О классификации изображений и музыкальных файлов

В. В. Осокин, Т. Д. Аипов, З. А. Ниязова

В работе рассматриваются две задачи — классификация музыкальных файлов по жанрам и классификация изображений. В первом случае требуется реализовать веб-приложение, которое определяет жанры загружаемых в него музыкальных файлов (джаз, классика, диско, метал, блюз). Во втором случае требуется реализовать веб-приложение с возможностью загрузки картинок-эталонов. При последующей загрузке картинок система должна определять, на какой из эталонов наиболее похожа загруженная картинка, и определять ее в соответствующий класс.

Ключевые слова: классификация, музыкальные файлы, mfcc, hog, php.

Введение

Данная работа посвящена разработке алгоритмов классификации музыкальных файлов по жанрам и классификации изображений.

Задача классификации музыкальных файлов по жанрам состоит в следующем. Дано множество музыкальных файлов, взятое с сайта [1]. Каждому файлу сопоставлен жанр. Используя данное множество в качестве обучающей выборки, требуется построить классификатор, который для вновь загруженного файла не из этого множества будет определять его жанр. Для этого нужно научиться парсить WAV-файлы, написать алгоритм построения вектора по музыкальному файлу, выбрать метрику для определения близости векторов, выбрать и реализовать метод машинного обучения для обучения классификатора на обучающей выборке, а также написать WEB-приложение, позволяющее загружать музыкальные файлы и сравнивать загруженные файлы с эталонами для определения их жанров.

Кроме задачи классификации музыкальных файлов в статье мы рассматриваем следующую задачу классификации изображений. Дано множество из 5 произвольных картинок. На вход подается картинка из этого множества, подверженная различным искажениям. Программа должна отнести картинку к своему оригиналу. Для этого нужно научиться считывать файлы изображений и представлять их в виде массива пикселей, написать алгоритм построения вектора по картинке, выбрать метрику для определения близости векторов, выбрать метод машинного обучения для обучения классификатора на обучающей выборке, а также реализовать WEB-приложение, позволяющее загружать изображение и сравнивать его с эталонами для выявления оригинала.

По всем перечисленным пунктам будут приведены полученные результаты.

Постановка задачи и полученные результаты

Нужно реализовать WEB-приложение на языках PHP и JavaScript, в котором должна быть реализована классификация музыкальных файлов по жанрам.

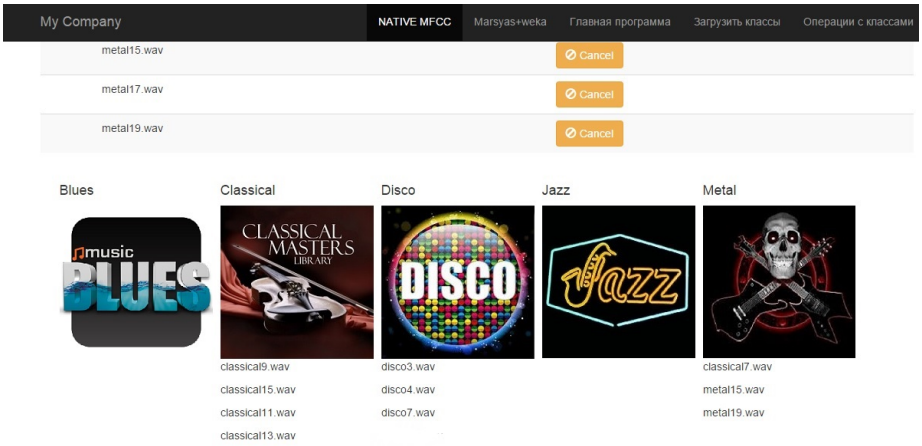

Рис. 1: Пример работы приложения. На вход подано 10 файлов, 9 из них классифицировано верно.

Для задачи классификации музыкальных жанров необходимо реализовать следующие подзадачи.

Первая подзадача состоит в парсинге WAV файла. На вход подается музыкальный файл формата WAV представляющий из себя бинарные данные. Для дальнейшей работы удобно представить данный файл в виде массива амплитуд. Требуется реализовать класс для работы с WAV на языке программирования PHP [2]. На вход классу подается файл, а на выходе получаем массив, состоящий из амлитуд длины–частота дескритизации умноженой на продолжительность музыкального файла (раздел «Первичная обработка wav-файлов»).

Вторая подзадача состоит в создании алгоритма построения вектора по музыкальному файлу. Данные вектора мы будем использовать для сравнения музыкальных файлов друг с другом. Авторы, исследующие данное направление, сосредотачивают усилия на извлечении частотной характеристики речевого тракта человека, отбрасывая при этом характеристики сигнала возбуждения. Для отделения сигнала возбуждения от сигнала речевого тракта прибегают к так называемому кепстральному анализу. Необходимо реализовать алгоритм параметризации основанный на MFCC коэффициентах [3] на языке программирования PHP. На вход которого подается массив аплитуд, а на выходе получаем массив мелкепстральных коэффициентов (раздел «Составление вектора признаков по музыкальному файлу»).

Третья подзадача состоит в выборе метрики, позволяющей определять близость векторов. В данной работе выбрана метрика хи-квадрат. Требуется реализовать метрику хи-квадрат на языке программирования PHP [4] (раздел «Классификация музыкальных файлов собственным алгоритмом»).

Четвертая подзадача состоит в выборе метода машинного обучения для обучения классификатора по обучающей выборке. Требуется реализовать алгоритм основанный на методе эталонов [5] (раздел «Классификация музыкальных файлов собственным алгоритмом»). Также для сравнения с полученными результатами использована библиотека для работы с алгоритмами машинного обучения Weka (раздел «Классификация музыкальных файлов при помощи систем Marsyas и Weka»).

Пятая подзадача состоит в написании WEB-приложения, позволяющего загружать музыкальные файлы и сравнивать загруженные файлы с эталонами для определения их жанров. Требуется реализо-

вать приложение на языке программирования PHP [6] (раздел «Классификация музыкальных файлов собственным алгоритмом»).

Таблица 1: Полученные результаты.

Алгоритм	Верно	Не верно	Успех(%)
SVM	187	13	93.5
Random forest	183	17	91.5
Байесовский классификатор	178	22	89

Скриншот приложения по классификации музыкальных файлов приведен на рис. 1.

В статье решается еще одна задача: нужно реализовать WEB-приложение на языках PHP и JavaScript, в котором должен быть реализован метод машинного обучения на обучающей выборке и классификация изображений, с целью нахождение оригинала загруженного изображения. Приложение также должно реализовывать пользовательский интерфейс для администрирования изображений.

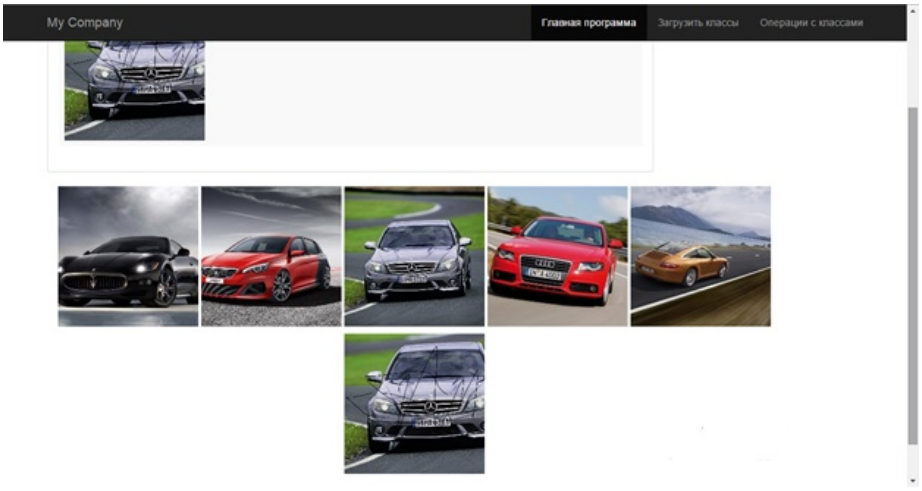

Рис. 2: Пример распознавания. Загруженная картинка была отнесена к классу правильно.

Первая подзадача состоит в считывании файла изображения для последующей работы с ним как с массивом пикселей. Для работы

с изображениями был реализован класс на языке программирования PHP, который считывает изображение возвращает массив пикселей [7] (раздел «Первичная обработка изображения»).

Вторая подзадача состоит в создании алгоритма построения вектора по картинке. Эти векторы мы будем использовать для сравнения картинок друг с другом. Для построения векторов был выбран алгоритм HOG [8] (раздел «Составление вектора признаков по изображению»).

Третья подзадача состояла в выборе метрики. Для этого была выбрана метрика хи-квадрат (раздел «Классификация музыкальных файлов собственным алгоритмом»).

Четвертая подзадача состоит в выборе машинного обучения для обучения по обучающей выборке. Для этой подзадачи был выбран метод ближайших соседей.

Пятая подзадача состоит в написании WEB-приложения, позволяющего загружать изображение и сравнивать его с эталонами для выявления оригинала [9] (раздел «Классификация изображений»).

Скриншот приложения по классификации изображений приведен на рис. 2.

Первичная обработка wav-файлов

Итак, рассмотрим самый обычный WAV файл (Windows PCM). Он состоит из двух частей. Одна из них — заголовок файла, другая — область данных. В заголовке файла хранится информация о размере файла, количестве каналов, частоте дискретизации, количестве бит в сэмпле (эту величину еще называют глубиной звучания).

Для большего понимания смысла величин в заголовке следует еще рассказать об области данных и оцифровке звука. Звук состоит из колебаний, которые при оцифровке приобретают ступенчатый вид. Этот вид обусловлен тем, что компьютер может воспроизводить в любой короткий промежуток времени звук определенной амплитуды (громкости) и этот короткий момент далеко не бесконечно короткий. Продолжительность этого промежутка и определяет частота дискретизации. Например, у нас файл с частотой дискретизации 44.1 kHz, это значит, что тот короткий промежуток времени равен 1/44100 се-

кунды (следует из размерности величины Гц = 1/с). Современные звуковые карты поддерживают частоту дискретизации до 192 kHz.

Амплитуда выражается числом, занимаемым в памяти (файле) 8, 16, 24, 32 бит (теоретически можно и больше). Как известно, 8 бит = 1 байту, следовательно, какая-то одна амплитуда в какой-то короткий промежуток времени в памяти (файле) может занимать 1, 2, 3, 4 байта соответственно. Таким образом, чем больше число занимает места в памяти (файле), тем больше диапазон значений для этого числа, а значит и для амплитуды. 1 байт — от 0 до 255, 2 байта — от 0 до 65 535, 3 байта — от 0 до 16 777 216, 4 байта — от 0 до 4 294 967 296.

В моно варианте значения амплитуды расположены последовательно. В стерео же, например, сначала идет значение амплитуды для левого канала, затем для правого, потом снова для левого и так далее.

Совокупность амплитуды и короткого промежутка времени носит название сэмпл.

Класс WavParse предназначен для работы с бинарными данными, считывает файл формата WAV и представляет его в виде массива амплитуд [2].

Рассмотрим конструктор **__construct** класса в нем происходит считывание .wav файла с проверкой его на читаемость. Далее производится проверка корректности файла. Размер файла не может быть меньше размера заголовков.

Затем, последовательно сдвигаясь по заголовкам, считываются данные в массив для дальнейшей работы с объектом.

После получения всех заголовков проверяется, является ли это моно или стерео файл и в зависимости от этого выбирается число пропускаемых амплитуд для оптимизации.

Далее считывается непосредственно сам массив амплитуд, пропуская некоторое количество амплитуд для оптимизации памяти. На этом рассмотрение функции **__construct** заканчивается.

Ниже приводятся методы для работы с бинарными данными. Реализованы методы преобразования бинарных данных в строку — readString, в числа — readLong и в слова — readWord. Кроме того реализована функция распаковки бинарных данных — readUnpacked.

Составление вектора признаков по музыкальному файлу

В качестве информативных признаков использовались мел-частотные кепстральные коэффициенты (МЧКК). Данные признаки широко применяются в задачах распознавания речи. Они основаны на двух ключевых понятиях: кепстр и мел-шкала. Кепстр (cepstrum) — это результат дискретного косинусного преобразования от логарифма амплитудного спектра сигнала. Формально:

$$c(n) = DCT\{\log(|F\{f(t)\}|^2)\}$$

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.

MFCC (mel frequency cepstral coefficients) — класс, который по массиву амплитуд строит вектор признаков [3].

Рассмотрим главную функцию **getMfcc**: используется метод скользящего окна, начало каждого последующего фрейма приходится на середину предыдущего. Проходим в цикле по всему массиву амплитуд $s[t]$. Далее $s[t]$ разбивается на K фреймов по N отсчетов, пересекающихся на $1/2$ длины:

$$s[t] \to S_n[t], n = 1, \ldots, K$$

Далее последовательно для каждого фрейма производятся различные преобразования описанные ниже. **Fourier** — вызываем дискретное преобразования Фурье для получения спектра исходного сигнала, далее вызываем **setCenters** для нахождения центральных частот треугольных фильтров, затем вызываем **setSmp** — функция, которая переводит центры треугольных фильтров в номера отсчета спектральной плотности мощности сигнала $P_n[k]$, после вызываем функцию **setXn**, которая вычисляет энергию текущего фрейма setXn и

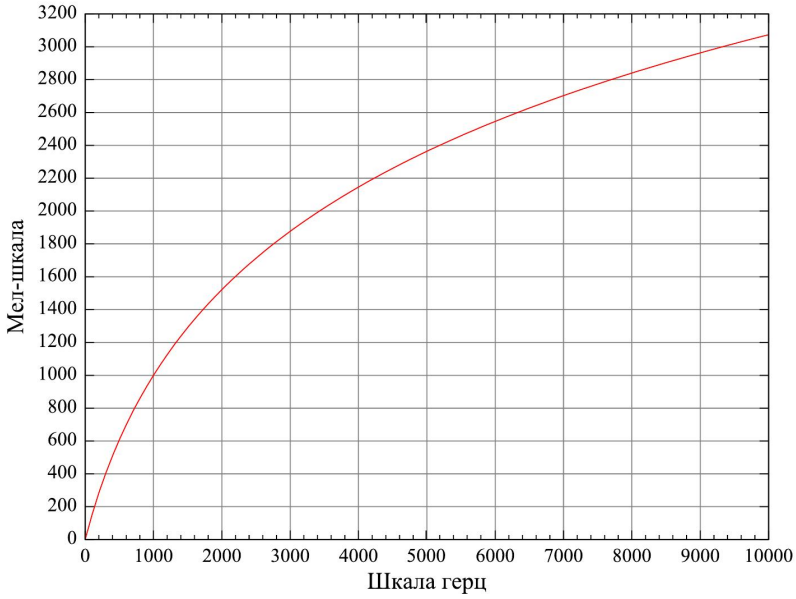

Рис. 3

последнее вызов функции **getCepstral** — вычисляет сами mfcc коэффициенты.

Первым делом нам нужен спектр исходного сигнала, который получается с помощью преобразования Фурье. Для каждого фрейма во всех амплитудах высчитывается дискретное преобразование Фурье.

$$ReX_n[k] = \frac{2}{N}\sum_{i=1}^{N} S_n[t]\cos(\frac{2\pi k(i-1)}{N}), k = 1,\ldots,M, M = \frac{N}{2},$$

$$ImX_n[k] = -\frac{2}{N}\sum_{i=1}^{N} S_n[t]\sin(\frac{2\pi k(i-1)}{N}), k = 1,\ldots,M, M = \frac{N}{2}.$$

Находится спектральная плотность мощности сигнала (фрейма) по полученным выше значениям:

$$P_n[k] = A_n[k]^2,$$
$$A_n[k] = \sqrt{ReX_n[k]^2 + ImX_n[k]^2}.$$

Рассмотрим функцию **Fourier**, вычисляющую дискретное преобразование Фурье. Зануляются переменные и осуществляется проход по циклу, высчитывая сумму. Аргументами функции являются input — текущий фрейм, k — индекс элемента фрейма. Вычисляется действительная и мнимая часть преобразования Фурье. Возвращается сумма квадратов действительной и мнимой части. На этом мы заканчиваем описание данной функции.

После получения спектра нужно расположить его на мел-шкале. Для этого мы используем фильтры, равномерно расположенные на мел-оси. Необходима функция, которая равномерно расположит эти фильтры. Нужно задать количество фильтров P, а также начальную частоту f_l и конечную частоту f_h. Далее понадобится перевести частоты в мелы.

$$f_l{}^m = \hat{f}_{mel}(f_l),$$
$$f_h{}^m = \hat{f}_{mel}(f_h).$$

Рассмотрим функцию **herz2mel** перевода из Герц в Мелы аргументом функции является величина в Герцах. Перевод происходит по следующей формуле.

$$\hat{f}_{mel}(f_{hz}) = 1127ln(1 + \frac{f_{hz}}{700}).$$

На этом описание функции **herz2mel** заканчивается.

Рассмотрим функцию **mel2herz** перевода из Мелов в в Герцы. Аргументом функции является величина в Мелах. Перевод величин происходит по следующей формуле.

$$\hat{f}_{hz}(f_{mel}) = 700(e^{\frac{f_{mel}}{1127}} - 1).$$

На этом описание функции **mel2herz** заканчивается.

Перейдем к рассмотрению функции **setCenters**, находящей центральные частоты треугольных фильтров. Объявляется массив хранящий центральные частоты. Начало и конец выбраного диапазона частот переводятся в мелы. Далее происходит задание центров частот в мел-шкале. В нашем случае начало — flStart = 300, а конец flEnd = 8000, filterCount — количество фильтров. На мел-шкале отрезок $[f_l{}^m, f_h{}^m]$ разбивается на $P + 1$ равных непересекающихся подотрезков $[f_j{}^m, f_{j+1}{}^m], j = 1 \dots P+1$ длины $len = \frac{f_h{}^m - f_l{}^m}{P+1}$. Далее находятся их центры

$$C^m[i] = {f_l}^m + i \cdot len, i = 1 \dots P$$

и переводятся в шкалу Герц. Затем необходимо вычислить центральные частоты треугольных фильтров

$$C[i] = \hat{f}_{hz}(C^m[i]), i = 1 \dots P$$

На этом описание функции **setCenters** заканчивается. Если перевести этот график (рис. 4) в частотную шкалу, можно увидеть такую картину (рис. 5).

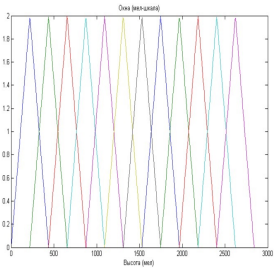

Рис. 4: Мел-шкала.

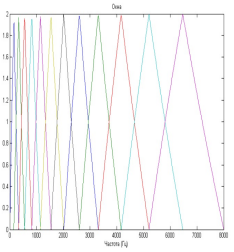

Рис. 5: Частота (Герц).

На этом графике заметно, что фильтры «собираются» в области низких частот, обеспечивая более высокое «разрешение» там, где оно необходимо для распознавания.

Центры треугольных фильтров переводятся из Гц в номера отсчетов массива $P_n[k]$. Рассмотрим функцию **setSmp**, где scaleCentre — массив центров полученный функцией setCenters, sampleRate — исходная частота дискретизации.

$$f_{smp}[i] = \frac{M}{F_s} C[i], i = 1 \dots P.$$

F_s — это частота дискретизации исходного сигнала. На этом мы заканчиваем рассмотрение функции **setSmp**.

Перемножением векторов спектра сигнала и оконной функции найдем энергию сигнала, которая попадает в каждый из фильтров анализа. Мы получили некоторый набор коэффициентов, но это еще

не те MFCC, которые мы ищем. Пока их условно можно назвать Мелчастотными спектральными коэффициентами. Возводим их в квадрат и логарифмируем. Нам осталось только получить из них кепстральные, или «спектр спектра». Для этого нам нужно применить дискретное косинусное преобразование.

Рассмотрим функцию **Hik**. Фильтр $H_i[k]$ Fsmp-массив, полученный функцией SetSmp. Функция реализует представленный ниже алгоритм

$$H_i[k] = \begin{cases} 0 & , \quad k < f_{smp}[i-1] \\ \frac{(k-f_{smp}[i-1])}{f_{smp}[i]-f_{smp}[i-1]} & , \quad f_{smp}[i-1] \le k \le f_{smp}[i] \\ \frac{(f_{smp}[i+1]-k)}{f_{smp}[i+1]-f_{smp}[i]} & , \quad f_{smp}[i] \le k \le f_{smp}[i+1] \\ 0 & , \quad k > f_{smp}[i+1] \end{cases}$$

На этом рассмотрение функции **Hik** заканчивается.

Далее рассмотрим функцию **setXn** получения фильтров отсчета спектральной плотности мощности сигнала, которая умножает его на специальный фильтр Hik, и происходит взятие логарифма от результата.

$$X_n[i] = \sum_{k=1}^{M} P_n[k] H_i[k], i = 1 \dots P,$$
$$X_n[i] = \ln(X_n[i]), i = 1 \dots P$$

Зануляются переменные суммы и массив фильтров отсчета. Проходим в цикле по количеству фильтров и по числу элементов во фрейме. Значение $P_n[k]$ умножаем на соответствующий фильтр $H_i[k]$.

От полученный суммы берется логарифм. На этом рассмотрение функции **setXn** заканчивается.

Разберем функцию **getCepstral**. Получение кепстральных коэффициентов . После высчитывания коэффициентов $X_n[k]$ берется дискретное косинусное преобразование.

$$C_n[j] = \sum_{k=1}^{P} X_n[k] cos(j(k - \frac{1}{2})\frac{\Pi}{P}), i = 1 \dots P, j = 1 \dots J$$

$C_n[j]$ — массив кепстральных коэффициентов, J — желаемое число коэффициентов ($J < P$)

Необходимо пройти в цикле по количеству коэффициентов coefCount и количеству фильтров filterCount

Далее высчитывается дискретное косинусное преобразования Фурье. На этом рассмотрение функции **getCepstral** заканчивается.

Классификация музыкальных файлов собственным алгоритмом

Определяется функция **chisqr** расстояния между векторами. В данной работе была выбрана метрика Хи-квадрат [4]. На этом рассмотрение данной функции заканчивается.

Метод центроид

Рассмотрим функцию **centroid** вычисляющую центроиду [5]. Считываем количество векторов, длину векторы и объявляем массив в котором будем хранить среднее значение.

Проходим циклом по количеству векторов и длине вектора. Вычисляем среднее значение и возвращаем его. На этом рассмотрение функции **centroid** заканчивается.

Музыка — полученные результаты

На базе алгоритма MFCC было написано WEB-приложение, определяющее жанр методом центроид (рис. 1). Для вычисления центроид выбрано 50 из 100 файлов каждого жанра. Для тестированя подавались другие 50 файлов. Тестовое множество было скачано с сайта marsyas.info. Алгоритм MFCC в связке с выбранным алгоритмом классификации показал от 70% до 85% успешного распознавания в зависимости от жанра. Лучший результат в 85% был достигнут на множестве классической музыки [6].

Классификация музыкальных файлов при помощи систем Marsyas и Weka

Marsyas (Анализ музыки, поиск и синтез аудиосигналов) — фреймворк с открытым исходным кодом. Данная система была разработана Джорджем Цанетакисом с помощью студентов и исследователей со всего мира. Marsyas был использован для различных проектов, как в академических, так и в промышленных кругах.

Weka — представляет собой набор средств визуализации и алгоритмов для интеллектуального анализа данных и решения задач прогнозирования, вместе с графической пользовательской оболочкой для доступа к ним.

Weka позволяет выполнять такие задачи анализа данных, как подготовка данных (preprocessing), отбор признаков (англ. feature selection), кластеризация, классификация, регрессионный анализ и визуализация результатов.

Для анализа музыки использовалась консольная оболочка библиотеки Marsyas.

Для получения вектора признаков нужно перевести файл (множество файлов) в специальный формат *.mf

Это делается с помощью команды

```
ls *.wav > test.mf
```

Для того, чтобы получить файл с множеством признаков, надо подать на вход программе классы (.mf файлы) с помощью команды “bextract”. В результате чего получим файл MARSY_EMPTYtest.arff, который содержит специальный формат. Этот формат мы можем подавать на вход Weka для дальнейшего анализа.

```
bextract -sv disco.mf jazz.mf -w test.arff
```

Приведем команду для создания обученной модели методом SVM для дальнейшего распознавания тестовых данных

```
java -Xmx1G -cp \/opt\/weka-3-6-12\/weka.jar weka.
    classifiers.trees.J48 -C 0.25\\ -M -t MARSYAS\
    _EMPTYgen.arff -d j48.model \\
```

Эта команда создает обученную модель. В данном случае используется алгоритм деревьев решений.

Далее нам нужно протестировать нашу обученную модель. Это делается следующей командой:

```
java -Xmx1G -cp \/opt\/weka-3-6-12\/weka.jar weka.
    classifiers.trees.J48 -T \\ MARSYAS\_EMPTYtest.arff -l
     smo.model -p 0
```

Создав и протестировав модель, были получены следующие результаты (тестовое множество было скачано с сайта marsyas.info).

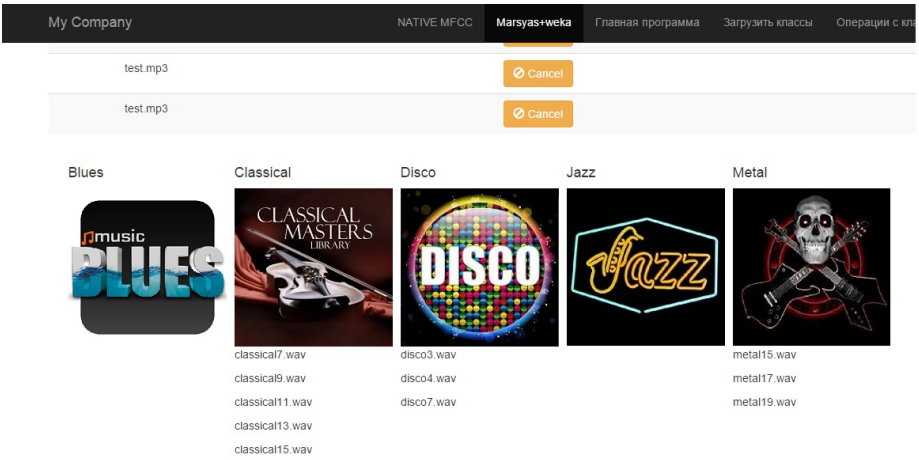

Рис. 6: Пример работы приложения. На вход поданы 11 файлов и все из них классифицированы верно.

Таблица 2: Полученные результаты.

Алгоритм	Верно	Не верно	Успех(%)
SVM	187	13	93.5
Random forest	183	17	91.5
Байесовский классификатор	178	22	89

На основе модели SVM был построен онлайн классификатор музыки.

Первичная обработка изображения

Для попиксельной обработки изображений нам понадобятся удобные инструменты для работы с картинками. Для этого была разработана небольшая библиотека на языке программирования PHP, работающая с JPEG и PNG изображениями. Библиотека может выдавать на выходе черно-белое изображение либо массив пикселей, представляющий изображение в GrayScale (оттенки серого). Класс ImageObj позволяет работать с изображением как с массивом пикселей [7].

Рассмотрим конструктор **construct**, который считывает изображение и определяет его размеры. Функция **imagecreatefromjpeg**

возвращает идентификатор изображения, полученного из файла с заданным именем. Устанавливается количество столбцов width и строк height в массиве, затем вызывается функция **setPixArray**, которая возвращает изображение в виде массива пикселей. На этом описание данной функции заканчивается.

RGB (аббревиатура английских слов Red, Green, Blue — красный, зелёный, синий) — аддитивная цветовая модель, как правило, описывающая способ синтеза цвета для цветовоспроизведения.

Изображение считывается попиксельно. Затем идет получение координат, полученные координаты присваиваются объекту Pixel. Рассмотрим функцию **setPixArray** считывающую значение координат RGB. Для каждой координаты изображения получаем значение R, G, B координат. И возвращаем массив пикселей. На этом рассмотрение функции **setPixArray** заканчиваем.

Преобразование изображения из цветного в оттенки серого (grayscale). Яркость пикселя grayscale-изображения рассчитывается по следующей формуле:

$Y = 0.299R + 0.587G + 0.114B$

Перейдем к описанию функции **getGreyScale** получения массива GrayScale (оттенков серого). Возвращает массив значений от 0 до 255 Во время прохода по массиву присваиваем его элементам значения по вышеопределенной формуле. На этом заканчиваем описание функции **getGreyScale**.

Перейдем к описанию функции **getGrayImage** Опишем функцию получения черно-белой фотографии. Создается новая картинка с теми же параметрами и в качестве RGB координат записывается его «серость». Функция imagecolorallocate(resource $image,int $red,int $green,int $blue) возвращает идентификатор цвета в соответствии с заданными RGB координатами. На этом заканчиваем описание функции **getGrayImage**.

Составление вектора признаков по изображению

Алгоритм HOG (Histogram of Oriented Gradients) позволяет получить вектор признаков по изображению. Основной идеей алгоритма является допущение, что внешний вид и форма объекта на участ-

ке изображения могут быть описаны распределением градиентов интенсивности или направлением краев. Реализация этих дескрипторов может быть произведена путем разделения изображения на маленькие связные области, именуемые ячейками, и расчетом для каждой ячейки гистограммы направлений градиентов или направлений краев для пикселов, находящихся внутри ячейки. Комбинация этих гистограмм и является дескриптором. Для увеличения точности локальные гистограммы подвергаются нормализации по контрасту. С этой целью вычисляется мера интенсивности на большем фрагменте изображения, который называется блоком, и полученное значение используется для нормализации. Нормализованные дескрипторы обладают лучшей инвариантностью по отношению к освещению.

Алгоритм HOG имеет несколько преимуществ над другими дескрипторами. Поскольку HOG работает локально, метод поддерживает инвариантность геометрических и фотометрических преобразований, за исключением ориентации объекта. Подобные изменения появятся только в больших фрагментах изображения. Более того, как обнаружили Далал и Триггс, грубое разбиение пространства, точное вычисление направлений и сильная локальная фотометрическая нормализация позволяют игнорировать движения пешеходов, если они поддерживают вертикальное положение тела. Дескриптор HOG, таким образом, является хорошим средством нахождения людей на изображениях. Перейдем к рассмотрению класса Hog [8].

Изображение разбивается на блоки 8x8 пикселей и для каждого блока вычисляется гистограмма, а также используется метод скользящего окна (блоки накладываются друг на друга).

Свертка с фильтрами Собеля. Сначала опишем основные понятия.

Понятие свертки изображений проиллюстрировано на приведенном рисунке (рис. 7).

Рассмотрим функцию **getGx** оператора Собеля, которая вычисляет производную яркости изображения в горизонтальном направлении.

Для каждого пикселя изображения вычисляется производная яркости. На этом рассмотрение функции **getGx** заканчиваем.

Рассмотрим функцию **getCell** для последовательного выделения клеток и сглаживания краев. Аргументами функции является координаты верхнего левого пикселя клетки. Далее выбираются клетки

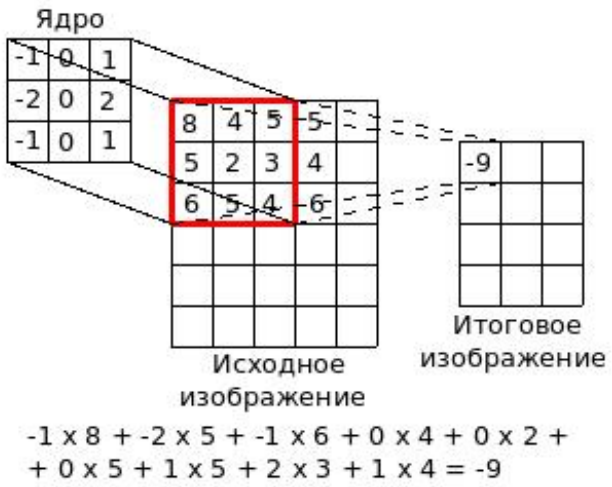

Рис. 7: Пример работы оператора Собеля.

соседние пикселю и диагональные клетки, тем самым получается 9 координат то есть массив размера 3x3. На этом рассмотрение функции **getCell** заканчиваем.

Вычисление гистограмм градиентов. Для начала вся область изменения направления градиента (например, $[0, 2\Pi]$) разбивается на некоторое (порядка 8–32) число сегментов. Выбирается некоторая клетка, для неё создается массив размера, равного количеству сегментов. Для каждого пикселя клетки высчитывается, в какой сегмент попадает направление градиента в этом пикселе и прибавляется значение модуля градиента в соответствующую ячейку массива. В результате получается гистограмма ориентированных градиентов пикселей выбранной клетки. Повторим процедуру получения гистограммы для всех клеток.

Подробно разберем функцию **getLocalHistogramm** получения гистограммы для клетки.

- Вычисляется производные яркости изображений G_x, G_y .
- Значением гистограмм является приближенное значение величины градиента $G = \sqrt{{G_x}^2 + {G_y}^2}$
- Записывается полученное значение в промежуток, определяемый вычисленной величиной $this->getDirection($Gx,$Gy).

- Нормализуется полученный вектор.

Каждую клетку, кроме первой,необходимо определять с середины предыдущей (метод скользящего окна). Проходя по всем элементам текущей клетки вычисляется градиент и угол яркости. Вычисляется градиент по x и y.

Абсолютное значение градиента присваивается в соответствующий промежуток. В конце метода происходит нормализация гистограмм. На этом рассмотрение функции **getLocalHistogramm** заканчиваем.

Перейдем к рассмотрению функции **getDirection** высчитывающего угол градиента в градусах (угол от 0 до 360) $\Theta = \arctan(\frac{G_y}{G_x})$: Аргументами функции является результат оператора Собеля в горизонтальном и вертикальном направлении. Функция возвращает угол в градусах. На этом рассмотрение функции **getDirection** заканчиваем.

Опишем функцию **getAbsValuegOfGradient**, который возвращает абсолютное значение градиента по x и y. Функция возвращает корень из суммы квадратов координат. Если значение меньше 40, то занулим его, тем самым избавимся от шума. На этом описание функции **getAbsValuegOfGradient** заканчиваем.

В зависимости от контрастности изображения значения градиентов могут существенно различаться. Поэтому важно нормализовать гистограммы, чтобы свойства освещения не влияли на результаты. Каждая гистограмма по сути является вектором — упорядоченным набором из нескольких чисел. Нормализация гистограммы — это нормирование вектора, то есть деление всех его значений на некоторую норму этого вектора. Нормы могут быть разные, но проще всего воспользоваться евклидовой нормой (корень из суммы квадратов элементов).

Разберем функцию **normalize** нормализации вектора.

Необходимо вычислить норму полученной гистограммы. Вычисляется Евклидова норма вектора (гистограммы). Значение вектора делится на значение его нормы. На этом описание функции **normalize** заканчиваем.

Объединение гистограмм в дескриптор. Все, что осталось сделать, — это соединить (конкатенировать, записать последовательно) полученные нормализованные гистограммы в один длинный массив,

получив тем самым дескриптор — вектор признаков, который описывает некоторые характерные признаки изображения. Именно с такими признаками, вместо самих изображений, будет работать классификатор.

Разберем главную функцию **getVector** получения вектора по изображению. Проходя двойным циклом по строкам и столбцам изображения, формируем некие квадратные клетки со стороной размером cellSize. Далее для каждой клетки вызываем функцию **getLocalHistogramm** и дописываем полученые значения в конец массива, который и будет конечным вектором. На этом описание функции **getVector** заканчиваем.

Рис. 8: Картинка в первичном виде.

Рис. 9: Картинка после обработки.

Рис. 10: Картинка в первичном виде.

Рис. 11: Применение горизонтального оператора Собеля.

Рис. 12: Применение вертикального оператора Собеля.

Рис. 13: Применение оператора Собеля в обоих направлениях.

Классификация изображений

Алгоритм был протестирован методом K ближайших соседей на картинках 96 на 96 пикселей. Было задано 2 множества слонов и лошадей, по 100 картинок из каждого множества.

Рис. 14: Лошади.

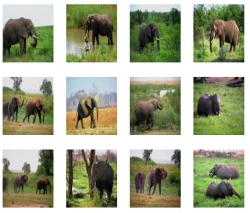

Рис. 15: Слоны.

В качестве тестового множества были взяты 50 картинок из каждого множества. Для картинок с изображением коней было правильно определено 70% картинок, для изображений со слонами — 63%.

На основе алгоритма было разработано WEB-приложение, демонстрирующее использование алгоритма (рис. 2) [9]. Суть приложения — определение картинки после некоторых воздействий (Сжатие, растяжение, обрезание или нанесение шумов). В приложение загружается 5 картинок, они считаются классами. Выбираем одну картинку, производим над ней различные операции (сжатие, обрезание, нанесение линий, текста и т. д.) После чего полученное изображение

загружается в приложение. Приложение должно отнести искаженное изображение к его оригиналу.

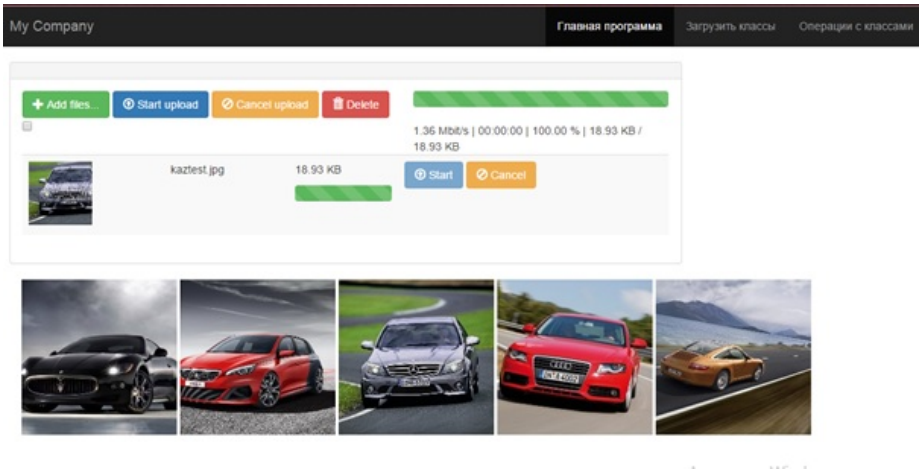

Рис. 16: Загрузка изображения.

Заключение

По результатам проделанной работы было реализовано WEB-приложение, состоящие из 3-х подсистем.

Первая подсистема — реализация MFCC дескриптора и классификация методом эталонов, которая показала хорошие результаты в определении жанра загружаемого музыкального файла.

Во второй подсистеме были использованы сторонние библиотеки Marsyas и WEKA, для параметризации музыкальных файлов и их классификации. Подсистема была протестирована. Были полученны следующие результаты, приведенные в таблице.

Таблица 3: Полученные результаты.

Алгоритм	Верно	Не верно	Успех(%)
SVM	187	13	93.5
Random forest	183	17	91.5
Байесовский классификатор	178	22	89

Третья подсистема — реализация WEB-приложения с возможностью загрузки искаженного изображения и определения его оригинала.

По итогам проделанной работы все подсистемы протестированы на реальных данных. Можно сделать заключение, что реализованные алгоритмы решают поставленные задачи.

Список литературы

[1] Каталог музыкальных файлов http://marsyas.cs.uvic.ca/sound/genres/.

[2] Листинг: первичная обработка wav-файлов http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#wawParse.

[3] Листинг: составление вектора признаков по музыкальному файлу http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#mfcc.

[4] Листинг: выбор метрики http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#chiSqr.

[5] Листинг: метод центроид http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#centroid.

[6] Музыка — полученные результаты http://tima.dvinemnauku.ru/frontend/web/index.php?r=music/index.

[7] Листинг: первичная обработка изображения http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#imageParse.

[8] Листинг: составление вектора признаков по изображению http://tima.dvinemnauku.ru/frontend/web/index.php?r=site/listing#hog.

[9] Классификация изображений http://tima.dvinemnauku.ru/frontend/web/index.php.

[10] Распознавание речевых команд с использованием сверточных нейронных сетей http://edu.botik.ru/upload/0cb3cff828c112050d3daebdfeee1ace.pdf.