

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ  
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И МНОГОПРОЦЕССО-  
СОРНЫХ СИСТЕМ

**Ромашов Дмитрий Сергеевич**  
**Дипломная работа**  
**Система определения музыкальных**  
**предпочтений**

Заведующий кафедрой,  
доктор физ.-мат. наук,  
профессор

Андрианов С.Н.

Научный руководитель,  
кандидат тех. наук,  
доцент

Гришкин В. М.

Рецензент,  
кандидат физико-математических наук,  
доцент

Степенко Н. А.

Санкт-Петербург

2016

# Содержание

|  |    |
|--|----|
| Введение .....   | 3  |
| Обзор существующих систем .....                              | 6  |
| Постановка задачи .....                                      | 9  |
| Глава 1. Теория .....  | 10 |
| 1.1 Получение амплитудно-частотного спектра .....            | 10 |
| 1.2 Определение параметров для классификации композиций..... | 12 |
| 1.3 Алгоритмы классификации.....                             | 13 |
| 1.3.1 Метод опорных векторов.....                            | 13 |
| 1.3.2 Метод ближайших соседей.....                           | 15 |
| 1.3.3 Решающее дерево.....                                   | 18 |
| Глава 2. Практическое исследование .....                     | 23 |
| 2.1 Используемые технологии.....                             | 23 |
| 2.2 Классификация .....                                      | 24 |
| 2.3 Перекрёстная проверка .....                              | 25 |
| 2.3.1 Варианты перекрёстной проверки.....                    | 27 |
| 2.3.2 Результаты перекрёстной проверки .....                 | 28 |
| 2.4 Результаты .....   | 31 |
| Выводы .....   | 36 |
| Заключение .....   | 37 |
| Список литературы.....                                       | 39 |
| Приложение .....   | 41 |

## Введение

В наше время людям доступно большое количество разнообразных музыкальных произведений на любой вкус из разных источников. Каждый человек хотел бы слушать ту музыку, которая больше нравится именно ему. Вручную прослушивать миллионы песен, чтобы отобрать понравившиеся для себя песни, достаточно долго и трудоёмко. Поэтому существуют рекомендательные системы [1], которые делают это за вас.

Существует два наиболее распространённых подхода к построению рекомендательных систем — коллаборативная фильтрация и рекомендации на основе содержимого (content based). Также существуют гибридные системы, которые являются комбинацией этих двух подходов. При рекомендациях на основе содержимого обо всех пользователях собирается информация, которая может говорить об их предпочтениях. Также для каждого объекта, который можно порекомендовать пользователям, выделяются признаки, по которым можно охарактеризовать этот объект. Для музыкальных композиций это могут быть альбом, жанр, исполнитель и так далее. На основе информации о пользователе ему подбираются объекты с нужными признаками. При коллаборативной фильтрации пользователю рекомендуются те объекты, которые понравились другим пользователям с похожими оценками.

Преимущество коллаборативной фильтрации перед content based системами состоит в том, что можно рекомендовать объекты, у которых нет явных признаков — мысли, идеи, мнения. Однако у коллаборативной фильтрации есть и значительные недостатки. Если предметов слишком много или пользователи мало оценивают эти предметы, то много объектов остаётся не оценёнными. Из-за этого трудно найти пользователей с похожими оценками.

Второй проблемой является то, что такая система плохо работает с новыми пользователями и предметами. Если новый пользователь ещё ничего не оценил, то система ничего не сможет ему порекомендовать. Также новый предмет может быть ещё никем не оценённым, поэтому рекомендательная система не сможет предложить этот предмет пользователям.

Для полноценной работы рекомендательных систем им нужна информация о пользователях, поэтому существуют различные способы получения этой информации.

Способы получения информации:

- оценка каких-либо объектов пользователем;
- сравнение каких-либо объектов пользователем;
- выбор лучшего объекта пользователем из группы объектов;
- отслеживание истории просмотров пользователя;
- отслеживание поведения пользователя в интернете.

Значительный толчок развитию рекомендательных систем принесло соревнование Netflix Prize [2]. Организатором соревнования была компания Netflix, занимающаяся прокатом DVD. Каждому просмотренному фильму клиент мог поставить свою оценку по пятибалльной шкале. В процессе работы компании ей удалось собрать более миллиарда оценок фильмов клиентами. Netflix использовала эту базу оценок для рекомендаций фильмов клиентам, поэтому компания была заинтересована в улучшении качества рекомендаций для получения большей прибыли с проката дисков. Для этих целей в 2006 году компания Netflix объявила соревнование Netflix Prize. На основе миллионов собранных оценок пользователей участники соревнования должны были разработать алгоритм, наилучшим образом определяющий оценку, которую поставит пользователь какому-либо фильму. Качество работы алгоритмов определялось при помощи метрики RMSE:

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(i,j) \in T} (x_j^i - y_j^i)^2},$$

где  $i$  — пользователь,  $j$  — предмет,  $y$  — оценка пользователя,  $x$  — предсказанная оценка,  $T$  — всё множество тестовых оценок.

Участники должны были улучшить разработанный Netflix алгоритм хотя бы на 10% по метрике RMSE. За победу присуждался приз в размере \$ 1 000 000. В конце третьего года соревнований только две команды смогли справиться с заданием. Несмотря на то, что только одна команда из двух получила обещанный приз, сам конкурс внёс значительный вклад в развитие данной области.

В ходе этого соревнования участники пытались улучшить или комбинировали существующие подходы рекомендательных систем. Однако они не принимали во внимание физическую природу музыкальных композиций, их амплитудно-частотный спектр. А что, если придумать метод поиска похожих композиций на основе их амплитудно-частотного спектра. Каковы будут его результаты? Ответ на этот вопрос я попытаюсь дать в своей работе.

# **Обзор существующих систем**

## **Вконтакте**

Хотя Вконтакте является социальной сетью, а не рекомендательным сервисом музыки, некоторые рекомендательные возможности у него всё-таки есть [3]. Например, есть кнопка “Показать похожие”, которая доступна для каждой музыкальной композиции. Как она работает? Сначала Вконтакте ищет пользователей, у которых в плейлистах есть эта песня. Затем находятся музыкальные композиции, которые наиболее часто встречаются у найденных пользователей вместе с этой песней. Эти музыкальные композиции и предлагаются пользователю для прослушивания. Поиск пользователей с похожими вкусами реализуется с помощью коллаборативной фильтрации.

## **Яндекс.Музыка**

### **Выявление предпочтений**

Один из способов выявления предпочтений пользователя, который используется в Яндекс.Музыке, это подсчёт числа прослушиваний песни пользователем. По истории прослушиваний выясняется, какие жанры и исполнители нравятся пользователю [4].

Чтобы определить, что пользователю нравится больше или меньше, существуют оценки «Нравится» и «Не нравится». Пользователь может поставить оценку «Нравится» треку, альбому или исполнителю. Музыкальным композициям, которые пользователь посчитал плохими, можно поставить оценку «Не нравится» в жанровом радио и в радио по исполнителю.

Дополнительно к этому учитываются также пропуски треков при прослушивании и добавления треков в плейлисты. Помимо учёта всех действий пользователя ещё происходит ранжирование этих действий в порядке важности. Например, среди положительных действий оценка “Мне нравится” имеет наибольший вес.

## **Построение прогноза**

Алгоритм даёт предсказания на основе собранной информации о музыкальных предпочтениях пользователя. После каждого прослушивания песни и некоторых других действий собранная информация обновляется и изменяется прогноз. Помимо этого на основе информации о пользователях выявляются люди со схожими вкусами, что позволяет одному человеку советовать то, что понравилось другому человеку со схожими предпочтениями.

## **Составление рекомендаций**

Полученный прогноз по предпочтениям пользователя ещё «разбавляется» полезной информацией, такой как музыка, которую слушают друзья, композиции с недавнего фильма или концерта, а также музыкальные композиции, которые рекомендует ваш любимый исполнитель. В Яндексе создали свой метод машинного обучения — Матрикснет, который обрабатывает все имеющиеся данные и на их основе формирует список музыкальных произведений, которые должны понравиться пользователю. При составлении рекомендаций алгоритм принимает во внимание множество факторов, вплоть до времени суток у пользователя, так как в разное время суток может нравиться разная музыка.

## Apple music

В Apple Music существует раздел «Для вас», в котором вам рекомендуются музыкальные композиции по вашим предпочтениям [5]. При этом учитываются такие параметры, как:

- **лайки** — можно ставить лайки композициям или целым плейлистам;
- **прослушивания** — после прослушивания музыкальных композиций Apple music предлагает похожих исполнителей в этом жанре. Учитываются только полные прослушивания, что может свидетельствовать о том, что пользователю понравилась песня;
- **ваша медиатека** — здесь учитываются как купленные в iTunes Store музыкальные композиции, так и добавленные в медиатеку iTunes с диска или другого носителя;
- **ваши указания** — в начале работы пользователя в Apple music система выясняет, какие жанры и исполнители вам больше нравятся. Позже это возможно изменить.

В процессе определения музыкальных композиций определённого жанра, которые следует порекомендовать пользователю, принимают участие эксперты из разных музыкальных журналов, что выгодно отличает данную рекомендательную систему от остальных.



## Постановка задачи

Так или иначе, все современные рекомендательные системы дают рекомендации на основе оценок пользователей или атрибутов предметов, таких как жанр, исполнитель и так далее. При этом не учитывается само содержание музыкальных композиций. Целью данной работы является разработка рекомендательной системы, которая предлагает к прослушиванию музыкальные композиции с учётом их содержания, то есть на основе амплитудно-частотной схожести музыкальных произведений.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Получение амплитудно-частотного спектра музыкальной композиции.
2. Определение параметров амплитудно-частотного спектра, по которым будет определяться их схожесть.
3. Определение алгоритма, который будет задавать шаги по нахождению похожих спектров.
4. Анализ полученных рекомендаций с целью определения успешности выбранного подхода.

# Глава 1. Теория

## 1.1 Получение амплитудно-частотного спектра

Гармонические колебания —  $x(t) = A \sin(\omega t + \phi)$  где  $A$  - амплитуда колебания,  $\omega$ (радиан/с) – угловая частота,  $\phi$  – начальная фаза колебаний. Как известно, большинство сигналов можно представить суммой гармонических колебаний, то есть в виде ряда Фурье. Из ряда Фурье можно получить непрерывное преобразование Фурье для непериодических сигналов [6]. Преобразование Фурье задаётся в виде

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t} dt$$

Значение  $F(\omega)$  однозначно задаётся  $|F(\omega)|$  и  $argF(\omega)$ .

В настоящее время звуковые сигналы представляются в памяти компьютера как массивы чисел. Процесс преобразования сигнала в цифровой вид представлен на рисунке 1.

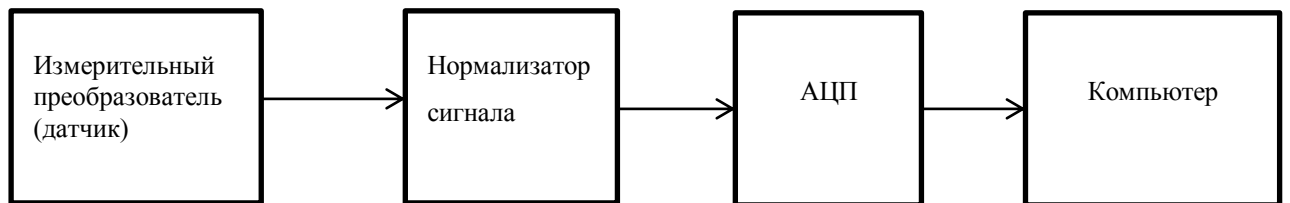


Рисунок 1 Схема оцифровки сигнала

Звуковой сигнал, измеренный с помощью измерительного преобразователя, попадает на АЦП (аналого-цифровой преобразователь), который переводит аналоговый сигнал в цифровой вид [7]. Далее значения сигнала берутся в дискретные моменты времени и сохраняются в памяти компьютера. Наглядное представление этого процесса дано на рисунке 2.

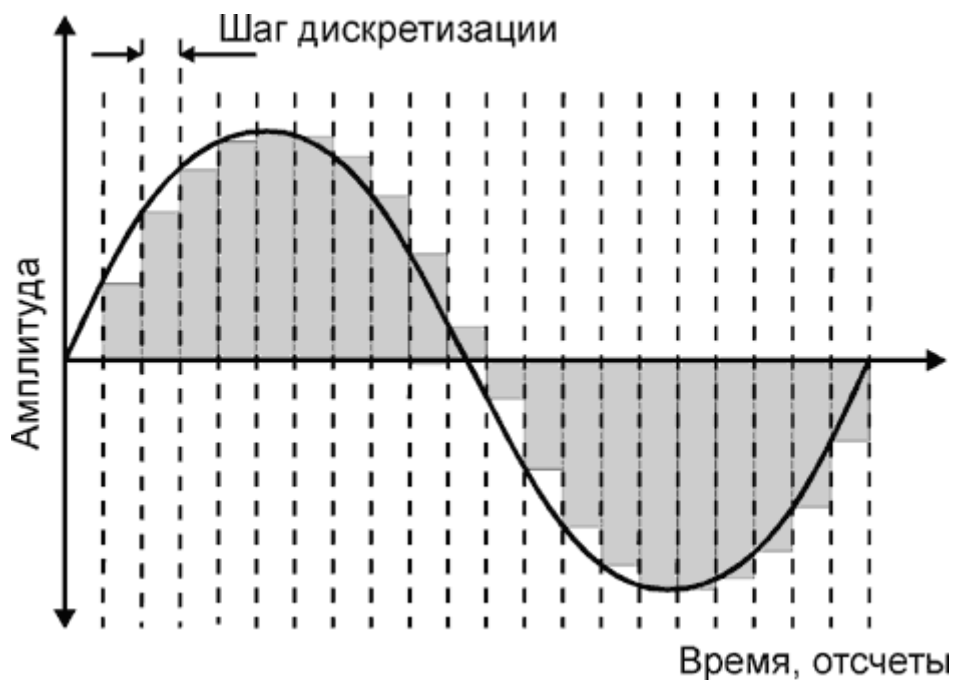


Рисунок 2 Взятие дискретных отсчётов сигнала за период времени  $T$

Количество взятых отсчётов сигнала за одну секунду называется частотой дискретизации. Частота дискретизации очень важна в процессе оцифровки сигнала. Теорема Найквиста – Шеннона говорит о том, что для того, чтобы непрерывный сигнал мог быть точно сохранён и воспроизведён, частота дискретизации должна в два раза превышать максимальную частоту в спектре сигнала. Если частота дискретизации будет меньше, чем указано в теореме Найквиста – Шеннона, все сигналы более высокой частоты будут понижаться. Это явление называют эффектом «алиасинга». На рисунке 3 наглядно представлено это явление.

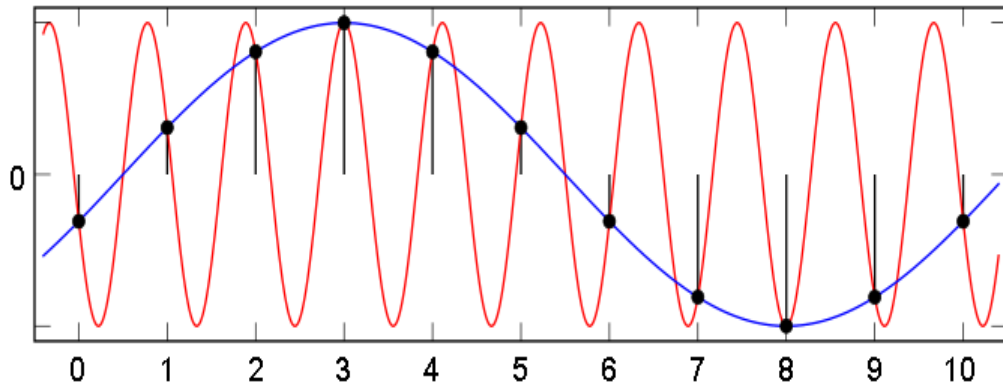


Рисунок 3 Эффект «алиасинга»

В задачах цифровой обработки сигналов применяется быстрое преобразование Фурье — алгоритм быстрого вычисления дискретного преобразования Фурье. На его основе, зная  $N$  значений сигнала в дискретные моменты времени, можно получить  $N$  комплексных амплитуд, с помощью которых можно получить амплитудно-частотный спектр сигнала.

Дискретное преобразование Фурье задаётся в виде:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i k n}{N}} \quad k = 0, \dots, N - 1,$$
 где  $N$  — число взятых отсчётов,  $x_n$  — дискретные отсчёты сигнала,  $X_n$  — комплексные амплитуды.

## 1.2 Определение параметров для классификации композиций

Музыкальная композиция хранится в компьютере в виде отсчётов сигнала, взятых в дискретные моменты времени с определённой частотой дискретизации. Обозначим частоту дискретизации как  $\nu$ . Каждый такой файл делится на  $M$  частей (снимков) длиной  $N$  отсчётов. К каждой части применяется быстрое преобразование Фурье. Подавая на вход быстрому преобразованию Фурье  $N$  значений сигнала  $x_n$ , получаем на выходе  $N$  комплексных значений  $X_n$ , где  $|X_n|$  — амплитуда  $n$ -ой гармоники. Частоты гармоник изменяются от

0 до  $\nu$  с шагом  $\frac{\nu}{N}$ . Так как спектр сигнала получается симметричным почти относительно середины, в полученном дискретном спектре используется только первая половина его отсчетов.

После получения вектора из амплитуд для каждой части необходимо сократить число амплитуд, с которыми производится работа, для ускорения работы алгоритмов классификации, поэтому производится сжатие в  $K$  раз числа отсчётов. Далее для каждой частоты по соответствующим ей значениям амплитуд в снимках считаются статистические характеристики, такие как математическое ожидание, среднеквадратичное отклонение, медиана, коэффициенты эксцесса и асимметрии [8]. Таким образом, каждая композиция представляется вектором из  $\frac{5N}{2K}$  параметров. По этим параметрам далее будет происходить классификация композиций. В качестве алгоритмов классификации были выбраны support vector machine (SVM) [9],  $k$ -nearest neighbors ( $k$ -NN) [10] и decision tree (DT) [11]. Эти методы работают с пространством признаков и с успехом применяются при распознавании изображений [12] и в других областях.

## 1.3 Алгоритмы классификации

В методах классификации каждая музыкальная композиция представляется вектором с  $p = \frac{5N}{2K}$  параметрами.

### 1.3.1 Метод опорных векторов

Метод опорных векторов (support vector machine, SVM) — набор алгоритмов, основанных на обучении с учителем, которые могут применяться в

задаче классификации. В этом методе все точки должны разделяться на два класса гиперплоскостью размерностью  $p-1$ . Чтобы её найти, через крайние точки обоих классов проводятся гиперплоскости, а разделяющая гиперплоскость находится между ними посередине. Гиперплоскости выбираются таким образом, чтобы между ними не лежали точки обоих классов. Для лучшей классификации максимизируют ширину этой полосы, то есть расстояние между ближайшими точками разных классов. Положим, что точки имеют вид:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\},$$

где  $y_i$  равны 1 или -1 в зависимости от класса  $x_i$ . Разделяющая гиперплоскость задаётся в виде:

$$v \cdot x + c = 0,$$

где  $v$  — вектор нормали к гиперплоскости,  $c$  — скаляр. Гиперплоскости, на которых лежат крайние точки двух классов — опорные вектора, задаются уравнениями:

$$v \cdot x + c = 1, \quad v \cdot x + c = -1.$$

Ширина полосы между ними равна  $\frac{2}{\|v\|}$ , поэтому для её максимизации минимизируется  $\|v\|$ . Точки вне полосы задаются как:

$$y_i(v \cdot x_i + c) \geq 1, \quad i = \overline{1, n}$$

Для поиска оптимальной гиперплоскости решается система уравнений:

$$\begin{cases} \|v\| \rightarrow \min, \\ y_i(v \cdot x_i + c) \geq 1, \quad i = \overline{1, n} \end{cases} \quad (1)$$

Если выборка линейно неразделима, то позволяют ошибки на обучающей выборке. Для этого вводятся дополнительные величины  $\varepsilon_i$  равные величине ошибки на точках  $x_i \quad i = \overline{1, n}$ . Тогда система (1) запишется как:

$$\begin{cases} \|v\| + C \sum_{i=1}^n \varepsilon_i \xrightarrow{v, c, \varepsilon_i} \min, \\ y_i(v \cdot x_i + c) \geq 1, \quad i = \overline{1, n}, \\ \varepsilon_i \geq 0, \quad i = \overline{1, n} \end{cases} \quad (2)$$

Параметр  $C$  необходимо подбирать. Также в случае линейно неразделимой выборки применяется так называемый «kernel trick». При этом происходит переход в пространство более высокой размерности, где точки оказываются линейно разделимыми. При этом скалярное произведение векторов в новом пространстве представляется функцией векторов в исходном пространстве. Вот примеры наиболее часто используемых ядер, которые применялись при анализе:

*Radial basis function:*  $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0;$

*Sigmoid:*  $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + c);$

*Exponential Chi2:*  $K(x_i, x_j) = e^{-\gamma \chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = \frac{(x_i - x_j)^2}{(x_i + x_j)}, \gamma > 0;$

*Inter:*  $K(x_i, x_j) = \min(x_i, x_j).$

### 1.3.2 Метод ближайших соседей

**Метод ближайших соседей** — алгоритм классификации, основанный на определении сходства между объектами. В этом алгоритме классифициру-

мый объект получает тот класс, объектов которого больше среди ближайших к нему в обучающей выборке.

В методе  **$k$  ближайших соседей** (*k-nearest neighbors*, *k-NN*) объекту присваивается тот класс, членов которого больше среди  $k$  ближайших к нему объектов. В моей задаче у музыкальных композиций всего два класса, поэтому чтобы не было одинакового числа объектов разных классов, лучше брать число  $k$  нечётным.

Обозначим за  $X$  — объекты, а  $Y$  — классы объектов.  $X^l = (x_i, y_i)_{i=1}^l$  — обучающая выборка, состоящая из пар объект и его класс,  $c$  — число классов.

Для определения меры близости объектов задаётся функция расстояния:

$$p: X \times X \rightarrow \{1, c\}.$$

Объекты являются похожими, если значение этой функции мало.

В моей задаче в качестве функции расстояния я использовал Евклидово расстояние:

$$p(x^1, x^2) = \sqrt{\sum_{i=1}^n (x_i^1 - x_i^2)^2},$$

где  $x^1 = (x_1^1, \dots, x_n^1)$ ,  $x^2 = (x_1^2, \dots, x_n^2)$  — объекты выборки.

Для произвольного объекта  $v$  его класс  $k(v)$  определяется как:

$$k(v) = \arg \max_{y \in Y} \sum_{i=1}^l [y(x_i) = y] w(i, v),$$



Где  $y(x_i)$  обозначает класс объекта в обучающей выборке,  $l$  — число ближайших объектов в обучающей выборке,  $w(i, v)$  — вес  $i$ -го соседа при классификации объекта  $v$ . В моём случае  $w(i, v) = 1, \quad i = \overline{1, l}$ .

В алгоритме  $k$  ближайших соседей остро стоит вопрос в определении этого параметра  $k$ , ведь от его выбора зависят результаты классификации.

Пример:

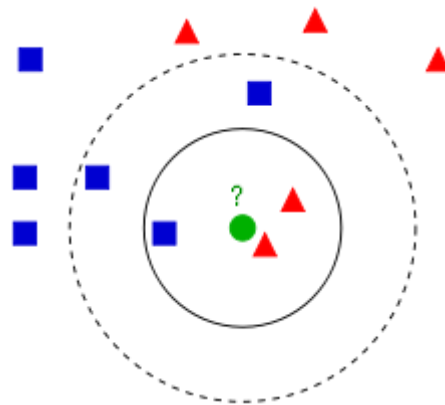


Рисунок 4 k-NN

Из рисунка 4 видно, что в зависимости от выбора  $k$  зелёный кружок может быть классифицирован как красный треугольник в случае  $k = 3$ , либо как синий квадрат, если  $k = 5$ .

Если  $k = 1$ , то алгоритм даёт неправильные результаты при наличии объектов-выбросов: алгоритм ошибается как на самих объектах-выбросах, так и на ближайших к ним соседях. В случае, когда  $k = l$ , всем объектам классификации присваивается один и тот же класс, что тоже неправильно. На практике параметр  $k$  подбирается скользящим контролем.

### 1.3.3 Решающее дерево

Решающее дерево (decision tree, DT) — алгоритм классификации, в работе которого используется дерево принятия решений. В моей работе использовалось бинарное решающее дерево. В этом дереве каждой внутренней вершине  $w \in W_{\text{внутр}}$  поставлен в соответствие предикат  $p_w: X \rightarrow \{0,1\}$ ,  $p \in P$ , каждому листу поставлена в соответствие метка класса  $c_w \in Y$ . В процессе классификации каждый объект  $x \in X$  спускается от корня до листа в соответствии с Алгоритмом 1 и получает свой класс.

Алгоритм 1:

- 1:  $w = w_0$ ;
- 2: **пока**  $w \in W_{\text{внутр}}$
- 3:   **если**  $p_w(x) = 1$  **то**
- 4:     переход вправо:  
       $w := R_w$ ;
- 5:   **иначе**
- 6:     переход влево:  
       $w := L_w$ ;
- 7: **вернуть**  $c_w$

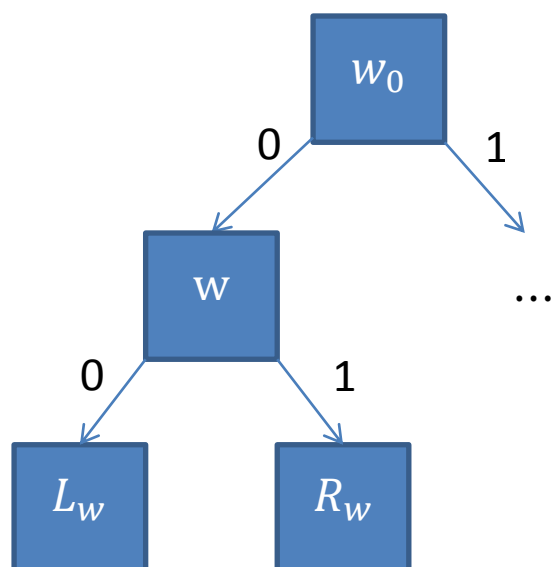


Рисунок 5 Схема дерева

Пусть  $F$  — множество терминальных вершин дерева. Обозначим за  $H_w(x)$  конъюнкцию, состоящую из всех предикатов, встреченных на пути от корня дерева до вершины  $w \in F$ . Тогда алгоритм классификации  $a: X \rightarrow Y$  запишется как:

$$a(x) = \arg \max_{y \in Y} \sum_{\substack{w \in F \\ c_w = y}} H_w(x).$$

## Критерии информативности предикатов

На каждом шаге алгоритма выбирается предикат с максимальной информативностью, то есть такой, при котором подвыборка, приходящая в данный лист, максимально разделяется на классы. В качестве предикатов наиболее часто выступают те, в которых определяется, больше ли порога значение признака объектов в выборке или нет.

Примеры критериев информативности:

1. Отделение одного класса:

В этом случае выбирается предикат, который наилучшим образом отделяет объекты одного класса от остальных.

$$I(p, X) = \max_{c \in Y} I_c(p, X).$$

2. Критерий Джини:

Выбирается предикат, при котором наибольшее число пар объектов одного класса идут вместе в один узел дерева.

$$I(p, X) = \text{count}\{(x_i, x_j): p(x_i) = p(x_j) \text{ и } y_i = y_j\}.$$

3.  $D$ -критерий В. И. Донского:

Здесь выбирается предикат, при котором наибольшее число пар объектов разных классов идут в разные узлы дерева.

$$I(p, X) = \text{count}\{(x_i, x_j): p(x_i) \neq p(x_j) \text{ и } y_i \neq y_j\}.$$

## Построение дерева по выборке

### Алгоритм ID3 (Induction of Decision Tree)

В этом алгоритме исходная выборка начинает делиться на части и продолжает делиться до тех пор, пока в каждой подвыборке не будут присутствовать объекты только одного класса. Этот алгоритм ниже записан в виде процедуры ID3, которая строит дерево по обучающей выборке  $S$ .  $P$  — множество предикатов.

- 1: **ПРОЦЕДУРА**  $ID3(U)$ ;
- 2: **если** все объекты в  $S$  имеют один класс  $y \in Y$  **то**
- 3:     создать новый лист  $w$ ;
- 4:      $c_w := y$ ;
- 5:     **вернуть**  $(w)$ ;
- 6: найти предикат с максимальной информативностью:  
$$p = \arg \max_{p \in P} I(p, S)$$
- 7: разделить выборку  $S$  на две части  $S_1$  и  $S_2$  по предикату  $p$ :  
$$S_1 := \{x \in S: p(x) = 0\}$$
  
$$S_2 := \{x \in S: p(x) = 1\}$$
- 8: **если**  $S_1 = \emptyset$  или  $S_2 = \emptyset$  **то**

- 9:     создать новый лист  $w$ ;
- 10:     $c_w :=$  класс, объектов которого больше в  $S$ ;
- 11: **иначе**
- 12:     создать новую внутреннюю вершину  $w$ ;
- 13:      $p_w = p$ ;
- 14:      $L_w := ID3(S_1)$ ; (строится левое поддерево)
- 15:      $R_w := ID3(S_2)$ ; (строится правое поддерево)
- 16: **вернуть** ( $w$ );

## Алгоритм С4.5

В моей работе применяется алгоритм С4.5, который является улучшенной версией алгоритма ID3. Одно из улучшений состоит в том, что происходит усечение тех ветвей, в которые вообще не попадают объекты или их очень мало. Пусть  $X^k$  — контрольная выборка, которая необходима для проверки работы алгоритма после обучения.

Процедура обрезания дерева:

- 1: **для всех**  $w \in W_{\text{внутр}}$
- 2:      $Q_w :=$  подмножество объектов  $X^k$ , дошедших до  $w$ ;
- 3:     **если**  $Q_w = \emptyset$  **то**
- 4:         вернуть новый лист  $w$ ,  $c_w :=$  мажоритарный класс( $S$ );

5: выбираем действие, чтобы ошибка при классификации  $Q_w$  была минимальной:

сохранить поддерево  $w$ ;

заменить поддерево  $w$  поддеревом его левой веткой  $L_w$ ;

заменить поддерево  $w$  поддеревом его правой веткой  $R_w$ ;

заменить поддерево  $w$  листом  $c_w$  так, чтобы ошибка была минимальна.

## Глава 2. Практическое исследование

### 2.1 Используемые технологии

Описанный выше подход реализован в виде программного обеспечения на языке программирования C#. C# является одним из самых популярных компилируемых языков программирования со статической типизацией. Статическая типизация позволяет значительно облегчить написание программы, так как это помогает отлавливать ошибки типов на этапе компиляции. Компилируемость программ на этом языке позволяет ускорить их работу, что может быть достаточно важным при получении амплитудно-частотного спектра музыкальных композиций и работе алгоритмов классификации. Также нельзя не отметить тот факт, что C# поддерживается программной платформой .NET Framework, что открывает доступ к огромному числу библиотек, в частности к библиотекам для работы со звуком и машинного обучения, которые используются в моей работе.

Для чтения звуковых файлов использовалась библиотека Naudio. Naudio написана на C#, поэтому хорошо подошла к моему проекту. Naudio предоставляет широкий набор возможностей для работы со звуком и хорошо протестирована для быстрого выполнения своих задач. Naudio имеет открытый исходный код, хорошую документацию и много tutorиалов, что облегчает работу с библиотекой.

Машинное обучение в моей программе производилось с использованием Emgu CV [13]— кроссплатформенной .NET «обёртки» библиотеки OpenCV и Accord.NET [14] — .NET фреймворка для машинного обучения. При классификации использовалась реализация SVM из Emgu CV, позволяющая автоматически подбирать параметр  $C$ , где для этого используется скользящий контроль. Реализации алгоритмов k-NN и решающего дерева

были взяты из Accord.NET. В отличие от библиотеки Emgu CV, больше предназначенной для работы с изображениями, Accord.NET предоставляет значительно больший функционал для машинного обучения.

## 2.2 Классификация

Работа классификатора производилась на 3 выборках размеченных песен от трёх разных людей. Все песни были помечены двумя классами: «нравится» и «не нравится». Первый тренировочный набор состоял из 240 песен разных жанров, проверочный набор состоял из 80 песен. Композиции с меткой «нравится» имели жанры: классика, альтернативный рок, психоделический рок, альтернативный метал. Не нравившиеся песни представлены в таких жанрах, как тяжёлый рок и метал, дабстеп, поп, шансон. Второй человек предпочитает песни таких жанров, как альтернативный рок, альтернативный метал, трип-хоп, прогрессивный рок. Негативно относится к жанрам дабстеп, поп, шансон, хип-хоп. Тренировочный набор состоял из 30 песен. Проверка производилась на 10 композициях. В третьей выборке понравившиеся композиции имеют такие жанры, как джаз, свинг, рок-н-рол. Не нравившиеся композиции были выбраны из подборки русской популярной музыки. Размеры тренировочной и проверочной выборок такие же, как и в предыдущем случае.

Рассмотрим возможные ситуации в процессе работы классификатора. Гипотеза состоит в том, что объект принадлежит к основному или первому классу (объект нравится пользователю).

Случаи:



**true positive (TP)** — гипотеза принята правильно,

**false positive (FP)** — гипотеза принята неправильно,

**false negative (FN)** — гипотеза отвергнута неверно,

**true negative (TN)** — гипотеза отвергнута правильно.

Качество работы классификаторов оценим с помощью следующих величин [15]:

- *Точность (precision)* — часть правильно распознанных объектов основного класса среди всех объектов, отнесённых к первому классу

$$precision = \frac{TP}{TP + FP}$$

- *Полнота (recall)* — часть правильно определённых объектов первого класса среди всех объектов первого класса, содержащихся в тестовой выборке

$$recall = \frac{TP}{TP + FN}$$

- *F-мера* — комбинирует показатели точности и полноты

$$F_{\beta} = (1 + \beta^2) \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

Если приоритет отдаётся точности, то  $0 < \beta < 1$ , а при  $\beta > 1$  приоритет отдаётся полноте. Обычно используют показатели  $F_{0.5}$  и  $F_2$ .

## 2.3 Перекрёстная проверка

В процессе перекрёстной проверки происходит разбиение всей выборки на обучающую и проверочную части различными способами. Для каждого разбиения происходит тренировка алгоритма на обучающей выборке и вы-

числение ошибки алгоритма на проверочной выборке. Оценка работы алгоритма является усреднённой ошибкой по всем разбиениям [16].

Пусть  $X$  — множество объектов,  $Y$  — множество классов объектов. Обозначим всю выборку, на которой осуществляется перекрёстная проверка, за  $X^P$ .  $X^P$  делится  $N$  различными способами на тренировочную и проверочную выборки. Эти выборки между собой не пересекаются:  $X^P = X_n^t \cup X_n^R$ , где  $X_n^t$  — тренировочная подвыборка, состоящая из  $t$  элементов,  $X_n^R$  — проверочная подвыборка, содержащая  $R = P - t$  элементов,  $n = 1, \dots, N$  — номер разбиения. Обозначим за  $a$  алгоритм классификации, который по произвольной выборке  $X^t$  определяет класс элементов  $x \in X^t$ . Качество работы классификатора оценим с помощью функционала качества  $F(a, X^t)$ :

$$F(a, X^t) = \frac{1}{t} \sum_{x_i \in X^t} L(a(x_i), y_i),$$

Где  $L(a(x_i), y_i)$  — функция потерь, определяющая величину ошибки алгоритма  $a$  при правильном ответе  $y_i$ .

В процессе перекрёстной проверки по каждому разбиению всей выборки на тренировочную и проверочную задаётся алгоритм  $a_n(X_n^t)$  и вычисляется значение функционала качества  $F_n = F(a_n, X_n^R)$ . *Оценкой перекрёстной проверки* называется среднее значение по всем  $F_n$ :

$$Q(a, X^P) = \frac{1}{N} \sum_{n=1}^N F(a(X_n^t), X_n^R).$$

В зависимости от способа деления исходной выборки на части выделяют различные варианты перекрёстной проверки. Перекрёстная проверка может применяться в методе  $k$  ближайших соседей для подбора параметра  $k$  и в методе опорных векторов для подбора параметра  $c$ .

## 2.3.1 Варианты перекрёстной проверки

### Полная перекрёстная проверка (complete cross-validation)

При этом варианте в процессе перекрёстной проверки производится  $N = C_P^t$  разбиений,  $t$  — длина тренировочной выборки. В случае, когда  $t \geq 2$  число разбиений очень большое и из-за этого долго выполняется перекрёстная проверка. На практике этот вариант перекрёстной проверки используется редко.

### Контроль по отдельным объектам (leave-one-out cross-validation (LOO))

Это частный случай полной перекрёстной проверки при  $t = 1$ , и,  $N = P$ . Это один из самых распространённых вариантов перекрёстной проверки.

Преимуществами LOO является то, что каждый объект всего один раз выступает в качестве проверочного и размер обучающих подвыборок не сильно отличается от размера полной выборки. Недостатком LOO является то, что процесс обучения происходит  $P$  раз.

### Перекрёстная проверка по $k$ блокам ( $k$ -fold cross-validation)

В этом случае выборка разделяется на  $k$  непересекающихся блоков равной длины  $m_1, \dots, m_k$ :

$$X^P = X^{m_1} \cup \dots \cup X^{m_k},$$

$m_1 + \dots + m_k = P$ . На каждом блоке по очереди происходит проверка, при этом обучение производится на оставшихся  $k - 1$  блоках. Оценка перекрёстной проверки вычисляется как средняя ошибка на контрольных подвыборках:

$$Q(a, X^P) = \frac{1}{k} \sum_{n=1}^k F(a(X^P \setminus X^{m_n}), X^{m_n}).$$

Достоинством этого вида перекрёстной проверки является то, что обучение проводится всего  $k$  раз, тогда как длина обучающей подвыборки не сильно отличается от длины полной выборки. Наиболее часто производится разделение выборки на 5, 10 или 15 блоков.

### 2.3.2 Результаты перекрёстной проверки

В моей работе производилась перекрёстная проверка для подбора параметра  $k$  в методе  $k$  ближайших соседей. Перекрёстная проверка производилась на обучающей выборке, а оценка результатов осуществлялась с помощью ранее введённых метрик, таких как *точность*, *полнота*, *F-мера* на проверочной выборке. В процессе перекрёстной проверки для каждого возможного  $k$  происходило разбиение тренировочной выборки различными способами, зависящими от типа перекрёстной проверки.

Для первой выборки:

|                             | <b>LOO</b> | <b>5-fold CV</b> | <b>15-fold CV</b> |
|-----------------------------|------------|------------------|-------------------|
| <b>Presicion</b>            | 0,76       | 0,8              | 0,76              |
| <b>Recall</b>               | 0,57       | 0,6              | 0,57              |
| <b><math>F_{0,5}</math></b> | 0,72       | 0,75             | 0,71              |

|                      |     |      |     |
|----------------------|-----|------|-----|
| $F_2$                | 0,6 | 0,63 | 0,6 |
| <b>BestK</b>         | 8   | 11   | 15  |
| <b>WorstSumError</b> | 84  | 119  | 98  |
| <b>Error</b>         | 24  | 22   | 24  |

Для второй выборки:

|                      | <b>LOO</b> | <b>10-fold CV</b> | <b>15-fold CV</b> |
|----------------------|------------|-------------------|-------------------|
| <b>Presicion</b>     | 0,5        | 0,5               | 0,5               |
| <b>Recall</b>        | 0,6        | 0,6               | 0,6               |
| $F_{0,5}$            | 0,51       | 0,51              | 0,51              |
| $F_2$                | 0,57       | 0,57              | 0,57              |
| <b>BestK</b>         | 3          | 2                 | 3                 |
| <b>WorstSumError</b> | 10         | 10                | 10                |
| <b>Error</b>         | 5          | 5                 | 5                 |

Скользящий контроль определил лучшие  $k$  равные 2 и 3, но результаты при них не очень хорошие.

Для третьей выборки:

|                  | <b>LOO</b> | <b>10-fold CV</b> | <b>15-fold CV</b> |
|------------------|------------|-------------------|-------------------|
| <b>Presicion</b> | 1          | 1                 | 1                 |
| <b>Recall</b>    | 1          | 1                 | 1                 |

|                      |   |   |   |
|----------------------|---|---|---|
| $F_{0,5}$            | 1 | 1 | 1 |
| $F_2$                | 1 | 1 | 1 |
| <b>BestK</b>         | 3 | 3 | 3 |
| <b>WorstSumError</b> | 7 | 7 | 7 |
| <b>Error</b>         | 0 | 0 | 0 |

Скольльзящий контроль единогласно определил лучшее  $k = 3$ .

В процессе этой перекрёстной проверки **WorstSumError** считалась как число несовпадений ответа классификатора с классом объекта на всех разделениях. В качестве **BestK** выбиралось то, при котором **WorstSumError** была наименьшей. **Error** это число неправильных ответов на проверочной выборке с подобранным лучшим  $k$ . Эти таблицы отражает только лучшие исходы, так как часто **Precision** было невозможно подсчитать, так как  $TP$  и  $FP$  были равны 0. Нужно отметить, что не всегда находилось такое  $k$ , при котором алгоритм давал лучшие результаты.

## 2.4 Результаты

Ниже приведены результаты классификации для трёх выборок, описанных выше. Первая выборка содержала 320 песен.

SVM (Emgu CV)

$k$ -fold — число частей, на которые делится обучающая выборка при скользящем контроле для подбора параметров.

$k$ -fold = 10

|                             | <b>Chi2</b> | <b>Inter</b> | <b>Rbf</b> | <b>Sigmoid</b> |
|-----------------------------|-------------|--------------|------------|----------------|
| <b>C</b>                    | 312,5       | 0,1          | 2,5        | 0,1            |
| <b>Precision</b>            | 0,64        | 0,63         | 0,65       | 0,57           |
| <b>Recall</b>               | 0,67        | 0,72         | 0,6        | 0,65           |
| <b><math>F_{0,5}</math></b> | 0,6         | 0,65         | 0,64       | 0,58           |
| <b><math>F_2</math></b>     | 0,67        | 0,7          | 0,6        | 0,63           |

Все ядра показали похожие результаты.

$k$ -fold = 15

|          | <b>Chi2</b> | <b>Inter</b> | <b>Rbf</b> | <b>Sigmoid</b> |
|----------|-------------|--------------|------------|----------------|
| <b>C</b> | 62,5        | 0,1          | 0,5        | 0,1            |

|                             |      |      |      |      |
|-----------------------------|------|------|------|------|
| <b>Precision</b>            | 0,65 | 0,63 | 0,63 | 0,57 |
| <b>Recall</b>               | 0,65 | 0,72 | 0,68 | 0,65 |
| <b><math>F_{0,5}</math></b> | 0,65 | 0,65 | 0,64 | 0,58 |
| <b><math>F_2</math></b>     | 0,65 | 0,7  | 0,67 | 0,63 |

*k*-NN (Accord.NET)

|                             |                |             |             |
|-----------------------------|----------------|-------------|-------------|
|                             | <b>K=33,37</b> | <b>K=39</b> | <b>K=11</b> |
| <b>Precision</b>            | 0,88           | 0,87        | 0,8         |
| <b>Recall</b>               | 0,55           | 0,52        | 0,6         |
| <b><math>F_{0,5}</math></b> | 0,79           | 0,77        | 0,75        |
| <b><math>F_2</math></b>     | 0,59           | 0,57        | 0,63        |

Метод *k* ближайших соседей хорошо себя показал.

Решающее дерево

|                  |               |                             |                         |
|------------------|---------------|-----------------------------|-------------------------|
| <b>Precision</b> | <b>Recall</b> | <b><math>F_{0,5}</math></b> | <b><math>F_2</math></b> |
| 0,7              | 0,65          | 0,69                        | 0,65                    |

Второй тренировочный набор содержал 30 песен разных жанров, проверочный набор состоял из 10 песен.



## SVM

 $k$ -fold = 10

|                             | <b>Chi2</b> | <b>Inter</b> | <b>Rbf</b> | <b>Sigmoid</b> |
|-----------------------------|-------------|--------------|------------|----------------|
| <b>C</b>                    | 2,5         | 0,1          | 12,5       | 12,5           |
| <b>Precision</b>            | 1           | 0,8          | 1          | 0,5            |
| <b>Recall</b>               | 0,8         | 0,8          | 0,2        | 0,6            |
| <b><math>F_{0,5}</math></b> | 0,95        | 0,8          | 0,55       | 0,51           |
| <b><math>F_2</math></b>     | 0,83        | 0,8          | 0,23       | 0,57           |

Лучше всего себя показали ядра **Chi2** и **Inter**. В целом результаты лучше, чем на первой выборке.

 $k$ -fold = 15

|                             | <b>Chi2</b> | <b>Inter</b> | <b>Rbf</b> | <b>Sigmoid</b> |
|-----------------------------|-------------|--------------|------------|----------------|
| <b>C</b>                    | 12,5        | 0,1          | 12,5       | 12,5           |
| <b>Precision</b>            | 1           | 0,8          | 1          | 0,4            |
| <b>Recall</b>               | 0,4         | 0,8          | 0,2        | 0,4            |
| <b><math>F_{0,5}</math></b> | 0,76        | 0,8          | 0,55       | 0,4            |
| <b><math>F_2</math></b>     | 0,45        | 0,8          | 0,23       | 0,4            |

Результаты остались прежними или ухудшились.

*k*-NN

|                             | <b>K=11</b> | <b>K=10</b> | <b>K=12</b> | <b>K=1,2,3</b> |
|-----------------------------|-------------|-------------|-------------|----------------|
| <b>Precision</b>            | 1           | 0,66        | 0,75        | 0,5            |
| <b>Recall</b>               | 0,2         | 0,8         | 0,6         | 0,6            |
| <b><math>F_{0,5}</math></b> | 0,55        | 0,69        | 0,71        | 0,52           |
| <b><math>F_2</math></b>     | 0,23        | 0,76        | 0,625       | 0,58           |

Результаты ухудшились по сравнению с первой выборкой.

Решающее дерево

| <b>Precision</b> | <b>Recall</b> | <b><math>F_{0,5}</math></b> | <b><math>F_2</math></b> |
|------------------|---------------|-----------------------------|-------------------------|
| 0,5              | 0,8           | 0,54                        | 0,71                    |

Ухудшилась точность, но улучшилась полнота.

Песни третьего человека также были разделены 30 на 10.

SVM

*k*-fold = 10, 15

|                  | <b>Chi2</b> | <b>Inter</b> | <b>Rbf</b> | <b>Sigmoid</b> |
|------------------|-------------|--------------|------------|----------------|
| <b>C</b>         | 62,5        | 0,1          | 2,5        | 0,5            |
| <b>Precision</b> | 0,83        | 0,83         | 0,83       | 0,625          |

|               |      |      |      |      |
|---------------|------|------|------|------|
| <b>Recall</b> | 1    | 1    | 1    | 1    |
| $F_{0,5}$     | 0,86 | 0,86 | 0,86 | 0,67 |
| $F_2$         | 0,96 | 0,96 | 0,96 | 0,89 |

Ядро **Sigmoid** показало худшие результаты.

$k$ -NN

|                  |              |                |
|------------------|--------------|----------------|
|                  | <b>K=3,4</b> | <b>K=5,6,8</b> |
| <b>Precision</b> | 1            | 1              |
| <b>Recall</b>    | 1            | 0,8            |
| $F_{0,5}$        | 1            | 0,95           |
| $F_2$            | 1            | 0,84           |

При найденных  $k$  метод  $k$  ближайших соседей показал себя очень хорошо.

Решающее дерево

|                  |               |           |       |
|------------------|---------------|-----------|-------|
| <b>Precision</b> | <b>Recall</b> | $F_{0,5}$ | $F_2$ |
| 0,8              | 0,8           | 0,8       | 0,8   |

Решающее дерево также показало лучшие результаты, чем на прошлых выборках.

## Выводы

По полученным результатам на всех трёх выборках можно заключить, что в большинстве случаев метод  $k$  ближайших соседей является лучшим среди алгоритмов классификации. Минусом этого алгоритма является то, что для каждой выборки нужно подбирать свой параметр  $k$ . Лучше всего алгоритм сработал, когда в тренировочной и проверочной выборке были музыкальные композиции похожих жанров. Метод опорных векторов хуже себя показывает на больших выборках, в которых содержатся композиции разных жанров. В моём случае композиции достаточно сильно отличались по звучанию, то есть выборка содержала много «шумовых» выбросов. Так как метод опорных векторов не устойчив к «шумовым» выбросам, по-видимому, это и явилось причиной ухудшения результатов. Хуже всего смог определить класс музыкальных композиций алгоритм классификации решающее дерево. Его не стоит использовать в этой задаче.

## Заключение

В настоящее время существует достаточно много рекомендательных систем музыки, которые позволяют пользователю быстро и удобно получать доступ к той музыке, которая удовлетворяет именно его вкусам. Однако исходя из отзывов других пользователей и своего опыта, можно заключить, что не все системы советуют именно ту музыку, которая нравится пользователю. Большинство подобных систем ищут пользователей, которым нравятся те же песни, что и вам, и предлагают к прослушиванию их другие музыкальные композиции. Этот подход может давать не лучшие результаты, так как трудно найти пользователей с одинаковыми плейлистами. Поэтому нет никаких гарантий, что то, что понравилось одному, понравится и другому. Необходимо ещё учитывать звучание голоса и музыки в музыкальной композиции. В данной работе проводилось исследование возможности рекомендаций музыкальных композиций на основе схожести их амплитудно-частотного спектра.

Для определения похожих композиций использовался подход на основе машинного обучения, в частности, алгоритмы классификации. В этих алгоритмах каждый объект представляется вектором признаков, по которым далее происходила классификация и присваивание объекту определённого класса. Для представления музыкальных композиций в подходящем для классификаторов виде использовались статистические характеристики, которые вычислялись по амплитудно-частотному спектру композиций.

В выбранных алгоритмах классификации есть неопределённые параметры, которые необходимо подбирать. Для этой цели зачастую применяют перекрёстную проверку, которая была опробована для метода  $k$  ближайших

соседей. Оценка работы классификаторов производилась с помощью общепринятых для этой задачи метрик, таких как точность, полнота и F-мера.

Так как музыкальные предпочтения у всех людей разные, не существует одной универсальной выборки для работы алгоритмов классификации. Поэтому в моём исследовании были взяты три выборки песен разных людей. Полученные результаты показали, что не существует одного алгоритма классификации, который бы давал одинаково хорошие результаты для разных людей, но можно в каждом случае такой алгоритм подобрать. Из полученных результатов также можно сделать вывод, что гораздо лучше классифицируются песни одного или нескольких похожих жанров. Например, на основе результатов по третьей выборке можно заключить, что джаз и рок-н-рол хорошо отделяются от современной попсы. Стоит заметить, что на всех трёх выборках данный подход показал неплохие результаты. В дальнейшем необходимо расширять тренировочную и проверочную выборки, чтобы сделать окончательные выводы по применимости данного подхода.

## Список литературы

1. Melville P., Mooney R., Nagarajan R. Content-Boosted Collaborative Filtering for Improved Recommendations // University of Texas, USA : Материалы конф. / AAAI-02, Austin, TX, USA, 2002. — 2002. P. 187-192.
2. James Bennett, Stan Lanning The Netflix Prize [Электронный ресурс] // Proceedings of KDD Cup and Workshop 2007, San Jose, California, Aug 12, 2007. URL: [http://www.netflixprize.com/assets/NetflixPrizeKDD\\_to\\_appear.pdf](http://www.netflixprize.com/assets/NetflixPrizeKDD_to_appear.pdf) (дата обращения: 01.03.16).
3. Вконтакте. Похожие аудиозаписи [Электронный ресурс]: URL: [https://vk.com/page-2158488\\_49041638](https://vk.com/page-2158488_49041638) (дата обращения: 2.04.16).
4. Как это работает? Рекомендации в Яндекс.Музыке [Электронный ресурс]: URL: <https://yandex.ru/blog/company/92883> (дата обращения: 2.04.16).
5. Как работают рекомендации в Apple Music и как им помочь [Электронный ресурс]: URL: <http://appleinsider.ru/tips-tricks/kak-rabotayut-rekomendacii-v-apple-music-i-kak-im-pomoch.html> (дата обращения: 2.04.16).
6. Сергиенко А. Б. Цифровая обработка сигналов. Спб: Питер, 2006. 751 с.
7. Применение преобразования Фурье для анализа сигналов [Электронный ресурс]: URL: <https://habrahabr.ru/post/269991/> (дата обращения: 7.04.16).
8. HOLO [Электронный ресурс]: URL: <https://habrahabr.ru/post/161005/> (дата обращения: 9.04.16).

9. Вьюгин В. В. Математические основы машинного обучения и прогнозирования. М.: 2013. 387 с.
10. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. Springer, 2009.
11. Воронцов К. В. Лекции по логическим алгоритмам классификации. 2007. 53 с.
12. Гришкин В.М., Власов Д.Ю., Жабко А.П., Ковшов А.М., Щигорец С.Б., Якушкин О.О. Система распознавания биологических загрязнителей поверхности памятников культурного наследия // Устойчивость и процессы управления: Материалы III международной конференции. 2015. С. 569-570.
13. Emgu CV [Электронный ресурс]: URL: [http://www.emgu.com/wiki/index.php/Main\\_Page](http://www.emgu.com/wiki/index.php/Main_Page) (дата обращения: 04.03.16).
14. Accord.NET [Электронный ресурс]: URL: <http://accord-framework.net/> (дата обращения: 26.04.16).
15. Маннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск. М.: Вильямс, 2011. 512 с.
16. Скользящий контроль [Электронный ресурс]: URL <http://www.machinelearning.ru/wiki/index.php?title=Кросс-валидация> (дата обращения: 26.04.16).



# Приложение

Ниже продемонстрировано применение трёх алгоритмов классификации, используемых в моём исследовании.

SVM:

В этом алгоритме по очереди для каждого из ядер: Chi2, Inter, Rbf, Sigmoid происходит обучение на тренировочной выборке с автоматическим подбором параметров для лучшей классификации. После классификации производится оценка результатов на проверочной выборке.

```
#region svm
SVM.SvmKernelType[] emguKernels = { SVM.SvmKernelType.Chi2, SVM.SvmKernelType.Inter,
SVM.SvmKernelType.Rbf, SVM.SvmKernelType.Sigmoid };
double[] presicions = new double[emguKernels.Length];
double[] recalls = new double[emguKernels.Length];
double[] f1 = new double[emguKernels.Length];
double[] f2 = new double[emguKernels.Length];
double[] C = new double[emguKernels.Length];
float[] responses = new float[checkSize];

int kFolds = 10; //число частей для перекрёстной проверки
for (int k = 0; k < emguKernels.Length; k++)
{
    using (SVM model = new SVM())
    {
        model.SetKernel(emguKernels[k]);
        model.Type = SVM.SvmType.CSvc; //со штрафным множителем C
        model.TermCriteria = new MCvTermCriteria(100, 0.00001); //критерий остано-
ки тренировки

        TrainData td = new TrainData(trainDataEmgu, Em-
gu.CV.ML.MlEnum.DataLayoutType.RowSample, trainClassesEmgu);
        bool trained = model.TrainAuto(td, kFolds); //обучение алгоритма вместе с
подбором параметров

        for (int i = 0; i < checkSize; ++i)
        {

            responses[i] = model.Predict(checkDataEmgu.GetRow(i));

        }
        tp = fp = fn = 0;
        for (int i = 0; i < checkSize; ++i)
        {

            if (checkClasses[i] == 1)
            {
                if (responses[i] == 1)
                {
                    tp += 1;
                }
            }
        }
    }
}
```

```

        }
        else fn += 1;
    }
    if ((checkClasses[i] == -1) && (responses[i] == 1))
    {
        fp += 1;
    }
}
C[k] = model.C;
presicions[k] = tp / (tp + fp);
recalls[k] = tp / (tp + fn);
f1[k] = 1.25 * (presicions[k] * recalls[k]) / (0.25 * presicions[k] + rec
alls[k]);
f2[k] = 5 * (presicions[k] * recalls[k]) / (4 * presicions[k] + re-
calls[k]);
}
}
#endregion

```

k-NN:

Работа алгоритма проводилась в двух вариантах: с перекрёстной проверкой и без.

В процессе перекрёстной проверки для каждого возможного  $k$  происходило разбиение тренировочной выборки различными способами, зависящими от параметра folds. Найденные  $k$ , при которых было обнаружено наименьшее число ошибок классификации, использовались в работе алгоритма для классификации песен из проверочной выборки. В случае без использования перекрёстной проверки просто производилась классификация и оценка её результатов при всех возможных  $k$ .

```

#region knn
//подбираем лучшее k скользящим контролем и потом классифицируем на проверочной выборке
Console.WriteLine();
Console.WriteLine("With cross-validation: enter y/n ");
ConsoleKeyInfo choice = Console.ReadKey();
if (choice.KeyChar == 'y')
{
    int kSize = 100;
    int[] kArray = new int[kSize];
    Dictionary<int, List<int>> bestKDict = new Dictionary<int, List<int>>
    int folds = 240;
    int bestK=0;
    int error = 0;
    int worstSumError = 60;
    double[] errors = new double[kSize];
}

```

```

for (int i = 0; i < kSize; ++i)
{
    kArray[i] = i + 1;
}
double[] fArray = new double[kSize];
for (int k = 0; k < kArray.Length; ++k)
{

    var sumError = customCrossValidationKNN(kArray[k], trainData, Classes, folds);
    if (sumError <= worstSumError)
    {
        bestK = kArray[k];
        worstSumError = (int)sumError;
        if (bestKDict.ContainsKey(worstSumError))
        {
            bestKDict[worstSumError].Add(bestK);
        }
        else {
            bestKDict.Add(worstSumError, new List<int>());
            bestKDict[worstSumError].Add(bestK);
        }
    }
}

for(int t=0; t < bestKDict[worstSumError].Count; ++t)
{
    bestK = bestKDict[worstSumError][t];
    KNearestNeighbors knn = new KNearestNeighbors(bestK, 2, trainData, train-
ClassesKNN);
    int[] answers = new int[checkSize];
    for (int i = 0; i < checkSize; i++)
    {
        answers[i] = knn.Compute(checkData[i]);
    }
    tp = fp = fn = 0;
    error = 0;
    for (int i = 0; i < checkSize; ++i)
    {

        if (checkClasses[i] == 1)
        {
            if (answers[i] == 1)
            {
                tp += 1;

            }
            else {
                fn += 1;
                error += 1;
            }
        }
        if ((checkClasses[i] == -1) && (answers[i] == 1))
        {
            fp += 1;
            error += 1;
        }
    }

    presicion = tp / (tp + fp);
}

```

```

        recall = tp / (tp + fn);
        f_1 = 1.25 * (presicion * recall) / (0.25 * presicion + recall);
        f_2 = 5 * (presicion * recall) / (4 * presicion + recall);
    }
}
else //просто перебираем k
{
    int kStart = 1;
    int kStep = 1;
    int kEnd = 100;
    List<int> kList = new List<int>();
    for (int i = kStart; i <= kEnd; i+=kStep)
    {
        kList.Add(i);
    }
    double[] presicions = new double[kList.Count];
    double[] recalls = new double[kList.Count];
    double[] f0_5 = new double[kList.Count];
    double[] f2 = new double[kList.Count];

    for(int k=0; k < kList.Count; ++k)
    {
        KNearestNeighbors knn = new KNearestNeighbors(kList[k], 2, trainData, train-
ClassesKNN);
        int[] answers = new int[checkSize];
        tp = fp = fn = 0;
        for (int i = 0; i < checkSize; i++)
        {
            answers[i] = knn.Compute(checkData[i]);
        }

        for (int i = 0; i < checkSize; ++i)
        {
            if (checkClasses[i] == 1)
            {
                if (answers[i] == 1)
                {
                    tp += 1;
                }
                else fn += 1;
            }
            if ((checkClasses[i] == -1) && (answers[i] == 1))
            {
                fp += 1;
            }
        }

        presicions[k] = tp / (tp + fp);
        recalls[k] = tp / (tp + fn);
        f0_5[k] = 1.25 * (presicions[k] * recalls[k]) / (0.25 * presicions[k] + re-
calls[k]);
        f2[k] = 5 * (presicions[k] * recalls[k]) / (4 * presicions[k] + recalls[k]);
    }
}
}
#endregion

```

## Решающее дерево:

```
#region decision tree

DecisionVariable[] variables = new DecisionVariable[5 * snapCompressionSamplesCount];
for (int i = 0; i < variables.Length; i++)
{
    variables[i] = new DecisionVariable(i.ToString(), new AForge.DoubleRange(-100, 100));
}

DecisionTree tree = new DecisionTree(variables, 2);
C45Learning teacher = new C45Learning(tree);
double error = teacher.Run(trainData, trainClassesKNN);
int[] answers = checkData.Apply(tree.Compute);
tp = fp = fn = 0;

for (int i = 0; i < checkSize; ++i)
{
    if (checkClasses[i] == 1)
    {
        if (answers[i] == 1)
        {
            tp += 1;
        }
        else fn += 1;
    }
    if ((checkClasses[i] == -1) && (answers[i] == 1))
    {
        fp += 1;
    }
}

presicion = tp / (tp + fp);
recall = tp / (tp + fn);
f_1 = 1.25 * (presicion * recall) / (0.25 * presicion + recall);
f_2 = 5 * (presicion * recall) / (4 * presicion + recall);

#endregion
```