_features.mean(axis=0), np.cov(generated_features, rowvar=False)

        # FID формула
        diff = mu1 - mu2
        covmean = sqrtm(sigma1.dot(sigma2))

        if np.iscomplexobj(covmean):
            covmean = covmean.real

        fid = diff.dot(diff) + np.trace(sigma1 + sigma2 - 2 * covmean)
        return fid

    def inception_score(generated_images, model, splits=10):
        """
        Inception Score (IS)
        Больше = лучше (более разнообразные и качественные)
        """
        pass

```

```

diff = mul - mu2
covmean = sqrtm(sigma1.dot(sigma2))

if np.iscomplexobj(covmean):
    covmean = covmean.real

fid = diff.dot(diff) + np.trace(sigma1)
return fid

def inception_score(generated_images,
                     **kwargs):
    """Inception Score (IS)
    Больше = лучше (более разнообразны)
    """
    # Получаем предсказания Inception
    with torch.no_grad():
        pred = model(generated_images)

    # Вычисляем IS
    split_scores = []
    for k in range(splits):
        part = pred[k * (len(preds) / splits): (k + 1) * (len(preds) / splits), :]
        py = part.mean(axis=0)
        scores = []
        for p in part:
            scores.append((p * (np.log(p) - np.log(py))).sum())
        split_scores.append(np.exp(np.mean(scores)))

    return np.mean(split_scores), np.std(split_scores)

minibatch_features = torch.sum(torch.exp(-abs_diffs), dim=0)
return torch.cat([x, minibatch_features], dim=1)

Sigmoid()

```

```

)
def forward(self, img, labels):
    label_input = self.label_emb(labels)
    x = torch.cat([img, label_input], dim=1)
    return self.model(x)

```

## ◆ 5. Training Process

Минимакс игра: G максимизирует  $\log(D(G(z)))$ , D максимизирует  $\log(D(x)) + \log(1-D(G(z)))$ .

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```

# Progressive Growing GAN
class ProgressiveGenerator(nn.Module):
    def __init__(self):
        super().__init__()
        self.blocks = nn.ModuleList([
            self._block(512, 512),
            self._block(512, 256),
            self._block(256, 128),
        ])

    def _block(self, in_ch, out_ch):
        return nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.LeakyReLU(0.2)
        )

```

## ◆ 6. Conditional GAN

Добавляет условие у (метку класса) к G и D для контролируемой генерации.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Wasserstein GAN (WGAN) Loss
import torch

def wasserstein_loss_d(real_output, fake_output):
    """Discriminator loss для WGAN"""
    return -(torch.mean(real_output)) - torch.mean(fake_output)

def wasserstein_loss_g(fake_output):
    """Generator loss для WGAN"""
    return -torch.mean(fake_output)

# Weight clipping для WGAN
def clip_weights(model, clip_value=0.01):
    for p in model.parameters():
        p.data.clamp_(-clip_value, clip_value)
```

## ◆ 7. Loss Functions

BCE loss для D и G. Опционально: Feature matching, Wasserstein loss.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Data Augmentation для GAN
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize(64),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])
```

## ◆ 8. Training Tips

Batch normalization, LeakyReLU, Adam optimizer, label smoothing.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Progressive Growing GAN
class ProgressiveGenerator(nn.Module):
    def __init__(self):
        super().__init__()
        self.blocks = nn.ModuleList([
            self._block(512, 512),
            self._block(512, 256),
            self._block(256, 128),
        ])

    def _block(self, in_ch, out_ch):
        return nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.LeakyReLU(0.2)
        )
```

## ◆ 9. Mode Collapse

Проблема когда G генерирует одинаковые изображения. Решение: minibatch discrimination.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Wasserstein GAN (WGAN) Loss
import torch

def wasserstein_loss_d(real_output, fake_output):
    """Discriminator loss для WGAN"""
    return -(torch.mean(real_output)) - torch.mean(fake_output)

def wasserstein_loss_g(fake_output):
    """Generator loss для WGAN"""
    return -torch.mean(fake_output)

# Weight clipping для WGAN
def clip_weights(model, clip_value=0.01):
    for p in model.parameters():
        p.data.clamp_(-clip_value, clip_value)
```

## ◆ 10. Evaluation Metrics

Inception Score, FID (Fréchet Inception Distance), визуальная оценка.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Data Augmentation для GAN
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize(64),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])
```

## ◆ 11. Applications

Генерация изображений, data augmentation, style transfer, super-resolution.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Progressive Growing GAN
class ProgressiveGenerator(nn.Module):
    def __init__(self):
        super().__init__()
        self.blocks = nn.ModuleList([
            self._block(512, 512),
            self._block(512, 256),
            self._block(256, 128),
        ])

    def _block(self, in_ch, out_ch):
        return nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.LeakyReLU(0.2)
        )
```

## ◆ 12. Code Implementation

PyTorch/TensorFlow реализация с примерами кода.

### Ключевые моменты:

- Концепция и принципы работы
- Математическая формулировка
- Алгоритмы и методы реализации
- Гиперпараметры и настройка
- Примеры кода на Python
- Преимущества и ограничения
- Практические применения
- Метрики и оценка качества

```
# Wasserstein GAN (WGAN) Loss
import torch

def wasserstein_loss_d(real_output, fake_output):
    """Discriminator loss для WGAN"""
    return -(torch.mean(real_output)) - torch.mean(fake_output)

def wasserstein_loss_g(fake_output):
    """Generator loss для WGAN"""
    return -torch.mean(fake_output)

# Weight clipping для WGAN
def clip_weights(model, clip_value=0.01):
    for p in model.parameters():
        p.data.clamp_(-clip_value, clip_value)
```



# Теория решений

Январь 2026

## ◆ 1. Суть

- **Decision Theory:** математическая основа принятия решений
- **Неопределенность:** выбор при неполной информации
- **Функция полезности:** количественная оценка последствий
- **Риск:** взвешивание возможных исходов
- **ML применение:** выбор моделей, порогов, действий

## ◆ 2. Основные компоненты

### Элементы задачи принятия решений:

- $A$ : множество действий (alternatives)
- $\Theta$ : множество состояний природы (states)
- $L(\theta, a)$ : функция потерь (loss function)
- $P(\theta)$ : распределение вероятностей состояний
- $R(a)$ : риск действия  $a$

### Общая формула риска:

$$R(a) = E[L(\theta, a)] = \sum_{\theta} P(\theta) L(\theta, a)$$

Цель:  $\min_{a \in A} R(a)$

## ◆ 3. Типы задач принятия решений

| Тип              | Информация             | Критерий                     |
|------------------|------------------------|------------------------------|
| Определенность   | Полная                 | Максимизация выгоды          |
| Риск             | Вероятности известны   | Минимизация ожидаемых потерь |
| Неопределенность | Вероятности неизвестны | Различные критерии           |
| Конфликт         | Противник              | Теория игр                   |

## ◆ 4. Функции потерь в ML

```

# Бинарная классификация
def zero_one_loss(y_true, y_pred):
    """0-1 loss: 0 если правильно, 1 если неправильно"""
    return (y_true != y_pred).astype(int)

def hinge_loss(y_true, y_score):
    """Hinge loss для SVM"""
    return np.maximum(0, 1 - y_true * y_score)

def log_loss(y_true, y_prob):
    """Логистическая функция потерь"""
    return -np.mean(y_true * np.log(y_prob) + (1 - y_true) * np.log(1 - y_prob))

# Регрессия
def squared_loss(y_true, y_pred):
    """L2 loss"""
    return (y_true - y_pred) ** 2

def absolute_loss(y_true, y_pred):
    """L1 loss"""
    return np.abs(y_true - y_pred)

def huber_loss(y_true, y_pred, delta=1.0):
    """Huber loss: комбинация L1 и L2"""
    error = y_true - y_pred
    is_small = np.abs(error) <= delta
    squared_loss = 0.5 * error ** 2
    linear_loss = delta * (np.abs(error) - 0.5 * delta)
    return np.where(is_small, squared_loss, linear_loss)

```

## ◆ 5. Bayesian Decision Theory

```
import numpy as np

# Байесовский классификатор
def bayesian_classifier(X, priors, likelihoods,
classes):
    """
    Выбирает класс с минимальным байесовским
    риском

    priors: P(C_i) - априорные вероятности классов
    likelihoods: P(X|C_i) - правдоподобия
    """

    posteriors = []
    for i, c in enumerate(classes):
        # Апостериорная вероятность: P(C_i|X) ∝
        # P(X|C_i) * P(C_i)
        posterior = likelihoods[i](X) * priors[i]
        posteriors.append(posterior)

    # Выбор класса с максимальной апостериорной
    # вероятностью
    return classes[np.argmax(posteriors)]

# Пример с Gaussian классами
from scipy.stats import norm

# Класс 0: N(0, 1), класс 1: N(2, 1)
priors = [0.6, 0.4] # P(C_0), P(C_1)
likelihoods = [
    lambda x: norm.pdf(x, 0, 1), # P(X|C_0)
    lambda x: norm.pdf(x, 2, 1) # P(X|C_1)
]

x_test = 1.0
predicted_class = bayesian_classifier(x_test,
priors, likelihoods, [0, 1])
print(f"Predicted class for x={x_test}:
{predicted_class}")
```

## ◆ 6. Минимаксный критерий

```
# Решение при полной неопределенности
def minimax_criterion(loss_matrix):
    """
    Минимаксный критерий: минимизация максимальных
    потерь

    loss_matrix[i, j]: потери для действия i при
    состоянии j
    """
    # Для каждого действия находим максимальные
    # потери
    max_losses = np.max(loss_matrix, axis=1)

    # Выбираем действие с минимальными из
    # максимальных потерь
    optimal_action = np.argmin(max_losses)

    return optimal_action,
max_losses[optimal_action]

# Пример: матрица потерь (3 действия × 4
# состояния)
loss_matrix = np.array([
    [10, 20, 15, 25], # действие 0
    [8, 18, 22, 12], # действие 1
    [15, 10, 18, 20] # действие 2
])

action, min_max_loss =
minimax_criterion(loss_matrix)
print(f"Optimal action: {action}, worst-case loss:
{min_max_loss}")

# Минимаксный критерий сожалений
def minimax_regret(loss_matrix):
    """
    Минимизация максимального сожаления"""
    # Для каждого состояния находим лучшее
    # действие
    min_losses = np.min(loss_matrix, axis=0)

    # Сожаление = потери - минимальные потери
    regret_matrix = loss_matrix - min_losses

    # Максимальное сожаление для каждого действия
    max_regrets = np.max(regret_matrix, axis=1)

    # Минимизируем максимальное сожаление
    return np.argmin(max_regrets), max_regrets
```

## ◆ 7. Критерий Лапласа

```
# Принцип недостаточного основания
def laplace_criterion(loss_matrix):
    """
    Критерий Лапласа: все состояния равновероятны
    Минимизация средних потерь
    """

    # Равномерное распределение по состояниям
    n_states = loss_matrix.shape[1]
    uniform_probs = np.ones(n_states) / n_states

    # Средние потери для каждого действия
    expected_losses = loss_matrix @ uniform_probs

    optimal_action = np.argmin(expected_losses)
    return optimal_action,
expected_losses[optimal_action]

# Критерий Байеса (общение с известными
# вероятностями)
def bayes_criterion(loss_matrix, probabilities):
    """
    Минимизация ожидаемых потерь при известных
    вероятностях
    """

    expected_losses = loss_matrix @ probabilities
    optimal_action = np.argmin(expected_losses)
    return optimal_action,
expected_losses[optimal_action]

# Пример
probabilities = np.array([0.3, 0.4, 0.2, 0.1])
action, exp_loss = bayes_criterion(loss_matrix,
probabilities)
print(f"Bayes optimal action: {action}, expected
loss: {exp_loss:.2f}")
```

## ◆ 8. Выбор порога классификации

```
from sklearn.metrics import confusion_matrix

def find_optimal_threshold(y_true, y_prob,
                           cost_matrix):
    """
    Находит оптимальный порог классификации

    cost_matrix: [[C(0,0), C(0,1)], # стоимости ошибок
                  [C(1,0), C(1,1)]]
    C(i,j) = стоимость предсказания j при истинном
    классе i
    """
    thresholds = np.linspace(0, 1, 100)
    costs = []

    for threshold in thresholds:
        y_pred = (y_prob >= threshold).astype(int)
        tn, fp, fn, tp = confusion_matrix(y_true,
   y_pred).ravel()

        # Общая стоимость
        cost = (tn * cost_matrix[0][0] +
                fp * cost_matrix[0][1] +
                fn * cost_matrix[1][0] +
                tp * cost_matrix[1][1])
        costs.append(cost)

    optimal_idx = np.argmin(costs)
    optimal_threshold = thresholds[optimal_idx]

    return optimal_threshold, costs[optimal_idx]

# Пример: детекция мошенничества
# Стоимости: [TN=0, FP=10, FN=1000, TP=0]
cost_matrix = [[0, 10], [1000, 0]]

optimal_threshold, min_cost =
find_optimal_threshold(
    y_true, y_prob, cost_matrix
)
print(f"Optimal threshold:
{optimal_threshold:.3f}, min cost: {min_cost}")
```

## ◆ 9. Sequential Decision Making

# Последовательное принятие решений

```
class SequentialDecisionMaker:
    def __init__(self, n_stages):
        self.n_stages = n_stages
        self.decisions = []
        self.outcomes = []

    def dynamic_programming(self, states, actions,
                           rewards, transitions):
        """
        Динамическое программирование для
        оптимальной политики

        V[s] = max_{a} (R(s, a) + γ * ∑ P(s'|s,a)
        * V[s'])
        """
        n_states = len(states)
        V = np.zeros(n_states) # value function
        policy = np.zeros(n_states, dtype=int)

        # Backward induction
        for stage in range(self.n_stages - 1, -1,
                            -1):
            V_new = np.zeros(n_states)

            for s in range(n_states):
                best_value = -np.inf
                best_action = 0

                for a in range(len(actions)):
                    # Expected value
                    value = rewards[s, a]
                    for s_next in range(n_states):
                        value += transitions[s, a,
   s_next] * V[s_next]

                    if value > best_value:
                        best_value = value
                        best_action = a

                V_new[s] = best_value
                policy[s] = best_action

            V = V_new

        return policy, V
```

# Пример: задача остановки

```
def optimal_stopping(values, costs):
    """
    Оптимальная остановка: когда прекратить поиск
    """
    n = len(values)
    V = np.zeros(n + 1)
```

```
# Backward induction
for i in range(n - 1, -1, -1):
    # Остановиться или продолжить
    stop_value = values[i]
    continue_value = V[i + 1] - costs[i]
    V[i] = max(stop_value, continue_value)

return V[0]
```

## ◆ 10. Multi-armed Bandit (краткое)

```
# Exploration vs Exploitation
class EpsilonGreedy:
    def __init__(self, n_arms, epsilon=0.1):
        self.n_arms = n_arms
        self.epsilon = epsilon
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)

    def select_arm(self):
        if np.random.random() < self.epsilon:
            # Explore: случайный выбор
            return np.random.randint(self.n_arms)
        else:
            # Exploit: лучшая текущая оценка
            return np.argmax(self.values)

    def update(self, arm, reward):
        self.counts[arm] += 1
        n = self.counts[arm]
        # Инкрементное обновление среднего
        self.values[arm] += (reward - self.values[arm]) / n

# UCB (Upper Confidence Bound)
class UCB:
    def __init__(self, n_arms, c=2):
        self.n_arms = n_arms
        self.c = c
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)
        self.t = 0

    def select_arm(self):
        self.t += 1
        # Сначала пробуем все руки
        if self.t <= self.n_arms:
            return self.t - 1

        # UCB formula
        ucb_values = self.values + self.c *
        np.sqrt(          np.log(self.t) / (self.counts + 1e-5)
        )                    return np.argmax(ucb_values)
```

## ◆ 11. Функция полезности

```
# Теория ожидаемой полезности (Expected Utility Theory)
def expected_utility(outcomes, probabilities,
utility_function):
    """
    EU = Σ P(outcome) * U(outcome)
    """
    return sum(p * utility_function(o)
               for o, p in zip(outcomes,
probabilities))

# Примеры функций полезности
def risk_averse_utility(x, gamma=0.5):
    """Неприятие риска (concave)"""
    return x ** gamma

def risk_seeking_utility(x, gamma=2):
    """Склонность к риску (convex)"""
    return x ** gamma

def risk_neutral_utility(x):
    """Нейтральность к риску (linear)"""
    return x

# Пример: выбор между гарантированной суммой и
лотерей
guaranteed = 50
lottery_outcomes = [0, 100]
lottery_probs = [0.5, 0.5]

# Для избегающего риска
utility_guaranteed =
risk_averse_utility(guaranteed)
utility_lottery = expected_utility(
    lottery_outcomes, lottery_probs,
risk_averse_utility
)

if utility_guaranteed > utility_lottery:
    print("Risk-averse: Choose guaranteed option")
else:
    print("Risk-averse: Choose lottery")
```

## ◆ 12. Применение в ML

- Выбор модели:** минимизация ожидаемых потерь
- Порог классификации:** учет стоимости ошибок
- Active Learning:** какие данные запросить
- Bandits:** A/B тестирование, рекомендации
- RL:** оптимальная политика действий
- Feature selection:** информационные критерии

## ◆ 13. Когда использовать

### ✓ Хорошо

- ✓ Неравные стоимости ошибок
- ✓ Несбалансированные классы
- ✓ Критические решения (медицина, финансы)
- ✓ Оптимизация бизнес-метрик
- ✓ Последовательные решения

### ✗ Плохо

- ✗ Равные стоимости всех ошибок
- ✗ Точность - единственная метрика
- ✗ Нет бизнес-контекста
- ✗ Простые задачи

## ◆ 14. Чек-лист

- [ ] Определить возможные действия
- [ ] Определить возможные состояния
- [ ] Задать функцию потерь/полезности
- [ ] Оценить вероятности (если известны)
- [ ] Выбрать критерий решения
- [ ] Учесть стоимость ошибок разных типов
- [ ] Рассмотреть последовательность решений
- [ ] Валидировать на реальных данных

### 💡 Объяснение заказчику:

«Теория решений — это математика правильного выбора. Вместо того чтобы просто максимизировать точность модели, мы учитываем реальные последствия: например, пропустить мошенническую транзакцию (потеря \$1000) хуже, чем ложная тревога (потеря \$10). Теория помогает найти баланс».