



## АННОТАЦИЯ

Тема: «Разработка алгоритма оценки уровня заинтересованности студентов учебным процессом».

Объектом исследования является процесс определения уровня заинтересованности студентов учебным процессом. В исследовании рассматриваются задачи разработки алгоритма указанного процесса и разработка программной реализации разрабатываемого алгоритма на платформе .NET с применением языка C#.

Структура ВКР представлена введением, тремя главами, заключением, списком литературы.

В введении описывается актуальность проводимого исследования, заключающаяся в отсутствии математического описания определения уровня заинтересованности учебным процессом. В первой главе проводится обзор научных трудов, связанных с определением эмоций и степени вовлеченности, а также алгоритмов big data в рамках решения данного объекта исследования, разрабатывается алгоритм оценки уровня заинтересованности.

Во второй главе описывается задача реализации программного обеспечения по разработанному алгоритму и взаимодействия с внешними системами с использованием особенностей платформы .NET, функционала прогнозирования необходимых параметров.

В третьей главе приводятся результаты тестирования по тесту «Cohn-Kanade», визуализируются результаты работы реализованного программного решения.

Данная бакалаврская работа состоит из пояснительной записки на 53 стр., включая 17 рисунков, 1 таблицу, списка из 23 источников на иностранном языке и 2 источников на русском языке и 4 приложения.

## **ABSTRACT**

Theme: "Algorithm for assessment the level of students interest in the learning process development."

The object of research is the process of determining the level of students interest in the learning process. The study examines the task of developing an algorithm for the specified process and developing a software implementation of the algorithm was developed on the .NET platform using the C # language.

The structure of the bachelor work represented by the introduction, three chapters, conclusion and list of references.

The introduction describes the research relevance, which consists in the absence of a mathematical description of the level of interest in the learning process definition. In the first chapter a review of scientific papers related to the definition of emotions and degree of interest, also big data algorithms within decision of this research object.

The second chapter describes the task of implementing the software according to the developed algorithm and interacting with external systems using the features of the .NET platform, the functionality of forecasting the necessary parameters.

The third chapter presents the test results for the «Cohn-Kanade» test, visualizing the results of the implemented software work.

This bachelor's work consists of an explanatory note on 53 pages, including 17 pictures, 1 table, a list of 23 sources in a foreign language and 2 sources in Russian language and 4 applications.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 АНАЛИЗ ПРОБЛЕМЫ СТЕПЕНИ ВОВЛЕЧЕННОСТИ СТУДЕНТОВ В УЧЕБНЫЙ ПРОЦЕСС .....	8
1.1 Анализ определения уровня заинтересованности по испытываемым личностью эмоциям .....	8
1.2 Изучение возможности использования уже существующих алгоритмов для решения поставленной задачи .....	9
1.3 Разработка под задачу нового алгоритма на основе уже имеющихся .	17
2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ОПРЕДЕЛЕНИЯ СТЕПЕНИ ВОВЛЕЧЕННОСТИ СТУДЕНТОВ В УЧЕБНЫЙ ПРОЦЕСС .....	20
2.1 Составление структуры разрабатываемого программного обеспечения .....	20
2.2 Составление модели базы данных, определение используемой системы управления базой данных .....	28
2.3 Прогнозирование итоговых показателей .....	31
3 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО АЛГОРИТМА .....	35
3.1 Описание тестируемой системы .....	35
3.2 Проведение теста «Cohn-Kanade» .....	41
3.3 Демонстрация работы реализованного программного обеспечения....	43
ЗАКЛЮЧЕНИЕ .....	46
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ .....	48
ПРИЛОЖЕНИЕ А Псевдокод реализации взаимодействия с внешними системами.....	50
ПРИЛОЖЕНИЕ Б Псевдокод аварийной обработки изображений.....	52
ПРИЛОЖЕНИЕ В Изображения из теста Kohn-Kanade для демонстрации работы.....	54
ПРИЛОЖЕНИЕ Г Набор фотографий для теста прогнозирования.....	55

## ВВЕДЕНИЕ

Автоматизация процесса аппаратными средствами определения уровня заинтересованности персон в чем-либо впервые появилась в 2009-м году, и в самом начале точность прогноза уже составляла около 40% по заверению психологов, проверявших результаты работы.

В 2012-м году свет увидел проект «Cohn-Kanade», позволяющий попрактиковаться в написании нейронных сетей, обучающихся на определение уровня заинтересованности персон только по фотографии.

На данный момент активно растет спрос на системы пассивного получения обратной связи компаниями от клиентов о качестве обслуживания, товаров, предоставляемых услуг и т.д. На текущий момент есть несколько систем, позволяющих считывать базовые эмоции посетителей и работников заведения без дальнейшей их обработки, но ни одна из этих систем не позволяет посчитать степень вовлеченности субъектов в какой-либо процесс.

Из косвенных аналогов есть несколько систем для определения вовлеченности персонала в рабочий процесс (например: Keencorp, Affectiva), однако данные системы по большей части опираются на результаты прямых опросов сотрудников, которые могут быть субъективными по ряду причин, из-за чего эти системы нельзя назвать прямым аналогом разрабатываемой системы.

Целью бакалаврской работы является разработка алгоритма для оценки уровня заинтересованности студентов учебным процессом на основе базовых эмоций с применением создаваемого программного обеспечения.

Задачи:

1. Изучить известные алгоритмы определения степени вовлеченности в какой-либо процесс;
2. Проанализировать научные работы в области оценки уровня заинтересованности;
3. Вывести формулу уровня заинтересованности учебным процессом с учетом имеющихся данных по эмоциям каждой персоны;

4. Разработать архитектуру приложения для автоматизации процесса оценки уровня заинтересованности;
5. Реализовать программное обеспечение по разработанной архитектуре.

Объект бакалаврской работы: процесс определения степени вовлеченности студентов в учебный процесс. Предмет бакалаврской работы: алгоритмизация степени вовлеченности на основе базовых эмоций. Данная работа основана на практическом методе исследования с применением моделирования.

Данная работа, как было написано ранее, не имеет прямых аналогов при условии учета вовлеченности с упором на пассивные методы получения информации. На текущий момент работа может использоваться в учебных учреждениях, однако при корректировке множителей эмоций может использоваться также в любом другом месте, где необходимо высчитывание вовлеченности людей во что-либо, будь то работа, развлечения или что бы то ни было еще.

В первой главе проведен анализ трудов Кана и Экмана по интересующей нас теме, подобраны самые подходящие под задачу и в рамках высчитывания степени вовлеченности в учебный процесс рассмотрены алгоритмы Big Data и составлен новый алгоритм под данные условия на основе рассмотренных алгоритмов.

Во второй главе описывается процесс построения архитектуры и реализации программного обеспечения для автоматизации составленного алгоритма с применением внешних систем для определения эмоций и визуализации данных.

В третьей главе проведено тестирование реализованного программного обеспечения на проекте «Cohn-Kanade», результаты точности вычислений сопоставлены средним показателям по всем пользователям проекта.

Выбранная тема совпадает с профессиональными компетенциями, поскольку здесь используется анализ имеющихся математических моделей

обработки данных и составление новой, также происходит реализация программного обеспечения.

# **1 АНАЛИЗ ПРОБЛЕМЫ СТЕПЕНИ ВОВЛЕЧЕННОСТИ СТУДЕНТОВ В УЧЕБНЫЙ ПРОЦЕСС**

## **1.1 Анализ определения уровня заинтересованности по испытываемым личностью эмоциям**

Как отмечал в своем труде об условиях вовлеченности Уильям Кан [20], «для определения степени вовлеченности персоны в какое-либо действие недостаточно просто посмотреть на его лицо и сказать, что данного человека не интересует или интересуется данный акт, - спектр его эмоций может отличаться от стандарта того, кто проводит определение, потому что нужно учитывать ранее испытываемые эмоции при составлении оценки, иначе результаты можно считать субъективными», что не дает нам конкретной схемы определения интересующего нас показателя, однако наводит на мысль, что для определения степени вовлеченности необходимо опираться на исторические показатели эмоционального состояния по каждой персоне. Пол Экман в своей работе по системе кодирования лицевых движений [9] пишет, что по проявляемым основным эмоциям нельзя однозначно сказать, насколько человек испытывает то или иное «компонентное» состояние, в том числе вовлеченность во что-либо, поскольку у каждой персоны данные состояния имеют индивидуальное проявление, однако их можно посчитать, зная спектр выражения каждой из эмоций, и подобрав значимость каждой из эмоций и степень ее влияния на результирующий показатель.

Опираясь на вышеописанные выводы психологов, посвятивших часть своей жизни на определение эмоционального состояния человека и изучение значимости каждой из эмоций можно с точностью сказать, что нам понадобятся значения по каждой из эмоций, несколько дополнительных значений для точности результата, а также набор данных по предыдущим эмоциям персоны за определенный промежуток времени для решения подобного рода задачи.

Исходя из физических особенностей проявления эмоционального состояния человека можно выделить 7 основных типов эмоций, в той или иной мере определяемых по чертам лица субъекта: злость, презрение, отвращение,



страх, грусть, радость, удивление. Также можно выделить степень нейтральности текущего состояния, степень визуализации хорошего состояния через улыбку и в качестве дополнительного параметра для точности результата резкость изображения.

В качестве примера используется проведение лекции на паре в университете. Основная задача текущей работы состоит в определении общей степени заинтересованности всех субъектов в преподносимом лектором материале на основе эмоционального состояния каждого из субъектов. В роли исходных данных при указанных условиях будут выступать данные об эмоциональном состоянии каждой персоны, включающие в себя показатели каждой из эмоций, уровень улыбки и уровень размытия изображения, на протяжении каждой пары из выбираемого диапазона дат/предметов/пар/университетов и т.д. В текущем случае процесс идентификации человека и определения по чертам лица эмоционального состояния отводится на внешнюю систему, которая предоставляет готовые показатели по каждой из эмоций, а также степень улыбки, если она присутствует, и степень размытия лица на изображении, что может отразиться на итоговом результате.

Не все эмоции могут одинаково повлиять на степень вовлечённости, например: нейтральное отношение имеет низкое влияние на итоговое значение, а отвращение и вовсе приносит отрицательное влияние. Из вышеназванного следует вопрос: как на основе этих данных получить всего один показатель? Для ответа на этот вопрос нужно обратиться к существующим алгоритмам Big Data и машинного обучения, но об этом ниже.

## **1.2 Изучение возможности использования уже существующих алгоритмов для решения поставленной задачи**

На момент написания данного материала существует несколько вариантов возможного решения вышеописанной задачи:

Обучение искусственной нейронной сети. Из всего многообразия различных архитектур нейросетей больше всего подходит перцептрон, так как позволяет начальную группу параметров (весов) разделить на несколько более усредненных параметров (весов) в зависимости от их весовых коэффициентов, а затем объединить их в результирующий элемент (вес) все так же на основе весовых коэффициентов, что изображено на рисунке 1.1, функция последнего представлена в формуле 1.1.

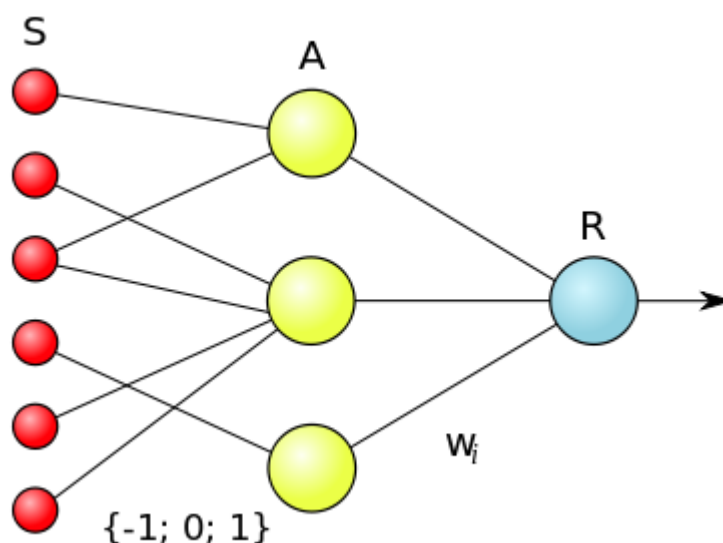


Рисунок 1.1 – Логическая схема элементарного перцептрона

$$f(x) = \text{sign}\left(\sum_{i=1}^n \omega_i x_i - \theta\right), \quad (1.1)$$

где  $n$  – количество предстоящих элементов в схеме,

$\omega_i$  – весовой коэффициент  $i$ -элемента,

$\theta$  – порог значения, определяющий выходящее значение  $R$ -элемента.

При том, если углубляться более подробно, для текущей задачи логичнее всего будет выбрать перцептрон с перекрестными связями, поскольку среди некоторых из передаваемых параметров (в данном случае – эмоций субъекта) могут существовать связи между собой, что может повлиять на исходный результат.

Принцип работы на примере текущей системы: нейросеть получает совокупность входных данных в виде упоминаемых выше результатов работы

внешней системы и результатов работы нашей системы по этим данным, нейросеть эти данные обрабатывает и учится на их основе автоматически подбирать результат от последующих передаваемых значений от внешней системы, построив алгоритмическую модель вычисления результата. К нашей задаче этот способ неприменим, потому что на данный момент описания работы предоставить достаточное для приемлемой точности работы нейросети количество информации невозможно по причине отсутствия автоматизированного механизма получения степени вовлеченности на основе получаемых от внешней системы данных, а при определении данного параметра алгоритмическим путем не имеет смысла использовать обучение нейросети, поскольку результат работы этого действия нам будет и так известен.

Обработка имеющихся показателей с использованием алгоритмов Big Data. Нельзя посчитать все количество данных алгоритмов, так как они создаются самыми разными группами лиц под совершенно разные задачи с использованием особенностей этих задач, потому мы будем выбирать только из основных, которые можно найти в большинстве книг, статей и прочих источниках информации. Каждый из них рассчитан на разные виды обработки данных, соответственно, и результат у каждого из них будет отличаться от других. Из них наиболее близкими к текущей задаче являются алгоритм k-means и Naive Bayes, разберём каждый из них:

K-means (К-средних). Сам алгоритм стремится минимизировать отклонение точек кластеров от центров кластеров по формуле (1.2), визуальное представление которой предоставлено на рисунке 1.2.

$$V = \sum_{i=1}^k \sum_{x \in S_i} (x - \mu_i)^2, \quad (1.2)$$

где  $k$  – число кластеров,

$x$  – вектор,

$S_i$  – полученные кластеры,

$\mu_i$  – центры всех векторов  $x$  из кластера  $S_i$ .

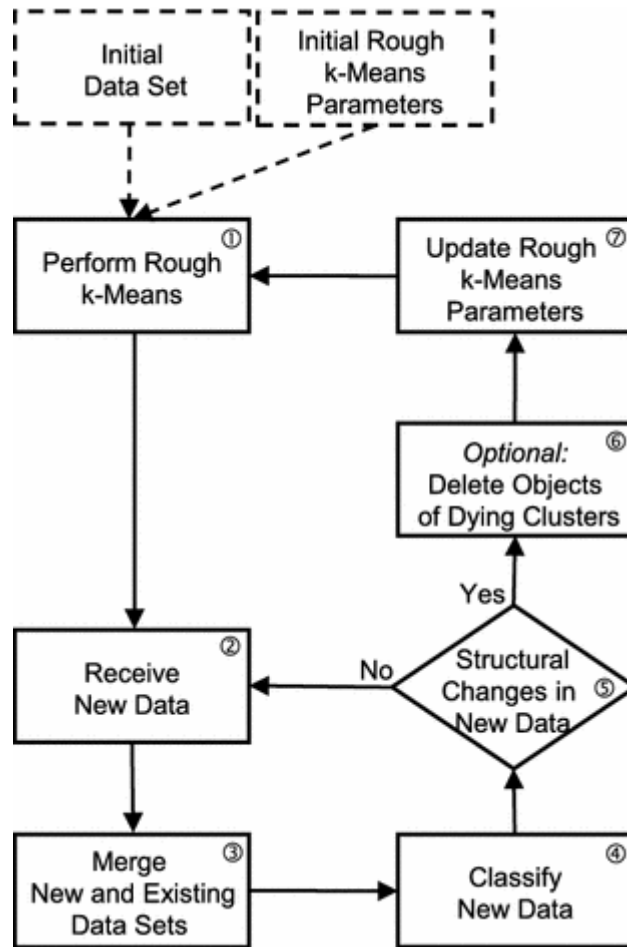
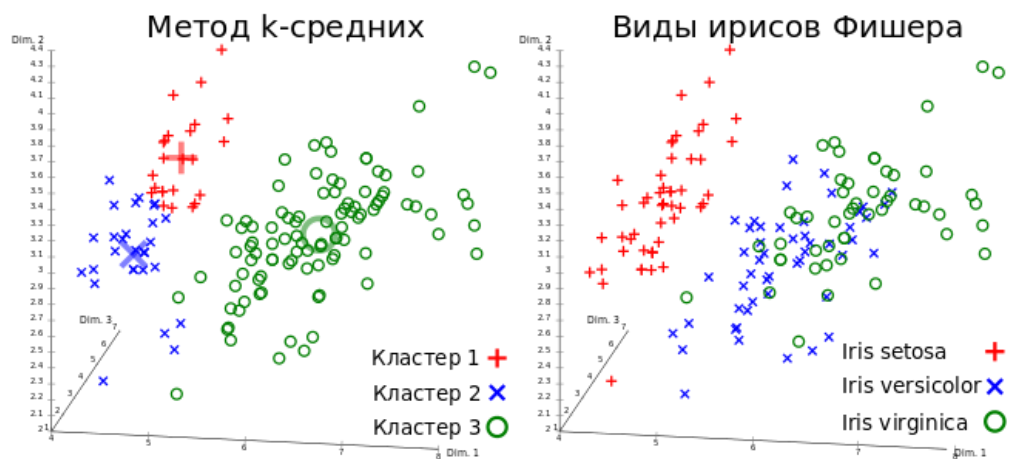


Рисунок 1.2 – Схема работы алгоритма К-means

Наглядно работу алгоритма можно увидеть с применением ирисов Фишера, что отображено на рисунке 1.3, – набора данных для задачи классификации, на примере которого была продемонстрирована работа разработанного метода дискриминантного анализа [5].



### Рисунок 1.3 – Результат метода k-means и базовый вид данных

Изначально этот алгоритм предлагает указать в результатах работы внешней системы несколько точек в качестве центров с дальнейшим изменением параметров каждой из групп и проводить на этой основе группировку данных в зависимости от удаленности каждой из точек группы, затем проводить перегруппировку точек на основе расстояния до ближайшего центра, повторение происходит до тех пор, пока группы не останутся неизменными. Подобный механизм подойдет в том случае, когда нам надо уточнить важность каждой из эмоций для итогового результата. Полностью алгоритм для нашей задачи использовать нельзя, потому что он не может предоставить результирующего параметра степени вовлеченности.

Naive Bayes (Наивная байесовская модель). Следует уточнить: в большинстве случаев при упоминании использования Naive Bayes подразумевается использование Наивного байесовского классификатора, построенного на основе Наивной байесовской модели, но не использующего Байесовскую сеть доверия, за исключением подкрепления подсчетов на основе теоремы Байеса. Классификатор позволяет просчитать вероятность отнесения к одной из двух групп определенного элемента с использованием логарифма правдоподобия [6]. Используя пример из поставленной задачи:

Нам нужно определить вероятность отнесения эмоции  $E$  к абстрактному классу  $S$  на основе нескольких показателей этой эмоции, что продемонстрировано в формуле (1.3).

$$p(E|S) = \prod_i p(e_i|S), \quad (1.3)$$

где  $e_i$  –  $i$ -й показатель эмоции.

Исходя из теоремы Байеса мы получим равенство, представленное в формуле (1.4).

$$p(S|E) = \frac{p(S)}{p(E)} p(E|S). \quad (1.4)$$

Допустим, есть классы G и B – показатели отнесения эмоции к «хорошему» и «плохому» показателям соответственно. Тогда при подстановке значений в имеющееся уравнение получим две новых формулы: формулу (1.5) и формулу (1.6).

$$p(G|E) = \frac{p(G)}{p(E)} \prod_i p(e_i|G), \quad (1.5)$$

$$p(B|E) = \frac{p(B)}{p(E)} \prod_i p(e_i|B). \quad (1.6)$$

После чего можно вывести формулу правдоподобия, опираясь на ранее приведенные формулы, что приведет к получению формулы (1.7).

$$\frac{p(G|E)}{p(B|E)} = \frac{p(G)}{p(B)} \prod_i \frac{p(e_i|G)}{p(e_i|B)}. \quad (1.7)$$

Из чего можно получить алгоритм правдоподобия, математическая модель которого описана в формуле (1.8).

$$\ln \frac{p(G|E)}{p(B|E)} = \ln \frac{p(G)}{p(B)} + \sum_i \ln \frac{p(e_i|G)}{p(e_i|B)}, \quad (1.8)$$

После чего при условии, что  $p(G|E) + p(B|E) = 1$ , можно высчитать действительную вероятность  $p(G|E)$  из  $\ln \frac{p(G|E)}{p(B|E)}$  по формуле (1.9).

$$p(G|E) = \frac{e^q}{1 + e^q}, \quad (1.9)$$

где  $q$  – натуральный логарифм  $\ln \frac{p(G|E)}{p(B|E)}$ .

Для дальнейшей классификации предлагается использовать логарифм правдоподобия, но поскольку в данной ситуации требовалось только определение вероятности отнесения эмоции к каждому из классов, то этим предложением можно пренебречь.

На основе данного алгоритма может функционировать также и еще один популярный алгоритм из Big Data – Decision Tree (Дерево решений), схема работы которого изображена на рисунке 1.4, в текущий список рассматриваемых решений он не попал как раз по причине использования

внутри реализации Naive Bayes, разбирать более обросшие дополнительными условиями, неприменимыми в виду специфики обработки данных в вышеописанной задаче, копии алгоритмов считается излишним.

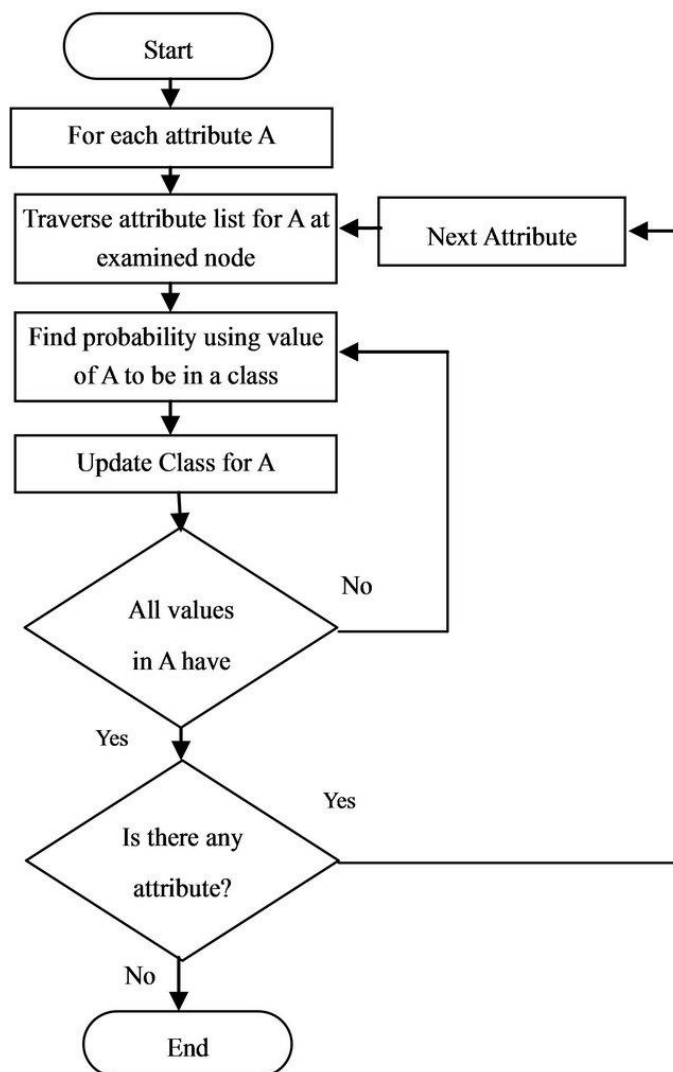


Рисунок 1.4 – Схема работы Деревя решений

В своей изначальной интерпретации постановка задачи, для которой составлен алгоритм Naive Bayes, состояла в выделении из текста некоторых слов-маркеров и назначения для них степени отнесения к «хорошим» и «плохим» показателям, в результате чего разница суммы этих показателей должна была дать оценку настроения, которое автор пытался, хоть и не всегда осознанно, передать в данном тексте. Для нашей задачи алгоритм по большей части подходит, но есть конкретный недочет в случае использования: алгоритм

не учитывает данные, полученные ранее, что в описываемой задаче может снизить точность статистики, что, является недопустимым в данном случае.



### 1.3 Разработка под задачу нового алгоритма на основе уже имеющихся

Полагаясь на все вышеперечисленное, следует для данной бакалаврской работы вместо трудоемкой разработки абсолютно нового адаптировать существующие решения. В силу психофизических и психосоматических особенностей каждого из субъектов и людей в целом нельзя полагаться на заранее устанавливаемые параметры по каждой эмоции в качестве точек начала и конца просчета данных, из-за чего необходимо привлекать данные из предыдущих записей, на их основе и соседних значений проводить кластеризацию по алгоритму k-means и брать значение центра второго кластера для снижения погрешностей при наличии крайних точек во множестве. Из Naïve Bayes при текущей постановке задачи понадобится разделение всех эмоций на две категории и расчет уровня влияния на итоговый показатель каждой эмоции для дальнейшей апроприации значения уровня заинтересованности учебным процессом на основе показателей его эмоционального состояния, блок-схема данного процесса продемонстрирована на изображении 1.5.

Пошаговое воспроизведение предполагаемой архитектуры разрабатываемого алгоритма:

- 1) за предыдущие k-записей собираются все данные, по каждой из записей производится кластеризация с привлечением соседних данных для получения более усредненного параметра для большей точности дальнейших просчетов;
- 2) находятся максимальное и минимальное значения за все время по каждой из эмоций и высчитывается на их основе и параметра из шага 1 уровня проявления эмоции относительно предыдущих данных;
- 3) высчитывается вероятность отнесения к «хорошему» и «плохому» классам эмоционального состояния по каждой переменной набора эмоций  $x$ ;

- 4) на основе полученного списка идет расчет множителя для «хорошего» и «плохого» классов нового показателя по текущей эмоции из набора параметров  $x$ ;
- 5) шаги 1-4 повторяются для каждой из  $n$  эмоций, результирующие показатели суммируются между собой и в конце делятся на количество эмоций;
- б) итоговый показатель  $f(x)$  является показателем оценки уровня заинтересованности студента учебным процессом и находится в диапазоне  $[-1; 1]$ .

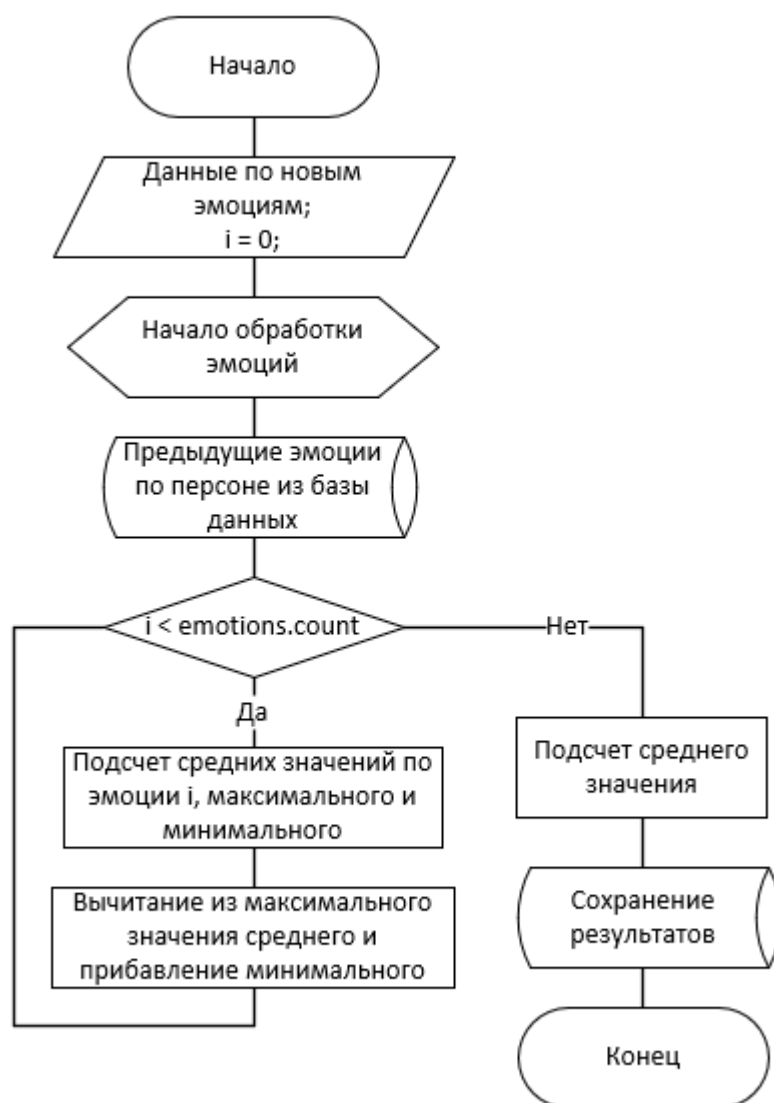


Рисунок 1.5 – Блок-схема работы составленного алгоритма

С математической стороны вопроса вышеописанный процесс можно представить в виде равенства, описанного в формуле (1.10).

$$f x = \frac{\sum_{i=1}^n \frac{x_i}{(\max q_i - \frac{\sum_{j=1}^k q_{ij}}{k} + \min q_i)} * m_i}{n}, \quad (1.10)$$

где  $x$  – набор текущих эмоций персоны,

$q_i$  – данные по эмоции  $i$  за все время,

$q_{ij}$  – данные по эмоции  $i$  в момент времени  $j$ ,

$m_i$  – множитель итогового значения для эмоции  $i$ .

Однако итоговый показатель не может передать полной картины уровня заинтересованности студентов учебным процессом, из-за чего стоит рассмотреть вариант с разделением ранее рассчитываемых показателей для «хорошего» класса и «плохого» класса. Данный выбор обуславливается тем, что при наличии высоких показателей у переменных для «хорошего» класса в тот же момент времени могут быть средние показатели у переменных «плохого» класса, что снизит итоговый показатель до низких показателей, при этом скрыв критично большое количество важной информации.

В итоге проделанной работы был разработан алгоритм оценки уровня заинтересованности студентов учебным процессом, в котором учитываются все ранее полученные эмоции для точности расчетов, при этом данный алгоритм может использоваться не только для определения итогового уровня вовлеченности, но и для определения только «хорошего» показателя эмоций и только «плохого» показателя, из которых в том числе состоит итоговый параметр, что может расширить возможности для дальнейшего анализа данных.

## 2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ОПРЕДЕЛЕНИЯ СТЕПЕНИ ВОВЛЕЧЕННОСТИ СТУДЕНТОВ В УЧЕБНЫЙ ПРОЦЕСС

### 2.1 Составление структуры разрабатываемого программного обеспечения

В ходе обсуждения структуры проекта было решено принимать данные от внешних системы для идентификации субъектов и распознавания эмоционального состояния и системы визуализации данных по технологии WebSocket [24], поскольку она позволяет для общения между клиентами передавать и получать данные без задержек на согласование отправки с обеих сторон, единожды соединившись и до момента отключения одного из участников соединения. И, в отличие от вышеуказанных внешних систем, в данной части всего проекта было решено создать два разных канала WebSocket, использование разрабатываемой программы одного из которых изображено на рисунке 2.1, для разделения данных внешних систем между собой в целях безопасности и большей устойчивости от ошибок от каждой из сторон.

```
var webSocketApplier = new WebSocketServer(  
    $"{wsType}://0.0.0.0:{Settings["WSInputPort"]}");  
  
webSocketApplier.Start(socket =>  
{  
    socket.OnOpen = () =>  
        Console.WriteLine($"New connection to applier" +  
            $" from {socket.ConnectionInfo.ClientIpAddress}");  
  
    socket.OnClose = () =>  
        Console.WriteLine($"Removed connection to applier" +  
            $" from {socket.ConnectionInfo.ClientIpAddress}");  
  
    socket.OnError = e =>  
    {  
        Console.WriteLine($"Error in connection to applier" +  
            $" with {socket.ConnectionInfo.ClientIpAddress}: {e.Message}");  
    };  
  
    socket.OnMessage = msg =>  
    {  
        var message = msg.FromJson<List<IAcceptanceData>>();  
        if (message == null) return;  
  
        mainDataProcessing.AddDataToProcessing(message).GetAwaiter().GetResult();  
    };  
});
```

Рисунок 2.1 – Применение WebSocket в разрабатываемом ПО

По причинам различия получаемых данных от внешних систем, оптимизации затрачиваемых ресурсов сервера и создания возможности разделения трафика при наличии нескольких источников передачи данных было решено разделить взаимодействие с внешними системой идентификации пользователей и распознавания их эмоционального состояния и системой визуализации данных на разные каналы WebSocket. По большей части данное решение исходит из причины оптимизации ресурсов сервера: поддержание дополнительного канала WebSocket занимает 4 килобайта оперативной памяти без дополнительной нагрузки на процессор, когда определение получаемого объекта требует 8 килобайт памяти на каждый предполагаемый тип объекта и задействует ресурсы процессора [17].

Также было решено при соответствующем запросе от системы визуализации данных передавать обработанные данные по мере их поступления в базу данных без прогнозирования данных, что позволяет конечным пользователям при необходимости видеть результаты работы системы в реальном времени.

В реализуемое программное обеспечение разрабатываемого алгоритма анализа больших данных решено добавить функционал обработки входящих запросов от вышеуказанных внешних систем, хранения обрабатываемых данных на сервере с базой данных под управлением СУБД Aerospike и получения их из этой базы, аварийной обработки изображений на случай внештатных ситуаций на стороне внешней системы для идентификации субъектов и распознавания эмоционального состояния каждого из них, а также отправки справочников, – списка сущностей, по которым может быть произведен запрос на получение данных, - при подключении внешней системы визуализации данных и отправки данных сразу после получения от первой системы и обработки согласно разрабатываемому алгоритму, логическая модель данных для которого отображена на рисунке 2.2.



- Date – Дата снимка с наличием на ней персон(-ы) в формате DateTime, допускается указание любого времени, но требуется подлинная дата в виду критичности учета последнего и отсутствия использования первого, является критичным для итоговой статистики;
- ClassroomId – Идентификатор кабинета в целочисленном формате, в котором или рядом с которым был сделан снимок, критичный параметр для возможности составления статистики по кабинетам, но некритичный для определения степени вовлеченности студентов;
- Lesson – Порядковый номер промежутка времени (в данном случае пары), в который был сделан снимок, имеет формат целого числа, критичен для составления большинства видов статистики;
- CurrentDataCounter – Порядковый номер создания снимка от момента начала текущего промежутка времени (в данном случае пары) в формате целого числа, критичен для составления детальной статистики по паре(-ам);
- список объектов Emotions типа PersonEmotions, который состоит из:
  - PersonId – Идентификатор в целочисленном формате персоны, по которой определен набор эмоций;
  - Smile – Дробное число, обозначающее степень улыбки у персоны, имеет диапазон [0, 1];
  - BlurLevel – Дробное число, обозначающее степень размытости изображения, может повлиять на качество оценки итогового параметра, имеет диапазон [0, 1];
  - Anger – Дробное число, отображающее степень злости персоны, имеет диапазон [0, 1];
  - Contempt – Дробное число, показывающее степень презрения в текущий момент времени, имеет диапазон [0, 1];
  - Disgust – Дробное число, содержащее степень отвращения персоны, имеет диапазон [0, 1];

- Fear – Дробное число, отображает степень страха/испуга на данный момент, имеет диапазон [0, 1];
- Happiness – Дробное число, показывает степень счастья/радости персоны, имеет диапазон [0, 1];
- Neutral – Дробное число, содержащее степень нейтральности к происходящему у субъекта, имеет диапазон [0, 1];
- Sadness – Дробное число, отображающее степень грусти/несчастья в текущий момент времени, имеет диапазон [0, 1];
- Surprise – Дробное число, показывающее степень удивления на данный момент, имеет диапазон [0, 1].

Для взаимодействия с системой визуализации данных из-за функционального предназначения реализуемого в рамках данной бакалаврской работы программного обеспечения передаваемые данные делятся на 2 вида: объект словарей и объект данных об эмоциональном состоянии субъектов. Первый вид состоит из всех объектов, реализуемых на этапе построения базы данных (Рисунок 8), а второй вид содержит следующие переменные:

- EntityName – Название сущности, по которой производится выборка данных, представляет собой массив объектов типа char, в большинстве языков программирования представленный классом String;
- список объектов Results типа ResultEntity, который состоит из:
- Date – Дата создания снимка с персонами в формате DateTime, в качестве времени содержит полдень по часовому поясу сервера, на котором расположено описываемое ПО;
- Lesson – Порядковый номер промежутка времени (в данном случае пары), в который был сделан снимок, имеет формат целого числа;
- список объектов PersonEmotions, реализующих тип PersonEmotionsEntity, в свою очередь представляющий следующие объекты:
  - PersonId – Идентификатор в целочисленном формате персоны, по которой определена степень вовлеченности в учебный процесс;



- GoodEmotionsList – Список «хороших» эмоций в формате дробных чисел, имеющих диапазон  $[0, 1]$ , отсортированный по порядковому номеру создания снимка от момента начала текущего промежутка времени (в данном случае пары);
- BadEmotionsList – Список «плохих» эмоций в формате дробных чисел, имеющих диапазон  $[0, 1]$ , отсортированный по порядковому номеру создания снимка от момента начала текущего промежутка времени (в данном случае пары);
- GoodEmotionPrediction – Результат предугадывания «хорошей» эмоции на основе текущего промежутка времени (в данном случае - пары) по текущей персоне через определенное количество аналогичных промежутков времени (количество промежутков настраиваемо);
- BadEmotionPrediction – Результат предугадывания «плохой» эмоции на основе текущего промежутка времени (в данном случае - пары) по текущей персоне через определенное количество аналогичных промежутков времени (количество промежутков настраиваемо).

Было решено передавать системе визуализации данных не одно суммарное значение по всем эмоциям, а два суммарных значения, представляющих из себя показатели двух разных групп данных, отличающихся видом влияния на ключевой параметр, поскольку возможность анализа разницы между этими двумя параметрами может принести конечному пользователю (в данном случае – преподавателю или представителю администрации) более детальную информацию о текущей или прошедшей ситуации с степенью вовлеченности студентов в учебный процесс.

Аварийная обработка представляет собой получение изображения и списка персон на изображении от внешней системы, которые она не успела обработать, для дальнейшего запроса в FaceApi [12], владельцем которого является компания Microsoft и обработка через который доступна только через отправку запроса на их сервер, на который есть лимит на отправку данных в секунду, либо через покупку корпоративной версии сервера для данного

продукта. Данная система считывает лицо человека на изображении при помощи сверточных нейронных сетей, затем на основе более восьмидесяти точек и движений на лице человека определяет уровень каждой из семи основных эмоций: удивления, страха, радости, печали, отвращения, презрения и гнева [21]. Все эти точки и движения описаны в трудах Пола Экмана и Уоллеса Фризена, в том числе в разработанной ими Системе кодирования лицевых движений, данные из которой пользуются популярностью у студентов, изучающих психологию, социологию, журналистику, юриспруденцию, многие медицинские направления.

Программное обеспечение было решено разрабатывать на платформе .NET, а именно на .NET Core – мультиплатформенной версии .NET Framework с некоторыми доработками [25]. Поскольку управление системой может осуществляться со стороны внешней системы визуализации данных, то в качестве типа приложения было выбрано консольное приложение.

Все вычисления в разрабатываемом приложении происходят в многопоточном режиме, для чего используется класс Parallel [15], фрагмент кода с его применением в разрабатываемом программном обеспечении продемонстрирован на рисунке 2.3. Так как данные не зависят друг от друга, и архитектура изначально разработана с низкой взаимозависимостью компонентов в плане обязательного внутривидеопотокового взаимодействия друг с другом, то никакие дополнительные средства в виде семафоров или мониторов для синхронизации потоков для стабильной работы приложения никак не пригодятся [19].

Благодаря постоянно растущему рынку облачных вычислений и циклу обновлений платформы .NET Core для разработки можно без труда использовать сервера Linux, что возможно благодаря кроссплатформенности .NET Core [2]. Если необходима бесперебойная работа системы, а в наличии есть минимум 2 виртуальных сервера, то благодаря контейнеризации [4] можно собрать свое приложение вместе со всеми зависимостями у себя на компьютере и развернуть контейнеры с приложением на вышеупомянутых серверах, что

придаст разрабатываемой системе бесперебойность работы даже в случае отключения одного из серверов [7].

```
Parallel.ForEach(query.OrderBy(x => x.Date).ThenBy(x => x.Lesson).ToList(), x =>
{
    var resultPersonEmotionsList = new List<PersonEmotionsEntity>();
    var personList = db.PersonEmotions.Where(p => p.SubjectEmotionsId == x.EmotionRatingAtSubjectId)
        .GroupBy(p => p.PersonId).ToList();

    // Getting data for each person at subject.
    Parallel.ForEach(personList, p =>
    {
        float? badPrediction = null;
        float? goodPrediction = null;

        // Prediction mustn't calculate at currently waiting request ('Till' date > DateTime.Now).
        if (isNeedPrediction)
        {
            badPrediction = Prediction(p, true);
            goodPrediction = Prediction(p, false);
        }

        // Adding person's emotions at subject to list.
        resultPersonEmotionsList.Add(new PersonEmotionsEntity {
            PersonId = p.Key,
            BadEmotionsList = p.Select(l => l.BadEmotionsRating).ToList(),
            GoodEmotionsList = p.Select(l => l.GoodEmotionsRating).ToList(),
            BadEmotionPrediction = badPrediction,
            GoodEmotionPrediction = goodPrediction
        });
    });

    // Add result entity to list.
    entities.Add(new ResultEntity
    {
        Date = x.Date,
        Lesson = x.Lesson,
        PersonEmotions = resultPersonEmotionsList,
        VoiceLevel = x.VoiceLevel.FromJson<List<int>>(),
        VoiceTiembre = x.VoiceTiembre.FromJson<List<int>>()
    });
});
```

Рисунок 2.3 – Фрагмент кода параллельных вычислений

Однако для разрабатываемой в рамках данной бакалаврской работы системы, учитывая ее специфику и уровень нагрузки, необходима не только бесперебойная работа, но и возможность масштабирования, как, например, это реализовано с использованием технологии Hadoop [10] в некоторых языках программирования: заводится кластер серверов с установленным на них Hadoop и соединенных между собой настройками самого Hadoop, затем на один из этих серверов переносится приложение, которое необходимо

масштабировать, и запускается процесс масштабирования через сам Hadoop, через какое-то время система сама подскажет, было ли установлено программное обеспечение на все серверы в кластере, и приложение начнет работать в единственном экземпляре без опасок на отключение какого-либо узла в кластере и со всей вычислительной мощностью кластера, хоть и экземпляров приложения несколько и они разбросаны по всему кластеру. Аналогичная технология не так давно появилась и у Microsoft для приложений на .NET – HDInsight. Оно содержит в себе Hadoop, Spark, Kafka и другие технологии для создания кластеров серверов для непрерывного вычисления больших объемов данных, выбор какой-то конкретной технологии остается за пользователем [14].

## **2.2 Составление модели базы данных, определение используемой системы управления базой данных**

Изначально для разработки программного обеспечения для автоматизации расчетов разрабатываемого в рамках данной бакалаврской работы алгоритма обработки больших данных была выбрана система управления базой данных Aerospike [3], однако уже на этапе реализации было заметно снижение скорости разработки за счет отсутствия в Aerospike обработки запросов на языке SQL, в следствие чего невозможность использовать различные дополнительные особенности выбранной платформы .NET, такие как LINQ to SQL и фреймворк Entity Framework [1], затем при больших нагрузках на сервер базы данных скорость обработки упала достаточно сильно, чтобы провести новое сравнение быстродействия СУБД Aerospike, на этот раз с СУБД MSSQL, отлично оптимизированной для работы через LINQ to SQL и Entity Framework, а также поддерживающей SQL в запросах, и результаты оказались не совсем такими, как было при сравнении только СУБД типа NoSQL: при записи, чтении и чтении дробных чисел Aerospike справляется лучше MSSQL на 27.5%-32.5%. Но не все так однозначно, как могло показаться после первого теста: в базах данных

существует такой механизм, как индексы [16], благодаря которому чтение за счет составления из данных бинарного дерева для хранения информации происходит в несколько раз быстрее (в данном случае – от 14 до 24 раз), но при этом замедляет запись в базу на 15% (Aerospike) - 30% (MSSQL) [23], что отображено на рисунке 2.4. Учитывая специфику работы алгоритма, – чтение истории нескольких предыдущих записей по эмоциям на добавление одной новой записи, - снижение скорости записи нивелируется многократно возросшей скоростью чтения.

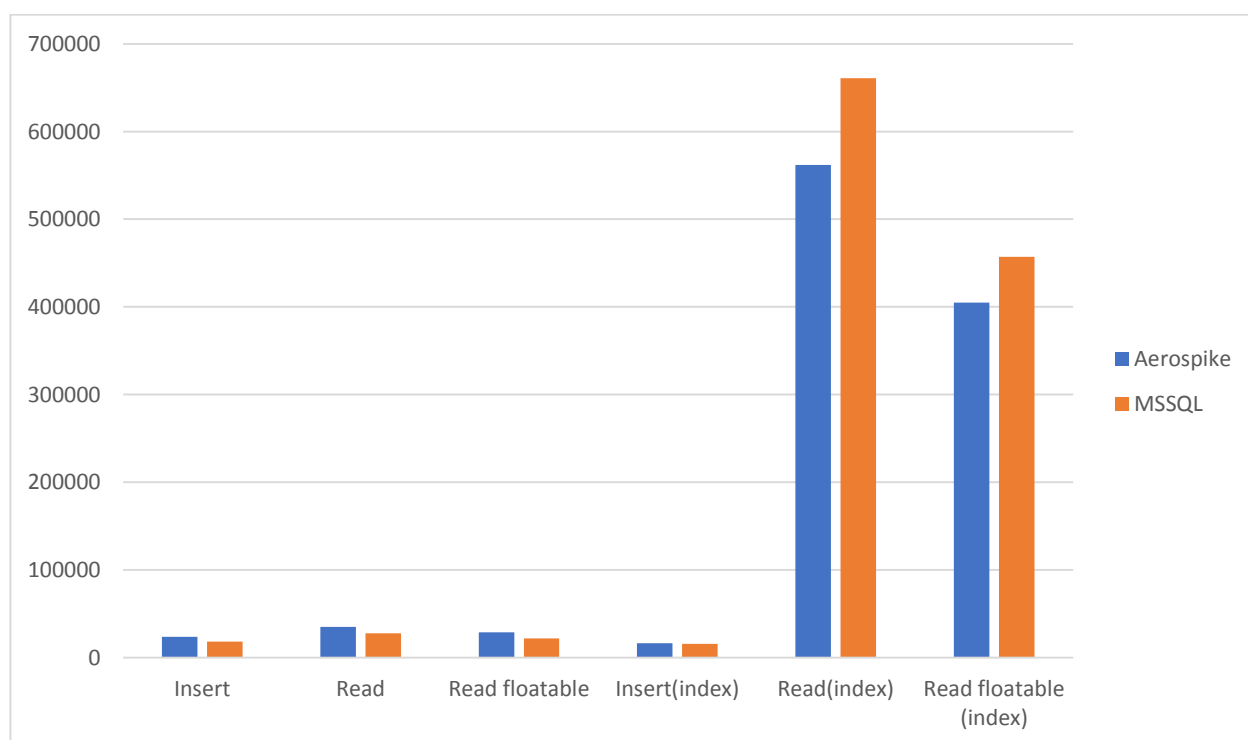


Рисунок 2.4 – Сравнение Aerospike и MSSQL

После настройки на таблицах данных индексов [22] результаты сравнения выставляют Aerospike под данную задачу не в лучшем свете: из-за постоянной нагрузки на сервер базы данных изначальное преимущество этой СУБД ослабевает по неизвестным на то причинам и подталкивает к выбору более надежного аналога.

После такого крупного изменения в разработке итоговый продукт никак не изменился, однако дальнейшее его обслуживание будет проходить гораздо легче, а разработка быстрее из-за выше указанных преимуществ и из-за

возможности создавать и редактировать базу данных напрямую из кода проекта, что и было сделано в первую очередь.

Поскольку была изменена система управления база данных на MSSQL, то в ту же секунду было решено переписать структуру базы данных, изображенной на рисунке 2.5, на подход CodeFirst [18], благодаря которому Entity Framework все классы, записанные в специальном контексте, создаваемом разработчиками, переводит в сущности базы данных [8], пример которых приведен на рисунке 2.6.

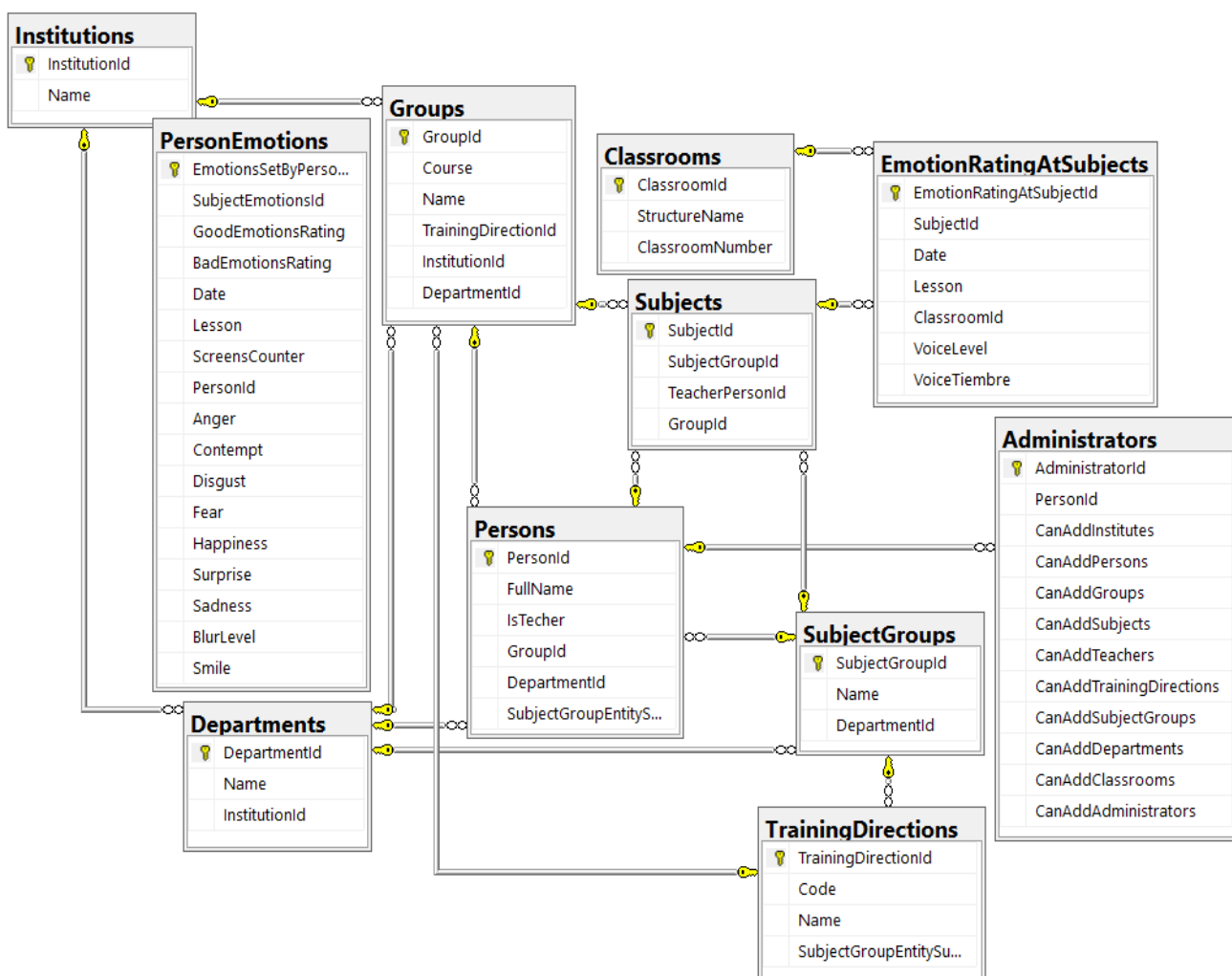


Рисунок 2.5 – Диаграмма представления сущностей базы данных

Также для увеличения информированности о степени вовлеченности студентов в учебный процесс было решено добавить возможность прогнозирования эмоционального состояния на основе данных за текущий промежуток времени на настраиваемое количество итераций наперед, что

позволит заинтересованным лицам получить более детальное представление о степени вовлеченности на финальных частях временного промежутка, что позволит сделать соответствующие выводы для дальнейшей обработки.

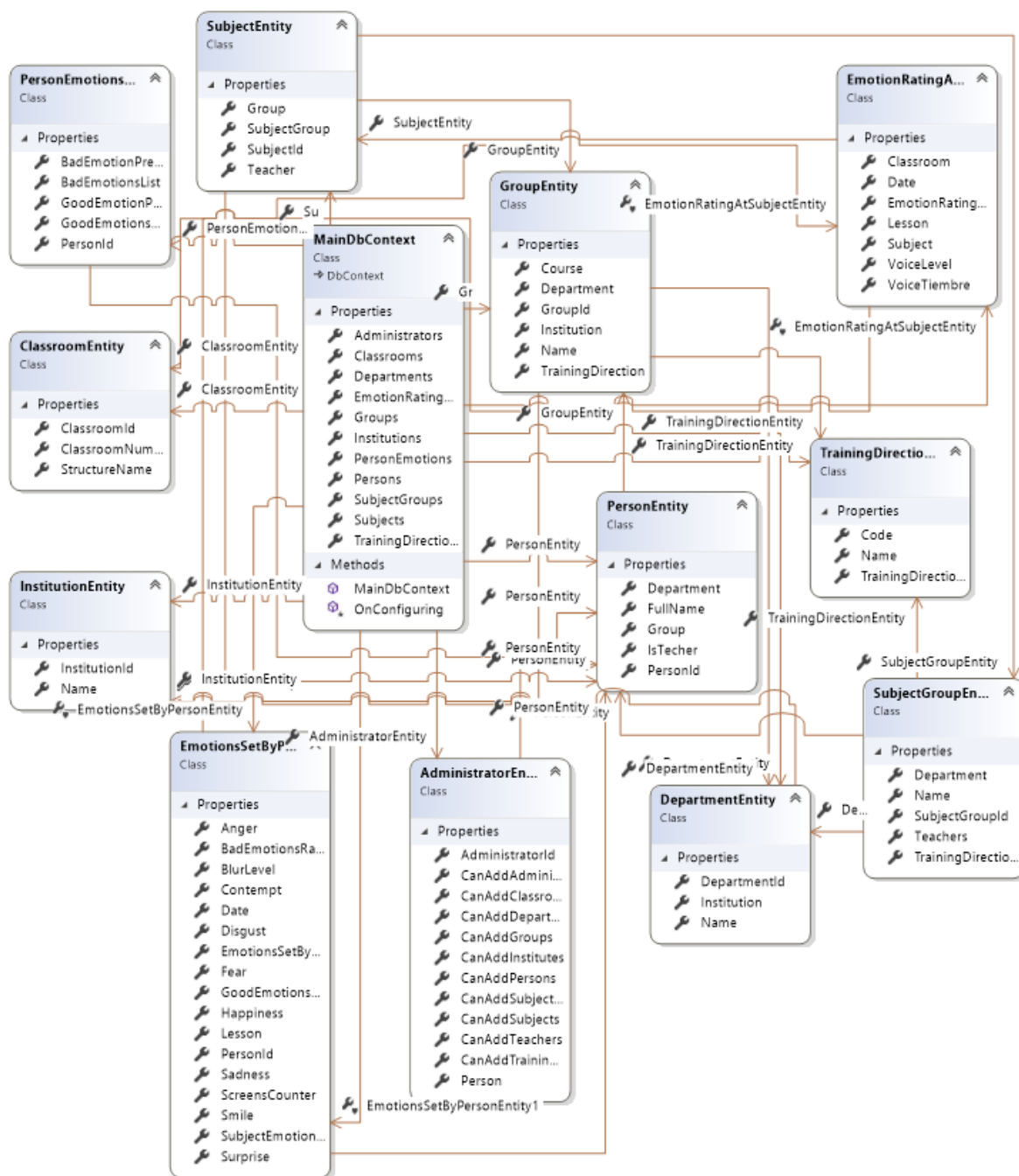


Рисунок 2.6 – Логическая модель базы данных на MSSQL

### 2.3 Прогнозирование итоговых показателей

Используя полученные путем обработки разработанным алгоритмом данных и начальные данные можно спрогнозировать грядущие показатели

уровня вовлеченности по каждой персоне [11]. К счастью, для используемого языка программирования уже есть готовые фреймворк ML.NET для работы с алгоритмами машинного обучения, что значительно сокращает время на разработку и построение архитектуры [13]. Неудивительно, что среди уже реализованных алгоритмов нет тех, которые разработаны под нашу задачу, однако есть реализованный алгоритм логистической регрессии, и вот почему это должно быть интересным: алгоритм логистической регрессии используют большие компании для прогнозирования продаж того или иного продукта на основе уже имеющихся данных о продажах, а также для прогнозирования цен на продукты с плавающей ценой на основе важных для расчета данных (например, дальность поездки и уровень движения транспортного потока на маршруте для цены на такси). Данный алгоритм опирается на логистическую функцию при расчете решения, результаты работы чего отображены на рисунке 2.7:

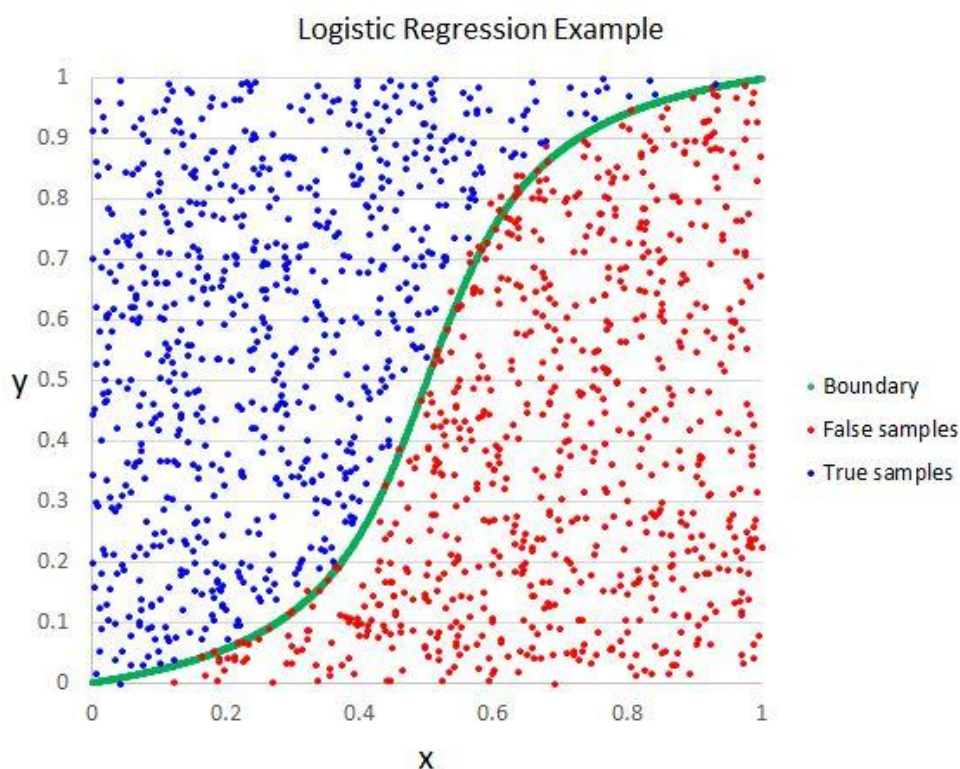


Рисунок 2.7 – Итог работы алгоритма логистической регрессии



Делается предположение, что вероятность события  $y = 1$  равна значению, описанному в формуле (2.1).

$$p\{y = 1 | x\} = f(z), \quad (2.1)$$

где  $z = \theta^T x = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$ ,

где  $\theta_i$  – коэффициент регрессии,

$f(z)$  – логистическая функция, значение которой вычисляется по формуле (2.2).

$$f(z) = \frac{1}{1 + e^{-z}}. \quad (2.2)$$

И, наоборот, при вероятности события  $y = 0$  уравнение принимает вид формулы (2.3).

$$p\{y = 0 | x\} = 1 - f(z) = 1 - f(\theta^T x), \quad (2.3)$$

Из последней формулы можем получить функцию распределения  $y$  при заданном  $x$ , что представлено в формуле (2.4).

$$p\{y | x\} = f(\theta^T x)^y (1 - f(\theta^T x))^{1-y}, y \in 0, 1. \quad (2.4)$$

Вызов метода алгоритма логистической регрессии с историей изменений всех эмоций субъекта за определенный промежуток времени и получение результата пишется в коде довольно легко, например так, как изображено на рисунке 2.8, однако время обработки запроса оставляет желать лучшего.

Если проанализировать получившиеся после работы алгоритма логистической регрессии результаты, то можно заметить некоторые незначительные погрешности в местах близости значений из разных групп, тем не менее в абсолютном большинстве алгоритм отработал точно, а это означает, что данный алгоритм можно использовать в итоговой версии разрабатываемого программного обеспечения для добавления функционала прогнозирования данных.

```

/// <summary> Прогнозирование данных.
private float Prediction(IEnumerable<EmotionsSetByPersonEntity> list, bool bad)
{
    var mlContext = new MLContext();
    var key = bad
        ? "BadEmotionsRating"
        : "GoodEmotionsRating";

    var data = mlContext.Data.LoadFromEnumerable(list);
    var pipeline = mlContext.Transforms.Conversion
        .MapValueToKey(key)
        .Append(mlContext.Transforms.Concatenate(
            "Features",
            "Anger",
            "Contempt",
            "Disgust",
            "Fear",
            "Happiness",
            "Surprise",
            "Sadness",
            "Smile"))
        .AppendCacheCheckpoint(mlContext)
        .Append(mlContext
            .MulticlassClassification
            .Trainers
            .LogisticRegression(labelColumnName: key, featureColumnName: "Features"))
        .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

    var model = pipeline.Fit(data);
    var prediction = mlContext
        .Model
        .CreatePredictionEngine<EmotionsSetByPersonEntity, PredicationResult>(model)
        .Predict(list.Last());

    return prediction.Result;
}

```

Рисунок 2.8 – Код вызова алгоритма прогнозирования данных ML.NET

В результатах проведенной описываемой в данной главе работы было реализовано программное обеспечение, которое получает данные от внешней системы об эмоциональном состоянии персон и на их основе вычисляет уровень заинтересованности в учебном процессе, прогнозирует данные по персоне, сохраняет полученные результаты в базу данных и при запросе от другой внешней системы передает на отображение полученные ранее данные.

## 3 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО АЛГОРИТМА

### 3.1 Описание тестируемой системы

В результате данной бакалаврской работы было реализовано программное обеспечение, которое принимает от внешней системы данные по персонам и их эмоциональному состоянию, обрабатывает последние по разработанному алгоритму оценки уровня заинтересованности студентов учебным процессом, и при соответствующем запросе от внешней системы отображения данных передает их этой системе. Реализованное ПО, пример работы которого изображен на рисунке 3.1, содержит взаимодействие с внешними системами (комментарии в коде и правила оформления кода по переносу символов и разделение строками участков кода разного назначения здесь и далее решено опустить, инициализация настроек скрыта для сохранения приватности данных) (Приложение А), аварийную обработку изображений, для которых не были идентифицированы эмоциональные состояния персон (Приложение Б), а также все виды обработки полученных данных и входящих запросов на предоставление данных для отображения, показанное в листинге 1. Модели данных, на основе которых работает система, приведены во второй главе данной работы.

Листинг 1. Псевдокод обработки входящих запросов от внешних систем.

```
class MainDataProcessing{
    private readonly List<FullAcceptanceData> _resultDataList;
    private readonly ImageResolver _imageResolver;
    public MainDataProcessing() {
        _resultDataList = new List<FullAcceptanceData>();
        _imageResolver = new ImageResolver();
        var random = new Random(); }
    public async Task AddDataToProcessing(List<IAcceptanceData> dataList) {
        Parallel.ForEach(dataList, async data => {
            if (data is ImageAcceptanceData image) {
                var imageData = await _imageResolver.TryUseDetecting(image);
```

```

    if (imageData != null) _resultDataList.Add(imageData); }
else if (data is FullAcceptanceData fullData) {
    _resultDataList.Add(fullData); } });
await Task.WhenAll();
await Task.Run(SaveData); }
private void SaveData() {
using (var db = new MainDbContext()) {
Parallel.ForEach(_resultDataList, data => {
var classroom = db.Classrooms.FirstOrDefault(x => x.ClassroomId == data.ClassroomId);
var subjectEmotions = new EmotionRatingAtSubjectEntity {
    Date = data.Date,
    Classroom = classroom,
    Lesson = data.Lesson,
    VoiceLevel = data.VoiceLevel,
    VoiceTiembre = data.VoiceTiembre };
db.EmotionRatingAtSubjects.Add(subjectEmotions);
db.SaveChanges();
Parallel.ForEach(data.Emotions, item => {
var previousEmotions = db.PersonEmotions.Where(x => x.PersonId ==
item.PersonId).OrderByDescending(x => x.Date.TimeOfDay).Take(10);
var badAnger = item.Emotions.Anger * (previousEmotions.Max(x => x.Anger) -
previousEmotions.Average(x => x.Anger));
var goodAnger = item.Emotions.Anger * (previousEmotions.Average(x => x.Anger) -
previousEmotions.Min(x => x.Anger));
var badContempt = item.Emotions.Contempt * (previousEmotions.Max(x => x.Contempt) -
previousEmotions.Average(x => x.Contempt));
var goodContempt = item.Emotions.Contempt * (previousEmotions.Average(x => x.Contempt) -
previousEmotions.Min(x => x.Contempt));
var badDisgust = item.Emotions.Disgust * (previousEmotions.Max(x => x.Disgust) -
previousEmotions.Average(x => x.Disgust));
var goodDisgust = item.Emotions.Disgust * (previousEmotions.Average(x => x.Disgust) -
previousEmotions.Min(x => x.Disgust));

```

```

var badFear = item.Emotions.Fear * (previousEmotions.Max(x => x.Fear) -
previousEmotions.Average(x => x.Fear));
var goodFear = item.Emotions.Fear * (previousEmotions.Average(x => x.Fear) -
previousEmotions.Min(x => x.Fear));
var badHappiness = item.Emotions.Happiness * (previousEmotions.Max(x => x.Happiness) -
previousEmotions.Average(x => x.Happiness));
var goodHappiness = item.Emotions.Happiness * (previousEmotions.Average(x => x.Happiness)
- previousEmotions.Min(x => x.Happiness));
var badSadness = item.Emotions.Sadness * (previousEmotions.Max(x => x.Sadness) -
previousEmotions.Average(x => x.Sadness));
var goodSadness = item.Emotions.Sadness * (previousEmotions.Average(x => x.Sadness) -
previousEmotions.Min(x => x.Sadness));
var badSurprise = item.Emotions.Surprise * (previousEmotions.Max(x => x.Surprise) -
previousEmotions.Average(x => x.Surprise));
var goodSurprise = item.Emotions.Surprise * (previousEmotions.Average(x => x.Surprise) -
previousEmotions.Min(x => x.Surprise));
var personSet = new EmotionsSetByPersonEntity{
Date = data.Date,
Lesson = data.Lesson,
PersonId = item.PersonId,
ScreensCounter = data.CurrentDataCounter,
GoodEmotionsRating = (float)(goodAnger + goodContempt + goodDisgust + goodFear +
goodHappiness + goodSurprise + goodSadness) * (1 - item.BlurLevel) * item.Smile,
BadEmotionsRating = (float)(badAnger + badContempt + badDisgust + badFear + badHappiness
+ badSurprise + badSadness) * (1 - item.BlurLevel) * (1 - item.Smile),
Anger = item.Emotions.Anger,
Contempt = item.Emotions.Contempt,
BlurLevel = item.BlurLevel,
Disgust = item.Emotions.Disgust,
Fear = item.Emotions.Fear,
Happiness = item.Emotions.Happiness,
Sadness = item.Emotions.Sadness,
Smile = item.Smile,

```

```

        Surprise = item.Emotions.Surprise,
        SubjectEmotionsId = db.EmotionRatingAtSubjects.First(x => x.Classroom.ClassroomId ==
data.ClassroomId && x.Date == data.Date && x.Lesson == data.Lesson).EmotionRatingAtSubjectId
};
        db.PersonEmotions.Add(personSet); }); });
        db.SaveChanges();}}
public List<ResultData> TypeRequest(IncomingMessage message, bool isNeedPrediction = true) {
    var result = new List<ResultData>();
    Parallel.ForEach(message.IdList, id => {
        var request = SingleTypeRequest(id, message.From, message.Till, message.MessageType,
isNeedPrediction);
        if (request != null) result.Add(request); });
    return result; }
private ResultData SingleTypeRequest(long id, DateTime from, DateTime till,
IncomingMessageType type, bool isNeedPrediction) {
    if (id == 0) return null;
    var result = new ResultData();
    var entities = new List<ResultEntity>();
    using (var db = new MainDbContext()) {
        var query = db.EmotionRatingAtSubjects.Where(x => x.Date >= from && x.Date <= till);
        if (query.Count() == 0) return null;
        switch (type) {
            case IncomingMessageType.Department: {
                query = db.EmotionRatingAtSubjects.Where(x =>
x.Subject.SubjectGroup.Department.DepartmentId == id);
                result.EntityName = db.Departments.FirstOrDefault(x => id == x.DepartmentId).Name; break; }
            case IncomingMessageType.Group: {
                query = db.EmotionRatingAtSubjects.Where(x => x.Subject.Group.GroupId == id);
                result.EntityName = db.Groups.FirstOrDefault(x => id == x.GroupId).Name; break; }
            case IncomingMessageType.Institution: {
                query = db.EmotionRatingAtSubjects.Where(x => x.Subject.Group.Institution.InstitutionId ==
id);
                result.EntityName = db.Institutions.FirstOrDefault(x => id == x.InstitutionId).Name; break; }

```

```

case IncomingMessageType.SubjectGroup: {
    query = db.EmotionRatingAtSubjects.Where(x => x.Subject.SubjectGroup.SubjectGroupId ==
id);
    result.EntityName = db.SubjectGroups.FirstOrDefault(x => id == x.SubjectGroupId).Name;
break; }

case IncomingMessageType.Teacher: {
    query = db.EmotionRatingAtSubjects.Where(x => x.Subject.Teacher.PersonId == id);
    result.EntityName = db.Persons.FirstOrDefault(x => id == x.PersonId && x.IsTeacher).FullName;
break; }

case IncomingMessageType.TrainingDirection: {
    query = db.EmotionRatingAtSubjects.Where(x =>
x.Subject.Group.TrainingDirection.TrainingDirectionId == id);
    result.EntityName = db.TrainingDirections.FirstOrDefault(x => id ==
x.TrainingDirectionId).Name; break; } }

var emotions = db.PersonEmotions.Where(x => query.Any(s => x.SubjectEmotionsId ==
s.EmotionRatingAtSubjectId));

Parallel.ForEach(query.OrderBy(x => x.Date).ThenBy(x => x.Lesson).ToList(), x => {
    var resultPersonEmotionsList = new List<PersonEmotionsEntity>();
    var personList = db.PersonEmotions.Where(p => p.SubjectEmotionsId ==
x.EmotionRatingAtSubjectId).GroupBy(p => p.PersonId).ToList();
    Parallel.ForEach(personList, p => {
        float? badPrediction = null;
        float? goodPrediction = null;
        if (isNeedPrediction) {
            badPrediction = Prediction(p, true);
            goodPrediction = Prediction(p, false); }
        resultPersonEmotionsList.Add(new PersonEmotionsEntity {
            PersonId = p.Key,
            BadEmotionsList = p.Select(l => l.BadEmotionsRating).ToList(),
            GoodEmotionsList = p.Select(l => l.GoodEmotionsRating).ToList(),
            BadEmotionPrediction = badPrediction,
            GoodEmotionPrediction = goodPrediction});});
    entities.Add(new ResultEntity {

```

```

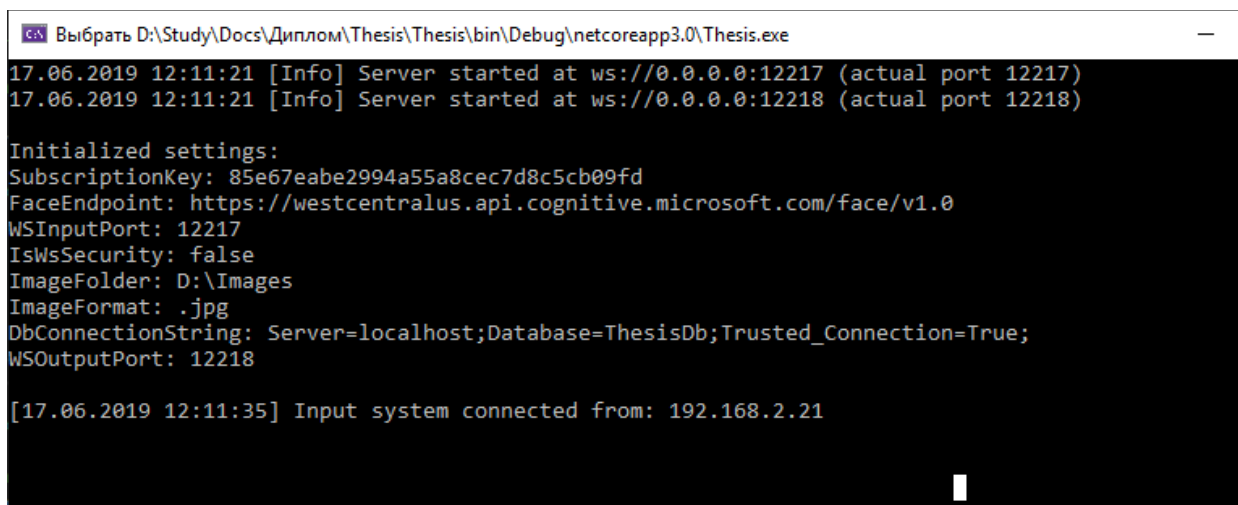
Date = x.Date,
Lesson = x.Lesson,
PersonEmotions = resultPersonEmotionsList,
VoiceLevel = x.VoiceLevel,
VoiceTiembre = x.VoiceTiembre}}});
result.Results = entities; } return result; }
public DictionariesList GetDictionaries() {
using (var db = new MainDbContext()) {
var result = new DictionariesList {
Administrators = db.Administrators.ToList(),
Classrooms = db.Classrooms.AsParallel().ToList(),
Departments = db.Departments.ToList(),
Groups = db.Groups.AsParallel().ToList(),
Institutions = db.Institutions.ToList(),
SubjectGroups = db.SubjectGroups.ToList(),
Subjects = db.Subjects.AsParallel().ToList(),
TrainingDirections = db.TrainingDirections.ToList()};return result; }}
private float Prediction(IEnumerable<EmotionsSetByPersonEntity> list, bool bad){
var mlContext = new MLContext();
var key = bad ? "BadEmotionsRating" : "GoodEmotionsRating";
var data = mlContext.Data.LoadFromEnumerable(list);
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(key)
.Append(mlContext.Transforms.Concatenate("Features", "Anger", "Contempt", "Disgust", "Fear",
"Happiness", "Surprise", "Sadness", "Smile"))
.AppendCacheCheckpoint(mlContext)
.Append(mlContext.MulticlassClassification.Trainers.LogisticRegression(labelColumnName: key,
featureColumnName: "Features"))
.Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
var model = pipeline.Fit(data);
var prediction = mlContext.Model.CreatePredictionEngine<EmotionsSetByPersonEntity,
PredicationResult>(model).Predict(list.Last());
return prediction.Result; }}
internal class PredicationResult{

```



```
[ColumnName("PredictedLabel")]
```

```
internal float Result;}
```



```
Выбрать D:\Study\Docs\Диплом\Thesis\Thesis\bin\Debug\netcoreapp3.0\Thesis.exe
17.06.2019 12:11:21 [Info] Server started at ws://0.0.0.0:12217 (actual port 12217)
17.06.2019 12:11:21 [Info] Server started at ws://0.0.0.0:12218 (actual port 12218)

Initialized settings:
SubscriptionKey: 85e67eabe2994a55a8cec7d8c5cb09fd
FaceEndpoint: https://westcentralus.api.cognitive.microsoft.com/face/v1.0
WSInputPort: 12217
IsWsSecurity: false
ImageFolder: D:\Images
ImageFormat: .jpg
DbConnectionString: Server=localhost;Database=ThesisDb;Trusted_Connection=True;
WSOutputPort: 12218

[17.06.2019 12:11:35] Input system connected from: 192.168.2.21
```

Рисунок 3.1 – Пример работы реализованной программы

Реализованное ПО, процесс разработки которого описан во второй главе данной работы, обладает всем заявленным функционалом, в том числе полностью автоматизирует процесс оценки уровня заинтересованности студентов учебным процессом.

### 3.2 Проведение теста «Cohn-Kanade»

Для тестирования определения степени вовлеченности персон в какое-либо действие существует проект «Cohn-Kanade». Данный проект представляет собой готовый набор данных из нескольких тысяч фотографий, для каждой из которых совместными усилиями разработчиков и психологов были добавлены показатели степени вовлеченности по каждой из персон. Данный проект предназначен для обучения нейронных сетей, однако не предоставляет возможности получения спектра эмоций, испытываемых каждой из персон до момента создания снимка.

На данном наборе фотографий был проведен тест разработанной системы, процесс реализации которой описан во второй главе данной работы, исходный код которой можно загрузить через сайт Github по адресу

<https://github.com/Corel-frim/Thesis>, а также диаграмма классов которой изображена на рисунке 3.2, и результат вычислений превзошел возложенные на данную систему ожидания: помимо достаточно высокой скорости обработки фотографий точность подсчета была выше среднего результата всех систем, когда-либо участвовавших на момент проведения тестирования в указанном выше проекте, на 1.49%, все полученные результаты тестирования приведены в таблице 3.1.

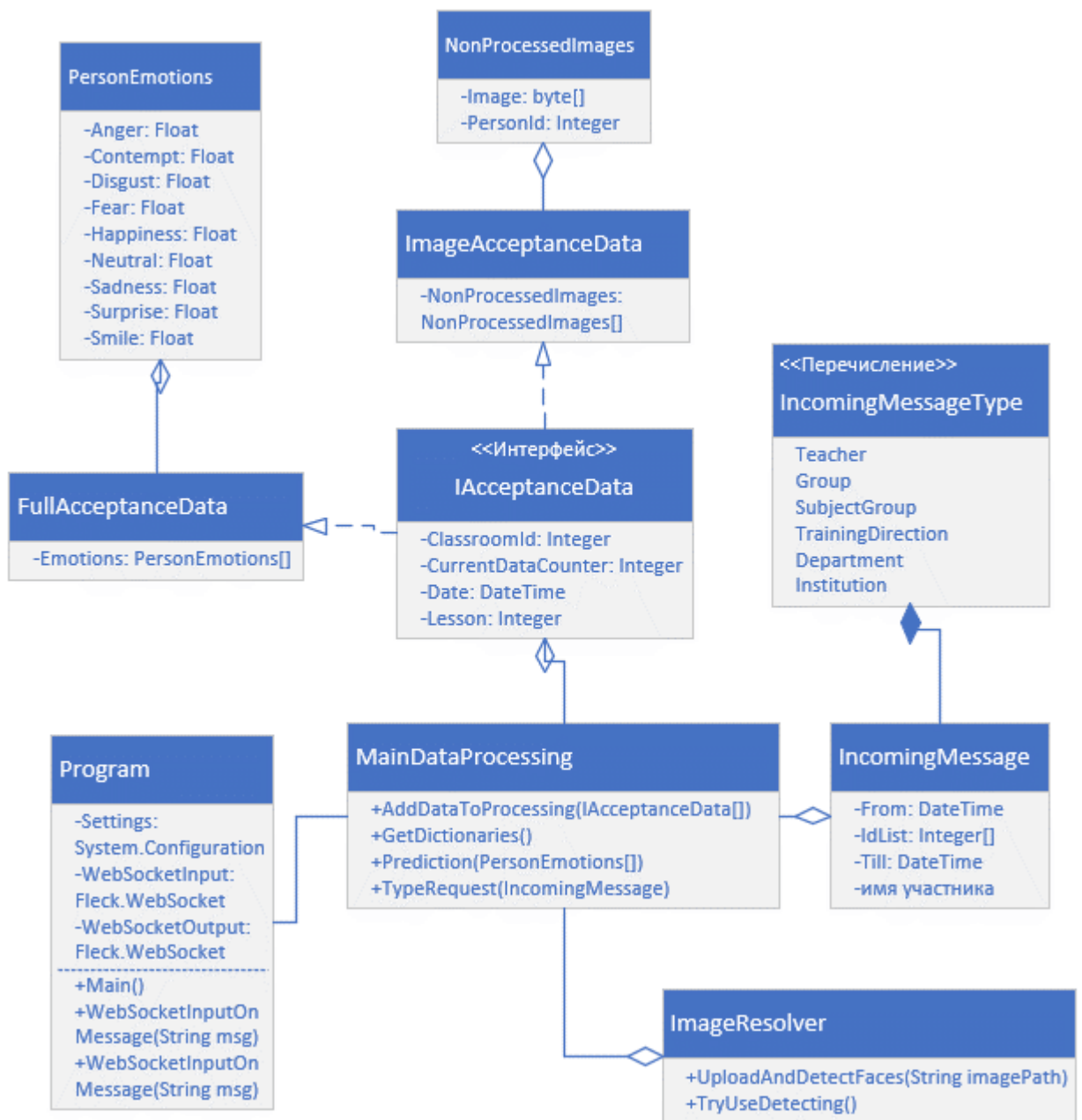


Рисунок 3.2 – Диаграмма классов реализованной системы

Таблица 3.1 – Результаты обработки данных проекта «Cohn-Kanade»

	Показатель разработанной в рамках текущей работы системы	Средний показатель за все время сторонними разработчиками
Общее количество фотографий	35692 ед.	35692 ед.
Среднее время обработки одного элемента в миллисекундах	6,0856865 мс	12,6030298 мс
Количество обработанных элементов в секунду	164,32 ед.	79,346 ед.
Общее время на обработку всех фото в секундах	217,21032 с	449,827338 с
Точность оценки разработанной системы	68,31%	66,82%

Учитывая тот факт, что данный проект не предоставляет никаких исторических данных по эмоциональному состоянию каждой персоны до снимка, можно предположить, что точность разработанной системы будет выше в среде, под которую она изначально разрабатывалась, но при этом будет задействовать больше системных ресурсов, что следует учитывать при выделении вычислительных мощностей под данное решение. Так же немного больше половины времени обработки одного элемента уходило на запись новых данных в базу данных, для решения данной проблемы может помочь только реорганизация части архитектуры, связанной с взаимодействием с базой данных.

### **3.3 Демонстрация работы реализованного программного обеспечения**

Для демонстрации результатов работы программы, приведенных на рисунке 3.3, были отобраны 10 фотографий одной персоны с разными испытываемыми эмоциями, порядок фотографий случайный (Приложение В, фотографии размещены в порядке слева направо и сверху вниз).

Также был протестирован функционал прогнозирования данных с учетом их естественного изменения с учетом времени, результаты представлены на рисунке 3.4. В качестве начальных данных взяты постепенно изменяющиеся эмоции человека (Приложение Г, фотографии размещены в порядке слева

направо и сверху вниз), однако прогнозирование составлено сразу на 4 последующих временных промежутка, при том что в основной функционал заложено прогнозирование только на 1 временной промежуток.

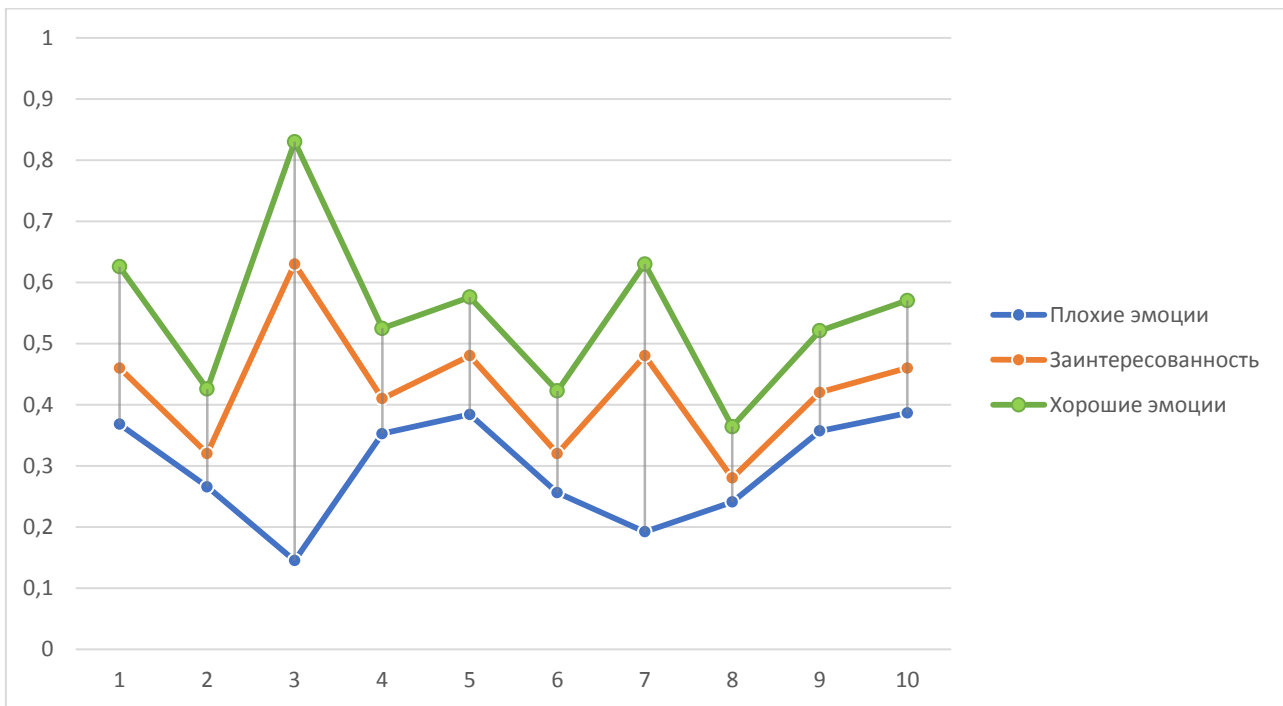


Рисунок 3.3 – Результат работы алгоритма на 10 снимках

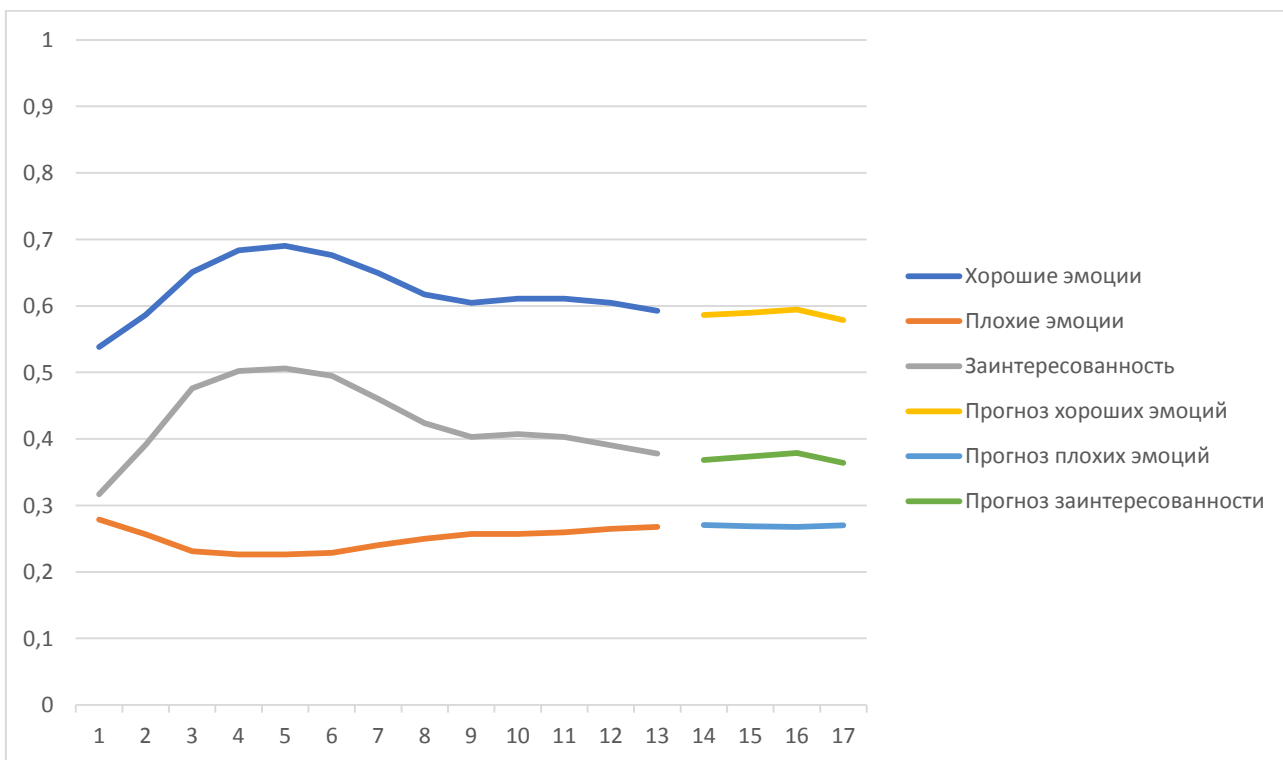


Рисунок 3.4 – Результат работы с прогнозированием

Как можно заметить по результатам, по разработанному алгоритму реализованным программным обеспечением были высчитаны сразу 3 показателя: «плохие» эмоции, «хорошие» эмоции и итоговая заинтересованность, при том, чем выше первый показатель и ниже последний, тем ближе итоговый результат к показателю «хороших» эмоций. Результаты прогнозирования проверить на достоверность, к сожалению, не удастся.

Опираясь на результаты теста «Сohn-Kanade» можно с точностью сказать, что реализованное программное обеспечение конкурентноспособно для нейронных сетей, обучаемых для определения степени заинтересованности персоны, и может использоваться по назначению.

## ЗАКЛЮЧЕНИЕ

Результаты работы, как и было сказано, превзошли возложенные ожидания, однако скорость работы с течением времени будет все меньше, поскольку изначально предполагалось использование предыдущих показателей из базы данных, что нельзя было использовать на подобном тестировании. В свою очередь, точность должна расти по мере использования исторических данных.

Выведенную формулу, опираясь на полученные ранее результаты теста, можно считать оптимальной, что полностью подтверждается показателем точности оценки оценки, однако можно попробовать принять участие с измененными множителями для каждой эмоции: возможно, с новыми данными результат будет еще выше.

Разработанное программное обеспечение также можно считать успешным, поскольку при тестировании не было никаких ошибок, притормаживаний и тому подобное, что могло бы привести процесс работы с приложением к непригодности. Также необходимо отметить результаты работы прогнозирования: их нельзя проверить естественным путем, а также прогнозирование строится без учета нелинейности происходящих во время учебного занятия событий, из чего в перспективе можно было бы добавить новый функционал системе по рекомендациям действий, которые могли бы привести к увеличению итогового показателя заинтересованности в текущем учебном занятии.

Результат данной работы при внедрении в учебное учреждение позволит повысить при должном анализе результатов работы разработанного программного обеспечения вовлеченность студентов в учебный процесс, что положительно скажется на эмоциональном состоянии преподавателей, что приведет к их ускоренному росту в профессиональном плане, что приведет к повышению конкурентоспособности ВУЗа среди других и так далее. Также точности анализа поспособствует добавление помимо итогового показателя показателей по положительным и отрицательным эмоциям, из которых и

составляется оценка уровня заинтересованности. Также при взаимодействии с системой визуального отображения передаются параметры уровня громкости речи и интонация, однако в описываемом в данной работе программном обеспечении данные показатели никак не обрабатываются, а только сохраняются для дальнейшего отображения, тем не менее при необходимости и должном анализе эти данные также могут помочь в определении точек улучшения собственного или чужого поведения для улучшения результатов по итоговому параметру.

Кроме предусмотренных заданием на бакалаврскую работу данную работу можно применять в уже имеющемся виде для определения степен вовлеченности в рабочие процессы сотрудников без физически тяжелого труда любого из предприятий, данное ограничение обусловлено необходимостью оптимизации алгоритма под эмоции, произвольно проявляемые во время использования физического труда.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Шумаков П. ADO.NET и создание приложений баз данных в среде Microsoft Visual Studio .NET (2-е издание) / Санкт-Петербург Изд-во Диалог-МИФИ, 2014. - 512 с.
2. Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы .NET и .NET Core. / Санкт-Петербург Изд-во Диалог-МИФИ, 2018. - 1328 с.
3. Gerardus Blokdyk. Aerospike database A Complete Guide. USA: 5STARCOoks, 2018.
4. Ritesh Modi. Azure for Architects: Implementing cloud design, DevOps, IoT, and serverless solutions on your public cloud. USA: Packt Publishing, 2017.
5. Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems. USA: Manning Publications, 2015.
6. Arshdeep Bahga. Big Data Science & Analytics: A Hands-On Approach. USA: VPT, 2016.
7. Ritesh Modi. Deployment of Microsoft Azure Cloud Solutions: A complete guide to cloud development using Microsoft Azure. USA: Packt Publishing, 2018.
8. Jon P Smith. Entity Framework Core in Action. USA: Manning Publications, 2018.
9. P. Ekman and W. Friesen. Facial Action Coding System: A Technique for the Measurement of Facial Movement. USA: Consulting Psychologists Press, 1978.
10. Tom White. Hadoop: The Definitive Guide. USA: O'Reilly Media, 2015.
11. Suresh Kumar Mukhiya, Tao Wei, James Lee. Hands-On Big Data Modeling. USA: Packt Publishing, 2018.
12. Leif Larsen. Learning Microsoft Cognitive Services. USA: Packt Publishing, 2017.
13. Dixon J. Mastering .NET Machine Learning. USA: Packt Publishing, 2016. ISBN 978-1785888403.



14. Copeland M., Soh J., Puca A., Manning M., Gollob D. Microsoft Azure: Planning, Deploying, and Managing Your Data Center in the Cloud. USA: Apress, 2015.
15. Agafonov E. Multithreading with C# Cookbook - Second Edition. USA: Packt Publishing, 2016. - 592 c., ISBN 9781785881251.
16. Carter P. Pro SQL Server Administration. USA: Apress, 2015. - 525 c. ISBN 978-1-4842-0711-6.
17. Dino Esposito. Programming ASP.NET Core. USA: Microsoft Press, 2018.
18. Lerman Julia, Miller Rowan. Programming Entity Framework: Code First. USA: O'Reilly Media, 2012.
19. Hillar G. C. Professional Parallel Programming with C#: Master Parallel Extensions With .NET 4.7. USA: Wrox, 2018. - 702 c. ISBN 9780470495995.
20. Kahn, William A. Psychological Conditions of Personal Engagement and Disengagement at Work. USA: Wrox Press, 1990.
21. Ekman P. Emotions Revealed: Recognizing Faces and Feelings to Improve Communication and Emotional Life. USA: Wrox Press, 2008.
22. Grant Fritchey. SQL Server Query Performance Tuning. USA: Apress, 2014.
23. Itzik Ben-Gan. T-SQL Fundamentals (3rd Edition). USA: Microsoft Press, 2016.
24. Andrew Lombardi. WebSocket: Lightweight Client-Server Communications. USA: O'Reilly Media, 2018.
25. Dustin Metzgar. .NET Core in Action, USA: Manning Publications, 2018.

## ПРИЛОЖЕНИЕ А

### Псевдокод реализации взаимодействия с внешними системами

```
public class Program {
    public static IConfiguration Settings;
    static void Main(string[] args) {
        Settings = new ConfigurationBuilder()
            .AddInMemoryCollection(new Dictionary<string, string>
                {*/Настройки скрыты в рамках приватности/*}).Build();
        var mainDataProcessing = new MainDataProcessing();
        var wsType = bool.Parse(Settings["IsWsSecurity"]) ? "wss"
: "ws";
        var websocketSender = new
WebSocketServer($"{wsType}://0.0.0.0:{Settings["WSOutputPort"]}");
        var websocketApplier = new
WebSocketServer($"{wsType}://0.0.0.0:{Settings["WSInputPort"]}");
        websocketApplier.Start(socket =>{
            socket.OnOpen = () =>
                Console.WriteLine($"New connection to applier from
{socket.ConnectionInfo.ClientIpAddress}");
            socket.OnClose = () =>
                Console.WriteLine($"Removed connection to applier
from {socket.ConnectionInfo.ClientIpAddress}");
            socket.OnError = e => {
                Console.WriteLine($"Error in connection to applier
with {socket.ConnectionInfo.ClientIpAddress}: {e.Message}");};
            socket.OnMessage = msg => {
                var message =
msg.FromJson<List<IAcceptanceData>>();
                if (message == null) return;
                mainDataProcessing.AddDataToProcessing(message).GetAwaiter().GetRe
sult();};});
        websocketSender.Start(socket => {
            socket.OnOpen = () => {
```

```

        Console.WriteLine($"New connection to sender from
{socket.ConnectionInfo.ClientIpAddress}");
        var dictionaries =
mainDataProcessing.GetDictionaries().ToJson();
        socket.Send(dictionaries);
    };
    socket.OnClose = () =>
        Console.WriteLine($"Removed connection to sender
from {socket.ConnectionInfo.ClientIpAddress}");
    socket.OnError = e =>
        {
            Console.WriteLine($"Error in connection to sender
with {socket.ConnectionInfo.ClientIpAddress}: {e.Message}");
        };
    socket.OnMessage = msg =>
        {
            var message = msg.FromJson<IncomingMessage>();
            if (message == null || message.IdList.Count == 0)
return;

            var result =
mainDataProcessing.TypeRequest(message);
            socket.Send(result.ToJson());
            var checkFrom = DateTime.Now;
            while (socket.IsAvailable && DateTime.Now <
message.Till) {
                message.From = checkFrom;
                Task.Delay(60000);
                checkFrom = DateTime.Now;
                result =
mainDataProcessing.TypeRequest(message, false);
                socket.Send(result.ToJson());}}}}}}}}

```

## ПРИЛОЖЕНИЕ Б

### Псевдокод аварийной обработки изображений

```
class ImageResolver {
    private IFaceClient faceClient { get; }
    private List<PersonEmotions> personEmotions { get; }
    internal MainDataProcessing MainDataProcessing { get; set; }
    public ImageResolver() {
        var subscriptionKey = Program.Settings["SubscriptionKey"];
        var faceEndpoint = Program.Settings["FaceEndpoint"];
        faceClient = new FaceClient(
            new ApiKeyServiceClientCredentials(subscriptionKey),
            new System.Net.Http.DelegatingHandler[] { });
        faceClient.Endpoint = faceEndpoint;
        personEmotions = new List<PersonEmotions>();
    }
    private async Task<IList<DetectedFace>>
UploadAndDetectFaces(string imagePath) {
        IList<FaceAttributeType> faceAttributes = new
FaceAttributeType[] {
            FaceAttributeType.Smile,
            FaceAttributeType.Emotion,
            FaceAttributeType.Makeup,
            FaceAttributeType.Blur};
        try{ using (var imageFileStream =
File.OpenRead(imageFilePath)) {
            IList<DetectedFace> faceList =
                await faceClient.Face.DetectWithStreamAsync(
                    imageFileStream, true, false,
faceAttributes);
            return faceList;} }catch (Exception) {
            return new List<DetectedFace>();}
    }
    public async Task<FullAcceptanceData>
TryUseDetecting(ImageAcceptanceData data) {
        data.NonProcessedImages.ForEach(async x => {
            var path =
string.Concat(Program.Settings["ImageFolder"],
```

```

Path.AltDirectorySeparatorChar, x.PersonId[0],
Program.Settings["ImageFormat"]);
    var counter = 0;
    File.WriteAllBytes(path, x.Image);
    var faces = await UploadAndDetectFaces(path);
    foreach (var face in faces){
        personEmotions.Add(new PersonEmotions {
            BlurLevel =
(float)face.FaceAttributes.Blur.Value,
            Smile =
(float)face.FaceAttributes.Smile.Value,
            PersonId = x.PersonId[counter++],
            Makeup = new Models.Input.Makeup {
                EyeMakeup =
face.FaceAttributes.Makeup.EyeMakeup,
                LipMakeup =
face.FaceAttributes.Makeup.LipMakeup},
            Emotions = new EmotionAsset{
Anger = (float)face.FaceAttributes.Emotion.Anger,
Contempt = (float)face.FaceAttributes.Emotion.Contempt,
Disgust = (float)face.FaceAttributes.Emotion.Disgust,
Fear = (float)face.FaceAttributes.Emotion.Fear,
Happiness = (float)face.FaceAttributes.Emotion.Happiness,
Neutral = (float)face.FaceAttributes.Emotion.Neutral,
Sadness = (float)face.FaceAttributes.Emotion.Sadness,
Surprise = (float)face.FaceAttributes.Emotion.Surprise } }));});
    var result = new FullAcceptanceData{
        Date = data.Date,
        ClassroomId = data.ClassroomId,
        CurrentDataCounter = data.CurrentDataCounter,
        Lesson = data.Lesson,
        Emotions = personEmotions};
    await Task.WhenAll();
    return result;}}}

```

## ПРИЛОЖЕНИЕ В

Изображения из теста Kohn-Kanade для демонстрации работы



# ПРИЛОЖЕНИЕ Г

## Набор фотографий для теста прогнозирования

