Модуль машинного обучения для PascalABC.NET

Михалкович С.С.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

miks@sfedu.ru

Доклад на пятой Всероссийской научно-методической конференции
«Использование системы программирования PascalABC.NET
в обучении программированию»
(Ростов-на-Дону, 28-29 марта 2025 г.)

Актуальность модуля машинного обучения

- Сегодня машинное обучение (МО) — одна из самых востребованных технологий, и знакомство с ней должно начинаться уже в школе.
- Даже на ЕГЭ стали появляться задачи машинного обучения (задача 27 – Кластеризация)
- Профессиональные инструменты (например Python с scikit-learn или TensorFlow) сложны для новичков
- PascalABC.NET всегда развивался как язык для обучения, в котором сложные концепции упрощены, поэтому реализация такого модуля актуальна
- Модуль должен реализовывать несколько распространенных моделей машинного обучения
- Вместе с модулем должны поставляться примеры, иллюстрирующие работу с известными датасетами и показывающие точность модели, сравнимую с точностью профессиональных моделей
- Скорость работы и универсальность – второстепенные факторы, главное в реализации – простой код с интуитивно понятным интерфейсом

Реализованные модели

- Было принято решение реализовать наиболее простые и популярные модели машинного обучения:
  - Линейная регрессия
  - Логистическая регрессия
  - Дерево решений
  - Метод К ближайших соседей
  - Нейронная сеть с прямым распространением
- Для помощи в быстром написании кода было решено использовать генеративный искусственный интеллект (DeepSeek, Claude, ChatGPT)
- Для реализации векторных операций активно используется LINQ и встроенная в PascalABC.NET библиотека MathNet.Numerics.dll (последняя – для матричных операций)

Основной модуль MLABC

- Основа модуля MLABC – класс **DataSet**, объект которого содержит данные задачи. Для простоты было решено все данные считать **вещественными**.
- Класс DataSet позволяет задавать данные вручную, а также считывать их из csv-файла.
- При считывании из файла учтено, что **заголовок** данных может либо присутствовать либо нет.
- Также учтено, что столбец Target, содержащий целевую переменную, **может быть не последним**.
- Второй по важности – класс **Scaler**, который позволяет масштабировать данные если они не нормализованы. Это позволяет улучшать точность в моделях, в которых плохо нормализованные данные дают низкую точность.

Класс Dataset

Данные, в объекте класса **DataSet** хранятся в виде csv-файла, а считываются в виде таблицы.

Метод TrainTestSplit позволяет разбивать данные на тренировочную и тестовую выборки.

```
type
  DataSet = class
    Features: array of array of real;
    Target: array of real;
    ColumnNames: array of string;

    constructor(filePath: string;
      hasHeaders: boolean := True;
      delimiter: char := ',';
      targetIndex: integer := -1);

    function TrainTestSplit(trainSize: real := 0.8):
      (DataSet, DataSet);
    property Shape: (integer, integer);
  end;
```

SepalLength	SepalWidth	PetalLength	PetalWidth	Target
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
5.8	2.7	4.1	1.0	1
6.0	2.2	4.0	1.0	1
6.9	3.1	5.4	2.1	2
5.5	2.3	4.0	1.3	1
6.3	2.5	4.9	1.5	2
5.0	3.6	1.4	0.2	0

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Target
5.1,3.5,1.4,0.2,0
4.9,3.0,1.4,0.2,0
5.8,2.7,4.1,1.0,1
6.0,2.2,4.0,1.0,1
6.9,3.1,5.4,2.1,2
5.5,2.3,4.0,1.3,1
6.3,2.5,4.9,1.5,2
5.0,3.6,1.4,0.2,0
```

Класс Scaler

Класс **Scaler** предназначен для масштабирования столбцов данных, сильно отличающихся друг от друга по масштабу значений.

Fit(X) — вычисляет mean (среднее) и std (стандартное отклонение) для каждого признака в данных X.

Transform(X) — применяет масштабирование к данным X на основе сохранённых mean и std.

FitTransform(X) — объединяет Fit и Transform в один шаг

```
type
  Scaler = class
    mean: array of real;
    std: array of real;
  public
    procedure Fit(X: array of array of real);
    function Transform(X: array of array of real):
      array of array of real;
    function FitTransform(X: array of array of
      real): array of array of real;
  end;
```

Ненормализованный датасет

Feature1	Feature2	Feature3	Feature4	Target
512.0	0.85	12000.0	3.5	1
150.0	0.12	3500.0	1.2	0
789.0	0.67	9800.0	2.9	1
45.0	0.05	1200.0	0.8	0
1023.0	0.92	15000.0	4.1	1

FitTransform

Нормализованный датасет

Feature1	Feature2	Feature3	Feature4	Target
0.4853	0.8947	0.7973	0.7561	1
0.1076	0.0789	0.1622	0.1219	0
0.7250	0.7105	0.6216	0.6829	1
0.0000	0.0000	0.0000	0.0000	0
1.0000	1.0000	1.0000	1.0000	1

Модель линейной регрессии

Линейная регрессия — это метод машинного обучения для поиска линейной зависимости между признаками и целевой переменной.

Для применения этого метода строится **модель**, которая представляет собой **линейное уравнение**, в котором целевая переменная выражается через сумму признаков, умноженных на **коэффициенты** (веса), плюс **смещение**. Неизвестные в этом уравнении — веса, которые метод подбирает так, чтобы предсказания модели были максимально близки к реальным данным.

Метод Fit – модель находит оптимальные веса и смещение методом наименьших квадратов

Метод Predict – модель использует найденные веса, чтобы предсказать значения для тестовой выборки.

Метод Score – модель сравнивает предсказанные данные с реальными и вычисляет, насколько хорошо она справилась, с помощью коэффициента R^2

```
type
  LinearRegression = class
  private
    weights: array of real;
    bias: real;
  public
    procedure Fit(dataset: DataSet);
    function Predict(X: array of array of real):
      array of real;
    function PredictOne(x: array of real): real;
    function Score(yTrue, yPred: array of real):
      real;
  end;
```

Датасет housing.csv

Датасет **housing.csv** – это стандартный датасет стоимости домов Boston Housing.

Этапы алгоритма:

1. Загрузка датасета
2. Разбиение на тестовую и тренировочную выборки
3. Создание и обучение модели
4. Прогнозирование на тестовых данных
5. Оценка качества модели на основе метрики R^2.

Результат: R2 Score: **0.69** – вполне приемлемый для линейной модели и согласуется с аналогичным кодом на Python

```
begin
  var dataset1 :=
    new DataSet('Datasets\housing.csv', False, ' ');

  // Разбиение на тестовую и тренировочную выборки
  var (trainSet, testSet)
    := dataset1.TrainTestSplit(0.8);

  // Создание и обучение модели
  var model := new LinearRegression;
  model.Fit(trainSet);

  // Прогнозирование на тестовых данных
  var predictions
    := model.Predict(testSet.Features);

  // Оценка качества модели
  var r2Score
    := model.Score(testSet.Target, predictions);
  Writeln('R2 Score: ', r2Score);
end.
```

```
R2 Score: 0.69
```

Модель логистической регрессии

Логистическая регрессия — это метод машинного обучения, который используется для задач классификации.

Логистическая регрессия предсказывает вероятность принадлежности объекта к определённому классу (например, 0 или 1). Это делает её особенно полезной для задач **классификации**, где нужно разделить данные на две или более категорий.

Learning Rate (скорость обучения) — **гиперпараметр**, который контролирует, насколько сильно модель изменяет свои веса на каждом шаге обучения.

Метод Fit – модель находит оптимальные веса и смещение чтобы минимизировать ошибку. Для этого используется метод градиентного спуска.

Метод Predict – модель вычисляет вероятность принадлежности объекта к классу 0 или 1.

Метод Accuracy – показывает долю правильных предсказаний.

```
type
  LogisticRegression = class
  public
    constructor(lr: real := 0.01;
      nIter: integer := 1000);
    procedure Fit(dataset: DataSet);
    function PredictOne(X: array of real): integer;
    function Predict(X: array of array of real):
      array of integer;
    function Accuracy(
      yTrue, yPred: array of integer): real;
  end;
```

Датасет diabetes.csv

Датасет **diabetes.csv** – это стандартный датасет заболеваемости диабетом.

Гиперпараметры модели:

- Скорость обучения – 0.01
- Количество итераций – 100

Поскольку данные в данном датасете имеют сильно разный масштаб, их необходимо отшкалировать с помощью класса Scaler

Этапы алгоритма:

1. Загрузка датасета
2. Разбиение на тестовую и тренировочную
3. Шкалирование
4. Создание и обучение модели
5. Прогнозирование на тестовых данных
6. Вычисление точности модели – метрика Accuracy

Результат: Accuracy: **0.81** – вполне приемлемый и согласуется с аналогичным кодом на Python

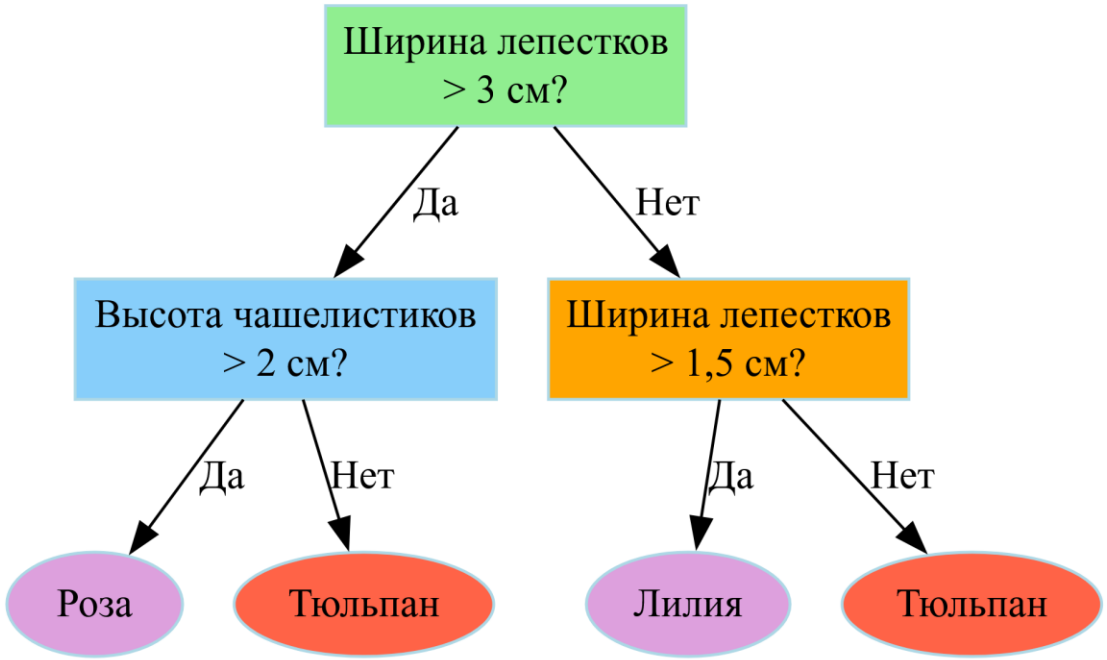
```
begin
  var dataset := new DataSet('Datasets/diabetes.csv');
  // Разбиение на тестовую и тренировочную выборки
  var (trainSet, testSet) := dataset.TrainTestSplit(0.8);

  var scaler := new Scaler();
  // Нормализация обучающей выборки
  trainSet.Features :=
    scaler.FitTransform(trainSet.Features);
  // Нормализация тестовой выборки (используем параметры,
вычисленные на обучающей выборке)
  testSet.Features := scaler.Transform(testSet.Features);
  // Создание и обучение модели
  var model := new LogisticRegression(0.1, 100);
  model.Fit(trainSet);
  // Предсказание на тестовой выборке
  var yPred := model.Predict(testSet.Features);
  // Вычисление точности
  var acc := model.Accuracy(testSet.Target.ConvertAll
    (y -> Round(y)), yPred);
  Println($'Accuracy: {acc:0.00}');
end.
```

```
Accuracy: 0.81
```

Дерево решений

Дерево решений — это один из самых популярных методов машинного обучения, используемый для задач классификации и регрессии. Суть метода заключается в разделении данных на группы с помощью последовательных простых решений, представленных в виде дерева, где каждый узел — это проверка условия, а ветви — возможные исходы.




```
type
  DecisionTree = class
    function BuildTree(X: array of array of real;
    y: array of integer; depth: integer): TreeNode;
  public
    constructor(maxDepth: integer := 5);
    procedure Fit(dataset: DataSet);
    begin
      var X := dataset.Features;
      var y := dataset.Target.ConvertAll(
        t -> Round(t));
      root := BuildTree(X, y, 0);
    end;
    function Predict(X: array of array of real):
      array of integer;
    function PredictOne(X: array of real): integer;
    function Accuracy(yTrue, yPred:
      array of integer): real;
  end;
```

Датасет breast-cancer.csv

Датасет **breast-cancer.csv** – это стандартный датасет заболеваемости.

Применим к нему модель решающего дерева, выбрав в качестве гиперпараметра максимальную глубину решающего дерева, равную 5.

Этапы алгоритма:

1. Загрузка датасета
2. Разбиение на тестовую и тренировочную
3. Создание и обучение модели
4. Прогнозирование на тестовых данных
5. Вычисление точности модели – метрика Accuracy

Результат: Accuracy: **0.95** – согласуется с аналогичным кодом на Python

```
begin
  // Первый столбец-Target: 1 - заболевание, 0 - нет
  var dataset := new DataSet
    ('Datasets/breast-cancer.csv',True,',',1);
  // Разделение на обучающую и тестовую выборки
  var (trainSet, testSet)
    := dataset.TrainTestSplit(0.8);
  // Создание и обучение модели
  var model := new DecisionTree(5);
  model.Fit(trainSet);
  // Предсказание на тестовой выборке
  var yPred := model.Predict(testSet.Features);
  // Вычисление точности
  var acc := model.Accuracy(testSet.Target.Сщ(y ->
Round(y)).ToArray(), yPred);
  // Вывод точности
  Println($'Accuracy: {acc:0.00}');
end.
```

```
Accuracy: 0.95
```

Метод k ближайших соседей

Метод k ближайших соседей (kNN) — это один из простейших алгоритмов машинного обучения, который используется для задач классификации и регрессии. В его основе лежит принцип, что объект относится к тому классу, которому принадлежит большинство из k ближайших к нему соседей.

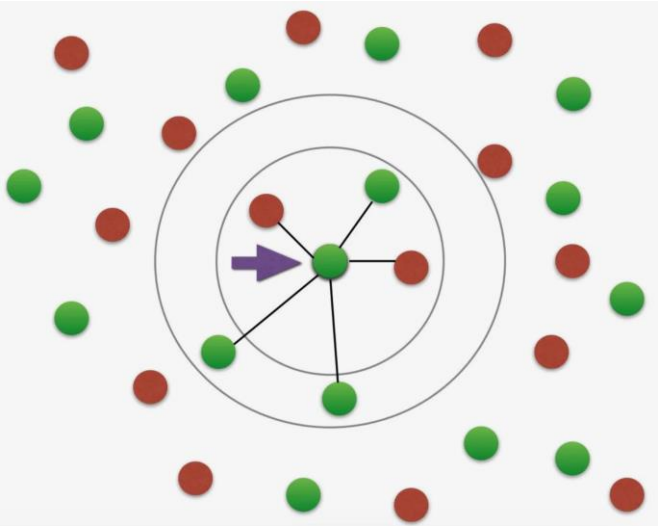
Гиперпараметр, задаваемый в конструкторе – kk – количество ближайших соседей.

Метод Fit – запоминает обучающие данные (модель не учится).

Метод Predict – возвращает предсказанные классы для всех переданных объектов.

Метод Accuracy – Сравнивает предсказания с истинными классами и возвращает точность (0–1).

```
type
  KNN = class
    constructor(kk: integer);
    procedure Fit(dataset: DataSet);
    function Predict(X_test: array of array
      of real): array of integer;
    function PredictOne(X_test: array of real)
      : integer;
    function Accuracy(yTrue, yPred: array of
      integer): real;
  end;
```



Датасет iris.csv

Датасет **iris.csv** – это стандартный датасет цветков Ириса.

Применим к нему метод k ближайших соседей с параметром k = 3.

Этапы алгоритма:

1. Загрузка датасета
2. Разбиение на тестовую и тренировочную выборки
3. Создание и обучение модели
4. Прогнозирование на тестовых данных
5. Прогнозирование класса одного объекта (класс 0 означает первый вид цветка Ириса)
6. Вычисление точности модели – метрика Accuracy

Результат: Accuracy: **0.967** – согласуется с аналогичным кодом на Python

```
begin
  var dataset := new DataSet('Datasets\iris.csv',
False, ',');
  // Разделение на обучающую и тестовую выборки
  var (trainSet, testSet) := dataset.TrainTestSplit;
  // Создание и обучение модели kNN
  var knn1 := new KNN(3);
  knn1.Fit(trainSet);
  // Прогнозирование на тестовых данных
  var predictions := knn1.Predict(testSet.Features);
  // Прогнозирование для одного объекта
  var newSample := [5.1, 3.5, 1.4, 0.2];
  var pred := knn1.PredictOne(newSample);
  Println($'Prediction for new sample: {pred}');
  // Оценка точности
  var accuracy := knn1.Accuracy(testSet.Target.
    ConvertAll(t -> Round(t)), predictions);
  Println($'Accuracy: {accuracy}');
end.
```

```
Prediction for new sample: 0
Accuracy: 0.966666666666667
```

Нейронная сеть прямого распространения

Нейронная сеть прямого распространения — это один из базовых типов искусственных нейронных сетей, где информация движется только в одном направлении: от входного слоя через скрытые слои к выходному.

Гиперпараметры, задаваемые в конструкторе:

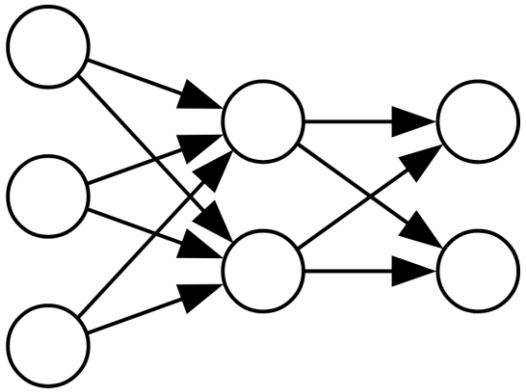
- inputSize – количество входных слоев.
- hiddenSize – количество скрытых слоев.
- learningRate – насколько сильно корректируются веса модели каждую итерацию

Метод Fit – обучает модель на данных, подбирая веса для минимизации ошибки. При обучении указывается количесово эпох (больше – точнее до некоторого момента, но дольше)

Метод Predict – возвращает предсказанные классы для всех переданных объектов.

Метод Accuracy – Сравнивает предсказания с истинными классами и возвращает точность (0–1).

```
type
  NeuralNetwork = class
    constructor(inputSize, hiddenSize: integer;
      learningRate: real := 0.01);
    procedure Fit(dataset: DataSet;
      epochs: integer := 1000);
    function Predict(X: array of array of real)
      : array of integer;
    function PredictOne(X: array of real): integer;
    function Accuracy(yTrue, yPred: array of
      integer): real;
  end;
```



Применение нейросети на датасете breast-cancer

Применим нейросеть прямого распространения на уже известном датасете **breast-cancer.csv**.

Количество нейронов на скрытом слое выберем равным 10.

Этапы алгоритма – те же самые, что и в предыдущих моделях – не будем это снова повторять.

Заметим, что нейросеть чувствительна к ненормализованным данным, поэтому перед обучением модели нормализуем данные с помощью класса Scaler.

Результат: Accuracy: **0.98** – отличный и согласуется с аналогичным кодом на Python

```
begin
  var dataset := new DataSet('Datasets/breast-
cancer.csv',True,',',1);

  var (trainSet, testSet) := dataset.TrainTestSplit();

  var scaler := new Scaler();
  trainSet.Features := scaler.FitTransform(trainSet.Features);
  testSet.Features := scaler.Transform(testSet.Features);

  var model := new NeuralNetwork(trainSet.Features[0].Length,
     10, 0.01);
  model.Fit(trainSet, epochs := 1000);

  var yPred := model.Predict(testSet.Features);

  var acc := model.Accuracy(testSet.Target.ConvertAll
    (y -> Round(y)), yPred);
  Println($'Accuracy: {acc:0.00}');
end.
```

```
Accuracy: 0.98
```

Выводы

Выводы

PascalABC.NET создавался как современный язык для образовательных целей. Он сочетает богатый набор синтаксических конструкций с мощными библиотеками, позволяя легко реализовывать даже сложные методы машинного обучения.

При этом синтаксис PascalABC.NET практически не отличается от синтаксиса библиотеки scikit-learn в Python, а точность методов остаётся на том же уровне. Ключевое преимущество – благодаря компиляции ошибки обнаруживаются на раннем этапе, что особенно важно в обучении.

- Примечательно, что все приведённые коды были реализованы с помощью **DeepSeek** и **Claude** менее чем за **14 часов**, что демонстрирует эффективность разработки на PascalABC.NET и возможность ИИ моделей генерировать современнный код на этом языке.

Текст на данном слайде тоже был сгенерирован Deepseek ☺
Правда, под управлением человека ☺

Михалкович С.С.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

miks@sfedu.ru