МИНОБРНАУКИ РОССИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, доцент, к.т.н.
_____ А.Л. Фукс
«_____»_____ 2014 г.

Ромацкий Даниил Борисович

АВТОМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ МУЗЫКАЛЬНЫХ ПРОИЗВЕДЕНИЙ ПО ЖАНРАМ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
на соискание степени магистра
по направлению подготовки
010300 «Фундаментальная информатика и информационные технологии»

Руководитель ВКР
доцент, к.ф.-м.н.
_____ А.И. Попков
подпись
«_____»_____ 2014 г.

Автор работы
студент группы № 1427
_____ Д.Б. Ромацкий
подпись

Электронная версия магистерской диссертации помещена в электронную библиотеку.

Администратор электронной библиотеки факультета
_____ Е.Н. Якунина
подпись

Томск 2014

Реферат

Магистерская диссертация 62 с., 10 рис., 32 источника, 4 табл., 2 прил.

МУЗЫКАЛЬНЫЕ ПРОИЗВЕДЕНИЯ, ЖАНРЫ, ИЗВЛЕЧЕНИЕ ПРИЗНАКОВ, ESSENTIA, КЛАССИФИКАЦИЯ, ДЕРЕВЬЯ ПРИНЯТИЯ РЕШЕНИЙ, RANDOM FOREST, C4.5, C++11

Объект исследования – алгоритмы автоматической классификации музыкальных произведений по жанрам.

Цель работы – исследование задачи автоматической классификации музыкальных произведений по жанрам и создание прототипа программной системы, позволяющей выполнять автоматическую классификацию с использованием алгоритма классификации *Random forest*.

Методы исследования – аналитический, экспериментальный на ЭВМ.

Результаты работы – проведён вычислительный эксперимент по классификации музыкальных произведений, описана процедура подбора входных параметров алгоритмов, приведены полученные результаты, произведён анализ влияния важности отдельных признаков на точность классификации. Разработан и описан прототип программной системы для автоматической классификации музыкальных произведений по жанрам.

Содержание

Введение

Музыкальные жанры – это высокоуровневые описания, создаваемые и используемые людьми для категоризации музыки. Они возникают в процессе сложного взаимодействия культур, исполнителей и рыночных сил и используются для описания сходства между музыкантами или композициями, а также для организации музыкальных коллекций. Слушатели используют жанры для поиска музыки, а также для получения примерного представления о том, понравится ли им то или иное произведение ещё до его прослушивания. В музыкальной индустрии жанры используются как ключевой способ определения целевого рынка.

В настоящее время количество музыкальных файлов в цифровом виде, доступных в сети Интернет, стремительно растёт; появляются различные сервисы, предоставляющие доступ к большому объёму музыки по подписке. Автоматический анализ музыки может стать одной из услуг, с помощью которой владельцы таких сервисов будут привлекать клиентов. Такая возможность может стать полезной и для администраторов сетевых архивов с информацией о музыке, в том числе включающей и жанр. Обычно такие сервисы полагаются на ручную классификацию, являющуюся медленной и громоздкой.

Целью данной работы является исследование задачи автоматической классификации музыкальных произведений по жанрам и создание прототипа программной системы, позволяющей выполнять автоматическую классификацию с использованием алгоритма классификации *Random forest*, который будет описан в главе 3. Несмотря на ряд достоинств этого алгоритма, до сих пор не было проведено полноценного исследования его применимости к задачи автоматической классификации музыкальных произведений по жанрам.

Для достижения поставленной цели требуется решить следующие задачи:

1. Описать постановку задачи, произвести обзор видов признаков, используемых для классификации, а также обзор различных алгоритмов классификации.
2. Выбрать и описать средство для извлечения значений признаков, а также определить сам набор признаков и описать алгоритмы, используемые для извлечения признаков.
3. Описать алгоритмы *Random forest* и *C4.5* (используется как составная часть *Random forest*).
4. Выбрать набор данных, на котором будет осуществляться классификация, и провести

вычислительный эксперимент по классификации музыкальных произведений, подобрать входные параметры алгоритмов, проанализировать полученные результаты.

5. Разработать прототип программной системы (приложения) для автоматической классификации музыкальных произведений по жанрам, описать эту систему, используя средства языка *UML*.

1 Обзор предметной области

1.1 Терминология и постановка задачи

В общем виде задача классификации определяется следующим образом: пусть X – множество описаний объектов (примеров), Y – конечное множество номеров (имён, меток) классов. Существует неизвестная целевая зависимость – отображение $y^*: X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_1, y_1), ..., (x_m, y_m)\}$. Требуется построить алгоритм $a: X \to Y$, способный классифицировать произвольный объект $x \in X$.

В задаче автоматический классификации музыкальных произведений по жанрам каждый пример представляет собой n-мерный вектор признаков, где признак (*feature*) – это термин, часто используемый в теории распознавания образов, в данном контексте означающий порцию информации, которая может быть извлечена из аудиозаписи и затем использована для её описания или классификации [17]. Иногда признаки также называют дескрипторами. В данной работе под аудиозаписями будут пониматься цифровые аудиофайлы в формате *mp3*, содержащие запись музыкального произведения или его часть. При этом такое ограничение не является обязательным, например, в качестве исходных данных могут использоваться такие форматы данных, как *MIDI* или *MusicXML* [6]. Метки классов представляют собой названия жанров.

Общую схему решения задачи классификации музыкальных произведений по жанрам можно условно разделить на пять этапов, которые будут описаны далее:

- определение набора признаков;
- выделение признаков из аудиозаписей;
- выбор алгоритма классификации;
- выбор таксономии жанров – иерархического множества категорий, отображаемого на музыкальную коллекцию;
- классификация выбранным алгоритмом.

К сожалению, задача точной идентификации жанра является сложной как для человека, так и для компьютера. Зачастую не существует общепринятого понимания того,

какие характеристики имеет тот или иной жанр, какие вообще жанры следует использовать, и как они соотносятся друг с другом. Дополнительной проблемой является то, что разные люди по-разному понимают жанры, что приводит к несоответствиям. Но хотя разделение музыки по жанрам является в немалой степени субъективным, для описания того или иного жанра можно использовать критерии, связанные с фактурой, инструментовкой и ритмической структурой музыки. Из всего этого следуют два важных вопроса автоматической классификации по жанрам: какие музыкальные признаки использовать для классификации, и как разработать таксономию жанров, в которой будут классифицироваться записи [17].

Наличие надёжных проверочных данных также является ключевым требованием для эффективного обучения классификаторов музыкальных произведений по жанрам. Так как между аннотаторами-людьми (производящими классификацию) может быть достигнуто только ограниченное согласие, то это ограничение неизбежно задаёт верхнюю границу эффективности автоматической классификации жанров. Отдельные люди могут не только по-разному классифицировать отдельную аудиозапись, но и иметь представление о большем или меньшем набор жанров, из которых можно выбирать. Очень малое количество жанров имеет чёткое определение, а имеющаяся информация зачастую неоднозначна – некоторые жанры значительно перекрываются между собой, и отдельные записи могут в разной степени одновременно принадлежать к разным жанрам. Между жанрами часто существуют сложные взаимоотношения, некоторые жанры более широкие, в то время как другие – более узкие.

Исследования в области автоматической классификации жанров могут помочь в понимании того, как люди создают музыкальные жанры, какие механизмы они используют для классификации музыки, и каким образом осознаются различия между жанрами. Механизмы, используемые людьми при классификации жанров, слабо изучены, и создание автоматического классификатора могло бы дать ценную информацию в этой области.

Техники, используемые в автоматической классификации жанров, могут быть распространены и на другие виды анализа музыки на основе только её содержания (класс задач музыкального информационного поиска), а также других задач классификации, таких как измерение степени сходства. Такие системы могут быть использованы для классификации музыки, например, на основе стиля исполнения, географического происхождения или исторического периода; для поиска новой музыки, которая могла бы понравиться пользователю, на основе содержимого его коллекции.

1.2 Обзор видов признаков для классификации

Основой систем автоматического анализа аудиосигналов любого типа является извлечение векторов признаков. Множество работ посвящено именно выбору признаков для классификации. Основополагающими в области классификации по жанрам являются работа Дж. Цанетакиса (G. Tzanetakis) и П. Кука (P. Cook) [31], которыми были взяты за основу работы по распознаванию речи, а также неречевых сигналов, таких как звуки музыкальных инструментов. Ими было предложено три набора признаков, служащих для представления тембра, ритма и высоты звука. Набор тембральных признаков был заимствован из областей распознавания речи и звуков в целом, но два других набора были разработаны именно для представления аспектов музыки (ритма и гармонии).

В литературе тембр определяется как характеристика звука, позволяющая двум звукам с одинаковой высотой и громкостью звучать по-разному [6]. Для извлечения признаков, описывающих тембр, используется спектральное распределение сигнала, хотя некоторые из таких признаков вычисляются во временной области. Тембральные признаки часто называют *низкоуровневыми*, так как обычно они вычисляются на коротких отрезках сигнала (от 10 до 60 мс). В работе [22] приводится исчерпывающий список признаков, использующихся для описания тембра инструментов. Большинство из этих признаков можно использовать и в контексте распознавания музыкального жанра, хотя некоторые признаки больше подходят для описания монофонических инструментов, нежели их полифонических смесей.

В работе [6] приводятся основные низкоуровневые признаки, используемые в приложениях для определения жанров:

- Временны́е признаки (*temporal features*) – вычисляются из кадров звукового сигнала (частота переходов через ноль, коэффициенты линейного предсказания и т. д.).
- Энергетические признаки (*energy features*) – признаки, относящиеся к энергии сигнала (среднеквадратичная энергия кадра сигнала, энергия гармонической составляющей спектра сигнала, энергия шумовой части спектра и т. д.).
- Признаки формы спектра (*spectral shape features*) – признаки, описывающие форму спектра звукового кадра (центроид, дисперсия, коэффициенты асимметрии и эксцесса, мел-частотные кепстральные коэффициенты (MFCC) и т.д.).
- Перцептивные признаки (*perceptual features*) – признаки, относящиеся к восприятию, вычисляются с использованием модели человеческого восприятия звука

(относительная громкость, чёткость и т. д.)

Преобразования значений признаков, такие как производные первого или второго порядков, являются распространенным методом создания новых признаков или повышения размерности векторов признаков.

В контексте задачи классификации значения тембральных признаков часто обобщаются (например, в [13]) с помощью применения статистик низкого порядка к более крупным окнам, называемым текстурными окнами (*texture windows*). Такое обобщение не только уменьшает вычислительные затраты, но и ближе к человеческому восприятию, так как короткие отрезки сигнала, используемые для вычисления признаков, являются недостаточно длинными для восприятия человеком (время восприятия человеком составляет около 250 мс или более [13]). Влияние размера текстурного окна на точность классификации также исследовалось в [31]. Было показано, что использование окна значительно повышает точность классификации в сравнении с прямым использованием анализируемых отрезков.

Под признаками, описывающими высоту сигнала, подразумеваются признаки, описывающие гармонию и мелодию аудиозаписи музыкального произведения. Гармония может быть определена как использование и изучение слаженности звуков и аккордов (фактических или подразумеваемых). Мелодия же, напротив, представляет собой последовательность звуков различной высоты, воспринимаемых как единое целое. Гармонию иногда называют «горизонтальным» измерением музыки, а мелодию – «вертикальным» измерением. В работе [6] говорится о том, что гармония и мелодия более надёжно описываются низкоуровневыми атрибутами, нежели нотами или аккордами. Попытка применить такого рода признаки для моделирования жанров аудиосигналов предпринималась в [31], однако более интенсивно они используются в контексте семантической сегментации и обобщения музыки. Обзор методик описания и извлечения мелодии приведён в [19].

Основной идеей является, как и в большинстве анализаторов мелодии или гармонии, использование функции, описывающей распределение высот на коротком участке композиции; разница состоит в том, что такие высокоуровневые признаки как тональность, основная частота, последовательность аккордов и др. не используются, вместо этого на основе функции вычисляется набор значений таких признаков, как амплитуды и расположения основных пиков, величины интервалов между пиками и любые другие статистические описатели распределения функции высотного содержимого.

Точного определения ритма не существует. Многие авторы рассматривают ритм как

идею временно́й регулярности. Фактически, воспринимаемая регулярность является отличительной чертой ритма и отличает его от не-ритма. В более общем смысле, слово «ритм» может использоваться для обозначения всех временны́х аспектов музыкального произведения. Интуитивно ясно, что ритм следует учитывать при различении жанров, так как ритмический рисунок зачастую является важной особенностью того или иного жанра [6].

Обзор систем автоматического описания ритма приведён в [11]. Такие системы могут быть ориентированы на использование в различных приложениях: определение темпа, отслеживание ударов, определение метра (метр – рисунок равномерного чередования сильных и слабых долей) и т. д. Современные системы определения ритма все ещё имеют ряд недостатков, поэтому в системах автоматической классификации по жанрам чаще используется низкоуровневый подход. Используя тот же подход, что и для низкоуровневых высотных признаков, можно извлечь признаки с помощью функции, оценивающей периодичности в диапазоне воспринимаемых темпов (обычно от 40 до 200 ударов в минуту).

Описанные выше признаки могут извлекаться из всего аудиосигнала. Однако во многих задачах классификации используется небольшой аудиосегмент, предполагая, что он может содержать достаточное количество информации для описания всего произведения, так как во многих музыкальных жанрах наблюдаются повторы музыкальной структуры. Кроме того, при использовании небольшой части сигнала можно значительно уменьшить вычислительные затраты. Часто используется один аудиосегмент на каждое произведение – обычно это 30-секундный отрезок, взятый через 30 секунд после начала композиции (для того, чтобы не рассматривать различные вступления, которые могут отличаться от произведения в целом) [6].

В работе [13] приводится другая классификация типов признаков – по временно́й шкале, однако в целом распределение признаков по категориям сохраняется:

- кратковременные – размер кадра 30 мс, содержат информацию о тембре;
- средневременные – размер кадра 740 мс, содержат временну́ю информацию, такую как модуляции;
- долговременные – размер кадра 9.62 с, содержат структурную информацию, такую как ритм.

Техники извлечения высокоуровневых признаков из полифонических аудиосигналов без ограничений ещё не достигли совершенства. Именно поэтому большинство подходов

нацелено на моделирование тембра на основе комбинаций низкоуровневых признаков. Тембр содержит достаточно информации для грубого определения музыкальных жанров; так, участники исследования, описанного в [23], не имеющие либо имеющие небольшую подготовку, были способны выполнять правильную классификацию музыку (по 10 жанрам) в 53% случаев после прослушивания всего лишь 250 миллисекунд, и в 72% случаев после прослушивания 3 секунд записи. Это позволяет предположить, что для определения жанров не требуется высокоуровневого понимания музыки, так как 250 миллисекунд и, в меньшей степени, 3 секунд недостаточно для определения музыкальной структуры. Однако в [5] приводится более пессимистичная точка зрения. С помощью современного алгоритма определения сходства тембра было проведено исследование на базе данных из 20000 произведений в 18 жанрах. Результаты показали, что между тембром и жанрами существует лишь небольшая корреляция, вследствие чего можно предположить, что схемы классификации, основанные только на тембре, ограничены по своей природе. Было предположено, что подобные схемы классификации плохо масштабируются как по количеству произведений, так и по количеству жанров. Кроме того, авторы предположили, что звуковой сигнал может содержать недостаточно информации для описания жанра музыкального произведения, и поэтому предложили принимать во внимание культурные особенности, которые можно добыть в сети Интернет, извлекая релевантные ключевые слова, связанные с музыкальными произведениями. В самом деле, когда кто-либо пытается определить жанр только на основе аудиозаписи, то он делает предположение, что жанр является таким же внутренним атрибутом произведения, как, например, его темп, что находится под вопросом [6].

1.3 Обзор алгоритмов классификации

Существует три основных класса алгоритмов классификации:

- экспертные системы;
- алгоритмы обучения без учителя
- алгоритмы обучения с учителем.

Экспертные системы определяют наборы правил в явном виде. Для задачи классификации по жанрам это было бы аналогично перечислению набора правил, которые бы точно и однозначно могли описать жанр. По имеющимся данным, для определения

музыкальных жанров пока не было предложено моделей, основанных на экспертных системах [6]. Такой подход, если он вообще возможен, не подходит для задачи классификации по жанрам из-за сложности как самой задачи, так и объективного описания отдельных поджанров. Кроме того, этот подход предполагает получение надёжных высокоуровневых описаний из аудиосигнала, а, как было сказано выше, соответствующие алгоритмы являются несовершенными. Экспертные системы хотя и содержат глубокие знания о своём предмете, являются дорогими в реализации и поддержке. С ростом числа правил, создаваемых вручную, могут проявляться непредвиденные побочные эффекты. В последнее время все больший интерес вызывает подход на основе машинного обучения, доминирующий в родственных областях, таких как автоматическое распознавание речи или лиц.

Хотя некоторые подходы стремятся классифицировать музыку в заданной таксономии жанров, другая точка зрения заключается в кластеризации данных неконтролируемым способом (без учителя) таким образом, что классификация сама возникает из данных на основе объективных мер сходства. Преимущество состоит в том, что такой подход позволяет избежать ограничений фиксированной таксономии, приводящих к неоднозначностям. Кроме того, некоторые произведения могут просто не вписываться в заданную таксономию. При обучении без учителя аудиозапись представляется набором признаков, а для сравнения произведений между собой используется мера схожести. Наиболее простым способом измерения расстояния между двумя векторами является, например, евклидово расстояние или косинусное расстояние. Однако такие метрики имеют смысл, только если вектора признаков не изменяются во времени. Иначе два субъективно схожих произведения могут оказаться удалёнными друг от друга по результатам измерений, если схожие значения признаков сдвинуты во времени. Для построения стационарного представления временных рядов векторов признаков обычно строится статистическая модель распределения признаков, а расстояние затем используется для непосредственного сравнения моделей.

Типовыми моделями являются гауссовские модели и смеси гауссовских моделей (*GMMs*). Естественным способом оценки удалённости двух вероятностных распределений является расстояние Кульбака-Лейблера, или относительная энтропия, однако для смесей гауссовских моделей этот способ не подходит. Альтернативные меры включают сэмплинг (*sampling*), *EMD*-расстояние (*Earth's Mover distance*) и аппроксимацию асимптотического правдоподобия (*Asymptotic Likelihood Approximation*) [6].

В отличие от большинства классических задач распознавания образов,

классифицируемые данные могут представлять собой временные ряды, поэтому можно использовать скрытые марковские модели (*HMM*) для описания отношений между признаками с течением времени [6].

Для кластеризации можно использовать следующие алгоритмы [6]:

- Метод *k*-средних (*k-means*) является одним из простейших и наиболее популярных алгоритмов кластеризации. Он позволяет разбить множество векторов на *K* непересекающихся подмножеств. Одним из недостатков является необходимость заранее знать количество кластеров (*K*).
- Алгоритм агломеративной иерархической кластеризации начинает с *N* кластеров-одиночек (где *N* – число элементов в базе данных), а затем формирует кластеры последовательным слиянием.
- Самоорганизующиеся карты (*SOM*) используются для кластеризации данных и их организации в двумерном пространстве таким образом, что схожие вектора признаков группируются друг с другом. Самоорганизующиеся карты – это нейронные сети с обучением без учителя, отображающие многомерные входные данные на выходные пространства более низкой размерности, сохраняя при этом топологические отношения между элементами входных данных, насколько это возможно.

Обучение без учителя хорошо подходит для реализации систем, определяющих сходство музыкальных произведений в целом, но не конкретно для задачи классификации музыкальных жанров, так как в этом случае получаемые категории могут быть несодержательными для пользователя. Хотя эти категории могут быть более точными, чем жанры, определяемые людьми, система, использующая свой собственный набор жанровых категорий, будет иметь ограниченную полезность для пользователей, желающих использовать знакомые и понятные для них жанры.

Методы обучения с учителем изучены более подробно и пытаются отобразить базу музыкальных произведений на заданную таксономию жанров с использованием алгоритмов машинного обучения. Система сначала обучается на каких-то данных, размеченных вручную, а затем используется для классификации неразмеченных данных. При обучении с учителем, в

отличие от экспертных систем, не требуется явно описывать музыкальные жанры: классификатор сам попытается автоматически сформировать связь между признаками обучающего множества и соответствующими категориями.

В [6] приводится список кратких описаний алгоритмов обучения с учителем с примерами использования в исследованиях, не претендующий на полноту, но содержащий алгоритмы, используемые именно в контексте классификации музыкальных произведений по жанрам:

- Метод *k* ближайших соседей (*kNN*) – для заданного вектора признаков из целевого множества (для элементов которого необходимо определить жанр) выбирается *k* ближайших векторов из обучающего множества (в соответствии с какой-то мерой расстояния), и целевому вектору присваивается метка класса с наибольшим числом вхождений из *k* соседей;
- Смеси гауссовских моделей (*GMMs*) – для каждого класса предполагается существование функции плотности вероятности, выражаемой в виде смеси набора многомерных нормальных (гауссовских) распределений. Для оценки параметров каждого компонента используется итеративный алгоритм максимизации ожидания (*EM-алгоритм*);
- Скрытые марковские модели (*HMM*) – широко используются в распознавании речи из-за возможности обработки временных рядов данных. Скрытая марковская модель представляет собой дважды стохастический процесс, состоящий из двух случайных процессов: основного и скрытого (ненаблюдаемого);
- Линейный дискриминантный анализ (*LDA*) – основной идеей является нахождение линейного преобразования, наилучшим образом разделяющего классы, и классификация в трансформированном пространстве на основе какой-то метрики, например, расстояния Евклида;
- Метод опорных векторов (*SVM*) – набор схожих алгоритмов вида «обучение с учителем». Основная идея метода опорных векторов – перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше

разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора;

- Искусственная нейронная сеть (*ANN*) – состоит из большого количества обрабатывающих элементов (нейронов), совместно решающих определённые задачи. В задачах распознавания образов чаще всего используется многослойный перцептрон.

Также для решения задачи автоматической классификации музыкальных произведений по жанрам используются алгоритмы, основанные на деревьях принятия решения. Дерево принятия решений – способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. Дерево состоит из листьев, указывающих на класс, и узлов. Для классификации объектов, не вошедших в обучающее множество, осуществляется поиск, начинающийся с корня, и продолжающийся, пока не будет обнаружен класс, соответствующий объекту. Так, в [4] и [8] использовался алгоритм *C4.5* (его открытая реализация *J48* на языке *Java*). В [12] использовался алгоритм *TILDE*, являющийся расширением *C4.5*, а также алгоритм *Random forest* («случайный лес»), заключающийся в использовании ансамбля деревьев принятия решений. *Random forest* в том числе использовался в [7] и [14].

1.4 Основные проблемы классификации по жанрам

С точки зрения современного программного обеспечения, ни одна система не достигла достаточно высоких показателей для реального применения. Например, наиболее высокий процент успешной классификации на *MIREX 2013* (ежегодное соревнование по различным областям музыкального информационного поиска, в том числе классификации произведений по жанрам) составил 76% при классификации по 10 жанрам [20]. В [18] говорится о том, что исследования в области автоматической классификации в последние годы дают все меньший прирост эффективности, поэтому ряд исследователей предлагает отказаться от классификации по жанрам и переключиться на более общие исследования схожести. В работе приводится ряд контраргументов, подчёркивающих важность продолжения исследований в области классификации музыкальных жанров, обсуждаются стратегии преодоления текущих ограничений эффективности классификации, а также выполнен обзор исследований, относящихся к задаче классификации жанров, в таких областях как психология и музыковедение. Для преодоления текущих ограничений

автоматической классификации жанров предлагается ряд модификаций существующих алгоритмов:

- Следует комбинировать низкоуровневые, высокоуровневые и культурные признаки.
- Должна быть возможность назначить каждой аудиозаписи более одного жанра, а жанры должны быть взвешены.
- Следует использовать большое число жанров различной степени широты, и эти жанры должны быть организованы в онтологические структуры.
- Ошибки классификации должны отражать степень сходства между различными жанрами.
- Должна быть возможность назначать различные жанры отдельным частям аудиозаписи.
- Следует экспериментировать с признаками и классификаторами, способными отслеживать изменение аудиозаписи во времени.
- Следует использовать методы уменьшения размерности, сохраняющие исходное содержание признаков.
- Следует принять во внимание знания из области психологии и музыковедении и на их основе проводить эмпирические исследования.

Если надёжные источники контрольных данных и существуют, то они малочисленны и часто содержат избыточное либо недостаточное количество информации. К тому же, классификация жанров в них обычно производится по исполнителю или альбому, а не по отдельной аудиозаписи. Источники, предоставляющие классификацию индивидуальных записей, такие как *Gracenote CDDB* или метаданные из *ID3*-тегов *mp3*-файлов, как правило, являются ненадёжными. Также обычно отсутствует описание того, каким образом производилась классификация, и представляется довольно сомнительным то, что были приложены серьёзные усилия для вдумчивой, методичной и последовательной аннотации.

Компетентность и затраты времени, требуемые для ручной классификации записей, создают серьёзные препятствия для получения качественных контрольных данных. Это особенно заметно, когда во избежание переобучения требуются большие наборы данных. Ещё больше усложняет ситуацию то, что не только постоянно появляются новые жанры, но со временем изменяется понимание и уже существующих, что может потребовать повторного

обучения систем и переаннотации контрольных данных. Необходимость большой обучающей выборки также имеет значение с точки зрения машинного обучения. Мощные алгоритмы обучения, такие как метод опорных векторов или *AdaBoost* эффективно моделируют сложные пространства жанров, но многие из таких алгоритмов плохо масштабируются.

2 Признаки для классификации

2.1 Библиотека Essentia

В настоящее время для извлечения значений признаков существуют различные готовые сторонние решения. В данной работе для этой цели будет использоваться библиотека *Essentia* [10], имеющая открытый исходным код на языке *C++* и предназначенная для анализа аудиозаписей и музыкального информационного поиска на основе аудиоданных. Она содержит обширную коллекцию алгоритмов, включающую алгоритмы, реализующие функциональность ввода/вывода аудиофайлов, стандартные алгоритмы обработки цифровых сигналов, алгоритмы для получения статистических описаний данных и большой набор спектральных, временны́х, тональных и высокоуровневых музыкальных дескрипторов.

Библиотека *Essentia* имеет ряд преимуществ в сравнении с аналогичными решениями, такими как *libxtract* [15], *MARSYAS* [16], *Sonic Annotator* [27], *Yaafe* [32] и др.:

- больше возможностей для обработки цифровых аудиосигналов, в том числе применимых конкретно к задаче классификации произведений по жанрам (*Essentia* включает в себя практически все возможности перечисленных библиотек);
- активная разработка библиотеки;
- удобная архитектура, в том числе возможность расширения библиотеки собственными алгоритмами;
- подробная документация и примеры использования.

Кроме того, данная библиотека ещё не использовалась в опубликованных научных работах (согласно веб-сайту библиотеки [10]), поэтому имеет смысл изучить её пригодность к использованию в научных исследованиях.

2.2 Архитектура Essentia

Essentia создавалась как библиотека блоков обработки сигналов. Каждый блок обработки называется алгоритмом и имеет три различных типа атрибутов:

- входы;
- выходы;

- параметры.

Алгоритмы представляют собой экземпляры подклассов класса *essentia::Algorithm*. Алгоритмы могут иметь произвольное число атрибутов каждого из перечисленных видов (в том числе ноль). В общем виде последовательность выполняемых действий при работе с алгоритмами выглядит следующим образом:

1. создаётся (инстанцируется) объект алгоритма;
2. объект конфигурируется желаемыми параметрами;
3. к объекту подключаются входы и выходы;
4. пункты 2 и/или 3 повторяются желаемое количество раз.

Класс *essentia::Algorithm*, в свою очередь, является подклассом *essentia::Configurable*, который отвечает за работу с параметрами, входами и выходами. От алгоритма требуется реализация следующих возможностей:

- возможность получать доступ к входам и выходам по их именам;
- вызов метода *compute()* для получения результатов, то есть применение конкретного алгоритма к заданным входам и получение соответствующих результатов;
- возможность сбросить внутреннее состояние алгоритма, если он его поддерживает, с помощью метода *reset()*.

Essentia поддерживает два режима работы: стандартный и потоковый. Стандартный режим является императивным: после создания объекта алгоритма и установки входов и выходов метод *compute()* вызывается вручную (столько раз, сколько это требуется). Так, например, для извлечения значений мел-частотных кепстральных коэффициентов (*MFCC*) в стандартном режиме необходимо выполнить несколько алгоритмов подряд (то есть создать, сконфигурировать и вызвать метод *compute()*): прочитать содержимое аудиофайла, разрезать его на отдельные кадры, затем в цикле по всем кадрам применить к каждому кадру оконную функцию, быстрое преобразование Фурье и после этого вычислить значения *MFCC*. Выход каждого алгоритма в явном виде подаётся на вход следующему алгоритму. Такой подход требует больше усилий для реализации, однако даёт больший контроль над процессом, так как предоставляет доступ к промежуточным данным (выходам каждого из алгоритмов).

В потоковом режиме не требуется вручную вызывать метод *compute()*. Вместо этого создаётся и запускается сеть связанных алгоритмов, а заботу о запуске отдельных алгоритмов

берет на себя встроенный планировщик. Этот подход проще в реализации, но разработчик получает доступ к выходным данным только последнего алгоритма из выполняемой последовательности. Потоковый режим можно рассматривать как режим потока данных: данные автоматически переходят от одного алгоритма к другому в соответствии с заданной сетью. В данной работе используется именно потоковый режим, так как доступ к значениям промежуточных данных не требуется, кроме того, сокращается объем программного кода, требуемого для извлечения значений признаков.

Для хранения данных в *Essentia* используется потокобезопасная структура, называемая пулом (объект класса *essentia::Pool*), которая может рассматриваться в качестве кэша. В процессе обработки входных данных часто образуется множество значений, которые будут обработаны в будущем (возможно, другими алгоритмами), и в этом случае пул является идеальным механизмом хранения данных. Пул хранит значения, в качестве идентификатора используя строку, которая может быть разделена точками для обозначения пространств имён. В качестве значений могут выступать как отдельные числа или строки, так и их массивы (в том числе многомерные).

Essentia предлагает следующие виды алгоритмов:

- ввод/вывод аудиофайлов – различные способы для чтения и записи аудиофайлов;
- стандартные алгоритмы обработки сигналов – ряд алгоритмов, являющихся стандартными для большинства библиотек цифровой обработки сигналов, таких как оконные функции, быстрое преобразование Фурье, дискретное косинусное преобразование и др.;
- статистические методы – алгоритмы, высчитывающие статистические данные для массива значений, либо производящие какую-либо агрегацию, например, алгоритмы, вычисляющие среднее, центральные моменты, различные параметры распределения вероятностей и др.;
- фильтры – различные фильтры аудиосигналов, например, фильтр равной громкости (*equal-loudness filter*);
- спектральные дескрипторы;
- дескрипторы временно́й области;
- дескрипторы ритма;
- дескрипторы звуковых эффектов – предназначены для использования на коротких

звуках вместо полноценных музыкальных аудиозаписей;

- другие высокоуровневые дескрипторы.

Алгоритмы в *Essentia* могут быть композитными (состоять из нескольких алгоритмов), поэтому разработчиками библиотеки был создан набор алгоритмов-экстракторов, готовых к использованию. Они позволяют извлекать из аудиозаписей такие данные, как громкость, различные низкоуровневые признаки, тональную и ритмическую информацию и др.

2.3 Используемые алгоритмы

Извлекаемые признаки помещаются в пул (экземпляр класса *essentia::Pool*), а с помощью алгоритма ***YamlOutput*** создаётся представление пула в формате *YAML* [29], которое записывается в текстовый файл. Такой подход даёт возможность повторного использования значений признаков для классификации одних и тех же данных с различными параметрами классификации или с применением различных алгоритмов. Для считывания данных используется алгоритм ***YamlReader***, который десериализует *YAML*-файл в объект пула.

Алгоритмы, извлекающие спектральные признаки, работают с амплитудным спектром сигнала (далее просто «спектр»). Спектр представляет собой массив вещественных чисел и вычисляется следующим образом:

- С помощью алгоритма ***EasyLoader*** считываются данные аудиофайла. Данный алгоритм принимает на вход имя аудиофайла и выдаёт необработанные звуковые данные – ряд отсчётов, взятых через равные промежутки времени. Отсчёты представляют собой вещественные числа. Стандартная частота дискретизации для *mp3*-файлов составляет 44100 Гц, соответственно, амплитуда сигнала измеряется 44100 раз в течение одной секунды. Звуковые каналы (если звук был многоканальным) при этом сводятся в один. Некоторые алгоритмы работают напрямую с этим представлением звукового сигнала. Также с помощью алгоритма ***MetadataReader*** считываются *ID3*-теги (метаданные) *mp3*-файлов (с целью более простой визуальной идентификации *YAML*-файлов разработчиком либо пользователем системы).
- С помощью алгоритма ***FrameCutter*** полученный массив значений разрезается на кадры. При каждом вызове алгоритм возвращает кадр заданного размера, после чего переходит на заданное число отсчётов вперёд. В примерах кода *Essentia* для

извлечения низкоуровневых признаков используется кадр из 2048 отсчётов, величина скачка составляет 1024 отсчёта, в данной работе используются эти же значения. При частоте дискретизации в 44100 Гц кадр в 2048 отсчётов имеет длительность в 2048/44100 ≈ 0.046 с = 46 мс. Данное значение попадает в интервал от 10 до 60 мс, который обычно используется для извлечения низкоуровневых признаков, как было показано в разделе 1.2.

- С помощью алгоритма ***Windowing*** к каждому кадру применяется оконная функция. Для извлечения низкоуровневых признаков используется окно Блэкмана – Харриса, есть возможность использовать и другие распространённые оконные функции.
- С помощью алгоритма ***Spectrum*** для каждого кадра вычисляется его амплитудный спектр.

Значения признаков, полученные для спектров отдельных кадров, затем усредняются с помощью алгоритма ***PoolAggregator*** – вычисляются математическое ожидание и дисперсия, за исключением случаев, когда вычисляется единственное значение признака для каждой аудиозаписи (будет явно указано в описании алгоритмов). Эти значения и используются при классификации. Если признак является многомерным, то есть представляет собой *n*-мерный вектор значений, то при классификации он рассматривается как *n* отдельных признаков.

Далее будут представлены описания алгоритмов, использующихся в данной работе для извлечения значений признаков. Данные алгоритмы выбирались исходя из популярности соответствующих признаков в литературе. Также были выбраны некоторые алгоритмы, представленные в примерах, поставляемых вместе с библиотекой *Essentia* – в основном, это алгоритмы для извлечения спектральных признаков, имеющие низкую вычислительную сложность. В дальнейшем к признакам будет применена процедура уменьшения размерности, которая позволит избавиться от признаков, слабо влияющих или не влияющих на качество классификации (а, возможно, и ухудшающих его). Более подробная информация об используемых, а также остальных алгоритмах со ссылками на источники и дополнительную литературу может быть найдена в разделе документации на веб-сайте *Essentia* [10].

Список алгоритмов:

1. ***ZeroCrossingRate***

   Алгоритм вычисляет частоту переходов аудиосигнала через ноль (*zero-crossing rate*), то есть число изменений знака между идущими подряд значениями сигнала, делённое

на общее число значений. Частота переходов через ноль является мерой шумности – шумные сигналы, как правило, имеют более высокие значения данной величины. Чтобы избежать малых колебаний значения величины около нуля, вызванных шумом, задаётся пороговое значение.

2. ***MFCC***

   Алгоритм вычисляет мел-частотные кепстральные коэффициенты (*mel-frequency cepstral coefficients*, *MFCC*). Данные признаки широко применяются в задачах распознавания речи, однако являются популярными и в задаче классификации музыкальных произведений по жанрам. Они основаны на двух ключевых понятиях: кепстр и мел-шкала. Кепстр (*cepstrum*) – это результат дискретного косинусного преобразования от логарифма амплитудного спектра сигнала. Мел-шкала моделирует частотную чувствительность человеческого слуха. Специалистами по психоакустике было установлено, что изменение частоты в два раза в диапазоне низких и высоких частот человек воспринимает по-разному. В частотной полосе до 1000 Гц субъективное восприятие удвоения частоты совпадает с реальным увеличением частоты в два раза, поэтому до 1000 Гц мел-шкала близка к линейной. Для частот выше 1000 Гц мел-шкала является логарифмической. Мел-частотные кепстральные коэффициенты – это значения кепстра, распределенные по мел-шкале с использованием банка фильтров [2].

   Стандартной реализации алгоритма не существует, поэтому по умолчанию используется MFCC-FB40:

   - используется банк фильтров из 40 полос от 0 до 11000 Гц;
   - в каждой полосе вычисляется значение логарифма от спектральной энергии;
   - над 40 полосами выполняется дискретное косинусное преобразование для получения 13 мел-коэффициентов.

   Для вычисления энергии в мел-полосах используется алгоритм ***MelBands***, применяющий банк фильтров к спектру сигнала. Дискретное косинусное преобразование реализуется алгоритмом ***DCT***.

3. ***DistributionShape***

   Алгоритм извлекает значения дисперсии, коэффициентов асимметрии и эксцесса (остроты пика распределения) из массива центральных моментов распределения.

Массив центральных моментов вычисляется с помощью алгоритма ***CentralMoments***, который для заданного массива значений (в данном случае – спектра аудиосигнала) вычисляет центральные моменты от 0 до 4 порядка включительно. Данные признаки являются хорошими описателями формы распределения.

4. ***Energy***

   Алгоритм вычисляет энергию входного массива значений (в данном случае – спектра аудиосигнала), которая равна сумме квадратов значений элементов массива.

5. ***EnergyBand***

   Аналогично алгоритму ***Energy***, данный алгоритм вычисляет энергию спектра аудиосигнала, но в заданной частотной полосе. Частотная полоса задаётся двумя значениями – минимальной и максимальной частотой. В данной работе значения вычисляются для четырёх полос: от 20 до 150 Гц, от 150 до 800 Гц, от 800 до 4000 Гц и от 4000 до 20000 Гц (значения граничных частот взяты из примеров кода *Essentia*).

6. ***HFC***

   Алгоритм вычисляет меру высокочастотного содержимого (*high frequency content*, *HFC*) для спектра аудиосигнала. Если *X(k)* – дискретный спектр с *N* значениями, то по умолчанию мера высокочастотного содержимого вычисляется по формуле: $\sum_{i=1}^{N} i \cdot \left| X(i)^2 \right|$. В *Essentia* также имеется возможность использовать ещё две незначительно отличающиеся формулы.

7. ***RMS***

   Алгоритм вычисляет среднеквадратичное значение входного массива значений (в данном случае – спектра аудиосигнала).

8. ***Flux***

   Алгоритм вычисляет спектральный поток (*spectral flux*) для заданного спектра, который определяется как квадрат разности между нормализованными значениями двух последовательных кадров амплитудного спектра: $F_t = \sum_{n=1}^{N} (N_t[n] - N_{t-1}[n])^2$, где $N_t[n]$ и $N_{t-1}[n]$ – нормализованные значения спектра в текущем кадре *t* и предыдущем кадре *t-1*, соответственно [31]. Спектральный поток является мерой локального

изменения спектра. По умолчанию в качестве нормы используется евклидова норма (*L2-norm*).

9. ***RollOff***

   Алгоритм вычисляет спектральный спад (*spectral rolloff*) для заданного спектра, который определяется как частота, ниже которой сосредоточен заданный процент (срез) общей энергии спектра. По умолчанию используется значение 85%. Частота спектрального спада может использоваться для отделения шума от значимого содержимого (шумовая часть находится выше частоты спада).

10. ***StrongPeak***

    Алгоритм вычисляет значение «сильного пика» (*strong peak*) для заданного спектра, которое определяется как отношение между значением максимального пика спектра и «шириной» этого пика над порогом, равным половине амплитуды максимального пика. Это отношение показывает, есть ли в спектре «ярко выраженный» максимальный пик, то есть, чем у́же и выше максимум спектра, тем больше значение отношения.

11. ***Crest***

    Алгоритм вычисляет значение гребня (*crest*) входного массива значений, которое определяется как отношение между максимальным и средним арифметическим значениями массива. Для вычисления входного массива используется алгоритм ***BarkBands***, который вычисляет спектральную энергию в заданных частотных полосах. По умолчанию число полос равно 27, и эти полосы представляют собой экстраполяцию барк-шкалы, состоящей из 24 полос. Границы этих полос составляют 0, 50, 100, 150, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, 15500, 20500 и 27000 Гц. Барк, как и мел, является психофизической единицей высоты звука.

12. ***FlatnessDB***

    Алгоритм вычисляет значение плоскости (*flatness*) входного массива значений (в данном случае – спектра аудиосигнала), которое определяется как отношение между средним геометрическим и средним арифметическим значениями массива, выраженное в децибелах. Для вычисления входного массива используется алгоритм ***BarkBands***.

13. ***SpectralComplexity***

Алгоритм вычисляет значение спектральной сложности (*spectral complexity*) для заданного спектра, которое основано на количестве пиков в спектре. Пики спектра вычисляются с помощью алгоритма ***SpectralPeaks***, который вычисляет частоты и амплитуды пиков в заданном спектре.

14. ***PitchSalience***

Алгоритм вычисляет значение рельефности высоты (*pitch salience*) для заданного спектра, которое определяется как отношение наибольшего значения автокорреляции спектра к несдвинутому значению автокорреляции. Вектор автокорреляции вычисляется с помощью алгоритма ***AutoCorrelation***, принимающего на вход массив значений (в данном случае – спектр аудиосигнала). Безвысотные звуки (немузыкальные звуковые эффекты) и чистые тона (звуки, совершающие гармонические колебания одинаковой частоты) имеют значение рельефности высоты, близкое к нулю, в то время как звуки, спектр которых содержит несколько гармоник, имеют большее значение рельефности высоты.

15. ***Centroid***

Алгоритм вычисляет значение спектрального центроида (*spectral centroid*), которое определяется как «центр масс» спектра:

$$C_t = \frac{\sum_{n=1}^{N} M_t[n] * n}{\sum_{n=1}^{N} M_t[n]}$$

где $M_t[n]$ – значение спектра в кадре t на частоте n [31].

16. ***Dissonance***

Алгоритм вычисляет сенсорный диссонанс (*sensory dissonance*) аудиосигнала на основе «грубости» пиков спектра аудиосигнала. На вход подаются два вектора: вектор частот и вектор амплитуд пиков спектра аудиосигнала, значения которых отсортированы по частоте. В теории музыки диссонансом называется категория гармонии, характеризующая неслияние в восприятии одновременно звучащих тонов, а также сами созвучия, воспринимаемые как неслитные (в отличие от консонанса, характеризующего слияние). При этом консонанс ощущается как мягкое звучание, диссонанс – как заострённое, раздражающее, беспокойное. Вычисленное значение

диссонанса находится в интервале от 0 до 1, где 0 означает абсолютно консонантный звук, а 1 – абсолютно диссонантный звук.

17. ***Loudness***

    Алгоритм вычисляет громкость аудиосигнала, которая определяется по закону Стивенса как энергия сигнала в степени 0.67. Для вычисления данного признака используются интервалы из 88200 отсчётов аудиосигнала (2 секунды при частоте дискретизации 44100 Гц) с размером скачка 44100 отсчётов (значения взяты из примеров кода *Essentia*). Значение громкости считается на каждом таком интервале, после чего вычисляется среднее значение громкости.

18. ***DynamicComplexity***

    Алгоритм вычисляет значение динамической сложности (*dynamic complexity*), которое определяется как среднее абсолютное отклонение от глобальной оценки уровня громкости в децибелах. Динамическая сложность показывает объем изменений громкости, присутствующих в аудиозаписи. На вход алгоритму подаётся аудиосигнал, на выходе – коэффициент динамической сложности, значение которого используется в качестве признака (вычисляется одно значение для каждой аудиозаписи), и оценка громкости в децибелах.

19. ***HPCP***

    Алгоритм вычисляет гармонический профиль высотных классов (*harmonic pitch class profile*, *HPCP*), который представляет собой k*12-мерный вектор. Высотный класс – множество всех звуковых высот, отстоящих друг от друга на целое число октав. Октава состоит из 12 полутонов, при $k = 1$ каждое значение вектора представляет интенсивность высотного класса соответствующего полутона, при $k > 1$ вместо полутонов используются их подразбиения. В данной работе используется $k = 1$.

20. ***RhythmExtractor2013***

    Алгоритм оценивает ритм музыкального произведения (аудиосигнала) в виде расположения ударов (не используется в данной работе), а также его темп, выраженный в ударах в минуту (*beats per minute* или *BPM*, вычисляется одно значение для каждой аудиозаписи). Также результатом работы алгоритма является список интервалов между ударами, использующийся для построения гистограммы ударов. Оценка расположения ударов выполняется с помощью алгоритма ***BeatTrackerDegara***,

на вход которому подаётся аудиосигнал, а выходом является массив временны́х отметок для ударов.

21. ***BpmHistogramDescriptors***

Алгоритм вычисляет значения статистик для самого высокого и второго по высоте пиков гистограммы ударов в минуту. Вычисляется амплитуда этих пиков, их вес и ширина. По оси абсцисс гистограммы откладывается темп в ударах в минуту, по оси ординат – сила ударов. Чем более выражен ритм произведения, тем выше будут пики на гистограмме; чем выше пик, тем больший вес он имеет [31]. Для каждой аудиозаписи вычисляется по одному значению каждого из признаков.

Для вычисления спектрального центроида, дисперсии, коэффициентов асимметрии и эксцесса спектра, а также спектрального диссонанса к входному сигналу с помощью алгоритма ***EqualLoudness*** применяется фильтр равной громкости (*equal-loudness filter*) [25]. Разработчиками *Essentia* было проведено исследование, показывающее, что применение фильтра равной громкости увеличивает вычислительную устойчивость (то есть малые изменения входных данных не приводят к заметным изменениям решения) для данных признаков.

В результате каждая аудиозапись описывается 99 признаками (с учётом представления n-мерных признаков как n одномерных).

3 Классификация

3.1 Проверочные данные

Как было сказано в разделе 1.1, важным требованием для эффективного обучения классификаторов музыкальных произведений по жанрам является наличие надёжных проверочных данных. Под проверочными данными понимается набор пар «музыкальное произведение – его жанр». В данной работе в качестве данных для классификации используется набор данных *GTZAN*, разработанный авторами работы [31]. Согласно исследованию, проведённому в работе [28], этот набор данных является одним из наиболее популярных наборов данных для оценки систем автоматической классификации музыкальных произведений – было рассмотрено 467 опубликованных работ, в 100 из которых использовался набор данных *GTZAN*. Он состоит из 1000 30-секундных музыкальных отрывков, каждый из которых может принадлежать к одному из 10 жанров. Несмотря на то, что данный набор не создавался намеренно для задач автоматической классификации музыкальных произведений, он стал своего рода тестом производительности (*benchmark*) систем автоматической классификации музыкальных произведений из-за своей доступности (существуют наборы данных, защищённые авторским правом и поэтому не находящиеся в свободном доступе), хотя он и имеет определённые недостатки, описанные в [28]. В работе [31], в которой набор данных GTZAN был использован впервые, была получена точность классификации, равная 61%. В [21] приводятся данные о значениях точности классификации с использованием набора *GTZAN*, равных 74%, 75%, 76.8%, 78.5% и 82.5%, а также данные для некоторых других наборов, который на данный момент не находятся в свободном доступе в сети Интернет.

3.2 Алгоритм Random forest

В качестве алгоритма классификации в данной работе будет использован алгоритм *Random forest* (далее для краткости будет использоваться обозначение *RF*) [24]. Алгоритм *Random forest* был предложен Л. Брейманом и А. Катлер в 2001 году. Основной его идеей является построение ансамбля деревьев принятия решений, то есть леса. Классификация нового объекта производится путём голосования: каждое дерево леса относит классифицируемый объект к одному из классов («голосует» за него), и побеждает тот класс,

за который проголосовало наибольшее число деревьев.

Пусть обучающая выборка состоит из N примеров, размерность пространства признаков равна M, и задано число m. Каждое дерево строится следующим образом:

1. Генерируется случайная подвыборка с повторением (бутстреп-выборка) размером N из обучающей выборки. Эта выборка будет являться обучающей выборкой для строящегося дерева.
2. При построении дерева в каждом узле случайным образом выбирается m из M переменных, и для разбиения узла используется наилучшее разбиение по этим m переменным. Значение m является постоянным в ходе построения дерева.
3. Каждое дерево строится до полного исчерпания подвыборки, отсечение ветвей не производится.

В оригинальной работе по алгоритму *RF* было показано, что процент ошибочной классификации алгоритма зависит от двух параметров:

- корреляция между двумя любыми деревьями леса – чем больше корреляция, тем выше частота ошибок;
- сила каждого отдельного дерева в лесе – дерево с малой частотой ошибок является сильным классификатором, увеличение силы отдельных деревьев уменьшает частоту ошибок леса.

Уменьшение значения m уменьшает как корреляцию, так и силу. Часто значение m выбирается равным $\sqrt{M}$ (например, в библиотеке машинного обучения *scikit-learn* [26] и пакете *randomForest* для языка программирования *R* [9]). В данной работе используется именно это значение.

Данный алгоритм практически не использовался для решения задачи автоматической классификации музыкальных произведений по жанрам, а в существующих работах с его использованием практически отсутствуют какие-либо детали реализации, несмотря на то, что алгоритм обладает рядом достоинств:

- высокая точность классификации;
- эффективная работа с большими объёмами данных;
- способность обрабатывать большое количество признаков;

- существуют методы оценивания важности отдельных признаков;
- существуют методы построения деревьев по данным с пропущенными значениями признаков;
- внутренняя оценка модели к обобщению (тест *out-of-bag*);
- и другие достоинства.

Существуют и недостатки, такие как:

- алгоритм может переобучаться (*overfit*) на зашумлённых наборах данных;
- большой размер получающихся моделей (что, в целом, не является большой проблемой в связи с развитием производительности вычислительной техники).

При создании бутстреп-выборки примерно 1/3 всех примеров в эту выборку не попадает. Эти данные называются *out-of-bag data*, или *OOB* (дословно – «данные за пределами мультимножества», а выборка с повторами как раз является мультимножеством). В алгоритме *RF* нет необходимости использовать перекрёстную проверку или даже отдельную проверочную выборку для получения объективной оценки ошибки классификации (хотя в данной работе используется отдельная проверочная выборка). Ошибка оценивается во время работы алгоритма следующим образом: каждый пример, не попавший в бутстреп-выборку k-го дерева, классифицируется этим деревом. Таким образом, примерно в 1/3 всех деревьев леса будут получены результаты классификации каждого примера. После этого выбирается класс j, получивший наибольшее количество голосов каждый раз, когда пример n находился вне обучающей выборки. Доля случаев, в которых класс j не равен истинному классу примера n, усреднённая по всем примерам, является оценкой ошибки *OOB*. Авторами алгоритма были проведены тесты, показывающие, что данная оценка является объективной.

Оценка ошибки *OOB* используется для определения важности признаков. Для этого производится следующая процедура: каждое дерево леса классифицирует примеры своей *OOB*-выборки, и подсчитывается количество голосов, отданных за правильный класс. После этого значения m-ой переменной в примерах *OOB*-выборки случайным образом перемешиваются, и полученные примеры снова классифицируются деревом. Далее вычисляется разность количества голосов, отданных за правильный класс на *OOB*-выборке с перестановкой и изначальной *OOB*-выборке. Среднее значение этой разности, взятое по всем деревьям леса, называется первичной оценкой важности для переменной m. Поделив

значение первичной оценки на её стандартное отклонение (равное СКО первичной оценки, делённому на значение корня квадратного из объёма выборки), получим *z*-оценку (*z-score*), которую и будем считать оценкой важности переменной *m*. Если число переменных велико, то можно построить случайный лес сначала с использованием всех переменных, а затем построить его ещё раз с использованием наиболее важных переменных, полученных после первого запуска.

Для построения отдельных деревьев случайного леса используется алгоритм *C4.5*, являющийся одним из популярных алгоритмов машинного обучения [30].

Подробное описание алгоритма приводится в [1]. Рассмотрим требования к структуре данных и самим данным, при выполнении которых алгоритм *C4.5* будет работоспособен:

- Описание атрибутов. Вся информация об объектах (примерах) должна описываться конечным набором признаков (атрибутов). Каждый атрибут должен иметь дискретное или числовое значение. Сами атрибуты не должны меняться от примера к примеру, и количество атрибутов должно быть фиксированным для всех примеров.
- Определённые классы. Каждый пример должен быть проассоциирован с конкретным классом, то есть один из атрибутов должен быть выбран в качестве метки класса.
- Дискретные классы. Классы должны быть дискретными, то есть иметь конечное число значений. Каждый пример должен однозначно относиться к конкретному классу. Случаи, когда примеры принадлежат к классу с вероятностными оценками, исключаются. Для качественной работы алгоритма количество классов должно быть значительно меньше количества примеров.

Рассмотрим сам алгоритм построения дерева. Пусть задано множество примеров *T*, где каждый элемент этого множества описывается *m* атрибутами. Количество примеров в *T* будем называть мощностью этого множества и будем обозначать |*T*|. Пусть метка класса принимает значения C_1, C_2, …, C_k.

Задача заключается в построении иерархической классификационной модели в виде дерева из множества примеров *T*. Процесс построения дерева будет происходить сверху вниз, то есть сначала создаётся корень дерева, затем потомки корня и так далее. На первом шаге имеем пустое дерево (имеется только корень) и множество *T* (ассоциированное с корнем). Требуется разбить исходное множество на подмножества – это можно сделать, выбрав один из атрибутов в качестве проверки. Тогда в результате разбиения получим *n* (по числу

значений атрибута) подмножеств и, соответственно, *n* потомков корня, каждому из которых поставлено в соответствие своё подмножество, полученное при разбиении множества *T*. Затем эта процедура рекурсивно применяется ко всем подмножествам.

Рассмотрим критерий выбора атрибута, по которому должно пойти ветвление. Требуется выбрать самый подходящий вариант разбиения из *m* возможных (по числу атрибутов). Пусть имеется проверка *X* (в качестве проверки может быть выбран любой атрибут), которая принимает *n* значений – A_1, A_2, …, A_n. Тогда разбиение *T* по проверке *X* даст подмножества T_1, T_2, …, T_n, при *X* равном соответственно A_1, A_2, …, A_n. Единственная доступная информация – это то, каким образом классы распределены в множестве *T* и его подмножествах, получаемых при разбиении по *X*.

Пусть $freq(C_j, S)$ – количество примеров из некоторого множества S, относящихся к одному и тому же классу C_j. Тогда вероятность того, что случайно выбранный пример из множества *S* будет принадлежать классу C_j, определяется как: $P = freq\dfrac{(C_j, S)}{|S|}$.

Согласно теории информации, количество содержащейся в сообщении информации зависит от её вероятности: $\log_2(\dfrac{1}{P})$ (1). Поскольку используется логарифм с двоичным основанием, то это выражение даёт количественную оценку в битах.

Выражение $Info(T) = -\sum_{j=1}^{k} \dfrac{freq(C_j, T)}{|T|} * \log_2(\dfrac{freq(C_j, T)}{|T|})$ (2) даёт оценку среднего количества информации, необходимого для определения класса примера из множества *T*, или энтропию множества *T*. Ту же оценку, но уже после разбиения *T* по *X,* даёт выражение $Info_X(T) = \sum_{i=1}^{n} \left|\dfrac{T_i}{T}\right| * Info(T_i)$ (3). Тогда критерием для выбора атрибута будет являться следующая формула: $Gain(X) = Info(T) - Info_X(T)$ (4).

Данный критерий считается для всех атрибутов, после чего выбирается атрибут, максимизирующий данной выражение. Этот атрибут будет являться проверкой в текущем узле дерева, а затем по этому атрибуту производится дальнейшее построение дерева, то есть в узле будет проверяться значение по этому атрибуту, и дальнейшее движение по дереву будет производиться в зависимости от полученного ответа. Такие же рассуждения можно применить к полученным подмножествам T_1, T_2, …, T_n и продолжить рекурсивно процесс построения дерева до тех пор, пока в узле не окажутся примеры из одного класса. Если в

процессе работы алгоритма получен узел, ассоциированный с пустым множеством (то есть ни один пример не попал в данный узел), то он помечается как лист, и в качестве решения листа выбирается наиболее часто встречающийся класс у непосредственного предка данного листа.

В случае с числовыми атрибутами следует выбрать некоторый порог, с которым должны сравниваться все значения атрибута. Пусть числовой атрибут имеет конечное число значений, обозначим их как $\{ v_1, v_2, \ldots, v_n \}$ и отсортируем. Тогда любое значение, лежащее между v_i и v_{i+1}, делит все примеры на два множества: те, которые лежат слева от этого значения, и те, которые лежат справа. В качестве порога можно выбрать среднее между значениями v_i и v_{i+1}: $TH_i = \frac{v_i + v_{i+1}}{2}$. Формулы (2), (3) и (4) последовательно применяются ко всем потенциальным пороговым значениям, и среди них выбирается то, которое даёт максимальное значение по критерию (4). Если выяснится, что среди всех атрибутов данный числовой атрибут имеет максимальное значение по критерию (4), то в качестве проверки выбирается именно он.

Для классификации нового объекта производится обход дерева, начинающийся с корня. На каждом внутреннем узле проверяется значение объекта Y по атрибуту, который соответствует проверке в данном узле, и, в зависимости от полученного ответа, находится соответствующее ветвление, и по этой дуге происходит переход к узлу, находящемуся на уровень ниже и так далее. Обход дерева заканчивается, как только встретится узел решения, который и даёт метку класса объекта Y.

При большом количестве атрибутов недостатком алгоритма будут являться ветвистые деревья. Этот недостаток можно устранить, используя механизм отсечения ветвей, однако, процедура алгоритма *RF* этого не требует, поэтому отсечение рассматриваться не будет. Также алгоритм содержит процедуру работы с пропущенными данными, то есть отсутствующими значениями некоторых признаков объекта, однако в контексте рассматриваемой задачи и наличие пропущенных данных означало бы ошибку на этапе извлечения значений признаков, которую необходимо устранить перед классификацией, поэтому такая ситуация не рассматривается.

4 Вычислительный эксперимент

4.1 Подбор параметров

Как было сказано в разделе 3.1, в качестве проверочных данных используется набор *GTZAN*, состоящий из 1000 примеров. Для проверки работы алгоритма этот набор был разбит на обучающую выборку, состоящую из 900 примеров (по 90 примеров на каждый жанр), и проверочную выборку, состоящую из 100 примеров (по 10 примеров на каждый жанр). Использование 10% данных в качестве проверочных является распространённой практикой (например, в [31]).

Фактически, параметрами алгоритма *RF* являются только количество деревьев и размер подмножества признаков, выбираемых для разбиения на каждом шаге построения дерева (как было сказано выше, в качестве этого значения используется округлённое значение корня квадратного из количества признаков, равное 10 при 99 используемых признаках). Однако в силу случайной природы алгоритма необходимо реализовать итеративный процесс, который позволит определить минимальное, максимальное и среднее значения точности классификации.

Итеративный процесс устроен следующим образом: на каждой итерации строится случайный лес с заданным числом деревьев на основе данных обучающей выборки, после чего вычисляется процентное значение успешно проклассифицированных данных проверочной выборки. На основе полученного значения в текущей итерации, а также всех прошлых итерациях, вычисляется среднеквадратичное отклонение значения точности классификации. Критерием остановки итеративного процесса является выполнение следующего условия: СКО значения точности классификации изменяется не более чем на 0.03% в течении 3 итераций. Данные значения были подобраны экспериментально – в среднем, до остановки итеративного процесса совершается 30-40 итераций. Такое количество итераций обеспечивает хорошее усреднение и приемлемое время работы всего итеративного процесса.

Кроме того, скорость работы алгоритма C4.5 зависит от количества различных значений каждого атрибута, так как число порогов, рассматриваемых при выборе наилучшего атрибута на каждом этапе ветвления, равно количеству значений атрибута минус единица. В процессе извлечения значений признаков из аудиофайлов было обнаружено, что большая

часть признаков имеет практически уникальные значения в каждом из примеров обучающей выборки. Так, в обучающей выборке из 900 примеров набора данных *GTZAN* практически все признаки имеют более 800 уникальных значений из 900 возможных (из них бóльшая часть имеет 888 значений и более). В связи с этим было принято решение для определения порогов использовать ручное разбиение вместо описанного выше, в котором в качестве порога используется среднее арифметическое двух соседних значений признака. Для этого экспериментально было подобрано число значений признака, находящееся в каждом классе разбиения. Так, если это число равно n, то в качестве первого порога будет использоваться значение $\frac{v_n + v_{n-1}}{2}$, где v_n – n-ое значение атрибута; в качестве второго порога – $\frac{v_{2n} + v_{2n-1}}{2}$, и так далее, пока индекс очередного порога не превысит количество значений атрибута.

Количество деревьев случайного леса также подбирается экспериментально. В целом, с увеличением количества деревьев улучшается и точность классификации, однако, постепенно прирост точности становится все менее заметным. Несмотря на используемую процедуру усреднения с помощью итеративного процесса, значение точности классификации все равно может варьироваться от запуска к запуску, из-за чего возникают сложности с оценкой прироста точности классификации. С увеличением количества деревьев также увеличивается и время работы алгоритма, поэтому необходимо подобрать такое значение количества деревьев, которое давало бы оптимальное соотношение времени работы алгоритма и точности классификации. Для экспериментального разбиения значений признаков на классы строился лес, состоящий из 200 деревьев – предварительная проверка показала, что такое значение даёт если и не оптимальный, то близкий к оптимальному результат, а также приемлемое время работы. Все вычисления производились на компьютере с процессором Intel Core 2 Duo @ 2.33 GHz. Результаты исследования приведены в таблице 1:

Таблица 1 – Определение оптимального разбиения множества значений признаков на классы

Количество значений в каждом классе разбиения	Средняя точность классификации	Максимальная точность классификации	Минимальная точность классификации	Среднее время выполнения одной итерации (с)
5	63.16%	66%	59%	1347
10	62.84%	66%	58%	660
20	62.9%	66%	59%	385
30	63.03%	68%	57%	274

40	63.49%	66%	58%	235
50	63.23%	68%	59%	236
60	63.19%	69%	57%	191
70	63.28%	67%	58%	197
80	62.5%	68%	58%	168
90	63.3%	68%	59%	159
100	63.57%	67%	60%	157
110	64%	68%	60%	173
120	62.42%	66%	59%	188
130	61.2%	67%	57%	197

Можно заметить, что практически все варианты разбиения дают близкие значения средней точности классификации, однако с ростом количества значений признака в каждом классе уменьшается среднее время работы алгоритма. Из приведённой таблицы можно сделать вывод, что оптимальным является разбиение множества значений признака на классы таким образом, чтобы в каждый класс попадало по 110 значений (таким образом, в наихудшем случае, то есть когда каждое значение атрибута является уникальным, всего будет 8 классов, причём в последний класс попадёт 130 значений). Полноценного исследования с количеством значений признака на каждый класс менее 5 не проводилось; так, если разбиение не применять вообще (то есть каждый класс будет содержать по 1 значению признака), то каждая итерация занимает около 45 минут при практически неизменном (по сравнению с ручным разбиением на меньшее число классов) значении успешности классификации, что делает сомнительным реальное применение алгоритма. При разбиении с количеством значений признака в каждом классе более 130 значение успешности классификации продолжает уменьшаться. Недостатком подобного подхода является необходимость подбирать оптимальное разбиение для каждого нового набора данных (по крайней мере, имеющего другой размер обучающей выборки). Среднее время выполнения, в свою очередь, зависит от загруженности центрального процессора другими задачами во время проведения эксперимента.

Следующим шагом является определение оптимального (или близкого к оптимальному) количества деревьев леса с применением разбиения, описанного выше. Результаты эксперимента приведены в таблице 2:

Таблица 2 – Определение оптимального количества деревьев случайного леса

Количество деревьев	Средняя точность классификации	Максимальная точность классификации	Минимальная точность классификации	СКО точности	Среднее время выполнения одной итерации (с)
10	45.25%	57%	33%	5%	8
50	59.2%	65%	54%	3%	40
100	61.6%	67%	56%	2.72%	86
150	63.03%	67%	57%	2.44%	120
200	64%	68%	60%	2.11%	173
300	64.1%	70%	60%	2.02%	258
400	64.35%	68%	60%	2.03%	329
500	64.19%	67%	61%	1.65%	423
600	64.14%	67%	61%	1.88%	492

Исходя из полученных данных, как и предполагалось при предварительной проверке, число деревьев, равное 200, является оптимальным или близким к оптимальному, учитывая как среднюю точность классификации, которая при количестве деревьев, большем 200, может незначительно увеличиться, так и среднее время выполнения одной итерации, растущее при увеличении количества деревьев, а также тот факт, что средняя точность классификации может незначительно изменяться от запуска к запуску в силу случайной природы алгоритма *RF*. Дальнейшие исследования будут проводиться с использованием случайных лесов, состоящих из 200 деревьев.

4.2 Оценка важности признаков

В главе 3.2 была описана процедура оценки важности отдельных признаков, использующаяся в алгоритме *RF*. Данная процедура была включена в итеративный процесс – на каждой итерации оценивались значения важности признаков для построенного леса, а после завершения итеративного процесса полученные значения усреднялись. Независимо от количества деревьев, все значения важности признаков оказались положительными, что говорит о том, что ни один признак не является «лишним», то есть не влияет отрицательно на качество классификации. Однако, так как значение успешности не привязано к какой-либо шкале, сложно оценить, приносят ли наименее важные признаки ощутимый прирост к

точности классификации – возможно, от некоторых признаков можно избавиться для сокращения времени работы алгоритма.

Для проверки влияния отдельных признаков на точность классификации признаки были упорядочены по убыванию значения важности. После чего последовательно исключались группы наименее важных признаков с последующей проверкой точности классификации до тех пор, пока точность классификации не начала ощутимо ухудшаться. Результаты представлены в таблице 3:

Таблица 3 – Исследование влияния важности наименее важных признаков на точность классификации

Количество признаков	Средняя точность классификации	Максимальная точность классификации	Минимальная точность классификации	Среднее время выполнения одной итерации (с)
99	64%	68%	60%	173
89	63.46%	67%	60%	151
79	63.67%	67%	59%	141
69	63%	67%	60%	129
59	62.58%	67%	56%	118
49	62.67%	66%	58%	110
39	62.26%	69%	56%	86
34	61%	65%	56%	61
29	60.5%	66%	56%	59
24	55.63%	60%	49%	47

В зависимости от того, какое ухудшение точности классификации считать ощутимым, можно исключить из рассмотрения разное количество признаков. Так, при допустимом ухудшении в 1%, можно исключить 30 признаков (примерно 30% от общего числа), а при допустимом ухудшении в 2% – 60 признаков (примерно 60% от общего числа), при исключении большего количества признаков точность начинает уменьшаться более быстрыми темпами. Однако, учитывая невысокие показатели точности классификации в целом, ухудшение точности на 2% является достаточно ощутимым. При достаточном количестве вычислительных ресурсов и времени можно вообще не проводить процедуру исключения наименее важных признаков.

Также для иллюстрации были исключены наиболее важные признаки с последующей

проверкой точности классификации. Результаты представлены в таблице 4:

Таблица 4 – Исследование влияния важности наиболее важных признаков на точность классификации

Количество признаков	Средняя точность классификации	Максимальная точность классификации	Минимальная точность классификации	Среднее время выполнения одной итерации (с)
99	64%	68%	60%	173
89	62.2%	69%	59%	158
79	55%	59%	51%	150

Можно заметить, что при исключении наиболее важных признаков точность классификации уменьшается гораздо быстрее, чем при исключении менее важных признаков, что показывает применимость самой процедуры оценки важности отдельных признаков.

Опять же, в силу случайной природы алгоритма *RF*, оценки важности признаков могут изменяться от запуска к запуску, однако данные изменения не являются значительными, и общая картина в рамках одного набора данных остаётся практически неизменной. Однако при использовании другого набора данных (в частности, с другим набором музыкальных жанров) ситуация может измениться – например, может оказаться, что некоторые признаки плохо описывают заданный набор жанров, и тогда такие признаки можно будет исключить из рассмотрения, или же наоборот – признаки с низкой степенью важности для набора данных *GTZAN* окажутся важны для другого набора данных. Как и в случае с разбиением множества значений признаков по классам, оценку важности переменных необходимо каждый раз производить заново для нового набора данных, что также является недостатком.

Кроме определения влияния отдельных признаков на точность классификации, важным результатом процедуры оценки важности признаков является общая картина относительная важности признаков. Как было сказано в разделе 1.2, высокоуровневые признаки редко применяются в задаче автоматической классификации музыкальных произведений по жанрам, уступая низкоуровневым признакам в точности описания музыкальных произведений. Данное утверждение подтверждается и экспериментальными результатами – так, среди наименее важных признаков оказалась бóльшая часть значений признака *HPCP* (гармонический профиль высотных классов). Данный признак является высокоуровневым и представлен 24 значениями (12 значений математического ожидания и 12

значений дисперсии), и 7 из 10 наименее важных признаков являются значениями признака *HPCP*, а 22 из 24 значений этого признака находятся во второй половине упорядоченного по важности списка признаков.

Также среди первых пяти наименее важных признаков находятся значения ширины самого высокого и второго по высоте пиков гистограммы ударов в минуту, тоже являющиеся высокоуровневыми признаками. Тем не менее, само значение количества ударов в минуту и, в меньшей степени, значения высоты и веса самого высокого пика, также являющиеся высокоуровневыми признаками, находятся среди наиболее важных признаков. Такой высокоуровневый признак, как динамическая сложность (алгоритм *DynamicComplexity*), находится в нижней трети списка признаков. Из всех значений признаков, описывающих энергию сигнала в различных частотных диапазонах (алгоритм *EnergyBand*), наиболее важными являются значения математического ожидания и дисперсии энергии в диапазоне от 20 до 150 Гц (то есть, в самом низком), и, как правило, чем выше частотный диапазон, тем менее важны признаки, описывающие его энергию, что говорит о том, что полезная информация сосредоточена в низких частотах аудиосигнала.

Наиболее важные признаки, в основном, являются спектральными. Мел-частотные кепстральные коэффициенты (*MFCC*), являющиеся популярными признаками в задачах автоматической классификации музыкальных жанров, также являются и одними из наиболее важных признаков. Так, 7 из 10 наиболее важных признаков являются значениями математического ожидания или дисперсии отдельных коэффициентов, а остальные значения *MFCC* также сконцентрированы в верхней части списка признаков. В работе [31] использовались только первые 5 коэффициентов из 13 как обеспечивающие наилучшую точность классификации, однако вычислительный эксперимент показал, что наиболее важными являются коэффициенты с 3-го по 7-й, а 1-й и 2-й коэффициенты имеют меньшее значение важности. В число наиболее важных признаков также входят спектральный срез и спектральный поток, впервые использовавшиеся в [31], а также значение дисперсии и коэффициента асимметрии спектра, являющиеся центральными моментами (как было сказано в описании алгоритма *DistributionShape*, эти признаки хорошо характеризуют форму распределения).

4.3 Выводы

Для того, чтобы сделать вывод о качестве классификации при использовании

алгоритма *RF*, необходимо решить, какое из значений точности будет использоваться в качестве оценки – среднее, максимальное или минимальное. Пессимистичный подход – использование минимального значения точности классификации, так как оно является гарантированным. На основе таблицы 2 можно сделать вывод, что при количестве деревьев леса, равном 200 или более, минимальная точность классификации составляет 60-61%, что сравнимо с точностью классификации, полученной в [31]. Среднее значение точности классификации составляет приблизительно 64% (значение немного увеличивается с ростом количества деревьев). Максимальная точность классификации, полученная в процессе исследований, составляет 70%, что все равно не превосходит результаты, полученные другими исследователями, описанные в разделе 3.1, за исключением [31], при том, что максимальное значение может быть и не достигнуто.

Невысокая точность может быть связана как со способом извлечения значений признаков, так и с самим набором признаков. В работе, в основном, использовались распространённые и зачастую довольно простые признаки, в то время как в работах, показывающих более высокую точность классификации, используются признаки, имеющие под собой сложный математический аппарат. Кроме того, для повышения точности классификации можно использовать улучшения, предложенные в разделе 1.4, а также экспериментировать с другими параметрами – например, использовать для представления музыкального произведения не один отрезок аудиофайла, а несколько, взятых в разные моменты времени. Библиотека *Essentia* предусматривает возможность создания собственных алгоритмов, что позволяет расширять систему в сторону увеличения количества используемых признаков.

По всей видимости, алгоритм *RF* не слишком подходит для решения задачи автоматической классификации музыкальных произведений по жанрам в силу своей случайной природы и, как следствие, довольно большого разброса между минимальным и максимальным значениями точности классификации, по крайней мере, на малых обучающих выборках. Возможно, на более крупных обучающих выборках алгоритм будет показывать более стабильные результаты, однако создание надёжного набора данных само по себе является сложной задачей, кроме того, с ростом обучающей выборки возрастёт и время выполнения алгоритма. К тому же, для повышения скорости работы требуется ручное разбиение множества значений каждого признака, которое, во-первых, необходимо подбирать вручную для разных наборов данных, а во-вторых, может ухудшить точность классификации; в противном случае алгоритм будет работать неприемлемо долго даже на небольших

обучающих выборках.

Однако, независимо от полученных результатов, заслуживает внимания процедура оценки важности признаков, так как результаты эксперимента показали, что можно избавиться от большого числа признаков при незначительном ухудшении точности классификации и значительном уменьшении времени работы алгоритма. Даже при использовании другого алгоритма классификации, можно предварительно один раз проклассифицировать данные алгоритмом *RF* с целью получения оценок важности признаков, и впоследствии использовать пространство признаков уменьшенной размерности.

5 Программная часть

5.1 Общие положения

Для проверки работы автоматического классификатора музыкальных произведений по жанрам с помощью описанных выше алгоритмов был разработан прототип системы автоматической классификации музыкальных произведений по жанрам на языке *C++* (версия *C++11*). Разработанное приложение имеет интерфейс командной строки.

Далее будут приведены диаграммы классов и диаграммы последовательности, описывающие приложение. Если операция не имеет возвращаемого значения (то есть имеет тип *void*), то на диаграмме тип возвращаемого значения опускается. Для удобства могут быть опущены типы атрибутов и параметров операций, если имя атрибута или параметра отражает его тип, а также если параметр с аналогичным именем и типом был описан на диаграмме ранее. Операции-аксессоры (геттеры) и операции-мутаторы (сеттеры) могут быть для краткости заменены на запись вида *{ get; set; }*, либо просто *{ get; }*, если в классе присутствует только операция-аксессор (имитация свойств (*property*), отсутствующих в языке *C++*), при этом сам атрибут, для которого были созданы соответствующие операции, на диаграмме может не указываться. Если между двумя объектами существует направленная ассоциация или объект одного класса агрегирует объекты другого класса, то предполагается, в классе существует соответствующий атрибут и, как правило, операция-аксессор, которые не указываются на диаграмме. Все ссылки на объекты в программном коде представлены «умными указателями» с автоматическим подсчётом ссылок типа *std::shared_ptr*, что помогает избежать утечек памяти. Основным видом коллекции, используемой в приложении, является класс *std::vector*, который на диаграммах классов для краткости будет представлен как массив, то есть символами []. На диаграммах классов используется стереотип *interface*, хотя в языке *C++* нет такого понятия, как «интерфейс», поэтому следует понимать, что имеется в виду абстрактный класс, у которого все методы чисто виртуальные, а под реализацией интерфейса следует понимать наследование от абстрактного класса.

Разрабатываемая система состоит из следующих составных частей:

- подсистема работы с признаками;
- подсистема классификации;

- подсистема взаимодействия с пользователем.

Система проектировалась с учётом возможного расширения в двух направлениях:

- добавление новых методов извлечения признаков (новые сторонние библиотеки, собственные алгоритмы);
- добавление новых алгоритмов классификации.

5.2 Архитектурные принципы и практики

На рисунке 1 представлена общая диаграмма классов приложения. Имена атрибутов и операций опущены для удобства представления. Далее будут представлены диаграммы классов и диаграммы последовательности для отдельных подсистем приложения с подробным описанием.

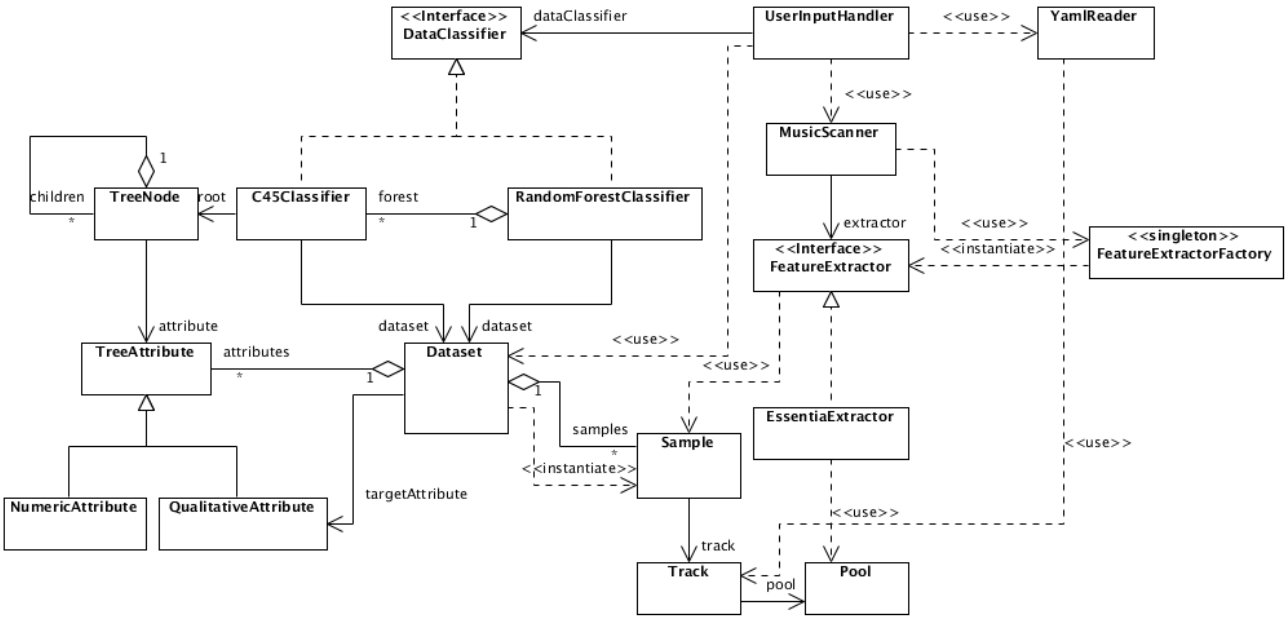

Рисунок 1 – Общая диаграмма классов приложения

На рисунке 2 представлена диаграмма классов для подсистемы взаимодействия с пользователем. Основным классом этой подсистемы является *UserInputHandler*, который выполняет операции в соответствии с командами, вводимыми пользователем в приложении. Обработка пользовательского ввода располагается в файле *main.cpp*, являющимся точкой входа в приложение. В конструктор класса *UserInputHandler* передаётся путь к файлу, содержащему список признаков, используемых для классификации. Операция *scan()* с помощью объекта класса *MusicScanner* выполняет обход директорий музыкальной коллекции

и извлекает значения признаков из аудиофайлов. Обучающая и проверочные выборки создаются с помощью операции *buildDatasetFromPath(...)*, принимающей на вход путь к файлу, содержащему список путей к директориям, содержащим *YAML*-файлы со значениями признаков. Операция *rmsdCycle(...)* запускает итеративный процесс, описанный в разделе 4.1, принимая на вход количество деревьев леса, строящегося на каждой итерации. Операция *test(...)* запускает процедуру проверки точности построенного классификатора на проверочной выборке, возвращая процентное значение точности классификации. Аргументом этой операции является имя текстового файла, каждая строка которого представляет собой путь к директории файловой системы, содержащей *YAML*-файлы для проверочной выборки.

Значения признаков считываются из *YAML*-файлов с помощью класса *YamlReader*. Операция *readYamlFilesFromPath(...)*, используя операцию *search(...)*, рекурсивно обходит директорию, путь к которой передаётся в аргументе операции, и для каждого найденного *YAML*-файла с помощью операции *createTrackForPath(...)* создаёт экземпляр класса *Track*, который, в свою очередь, содержит экземпляр класса *Pool* библиотеки *Essentia* со считанными значениями признаков для данного музыкального произведения.

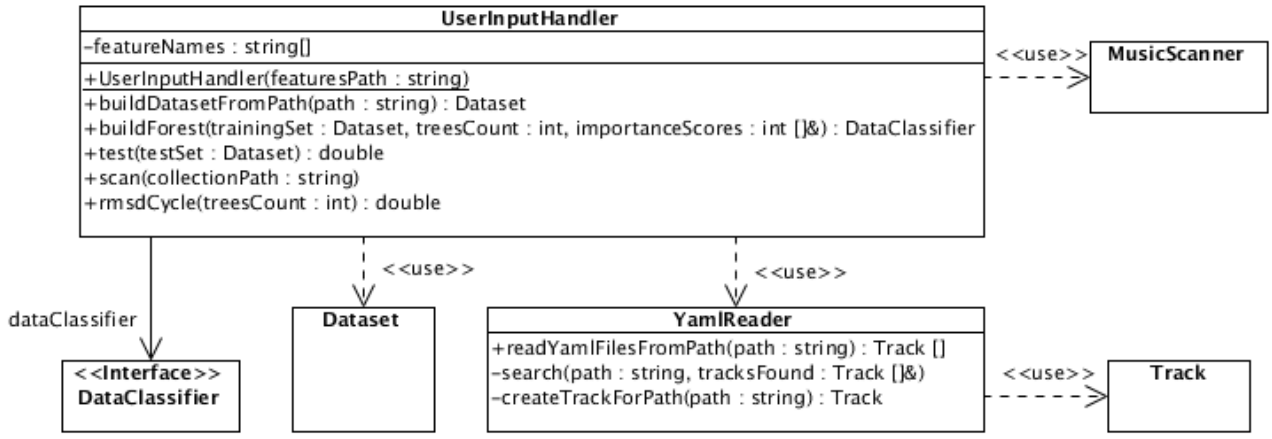

Рисунок 2 – Диаграмма классов для подсистемы взаимодействия с пользователем

На рисунке 3 представлена диаграмма классов подсистемы работы с признаками. Интерфейс *FeatureExtractor* определяет операцию *compute(...)*, которую должен реализовать каждый класс, имеющий функциональность для извлечения значений признаков из аудиофайлов. Аргументами этой операции являются имя входного аудиофайла и имя выходного файла с текстовым представлением извлечённых значений признаков. Класс *EssentiaExtractor* реализует этот интерфейс и позволяет извлекать значения признаков с использованием библиотеки *Essentia*. Извлечение признаков выполняется последовательным

выполнением ряда операций, определённых в классе *EssentiaExtractor*, с использованием экземпляров класса *Pool* из библиотеки *Essentia*.

Класс *FeatureExtractorFactory* является видоизмененным вариантом шаблона проектирования «Фабричный метод» [3], а также является классом-одиночкой (шаблон проектирования «Одиночка» [3]). Операция *createFeatureExtractorOfType()* позволяет создать экземпляр класса, реализующего интерфейс *FeatureExtractor*, на основе переданного аргумента *type* перечисляемого типа (*enum*) *FeatureExtractorType*. Так, для создания экземпляра класса *EssentiaExtractor*, аргумент *type* должен принимать значение *FeatureExtractorTypeEssentia*.

Класс *MusicScanner* предоставляет простой интерфейс для запуска процесса извлечения значений признаков из аудиофайлов. Операция *scan(...)* принимает в качестве аргумента путь к текстовому файлу, каждая строка которого является путём к директории файловой системы. Данная операция выполняет рекурсивный обход каждой из директорий коллекции, из всех файлов выбирает аудиофайлы формата *mp3* и для каждого аудиофайла вызывает метод *compute()* класса, реализующего интерфейс *FeatureExtractor*, в данной работе – *EssentiaExtractor*. Для создания экземпляра такого класса *MusicScanner* использует класс *FeatureExtractorFactory*. Полученные текстовые файлы в формате *YAML* со значениями признаков сохраняются в текущей директории (в той, откуда было запущено приложение).

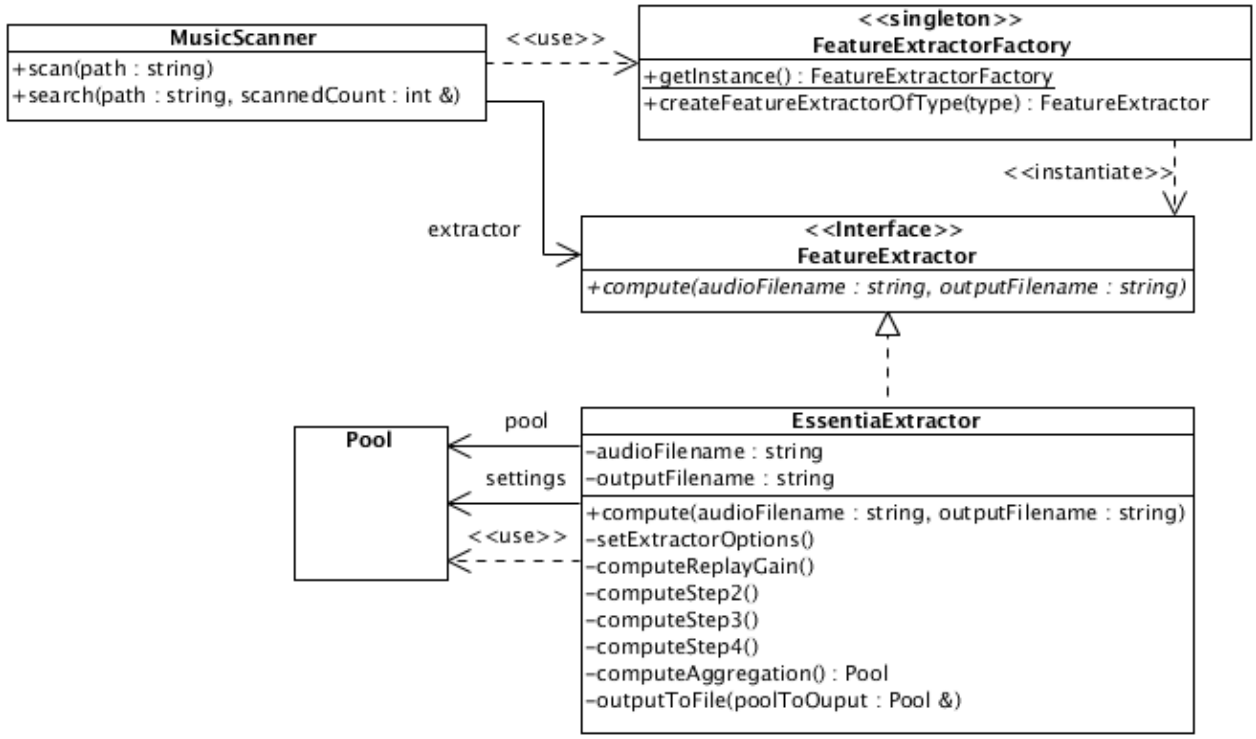

Рисунок 3 – Диаграмма классов подсистемы работы с признаками

Диаграмма последовательностей для процесса извлечения значений признаков представлена на рисунке 4. При вводе с клавиатуры пользователем команды *scan* вызывается соответствующая операция *scan(...)* класса *UserInputHandler*, запускающая рекурсивный обход директорий, содержащих аудиофайлы. Параметр *outputFile* операции *compute(...)* класса *EssentiaExtractor* конструируется из имени *mp3*-файла путём замены расширения *.mp3* на *.yaml*.

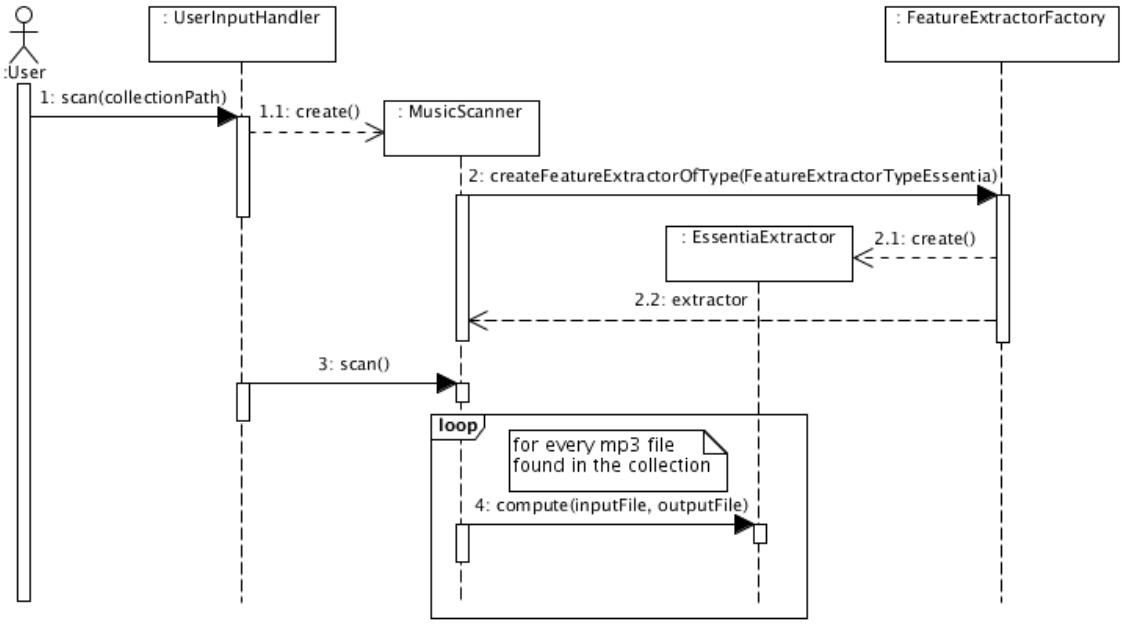

Рисунок 4 – Диаграмма последовательности для процесса извлечения значений признаков

На рисунке 5 представлена диаграмма классов для подсистемы классификации. Классы, отвечающие за алгоритмы классификации, будут подробно представлены далее.

Класс *TreeAttribute* предназначен для представления отдельного атрибута дерева принятия решений и хранит его имя, индекс в общем списке атрибутов (используется для быстрого поиска) и флаг, показывающий, является ли атрибут целевым. Наследниками данного класса являются классы *NumericAttribute* (числовой атрибут) и *QualitativeAttribute* (качественный атрибут). Класс *NumericAttribute* расширяет базовый класс *TreeAttribute* операциями для доступа к минимальному, максимальному и среднему значению атрибута, хранит список всех значений атрибута и предоставляет операцию доступа к списку, а также список пороговых значений, использующихся в алгоритме *C4.5* при ветвлении, и операцию для вычисления списка пороговых значений. Класс *QualitativeAttribute* хранит список значений качественного атрибута и предоставляет операцию для добавления нового значения в список.

Класс *TreeNode* предназначен для представления отдельного узла дерева принятия решений. Каждый узел, в свою очередь, хранит список дочерних узлов (которым может быть пустым, если узел является листом) и и предоставляет операцию для добавления нового дочернего узла в список. Атрибут *nodeType* может принимать одно из двух значений перечисляемого типа *TreeNodeType* – «меньше или равно» либо «больше», и служит для определения направления перехода от родительского узла при классификации примеров проверочной выборки, а атрибут *splitValue* содержит само значение порога, с которым сравнивается значение атрибута при выборе направления перехода. У листовых узлов атрибут *decision* содержит метку класса, то есть имя жанра, который будет являться результатом классификации примера обучающей выборки, дошедшего вниз по дереву до данного узла.

Класс *Track* предназначен для хранения информации об отдельном музыкальном произведении, представленном аудиофайлом. Атрибутами данного класса являются название музыкального произведения, его жанр, имя его исполнителя, путь к соответствующему файлу, а также ссылка на экземпляр класса *Pool* библиотеки *Essentia*, содержащий извлечённые значения признаков.

Класс *Sample* служит для представления объектов (примеров), используемых для классификации. Членами этого класса являются: экземпляр класса *Track*, представляющий соответствующее музыкальное произведение, список значений атрибутов (признаков) для данного произведения, значение целевого атрибута (жанра), если этот объект принадлежит к обучающей выборке (то есть значение целевого атрибута известно), а также операции для доступа к значениям атрибутов.

Класс *Dataset* представляет собой набор данных, то есть множество объектов для классификации – экземпляров класса *Sample*. Объекты этого класса хранят информацию о списке атрибутов и их имён, а также о целевом атрибуте, используемых в данном наборе, и предоставляют операции доступа к этим данным. Операция *sampleFromTrack(...)* предназначена для инстанцирования класса *Sample*. С помощью операции *getBootstrapAndOOBDataset(...)* из текущего набора данных создаются два набора данных, используемых в алгоритме *RF* – бутстреп-выборка и набор *OOB*-данных.

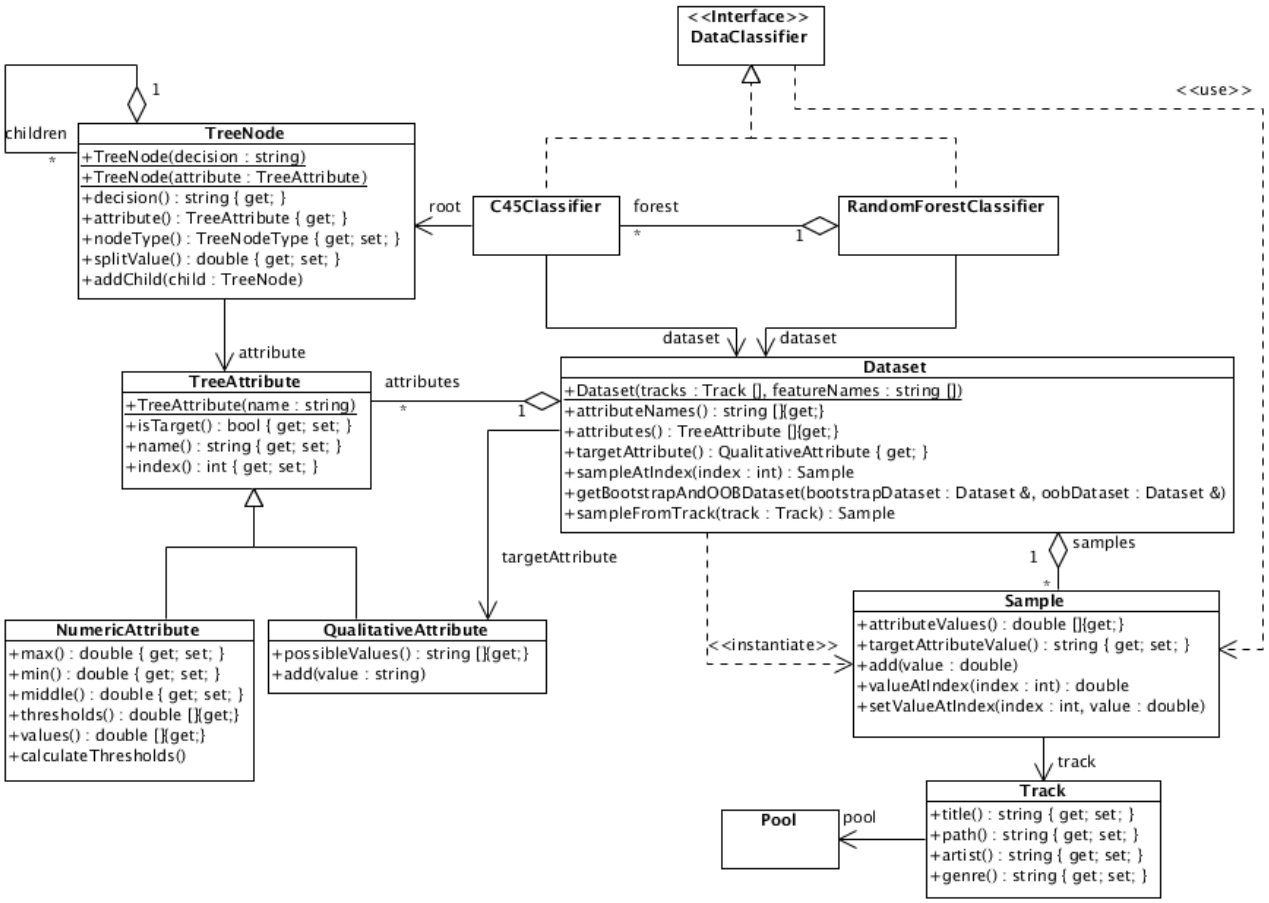

Рисунок 5 – Диаграмма классов подсистемы классификации

На рисунке 6 представлена диаграмма последовательности, описывающая процесс создания набора данных для обучающей или проверочной выборок, то есть объекта класса *Dataset*, на основе *YAML*-файлов, содержащих значения признаков. Для десериализации *YAML*-файлов используется класс *YamlReader*, который рекурсивно обходит директории, хранящие *YAML*-файлы, и с помощью алгоритма *YamlOutput* из библиотеки *Essentia* десериализует содержимое каждого найденного *YAML*-файла в объект класса *Pool*. Полученный объект класса *Pool* используется для создания объекта класса *Track*, который локально сохраняется в объекте *YamlReader*, а полученный список объектов *Track* затем передаётся в конструктор класса *Dataset*.

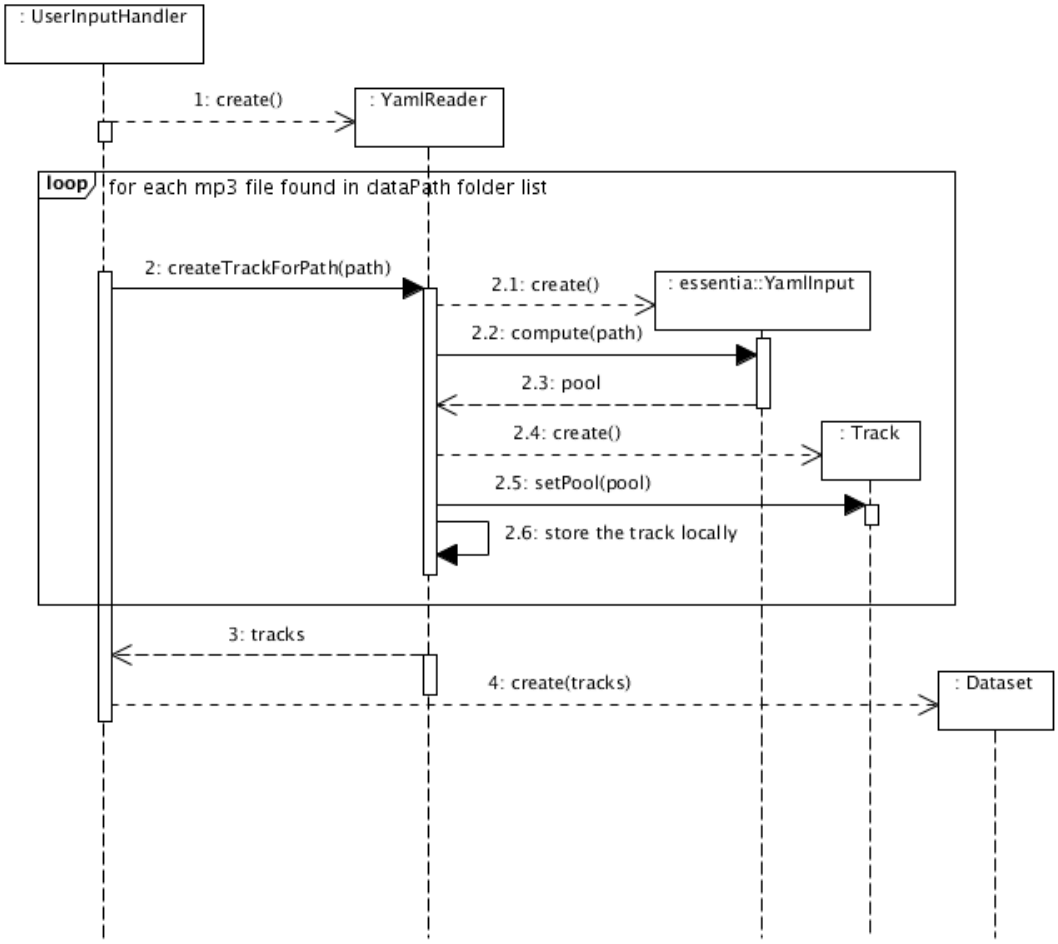

Рисунок 6 – Диаграмма последовательности для процесса создания объекта набора данных

На рисунке 7 представлены классы, представляющие алгоритмы классификации. Интерфейс *DataClassifier* определяет операцию *classify(...)*, которая принимает на вход пример – объект класса *Sample*, а возвращаемым значением является имя жанра, являющееся результатом классификации. Этот интерфейс реализуется классами *C45Classifier* и *RandomForestClassifier*, представляющими алгоритмы *C4.5* и *Random forest*, соответственно.

Класс *C45Classifier* содержит ссылку на корневой узел, который является пустым в начале классификации и постепенно заполняется потомками. Класс содержит набор операций, использующихся в алгоритме *C4.5* для построения дерева принятия решений, а также ряд других, вспомогательных операций. В конструктор класса передаётся объект класса *Dataset*, содержащий обучающую выборку, а также размер случайного подмножества атрибутов (параметр алгоритма *RF*), из которого будет выбираться атрибут на каждом этапе ветвления.

Класс *RandomForestClassifier* хранит список классификаторов типа *C45Classifier*, образующих лес деревьев принятия решений. Операция *grow()* запускает процесс построения

леса на основе заданной обучающей выборки и количества деревьев леса.

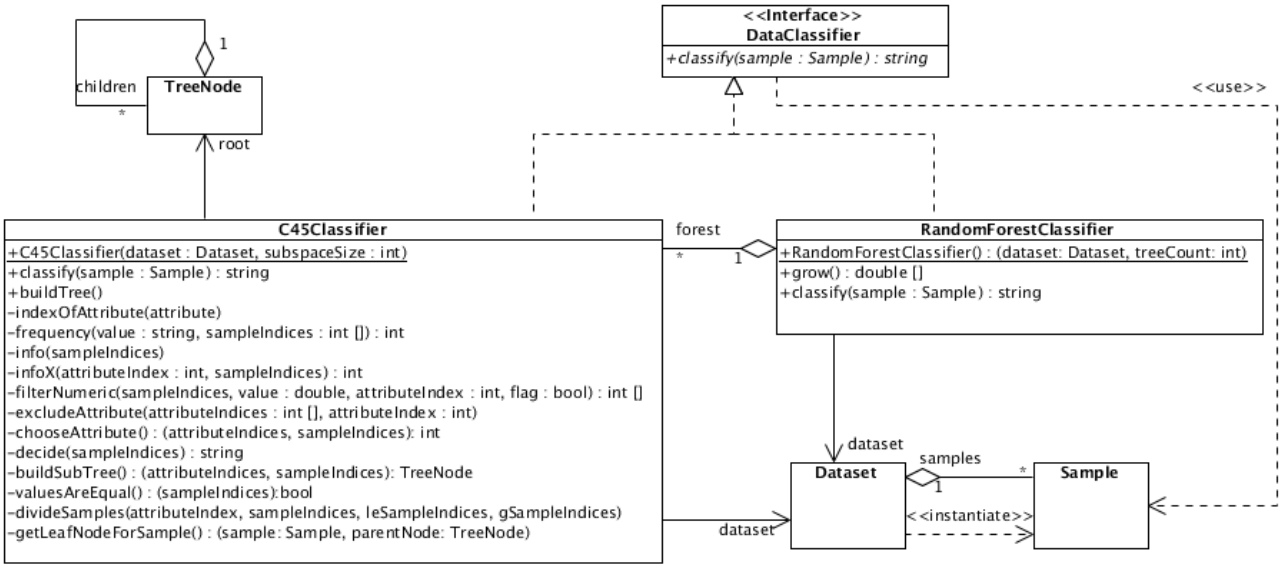

Рисунок 7 – Диаграмма классов, представляющих алгоритмы классификации

Диаграмма последовательности, описывающая итеративный процесс классификации, представлена на рисунке 8. При вводе с клавиатуры пользователем команды *rmsd* вызывается соответствующая операция *rmsdCycle(...)* класса *UserInputHandler*, на вход которой подаётся количество деревьев, с которым на каждой итерации будет строиться случайный лес. Класс *UserInputHandler* создаёт два объекта класса *Dataset* – обучающую и проверочную выборки. После этого запускается цикл, который выполняется до тех пор, пока не будет выполнено условие остановки итеративного процесса (описано в разделе 4.1). На каждой итерации цикла строится случайный лес на основе данных обучающей выборки, при этом вычисляются оценки важности признаков, после чего последовательно классифицируются все объекты проверочной выборки и вычисляется процентное значение точности классификации, а также его СКО на основе результатов предыдущих итераций. После завершения итеративного процесса вычисляется среднее значение успешности и средние значения оценок важности признаков по всем итерациям. Сами алгоритмы *Random forest* и *C4.5*, используемые для классификации, были подробно описаны в разделе 3.2 и поэтому на диаграммах последовательности не приводятся.

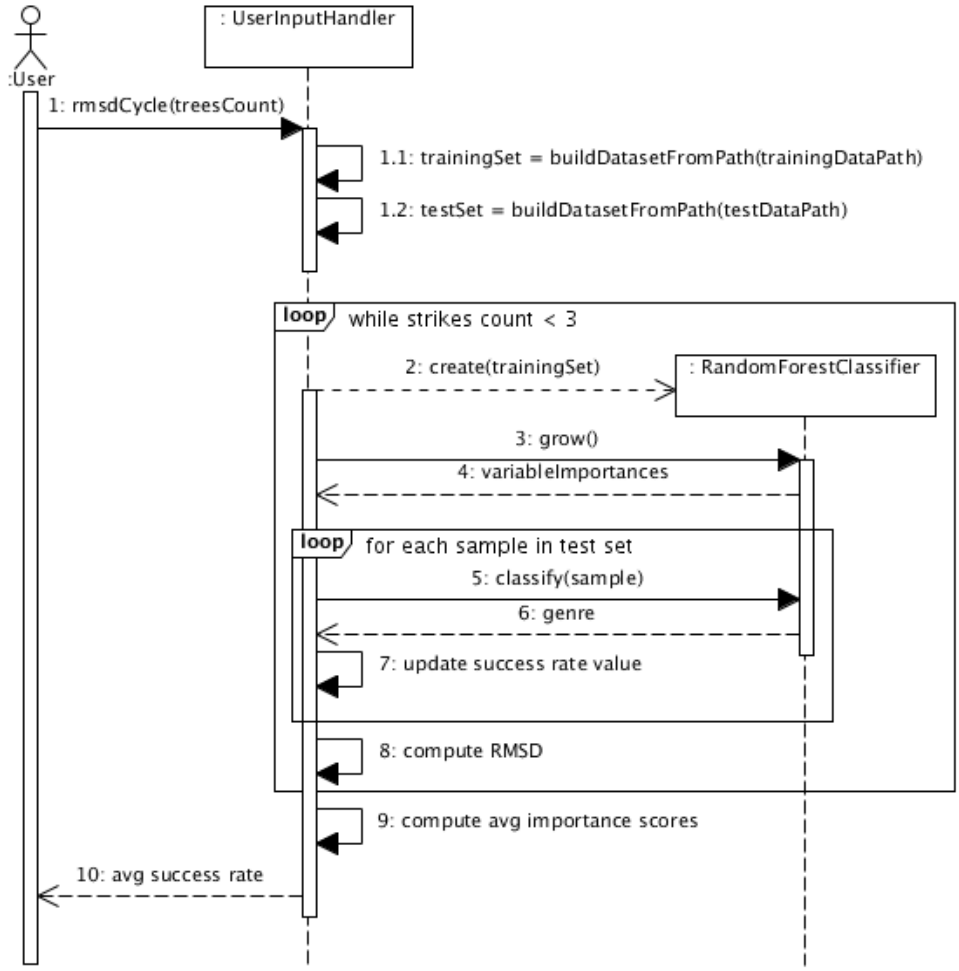

Рисунок 8 – Диаграмма последовательности для итеративного процесса классификации

Заключение

В рамках данной работы были решены все поставленные задачи:

- Описана постановка задачи, произведён обзор видов признаков, используемых для классификации, а также обзор различных алгоритмов классификации.
- Приведено описание библиотеки *Essentia*, используемой для извлечения значений признаков, а также описание соответствующих алгоритмов.
- Рассмотрена проблема наличия достоверных проверочных данных, приведены описания алгоритмов *Random forest* и *C4.5*.
- Проведён вычислительный эксперимент по классификации музыкальных произведений, описана процедура подбора входных параметров алгоритмов, приведены полученные результаты, произведён анализ влияния важности отдельных признаков на точность классификации.
- Описан разработанный прототип программной системы для автоматической классификации. Приведены как диаграммы классов системы, так и диаграммы взаимодействия; описано назначение отдельных классов, их атрибутов и операций.

Таким образом, достигнута цель работы – исследование задачи автоматической классификации музыкальных произведений по жанрам и создание прототипа программной системы, позволяющей выполнять автоматическую классификацию с использованием алгоритма классификации *Random forest*. Результаты вычислительного эксперимента показали, что алгоритм *RF* имеет ряд недостатков, затрудняющих его применение для решения задачи автоматической классификации музыкальных произведений по жанрам, а полученная точность классификации не превосходит результаты, представленные в других работах (в том числе и из-за использованного набора признаков), однако заслуживает внимания процедура оценки важности признаков, встроенная в алгоритм *RF*, которая может применяться и в сочетании с другими алгоритмами классификации. Прототип разработанного приложения может быть расширен как новыми алгоритмами классификации, так и новыми способами извлечения значений признаков.

Список использованных источников и литературы

1. Деревья решений – C4.5 математический аппарат [Электронный ресурс]. – URL: http://www.basegroup.ru/library/analysis/tree/math_c45_part1/ (дата обращения: 2.06.2014)
2. Котомин А.В. Распознавание речевых команд с использованием сверточных нейронных сетей / А.В. Котомин – Труды всероссийской конференции (с международным участием) "Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем", 18-22 апреля 2011 г., Москва, Российский университет дружбы народов.
3. Приемы объектно–ориентированного проектирования : паттерны проектирования : пер. с англ. / Э. Гамма [и др.]. – СПб. : Питер, 2001. – 368 с.
4. A feature selection approach for automatic music genre classification / C. N. Silla Jr., A. L. Koerich, C. A. A. Kaestner - International Journal of Semantic Computing, vol. 3, no. 2, pp. 183-208, 2009.
5. Aucouturier J.J., Pachet F. Representing musical genre: a state of the art / J.J. Aucouturier, F. Pachet – Journal of New Music Research, vol. 32, no. 1, pp. 83-93, 2003.
6. Automatic genre classification of music content: a survey / N. Scaringella, G. Zoia, D. Mlynek – Signal Processing Magazine, IEEE (Volume 23, Issue 2), March 2006.
7. Borg N., Hokkanen G. What Makes for a Hit Pop Song? / N. Borg, G. Hokkanen – Stanford University, CS 229 Machine Learning Final Projects, Autumn 2011.
8. Classification of musical genre: a machine learning approach / R. Basili, A. Serafini, A. Stellato – ISMIR, 2004.
9. CRAN – Package randomForest [Electronic resource]. – URL: http://cran.r-project.org/web/packages/randomForest/index.html (retrieval date: 2.06.2014)
10. ESSENTIA | Open-source C++ library for audio analysis and audio-based music information retrieval [Electronic resource]. – URL: http://essentia.upf.edu/ (retrieval date: 2.06.2014)
11. Gouyon F., Dixon S. A review of automatic rhythm description system / F. Gouyon, S. Dixon – Computer Music Journal, vol. 29, pp. 34-54, 2005.
12. Improving Music Genre Classification Using Automatically Induced Harmony Rules / A.

Anglade, E. Benetos, M. Mauch, S. Dixon – Journal of New Music Research, vol. 39, no. 4, pp. 349-361, 2010.

13. Improving Music Genre Classification by Short-Time Feature Integration / A. Meng, P. Ahrendt, J. Larsen – IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.
14. Jin X., Bie R. Random Forest and PCA for Self-Organizing Maps based Automatic Music Genre Discrimination / X. Jin, R. Bie – Conference on Data Mining, pp. 414-417, 2006.
15. LibXtract [Electronic resource]. – URL: https://code.soundsoftware.ac.uk/projects/libxtract (retrieval date: 2.06.2014)
16. Marsyas [Electronic resource]. – URL: http://marsyas.info/ (retrieval date: 2.06.2014)
17. McKay C. Issues in automatic musical genre classification. / C. McKay – Presented at the McGill Graduate Students Society Symposium, 2004.
18. McKay C., Fujinaga I. Musical genre classification: Is it worth pursuing and how can it be improved? / C. McKay, I. Fujinaga – Proceedings of the International Conference on Music Information Retrieval, 2006.
19. Melody description and extraction in the context of music content processing / E. Gomez, A. Klapuri, B. Meudic – Journal of New Music Research, Vol. 32(1), 2003.
20. MIREX 2013: Audio Train/Test: Genre Classification (Mixed) - MIREX08 Dataset [Electronic resource]. – URL: http://www.music-ir.org/nema_out/mirex2013/results/act/mixed_report/summary.html (retrieval date: 2.06.2014)
21. Music Genre Classification: A Multilinear Approach / I. Panagakis, E. Benetos, C. Kotropoulos – ISMIR 2008: Proceedings of the 9th International Conference of Music Retrieval, pp. 583-588, 2008.
22. Peeters G. A large set of audio features for sound description (similarity and classification) in the CUIDADO project / G. Peeters – CUIDADO I.S.T. Project Report, 2004.
23. Perrott D., Gjerdingen R.O. Scanning the dial: an exploration of factors in the identification of musical style / D. Perrott, R.O. Gjerdingen – Research Notes, Department of music, Northwestern University, Illinois, 1999.
24. Random forests – classification description [Electronic resource]. – URL:

http://stat-www.berkeley.edu/users/breiman/RandomForests/cc_home.htm (retrieval date: 2.06.2014)

25. Replay Gain – A Proposed Standard [Electronic resource]. – URL http://replaygain.hydrogenaudio.org/proposal/equal_loudness.html (retrieval date: 2.06.2014)
26. sklearn.ensemble.RandomForestClassifier – skikit-learn 0.14 documentation [Electronic resource]. – URL: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (retrieval date: 2.06.2014)
27. Sonic Annotator [Electronic resource]. – URL: http://vamp-plugins.org/sonic-annotator/ (retrieval date: 2.06.2014)
28. The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use / B. Sturm. – 2013. – Electronic Data. – URL: http://arxiv.org/abs/1306.1461 (retrieval date: 2.06.2014)
29. The Official YAML Web Site [Electronic resource]. – URL: http://www.yaml.org/ (retrieval date: 2.06.2014)
30. Top 10 algorithms in data mining / X. Wu et al. – Knowledge and Information Systems, vol. 14, pp. 1-37, 2007.
31. Tzanetakis G., Cook P. Musical Genre Classification of Audio Signals. / G. Tzanetakis, P. Cook – IEEE Transactions on Speech and Audio Processing, vol. 10, no. 5, July 2002.
32. Yaafe - audio features extraction [Electronic resource]. – URL: http://yaafe.sourceforge.net/ (retrieval date: 2.06.2014)

Приложение А. Руководство пользователя

Для запуска приложения требуется наличие операционной системы семейства *Linux* (желательно – семейства *Ubuntu Linux*, разработка велась на *Linux Mint*), на других операционных системах работоспособность приложения не проверялась. Так как целью разработки приложения являлась проверка выдвинутых предположений и получение результатов классификации на конкретном наборе данных, то разработанный прототип не обладает богатой функциональностью. Однако он реализует следующие функции, доступные пользователю:

- Сканирование директорий файловой системы и извлечение значений признаков из найденных mp3-файлов.
- Запуск итеративного процесса классификации с использованием алгоритма *Random forest*.

Для работы приложения требуется, чтобы в директории с исполняемым модулем приложения находились следующие файлы:

- *features.txt* – текстовый файл со списком имён используемых признаков, каждый признак должен располагаться на отдельной строке. Присутствие этого файла обязательно для работы приложения. Если перед именем признака располагается символ #, то такой признак при классификации не учитывается.
- *collection.txt* – текстовый файл со списком директорий, в которых располагается пользовательская коллекция аудиозаписей, каждая директория должна располагаться на отдельной строке. Данный файл требуется для работы процедуры сканирования. Если первым символом строки является символ #, то такая строка не учитывается в процессе поиска *mp3*-файлов (таким способом можно оставлять комментарии).
- *training_data.txt* и *test_data.txt* – текстовые файлы со списком директорий, в которых располагаются *YAML*-файлы для обучающей и проверочной выборок, соответственно. Данные файлы требуются для работы итеративного процесса классификации, и к ним применимы те же требования, что и к файлу *collection.txt*.

После запуска приложение выводит на экран список доступных команд. Интерфейс приложения представлен на рисунке А.1:

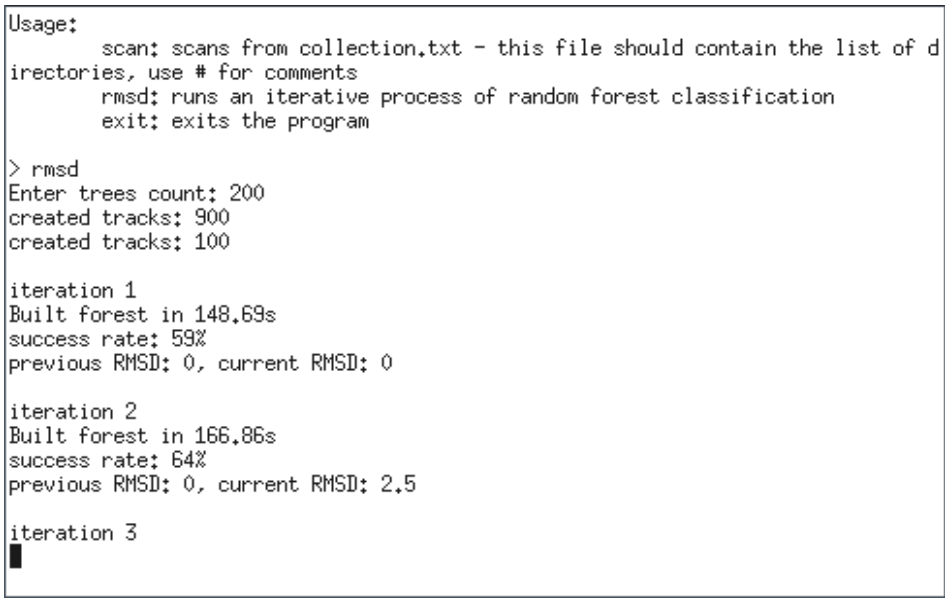

Рисунок А.1

Команда *scan* запускает процесс сканирования директорий коллекции, пользователь оповещается системой по мере извлечения значений признаков из очередного *mp3*-файла (в консоль выводится соответствующее сообщение).

Команда *rmsd* запускает итеративный процесс классификации, запрашивая у пользователя количество деревьев, которое будет содержать каждый строящийся случайный лес. На экран выводятся данные об объёмах обучающей и проверочной выборки, а также результаты каждой итерации – её номер, время построения леса, процентное значение точности классификации, а также текущее и предыдущее значения СКО точности классификации. По завершении итеративного процесса на экран выводятся значения важности признаков, упорядоченные по убыванию, а также общая информация об итеративном процессе: количество итераций, среднее, максимальное и минимальное значения точности классификации, значение СКО точности классификации и среднее время выполнения одной итерации.

На рисунке А.2 представлен интерфейс приложения после окончания итеративного процесса – в верхней части находится конец списка признаков, упорядоченных по важности, в нижней части – информация об итеративном процессе:

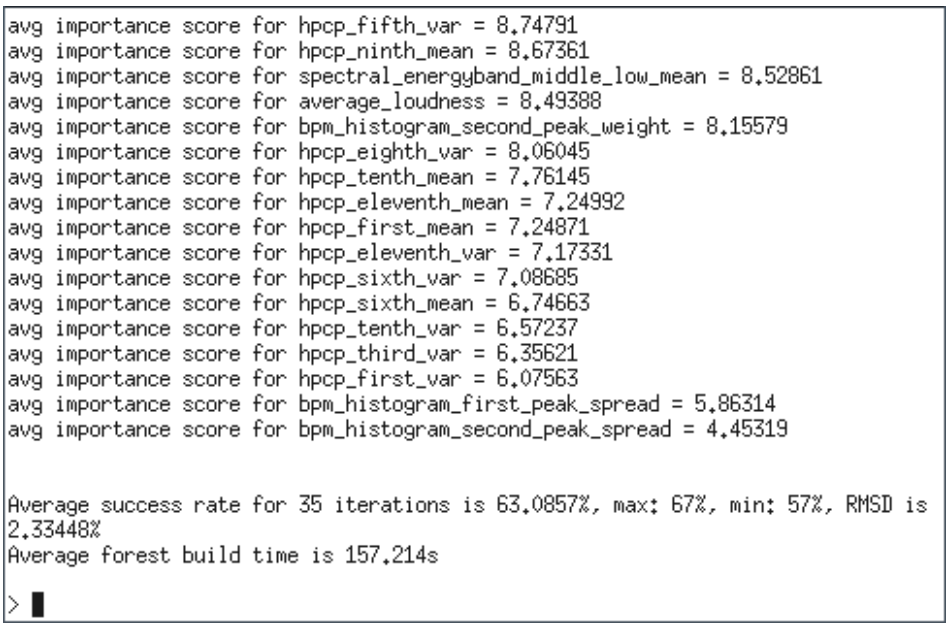

Рисунок А.2

Для выхода из приложения используется команда *exit*.

Приложение Б. Руководство программиста

Приложение разрабатывалось и тестировалось на операционной системе *Linux Mint 16 Petra* с использованием компилятора *gcc 4.8.1*. Так как не использовалось платформенно-зависимых компонентов, то приложение должно компилироваться и запускаться и на других операционных системах (семейств *Windows* и *OS X*). Однако, например, в разрабатываемом приложении класс `std::shared_ptr` находится в пространстве имён `<memory>`, в то время как, например, в операционной системе *OS X 10.8* с компилятором *clang* этот класс находится в пространстве имён `<tr1/memory>`, то есть, возможно, для компиляции приложения придётся внести изменения подобного рода. В качестве среды разработки использовалась кроссплатформенная среда *Codelite*.

Для запуска приложения требуется установить библиотеку Essentia, которая, в свою очередь, имеет ряд зависимостей, требуемых для её компиляции. Инструкции по установке как самой библиотеки, так и её зависимостей, могут быть найдены на веб-сайте библиотеки в разделе документации (http://essentia.upf.edu/documentation/installing.html).

Также для ряда задач в приложении используется коллекция *C++*-библиотек *Boost* (http://www.boost.org/), конкретно – библиотеки *Boost.Filesystem*, *Boost.Random*, *Boost.Lexical_Cast*, *Boost.Assignment*.

Описание классов приложения было произведено в главе 5, поэтому далее будут описаны способы расширения приложения новыми способами извлечения признаков и новыми алгоритмами классификации.

Для добавления в приложение нового экстрактора (то есть класса для извлечения значений признаков, например, с использованием другого набора признаков или отличной от *Essentia* библиотеки) необходимо создать наследника класса *FeatureExtractor* и реализовать метод *compute(...)*, принимающий на вход имя входного аудиофайла и выходного файла с извлеченными значениями признаков. После чего нужно добавить новое значение в перечисляемый тип *FeatureExtractorType*, соответствующее добавленному экстрактору, и в метод *createFeatureExtractorOfType(...)* класса *FeatureExtractorFactory* добавить необходимый код, инстанцирующий объект нового экстрактора.

Для добавления в приложение нового алгоритма классификации необходимо создать наследника класса *DataClassifier* и реализовать метод *classify(...)*. При этом надо учитывать, что данный метод работает с объектом класса *Sample*, представляющим пример, который

нужно проклассифицировать.

Запуск алгоритмов осуществляется из класса *UserInputHandler*, который, в свою очередь, создаётся в функции *main()* файла *main.cpp*. В класс *UserInputHandler* нужно добавить метод, запускающий классификацию с использованием добавленного алгоритма. После чего в функции *main()* файла *main.cpp* следует добавить условие, проверяющее значение, введённое пользователем с клавиатуры, и запустить соответствующий метод класса *UserInputHandler*.