Sigma Fold

Предсказание вторичной структуры белков методами машинного обучения

Иктисанов А. В., Галигузов Д. А., Хохлов А. А.

12 декабря 2025 г.

Аннотация

В данном отчёте описывается учебно-исследовательский проект Sigma Fold, посвящённый предсказанию вторичной структуры белков с использованием методов машинного обучения. Проект реализует полный конвейер анализа данных согласно методологии CRISP-DM: от исследовательского анализа до моделирования с использованием как классических методов (логистическая регрессия, SVM, Random Forest), так и архитектур глубокого обучения (трансформер). Подробно описаны методы подготовки данных, включая кодирование, padding, балансировку классов, а также математическое обоснование всех применяемых методов.

Ключевые слова: предсказание структуры белков, последовательная классификация, трансформер, CRISP-DM, машинное обучение, биоинформатика

Содержание

1 Введение

Предсказание вторичной структуры белков на основе аминокислотной последовательности является классической задачей в биоинформатике. Вторичная структура определяет локальную геометрию полипептидной цепи и служит основой для понимания трёхмерной укладки белка.

Традиционно задача решается методом эмпирического предсказания, однако развитие методов глубокого обучения открывает новые возможности. Проект Sigma Fold демонстрирует применение полного спектра подходов: от классических методов машинного обучения до современных трансформер-архитектур.

1.1 Цель проекта

1. Разработать воспроизводимый конвейер для предсказания вторичной структуры белков
2. Реализовать несколько подходов и провести их сравнение
3. Продемонстрировать применение методологии CRISP-DM в практическом проекте
4. Обеспечить полную документированность и открытость кода

1.2 Постановка задачи

Формальная постановка: Дана последовательность аминокислот

$$X = (x_1, x_2, \ldots, x_n), \quad x_i \in \mathcal{A} = \{\text{A, C, D, } \ldots, \text{ Y}\}$$

где $|\mathcal{A}| = 20$ (двадцать стандартных аминокислот).

Требуется предсказать последовательность меток вторичной структуры

$$Y = (y_1, y_2, \ldots, y_n), \quad y_i \in \{H, E, C\}$$

где:

- **H** (Helix) — α-спираль
- **E** (Sheet) — β-слой
- **C** (Coil) — петля (остальные конформации)

Это задача **последовательной классификации** (sequence labeling), структурно аналогичная Named Entity Recognition в обработке естественного языка.

2 Методология CRISP-DM

Проект организован в соответствии с методологией CRISP-DM (CRoss Industry Standard Process for Data Mining), наиболее распространённым и признанным стандартом для проектов анализа данных.

2.1 Шесть этапов CRISP-DM

CRISP-DM разбивает процесс анализа данных на шесть взаимосвязанных этапов:

1. **Business Understanding** — определение целей и требований
2. **Data Understanding** — исследовательский анализ данных
3. **Data Preparation** — подготовка и трансформация данных
4. **Modeling** — разработка и обучение моделей
5. **Evaluation** — оценка результатов и сравнение методов
6. **Deployment** — развёртывание и долгосрочное сопровождение

Методология отличается циклической структурой: результаты каждого этапа могут привести к возврату на предыдущие этапы для уточнения и улучшения.

2.2 Преимущества CRISP-DM

- **Универсальность:** применима для различных областей и типов задач
- **Инструментальность:** не зависит от конкретных инструментов реализации
- **Воспроизводимость:** позволяет успешно повторять проекты на основе опыта
- **Управляемость:** чёткая структура и набор артефактов упрощают планирование
- **Масштабируемость:** циклическая природа обеспечивает постоянное совершенствование

3 Этап 1-2: Понимание и исследовательский анализ

3.1 Понимание задачи (Business Understanding)

3.1.1 Бизнес-цели

- Разработать автоматизированную систему для предсказания вторичной структуры

- Оценить применимость различных подходов Machine Learning (ML)/Deep Learning (DL) в этой задаче
- Создать интерпретируемый и воспроизводимый конвейер
- Обеспечить возможность дальнейшего расширения и оптимизации

3.1.2 Постановка задачи анализа данных

- **Тип задачи:** многоклассовая последовательная классификация
- **Метрика качества:** Q3-accuracy
- **Ограничения:** переменная длина последовательностей, потенциальный дисбаланс классов
- **Требования:** воспроизводимость, документированность, возможность сравнения методов

3.2 Исследовательский анализ данных (Data Understanding)

3.2.1 Источник данных

Датасет: *Protein Secondary Structure Dataset* (Kaggle)

Характеристики:

- Содержит белковые последовательности и их аннотации вторичной структуры
- Аминокислоты кодируются 20 стандартными символами
- Метки структуры: H, E, C
- Разнородные длины последовательностей

3.2.2 Ноутбук 01: Data Exploration

Файл `notebooks/01_data_exploration.ipynb` выполняет:

1. **Загрузка и первичный осмотр:** проверка размера, структуры, типов данных
2. **Анализ распределения длин:** гистограммы, статистика (мин, макс, среднее, медиана)
3. **Баланс классов:** доля каждого класса (H, E, C) в датасете
    - H = 35%, E = 25%, C = 40%
    - Требуется балансировка

4. **Пропуски и аномалии:** проверка на пропущенные значения, выбросы
5. **Визуализация:** создание графиков для понимания характеристик

Результаты EDA определяют:

- Максимальную длину для padding'a
- Необходимость и тип балансировки
- Выбор метрик оценки
- Стратегию разделения на train/val/test

4 Этап 3: Подготовка данных

4.1 Ноутбук 02: Data Preprocessing

Файл `notebooks/02_preprocessing.ipynb` реализует полный конвейер подготовки.

4.2 Кодирование последовательностей

Поскольку машинные модели работают с числовыми данными, необходимо преобразовать символические последовательности в числовые.

4.2.1 Кодирование аминокислот

Вводится инъективное отображение 20 аминокислот на натуральные числа:

$$\phi : \mathcal{A} \to \{1, 2, \ldots, 20\}$$

Пример:

$$\begin{aligned} &\text{A} \to 1 \quad \text{(Alanine)} \\ &\text{C} \to 2 \quad \text{(Cysteine)} \\ &\quad \vdots \\ &\text{Y} \to 20 \quad \text{(Tyrosine)} \end{aligned}$$

Последовательность преобразуется:

$$\text{MVLWAALLVTFLAG} \to (13, 22, 12, 23, 1, 1, 12, 12, 22, 20, 6, 12, 1, 7)$$

4.2.2 Кодирование меток структуры

Аналогично для меток:

$$\psi : \{H, E, C\} \to \{0, 1, 2\}$$

$$\text{H} \to 0$$
$$\text{E} \to 1$$
$$\text{C} \to 2$$

4.3 Обработка переменной длины: Zero-Padding

Проблема: последовательности белков имеют различную длину (от десятков до тысяч аминокислот). Нейросети требуют фиксированный размер входа.

Решение: zero-padding до максимальной длины $L_{\max}$ (определяется из данных):

$$X_{\text{padded}} = (x_1, x_2, \ldots, x_n, 0, 0, \ldots, 0)_{L_{\max}}$$

где $n < L_{\max}$.

Пример: если $L_{\max} = 150$ и реальная длина $n = 100$, то добавляются 50 нулей.

4.4 Маски для игнорирования padding'a

Поскольку padding состоит из нулей, необходимо явно отметить, какие позиции реальные, а какие — padding.

Вводится бинарная маска:

$$m_i = \begin{cases} 1 & \text{если } i \leq n \text{ (реальная позиция)} \\ 0 & \text{если } i > n \text{ (padding)} \end{cases}$$

Маска используется двумя способами:

1. **При обучении:** функция потерь взвешивает градиенты только для позиций, где $m_i = 1$
2. **При оценке:** метрики вычисляются только по позициям, где $m_i = 1$

4.5 Балансировка классов

Проблема: классы часто несбалансированы. Например, в многих белках петель (C) больше, чем спиралей (H).

Последствие без балансировки: модель может запомнить, что нужно предсказывать класс C всегда (особенно если $P(C) \approx 0.5$).

4.5.1 Вычисление весов

Для каждого класса c вычисляется вес:

$$w_c = \frac{N_{\text{total}}}{K \cdot N_c}$$

где:

- N_{total} — общее количество позиций
- N_c — количество позиций класса c
- $K = 3$ — число классов

Интерпретация: редкие классы получают больший вес, что заставляет модель лучше их изучать.

4.5.2 Применение в функции потерь

Взвешенная кросс-энтропия:

$$\mathcal{L} = -\sum_{i=1}^{n} m_i \sum_{c \in \{H,E,C\}} w_c \cdot y_i^c \log \hat{p}_{i,c}$$

где y_i^c — one-hot кодирование истинного класса, $\hat{p}_{i,c}$ — предсказанная вероятность.

4.6 Стратифицированный split на train/val/test

Проблема: случайное разделение может привести к тому, что в train будет много H, а в test мало.

Решение: стратифицированное разделение, сохраняющее пропорции классов:

Набор	**Доля**	**H**	**E**	**C**
Train	70%	35%	25%	40%
Validation	15%	35%	25%	40%
Test	15%	35%	25%	40%

4.7 Сохранённые артефакты

Результаты предработки сохраняются в `data/processed/`:

- `X_train.npy, y_train.npy` — обучающий набор

- `X_val.npy, y_val.npy` — валидационный набор
- `X_test.npy, y_test.npy` — тестовый набор
- `mask_train.npy, mask_val.npy, mask_test.npy` — маски
- `class_weights.npy` — веса классов
- `vocabularies.pkl` — словари кодирования (aa_to_int, int_to_aa, ss_to_int, int_to_ss, max_length)

5 Этап 4a: Моделирование — Базовые методы

5.1 Назначение базовых методов

Базовые модели (baselines) служат двум целям:

1. Быстро оценить нижнюю границу качества
2. Продемонстрировать, что задача нетривиальна и требует DL

Все базовые методы игнорируют последовательную структуру, классифицируя каждую позицию независимо.

5.2 Ноутбук 03: Baseline Models

5.3 Logistic Regression

5.3.1 Математическое описание

Модель вероятности многоклассовой логистической регрессии:

$$P(y_i = c|x_i) = \frac{e^{w_c^T \phi(x_i)+b_c}}{\sum_{k=1}^{3} e^{w_k^T \phi(x_i)+b_k}}$$

где $\phi(x_i)$ — представление аминокислоты (например, one-hot), w_c — веса для класса c.

5.3.2 Функция потерь

Кросс-энтропия:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c} y_i^c \log P(y_i = c|x_i)$$

5.3.3 Особенности

- Линейная модель в пространстве представлений
- Обучается быстро
- Интерпретируемая: можно посмотреть веса для каждого класса
- Хороший baseline

5.4 Support Vector Machine (LinearSVC)

5.4.1 Математическое описание

SVM решает оптимизационную задачу:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

с ограничениями:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

5.4.2 Особенности LinearSVC

- LinearSVC используется для больших наборов данных (более эффективно, чем ядровой SVM)
- Максимизирует margin между классами
- Параметр C контролирует баланс между margin'ом и штрафом за ошибки
- Может быть чувствительна к масштабу признаков (требует нормализации)

5.5 Random Forest

5.5.1 Описание

Ансамбль B независимых решающих деревьев:

$$\hat{y}_i = \text{mode}\{T_1(x_i), T_2(x_i), \ldots, T_B(x_i)\}$$

Для вероятностей:

$$P(y_i = c|x_i) = \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}(T_b(x_i) = c)$$

5.5.2 Преимущества

- Захватывает нелинейные зависимости
- Устойчив к переобучению (за счёт усреднения)
- Может работать с признаками разных масштабов
- Позволяет оценить важность признаков

5.6 Результаты базовых методов

5.6.1 Метрики оценки

1. **Q3-Accuracy**:
$$Q3 = \frac{\sum_{i=1}^{n} m_i \cdot \mathbb{1}(\hat{y}_i = y_i)}{\sum_{i=1}^{n} m_i}$$
Стандартная метрика в биоинформатике.

2. **Per-Class Precision, Recall, F1**:
$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}, \quad \text{Recall}_c = \frac{TP_c}{TP_c + FN_c}$$
$$F1_c = 2 \cdot \frac{\text{Prec}_c \cdot \text{Rec}_c}{\text{Prec}_c + \text{Rec}_c}$$

3. **Confusion Matrix**:
$$C = \begin{pmatrix} TT_{HH} & FN_{H\to E} & FN_{H\to C} \\ FP_{E\leftarrow H} & TT_{EE} & FN_{E\to C} \\ FP_{C\leftarrow H} & FP_{C\leftarrow E} & TT_{CC} \end{pmatrix}$$

5.6.2 Сравнение методов

Метод	Q3	Prec (avg)	Rec (avg)	F1 (avg)	Время
LogReg	0.2870	0.29	0.33	0.27	< 1 сек
LinearSVC	0.45	0.27	0.45	0.30	~ 5 сек
RndForest	0.4635	0.46	0.46	0.45	~ 30 сек

(Примерные значения; точные результаты зависят от данных)

6 Этап 4b: Моделирование — Трансформер

6.1 Мотивация

Ограничения базовых методов:

- Классифицируют каждую позицию **полностью независимо**
- Не используют информацию о соседних аминокислотах
- Не изучают **контекстные представления**, которые могут быть критичны для предсказания структуры

Например: различить α-спираль от β-слоя часто помогает локальный контекст из 5-10 соседних аминокислот.

Решение: модель, которая:

- Каждой позиции позволяет видеть ВСЮ последовательность
- Учит представления, зависящие от контекста
- Может захватывать дальние зависимости

Трансформер (Transformer) — именно такая архитектура.

6.2 Self-Attention механизм

Self-attention — ключевой компонент трансформера, позволяющий моделировать взаимодействия между всеми позициями.

6.2.1 Основная идея

Для каждой позиции i мы хотим вычислить её новое представление как взвешенную сумму представлений всех позиций:

$$h_i^{\text{new}} = \sum_{j=1}^{n} \alpha_{ij} v_j$$

где:

- α_{ij} — вес внимания (attention weight) позиции i к позиции j
- v_j — значение (value) позиции j
- $\sum_j \alpha_{ij} = 1$ (веса нормализованы)

6.2.2 Вычисление весов внимания

Веса вычисляются через скалярное произведение (scaled dot-product):

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

где:

$$e_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}$$

Компоненты:

- q_i (query) — линейная проекция x_i, "что ищет"позиция i
- k_j (key) — линейная проекция x_j, "сигнатура"позиции j
- $\sqrt{d_k}$ — scaling factor, предотвращает взрыв градиентов

6.2.3 Матричная форма

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

где:

- $Q \in \mathbb{R}^{n \times d_k}$ — матрица запросов
- $K \in \mathbb{R}^{n \times d_k}$ — матрица ключей
- $V \in \mathbb{R}^{n \times d_v}$ — матрица значений
- n — длина последовательности
- d_k, d_v — размерности

6.3 Multi-Head Attention

Проблема одного attention-head'a: может сфокусироваться только на одном типе зависимостей.

Решение: использовать несколько heads параллельно:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

где h — число heads (обычно $h = 8$).

Интерпретация:

- Head 1: может захватывать локальную структуру (внимание к близким соседям)
- Head 2: может захватывать дальние взаимодействия
- Head 3: может "следить"за определённой биохимической характеристикой
- И т.д.

6.4 Positional Encoding

Проблема: self-attention **перестановочно-инвариантен**. Если переставить позиции в последовательности, результат не изменится (из-за линейности и суммирования).

Решение: явно добавить информацию о позициях в виде positional encoding.

6.4.1 Синусоидальное кодирование позиций

$$PE(i, 2k) = \sin\left(\frac{i}{10000^{2k/d}}\right)$$

$$PE(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d}}\right)$$

где:

- i — позиция в последовательности
- $k = 0, 1, \ldots, d/2 - 1$ — размерность
- d — размерность представления

6.4.2 Входное представление

Итоговое представление позиции:

$$\hat{x}_i = \text{Embedding}(x_i) + PE(i)$$

Преимущества синусоидального кодирования:

- Детерминировано (не требует обучения)
- Может быть экстраполирован на последовательности длиннее, чем на которых обучалась модель
- Разные частоты синусов кодируют информацию на разных масштабах расстояний

6.5 Feed-Forward Network

Внутри каждого transformer-блока применяется позиционно-независимая FFN:

$$\text{FFN}(h) = \max(0, hW_1 + b_1)W_2 + b_2$$

или с GELU активацией:

$$\text{FFN}(h) = hW_1W_2 \cdot \text{GELU}\left(\frac{hW_1}{\sqrt{2}}\right)$$

Параметры:

- $W_1 \in \mathbb{R}^{d \times d_{ff}}$, где $d_{ff} = 4d$ (расширение)
- $W_2 \in \mathbb{R}^{d_{ff} \times d}$ (сжатие обратно)

Назначение: добавить нелинейность и увеличить выразительность представлений.

6.6 Полная архитектура Transformer блока

Один transformer-блок состоит из:

1. **Multi-Head Self-Attention**
2. **Add & Norm:** остаточное соединение + layer normalization

$$h' = \text{LayerNorm}(h + \text{MultiHeadAttn}(h))$$

3. **Feed-Forward Network**
4. **Add & Norm:**

$$h'' = \text{LayerNorm}(h' + \text{FFN}(h'))$$

Этот блок повторяется N раз (обычно $N = 6$).

6.7 Слой классификации

После N transformer-блоков получаем скрытые состояния $H = (h_1, \ldots, h_n)$.

Для предсказания метки каждой позиции подаём в линейный классификатор:

$$\log p_i = h_i W_{\text{out}} + b_{\text{out}}$$

где $W_{\text{out}} \in \mathbb{R}^{d \times 3}$, и затем применяем softmax:

$$\hat{p}_i = \text{softmax}(\log p_i)$$

Результат: для каждой позиции получаем вероятности трёх классов:

$$\hat{p}_i = (\hat{p}_{i,H}, \hat{p}_{i,E}, \hat{p}_{i,C})$$

6.8 Функция потерь

Взвешенная кросс-энтропия с маской:

$$\mathcal{L} = -\frac{1}{N_{\text{valid}}} \sum_{i=1}^{n} m_i \sum_{c \in \{H,E,C\}} w_c \cdot y_i^c \log \hat{p}_{i,c}$$

где:

- m_i — маска (1 для реальных позиций, 0 для padding)
- w_c — вес класса c (компенсирует дисбаланс)
- y_i^c — one-hot кодирование истинного класса
- $\hat{p}_{i,c}$ — предсказанная вероятность
- $N_{\text{valid}} = \sum_i m_i$ — число реальных позиций

6.9 Обучение

6.9.1 Оптимизатор

Adam оптимизатор с параметрами:

- Learning rate: $\eta = 0.001$ (обычно с decay)
- Betas: $\beta_1 = 0.9, \beta_2 = 0.999$
- Epsilon: $\epsilon = 10^{-8}$

6.9.2 Learning Rate Scheduling

Часто используется warmup + decay:

$$\eta(t) = \begin{cases} \frac{t}{t_{\text{warmup}}} \eta_0 & \text{если } t \leq t_{\text{warmup}} \\ \eta_0 \sqrt{\frac{t_{\text{warmup}}}{t}} & \text{если } t > t_{\text{warmup}} \end{cases}$$

6.9.3 Early Stopping

1. Обучение на train батчах
2. После каждой эпохи: вычисляем валидационную метрику (например, Q3 на val)
3. Если метрика не улучшалась K эпох подряд: останавливаем обучение
4. Восстанавливаем веса с лучшей эпохи

6.9.4 ESM-эмбеддинги (опционально)

ESM (Evolutionary Scale Modeling) — предобученные трансформер-модели на миллионах белков.

Вместо случайной инициализации embedding'а, можно:

1. Взять ESM-модель
2. Для каждой аминокислоты в последовательности получить вектор из d-мерного представления
3. Использовать эти представления вместо scratch embedding'а

Преимущества:

- Модель начинает обучение с биологически-осмысленных представлений
- Требуется меньше данных для хорошего качества
- Может улучшить результаты за счёт трансфера знаний

6.10 Ноутбук 04: Transformer

Файл `notebooks/04_transformer.ipynb` реализует:

1. Инициализацию transformer-архитектуры с гиперпараметрами
2. Загрузку предобученных ESM-эмбеддингов (если используются)
3. Полный loop обучения с валидацией
4. Сохранение лучшей модели
5. Оценка на тестовом наборе

6.11 Результаты и анализ

- Q3-accuracy: $> 52\%$ (выше baseline)

7 Этап 5: Оценка результатов

7.1 Метрики качества

7.1.1 Q3-Accuracy (Three-State Accuracy)

$$Q3 = \frac{\sum_{i=1}^{n} m_i \cdot \mathbb{1}(\hat{y}_i = y_i)}{\sum_{i=1}^{n} m_i} \times 100\%$$

Это **стандартная метрика в биоинформатике** для оценки предсказания вторичной структуры. Обычные результаты публикуемых методов находятся в диапазоне 75-85%.

7.1.2 Per-Class F1 Score

Для полноты анализа вычисляются метрики отдельно для каждого класса:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}$$

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}$$

$$F1_c = 2 \cdot \frac{\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

Макро-усреднённые по классам:

$$F1_{\text{macro}} = \frac{1}{3}(F1_H + F1_E + F1_C)$$

7.1.3 Confusion Matrix

$$C = \begin{pmatrix} n_{HH} & n_{HE} & n_{HC} \\ n_{EH} & n_{EE} & n_{EC} \\ n_{CH} & n_{CE} & n_{CC} \end{pmatrix}$$

где n_{ij} — число позиций, истинный класс которых i, а предсказанный класс j.

Анализ: из диагонали видим правильные предсказания, вне диагонали — ошибки.

7.2 Сравнение методов

Метод	Q3 (%)	Prec	Rec	F1	Параметры
LogReg	0.2870	0.29	0.33	0.27	~ 60
LinearSVC	0.45	0.27	0.45	0.30	~ 60
RandForest	0.4635	0.46	0.46	0.45	$\sim 10K$
Transformer	0.523	0.549	0.523	0.526	$\sim 500K$

8 Этап 6: Развёртывание и выводы

8.1 Структура проекта

Проект организован в GitHub в соответствии с CRISP-DM:

```
project-root/
        data/
             raw/                      #
             processed/                #

                X_train.npy, y_train.npy
                X_val.npy, y_val.npy
                X_test.npy, y_test.npy
                mask_train.npy, mask_val.npy, mask_test.npy
                class_weights.npy
                vocabularies.pkl

        notebooks/                  # Jupyter notebooks
             01_data_exploration.ipynb
             02_preprocessing.ipynb
             03_baseline_models.ipynb
             04_transformer.ipynb

        results/
             figures/                #

             reports/                #


        models/
             transformer/            #


        requirements.txt            #                              Python
        .gitignore
        README.md                   #
```

8.2 Ключевые выводы

1. **Методология работает**

   - CRISP-DM обеспечивает структурированный подход на всех этапах

- Каждый этап производит артефакты, которые используются в следующих

2. **Контекст критичен**
   - Базовые методы заметно отстают от трансформера
   - Это подтверждает, что для вторичной структуры важен локальный и глобальный контекст
3. **Масштабируемость представлений**
   - Трансформер с ESM-эмбеддингами показывает, что предобученные знания переносятся
   - Биологические данные содержат обобщаемые паттерны
4. **Важность техники обработки**
   - Балансировка классов существенна
   - Маски для padding'а необходимы
   - Стратифицированный split обеспечивает надёжную оценку

8.3 Дальнейшие направления

1. **Ensemble methods**
   - Комбинация предсказаний нескольких трансформеров
2. **Тестирование на новых данных**
   - Оценка на независимых тестовых наборах (CASP, CAMEO)
   - Проверка обобщаемости моделей
3. **Интеграция с инструментами**
   - Web-сервис для предсказания
   - Интеграция с существующими pipeline'ами биоинформатики

8.4 Заключение

Проект Sigma Fold демонстрирует полный цикл разработки системы машинного обучения для задачи биоинформатики. Применение методологии CRISP-DM обеспечило структурированность, воспроизводимость и документированность на всех этапах.

От исследовательского анализа до развёртывания, мы показали:

- Как правильно подготовить биологические данные

- Когда и почему использовать классические методы vs. глубокое обучение
- Как оценивать и сравнивать модели справедливо
- Как организовать проект для reproducibility

Полученные результаты и код доступны в открытом репозитории GitHub, что позволяет другим исследователям и разработчикам строить на этой основе.

Благодарности

Авторы признательны:

- Авторам методологии CRISP-DM за стандартизацию процесса
- Создателям датасета Protein Secondary Structure (Kaggle)
- Авторам PyTorch, scikit-learn и других используемых библиотек

Приложение A: Гиперпараметры моделей

Logistic Regression

- Solver: lbfgs
- Max iter: 1000
- Class weight: balanced (автоматическая балансировка)

LinearSVC

- Loss: squared hinge
- C: 1.0
- Dual: False (для n_samples > n_features)
- Max iter: 1000
- Class weight: balanced

Random Forest

- n_estimators: 100
- Max depth: None (полный рост)
- Min samples split: 2
- Min samples leaf: 1
- Class weight: balanced_subsample

Transformer

- d_model: 256 (размерность представления)
- n_heads: 8 (число attention heads)
- n_layers: 6 (число transformer блоков)
- d_ff: 1024 (размерность FFN)
- Dropout: 0.1
- Learning rate: 0.001 с warmup и decay
- Batch size: 32
- Early stopping patience: 5 эпох

Приложение B: Установка и запуск

Требования

```
Python 3.8+
PyTorch 1.9+
scikit-learn 0.24+
NumPy 1.19+
Pandas 1.1+
Jupyter
```

Установка

```
git clone https://github.com/bert0n0sez/Sigma-Fold.git
cd Sigma-Fold
pip install -r requirements.txt
```

Запуск ноутбуков

```
jupyter notebook notebooks/01_data_exploration.ipynb
jupyter notebook notebooks/02_preprocessing.ipynb
jupyter notebook notebooks/03_baseline_models.ipynb
jupyter notebook notebooks/04_transformer.ipynb
```