МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ
В ГЭК И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
к.т.н.,доцент
_____М. С. Воробьева
_____2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(бакалаврская работа)

ИССЛЕДОВАНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ ПОИСКА МУЗЫКАЛЬНЫХ ПРОИЗВЕДЕНИЙ С ПОМОЩЬЮ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

02.03.03. Математическое обеспечение и администрирование информационных систем

Выполнила работу
Студентка 4 курса
очной формы обучения

Метелёва
Елена
Сергеевна

Руководитель работы
д.п.н., профессор

Захарова
Ирина
Гелиевна

г. Тюмень, 2019

Оглавление

Введение

Поиск музыкальных произведений является задачей, которую зачастую приходится решать музыкантам, учителям и ученикам музыкальных школ. Для выбора репертуара необходимо учитывать возможности музыканта, т.е. какое произведение он может сыграть, а также его предпочтения – какие произведения он играет, что вызывает у него трудности, а что дается с легкостью.

Поиск аналогов приложения, которое бы могло подбирать музыкальные произведения на основе нотной записи, а не музыкального воспроизведения, не дал результатов. В итоге, музыкантам для подбора репертуара приходится либо просматривать множество вариантов самостоятельно, либо обращаться к специалисту, который знаком с большим набором музыкальных произведений и обладает достаточными опытом и знаниями, чтобы предложить что-то подходящее (аналогично тому, как учитель в музыкальной школе составляет новую программу для каждого ученика, исходя из того, в каком он классе, уровню его подготовки, знаний и умений).

Исходя из вышесказанного, появилась идея написания приложения, которое бы могло решать данную проблему.

Была выдвинута гипотеза о том, что на основании нотной записи музыкант может определить, доступно ли ему для исполнения данное произведение или нет.

Целью выпускной квалификационной работы является разработка приложения для поиска музыкальных произведений, доступных для исполнения определенному пользователю, на основе анализа нотной записи.

Для достижения данной цели требуется выполнить следующие задачи:

- Определить источник базы данных музыкальных произведений.
- Изучить форматы представления музыкальных произведений в текстовом виде и методы извлечения признаков.

- Определить признаки, характеризующие музыкальное произведение с точки зрения сложности исполнения, и меру схожести музыкальных произведений.
- Разработать приложение для поиска музыкальных произведений с использованием методов машинного обучения.

Глава 1. Описание постановки задачи и методов ее решения

1.1. Форматы представления нотной записи

Музыкальная нотация (в широком смысле) — система всех обозначений (в том числе нотных, словесных, цифровых и т. д.), с помощью которой автор записывает свое музыкальное произведение.

На сегодняшний день широкое распространение получают коммерческие проекты, связанные с продажей нот через Интернет. Как правило, нотная запись предоставляется в формате электронных документов PDF. Однако данный формат в таких проектах применяется до тех пор, пока пользователей интересует графическое представление нот. Как только возникает необходимость последующей работы с нотами при помощи программного обеспечения, формат PDF уже не является подходящим для удовлетворения этой потребности. Более того, представление информации в графическом виде требует больших объемов памяти и, следовательно, большего времени передачи данных, нежели при её «осмысленном» представлении. Если передавать описание графического изображения, то оно будет занимать несколько килобайт. Представление одной ноты в формате MIDI занимает несколько байт, в то время как размер ее графического представления измеряется килобайтами.

В качестве представления нотной записи в компьютере крупнейшими разработчиками программного обеспечения за основу принят формат MusicXML.

MusicXML - это основанный на XML формат файла для представления музыкальной нотации, основная задача которого - корректное графическое отображение, то есть демонстрация того, как музыкальное произведение будет выглядеть. Также, данный формат содержит информацию, как произведение будет звучать. Например, он используется в качестве исходных данных для создания MIDI файла. MusicXML имеет нечто общее с MIDI. Но в отличие от MusicXML, формат MIDI разработан для воспроизведения

музыки и не отражают всех тонкостей музыкальной грамоты (нотного стана, орнаментики и др.).

Формат MusicXML позволяет получить точное описание исходного графического представления произведения (нотной записи). Файлы данного формата содержат большое количество информации, подробное описание как музыкальной, так и графической составляющей, в связи с чем имеют большой размер (см. Приложение 1). Данная проблема была решена созданием формата Compressed MusicXML - это сжатый файл MusicXML, который занимает меньше места (при помощи создания сжатого zip-файла с суффиксом .mxl, который уменьшает размер файла примерно на одну двадцатую от несжатой версии). Compressed MusicXML более новый формат, который не так широко используется на данный момент.

1.2. Постановка задачи

Исходные данные: набор из файлов формата MusicXML.

В формате MusicXML можно увидеть наличие следующих 18 признаков, характеризующих музыкальное произведение:

1. среднее число нот в такте
2. количество ключевых знаков
3. количество ключей
4. количество случайных знаков
5. среднее число аккордов на такт
6. объем
7. ритмическое разнообразие
8. пунктирный ритм
9. педаль
10. глиссандо
11. триоль
12. мордент
13. группетто

14.стаккато

15.трель

16.форшлаг

17.арпеджио

18.тремоло

Для определения уровня сложности произведения нужно использовать только часть из них.

Выдвинуто предположение: если пользователь отмечает определенные музыкальные произведения, исходя из нотной записи, то это позволяет выбрать из набора близкие по уровню сложности.

1.3. Алгоритм решения поставленной задачи

Этапы решения поставленной задачи:

1.3.1. Извлечение данных

В качестве источника данных необходимо было найти базу музыкальных произведений, содержащую данные в нужных форматах. Для решения данной задачи была выбрана база MuseScore. Данный ресурс позволяет скачивать данные в форматах MuseScore, MusicXML, PDF, PNG, MIDI и MP3. Для получения подробной информации обо всех общедоступных произведениях с ключами для скачивания MuseScore предоставляет API для разработчиков.

Всего в базе 2000 произведений.

В работе использовались только музыкальные произведения с одной партией для фортепиано. Итого это составило 1342 произведения.

1.3.2. Извлечение и анализ признаков

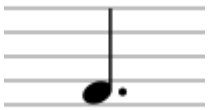
В качестве признаков были отобраны те, которые отображают академическую сложность произведения, описывают его объем, насыщенность элементами.

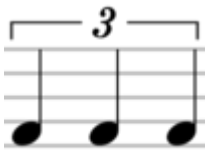
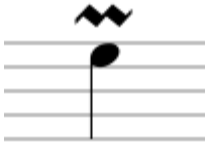
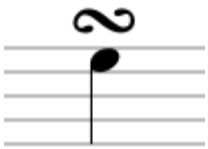
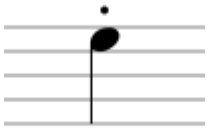
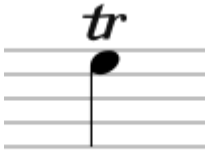
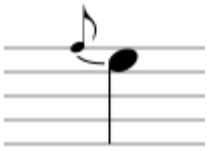
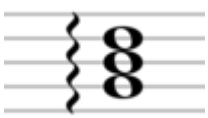
Каждому произведению ставится в соответствие вектор, каждая координата которого отвечает за значение соответствующего признака. Признаки приведены в таблице №1.

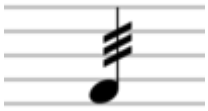
Таблица №1.

Название	**Графическое представление**	**Тип**	**Описание**
1. *Среднее число нот в такте*		Вещественный	Чем меньше нот в такте, тем проще произведение выглядит.
2. *Количество ключевых знаков*		Вещественный	Чем больше знаков – тем сложнее играть.
3. *Количество ключей (частота смены размеров/ ключей/ тональности)*		Вещественный	Как часто меняется число нот и их длительности на такт/расположение руки и нот на стане/появляются другие знаки в тональности – постоянная смена усложняет процесс исполнения в реальном времени.
4. *Количество случайных*		Вещественный	Случайные знаки применяются

знаков (знаки, стоящие при ключе, не учитываются)			только к одному такту, в котором они встречаются. Чем их больше, тем чаще нужно обращать внимание на то, что данную ноту нужно играть по-другому, но на время определенного такта.
5. *Среднее число аккордов в такте*		Вещественный	Аккорды задействуют несколько пальцев одновременно.
6. *Объем (число страниц)*		Вещественный	Произведение может быть простым, но большой объем может вызвать отторжение (слишком долго по времени) или исполнение такого произведения

			будет затруднительным (недостаточность игровой практики произведений большого размера - устают пальцы и запястья).
7. *Ритмическое разнообразие*		Вещественный	Сумма всех различных длительностей нот, встречающихся в произведении.
8. *Пунктирный ритм*		Бинарный	Если ритм неровный или присутствует частая смена длительностей – вызывает трудности при исполнении.
9. *Педаль*		Бинарный	

10.Глиссандо		Бинарный	
11.Триоль		Бинарный	
12.Мордент		Бинарный	
13.Группетто		Бинарный	
14.Стаккато		Бинарный	
15.Трель		Бинарный	
16.Форшлаг		Бинарный	
17.Арпеджио		Бинарный	

18.*Тремоло*		Бинарный	

Признаки с 10 по 18 отвечают за академическую сложность произведения. Данные понятия обычно вводятся постепенно, и не каждый музыкант обладает достаточным опытом, знаниями и умениями для их исполнения. Это именно те обозначения, которые могут быть неизвестны пользователю или вызывать трудности во время исполнения произведения.

Описание значений признаков приведено на рисунке 1.3.2.1. Заголовки столбцов соответствуют названиям признаков, а строки:

- Count – количество
- Mean – среднее значение
- Std – стандартное отклонение
- Min – минимальное значение
- Max – максимальное значение
- Квартили распределения — квантили, кратные 25%: 25%, 50% и 75%.

Результаты получены средствами библиотеки pandas (метод pandas.DataFrame.describe) языка программирования python.

	countNotesAvg	countKeys	countClefs	countAccidentals	countChords
count	1327.000000	1327.000000	1327.000000	1327.000000	1327.000000
mean	17.645084	1.975885	5.984928	88.267521	3.415008
std	21.052874	1.663019	13.382518	543.071209	3.609161
min	1.816327	0.000000	1.000000	0.000000	0.000000
25%	11.469444	1.000000	1.000000	1.000000	1.476872
50%	15.279070	2.000000	2.000000	17.000000	2.732759
75%	20.483036	3.000000	6.000000	57.500000	4.420046
max	524.335664	7.000000	198.000000	16390.000000	64.860140

	countPages	rhythmicDiversity	hasGlissando	hasTuplet	hasMordent
count	1327.000000	1327.000000	1327.000000	1327.000000	1327.000000
mean	4.507159	15.196684	0.027882	0.371515	0.030897
std	4.876428	1.543650	0.164698	0.483392	0.173103
min	1.000000	2.000000	0.000000	0.000000	0.000000
25%	2.000000	15.000000	0.000000	0.000000	0.000000
50%	4.000000	16.000000	0.000000	0.000000	0.000000
75%	5.000000	16.000000	0.000000	1.000000	0.000000
max	87.000000	16.000000	1.000000	1.000000	1.000000

	hasTurn	hasStaccato	hasTrill	hasDot	hasPedal
count	1327.000000	1327.000000	1327.000000	1327.000000	1327.000000
mean	0.005275	0.354936	0.037679	0.941974	0.333082
std	0.072465	0.478674	0.190490	0.233880	0.471493
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000	0.000000
50%	0.000000	0.000000	0.000000	1.000000	0.000000
75%	0.000000	1.000000	0.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	hasGrace	isArpeggiate	hasTremolo
count	1327.000000	1327.000000	1327.000000
mean	0.238131	0.331575	0.050490
std	0.426100	0.470957	0.219036
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000

Рис. 1.3.2.1. Характеристики значений признаков

Исходя из характеристик данных, представленных на рисунке 1.3.2.1, видно, что значения признака rhytmicDiversity изменяются незначительно и имеют одно и то же значение минимум для 50% выборки. Данный признак можно не учитывать в дальнейшей работе. Также можно видеть присутствие выбросов у признаков countNotesAvg, countClefs, countAccidentals, countChords и countPages – у данных признаков максимальное значение значительно отличается от среднего и от 75% выборки. Также при сравнении

среднего значения и значения, соответствующего 50% выборки, у признаков countAccidentals и countClefs видна существенная разница между этими значениями. Произведения с выбросами необходимо исключить из выборки.

После удаления выбросов осталось 1100 произведений.

Для бинарных признаков среднее значение свидетельствует о количестве элементов, у которых значение данного признака отлично от нуля. Присутствуют бинарные признаки, для которых число значений, отличное от нуля, очень мало, и они имеют очень малый процент по отношению ко всей выборке (см. рис.1.3.2.2) . Но есть и признаки, которыми обладают более 90% элементов в выборке (см. рис.1.3.2.3). Данные признаки также можно не учитывать.

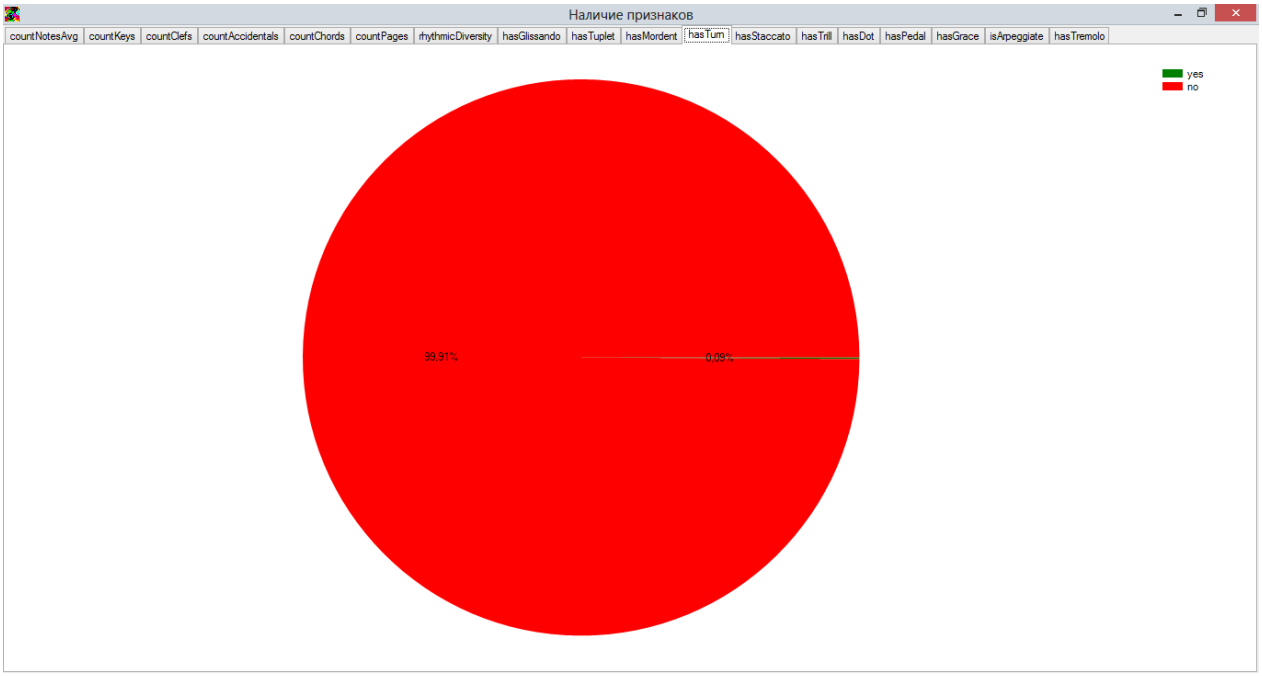


Рис. 1.3.2.2. Диаграмма распределения признака hasTurn в выборке

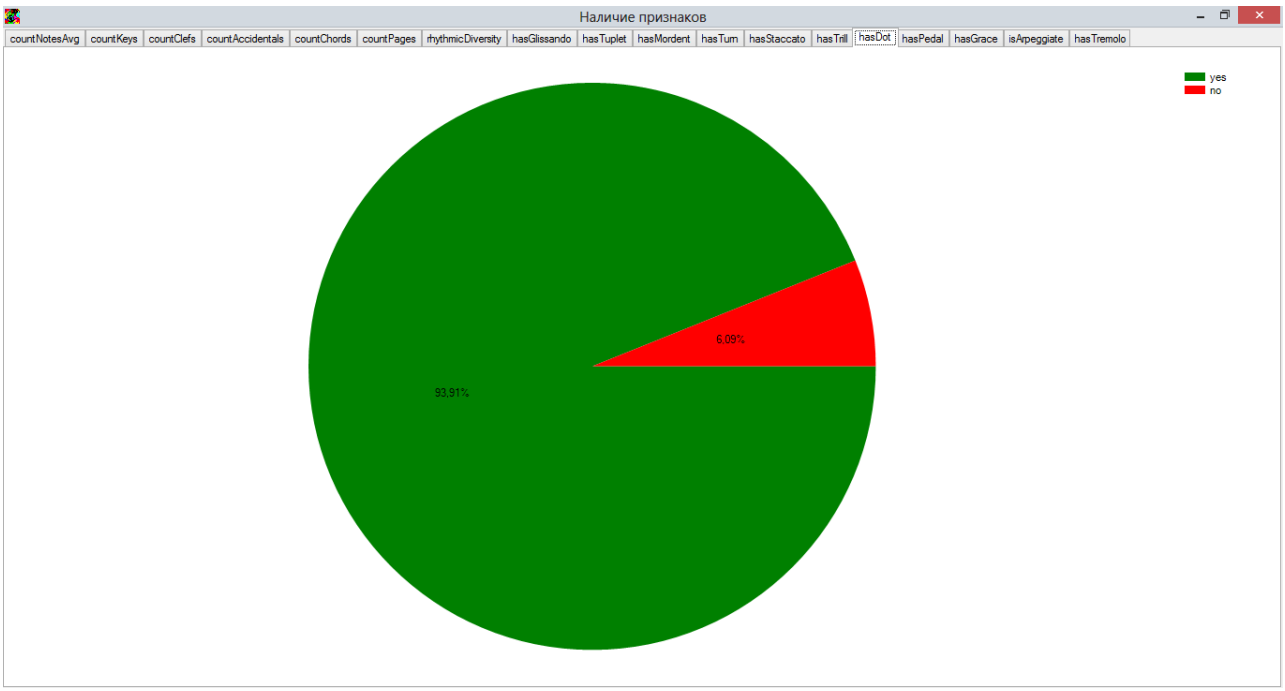


Рис. 1.3.2.3. Диаграмма распределения признака hasDot в выборке

В результате из признаков были исключены rhythmicDiversity, hasTurn и hasDot.

1.3.3. Подбор похожих произведений

Для подбора похожих произведений можно было бы использовать метод кластеризации. Но этого недостаточно: произведения необходимо подбирать под каждого пользователя отдельно. Необходимо определить, что вызывает трудности у пользователя, что для него сложно, что нет, и на основе этого уже производить поиск.

Для решения данной задачи был использован следующий алгоритм.

1. Предварительная кластеризация по всем признакам, приведенным к единой шкале (значения находятся в интервале [0,1]). Ожидаются различия по сложности (см. рис.1.3.3.1)

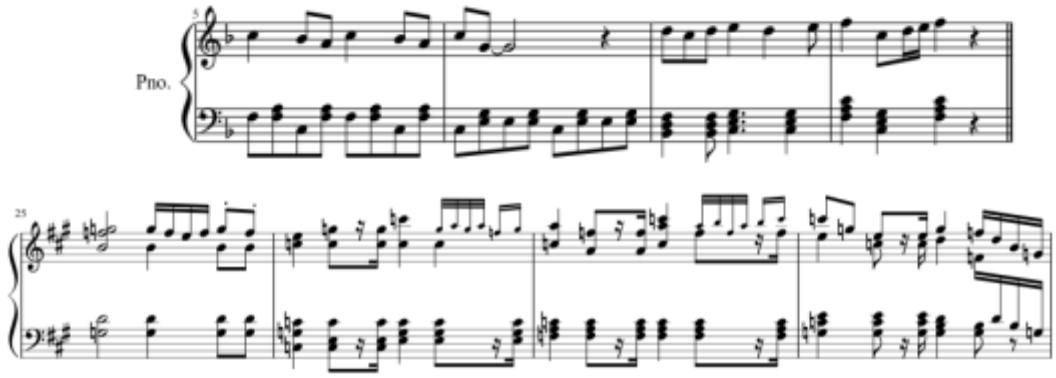


Рис.1.3.3.1. Примеры произведений разного уровня сложности

Кластеризация — это разбиение заданной выборки объектов на непересекающиеся подмножества (кластеры) такие, что каждый кластер состоит из близких объектов.

Пусть X — множество объектов, Y — множество идентификаторов (меток) кластеров. На множестве X задана функция расстояния между объектами ρ (x,x′).

Алгоритм кластеризации — функция a:X→Y, которая любому объекту x ∈ X ставит в соответствие идентификатор кластера y ∈ Y [4].

В качестве метода кластеризации был выбран алгоритм k-средних (англ. k-means). Итеративный алгоритм, основанный на минимизации суммарного квадратичного отклонения точек кластеров от центров этих кластеров.

Мера близости между векторами определяется как значение Евклидового расстояния.

Для определения числа кластеров используется следующий критерий – сумма квадратов расстояний от точек до центроидов кластеров, к которым они относятся [11]:

$$J(C) = \sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - \mu_k||^2 \to \min_C,$$

Где C – множество кластеров мощности K, μ_k – центроид кластера C_k.

Выбирается то число кластеров, начиная с которого величина J(C) падает "уже не так быстро", т.е. то число кластеров, после которого отсутствуют скачкообразные изменения значений величины [12].

В данной работе исходная выборка делится на 20 кластеров. Именно после 20 кластеров отсутствуют скачкообразные значения величины J(C) (см. рис. 1.3.3.2).

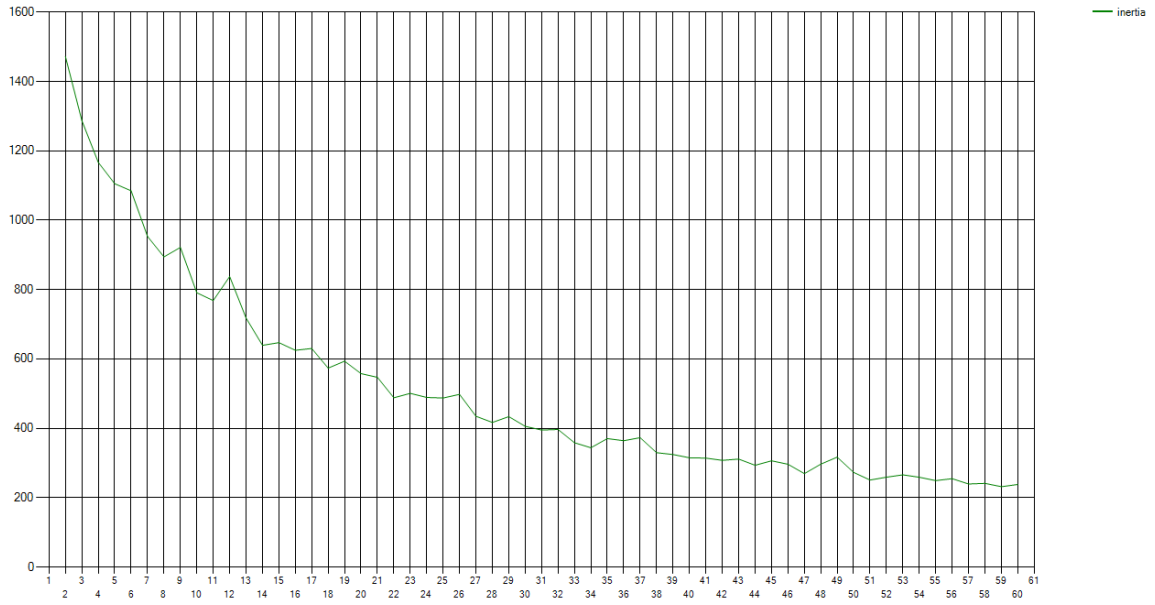


Рис. 1.3.3.2. График зависимости между числом кластеров и величиной коэффициента слияния J(C).

Центроиды полученных кластеров представлены в приложении 2.

2. Пользователю предлагается несколько произведений из различных кластеров. Он отмечает доступность: сможет он сыграть произведение или нет (пользователь получает наборы произведений до тех пор, пока не отметит).

В процессе предложения новых произведений используется адаптивный способ предъявления: даже если пользователь произведение отметил произведение, он получит еще несколько из этого же кластера, чтобы проверить, есть ли в этом кластере произведения, которые

пользователь не может сыграть, и составить представление о том, что вызывает сложность для данного пользователя.

3.Обучающая выборка

Обучающая выборка формируется на основе тех произведений, которые были предложены пользователю.

4. Построение классификатора

Отбор значимых признаков производится при помощи логистической регрессии, т.е. выполнения бинарной классификации произведений (доступно или нет произведение). Значимость признаков может варьироваться от пользователя к пользователю.

Логистическая регрессия – статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём подгонки данных к логистической кривой:

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

Где z - управляющий параметр (вход), а f(z) - функция (выход). Управляющий параметр может принимать значения от минус бесконечности до плюс бесконечности, тогда как функция ограничена диапазоном [0; 1]. f(z) представляет вероятность конкретного исхода, при заданном наборе параметров, а переменная z является мерой полного вклада параметров, используемых в модели:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k,$$

Где β_0 называют "точкой пересечения", а $\beta_1, \beta_2, \beta_3,$ и т. д. называют "коэффициентами регрессии" для управляющих параметров (признаков) x_1, x_2, x_3 соответственно. Точка пересечения – величина z при нулевых значениях всех управляющих параметров.

Каждый из коэффициентов регрессии описывает размер вклада соответствующего признака. Положительный коэффициент регрессии означает, что признак повышает вероятность анализируемого исхода, в то время как отрицательный коэффициент - признак уменьшает вероятность

анализируемого исхода. Большое значение коэффициента регрессии означает, что данный фактор существенно влияет на вероятность результата, в то время как почти нулевой коэффициент регрессии означает, что этот фактор имеет небольшое влияние. Логистическая регрессия – описание влияния одного или нескольких признаков (например, возраст, пол, и т.д.) на результат какого-либо события [10].

После построения данной модели можно получить значение коэффициента каждого признака и сделать оценку его влияния на вероятность принадлежности классу.

5. Кластеризация по значимым признакам для определенного пользователя

Из неразмеченных произведений осуществляется выбор так, чтобы они были из тех же кластеров что и произведения, отмеченные пользователем.

В итоге, объекты, которые при предварительной кластеризации были в одном кластере, в результате вторичной кластеризации уже по значимым признакам могут оказаться в разных кластерах (см. рис.1.3.3.3).

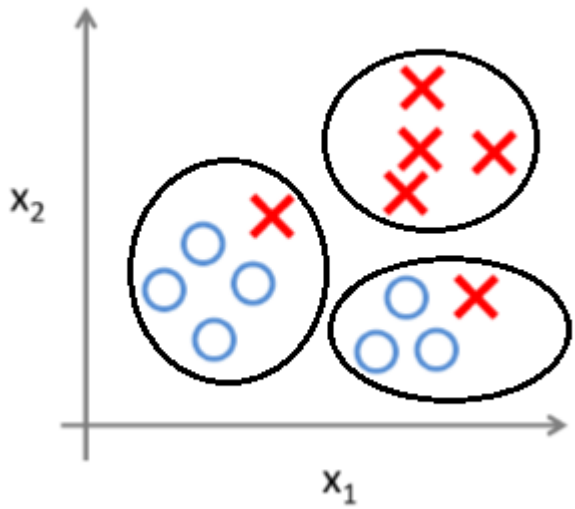


Рис.1.3.3.3. Распределение объектов по кластерам после кластеризации по значимым признакам

Глава 2. Разработка программного обеспечения

2.1. Используемые технологии

В качестве языка программирования для реализации поставленных задач был выбран язык C# в среде разработки Visual Studio.

Из сторонних библиотек для разработки использовались:

- Newtonsoft.Json – библиотека для работы с данными в формате json.
- Accord.NET Framework - .Net фреймворк для машинного обучения, включающий в себя библиотеки для обработки аудио- и видео, написанный на языке C#. В ходе выполнения данной работы были использованы следующие библиотеки из Accord .NET:
    - Accord.Statistics – данная библиотека содержит модели для прогнозирования, в том числе логистическую регрессию.
    - Accord.MachineLearning – библиотека, включающая классы для классификации, кластеризации, а также для обработки и визуализации полученных результатов.

Вышеперечисленные технологии были использованы для решения поставленной задачи, разбитой на 2 проекта:

1. Генерация CSV файла с набором данных для исследования
2. Работа с данными – поиск похожих произведений

2.2. Генерация CSV файла

2.2.1. Описание работы

В задачах машинного обучения данные представлены в виде текстового файла с расширением .csv. Данный проект на выходе генерирует этот файл.

Для решения данной задачи были изучены источники баз музыкальных произведений, предоставляющие доступ к своим данным в открытом доступе

или через специальные инструменты для разработчиков. Данными возможностями обладает источник MuseScore.

MuseScore – программное обеспечение для редактирования нотных партитур, предназначенное для операционных систем Windows, Mac OS X и Linux. Помимо этого, MuseScore – публичное сообщество, где пользователи со всего мира могут делиться, скачивать и печатать ноты музыкальных произведений совершенно бесплатно.

Доступ к файлам MuseScore осуществляется по ссылке:

http://musescore.com/static/musescore/scoredata/gen/{x}/{y}/{z}/{id}/{secret}/score.{extension},

где *extension* – расширение: *pdf*, *mid* (General MIDI), *mxl* (Compressed MusicXML), *mscz* (MuseScorefile), *mp3*.

x, *y*, *z* – это последние цифры идентификатора произведения. Например, если: id - 123456789, тогда x = 9, y = 8, z = 7.

Наборы файлов, таких как PNG, доступны по адресу: http://musescore.com/static/musescore/scoredata/gen/{x}/{y}/{z}/{id}/{secret}/score_{page_number}.png

где *{page_number}* – номер страницы (целое число, начиная с 0).

Для получения id и secret необходимо получить сведения о произведении, которые предоставляет MuseScore API.

Объекты от MuseScore API приходят в виде .json (или .xml), включают в себя метаданные (имя композитора, название произведения, число страниц и т.д.) и описание представления самого произведения в базе и доступа к нему (см. Приложение 3).

Для получения доступа к объектам базы данных MuseScore необходимы id и secret. Для получения информации о произведении нужны название, композитор и число страниц (title, composer и pages).

Также из всех произведений выбираются только с одной партией для фортепиано, что тоже можно учитывать: в массиве “parts” должна быть одна партия с названием “Piano” и кодом “0”.

Запрос на получение данных осуществляется по следующему адресу: http://api.musescore.com/services/rest/score.xml?oauth_consumer_key={api_key}&text={text_to_search}&page={page_number}, где

api_key –MuseScore API ключ,

page_number – номер станицы (данные приходят порционно по 20 объектов на страницу),

text_to_search – текст для поиска. В результирующем запросе, который используется для получения данных, производится запрос с ключевым словом “piano”. Не для всех объектов заполнено количество и виды партий, но они все равно удовлетворяют условиям поставленной задачи (произведение для фортепиано с одной партией), поэтому включение данного фильтра помогло увеличить количество произведений для дальнейшей работы.

После проверки на удовлетворение всем условиям идет обработка произведения – получение файла .xml и извлечение признаков.

http://musescore.com/static/musescore/scoredata/gen/{x}/{y}/{z}/{id}/{secret}/score.mxl

*.mxl – сжатый MusicXML, который необходимо разархивировать и найти в архиве нужный файл с расширением .xml.

Далее из полученного файла .xml извлекаются признаки.

Для извлечения признаков необходимо распарсить MusicXML. По каждому тегу предоставлено описание в документации MusicXML, остается только извлечь их средствами C#.

2.2.2. Диаграммы классов и их описание

На рисунке 2.2.2.1 представлены классы, которые были созданы для представления музыкального произведения в компьютере.

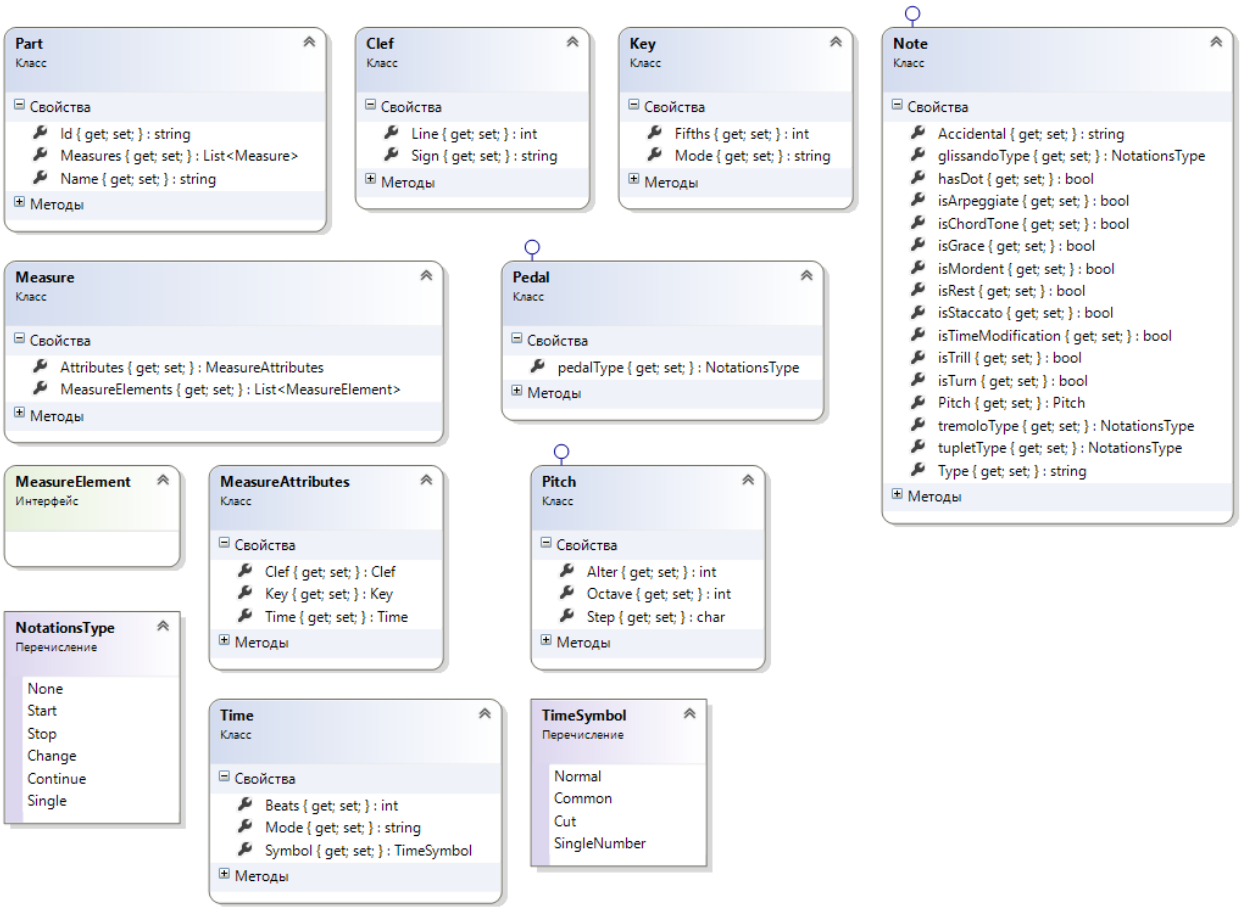


Рис.2.2.2.1. Диаграмма классов проекта GenerateCSV, часть 1.

Класс **Part** – партия. В MusicXML у каждой партии есть идентификатор (id), такты (Measures) и имя (Name).

Класс **Measure** – такт. Каждый такт обладает своими атрибутами (Attributes) и структурными элементами (MeasureElements).

Класс **MeasureAttributes** – атрибуты такта. В данные атрибуты входят: ключ (Clef), ключевые знаки (Key), размер (Time).

Класс **Clef** – ключ. Содержит в себе название ноты (Sign) и номер линии (Line) для определения вида ключа.

Класс **Key** – ключевые знаки. К ним относятся число ключевых знаков (Fifth) и лад тональности (Mode).

Класс **Time** – размер. Содержит в себе два значения: число основных долей в такте (Beats) и их относительную длительность (Mode), но размер может быть представлен не дробью, а символом (Symbol).

Перечисление **TimeSymbol** – содержит названия символов размера.

Интерфейс **MeasureElements –** для обозначения возможных элементов в такте. Его наследуют классы: Note, Pitch и Pedal.

Класс **Note** – нота, обладает следующими свойствами:

- *String Type* – тип ноты (целая, половинная, четверть, восьмая и т.д.)
- *Pitch Pitch* – высота звука
- *bool isChordTone* – входит ли текущая нота в аккорд
- *bool isRest* – является ли паузой
- *bool isGrace* – является ли форшлагом
- *NotationsTyp*e *glissandoType* – входит ли в глиссандо
- *NotationsType tupletType* – входит ли в триоль
- *bool isTimeModification* – присутствует ли изменения в ритме
- *bool isMordent* – присутствует ли мордент
- *bool isTurn* – присутствует ли группетто
- *bool isStaccato* – присутствует ли стаккато
- *string Accidental* – знак перед нотой (исключая ключевые)
- *bool isTrill* – присутствует ли трель
- *bool isArpeggiate* – входит ли нота в арпеджио
- *boo*l *hasDot* – присутствует ли пунктирный ритм
- *NotationsType tremoloType* – входит ли в тремоло

Перечисление **NotationsType** – содержит состояние музыкального элемента на момент исполнения данной ноты (начать, продолжать, завершить и др.)

Класс **Pitch** – высота звука. Отвечает за высоту ноты (Step) – обозначения от A до G, знак перед ней (Alter) и номер октавы (Octave).

Класс **Pedal** – педаль. Содержит состояние педали (PedalType).

Перечисление **PedalType** – содержит состояние педали (нажать, держать, убрать, отсутствует).

На рисунке 2.2.2.2 представлены классы, которые были созданы для распознавания файлов *.xml.

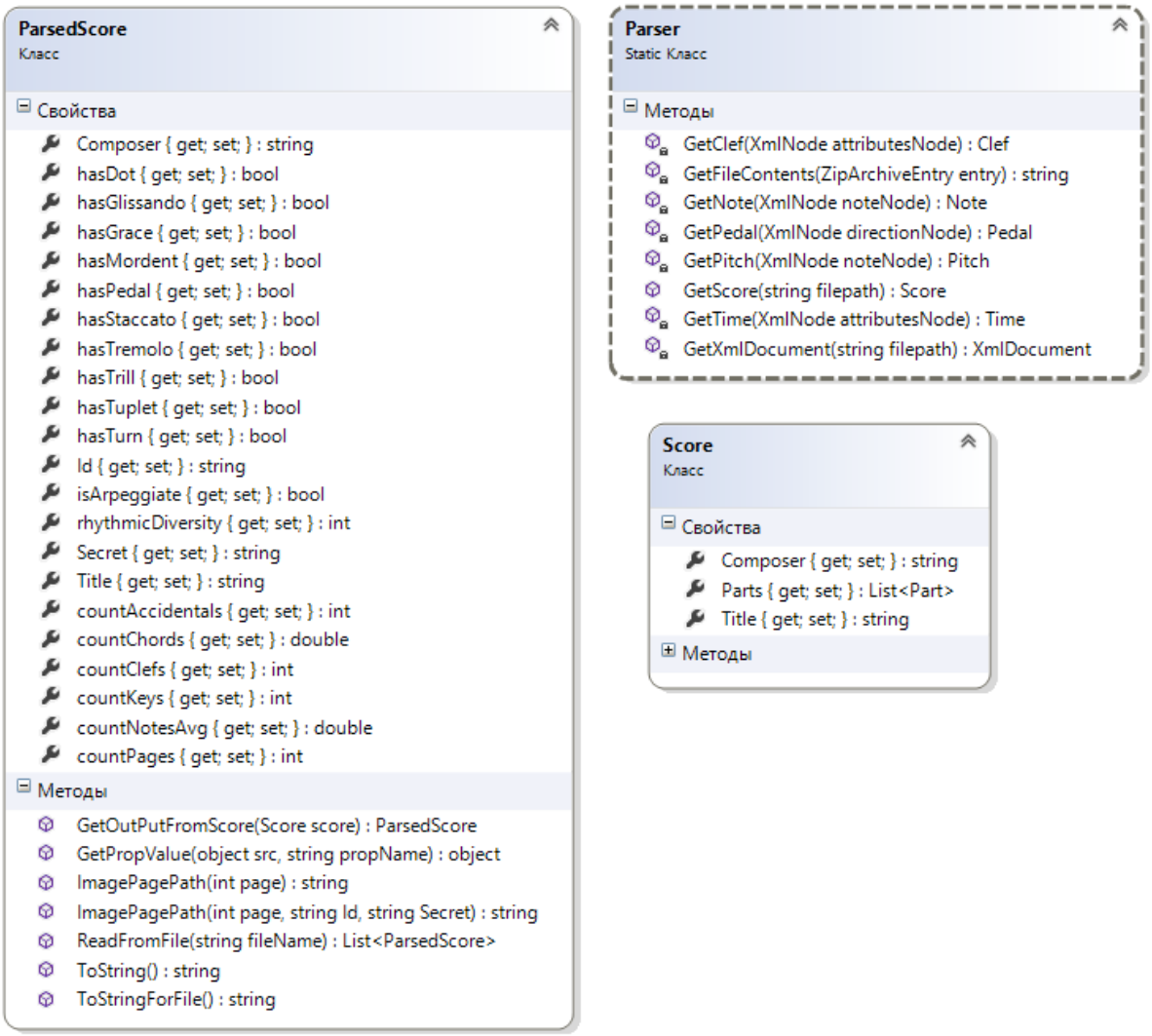


Рис.2.2.2.2. Диаграмма классов проекта GenerateCSV, часть 2.

Класс **Score** – произведение. У каждого произведения есть композитор (Composer), название (Title) и партии (Parts).

Класс **Parser** – для распознавания MusicXML. Содержит следующие методы:

public static Score GetScore(string filepath) – метод, который на вход получает строку для получения файла .mxl, на выходе генерирует объект класса Score (музыкальное произведение). При неверном пути или некорректном составлении XML - документа возвращает null. Для работы с узлами XML- документа используется библиотека System.Xml.

private static Pedal GetPedal(XmlNode directionNode) – метод для получения педали; на вход получает узел, одним из дочерних элементов которого выступает узел «pedal», на выходе генерирует объект класса Pedal, при условии присутствия педали в этом такте произведения (см. Приложение 4).

private static Note GetNote(XmlNode noteNode) – метод для получения ноты; на вход получает узел «note». Именно в этом методе происходит распознавание всех тех элементов, которые непосредственно относятся к ноте. На выходе генерирует объект класса Note.

private static Pitch GetPitch(XmlNode noteNode) – метод для получения высоты ноты.

private static Clef GetClef(XmlNode attributesNode) - метод для получения ключа; на вход передается узел атрибутов такта.

private static Time GetTime(XmlNode attributesNode) – метод для получения размера такта; на вход передается узел атрибутов такта.

private static XmlDocument GetXmlDocument(string filepath) – метод, который по адресной строке генерирует XML-документ. Средствами библиотеки System.Net отправляется запрос на получения документа по адресной строке (входной параметр), результат запроса сохраняется в поток. Затем выполняется распаковка полученного архива и получение XML-документа (см. Приложение 5).

private static string GetFileContents(ZipArchiveEntry entry) – на вход получает сжатый файл из zip-архива, возвращает содержимое файла одной строкой.

Класс **ParserScore** – класс для извлечения признаков. Содержит следующие свойства:

string Id – уникальный идентификатор MuseScore.

string Secret – уникальная строка MuseScore.

string Composer – композитор.

string Title – название.

double countNotesAvg – среднее число нот в такте.

int countKeys – количество ключевых знаков.

int countClefs – количество ключей.

int countAccidentals – количество случайных знаков.

double countChords – среднее число аккордов на такт.

int countPages – объем.

int rhythmicDiversity – ритмическое разнообразие.

bool hasGlissando – глиссандо.

bool hasTuplet – триоль.

bool hasMordent – мордент.

bool hasTurn – группетто.

bool hasStaccato – стаккато.

bool hasTrill – трель.

bool hasDot – пунктирный ритм.

bool hasPedal – педаль.

bool hasGrace – форшлаг.

bool isArpeggiate – арпеджио.

bool hasTremolo – тремоло.

Содержит следующие методы:

public ParsedScore GetOutPutFromScore(Score score) – метод для извлечения признаков. На вход получает произведение, считанное в класс Score парсером Parser, на выходе генерирует объект класса ParsedScore с извлеченными признаками.

public static object GetPropValue (object src, string propName) – метод для получения значения свойства по его названию. На вход получает объект, строку названия свойства, на выходе возвращает значение соответствующего свойства у объекта.

public override string ToString() – возвращает строковое представление у вызывающего данный метод экземпляра класса. Возвращает имя композитора и название произведения.

public string ToStringForFile() – возвращает строку из элементов класса, перечисленных через запятую, для записи в csv-файл.

public string ImagePagePath(int page) – возвращает строку – адрес для получения страницы номера page текущего произведения.

public static string ImagePagePath(int page,string Id,string Secret) – возвращает строку – адрес для получения страницы номера page произведения с соответствующими значениями Id и Secret из входных параметров.

public static List<ParsedScore> ReadFromFile(string fileName) – метод для считывания признаков из csv-файла. Входной параметр - путь к файлу (fileName).

За интерфейс приложения отвечает класс **GenerateCSVForm** (см. рис. 2.2.2.3).

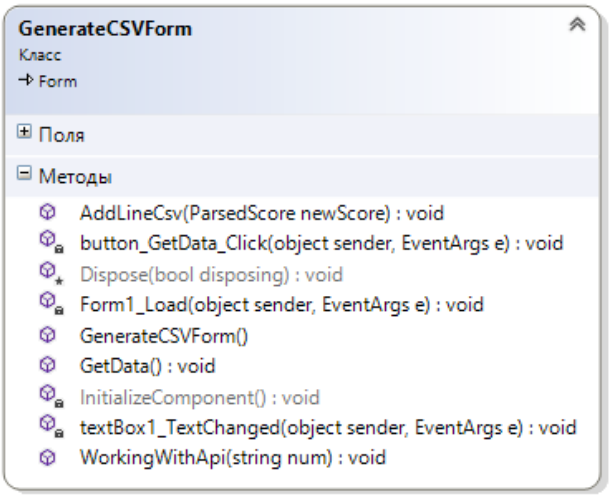


Рис.2.2.2.3. Класс GenerateCSVForm.

Основные методы:

public void WorkingWithApi(string num) – метод для получения и обработки данных с MuseScoreAPI. На вход получает номер страницы ответа от MuseScore API. Для каждой страницы выполняется запрос на получение данных в формате json. Далее для каждого полученного объекта выполняется проверка на соответствие критериям поиска, в случае ее успеха – получение XML – документа, его распознавание, извлечение признаков и их последующее сохранение в строку для записи в файл (см. Приложение 6).

public void GetData() – данный метод предназначен для запуска работы с API (метода WorkingWithApi). Обновляет компоненты состояния на форме, записывает итоговую строку с признаками в файл *.csv.

2.3. Поиск похожих произведений

2.3.1. Разработка приложения

Работа приложения разделена на несколько этапов:

1. Вывод нот на экран: просмотр всех страниц, краткой информации и отметки пользователя.
2. Подбор похожих произведений
3. Вывод результатов поиска на экран

2.3.2. Вывод нот на экран

Для вывода нот на экран и сохранения выбора пользователя реализован класс State (см. рис.2.3.2.1).

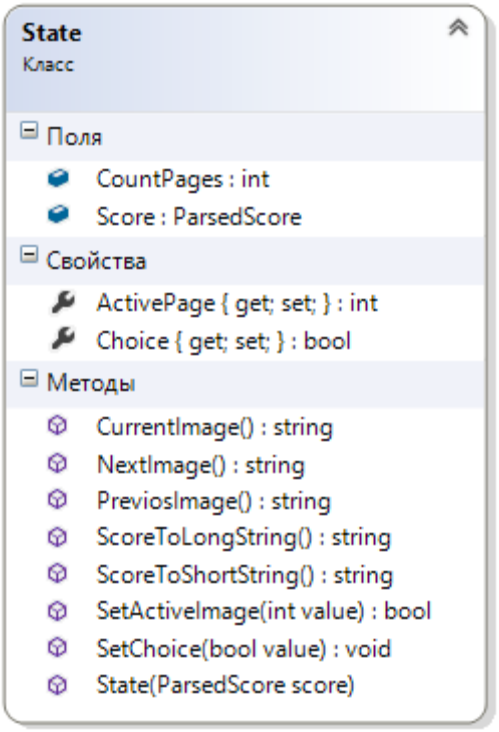


Рис. 2.3.2.1. Класс State

Поля:

public readonly int CountPages – количество страниц в произведении.

public readonly ParsedScore Score – произведение для вывода на экран.

Свойства:

public int ActivePage – текущая страница для отображения на экран (индекс с 0) – только для чтения.

public bool Choice – выбор пользователя (положительный/отрицательный) – только для чтения.

Методы:

public string CurrentImage() – возвращает ссылку на адрес текущей страницы (используя значение ActivePage).

public string NextImage() – увеличивает значение текущей страницы на 1 и возвращает ссылку на адрес текущей страницы.

public string PreviosImage() – уменьшает значение текущей страницы на 1 и возвращает ссылку на адрес текущей страницы.

public string ScoreToLongString() – возвращает строку подробной информации о произведении.

public string ScoreToShortString() – возвращает строку краткой информации о произведении.

public bool SetActiveImage(int value) – устанавливает номер текущей страницы, при условии корректного значения входного параметра (в диапазоне количества страниц). Возвращает результат операции.

public void SetChoice(bool value) – устанавливает значение выбора пользователя.

public State(ParsedScore score) – конструктор, входной параметр – объект признаков произведения (ParsedScore).

2.3.3. Подбор похожих произведений

Класс Searcher отвечает за реализацию алгоритма подбора похожих объектов (см. рис.2.3.3.1).

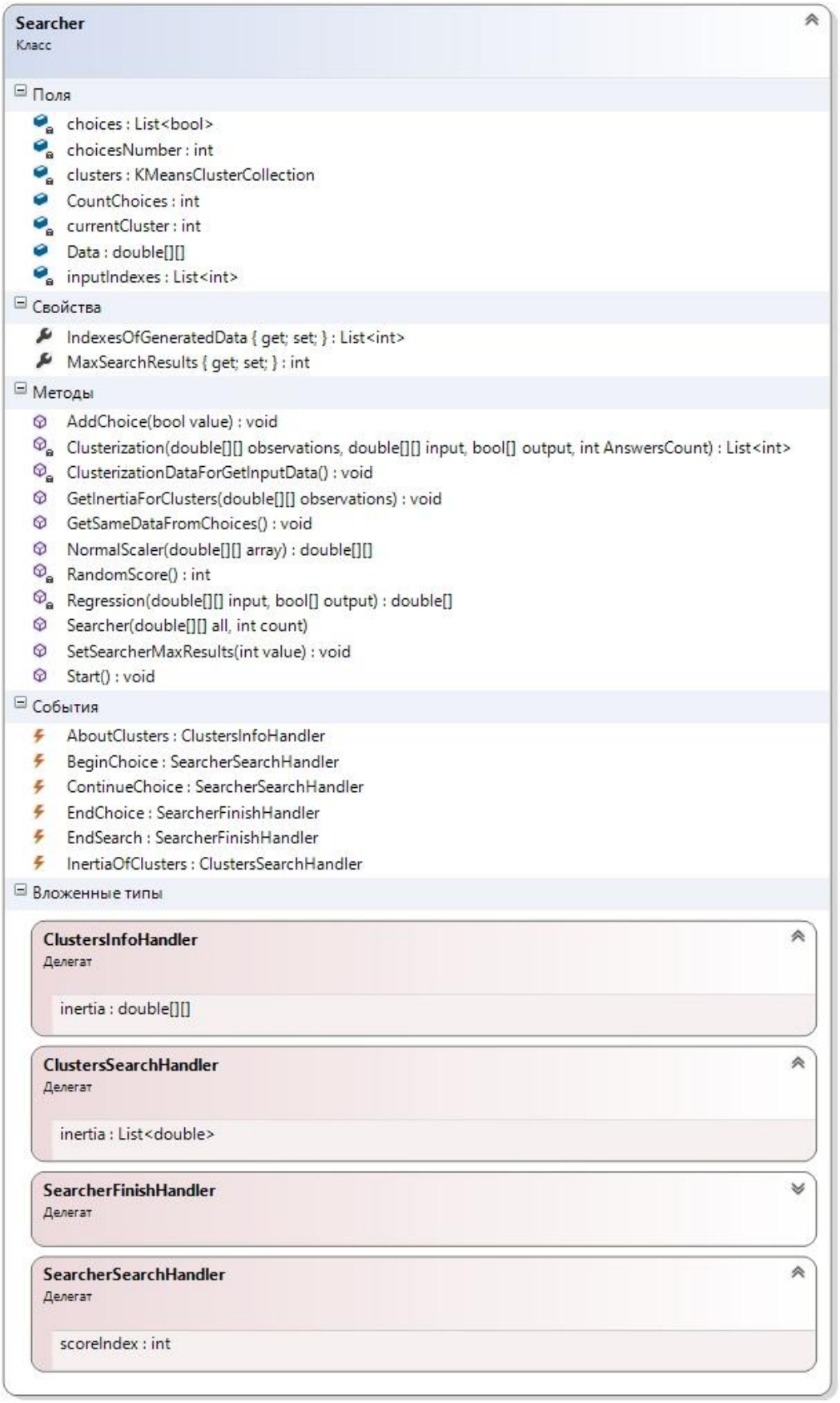


Рис. 2.3.3.1. Класс Searcher

Поля:

public readonly double[][] Data – набор данных, по которым производится поиск.

public readonly int CountChoices – минимальное число объектов в выборке, предлагаемой пользователю.

private int choicesNumber – порядковый номер выборки, предлагаемой пользователю.

private int currentCluster – номер текущего кластера, из которого пользователю предлагаются произведения.

private KMeansClusterCollection clusters – коллекция полученных кластеров для генерации выборки из исходного набора.

private List<State> choices – список для хранения значений ответов пользователя.

private List<int> inputIndexes – список для хранения индексов случайных объектов из исходного набора, предлагаемых пользователю для оценки.

Свойства:

public List<int> IndexesOfGeneratedData { get; private set; } – результат поиска похожих объектов (список индексов объектов для поиска в исходном наборе).

public int MaxSearchResults{ get; private set; } – максимальное число произведений для поиска.

Типы:

public delegate void SearcherSearchHandler (int scoreIndex) – тип метода с входным параметром целочисленного типа.

public delegate void SearcherFinishHandler() – тип метода без входных параметров.

public delegate void ClustersSearchHandler(List<double> inertia) – тип метода с входным параметром в виде списка вещественных значений.

public delegate void ClustersInfoHandler(double[][] inertia) – тип метода с входным параметром в виде массива массивов вещественных значений.

События:

public event SearcherSearchHandler BeginChoice – событие, возникающее в момент начала выбора пользователем объектов.

public event SearcherFinishHandler EndChoice – событие, возникающее в момент завершения выбора пользователем объектов.

public event SearcherFinishHandler EndSearch – событие, возникающее в момент завершения поиска похожих объектов.

public event SearcherSearchHandler ContinueChoice – событие, возникающее в момент следующего выбора пользователем объектов.

public event ClustersSearchHandler InertiaOfClusters – событие, возникающее в момент завершения вычисления инерции кластеров.

public event ClustersInfoHandler AboutClusters – событие, возникающее в момент завершения кластеризации исходных данных.

Методы:

public Searcher(double[][] all, int count) – конструктор, входные параметры: исходный набор данных all и минимальное число объектов для оценки пользователем (count). Инициализация данных, приведение исходного набора (Data) к единой шкале (от 0 до 1) .

public void SetSearcherMaxResults(int value) – метод, который устанавливает значение для максимального количества произведений для поиска (MaxSearchResults) при условии корректного значения value.

public void Start() – запуск поиска новых произведений, инициализация списков choices и inputIndexes, задание значений по умолчанию для полей choicesNumber и currentCluster. Вызов событий AboutClusters и BeginChoice.

public void AddChoice (bool value) – сохранение оценки пользователя. В данном методе проверяется, достаточно ли данных, которые были получены от пользователя, и если их недостаточно – увеличивается номер текущей

выборки (choicesNumber), обнуляется номер текущего кластера (currentCluster) и пользователю предлагается следующее произведение из выборки посредством вызова события ContinueChoice. Если данных недостаточно, пользователю предлагается следующее произведение из выборки посредством вызова события ContinueChoice. В противном случае вызывается событие EndChoice и метод GetSameDataFromChoices.

public void GetSameDataFromChoices() – основной метод класса. В данном методе определяются похожие объекты. В данном методе выделяются наиболее важные признаки из выборки, предложенной пользователю (метод Regression), и выполняется кластеризация всего набора данных с последовательным выбором подходящих объектов на основе кластеров (метод Clusterization). Индексы полученных объектов сохраняются в список IndexesOfGeneratedData и вызывается событие завершения поиска EndSearch (см. Приложение 7).

private int RandomScore() – метод возвращает индекс случайного объекта, номер кластера которого совпадает с номером текущего кластера (currentCluster). Если последнее произведение не было отмечено пользователем положительно, значение текущего кластера увеличивается, в противном случае – не изменяется.

public double[][] NormalScaler(double[][] array) – приведение значений элементов массива (array) к единой шкале (интервал [0,1]).

private double[] Regression(double[][] input, bool[] output) – метод для логистической регрессии. На вход подается массив объектов (input) и бинарные значения, соответствующие этим объектам (output). На выходе: массив коэффициентов логистической функции для каждого признака.

private void ClusterizationDataForGetInputData() – кластеризация для исходного набора данных. Данные о полученных кластерах сохраняются в переменную clusters. Вызывается событие для получения информации о кластерах AboutClusters.

private void public void GetInertiaForClusters(double[][] observations) – метод для вычисления инерции для кластеров, которая вычисляется на основе суммы квадратов расстояний от точек до центроидов кластеров, к которым они относятся.

private List<int> Clusterization(double[][] observations, double[][] input, bool[] output, int AnswersCount) – кластеризация выборки по отобранным признакам. На вход передаются исходный набор данных (observations), количество произведений, которые необходимо найти (AnswersCount), выборка для пользователя (input) – с признаками, выделенными регрессией; для выборки передается массив бинарных значений ответов пользователя (output) (см. Приложение 8).

За интерфейс приложения отвечает класс SearcherForm (см. рис.2.3.3.3).

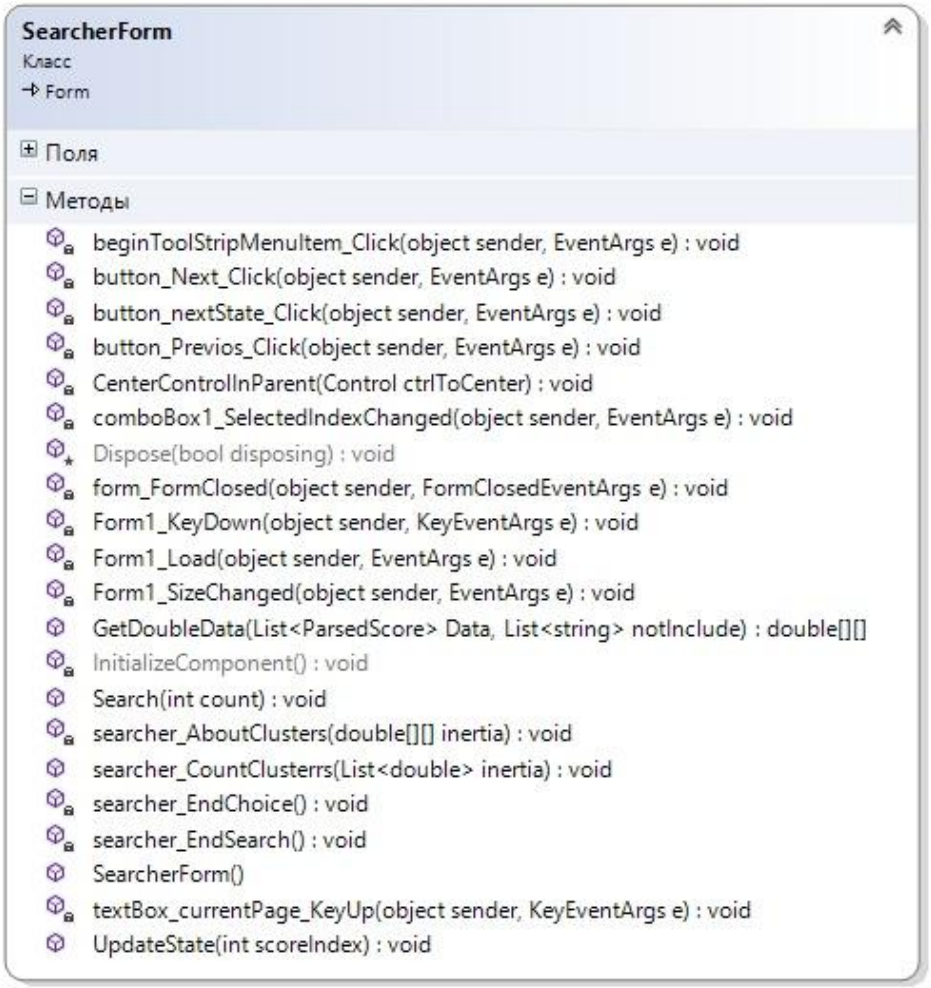


Рис. 2.3.3.3. Класс FormSearcher

Основные методы:

public void Search(int count) – метод активации поиска: инициализация класса Searcher, добавление обработчиков на его события, изменения в интерфейсе. Входной параметр – параметр для Searcher для количества объектов в выборке.

public double[][] GetDoubleData(List<ParsedScore> Data, List<string> notInclude) – подготовка данных для класса Searcher. Выделение всех не строковых признаков из списка типа ParsedScore и тех, которые не входят в список исключенных признаков, и запись их в массив вещественных чисел (см. Приложение 9).

void searcher_EndSearch() – вывод результатов поиска в ListBox.

void searcher_EndChoice() – вывод формы для установки значения количества произведений для поиска.

public void UpdateState(int scoreIndex) – обновление внешнего интерфейса. На вход передается индекс объекта в исходном наборе, который необходимо вывести на экран.

Для установки значения количества произведений для поиска реализован класс FromSetValue (см. рис.2.3.3.4).

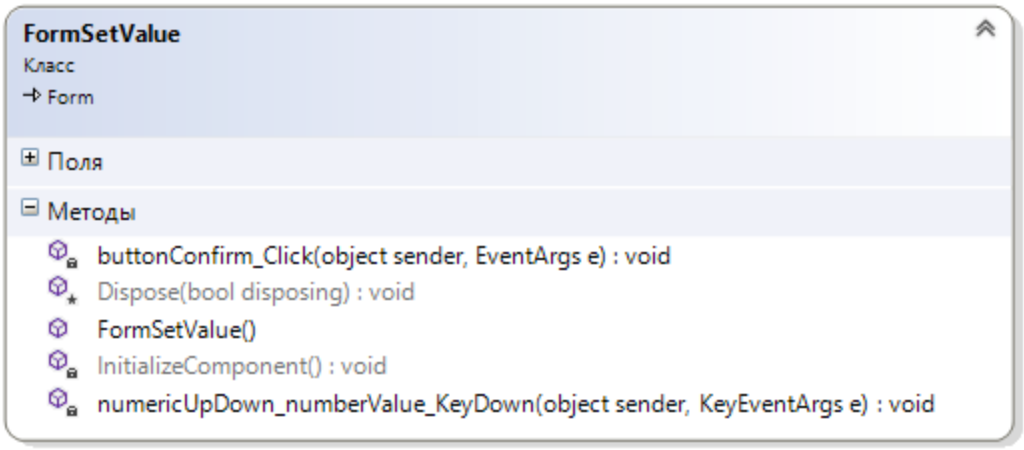


Рис. 2.3.3.4. Класс FormSetValue.

Данная форма содержит поле для установки значения и кнопку для подтверждения введенного значения. По умолчанию установлено значение 5.

2.4. Описание работы приложений

2.4.1. Приложение для извлечения признаков

При первом запуске приложения появляется главное окно приложения (см. рис. 2.4.1.1).

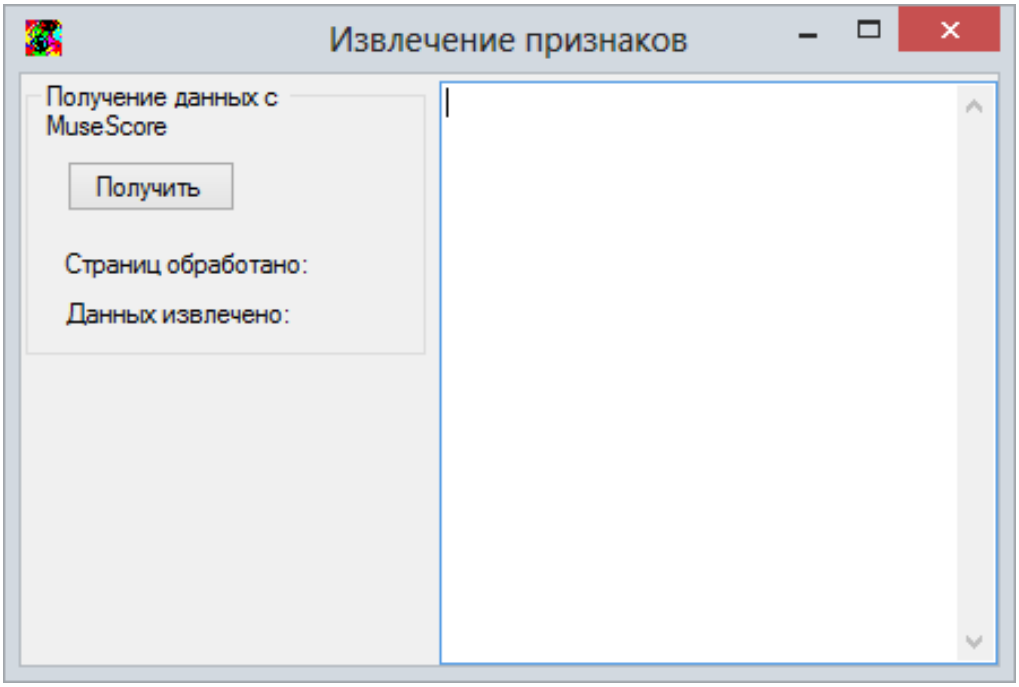


Рис.2.4.1.1. Главное окно приложения

При нажатии на кнопку «Получить» выполняется запуск работы с MuseScore API и получение данных (см. рис.2.4.1.2). В строках состояния («Страниц обработано:» и «Данных извлечено:») отображаются текущие результаты работы приложения. Также в текстовое поле записывается краткая информация об извлекаемых данных (название, композитор, количество страниц). По нажатию кнопки «Завершить» в результирующий файл записываются все данные (извлеченные признаки), полученные на момент нажатия кнопки.

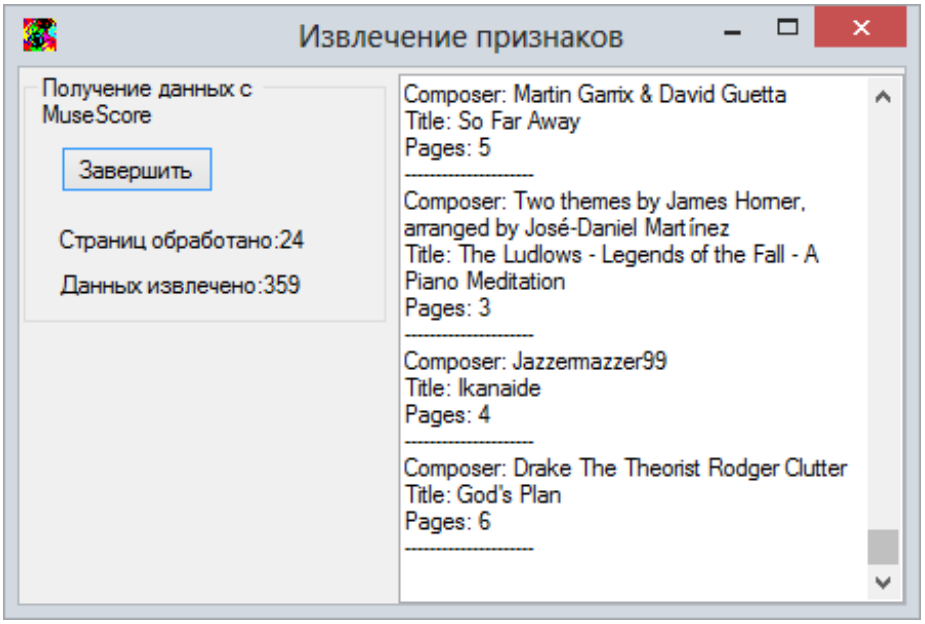


Рис.2.4.1.2 Процесс извлечения признаков

По завершению работы приложения (см. рис. 2.4.1.3) отображается, сколько страниц было обработано и сколько объектов с данными было извлечено и записано в файл.

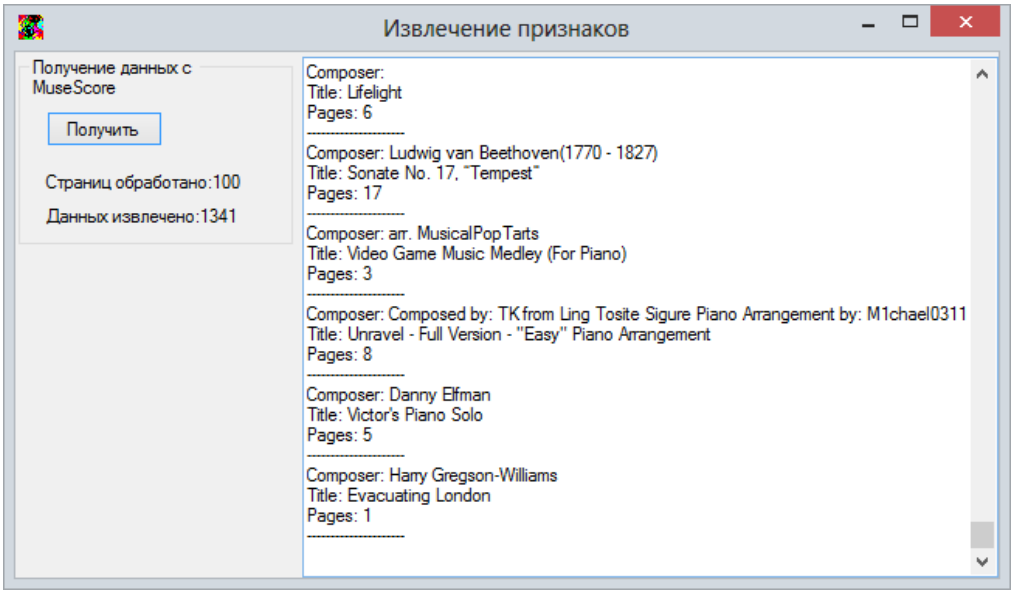


Рис.2.4.1.3. Извлечение признаков завершено

2.4.2. Приложение для поиска произведений

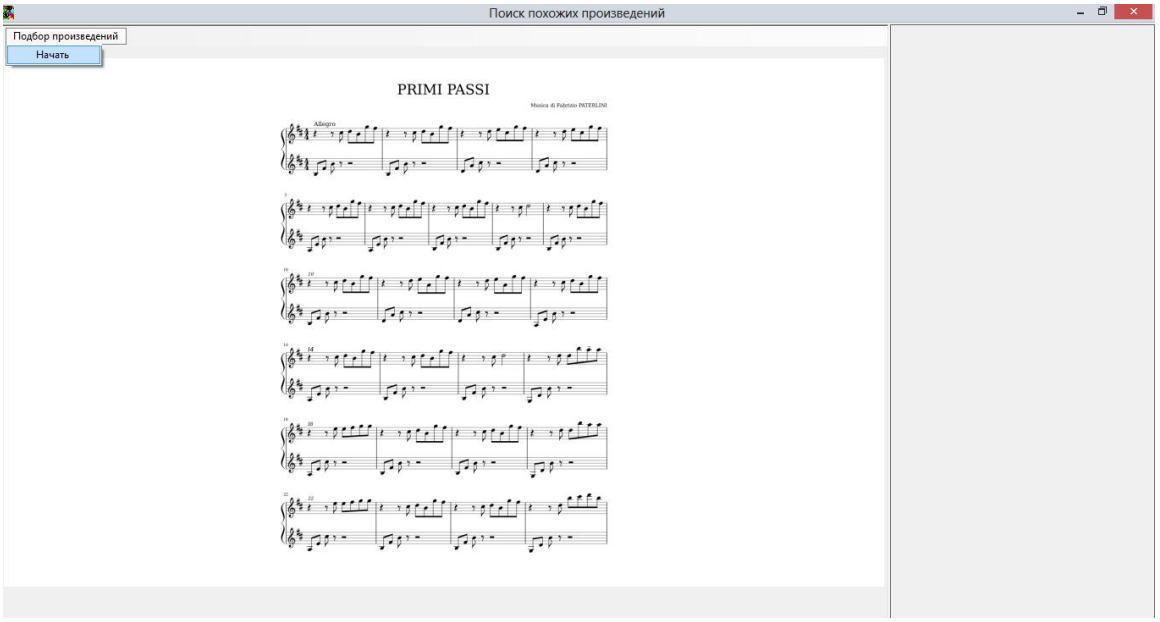


Рис. 2.4.2.1. Стартовое окно приложения

При запуске приложения пользователю открывается стартовое окно приложения (см. рис.2.4.2.1). Через меню «Подбор произведений», «Начать» начинается подбор произведений (см. рис. 2.4.2.2).

Пользователю предлагается несколько произведений. По каждому произведению предоставляется основная информация о произведении, поле для передачи ответа о возможности сыграть произведение (если пользователь может сыграть – ставит отметку, в противном случае сразу нажимает кнопку «Следующее произведение»). Также пользователь может просмотреть произведение целиком. Для этого есть панель навигации для перехода на следующую/предыдущую страницу через кнопки или по номеру напрямую. Нажатие на кнопку «Следующее произведение» выдает пользователю следующее произведение.

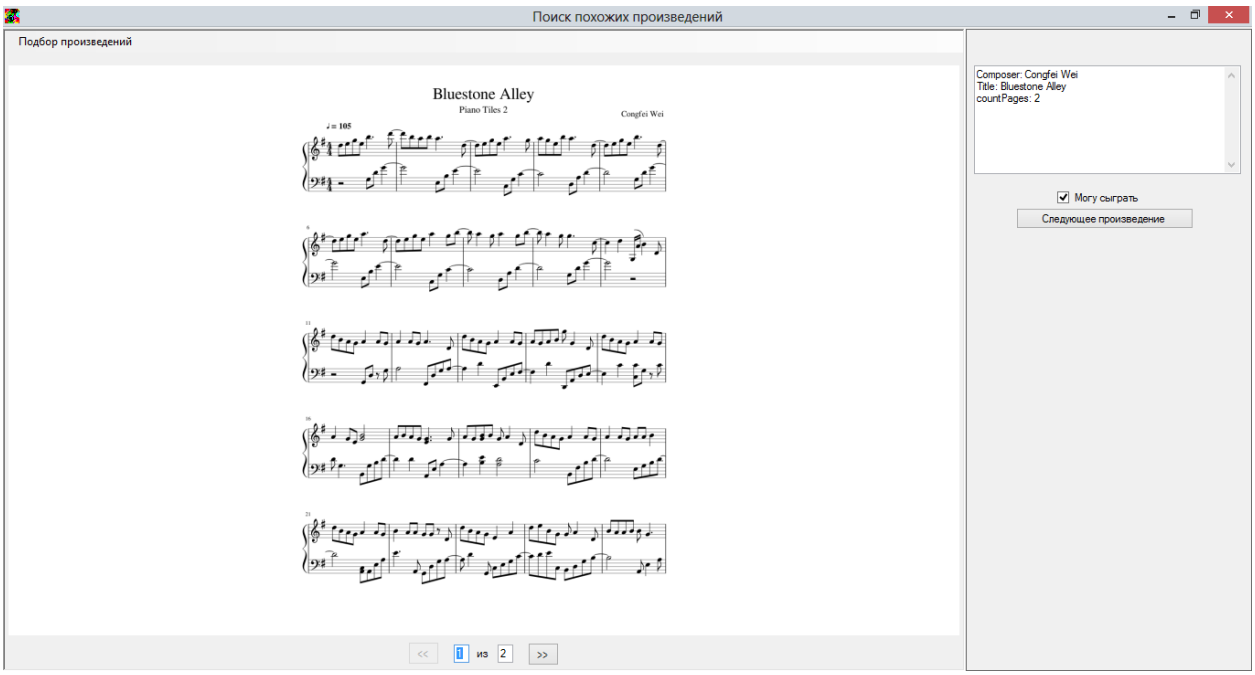


Рис.2.4.2.2. Окно подбора произведений

После сбора необходимой информации пользователю предоставляется возможность установить количество произведений для поиска (см. рис. 2.4.2.3.). Значение можно подтвердить нажатием кнопки «Подтвердить» или нажатием клавиши «Enter» на клавиатуре.

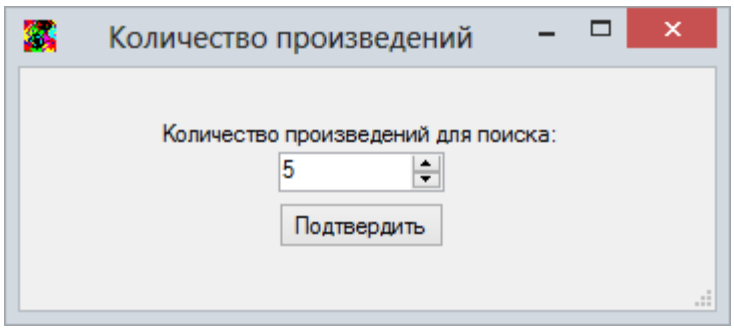


Рис.2.4.2.3. Окно для ввода значения количества произведений для поиска

Затем на экран выводится список произведений, подобранных приложением (см. рис. 2.4.2.4). Пользователь также может просмотреть полученные произведения и краткую информацию, которая позволит ему скачать произведение с MuseScore.com или же с другого ресурса в случае, если произведение удовлетворяет предпочтениям пользователя.

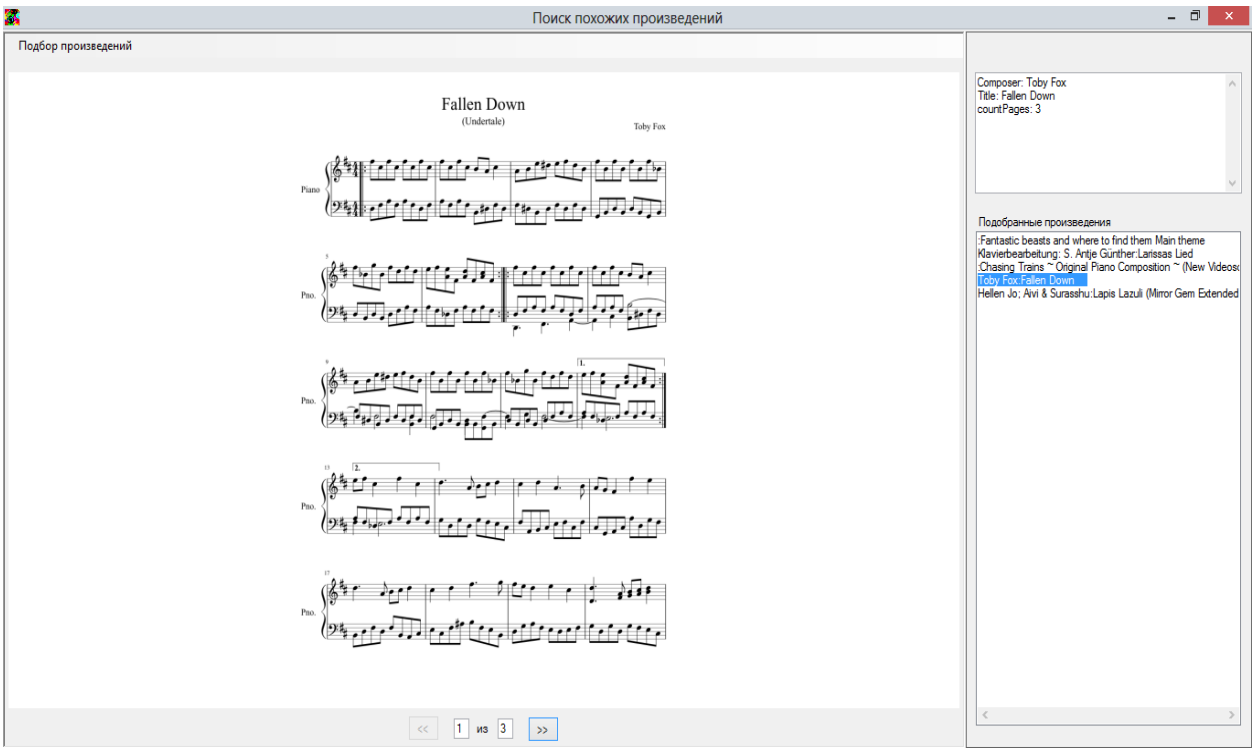


Рис.2.4.2.4. Окно вывода результатов работы приложения

Заключение

В результате выполнения выпускной квалификационной работы было разработано приложение для поиска музыкальных произведений с помощью методов машинного обучения.

Работа приложения основана на использовании базы данных музыкальных произведений сообщества MuseScore, которое предоставляет доступ к произведениям в форматах MuseScore, MusicXML, PDF, PNG, MIDI и MP3. На основе собственной экспертной оценки были определены признаки, характеризующие музыкальное произведение. Для извлечения этих признаков было написано специально приложение.

Для поиска похожих произведений был разработан алгоритм, основанный на предварительном опросе пользователя для уточнения его предпочтений, выявлении значимых признаков на основе выявленных предпочтений и подборе произведений на основе доступности для пользователя.

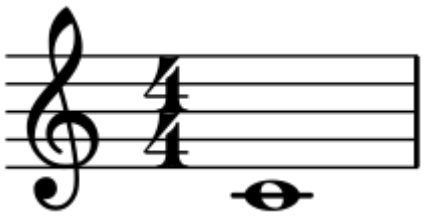
Список литературы

1. Троелсен Э., Язык программирования C# 5.0 и платформа .NET 4.5. — 6-е издание. — Вильямс, 2013. — 1311 с.
2. Accord .NET. [Электронный ресурс], - accord-framework.net/docs/Index.html (дата обращения 13.05.2019)
3. Логистическая регрессия [Электронный ресурс], - http://neerc.ifmo.ru/wiki/index.php?title=Логистическая_регрессия (дата обращения 15.04.2019)
4. Кластеризация. [Электронный ресурс], - http://neerc.ifmo.ru/wiki/index.php?title=Кластеризация (дата обращения 22.04.2019)
5. Рашка С..Python и машинное обучение. — М.: ДМК Пресс,2017 – 418с.
6. Луис Педро Коэло, Вилли Ричарт. Построение систем машинного обучения на языке Python. . — 2-е изд. — М.: ДМК Пресс,2016 – 302 с.
7. MuseScore API documentation [Электронный ресурс], - http://developers.musescore.com/ (дата обращения 16.03.2019)
8. MusicXML Documentation [Электронный ресурс], - http://usermanuals.musicxml.com/MusicXML/MusicXML.htm (дата обращения 28.02.2019)
9. Современная музыкальная нотация – Википедия https://ru.wikipedia.org/wiki/Современная_музыкальная_нотация (дата обращения 20.03.2019)
10. Логит-анализ [Электронный ресурс], - http://www.machinelearning.ru/wiki/index.php?title= Логит-анализ (дата обращения 22.04.2019)
11. Открытый курс машинного обучения. Тема 7. Обучение без учителя: PCA и кластеризация / Блог компании Open Data Science / Хабр [Электронный ресурс], - https://habr.com/ru/company/ods/blog/325654/ (дата обращения 23.05.2019)

12. Факторный, дискриминантный и кластерный анализ: Пер. с англ./Дж.-О. Ким, Ч. У. Мьюллер, У. Р. Клекка и др.; Под ред. И. С. Енюкова. — М.: Финансы и статистика, 1989.— 215 с.

Приложение 1.

Пример представления произведения с одним тактом, содержащим одну целую ноту до первой октавы, с размером 4/4 в MusicXML.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
   "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
   "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.1">
 <part-list>
  <score-part id="P1">
   <part-name>Music</part-name>
  </score-part>
 </part-list>
 <part id="P1">
  <measure number="1">
   <attributes>
    <divisions>1</divisions>
    <key>
     <fifths>0</fifths>
    </key>
    <time>
     <beats>4</beats>
     <beat-type>4</beat-type>
    </time>
    <clef>
```

```
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
    <note>
      <pitch>
        <step>C</step>
        <octave>4</octave>
      </pitch>
      <duration>4</duration>
      <type>whole</type>
    </note>
  </measure>
 </part>
</score-partwise>
```

Приложение 2.

Центроиды кластеров

	countNotesAvg	countKeys	countClefs	countAccidentals	countChords	countPages	hasGlissando	hasTuplet	hasMordent	hasStaccato	hasTrill	hasPedal	hasGrace	isArpeggiate	hasTremolo
1	0,2807	0,2755	0,0408	0,2521	0,2504	0,126	0,0102	1	0	0	0	0	0	0,2857	0,0408
2	0,3822	0,3314	0,0591	0,2186	0,2878	0,1583	0,04	1	0,08	0	0,04	1	1	0,72	0,08
3	0,2768	0,2746	0,0255	0,2166	0,2941	0,1135	0,013	0	0,026	1	0,013	0	0	0	0
4	0,326	0,315	0,076	0,3351	0,3218	0,1616	0,0256	0,5641	0,0769	1	0,0513	1	0	1	0
5	0,2197	0,119	0,0872	0,4091	0,1715	0,1304	0	0,5	0,1667	0,1667	1	0,1667	0,8333	1	0
6	0,3201	0,2318	0,0702	0,3817	0,3256	0,1411	0,0566	1	0,0566	1	0,0189	0,2642	0	0	0,1321
7	0,1843	0,1499	0,0129	0,0972	0,1363	0,0673	0,0049	0	0,0049	0	0,0049	0	0	0	0
8	0,4361	0,3197	0,062	0,3405	0,4458	0,2153	0,1429	0,5714	0	0,9524	0	1	1	0,9524	0,381
9	0,3462	0,3	0,0271	0,229	0,3277	0,1145	0,0667	0	0,0333	1	0,0333	0,3333	1	0	0,0333
10	0,3059	0,3131	0,028	0,1648	0,283	0,1135	0,0142	0	0	0,2411	0	1	0	0,2624	0,0496
11	0,3231	0,5205	0,0198	0,206	0,3746	0,1309	0,0099	0	0,0198	0	0	0	0	0	0,0099
12	0,3191	0,3333	0,0323	0,0707	0,2671	0,1256	0	0	0,1111	0	0	0	1	1	0
13	0,3049	0,3071	0,0471	0,2793	0,2783	0,1609	0,05	1	0	1	0	0	0	1	0
14	0,3527	0,2275	0,0293	0,4119	0,3466	0,124	0,0741	1	0,0741	1	0,1111	0,2222	1	0,1852	0
15	0,3078	0,3243	0,0409	0,3808	0,2668	0,1304	0,0541	1	0,0541	0	0	0	1	0,4595	0,0541
16	0,2444	0,0952	0,0698	0,3771	0,0485	0,058	0	0,3333	0	0	1	0	1	0	0
17	0,2581	0,3172	0,0236	0,1441	0,2659	0,0972	0	0	0,0294	0	0	0	0	1	0
18	0,3709	0,28	0,047	0,2024	0,3954	0,1504	0	1	0,04	0	0,02	1	0	0,52	0,06
19	0,3233	0,3203	0,0334	0,1303	0,2577	0,1094	0	0	0,0323	0	0	0,629	1	0,2903	0
20	0,2964	0,2906	0,0417	0,1975	0,2753	0,1424	0,069	0	0,0345	1	0	0	0,2759	1	0,069

Приложение 3.

Пример объекта, возвращаемого MuseScore API в формате JSON:


```
{
  "id":"1111",
  "vid":"152152",
  "dates":{
    "posted":"1370593443",
    "lastupdate":"1370593602",
    "revised":"1370593603"
  },
  "secret":"b364acd291",
  "uri":"https:\/\/api.musescore.com\/services\/rest\/score\/1111",
  "permalink":"https:\/\/musescore.com\/score\/1111",
  "custom_url":"https://musescore.com\/nicolas\/great-score",
  "user":{
    "uid":"400",
    "username":"Mozart",
    "custom_url":"https://musescore.com\/nicolas"
  },
  "status":"ready",
  "sharing":"public",
  "comment_count":"0",
  "favoriting_count":0,
  "user_favorite":0,
  "view_count":"14",
  "playback_count":"6",
  "download_count":4,
  "genre":"",
  "format":"",
```

"tags":"tag1, tag2",
"license":"publicdomain",
"language":null,
"title":"Wilder Reiter",
"description":"Album f\u00fcr die Jugend (Album for the Young), Op. 68, was composed by Robert Schumann in 1848 for his three daughters. The album consists of a collection of 43 short works. The Wild Rider (Wilder Reiter) in A minor is more commonly known in English as The Wild Horseman.",
"metadata":{
"title":"My First score",
"subtitle":null,
"composer":"WA Mozart",
"poet":null,
"pages":"9",
"measures":"102",
"lyrics":"0",
"chordnames":"0",
"keysig":"0",
"duration":"474",
"dimensions":"210x297",
"parts":[
{
"part":{
"name":"Flute",
"program":"73"
}
},
{
"part":{

```
            "name":"Piano",
            "program":"0"
          }
        }
      ]
    }
}
```

Приложение 4.

Код метода GetPedal класса Parser:

```
            private static Pedal GetPedal(XmlNode directionNode)
            {
                var pedal = new Pedal();
                if (directionNode != null)
                {
                    var directiontypeNode =
directionNode.SelectSingleNode("direction-type");
                    if (directiontypeNode != null)
                    {
                        var pedalNode =
directiontypeNode.SelectSingleNode("pedal");
                        if (pedalNode != null)
                        {
                            var type =
pedalNode.Attributes.GetNamedItem("type").Value;
                            pedal.pedalType =
(NotationsType)Enum.Parse(typeof(NotationsType),
type.First().ToString().ToUpper() + type.Substring(1));
                        }
                    }

                }
                return pedal.pedalType!=NotationsType.None?pedal:null;
        }
```

Приложение 5.

Код метода GetXmlDocument класса Parser:

```
private static XmlDocument GetXmlDocument(string filepath)
            {
                var document = new XmlDocument();

                WebRequest request = WebRequest.Create(filepath);
                System.Net.WebResponse response =
request.GetResponse();
                System.IO.Stream responseStream =
response.GetResponseStream();
                using (ZipArchive archive = new
ZipArchive(responseStream, ZipArchiveMode.Read))
                {
                    var entry = archive.Entries[1];

                    string xml = GetFileContents(entry);
                    document.XmlResolver = null;

                    document.LoadXml(xml);

                }
                return document;
                 }
```

Приложение 6.

Код метода WorkingWithApi класса GenerateCSVForm:

```
public void WorkingWithApi(string num)
        {
            WebRequest request =
WebRequest.Create("http://api.musescore.com/services/rest/score.json?"
            + "oauth_consumer_key=musichackday2016"
            + "&text=piano"
            + "&page=" + num);

            WebResponse response = request.GetResponse();

            using (StreamReader stream = new
StreamReader(response.GetResponseStream()))
            {
                string line;
                if ((line = stream.ReadLine()) != null)
                {
                    var curr =
JsonConvert.DeserializeObject<dynamic>(line);
                    foreach (var obj in curr)
                    {
                        var meta = obj.metadata;
                        var title = meta.title;
                        if (meta.parts == null || meta.parts.Count ==
1 && meta.parts[0].program == 0)
                        {
                            if (meta.title == null || meta.title ==
"")
                            {
                                title = obj.title;
                            }

                            var id = obj.id.ToString();
                            var secret = obj.secret.ToString();
                            var filepath =
string.Format("https://musescore.com/static/musescore/scoredata/gen/{0
}/{1}/{2}/{3}/{4}/score.mxl", id[id.Length - 1], id[id.Length - 2],
id[id.Length - 3], id, secret);
                            var score = Parser.GetScore(filepath);
                            if (score != null)
                            {
                                scores.Add(score);
                                var output = new ParsedScore();
                                output.countPages = meta.pages;
                                output.Id = id;
                                output.Secret = secret;
```

```
                                output.Composer =
meta.composer.ToString().Replace(',', ';').Replace('\n', ';');
                                output.Title =
title.ToString().Replace(',', ';').Replace('\n', ';');

AddLineCsv(output.GetOutPutFromScore(score).ToStringForFile());
                                outputs.Add(output);

                                this.Invoke(new Action(() =>
                                {

                                    textBox_Info.Text += "Composer: "
+ meta.composer + "\r\n";
                                    textBox_Info.Text += "Title: " +
title + "\r\n";
                                    textBox_Info.Text += "Pages: " +
meta.pages + "\r\n";
                                    textBox_Info.Text += "----------
---------" + "\r\n";
                                    label_Data.Text =
label_Data.Tag.ToString() + outputs.Count.ToString();

                                }));
                            }
                        }
                    }
                }
            }
              }
```

Приложение 7.

Код метода GetSameDataFromChoices класса Searcher:

```
        public void GetSameDataFromChoices()
        {
            var input = inputIndexes.Select(p => Data[p]).ToArray();
            var output = choices.ToArray();

            var coeffsGr = Regression(input,
output).Zip(Enumerable.Range(0, input.Length), (c, p) => new { coeff =
c, index = p });

            var orderedGr = coeffsGr.OrderByDescending(p =>
p.coeff).ToList();

            var dataforCLusteriztion = orderedGr.TakeWhile(p =>
p.coeff > 0).ToList();
            if(dataforCLusteriztion.Count<2)
                dataforCLusteriztion = orderedGr.TakeWhile(p =>
p.coeff >= 0).ToList();
            double[][] observations = new double[Data.GetLength(0)][];
            int j = 0;
            foreach (var data in Data)
            {
                observations[j] = dataforCLusteriztion.Select(p =>
data[p.index]).ToArray<double>();
                j++;
            }
            var changedInput = inputIndexes.Select(p =>
observations[p]).ToArray();

            IndexesOfGeneratedData = Clusterization(observations,
changedInput,output, MaxSearchResults);
            if (EndSearch != null)
                EndSearch();
        }
```

Приложение 8.

Код метода Clusterization класса Searcher:

```
        private List<int> Clusterization(double[][] observations,
double[][] input, bool[] output, int AnswersCount)
        {
            KMeans kmeans = new KMeans(10);

KMeansClusterCollection clusters = kmeans.Learn(observations);
            var labels = clusters.Decide(input);

            var answer = new List<int>();
            Random rnd = new Random();
            while (AnswersCount > 0 && observations.GetLength(0) -
AnswersCount > 0)
            {

                int number = rnd.Next(observations.GetLength(0));
                if (answer.IndexOf(number) >= 0) continue;
                var label = clusters.Decide(observations[number]);
                var indexToLabels = new List<int>();
                for (int i = 0; i < labels.Length; i++)
                    if (labels[i] == label)
                        indexToLabels.Add(i);
                if (indexToLabels.Count() >= 0 &&
indexToLabels.Where(p => output[p]).Count() > 0)
                {
                    answer.Add(number);
                    AnswersCount--;
                }
                else
                {
                    observations = observations.Where(p => p !=
observations[number]).ToArray();
                }
            }
            return answer;
        }
```

Приложение 9.

Код метода GetDoubleData класса SearcherForm:

```
        public double[][] GetDoubleData(List<ParsedScore> Data,
List<string>notInclude)
        {
            var observations = new double[Data.Count][];

            properties = typeof(ParsedScore).GetProperties();
            indexes = new List<int>();
            for (int i = 0; i < properties.Length; i++)
            {
                if (!properties[i].PropertyType.Equals(typeof(string))
&& notInclude.IndexOf(properties[i].Name)<0)
                    indexes.Add(i);
            }
            int j=0;
            foreach (var data in Data)
            {
                observations[j] = indexes.Select(p =>
Convert.ToDouble(ParsedScore.GetPropValue(data as object,
properties[p].Name))).ToArray<double>();
                j++;
            }
            return observations;
        }
```