Библиотека машинного обучения для системы программирования PascalABC.NET

Михалкович С.С.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

miks@sfedu.ru

Доклад на XXXII научной конференции
«Современные информационные технологии: тенденции и перспективы развития (СИТО 2026)»
(Ростов-на-Дону, 16-18 апреля 2026 г.)

О проекте ML PascalABC.NET

Код библиотеки в основном сгенерирован с использованием
генеративных моделей (ChatGPT и др.) за **3 месяца:** 26 января – 16 апреля 2026 г.

При этом:

- архитектура системы, декомпозиция и ключевые решения определялись автором;
- все компоненты проходили отбор, доработку и интеграцию вручную;
- финальная ответственность за корректность и качество результата — на стороне автора.

Это не «автоматическая генерация кода».

Это — **разработка при поддержке ИИ** (**AI-assisted development**).

Анализ обучения ML сегодня

🔴 **Экосистема**

- Python — стандарт де-факто. Краткий код. Язык № 1 для ML.
- Динамическая типизация → ошибки на этапе выполнения ☹
- Огромное количество внешних библиотек ☺☹

🔵 **Данные**

- Обучение на типовых датасетах (Boston, Titanic)
- Дефицит качественных русскоязычных датасетов

🟢 **Образование**

- Документация преимущественно на английском
- Появление школьных олимпиад по ИИ

🟣 **Архитектура**

- Отсутствие прозрачности моделей
- Нет единой модели данных (NumPy vs Pandas)

Цели создания библиотеки ML PascalABC.NET

Образование

- Промышленный ML для студентов и школьников
- Преимущества компилятора – раннее обнаружение ошибок ☺
- Код проще, чем scikit-learn

Архитектура

- Чистая система без внешних зависимостей
- Интерфейсы как контракты
- Полный контроль над обучением моделей

Данные

- Узнаваемые **русскоязычные датасеты**, заточенные под обучение (от простых до сложных)
- Репозиторий учебных данных

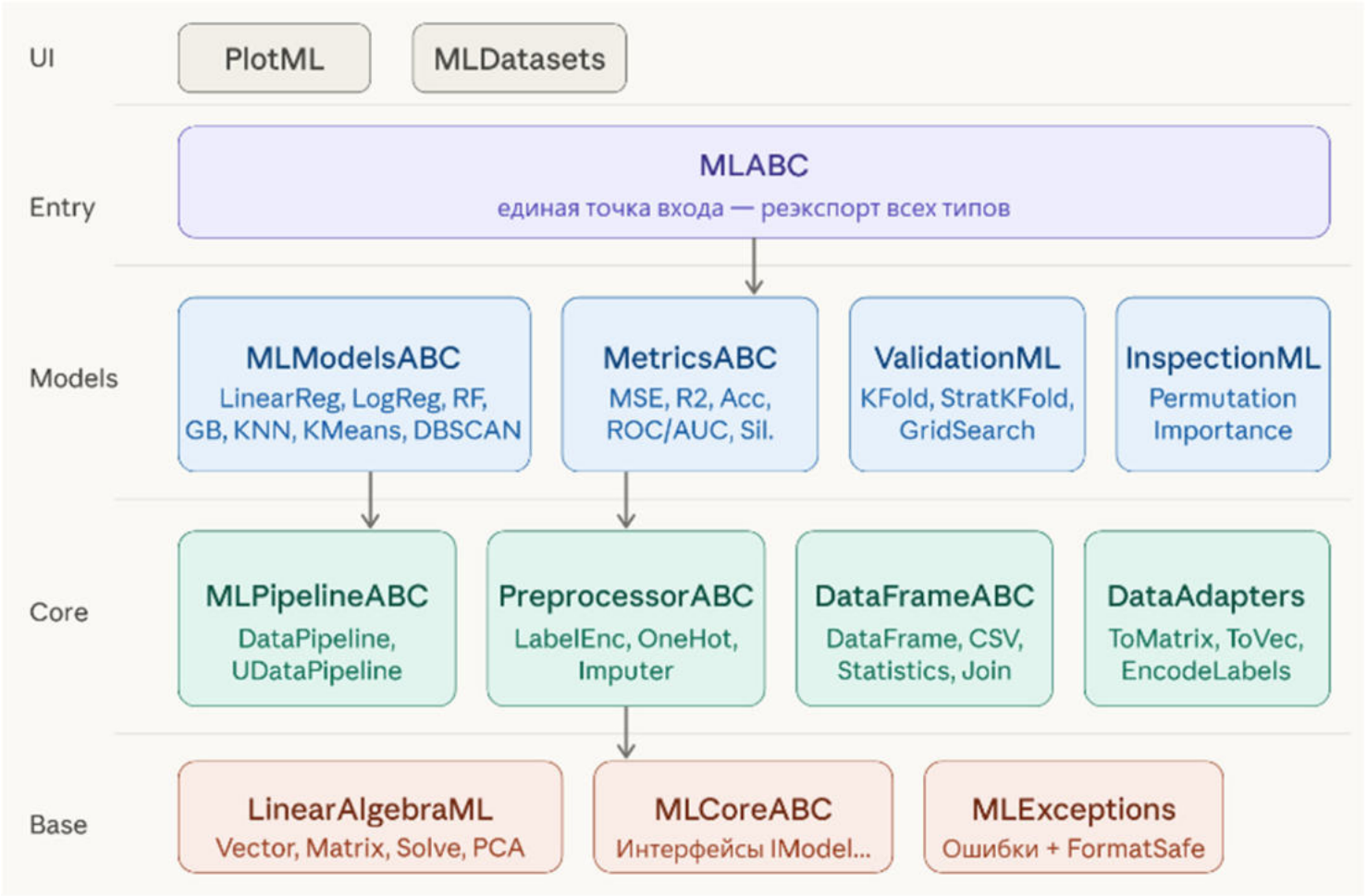
Экосистема

- Интеграция в школьные олимпиады по ИИ
- Альтернативный подход в обучении ML

Исследование

- Разработка через AI (**ChatGPT**). Анализ преимуществ и ограничений

Архитектура проекта

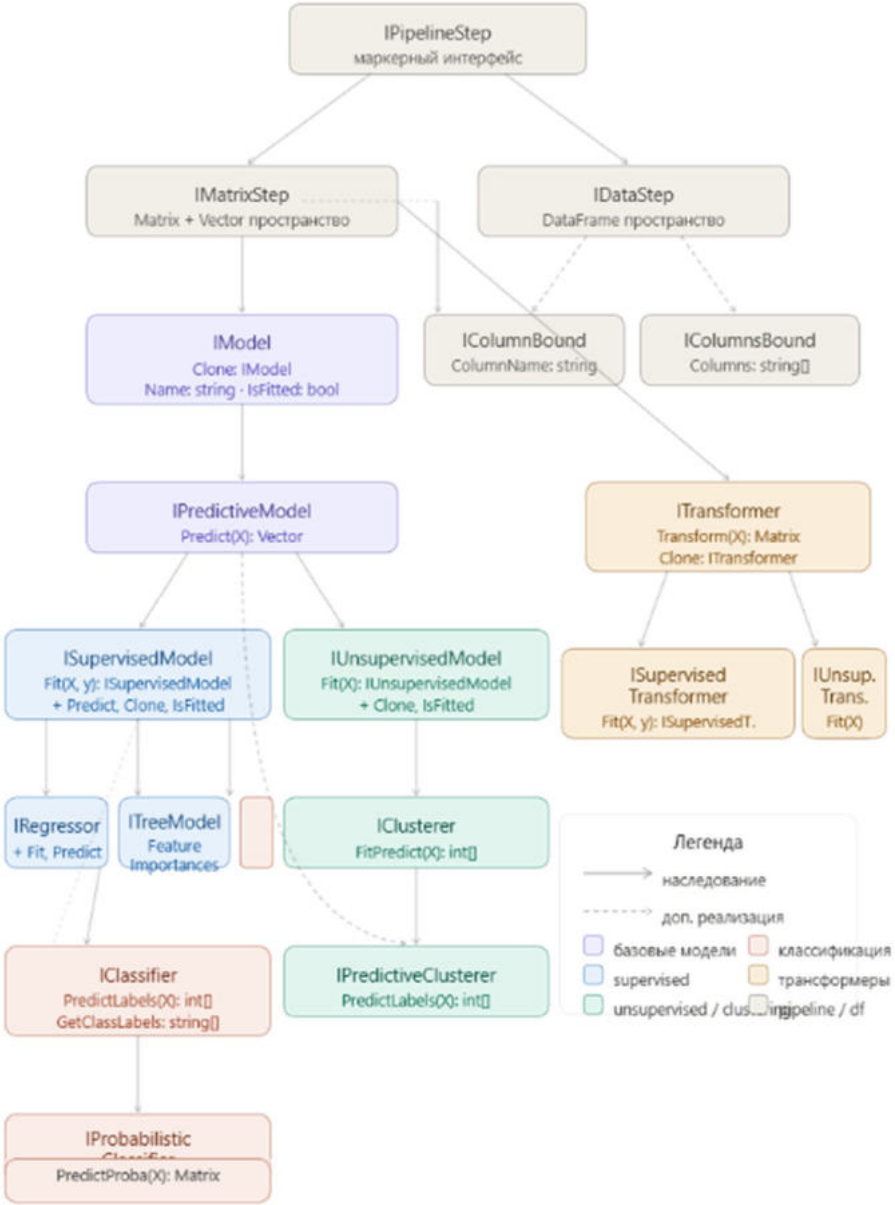


Генерация диаграмм: Anthropic Claude

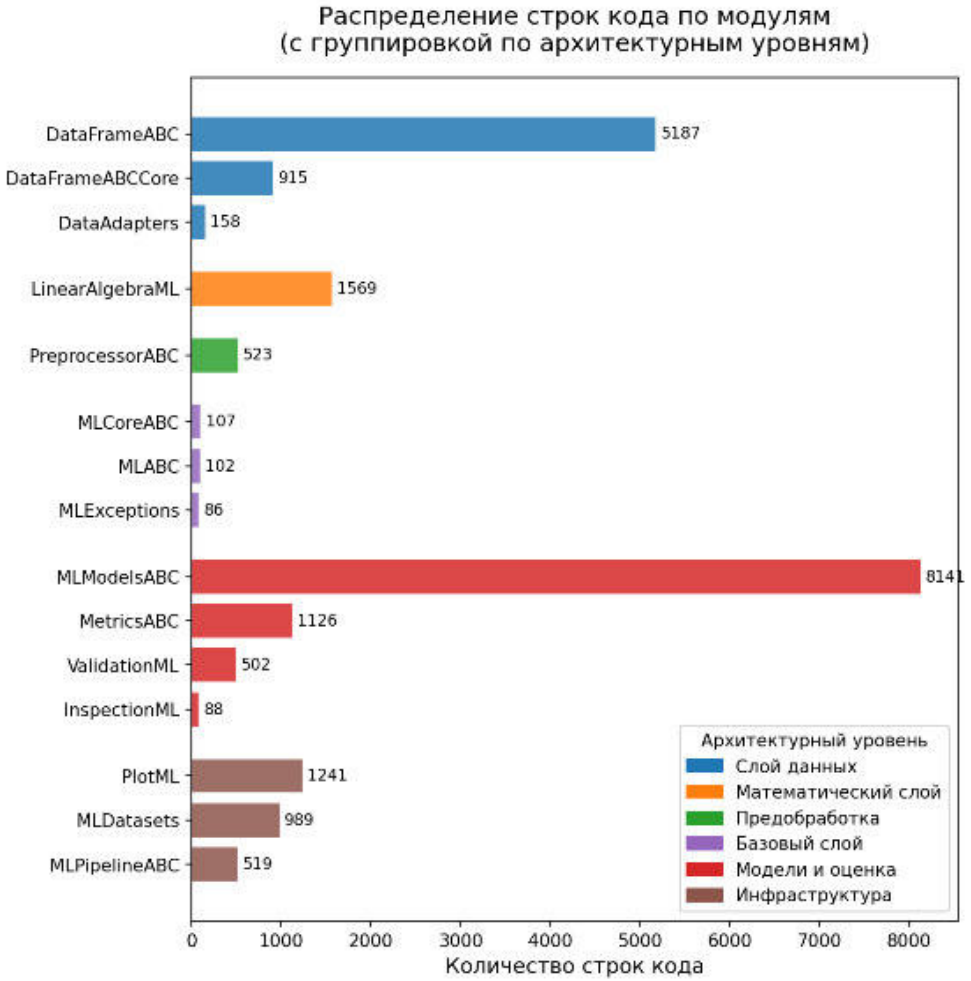
Диаграмма иерархии интерфейсов

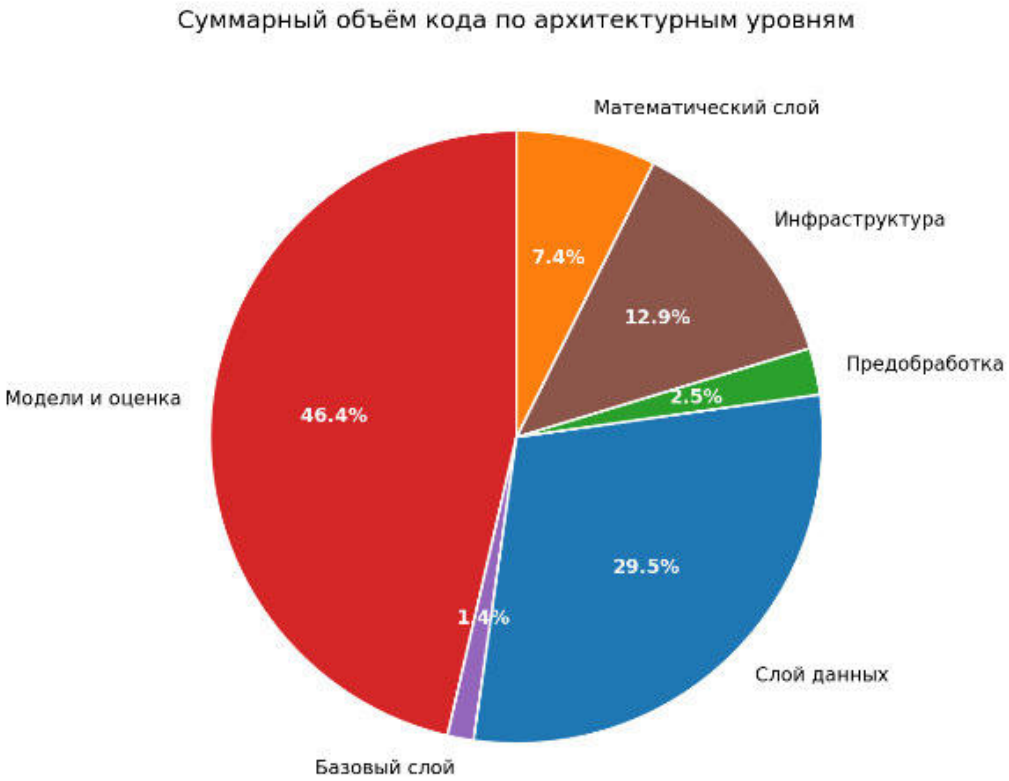
- Все классы моделей, трансформеров реализуют ряд **интерфейсов**, которые задают **контракт**
- Контракт проверяется на этапе компиляции или (реже) на этапе выполнения
- Наличие иерархии интерфейсов позволяет легко расширять библиотеку новыми моделями и трансформерами

Генерация диаграмм: Anthropic Claude



Размер и структура проекта

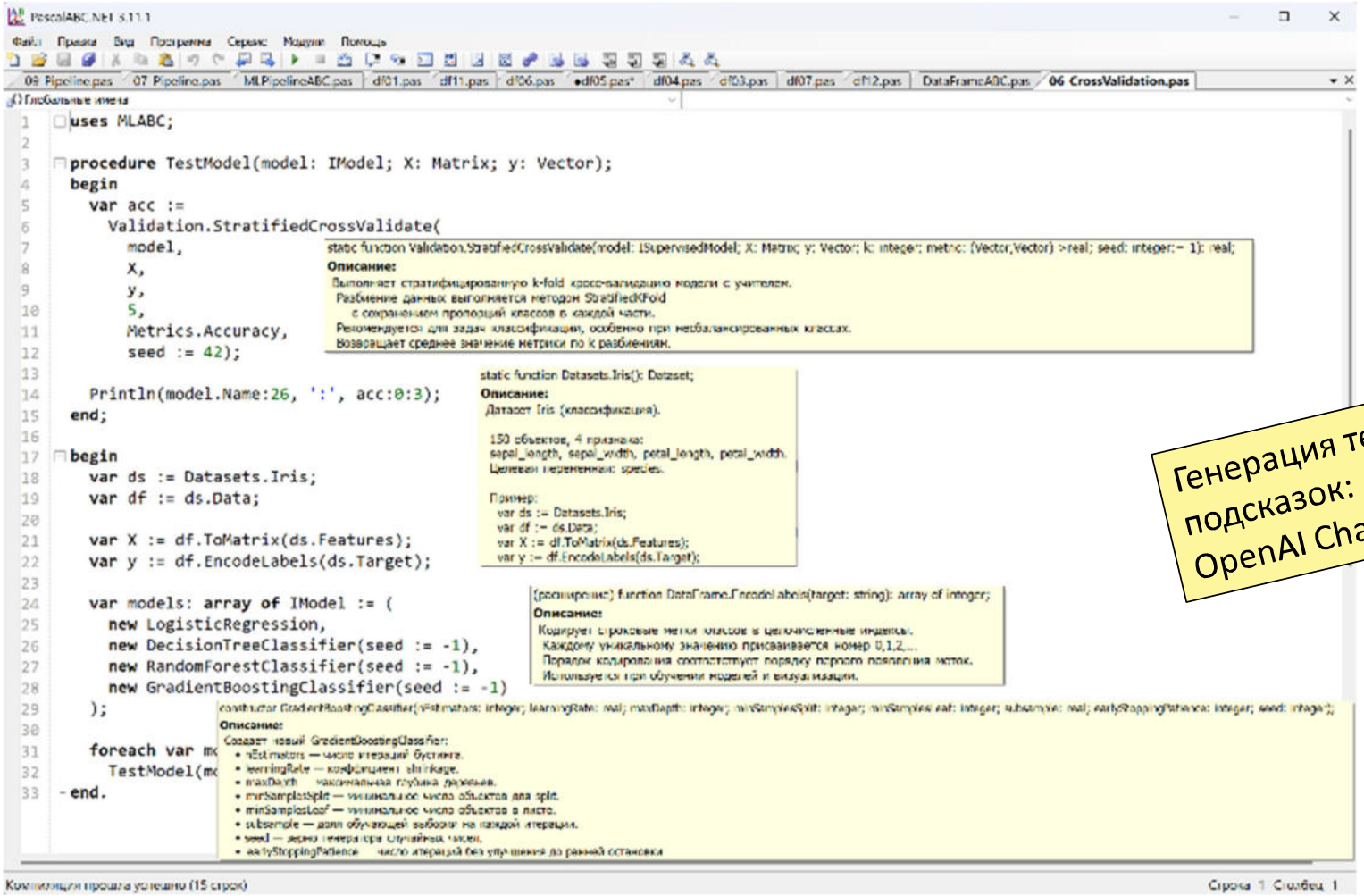




Генерация диаграмм: DeepSeek через Python-скрипт

Объём кода — 21,253 строк

Русскоязычные подсказки Intellisense



Генерация текста подсказок: OpenAI ChatGPT

Модуль DataFrameABC

- Колоночное хранение данных (column-store)
- Фиксированные типы: Int / Float / Str / Bool
- Предсказуемые операции (Filter, Select, GroupBy)
- Без неявных преобразований типов. Проверка типов на этапе компиляции

PascalABC.NET

```
uses DataFrameABC;

begin
  var df := DataFrame.FromCsvText('''
  name,age,score
  Alice,20,85
  Bob,22,90
  Charlie,21,78
  ''');

  df.Filter(row -> row.Int('age') >= 21).Print;
  df.Select(['name', 'score']).Print;
  df.Mean('score').Println;
  df.GroupBy('age').Mean('score').Print;
end.
```

pandas

```
import pandas as pd
from io import StringIO

df = pd.read_csv(StringIO("""
name,age,score
Alice,20,85
Bob,22,90
Charlie,21,78
"""))

print(df[df['age'] >= 21])
print(df[['name', 'score']])
print(df['score'].mean())
print(df.groupby('age')['score'].mean())
```

Модуль DataFrameABC

- Колоночное хранение данных (column-store)
- Фиксированные типы: Int / Float / String / Bool
- Предсказуемые операции с **явными именами** (Filter, Select, GroupBy)
- Без неявных преобразований типов. **Проверка типов** на этапе компиляции

PascalABC.NET

```
uses DataFrameABC;

begin
  var df := DataFrame.FromCsvText('''
  name,age,score
  Alice,20,85
  Bob,22,90
  Charlie,21,78
  ''');

  df.Filter(row -> row.Int('age') >= 21).Print;
  df.Select(['name', 'score']).Print;
  df.Mean('score').Println;
  df.GroupBy('age').Mean('score').Print;
end.
```

```
// Цепочки запросов
df
  .Filter(r -> r.IsValid('score'))
  .SortBy('age', descending := True)
  .GroupBy('age')
  .Mean('score')
  .PrintPreview(3);
```

pandas

```
import pandas as pd
from io import StringIO

df = pd.read_csv(StringIO("""
name,age,score
Alice,20,85
Bob,22,90
Charlie,21,78
"""))

print(df[df['age'] 
print(df[['name', 'score']])
print(df['score'].mean())
print(df.groupby('age')['score'].mean())
```

```
// pandas – тоже может
result =
  (df[df['score'].notna()]
   .sort_values('age', ascending=False)
   .groupby('age')
    ['score'].mean()
   .head(3)
  )
print(result)
```

Модуль LinearAlgebraML — численное ядро ML

- Типы Vector и Matrix с перегрузкой операторов
- Базовые операции линейной алгебры (dot, norm, агрегации)
- Векторизованные функции (Exp, Ln, Sqrt, Apply)
- Численные методы для обучения моделей
  - **EigenSymmetric** — собственные значения и векторы симметричной матрицы (метод Якоби)
  - **PCA** — метод главных компонент для анализа и снижения размерности
  - **Solve** — решение систем линейных уравнений (LU-разложение)
  - **SolveSPD** — решение для симметричных положительно определённых матриц (разложение Холецкого)
  - **SolveLeastSquaresQR** — метод наименьших квадратов (QR-разложение)
  - **SolveRidge** — регрессия с L2-регуляризацией для устойчивых решений
- Проверка размерностей и корректности операций
- Оптимизация под задачи машинного обучения
- Без внешних зависимостей

Модели ML в платформе

📈 Регрессия

- **LinearRegression** — базовая линейная зависимость (без регуляризации)
- **RidgeRegression** — устойчивая регрессия (L2-регуляризация)
- **LassoRegression** — отбор признаков (L1-регуляризация)
- **ElasticNet** — отбор признаков + устойчивость (L1 + L2)

🎯 Классификация

- **LogisticRegression** — вероятностная классификация (softmax)

🌳 Деревья и ансамбли

- **DecisionTree (Regressor / Classifier)** — интерпретируемые правила решений
- **RandomForest (Regressor / Classifier)** — ансамбль деревьев (устойчивость)
- **GradientBoosting (Regressor / Classifier)** — последовательное улучшение

👥 Методы на основе экземпляров

- **kNN (Regressor / Classifier)** — прогноз по ближайшим объектам

🧠 Обучение без учителя

- **KMeans** — разбиение на кластеры
- **DBSCAN** — выделение плотных кластеров

Платформа покрывает весь базовый стек ML — от линейных моделей до ансамблей и методов без учителя

Обучение и оценка модели (PascalABC.NET)

```
uses MLABC;

begin
  var X: Matrix := [[1.0,1.0], [1.2,1.3], [1.5,1.2], [1.8,1.6], [2.0,1.9], [2.2,2.1],
                    [2.4,2.3], [2.6,2.5], [2.8,2.7], [3.0,3.0], [3.2,3.1], [3.4,3.3]];

  var y: Vector := [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1];

  var (Xtrain, Xtest, ytrain, ytest) :=
    Validation.TrainTestSplit(X, y, 0.3, seed := 4);

  var model := new LogisticRegression;
  model.Fit(Xtrain, ytrain);

  var pred := model.Predict(Xtest);

  Println('Accuracy:', Metrics.Accuracy(ytest, pred):0:3);
end.
```

Содержательных строк: **7**

```
Accuracy: 0.750 (test)
```

Обучение и оценка модели (Python)

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = [[1.0, 1.0], [1.2, 1.3], [1.5, 1.2], [1.8, 1.6],[2.0, 1.9], [2.2, 2.1],
     [2.4, 2.3], [2.6, 2.5],[2.8, 2.7], [3.0, 3.0], [3.2, 3.1], [3.4, 3.3]]

y = [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=4
)

model = LogisticRegression()
model.fit(X_train, y_train)

pred = model.predict(X_test)

print(f'Accuracy: {accuracy_score(y_test, pred):.3f}')
```

Содержательных строк: тоже **7**

sklearn уменьшает **визуальную** сложность
ML PascalABC.NET уменьшает **скрытую** сложность

```
Accuracy: 0.750 (test)
```

Предварительная подготовка данных (PascalABC.NET)

```
uses MLABC;
begin
  var df := CsvLoader.Load('towns_russia.csv', inferCategorical := true);

  df := df.Select(['city','population','lat','lon','region_name','federal_district']);

  df.Schema.Println;

  // Заполняем пропуски в числовых столбцах средним значением
  var imputer := new Imputer('population', 'lat', 'lon');
  df := imputer.FitTransform(df);

  // Кодируем категориальные признаки
  var encoder := new LabelEncoder('region_name');
  df := encoder.FitTransform(df);

  var encoder2 := new LabelEncoder('federal_district');
  df := encoder2.FitTransform(df);

  df.PrintPreview(6);
end.
```

Содержательных строк: **10**

Предварительная подготовка данных (Python)

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

df = pd.read_csv('towns_russia.csv')

df = df[['city', 'population', 'lat', 'lon', 'region_name', 'federal_district']]

print(df.dtypes)

# Заполняем пропуски в числовых столбцах средним значением
imputer = SimpleImputer(strategy='mean')
df[['population', 'lat', 'lon']] = imputer.fit_transform(df[['population', 'lat', 'lon']])

# Кодируем категориальные признаки
encoder = LabelEncoder()
df['region_name'] = encoder.fit_transform(df['region_name'])

encoder2 = LabelEncoder()
df['federal_district'] = encoder2.fit_transform(df['federal_district'])

print(df.head(6))
```

Содержательных строк: тоже **10**

В данном примере Python **короче не стал**; он просто заменил явные типы на неявные и **перенёс часть сложности** в операции над DataFrame-срезами.

Конвейер данных (PascalABC.NET)

```
uses MLABC;
begin
  var ds := Datasets.MoscowHousing;
  var df := ds.Data;

  var features := ['rooms', 'area', 'kitchen_area', 'floor', 'floors_total', 'metro_minutes', 'renovation'];
  var target := 'price';

  var (trainDf, testDf) := df.TrainTestSplit(0.2, seed := 42);

  var pipe :=
    DataPipeline.Build(      // сборка pipeline: target + features + шаги
      TaskKind.tkRegression, // тип задачи
      target,                // целевая переменная
      features,              // список признаков
      new OneHotEncoder('renovation'), // DataFrame-уровень: кодирование категории
                                       // Внутри - преобразование .ToMatrix .ToVector
      new StandardScaler,              // Matrix-уровень: масштабирование признаков
      new LinearRegression             // модель (всегда последний шаг)
    );
  pipe.Fit(trainDf); // обучение: encoding → scaling → model

  var pred := pipe.Predict(testDf); // предсказание: автоматически применяются те же преобразования
  var y := testDf.ToVector(target);
  Println('R²:', Metrics.R2(y, pred):0:3);
end.
```

Конвейер данных (Python)

```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

df = moscow_housing.copy()

features = ['rooms', 'area', 'kitchen_area', 'floor', 'floors_total', 'metro_minutes', 'renovation']
target = 'price'
X = df[features]
y = df[target]

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

preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['renovation']),
        ('num', StandardScaler(), ['rooms', 'area', 'kitchen_area', 'floor', 'floors_total', 'metro_minutes'])
    ]
)

pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('model', LinearRegression())
])

pipe.fit(X_train, y_train)

pred = pipe.predict(X_test)
print(f'R2: {r2_score(y_test, pred):.3f}')
```

- В данном примере sklearn заметно более многословный
- Сборка конвейера из компонентов
- Для StandardScaler надо повторять почти все столбцы
- Менее прозрачный код
- Строковые имена шагов — скрытая уязвимость

Встроенные датасеты PascalABC.NET

📊 Типы датасетов

- **Синтетические:**
  MakeBlobs, MakeMoons, MakeCircles, MakeSpiral, MakeRegression, MakeClassification
- **Реальные (преимущественно русские, качественные, учебные):**
  Iris, MoscowHousing, RussianCities,
  StudentExam, BankClients, TaxiTrips, OnlineShopping (в планах)

📥 Использование

```
var ds := Datasets.MoscowHousing;
var df := ds.Data;

var ds2 := Datasets.Load('file_name'); // внешний датасет (загрузка из Files\Datasets)
```

💡 Зачем

- быстрый старт без подготовки данных
- единый формат для всех задач
- удобно для обучения

Репозиторий русскоязычных датасетов (план)

💡 **Идея**

- Единая библиотека учебных датасетов на русском языке для задач классификации, регрессии и кластеризации

📚 **Содержимое**

- • реальные и синтетические датасеты
  • понятные русскоязычные признаки и категории
  • метаданные задачи: признаки, target, тип задачи, описание

🚀 **Зачем это нужно**

- • снижение языкового барьера при изучении ML
  • связь обучения с реальными данными и прикладными кейсами
  • готовая база для лекций, практик, домашних заданий и проектов

🎯 **Ключевая идея**

- **Dataset — это не просто таблица, а объект задачи машинного обучения:** данные, признаки, целевая переменная, описание и способы работы с ними

Детали рабочего процесса

- **Работа с контекстом:** когда чат переполняется (в Firefox), прошу сформировать контекст для новой беседы, переношу в новую беседу, бросаю все модули и прошу провести аудит
- **Аудит кода:** делаю ревью с помощью Anthropic Claude, после чего бросаю ChatGPT каждый пункт и обсуждаю, как исправить
- **Тестирование:** прошу сгенерировать примеры и unit-тесты, проверяю поведение, ловлю расхождения
- **Объяснение:** прошу объяснить теорию, математику, варианты
- **Поиск ошибок:** кидаю подозрительные методы, прошу предложить отладку через Print, локализую проблему, дальше итеративно чиню
- **Рефакторинг:** прошу упростить, найти неоптимальное, оптимизировать
- **Развитие:** обсуждаю, чем библиотека отличается, куда развивается, как сделать ее уникальной (например, русскоязычные датасеты)

Примеры аудита Anthropic Claude

2. `PermutationImportance` — некорректный дефолт `seed=0`

В `InspectionML.pas` (строка 48) параметр `seed` имеет дефолтное значение `0`, тогда как во всей остальной библиотеке соглашение — `seed = -1` означает «случайный». Это нарушает LSP на уровне API: пользователь, привыкший к `seed=-1`, получит воспроизводимый, но не случайный результат. Следует изменить дефолт на `-1` и в ветке проверки `seed >= 0` обрабатывать корректно.

4. `DecisionTreeClassifier` — не экспонирует критерий разбиения

Классификатор жёстко привязан к `GiniCriterion` (строка 3554), `EntropyCriterion` объявлен но недоступен через конструктор. Пользователь не может выбрать критерий, хотя класс существует. Это особенно заметно, поскольку sklearn-совместимость предполагает `criterion='entropy'`. Нужно добавить параметр `criterion: string := 'gini'` в конструктор и инстанцировать нужный критерий.

5. `Metrics.Precision/Recall/F1/Specificity` — только бинарные

В публичном API нет многоклассовых версий `Precision`, `Recall`, `F1` с усреднением (`macro`, `weighted`, `micro`). Для задач с 3+ классами пользователь либо получит некорректный результат, либо не сможет воспользоваться метриками вообще. Это очень существенный пробел для производственной библиотеки.

Общая оценка проекта (Anthropic Claude)

Это редкое сочетание целей, и оно реализовано на удивление последовательно. Разберу по каждому измерению.

Архитектурная зрелость — высокая. Интерфейсная иерархия `IModel → ISupervisedModel → IClassifier → IProbabilisticClassifier` выстроена правильно, с чётким разделением supervised/unsupervised, и не перегружена. `IPipelineStep → IMatrixStep / IDataStep` — элегантное решение двухуровневого pipeline, редкое даже в зрелых библиотеках. `IColumnBoundStep / IColumnsBoundStep` для защиты целевой переменной от случайного преобразования — деталь, о которой новички не думают, а здесь она встроена в архитектуру. Иммутабельность `DataFrameSchema` и столбцов — правильный выбор, предотвращающий целый класс ошибок. Всё это уровень осознанного API-дизайна, а не «навалили классов».

Покрытие алгоритмов — достаточное для учебного и прикладного использования. LinearRegression, Ridge, ElasticNet, LogisticRegression, DecisionTree (классификатор + регрессор), RandomForest, GradientBoosting, KNN, KMeans, DBSCAN — это полный минимум production-ML. Трансформеры (StandardScaler, MinMaxScaler, PCA, SelectKBest, Normalizer, VarianceThreshold) закрывают типичный preprocessing-pipeline. Метрики — исчерпывающие: MSE/RMSE/MAE/MAPE/MedianAE/R2, Accuracy/Precision/Recall/F1/AUC/LogLoss, Silhouette/CalinskiHarabasz/DaviesBouldin/ARI — это серьёзно.

Общая оценка проекта (DeepSeek)

7. Итоговая оценка

Библиотека находится на **высоком уровне зрелости** для проекта такого масштаба. Проделана колоссальная работа по проектированию архитектуры, реализации алгоритмов и документированию. Код демонстрирует глубокое понимание как предметной области машинного обучения, так и принципов промышленной разработки.

Уникальность и развитие и проекта (фрагмент)

Intellisense-документация с формулами и примерами прямо у курсора — sklearn этого не имеет в принципе потому что Python-экосистема смотрит документацию в браузере. Вы можете сделать это настолько хорошо что это станет маркетинговым аргументом. Каждый метод — формула, интерпретация результата, типичная ошибка, пример вызова. Это несколько недель работы но не месяцев.

Встроенные объяснения ошибок. Сейчас ваши сообщения об ошибках информативнее чем у sklearn, но можно пойти дальше. Когда `DimensionError` — не просто «размерности не совпадают», а «вы передали X с 3 признаками, но модель обучена на 5. Возможно вы забыли применить тот же StandardScaler что использовали при обучении». Это диагностическая система которой у sklearn нет. Реализуется через расширение текущей системы ошибок.

Русскоязычные датасеты и задачи. sklearn поставляется с iris, digits, boston — историческими американскими датасетами. У вас может быть коллекция реальных российских данных — зарплаты по регионам, недвижимость, демография, успеваемость. Это уникальный контент который не появится в sklearn никогда.

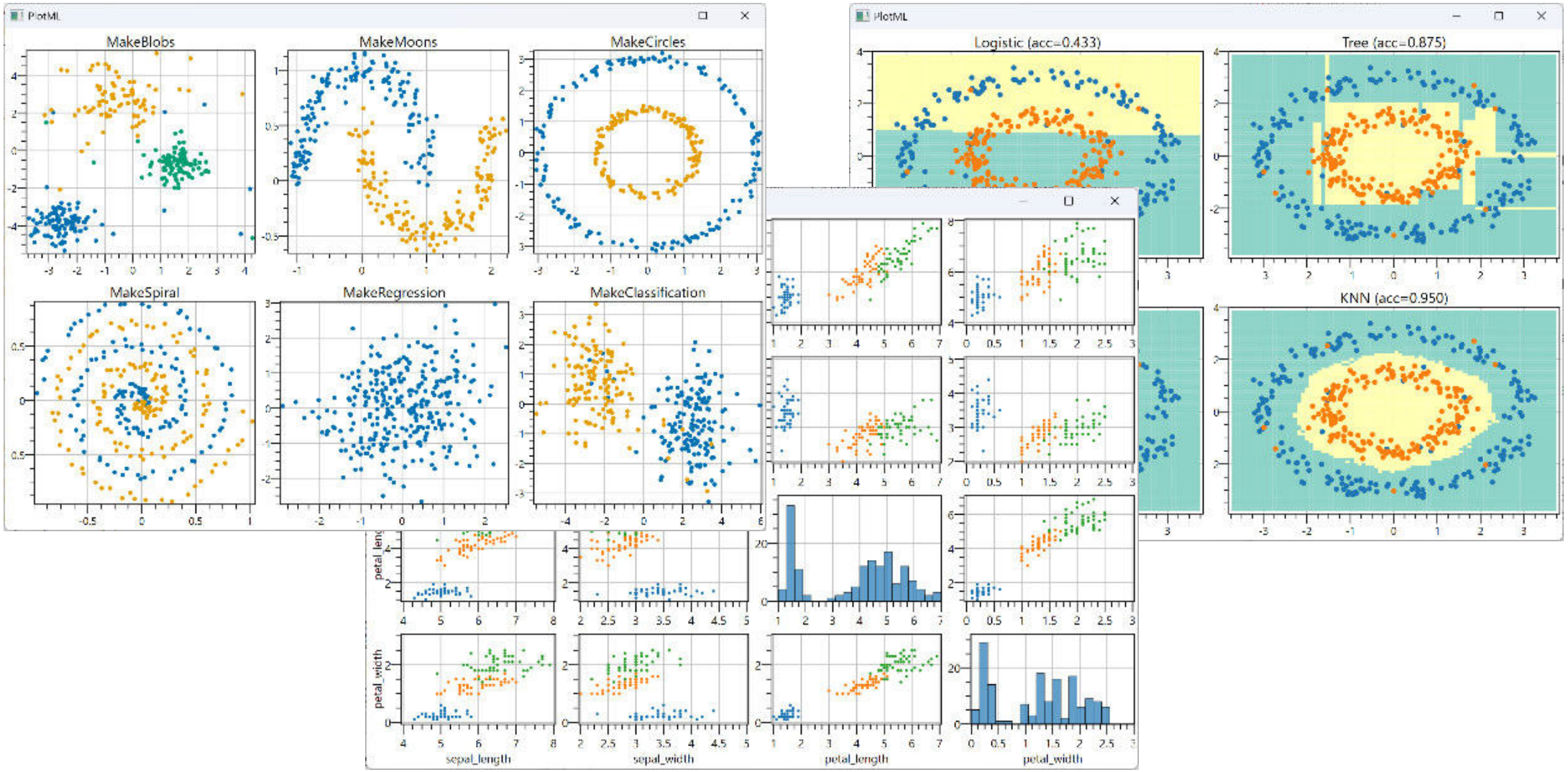
Усиливать то что уже лучше, а не догонять там где хуже. Конкретно — сделать из библиотеки лучший в мире инструмент для *изучения* ML на .NET. Это означает:

Первое — `Explain`-методы для каждой модели. Не просто `FeatureImportances` как сейчас, а текстовое объяснение что сделала модель. `LinearRegression.Explain()` выводит «признак X1 увеличивает предсказание на 2.3 единицы при росте на 1». Это уникальная вещь которой нет ни в sklearn ни в других библиотеках в таком виде.

Выводы по разработке при поддержке ИИ

- **LLM как инструмент разработки:** это не генератор кода, а полноценный напарник — архитектор, ревьюер, дебаггер в одном лице
- **Новый стиль разработки:** диалоговый, итеративный, с постоянным уточнением решений вместо «написал и забыл»
- **Скорость разработки:** выше, но только при жёсткой дисциплине
- **Контекст ограничен:** большие проекты не помещаются, приходится переносить, пересобирать и держать в голове архитектуру
- **Склонность к «галлюцинациям»:** может придумывать API, методы, детали реализации
- **Потеря связности:** без жёстких инвариантов начинает противоречить самому себе в разных частях системы. Надо напоминать эти инварианты
- **Роль разработчика** – человека – центральная!

Графическая составляющая ML библиотеки



Благодарности 🥲

- Спасибо **ChatGPT** — за код, объяснения и математические модели, лежащие в его основе
- Спасибо **ChatGPT** и **DeepSeek** — за подробные русские комментарии, которые выглядят лучше, чем мои
- Спасибо **DeepSeek** и **Claude** за генерацию диаграмм в этой презентации
- Спасибо **ChatGPT** за тестирование и устранение ошибок в коде, а также за генерацию большого количества примеров
- Спасибо **Claude** за строгие аудиты кода и формулировки в духе «в целом хорошо, но давайте перепишем всё аккуратно»
- Спасибо **GigaChat** за качественную генерацию отдельных русскоязычных формулировок
- Спасибо **ChatGPT** — за помощь в том, чтобы эта презентация выглядела лучше, чем могла бы ☺
- Спасибо **ChatGPT** — за то что хвалил когда всё получалось ☺
- Отдельное спасибо **Claude** за общую высокую оценку проекта и направления его развития

- И, конечно, **спасибо всем моделям вместе** — за самую необычную разработку в моей жизни, в которой я узнавал, что именно реализовывал, уже после того как всё заработало ☺