Архитектура приложения

1.1 Общая архитектура приложения

Для реализации требований, предъявляемых к дипломной работе, было решено создать desktop-приложение с использованием стека технологий .Net Framework. Реализованное приложение имеет строгую, но гибкую архитектуру, которая позволяет с легкостью расширять и изменять приложение и его части в дальнейшем.

Приложение разбито на несколько проектов, каждый из которых реализует определенную логику в приложении. Архитектура реализованного приложения подразумевает слабое связывание компонентов и, в частности, проектов, что позволяет без потерь и доработок заменить какой-либо проект аналогичным. Например, логика сохранения структуры нейронных сетей и кластеров реализована с использованием базы данных PostgreSQL; если в дальнейшем появится необходимость заменить использование БД на какое-либо no-sql хранилище (или любое другое), потребуется всего лишь изменить один проект приложения, и эти изменения никак не скажутся на остальной логике программы.

Все проекты приложения отображены на Рис. 1.

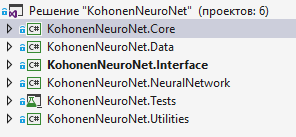


Рис 1. Список проектов приложения.

Рассмотрим структуру приложения более подробно.

* 1. Проект KohonenNeuroNet.Core

Проект KohonenNeuroNet.Core реализует CRUD-ядро (от англ. Create Read Update Delete – Создание Чтение Изменение Удаление) системы. В нем определены интерфейсы работы с данными, декларативно описывается модель взаимодействия основного приложения и хранилища данных для него. В этом проекте не реализован сам способ хранения данных – только интерфейсы работы с ними. Соответственно, этот проект никак не привязан к какому-либо определенному хранилищу данных. Разработчик может создать несколько способов хранения данных, но они должны реализовать собранные в данном проекте интерфейсы.

Реализован этот проект как библиотека классов. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework.

Проект состоит из 3 основных частей. Первая часть – интерфейсы для доступа к данным. В качестве методики доступа к данным были выбраны паттерны Repository и Unit of work – интерфейсы для данных паттернов и описаны в этой части проекта. Интерфейсы репозиториев довольно схожи, реализуют однотипный функционал для различных сущностей. Как пример, разберем интерфейс репозитория нейронной сети - INetworkService. Он содержит следующие методы:

NetworkBase GetByID(int id); - Получить нейронную сеть по идентификатору

IEnumerable<NetworkBase> GetAll(); - Получить список всех нейронных сетей

int Insert(NetworkBase entity); - Вставить новую нейронную сеть в хранилище.

void Update(NetworkBase entity); - Обновить нейронную сеть.

void Delete(NetworkBase entity); - Удалить нейронную сеть.

Остальные репозитории имеют идентичную структуру с точностью до сущности, с которой они работают.

Интерфейс для реализации паттерна Unit of work предоставляет описание сервиса, управляющего непосредственно работой с базой. В нем определены методы для работы с транзакциями, а также список репозиториев, с которыми можно работать:

void BeginTransaction(); - Открыть транзакцию

void CommitTransaction(); - Подтвердить транзакцию, если она открыта

void RollbackTransaction(); - Откатить транзакцию, если она открыта

Также в этой части реализован интерфейс сервиса для обработки бизнес-логики работы с нейросетью - INetworkService.

Вторая часть проекта содержит модели данных приложения. В ней описаны как доменные модели (модели-слепки базы данных), так и бизнес-модели, которые по сути являются аггрегациями доменных моделей. Модели содержат только открытые C# свойства и никакой логики больше. Пример модели – модель нейрона - класс NeuronBase, содержит следующие свойства:

public int NeuronId { get; set; } - Идентификатор нейрона

public int NeuronNumber { get; set; } - Порядковый номер нейрона

public int NetworkId { get; set; } - Ссылка на нейронную сеть

Последняя часть проекта содержит реализацию сервисов бизнес-логики приложения. Эти сервисы реализуют интерфейсы, описанные в 1 части текущего проекта. Там пока реализован лишь 1 сервис – бизнес-логика нейронной сети – класс NetworkService. Он реализует интерфейс INetworkService и содержит следующие методы:

public List<NetworkBase> LoadAllNetworks() - Загрузить список нейронных сетей

public NeuralNetworkData GetNetworkData(int networkId) - Получить данные о нейронной сети

public void SaveNetworkData(NeuralNetworkData networkData) - Сохранить данные о сети

public void DeleteNetwork(int networkId) - Удалить сеть

На Рис 2. Представлена диаграмма классов проекта.

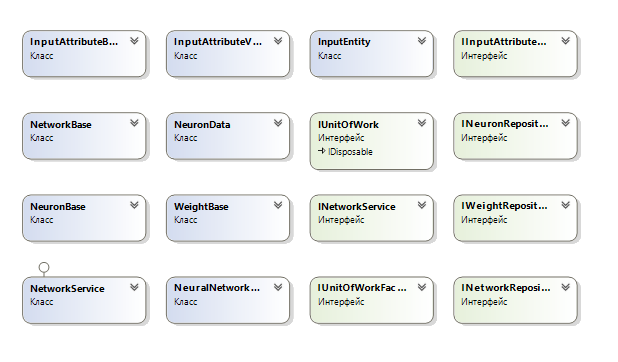


Рис 2. Диаграмма классов проекта KohonenNeuroNet.Core

* 1. Проект KohonenNeuroNet.Data

Проект KohonenNeuroNet.Data также реализует CRUD-ядро системы, однако он уже нацелен на работу с конкретным хранилищем данных. В качестве хранилища была выбрана СУБД PostgreSQL. На самом деле, на месте PostgreSQL могла оказаться любая другая. Реляционная база данных как способ хранения данных была выбрана благодаря удобству написания запросов к данным и надежности их хранения. Что же касается PostgreSQL – она бесплатна и реализует весь необходимый функционал для хранения данных приложения.

Реализован этот проект как библиотека классов. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework, Dapper в качестве ORM (Object Relation Mapping). Dapper позволяет избавиться от написания SQL запросов напрямую к базе вручную, он берет эту ношу на себя, предоставляя возможность оперировать классами и их свойствами.

Проект также содержит 3 основные части. Первая часть реализует маппинг доменных моделей из проекта KohonenNeuroNet.Core на сущности БД. Для каждой доменной сущности прописываются мапперы, которые сопоставляют свойства моделей полям в БД. Мапперы считываются при старте приложения и инициализируют контекст работы с базой. Все мапперы наследуются от класса DommelEntityMap. Как пример маппера можно рассмотреть маппер для сущности нейронной сети - NetworkMap. Он сопоставляет поля следующим образом:

public NetworkMap()

{

ToTable("network");

Map(e => e.NetworkId)

.IsKey()

.IsIdentity()

.ToColumn("network\_id");

Map(e => e.Name)

.ToColumn("network\_name");

Map(e => e.CreatedOn)

.ToColumn("created\_on");

Map(e => e.ParentNeuronId)

.ToColumn("parent\_neuron\_id");

}

Вторая часть проекта представляет репозитории для доступа к данным. Для каждой доменной модели реализован свой репозиторий, который имплементит соответствующий интерфейс проекта KohonenNeuroNet.Core. Доступ к данным производится с помощью описанной выше ORM. Пример репозитория – репозиторий нейронов – NeuronRepository, реализует интерфейс INeuronRepository и, с очевидностью, реализует следующие методы:

public int Insert(NeuronBase item) - Вставить новый экземпляр сущности

public void Update(NeuronBase item) - Обновить существующий экземпляр сущности

public void Delete(NeuronBase item) - Удалить существующий экземпляр сущности

public IEnumerable<NeuronBase> GetAll() - Получить список всех экземпляров сущности

public NeuronBase GetByID(int id) - Получить экземпляр сущности по его идентификатору

Конструктор репозиториев принимает как параметр контекст данных, с помощью которого осуществляется доступ к БД.

Здесь же описана реализация паттерна Unit of work – класс UnitOfWork. Помимо интерфейсных методов он имеет приватный метод создания соединения с БД:

private DataContext \_CreateDataContext(IConfiguration configuration)

Данные считываются из объекта конфигурации и создается коннект к базе. Соответственно, сервисы бизнес-логики напрямую (вернее, через интерфейс) используют этот класс для доступа к данным.

На Рис 3. Представлена диаграмма классов проекта.

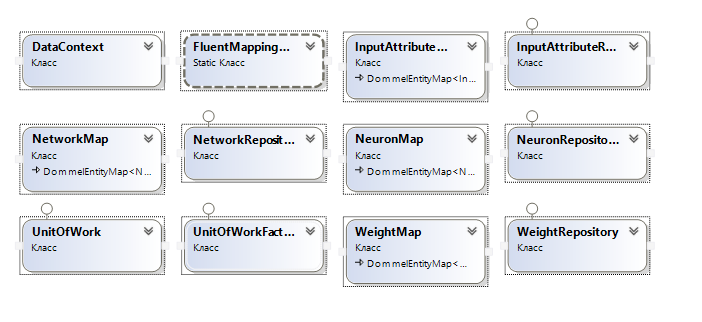


Рис 3. Диаграмма классов проекта KohonenNeuroNet.Data

* 1. Проект KohonenNeuroNet.Utilities

Пожалуй, данный проект является самым простым проектом в приложении. Он предоставляет утилитарные методы типа методов расширения различных классов и т.п. Все проекты решения имеют ссылку на этот проект и могут пользоваться описанными утилитами.

Реализован этот проект как библиотека классов. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework.

Проект содержит методы расширения класса DataTable, описанные в модуле DataTableExtension:

public static T Max<T>(this DataTable table, DataColumn column) where T : struct - Получить максимальное значение колонки таблицы

public static T Min<T>(this DataTable table, DataColumn column) where T : struct - Получить минимальное значение колонки таблицы

Эти методы используются для нормализации значений атрибутов сущностей.

Также проект содержит класс, считывающий данные из Excel таблицы в объект DataTable - ExcelReader. Данные считываются с помощью библиотеки Microsoft.Office.Interop.Excel. Тут же описан интерфейс для сервиса считывания данных, который реализует ExcelReader – Ireader. В случае изменения формата исходных данных с Excel таблицы на что-то другое, может быть реализован данный интерфейс.

На Рис 4. Представлена диаграмма классов проекта.

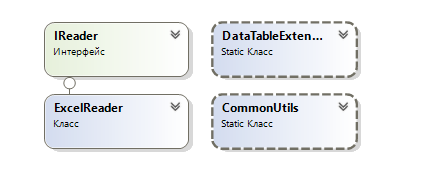


Рис 4. Диаграмма классов проекта KohonenNeuroNet.Utilities

* 1. Проект KohonenNeuroNet.Tests

Проект KohonenNeuroNet.Tests также довольно прост – он содержит unit-тесты для всей логике приложения. При разработке проекта, кодировании, я старался придерживаться концепции TDD (Test Driven Development – Разработка через тестирование). Ее суть заключается в том, что перед написанием очередного блока кода пишется unit тест, который, очевидно, фейлится. Соответственно, программисту остается написать модуль таким образом, чтобы unit-тест завершился успешно. Это позволяет в разы улучшить качество кода. Во-первых, программист на момент написания кода знакомится с поставленной задачей и уже обдумывает способы ее решения. А во-вторых, автоматические тесты нельзя обмануть, правильно написанный тест гарантирует корректную работу тестируемого модуля, в то время, как человек может не заметить ошибку при проведении codereview.

Реализован этот проект как проект Unit тестов от MS Unit. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework, библиотека Microsoft.VisualStudio.TestTools.UnitTesting.

Unit тесты покрывают значительную часть проекта. Они устроены практически одинаково – тестовый класс создается для тестирования каждого модуля. Базовый класс для всех тестов – BaseTest – содержит информацию о подключении к БД для интеграционных тестов. Рассмотрим класс тестов для репозитория нейронов – NeuronRepositoryTest. Он содержит следующие методы:

[TestMethod]

public void ShouldGetNeuronById() - Тест выборки по идентификатору

[TestMethod]

public void ShouldGetAllNeurons() - Тест выборки всех нейронов

[TestMethod]

public void ShouldCrudNeuron() - Тест вставки, изменения, удаления нейронов

На Рис 5. Представлена диаграмма классов проекта.

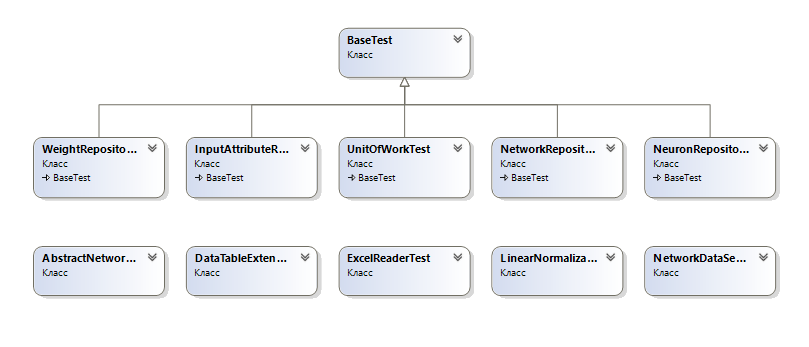


Рис 5. Диаграмма классов проекта KohonenNeuroNet.Tests

* 1. Проект KohonenNeuroNet.NeuralNetwork

Данный проект, что очевидно по названию, реализует саму логику нейронной сети Кохонена. Архитектура проекта закладывает независимость логики нейронной сети от хранения данных, отображения сети пользователю, что достаточно удобно в плане расширения и изменения проекта. То есть проект KohonenNeuroNet.NeuralNetwork не зависит ни от проекта KohonenNeuroNet.Data, ни от проекта KohonenNeuroNet.Interface. Таким образом, если в дальнейшем будет принято решение реализовать веб интерфейс нейронной сети или хранить данные в другом хранилище – доработки никак не скажутся на проекте KohonenNeuroNet.NeuralNetwork. Говоря терминами архитектуры ПО, реализовано слабое связывание проектов (Loose coupling) и модулей, что позволяет улучшить качество, расширяемость и сопровождаемость разрабатываемого решения.

Реализован этот проект как библиотека классов. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework.

Проект содержит 3 основные части. Первая часть реализует логику нормализации данных нейронной сети. Для удобства был создан интерфейс INormalizatiionType, описывающий основные методы нормализации данных:

double GetAttributeValue(NetworkEntityAttributeValue attribute); - Получить нормализованное значение атрибута

double GetNeuronWeight(int inputsCount); - Получить нормализованное значение веса нейрона.

Соответственно, чтобы создать свое правило нормализации данных, необходимо реализовать этот интерфейс. Я создал два основных правила нормализации, которые обычно используются в нейронных сетях – линейная нормализация от 0 до 1 (LinearNormalizationType\_0\_1) и линейная нормализация от -1 до 1 (LinearNormalizationType\_\_1\_1). Реализация этих классов достаточно проста.

Вторая часть проекта – модели, с которыми работает нейронная сеть. Здесь описаны классы, содержащие в основном только открытые свойства. Среди основных значимых моделей реализованы следующие:

NetworkAttribute – модель атрибута сущности (колонка таблицы). У нее есть свойства:

* public double Max { get; set; } - Максимальное значение атрибута
* public double Min { get; set; } - Минимальное значение атрибута
* public string Name { get; set; } - Название атрибута
* public int OrderNumber { get; set; } - Порядковый номер атрибута

NetworkCluster – модель кластера нейронной сети. У нее есть свойства:

* public int Number { get; set; } - Номер кластера
* public List<NetworkDataEntity> Entities { get; set; } - Сущности этого кластера
* public List<NetworkCluster> Clusters { get; set; } - Дочерние кластеры
* public int? NetworkId { get; set; } - Нейронная сеть для дочерних кластеров

NetworkEntityAttributeValue – модель значения атрибута элемента данных (ячейка таблицы). У нее есть свойства:

* public NetworkAttribute Attribute { get; set; } - Атрибут сущности
* public double Value { get; set; } - Значение атрибута

NetworkDataEntity – модель элемента данных (строка таблицы). У нее есть свойства:

* public List<NetworkEntityAttributeValue> AttributeValues { get; set; } - Значения атрибутов
* public string Name { get; set; } - Название элемента данных
* public int OrderNumber { get; set; } - Порядковый номер элемента данных.

Здесь же описан конвертер для маппинга объекта DataSet на модели для нейронной сети - NetworkDataSetConverter.

Третья часть – описание логики самой нейронной сети. Был выделен базовый абстрактный класс нейронной сети – AbstractNetwork. Он является каркасом нейронной сети, содержит как абстрактные, так и виртуальные элементы класса. Рассмотрим основные его составляющие более подробно:

public abstract INormalizatiionType NormalizationType { get; } - Тип нормализации, должен быть задан в реализующем его классе

public List<NeuronBase> Neurons { get; set; } - Список нейронов сети

public List<InputAttributeBase> InputAttributes { get; set; } – Список элементов входного слоя сети

public List<WeightBase> Weights { get; set; } - Веса нейрона - синапсы, связывающие нейрон с входным слоем.

public const double LEARNING\_ERROR = 0.01; - Минимальная ошибка обучения

public virtual void Study(NetworkDataSet inputDataSet, int neuronsCount, int iterationsCount) - Обучить входной вектор (провести итерацию обучения). Виртуальный метод, реализованный в абстрактном классе, но допускающий переопределение в дочерних классах. По сути создает нейронную сеть и обучает ее, вызывая в цикле метод StudyInputEntity.

public abstract double StudyInputEntity(List<InputAttributeValue> attributes, int currentIteration, int iterationsCount); - Обучить входной вектор (провести итерацию обучения). Абстрактный метод, необходимо реализовать его в дочерних классах.

public abstract NeuronBase GetNeuronWinner(IEnumerable<InputAttributeValue> attributeValues); - Получить нейрон-победитель. Абстрактный метод, необходимо реализовать его в дочерних классах.

public abstract double GetLearningRate(int currentIteration, int iterationsCount); - Получить скорость обучения на текущем цикле обучения. Абстрактный метод, необходимо реализовать его в дочерних классах.

Описанный здесь же класс SelfOrganizingMap наследуется от AbstractNetwork, реализуя все его абстрактные части. В качестве типа нормализации была выбрана линейная нормализация от -1 до 1. Остальные методы реализованы в соответствии с теорией нейронной сети Кохонена.

На Рис 6. Представлена диаграмма классов проета.

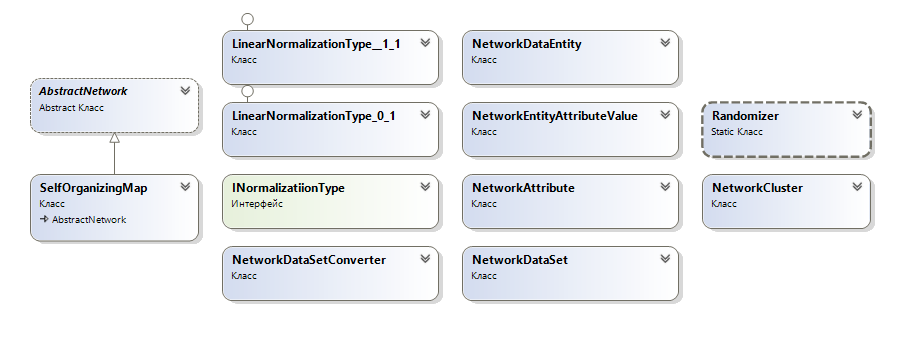


Рис 6. Диаграмма классов проекта KohonenNeuroNet.NeuralNetwork

* 1. Проект KohonenNeuroNet.Interface

Проект KohonenNeuroNet.Interface создан для визуализации работы нейронной сети. Интерфейс был реализован с помощью winforms, поскольку это самый простой способ отобразить данные в .Net приложении. При необходимости, можно поменять способ визуализации сети, например, на веб интерфейс. При этом не нужно будет менять ни один другой проект – это возможно благодаря слабому связыванию проектов.

Реализован этот проект как библиотека классов. Стек технологий, используемых для разработки модулей проекта: язык программирования C#, .Net Framework, winforms, IoC (Ninject).

Также проект реализует паттерн Внедрение зависимостей (Dependency Injecttion). Этот паттерн позволяет получать реализацию сервисов по их интерфейсам, благодаря чему код становится чище и понятней, а изменения – проще и быстрее. В качестве контейнера зависимостей был выбран Ninject, его функциональности достаточно для нужд проета. Сама реализация контейнера выделена в 2 класса: IoC, представляющий сам контейнер, и ServiceConfigurator, регистрирующий зависимости. IoC реализует паттерн Одиночка (Singleton), хранит в себе ядро Ninject-контейнера и использует ServiceConfigurator для установки зависимостей. Контейнер зависимостей инициализируется 1 раз при старте проекта и используется в дальнейшем во всех частях проекта.

В проекте созданы 2 формы. Первая форма – список нейронных сетей. При запуске проекта она загружает список сетей и отображает их пользователю. Также она предоставляет позможность добавлять, редактировать и удалять нейронные сети. Внешний вид формы представлен на Рис. 7.

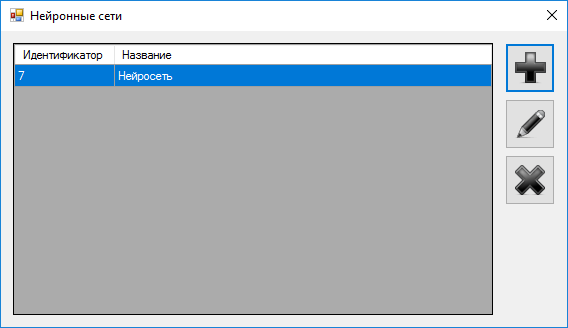


Рис 7. Форма списка нейросетей

Вторая форма – форма редактирования нейросети. Она открывается при создании и редактировании сети из формы списка нейросетей. Сама форма представлена на Рис 8.

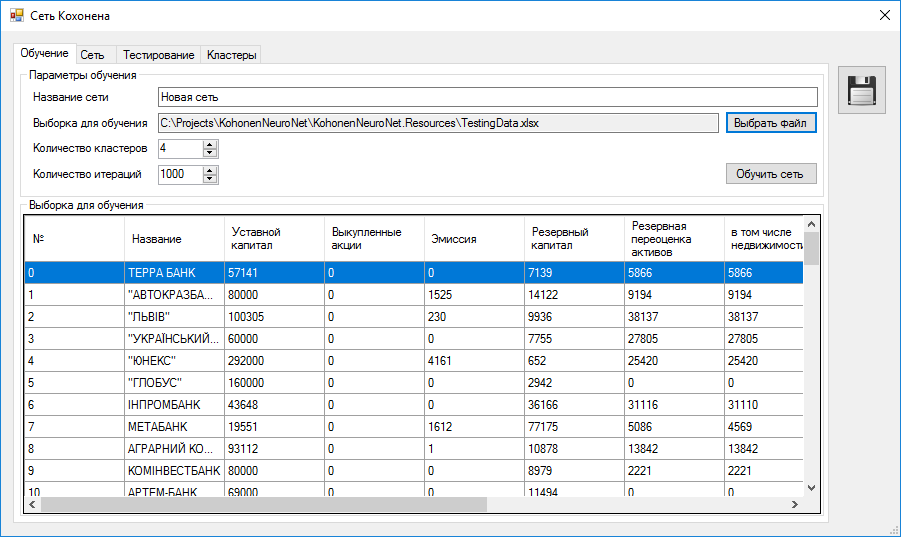


Рис 8. Форма редактирования нейросети

Форма содержит 4 вкладки:

* Обучение – настройки для обучения, данные для обучения из Excel
* Сеть – таблица весов сети
* Тестирование – настройки и данные для тестирования сети
* Кластеры – дерево кластеров с динамическим списком элементов кластера

Форма позволяет сохранять нейронные сети, производить повторную кластеризацию данных и другие функции. Количество кода при разработке этой формы оказалось достаточно большим, поэтому было принято решение выделить в отдельный модуль - InterfaceHelpers - вспомогательные функции для отображения данных в таблицах, деревьях и т.п.

Диаграмма классов проекта представлена на Рис. 9.

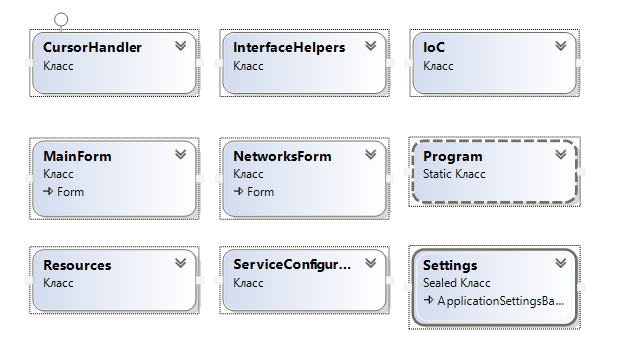


Рис 9. Диаграмма классов проекта KohonenNeuroNet.Interface

* 1. Хранилище данных

Задание дипломной работы подразумевает хранение созданной структуры кластеров в каком-либо хранилище с возможностью достать информацию о кластерах и кластеризовать новые данные без обучения сети. Для этих целей была использована база данных. В качестве СУБД был выбран PostgreSQL, т.к. это полностью бесплатная СУБД, функционала которой вполне достаточно для выполнения требования дипломной работы.

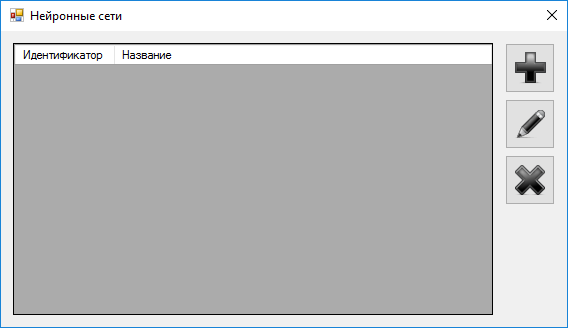
Для хранения информации о сети была создана база neural\_network с 4 таблицами. Рассмотрим их подробнее.

1. Таблица входных атрибутов нейронной сети – представлена на Рис 10. Она является хранилищем для входных атрибутов (колонок таблиц excel). Количество входных атрибутов сети совпадает с количеством значимых колонок excel. У нее есть следующие колонки:
   1. Input\_attribute\_id – идентификатор сущности, первичный ключ таблицы
   2. Input\_attribute\_name – название атрибута (колонки)
   3. network\_id – ссылка на нейронную сеть, внешний ключ
   4. Input\_attribute\_number – порядковый номер атрибута
2. Таблица нейронной сети (network). В ней хранится базовая информация о сети. Содержит следующие колонки:
   1. network\_id – идентификатор сущности, первичный ключ таблицы
   2. network\_name – название сети
   3. created\_on – дата создания сети
   4. parent\_neuron\_id – родительский нейрон сети. Эта колонка «реализует» возможность каскада кластеров.
3. Таблица нейронов сети (neuron). Хранит список нейронов для сети. Включает следующие колонки:
   1. neuron\_id - идентификатор сущности, первичный ключ таблицы
   2. network\_id – ссылка на нейронную сеть, внешний ключ
   3. neuron\_number - порядковый номер нейрона, также является номером кластера
4. Таблица весов сети (weight). Хранит список весов для нейронов сети. Содержит колонки:
   1. weight\_id - идентификатор сущности, первичный ключ таблицы
   2. Input\_attribute\_id – ссылка на входной атрибут, внешний ключ
   3. neuron\_id - ссылка на нейрон, внешний ключ
   4. value – значение веса
   5. neuron\_number - порядковый номер нейрона, также является номером кластера
   6. Input\_attribute\_number – порядковый номер входного атрибута

Соответственно, каскад кластеров реализуется через поле parent\_neuron\_id у таблицы нейронной сети. Т.к. сеть Кохонена предполагает, что нейроны сети являются кластерами, дочерние кластеры (т.е. сеть, представляющая эти кластеры) связана с конкретным нейроном (кластером).

1. Мануалка (неофициальная ☺)

2.1. Форма списка сетей



Все очевидно, список, 3 кнопки, обновляется после сохранения/удаления сетей.

* 1. Форма редактирования сети

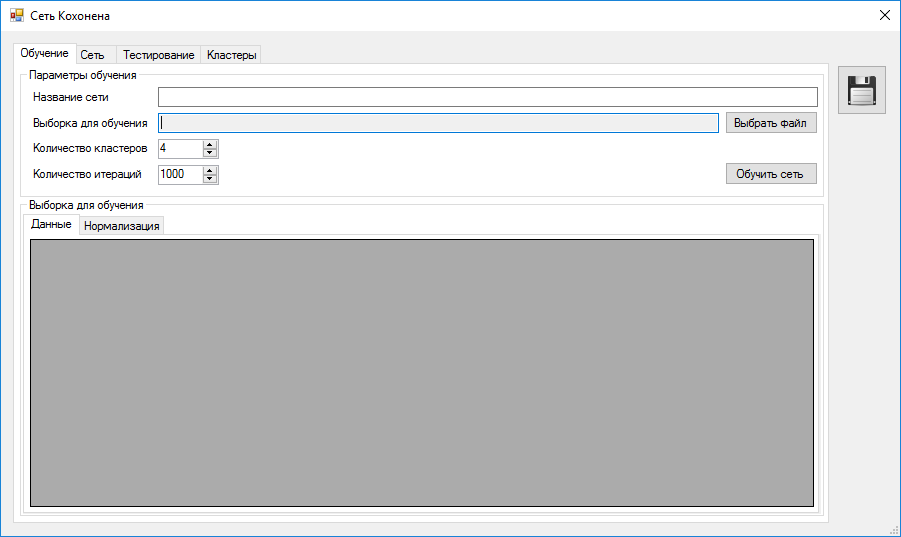


Рис 2.2.1. Форма редактирования сети при добавлении новой сети.

При добавлении новой сети открывается форма, в ней 4 вкладки. На вкладке «Обучение» заполняем название сети, выбираем файл для обучения, параметры обучения и нажимаем на «Обучить сеть». После выбора файла для обучения загрузятся данные в таблицы на подвкладках «Данные» и «Нормализация» - соответственно исходные и нормализованные данные. После обучения программа перекинет пользователя на вкладку «Тестирование».

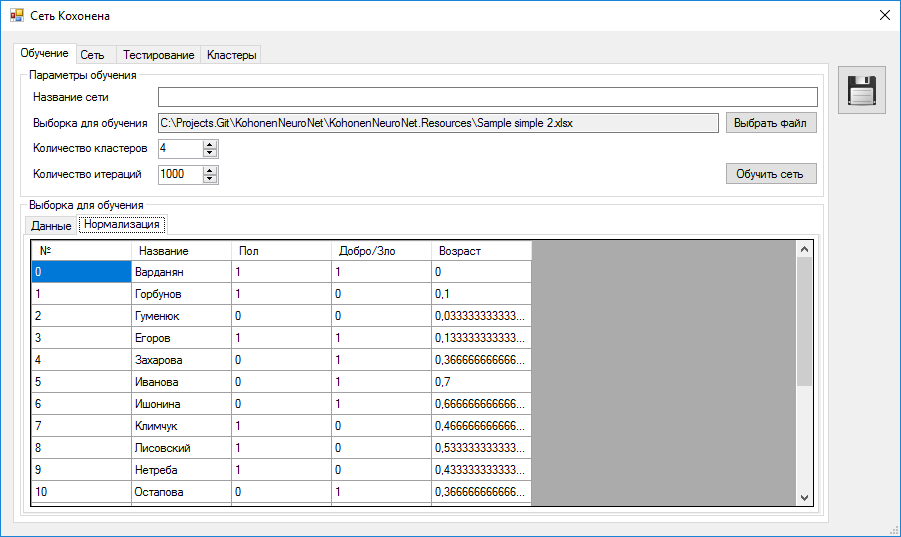


Рис 2.2.2. Форма редактирования сети – загрузили данные, они автоматически нормализовались

На вкладке «Сеть» по сути «бесполезная» инфа – таблица весов сети. Вывел ее просто для отладки, чтобы проще было смотреть, как обучается сеть.

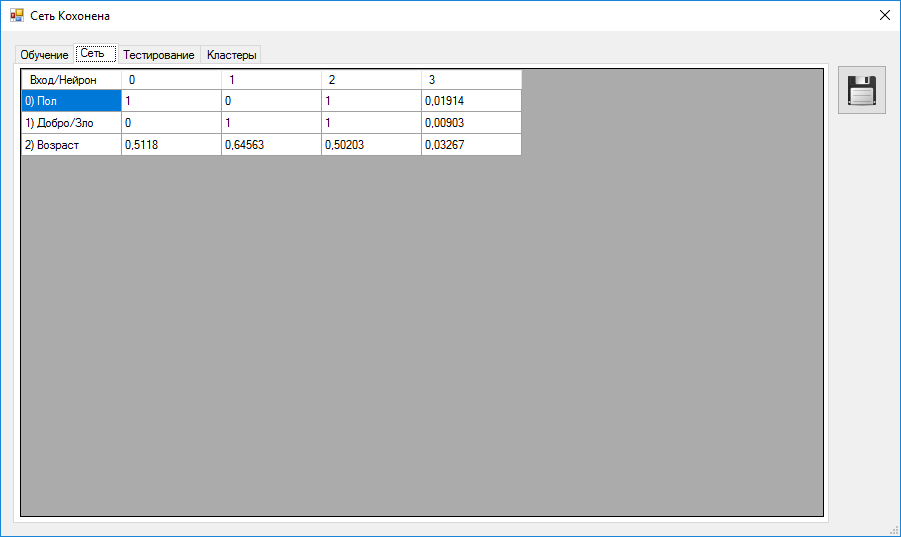


Рис 2.2.3. Обучили сеть – появились веса сети

После обучения (до обучения нельзя, важно!) сети ее можно сохранить, т.к. веса сети заполнились. На вкладке «Тестирование» выбираем файл для тестирования и нажимаем «Кластеризовать». Пользака перебрасывает на вкладку «Кластеры».

На вкладке «Кластеры» отобразится дерево кластеров сети. 1 уровень дерева – «Нейронная сеть». 2 уровень дерева – кластеры текущей (важно!) нейронной сети. 3 и больше уровни – кластеры дочерних сетей – появятся после повторной кластеризации какого-то кластера. Элементы дерева кликабельны, при выборе кластеров 1 и более уровня в таблице «Элементы кластера» отображается список элементов.

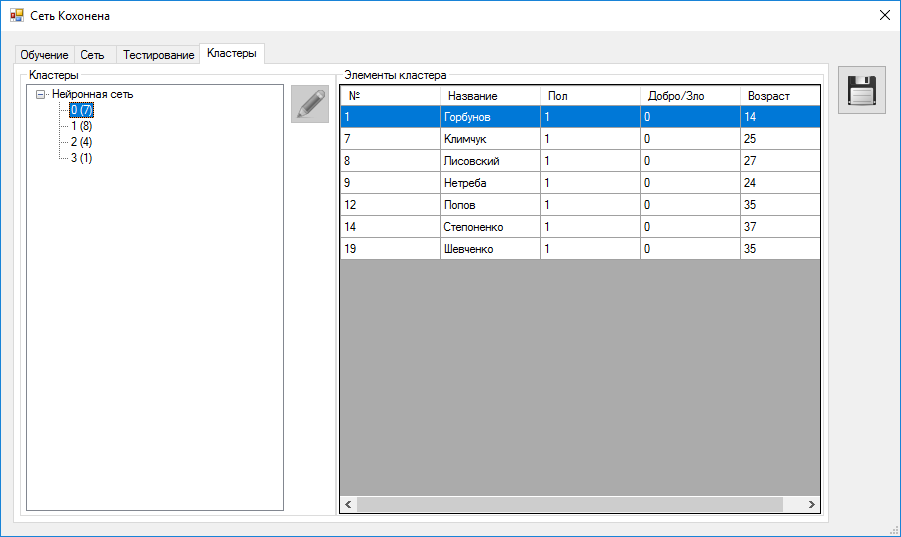


Рис 2.2.4. Кластеризовали тестовый набор данных – заполнилось дерево кластеров, при выборе кластера – отображается список элементов.

* 1. Повторная кластеризация

Повторная кластеризация реализуется на вкладке «Кластеры» с помощью кнопки «Редактировать».

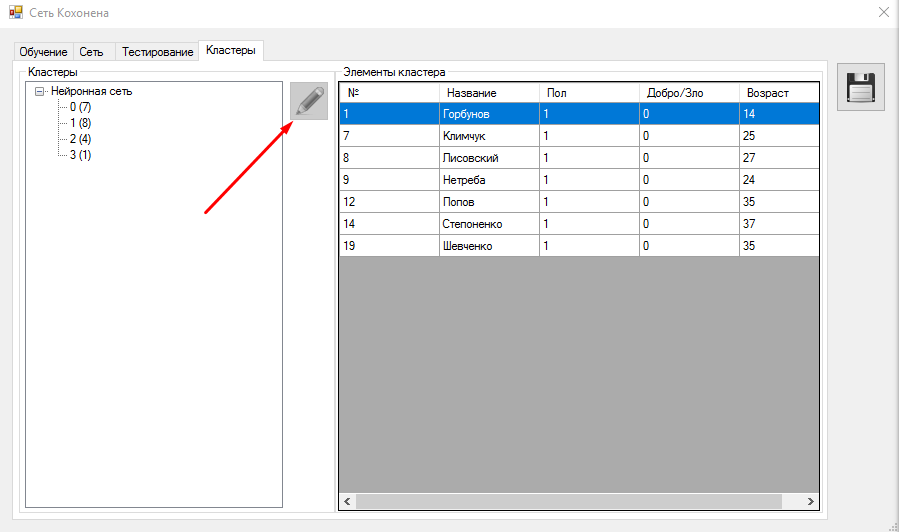
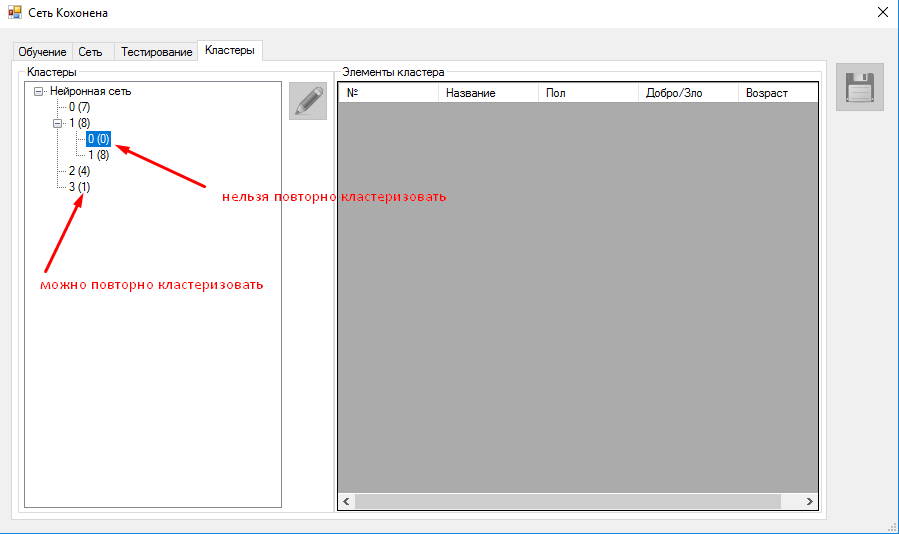


Рис 2.3.1. Кнопка повторной кластеризации

Кнопка активна только при следующих условиях:

* Сеть сохранена
* Выбран родительский кластер из 1-го уровня дерева кластеров
* У выбранного кластера более 0 элементов



Т.е. чтобы повторно кластеризовать кластер 2 уровня, надо выбрать его родительский кластер 1 уровня, зайти в него через кнопку «Редактировать» и уже там повторно кластеризовать кластер.

Еще 1 ограничение – нельзя сохранить уже сохраненный кластер, чтобы не попортить его веса.

При выборке повторной кластеризации откроется форма создания сети – но уже с заданными данными для обучения и тестирования – остается только обучить и потестить данные, а потом сохранить подсеть.