Определение пола и возрастной группы человека по аудиозаписи его голоса

Есьмуков, 11-402

01.2017

Содержание

[Введение 3](#_Toc471678270)

[Набор данных 4](#_Toc471678271)

[Источник 4](#_Toc471678272)

[Получение данных 5](#_Toc471678273)

[Изучение загруженных данных 7](#_Toc471678274)

[Приведение данных к пригодному для анализа формату 10](#_Toc471678275)

[Преобразования над данными 11](#_Toc471678276)

[Обработка аудиозаписей 11](#_Toc471678277)

[Формирование итогового датасета 12](#_Toc471678278)

[Методы машинного обучения 14](#_Toc471678279)

[Обзор классификаторов 14](#_Toc471678280)

[Принцип работы RandomForestClassifier 14](#_Toc471678281)

[Итоговая модель 14](#_Toc471678282)

[Оценка качества алгоритма 16](#_Toc471678283)

[Дерево принятия решений 16](#_Toc471678284)

[Validation curve 18](#_Toc471678285)

[ROC 21](#_Toc471678286)

[Learning curve 23](#_Toc471678287)

[Вывод 26](#_Toc471678288)

[Источники 27](#_Toc471678289)

[Приложения 28](#_Toc471678290)

[Приложение 1 28](#_Toc471678291)

[Приложение 2 34](#_Toc471678292)

[Приложение 3 45](#_Toc471678293)

[Приложение 4 47](#_Toc471678294)

[Приложение 5 49](#_Toc471678295)

[Приложение 6 51](#_Toc471678296)

[Приложение 7 63](#_Toc471678297)

[Приложение 8 64](#_Toc471678298)

[Приложение 9 70](#_Toc471678299)

[Приложение 10 71](#_Toc471678300)

# Введение

Цель данного проекта – определение пола и возрастной группы человека по аудиозаписи его голоса. Этот проект можно применить, например, для подбора таргетированной рекламы для пользователей социальных сетей, которые не предоставили данных о своем поле и возрасте.

Небольшой набор данных и обзор методов машинного обучения по данной проблеме можно найти на Kaggle (1), но в нем отсутствуют данные о возрасте, поэтому было необходимо найти подходящий набор данных и сделать его пригодным для решения поставленной задачи.

# Набор данных

## Источник

Крупнейшей общедоступной базой данных аудиозаписей людских голосов является VoxForge (2). На этом сайте собрано порядка 15 тысяч архивов, в каждом архиве содержится несколько аудиозаписей зачитанного текста.

Структура архива:

.

├── etc

│   ├── PROMPTS // фразы, зачитанные спикером

│   └── README // описание параметров спикера

├── LICENSE // лицензия. В основном GNU GPLv3.

└── wav // директория с wav файлами – записями фраз.

Содержимое файла README:

User Name:Aaron

Speaker Characteristics:

**Gender: Male**

**Age Range: Adult**

Language: EN

Pronunciation dialect: New Zealand English

Recording Information:

Microphone make: n/a

Microphone type: Headset mic

Audio card make: unknown

Audio card type: unknown

Audio Recording Software: VoxForge Speech Submission Application

O/S:

File Info:

File type: wav

Sampling Rate: 48000

Sample rate format: 16

Number of channels: 1

Для решения поставленной задачи нам необходимо:

1. Считать пол и возрастную группу каждого спикера из файла README внутри архива.
2. Посчитать для каждой аудиозаписи набор акустических свойств, по которым будем обучать классификатор. Подробнее об этом в следующем разделе данного отчета.
3. Соединить данные первых двух пунктов и использовать на них методы машинного обучения.

## Получение данных

Сначала архивы необходимо скачать. В разделе загрузок на сайте архивы хранятся в достаточно нестрогом формате, поэтому нужно разобраться в структуре.

Например, загрузки английских спикеров находятся в [http://www.repository.voxforge1.org/downloads**/SpeechCorpus/**](http://www.repository.voxforge1.org/downloads/SpeechCorpus/) , русских – в [http://www.repository.voxforge1.org/downloads**/Russian/**](http://www.repository.voxforge1.org/downloads/Russian/) , немецких – в [http://www.repository.voxforge1.org/downloads**/de/**](http://www.repository.voxforge1.org/downloads/de/) и т.д.

Но структура директорий внутри каждого языка едина: архивы находятся в "Trunk/Audio/Main/". Они представлены в двух вариантах: "16kHz\_16bit" и "8kHz\_16bit" – по частоте дискретизации аудиозаписей. Мы загрузили первый, поскольку чем выше частота дискретизации, тем лучше качество. Но на самом деле для решения нашей задачи вполне подошли бы и архивы с 8 кГц.

Таким образом, если рекурсивно получить листинги всех директорий веб сервера, можно извлечь оттуда прямые ссылки на архивы, которые нам нужны.

Мы сделали это командой

wget -r -l10 -np -k -A html -nc --reject-regex ".\*[?].\*" http://www.repository.voxforge1.org/downloads/

Описание ключей wget данной команды:

* -r – рекурсивная загрузка;
* -l10 – глубина рекурсии – 10 директорий;
* -np – не загружать файлы, находящиеся выше в иерархии;
* -k – конвертировать ссылки в загруженных html файлах, в нашем случае относительные ссылки на архивы будут преобразованы в абсолютные;
* -A html – загружать только html файлы;
* -nc – не перезаписывать уже загруженные html файлы;
* --reject-regex ".\*[?].\*" – отклонять загрузку файлов, удовлетворяющих данному регулярному выражению. В нашем случае были отсеяны html файлы с query параметрами, возвращавшие одинаковые списки файлов, но с различными вариантами сортировки.

В итоге мы получили директорию, содержащую множество index.html файлов с листингами директорий web-сервера.

Архивы имеют формат ".tgz", поэтому можно просто извлечь ссылки на все файлы, имеющие такой формат и содержащие в пути "Audio/Main/16kHz\_16bit". Мы разделили данные на 2 группы: *Main* (/SpeechCorpus/ – английский язык) и *other* (все остальные языки).

find www.repository.voxforge1.org/ -name '\*.html' -exec grep -o '"[^"]\+\.tgz"' {} \; | tr -d '"' | grep Audio/Main/16kHz\_16bit | grep downloads/SpeechCorpus/Trunk/Audio/ > urls.Main.16.txt

find www.repository.voxforge1.org/ -name '\*.html' -exec grep -o '"[^"]\+\.tgz"' {} \; | tr -d '"' | grep Audio/Main/16kHz\_16bit | grep -v downloads/SpeechCorpus/Trunk/Audio/ > urls.other.16.txt

cat urls.Main.16.txt | wc -l

6182

cat urls.other.16.txt | wc -l

9090

Так мы видим, что подходящих нам архивов – 15272 штуки.

В целях упрощения работы с данными, архивы в пределах группы мы загружали в одну директорию.

Файлы из группы Main находятся на сервере в одной директории, поэтому пересекаться по именам они не могут. А вот файлы из группы other – могут, поскольку находятся в разных директориях. Убедимся, что пересечений по именам нет:  
cat urls.other.16.txt | xargs -n 1 basename | sort | uniq | wc -l

9090

Теперь скачаем архивы:  
cd ~/voxforge/dl/Main

xargs -P 20 -n 1 curl -O < ~/voxforge/urls.Main.16.txt

cd ~/voxforge/dl/other

xargs -P 20 -n 1 curl -O < ~/voxforge/urls.other.16.txt

Распакуем архивы:

cd ~/voxforge/unpacked/Main

find ~/voxforge/dl/Main -type f -exec tar xaf {} \;

cd ~/voxforge/unpacked/other

find ~/voxforge/dl/other -type f -exec tar xaf {} \;

Объем распакованных данных:

cd ~/voxforge/unpacked

du -ch -d 1

14G ./Main

23G ./other

37G total

## Изучение загруженных данных

Перед тем, как начать работу с данными, необходимо изучить их структуру.

cd ~/vorforge/unpacked

find . -type f -name README | wc -l

15188

Заметим, что не хватает 84 файлов. Но 15 тысяч нам тоже хватит, поэтому можно не обращать на это внимания.

Посмотрим на значения пола и возраста:

find . -type f -name README -exec grep -i 'Gender:' {} \; | tr -d ' \t;][\r' | sort | uniq -c

1 Gender:adult

2 Gender:bilinmiyor

1 Gender:Bittewählen

3 Gender:desconegut

4 Gender:desconhecido

22 Gender:desconocido

153 Gender:Erkek

77 Gender:female

387 Gender:Female

1 Gender:Femení

466 Gender:Femenino

3 Gender:Femër

89 Gender:Féminin

22 Gender:Feminino

67 Gender:Femmina

30 Gender:inconnu

1 Gender:Kadın

1 Gender:Maakeenkeuze

65 Gender:Machio

1 Gender:made

5 Gender:make

623 Gender:male

5260 Gender:Male

1 Gender:Male&Femalemix

734 Gender:mannelijk

1056 Gender:Männlich

884 Gender:Maschio

20 Gender:Masculí

1996 Gender:Masculin

2027 Gender:Masculino

30 Gender:Mashkull

29 Gender:M�nnlich

12 Gender:Muško

1 Gender:nepoznato

19 Gender:nondefinito

15 Gender:PleaseSelect

2 Gender:PorfavorSeleccione

1 Gender:Selecione

15 Gender:Sélectionnez

1 Gender:Selezionare

3 Gender:unbekannt

28 Gender:unknown

34 Gender:vrouwelijk

109 Gender:Weiblich

1 Gender:لطفاانتخابکنید

9 Gender:مذکر

1 Gender:مونث

1 Gender:אנאבחר

26 Gender:זכר

2 Gender:נקבה

3 Gender:άγνωστο

130 Gender:Άνδρας

3 Gender:Γυναίκα

5 Gender:Выберите

19 Gender:другое

8 Gender:Жена

25 Gender:Женщина

1 Gender:Жінка

1 Gender:Изберете

567 Gender:Мужчина

38 Gender:Мъж

3 Gender:неизвестен

37 Gender:Чоловік

find . -type f -name README -exec grep -i 'Age Range:' {} \; | tr -d ' \t;][\r' | sort | uniq -c

529 Agerange:adult

10 Agerange:Adult

143 AgeRange:adult

4778 AgeRange:Adult

2 Agerange:adult(bornin1983)

1775 AgeRange:Adulte

3185 AgeRange:Adulto

14 AgeRange:Anziano

2 AgeRange:bilinmiyor

1 AgeRange:Bittewählen

3 AgeRange:desconegut

6 AgeRange:desconhecido

19 AgeRange:desconocido

1090 AgeRange:Erwachsener

79 AgeRange:Genç

205 AgeRange:Giovane

2 AgeRange:Idoso

1 AgeRange:Imoshuar

26 AgeRange:inconnu

32 AgeRange:Irritur

155 AgeRange:jeugd

273 AgeRange:Jeune

11 AgeRange:Jove

100 AgeRange:Jovem

99 AgeRange:Jugendlicher

1 AgeRange:Maakeenkeuze

1 AgeRange:male

1 AgeRange:nepoznato

21 AgeRange:Niño

18 AgeRange:nondefinito

12 AgeRange:Odrasli

31 AgeRange:PleaseSelect

1 AgeRange:PorfavorSeleccione

1 AgeRange:Selecione

11 AgeRange:Sélectionnez

3 AgeRange:Selezionare

13 Agerange:senior

18 AgeRange:senior

182 AgeRange:Senior

5 AgeRange:TerceraEdad

4 AgeRange:unbekannt

27 AgeRange:unknown

595 AgeRange:volwassen

1 AgeRange:Yaşlı

74 AgeRange:Yetişkin

7 Agerange:youth

6 AgeRange:youth

729 AgeRange:Youth

10 AgeRange:بزرگسال

1 AgeRange:لطفاانتخابکنید

1 AgeRange:אנאבחר

12 AgeRange:מבוגר

16 AgeRange:צעיר

4 AgeRange:άγνωστο

111 AgeRange:Ενήλικας

21 AgeRange:Νεαρός

1 AgeRange:более55

28 AgeRange:Възрастен

4 AgeRange:Выберите

285 AgeRange:до25

9 AgeRange:До25років

311 AgeRange:до55

29 AgeRange:До55років

15 AgeRange:другое

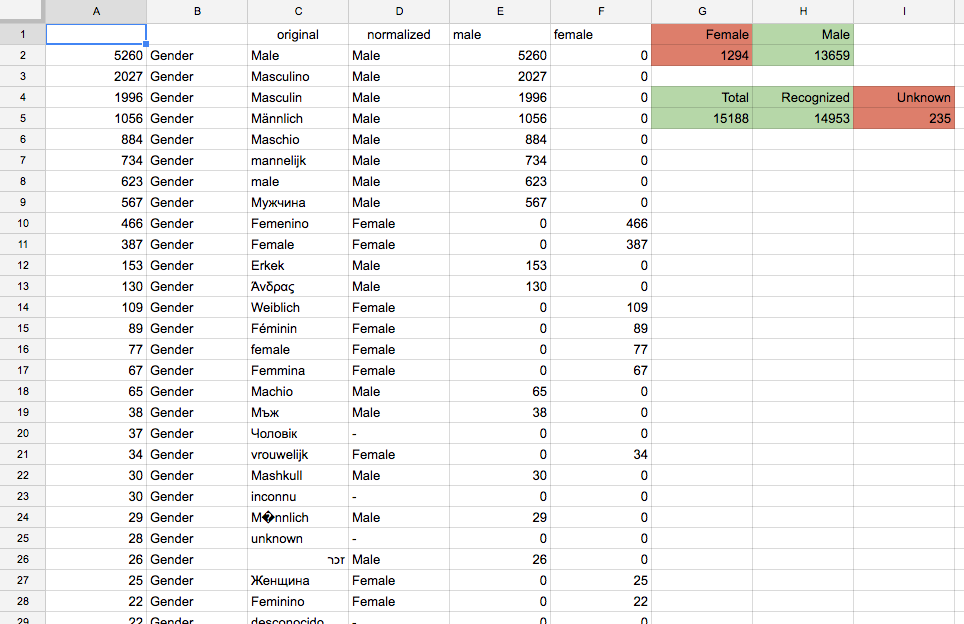
1 AgeRange:Изберете

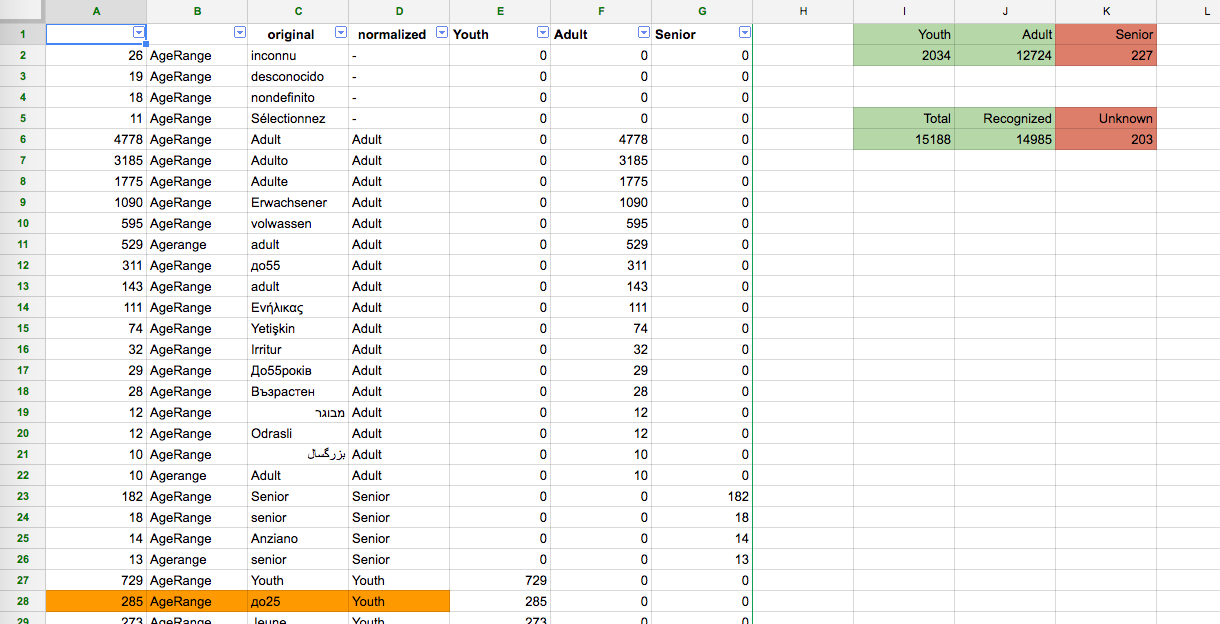
18 AgeRange:Младеж/Девойка

3 AgeRange:неизвестен

Очевидно, что без приведения этих значений к общему виду нам не обойтись. Также можно заметить, что некоторые значения явно не указывают пол или возраст. Их надо будет отбросить.

Была составлена таблица, в которой значения из README сопоставлены с нормализованными значениями (male/female, youth/adult/senior):





В результате получили такое распределение параметров спикеров по количеству архивов:

|  |  |  |
| --- | --- | --- |
| Gender | Age Range | Count |
| Male | Adult | 11494 |
| Male | Youth | 1894 |
| Female | Adult | 1167 |
| Male | Senior | 225 |
| Female | Youth | 123 |
| Female | Senior | 3 |

Сразу обратим внимание на некоторые особенности полученного набора данных:

1. крайне мало архивов с записями голосов пожилых женщин;
2. очень много архивов с записями голосов взрослых мужчин.

Некоторые архивы содержат записи в формате FLAC. Впрочем, 190 тысяч WAV записей нам вполне достаточно, поэтому, для простоты, использовать будем только их.

cd ~/vorforge/unpacked

find . -type f -iname '\*.wav' | wc -l

190688

find . -type f -iname '\*.flac' | wc -l

26421

## Приведение данных к пригодному для анализа формату

Используя данные вышеприведенной таблицы, был разработан скрипт [приложение 1], который для каждого README файла пишет строки в формате csv.

Столбцы csv таблицы:

1. m/f/na/???... – пол – Male/Female/Not Specified/Unrecognized
2. y/a/s/na/???... – возрастная группа – Youth/Adult/Senior/Not Specified/Unrecognized
3. путь до README файла

Пример:

m,y,./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README,

m,s,./Main/16kHz\_16bit/ColinBeckingham-20100128-gzl/etc/README,

m,a,./Main/16kHz\_16bit/JohanLingen-20080306-nov/etc/README,

m,a,./Main/16kHz\_16bit/anonymous-20090605-dlc/etc/README,

m,a,./Main/16kHz\_16bit/anonymous-20110221-djy/etc/README,

m,a,./Main/16kHz\_16bit/anonymous-20120615-een/etc/README,

m,a,./Main/16kHz\_16bit/GamaBedolla-20150210-jbr/etc/README,

m,a,./Main/16kHz\_16bit/anonymous-20130516-wkd/etc/README,

f,a,./Main/16kHz\_16bit/anonymous-20090618-emm/etc/README,

m,y,./Main/16kHz\_16bit/ethanoconnors-20080607-hyw/etc/README,

Далее нам необходимо посчитать набор акустических свойств для каждого аудиофайла, который мы будем сопоставлять с данными этой csv таблицы.

# Преобразования над данными

## Обработка аудиозаписей

Голосовой аппарат человека очень сложен. Например, голос имеет такие характеристики, как высота, сила, тембр. Зависят они от объема и формы резонаторов (грудная клетка, рот, глотка, пазухи и т.д.), интенсивности выдыхаемого воздуха, положения голосовых связок. Многими из этих величин человек способен управлять, таким образом изменяя свой голос.

Однако, если рассматривать обычную речь человека, то основными факторами, зависящими от пола и возраста, являются (3):

1. длина речевого тракта (vocal tract length);
2. артикуляция;
3. объемы резонаторов.

С точки зрения звука как аудиосигнала, эти характеристики влияют на частоту звука, на основную частоту (4) (fundamental frequency), на частоты формант (5).

В комментариях к датасету с Kaggle можно найти ссылку на скрипт на языке R (6), который высчитывает акустические характеристики аудиозаписей. Этот скрипт был переписан на Python и дополнен [приложение 2], чтобы иметь возможность гибко менять параметры сбора данных и встраивать этот функционал в другие программы.

В итоговой программе для каждой аудиозаписи подсчитывается следующий набор характеристик:

* meanfreq – средняя частота звука (кГц)
* sd – стандартное отклонение частоты звука (кГц)
* median – медиана (второй квартиль) частоты звука (кГц)
* Q25 – первый квартиль частоты звука (кГц)
* Q75 – третий квартиль частоты звука (кГц)
* IQR – интерквартильный размах (кГц)
* skew – skewness – коэффициент асимметрии (7) распределения амплитуд звука в частотном пространстве
* kurt – kurtosis – коэффициент эксцесса (8) распределения амплитуд звука в частотном пространстве
* sp.ent – Shannon entropy – энтропия Шеннона (9) распределения амплитуд звука в частотном пространстве
* sfm – spectral flatness – энтропия Винера (10) распределения амплитуд звука в частотном пространстве
* mode – мода частоты звука (кГц)
* meanfun – среднее основных частот (4) (кГц)
* minfun – минимум основных частот (кГц)
* maxfun – максимум основных частот (кГц)
* meandom – среднее доминирующих частот (dominant frequency) (кГц)
* mindom – минимум доминирующих частот (кГц)
* maxdom – максимум доминирующих частот (кГц)
* dfrange – разность между максимумом и минимумом доминирующих частот (кГц)
* modindx – индекс модуляции доминирующих частот
* startdom – первая доминирующая частота (кГц)
* enddom – последняя доминирующая частота (кГц)
* dfslope – крутизна доминирующей частоты (кГц)
* meanvtl – среднее длины речевого тракта по формантам (11) (см)
* minvtl – минимум длины речевого тракта по формантам (см)
* maxvtl – максимум длины речевого тракта по формантам (см)

При подсчете используются только первые 20 секунд аудиозаписи и рассматривается только голосовой частотный диапазон (0-280 Гц) (12).

Программа принимает на вход список wav файлов. На выход выдает csv таблицу с полями: "","sound.files","selec","duration","meanfreq","sd","median","Q25","Q75","IQR","skew","kurt","sp.ent","sfm","mode","meanfun","minfun","maxfun","meandom","mindom","maxdom","dfrange","modindx","startdom","enddom","dfslope","meanvtl","minvtl","maxvtl"

## Формирование итогового датасета

Итоговый датасет должен представлять собой таблицу, содержащую параметры аудиозаписей и информацию о поле и возрасте. Для этого был разработан скрипт на Python, который соединяет две описанных выше таблицы [приложение 3].

Таблица итогового датасета содержит следующие поля: "sound.files","meanfreq","sd","median","Q25","Q75","IQR","skew","kurt","sp.ent","sfm","mode","centroid","meanfun","minfun","maxfun","meandom","mindom","maxdom","dfrange","modindx","startdom","enddom","dfslope","sample\_name","gender","age\_range","readme\_file\_path","gender\_and\_age\_range"

Для ускорения обучения моделей, была создана уменьшенная версия датасета, содержащая по 500 аудиозаписей для каждой пары пол-возраст:

tail -n +2 voxforge\_dataset\_py\_full.csv | perl -MList::Util=shuffle -e 'print shuffle<STDIN>' > voxforge\_dataset\_py\_shuffled.csv

head -1 voxforge\_dataset\_py\_full.csv > voxforge\_dataset\_py\_short.csv

grep -e '"ma"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

grep -e '"fa"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

grep -e '"my"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

grep -e '"ms"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

grep -e '"fy"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

grep -e '"fs"$' voxforge\_dataset\_py\_shuffled.csv | head -500 >> voxforge\_dataset\_py\_short.csv

# Методы машинного обучения

## Обзор классификаторов

Для определения подходящей модели классификатора была разработана программа на Python [приложение 4]. Обучение производилось на уменьшенной версии датасета. Полный вывод программы приложен к отчету [приложение 6].

|  |  |  |
| --- | --- | --- |
| **Классификатор** | **Scores united s+a age range [gender, male, female]** | **Scores distinct age range [gender, male, female]** |
| SVC | [0.93267326732673272, 0.71368948247078468, 0.76642335766423353] | [0.93267326732673272, 0.53672787979966607, 0.76763990267639903] |
| **RandomForestClassifier** | [0.93019801980198025, 0.71202003338898168, 0.76034063260340634] | [0.93019801980198025, 0.54424040066777968, 0.77007299270072993] |
| SGDClassifier | [0.9133663366336634, 0.63689482470784642, 0.66058394160583944] | [0.9133663366336634, 0.44073455759599334, 0.64355231143552316] |
| KNeighborsClassifier | [0.89851485148514854, 0.70868113522537568, 0.75060827250608275] | [0.89851485148514854, 0.51669449081803009, 0.74695863746958635] |

Определение возраста по трем возрастным категориям на всех классификаторах дает неудовлетворительный результат: точность среди мужчин около 54%. Путем объединения двух возрастных групп (взрослых и пожилых) в одну, удалось добиться точности 71% среди мужчин и 76% среди женщин.

В итоге классификатором был выбран **RandomForestClassifier** с параметром n\_estimators=50, поскольку он выдает высокие результаты, сравнимые с SVC, но обучается значительно быстрее, чем SVC.

## Принцип работы RandomForestClassifier

RandomForestClassifier представляет собой совокупность (ensemble) n\_estimators штук решающих деревьев. В реализации scikit-learn класс выбирается путем усреднения вероятностных предсказаний всех деревьев (13).

## Итоговая модель

Итоговая модель состоит из 3 отдельных классификаторов RandomForestClassifier: для пола, для возрастной группы среди мужчин, для возрастной группы среди женщин [приложение 7]. Полный вывод программы приложен к отчету [приложение 8].

|  |  |
| --- | --- |
| **Датасет** | **Scores [gender, male age, female age]** |
| короткий | [0.939, 0.693, 0.752] |
| полный | [0.972, 0.890, 0.948] |

# Оценка качества алгоритма

Все данные в этом разделе получены на коротком датасете.

## Дерево принятия решений

Для оценки степени важности каждого параметра акустической модели, была разработана программа [приложение 9], обучающая деревья решений и строящая графические представления графов итоговых деревьев.

gender

**meanfun 0.694577**

meandom 0.056149

median 0.032175

sd 0.022374

Q25 0.022328

enddom 0.014110

sp.ent 0.013674

Q75 0.013514

sfm 0.013276

maxvtl 0.012965

minfun 0.012001

skew 0.011432

modindx 0.011191

minvtl 0.010559

IQR 0.010205

mode 0.010003

meanvtl 0.008719

kurt 0.008397

maxdom 0.005691

startdom 0.005080

dfslope 0.003810

maxfun 0.002734

meanfreq 0.002630

dfrange 0.002406

centroid 0.000000

mindom 0.000000

male age

**meanfun 0.094207**

median 0.063496

kurt 0.058888

meandom 0.058836

skew 0.051202

Q25 0.050212

meanvtl 0.048893

Q75 0.048749

sp.ent 0.046815

dfslope 0.044190

mode 0.043799

sd 0.043100

maxvtl 0.042501

enddom 0.039341

minfun 0.039245

IQR 0.037854

minvtl 0.030225

modindx 0.029103

startdom 0.026597

centroid 0.021498

maxfun 0.020695

dfrange 0.019156

meanfreq 0.016341

sfm 0.013206

maxdom 0.011851

mindom 0.000000

female age

**sd 0.200185**

sp.ent 0.072181

skew 0.065616

mode 0.058667

sfm 0.054831

meandom 0.053764

median 0.046570

maxvtl 0.043869

meanfreq 0.038774

modindx 0.036228

maxdom 0.035390

kurt 0.034659

IQR 0.032097

Q25 0.031691

minvtl 0.031209

startdom 0.031002

Q75 0.025521

meanvtl 0.022569

dfslope 0.019291

meanfun 0.016617

minfun 0.015650

dfrange 0.013610

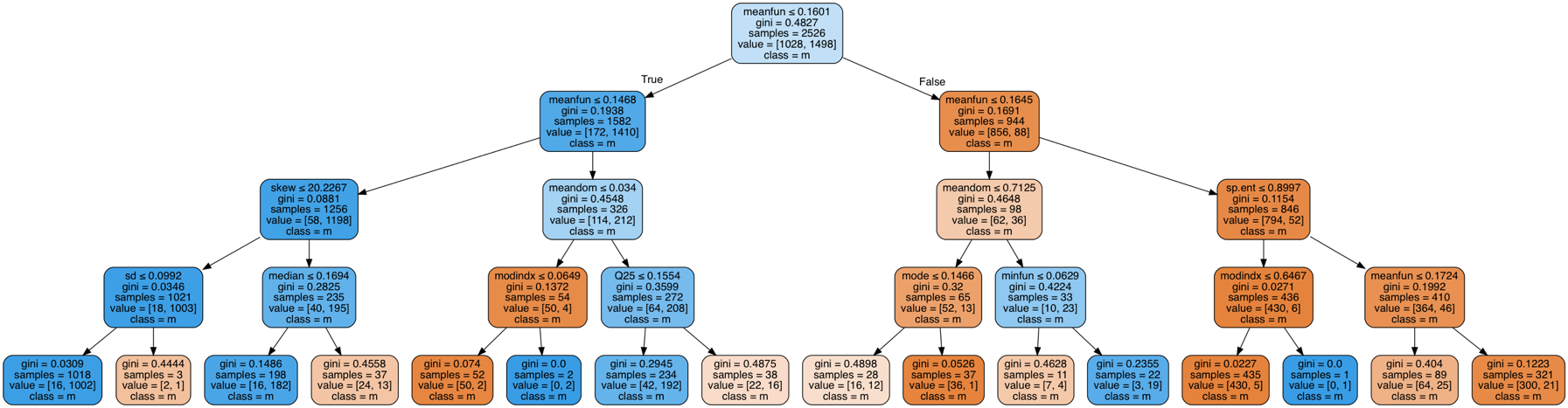
centroid 0.011575

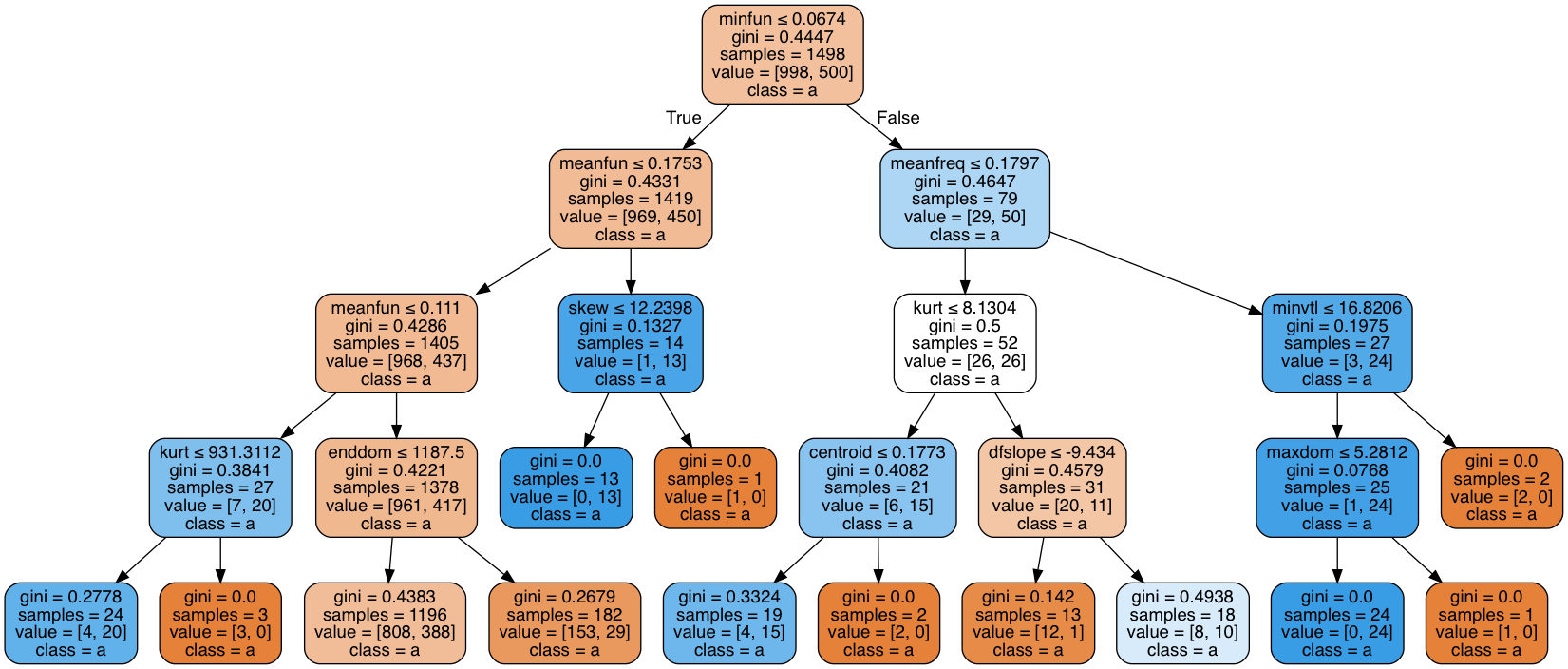
maxfun 0.008436

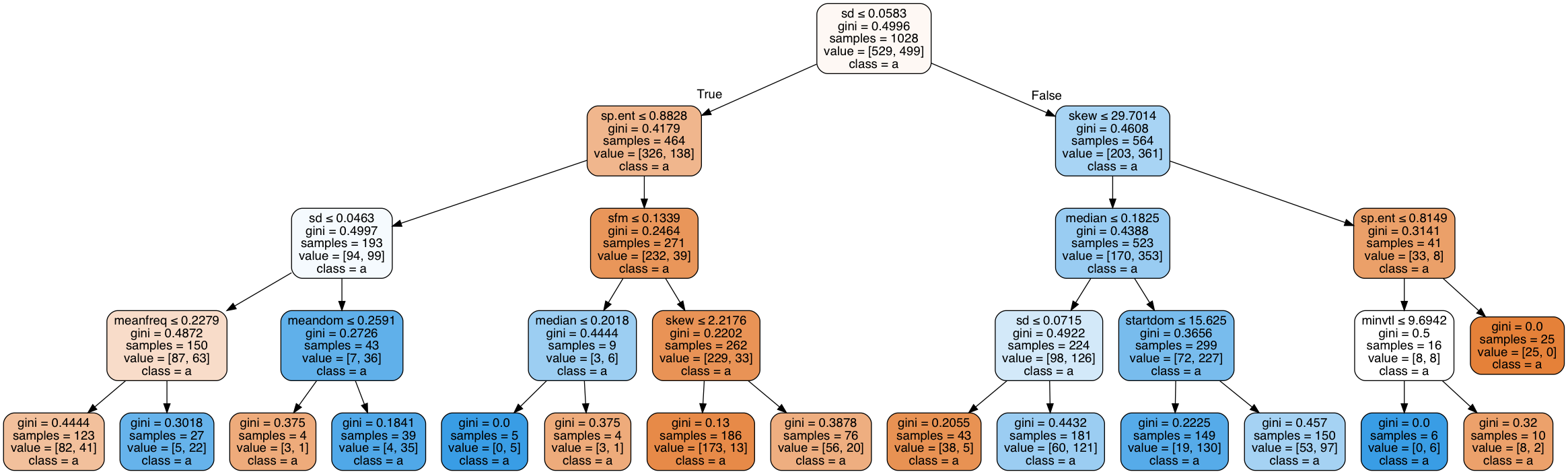
mindom 0.000000

enddom 0.000000

Графы (ограниченны глубиной 4 для наглядности):

**gender**

**male age**

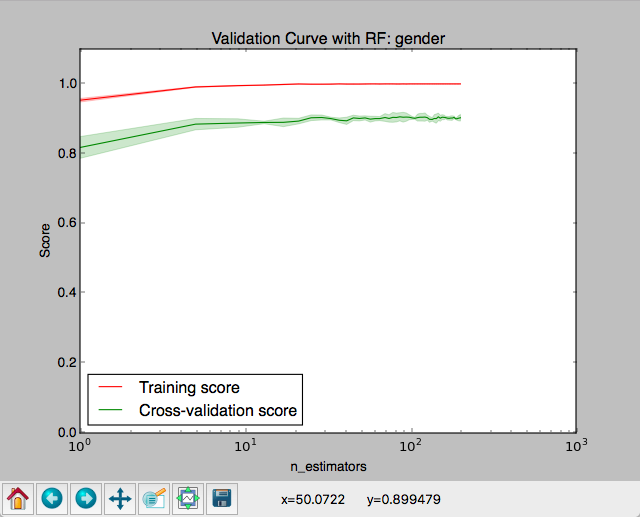


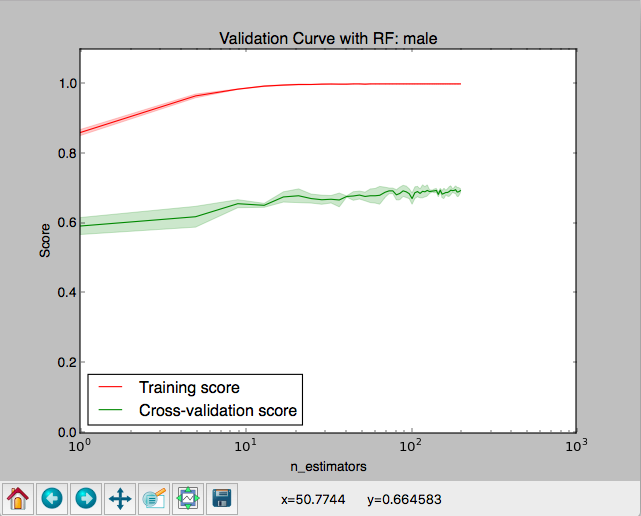
**female age**

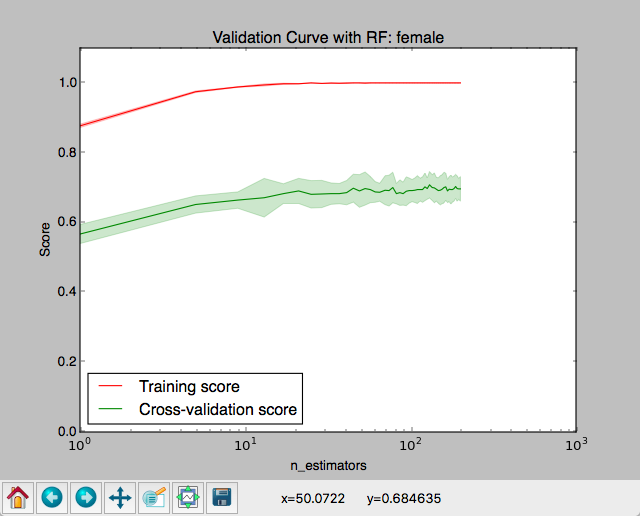
Для визуализации качества классификаторов была разработана программа [приложение 10], выдающая следующие виды графиков:

1. validation curve;
2. receiver operating characteristic;
3. learning curve.

## Validation curve

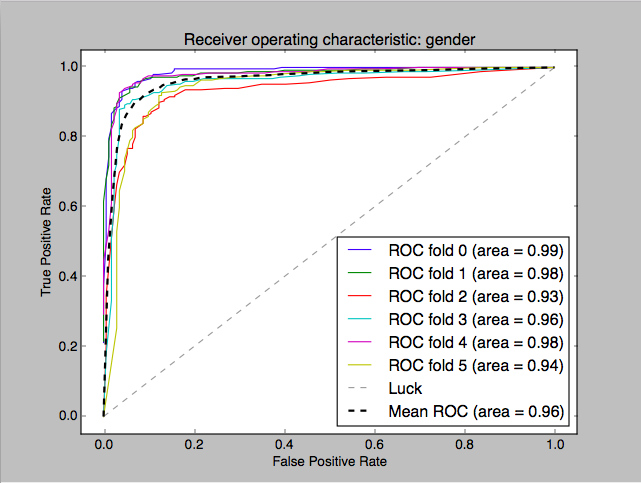


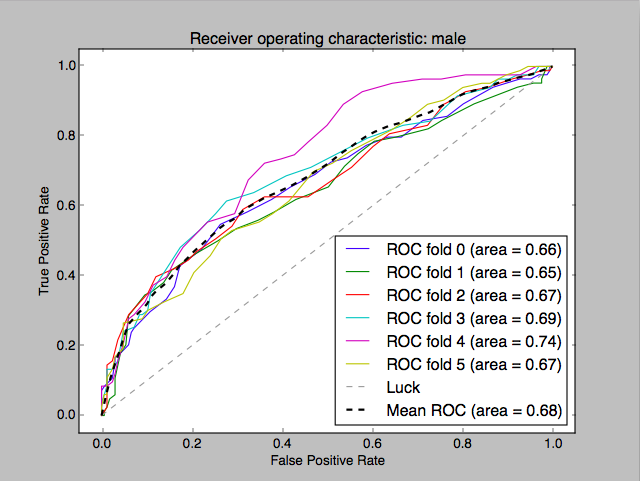


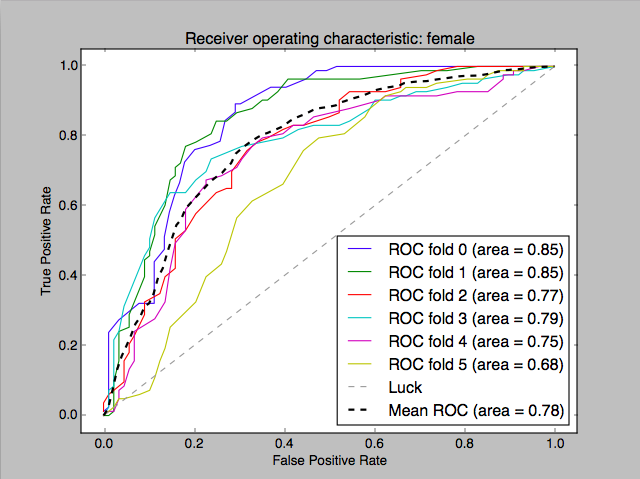


С ростом количества деревьев score увеличивается. Для пола при 7 деревьях достигается значение score близкое к максимальному. Для возраста и среди мужчин, и среди женщин это число приблизительно равно 40. В предыдущем разделе мы выбрали 50 деревьев – это количество является оптимальным среди всех трех категорий.

## ROC

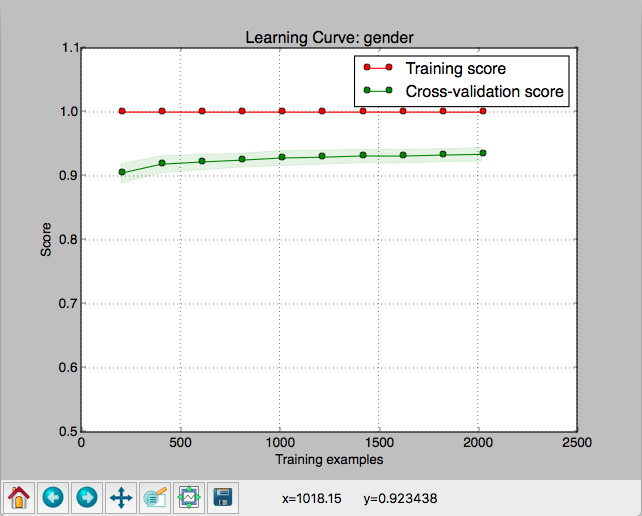


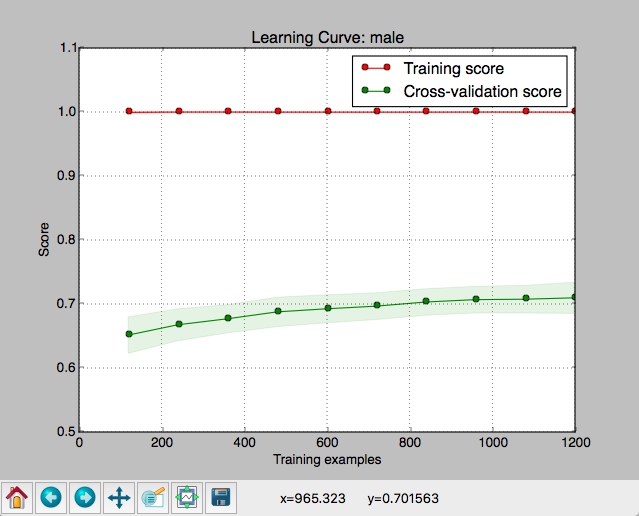


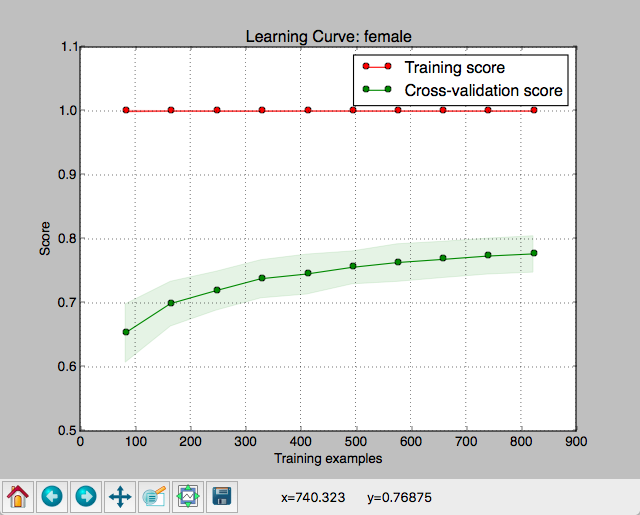


По полу результат хороший: площадь под кривой близка к максимальной. Но по возрасту, как и следовало ожидать, виден значительный разброс между разными вариантами разбиения датасета. Это может говорить о плохом качестве данных.

## Learning curve







На этих графиках мы видим, что по полу и возрасту среди мужчин данных достаточно: с ростом данных кривая возрастает значительно медленнее. Но по возрасту среди женщин рост кривой замедляется медленнее. Это может говорить о том, что данных в коротком датасете не достаточно для качественного обучения классификатора возраста среди женщин.

# Вывод

На датасете VoxForge нам удалось получить точность определения пола 97%, возрастной группы среди мужчин – 89%, возрастной группы среди женщин – 94%.

Однако возрастная группа определена только двумя классами: ребенок и взрослый. Для трех классов (ребенок, взрослый, пожилой) добиться удовлетворительных результатов не удалось. Возможно, причина заключается в низком качестве датасета. Во-первых, он содержит очень малое количество записей голосов пожилых людей и детей. Во-вторых, категории ребенок/взрослый/пожилой выбираются контрибьюторами датасета на основе личной субъективной оценки своего возраста, поэтому могут иметь широкий и перекрывающийся разброс в годах жизни.

Тем не менее, данный алгоритм вполне применим в тех областях, где не требуется высокой точности определения возраста.

# Источники

1. Gender Recognition by Voice | Kaggle. [В Интернете] https://www.kaggle.com/primaryobjects/voicegender.

2. Free Speech... Recognition (Linux, Windows and Mac) - voxforge.org. [В Интернете] http://www.voxforge.org/.

3. Spontaneous Voice Gender Imitation Abilities in Adult Speakers. [В Интернете] http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0031353.

4. Fundamental frequency - Wikipedia. [В Интернете] https://en.wikipedia.org/wiki/Fundamental\_frequency.

5. Форманта — Википедия. [В Интернете] https://ru.wikipedia.org/wiki/Форманта.

6. voice-gender/sound.R at master · primaryobjects/voice-gender · GitHub. [В Интернете] https://github.com/primaryobjects/voice-gender/blob/master/sound.R.

7. Коэффициент асимметрии — Википедия. *https://ru.wikipedia.org/wiki/Коэффициент\_асимметрии.* [В Интернете]

8. Коэффициент эксцесса — Википедия. [В Интернете] https://ru.wikipedia.org/wiki/Коэффициент\_эксцесса.

9. Информационная энтропия — Википедия. [В Интернете] https://ru.wikipedia.org/wiki/Информационная\_энтропия#.D0.9E.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5\_.D0.BF.D0.BE\_.D0.A8.D0.B5.D0.BD.D0.BD.D0.BE.D0.BD.D1.83.

10. Spectral flatness - Wikipedia. [В Интернете] https://en.wikipedia.org/wiki/Spectral\_flatness.

11. How can I estimate a person's vocal tract length, using a recorded audio file? - ResearchGate. [В Интернете] https://www.researchgate.net/post/How\_can\_I\_estimate\_a\_persons\_vocal\_tract\_length\_using\_a\_recorded\_audio\_file.

12. Identifying the Gender of a Voice using Machine Learning | Primary Objects. [В Интернете] http://www.primaryobjects.com/2016/06/22/identifying-the-gender-of-a-voice-using-machine-learning/.

13. 1.11. Ensemble methods - scikit-learn 0.18.1 documentation. [В Интернете] http://scikit-learn.org/stable/modules/ensemble.html#random-forests.

# Приложения

## Приложение 1

pipe\_voxforge\_readme.py:

*#!/usr/bin/env python3  
"""  
Extracts Gender and Age Range parameters from the voxforge dataset in a  
normalized form  
  
Prints csv which looks like:  
"gender","age\_range","readme\_file\_path"  
"m","y","./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README"  
"m","s","./Main/16kHz\_16bit/ColinBeckingham-20100128-gzl/etc/README"  
"m","a","./Main/16kHz\_16bit/JohanLingen-20080306-nov/etc/README"  
...  
  
Columns:  
m/f/na/???... - Male/Female/Not Specified/Unrecognized  
y/a/s/na/???... - Youth/Adult/Senior/Not Specified/Unrecognized  
path to the readme file which that info was taken from  
(empty column)  
  
---  
  
USAGE  
  
find . -name README -type f | ../pipe\_voxforge\_readme.py > voxforge\_readme.csv  
  
# dataset size (subtract 1 line from result - the csv header)  
cat voxforge\_readme.csv | wc -l  
15180  
  
# unrecognized  
cat voxforge\_readme.csv | grep '??? ' | wc -l  
36  
  
# gender-age range pairs count  
cat voxforge\_readme.csv | awk -F , '{print $1,$2}' | sort | uniq -c | sort -nr  
  
# Useful to find out single byte charsets of README files  
cat README | hexdump -v -C  
  
"""***import** sys  
**import** re  
  
EXAMPLE\_README\_FILE\_CONTENTS = \  
 **"""User Name:aaa  
  
Speaker Characteristics:  
  
Gender: Male  
Age Range: Youth  
Language: EN  
Pronunciation dialect: Other  
  
Recording Information:  
  
Microphone make: n/a  
Microphone type: Headset mic  
Audio card make: unknown  
Audio card type: unknown  
Audio Recording Software: VoxForge Speech Submission Application  
O/S:  
  
File Info:  
  
File type: wav  
Sampling Rate: 48000  
Sample rate format: 16  
Number of channels: 1  
"""**IGNORE\_BAD\_FORMAT = [  
 **'kmaclean-20071108-poe/etc/README'**, *# 'Sex' instead of the 'Gender'* **'anonymous-20130731-nuz/etc/README'**, *# Gender and Age Range  
 # values are swapped  
  
 # no valid metadata* **'an4/etc/README'**,  
 **'cmu\_us\_rms\_arctic/etc/README'**,  
 **'cmu\_us\_jmk\_arctic/etc/README'**,  
 **'cmu\_us\_awb\_arctic/etc/README'**,  
 **'cmu\_us\_clb\_arctic/etc/README'**,  
 **'cmu\_us\_ksp\_arctic/etc/README'**,  
  
 *# Gender:Male&Femalemix* **'rdc/etc/README'**, *# http://www.repository.voxforge1.org/downloads/Russian/Trunk/Audio/Main/16kHz\_16bit/rdc.tgz*]  
  
  
**def** main():  
 *# print csv header* out\_csv\_strings((**"gender"**, **"age\_range"**, **"readme\_file\_path"**))  
  
 **for** readme\_file\_path **in** sys.stdin:  
 readme\_file\_path = readme\_file\_path.strip()  
 **if** any(readme\_file\_path.endswith(s) **for** s **in** IGNORE\_BAD\_FORMAT):  
 **continue** process\_readme\_file(readme\_file\_path)  
  
  
**def** out\_csv\_strings(strings\_list):  
 sys.stdout.write(**"%s\n"** % **','**.join(**'"%s"'** % s **for** s **in** strings\_list))  
  
  
**def** process\_readme\_file(readme\_file\_path):  
 **try**:  
 **with** open(readme\_file\_path, **'rb'**) **as** file:  
 readme\_contents\_binary = file.read()  
 **try**:  
 readme\_contents = readme\_contents\_binary.decode()  
 **except** UnicodeDecodeError:  
 *# some files are stored in a single-byte charset  
 # ./other/16kHz\_16bit/anonymous-20080324-dpi/etc/README  
 # ./other/16kHz\_16bit/Hornochse-20080318-qok/etc/README* **if b'Language: DE' in** readme\_contents\_binary:  
 readme\_contents = readme\_contents\_binary.decode(**'cp1252'**)  
 **else**:  
 **raise** gender, agerange = ReadmeParser.parse(readme\_contents)  
 out\_csv\_strings((gender, agerange, readme\_file\_path))  
 **except**:  
 sys.stderr.write(**"readme\_file\_path: %s\n"** % readme\_file\_path)  
 **raise  
  
  
class** ReadmeParser:  
 GENDER\_MATCHER = re.compile(**r'^[\s]\*gender[\s]\*:([^\n]\*)$'**,  
 re.IGNORECASE + re.DOTALL  
 + re.MULTILINE + re.UNICODE)  
 AGE\_RANGE\_MATCHER = re.compile(**r'^[\s]\*age[\s]\*range[\s]\*:([^\n]\*)$'**,  
 re.IGNORECASE + re.DOTALL  
 + re.MULTILINE + re.UNICODE)  
  
 @classmethod  
 **def** parse(cls, readme\_contents):  
 **return** (  
 Gender.get\_by\_value(next(iter(  
 cls.GENDER\_MATCHER.finditer(readme\_contents))).group(1)),  
 AgeRange.get\_by\_value(next(iter(  
 cls.AGE\_RANGE\_MATCHER.finditer(readme\_contents))).group(1)),  
 )  
  
  
**class** ValueSanitizer:  
 *# strip all whitespaces and some meaningful chars* VALUE\_STRIP\_REGEX = re.compile(**r'([\s\t;\r]|\[|\])'**, re.UNICODE)  
  
 @classmethod  
 **def** clean(cls, s):  
 **return** cls.VALUE\_STRIP\_REGEX.sub(**''**, s).casefold()  
  
  
**class** BaseMapMetaclass(type):  
 *"""  
 Sanitize \_MAP values and turn them to a readme\_value -> normalized\_value map  
 """* **def** \_\_new\_\_(cls, name, bases, props):  
 i = super().\_\_new\_\_(cls, name, bases, props)  
 **if** hasattr(i, **'\_MAP'**):  
 i.map = {  
 ValueSanitizer.clean(val): key  
 **for** key **in** i.\_MAP  
 **for** val **in** i.\_MAP[key]  
 }  
 **return** i  
  
  
**class** BaseMap(metaclass=BaseMapMetaclass):  
 map = **None** @classmethod  
 **def** get\_by\_value(cls, s):  
 clean\_value = ValueSanitizer.clean(s)  
 **return** cls.map.get(clean\_value, **'??? %s'** % clean\_value)  
  
  
**class** Gender(BaseMap):  
 MALE = **'m'** FEMALE = **'f'** NOT\_SPECIFIED = **'na'** \_MAP = {  
 MALE: {  
 **'Male'**,  
 **'Masculino'**,  
 **'Masculin'**,  
 **'Männlich'**,  
 **'Maschio'**,  
 **'mannelijk'**,  
 **'male'**,  
 **'Мужчина'**,  
 **'Erkek'**,  
 **'Άνδρας'**,  
 **'Machio'**,  
 **'Мъж'**,  
 **'Mashkull'**,  
 *# 'M�nnlich',* **'זכר'**,  
 **'Masculí'**,  
 **'Muško'**,  
 **'مذکر'**,  
 },  
 FEMALE: {  
 **'Femenino'**,  
 **'Female'**,  
 **'Weiblich'**,  
 **'Féminin'**,  
 **'female'**,  
 **'Femmina'**,  
 **'vrouwelijk'**,  
 **'Женщина'**,  
 **'Feminino'**,  
 **'Жена'**,  
 **'Γυναίκα'**,  
 **'Femër'**,  
 **'נקבה'**,  
 **'Жінка'**,  
 **'مونث'**,  
 **'kadın'**,  
 **'femení'**,  
 },  
 NOT\_SPECIFIED: {  
 **'inconnu'**,  
 **'unknown'**,  
 **'desconocido'**,  
 **'другое'**,  
 **'nondefinito'**,  
 **'Sélectionnez'**,  
 **'PleaseSelect'**,  
 **'Выберите'**,  
 **'make'**,  
 **'desconhecido'**,  
 **'неизвестен'**,  
 **'άγνωστο'**,  
 **'unbekannt'**,  
 **'desconegut'**,  
 **'PorfavorSeleccione'**,  
 **'Изберете'**,  
 **'لطفاانتخابکنید'**,  
 **'Selezionare'**,  
 **'Selecione'**,  
 **'nepoznato'**,  
 **'made'**, *# male??* **'Чоловік'**,  
 }  
 }  
  
  
**class** AgeRange(BaseMap):  
 YOUTH = **'y'** ADULT = **'a'** SENIOR = **'s'** NOT\_SPECIFIED = **'na'** \_MAP = {  
 YOUTH: {  
 **'Youth'**,  
 **'до25'**,  
 **'Jeune'**,  
 **'Giovane'**,  
 **'jeugd'**,  
 **'Jovem'**,  
 **'Jugendlicher'**,  
 **'Genç'**,  
 **'Νεαρός'**,  
 **'Niño'**,  
 **'Младеж/Девойка'**,  
 **'צעיר'**,  
 **'Jove'**,  
 **'До25років'**,  
 **'youth'**,  
 **'youth'**,  
 },  
 ADULT: {  
 **'Adult'**,  
 **'Adulto'**,  
 **'Adulte'**,  
 **'Erwachsener'**,  
 **'volwassen'**,  
 **'adult'**,  
 **'до55'**,  
 **'adult'**,  
 **'Ενήλικας'**,  
 **'Yetişkin'**,  
 **'Irritur'**,  
 **'До55років'**,  
 **'Възрастен'**,  
 **'מבוגר'**,  
 **'Odrasli'**,  
 **'بزرگسال'**,  
 **'Adult'**,  
 **'adult(bornin1983)'**,  
 **'بزرگسال'**,  
 },  
 SENIOR: {  
 **'Senior'**,  
 **'senior'**,  
 **'Anziano'**,  
 **'senior'**,  
 **'более55'**,  
 **'terceraedad'**,  
 },  
 NOT\_SPECIFIED: {  
 **'inconnu'**,  
 **'desconocido'**,  
 **'nondefinito'**,  
 **'Sélectionnez'**,  
 **'Please Select'**,  
 **'другое'**,  
 **'выберите'**,  
 **'неизвестен'**,  
 **'selecione'**,  
 **'unknown'**,  
 **'selezionare'**,  
 **'nepoznato'**,  
 **'изберете'**,  
 **'porfavorseleccione'**,  
 **'لطفاانتخابکنید'**,  
 }  
 }  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

## Приложение 2

pipe\_process\_recording.py:

*#!/usr/bin/env python3  
"""  
  
USAGE  
  
echo "b0138.wav" | ./pipe\_process\_recording.py  
  
find . -type f -name '\*.wav' | ./pipe\_process\_recording.py  
  
  
"""***import** sys  
**import** traceback  
**from** abc **import** ABCMeta  
**from** collections **import** deque  
**import** multiprocessing  
**from** timeit **import** default\_timer  
  
**from** scipy **import** signal  
**import** numpy **as** np  
**import** soundfile **as** sf  
**from** pip.utils **import** cached\_property  
**from** spectrum.lpc **import** lpc  
  
  
**"""  
https://github.com/maRce10/warbleR/blob/master/R/specan.R#L149  
https://github.com/cran/seewave/blob/master/R/seewave.r#L6030  
https://github.com/primaryobjects/voice-gender/blob/master/sound.R#L146  
  
https://cran.r-project.org/web/packages/seewave/index.html  
https://cran.r-project.org/web/packages/seewave/seewave.pdf 162  
https://cran.r-project.org/web/packages/tuneR/index.html  
https://cran.r-project.org/web/packages/tuneR/tuneR.pdf  
  
https://www.reddit.com/r/musicprogramming/comments/3mcqaf/r\_packages\_or\_python\_libraries\_for\_music/  
https://github.com/belangeo/pyo  
  
"""  
  
  
def** main():  
 table\_printer = TablePrinter()  
 table\_printer.print\_table\_header()  
  
 **with** Progress() **as** progress:  
 **with** multiprocessing.Pool(2) **as** p: *#* ***todo args*** results = deque()  
  
 *# read recording paths and add them to the processing queue* **for** recording\_file\_path **in** sys.stdin:  
 recording\_file\_path = recording\_file\_path.strip()  
 progress.inc\_added()  
 results.append(  
 (  
 recording\_file\_path,  
 p.apply\_async(  
 print\_audio\_record\_stats,  
 args=(table\_printer, recording\_file\_path),  
 callback=**lambda** \_: progress.inc\_completed(),  
 error\_callback=**lambda** \_: progress.inc\_completed())  
 )  
 )  
 progress.close()  
 p.close()  
 **while** results:  
 recording\_file\_path, r = results.popleft()  
 **try**:  
 r.get() *# this will raise if the task has failed* **except**:  
 progress.print\_exc(recording\_file\_path)  
 p.join()  
  
  
**def** print\_audio\_record\_stats(table\_printer, recording\_file\_path):  
 *"""Processes recording and prints CSV line with result"""* table\_printer.print\_record\_stats(  
 RecordStats.from\_file\_path(recording\_file\_path))  
  
  
**class** Progress:  
 *"""Prints current progress to the stderr"""* print\_min\_interval\_seconds = 1.0  
 out = sys.stderr  
  
 **def** \_\_init\_\_(self):  
 self.lock = multiprocessing.Lock()  
 self.clock\_started = default\_timer()  
 self.added = 0  
 self.completed = 0  
 self.prev\_print\_len = 0  
 self.closed = **False** self.last\_print\_clock = 0  
  
 **def** inc\_added(self):  
 **with** self.lock:  
 self.added += 1  
 self.print()  
  
 **def** inc\_completed(self):  
 **with** self.lock:  
 self.completed += 1  
 self.print()  
  
 **def** close(self):  
 self.closed = **True** self.print()  
  
 @property  
 **def** elapsed(self):  
 **return** default\_timer() - self.clock\_started  
  
 @property  
 **def** progress(self):  
 **return** self.completed / self.added \* 100  
  
 **def** \_\_repr\_\_(self):  
 **return 'elapsed: %.1fs, added: %s, completed: %s, '** \  
 **'progress: %.1f%%'** \  
 **'%s'** % (  
 self.elapsed, self.added, self.completed,  
 self.progress,  
 [**''**, **', closed'**][self.closed])  
  
 **def** print(self):  
 self.\_print()  
  
 **def** force\_print(self):  
 self.\_print(**True**)  
  
 **def** print\_exc(self, recording\_file\_path):  
 *"""call in an exception block"""* self.out.write(**'\n'**)  
 self.out.write(recording\_file\_path)  
 self.out.write(**'\n'**)  
 self.out.flush()  
 traceback.print\_exc()  
  
 **def** \_\_enter\_\_(self):  
 **return** self  
  
 **def** \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 self.force\_print()  
 self.out.write(**'\n'**)  
  
 **def** \_is\_it\_time\_to\_print(self, cur\_clock):  
 **return** (cur\_clock   
 - self.last\_print\_clock) >= self.print\_min\_interval\_seconds  
  
 **def** \_print(self, force=**False**):  
 **with** self.lock:  
 cur\_clock = default\_timer()  
 **if not** force **and not** self.\_is\_it\_time\_to\_print(cur\_clock):  
 **return** self.last\_print\_clock = cur\_clock  
 *# overwrite last line with spaces* self.out.write(**'\r'**)  
 self.out.write(**' '** \* self.prev\_print\_len)  
 *# print current progress* self.out.write(**'\r'**)  
 s = repr(self)  
 self.prev\_print\_len = len(s)  
 self.out.write(s)  
 self.out.flush()  
  
  
**class** BaseRecordStats(metaclass=ABCMeta):  
 sample\_rate = **None** *# type: int* freq = **None** *# type: np.ndarray* duration = **None** *# type: float* audio\_data\_normalized = **None** *# type: np.ndarray* frequency\_limit = **None** *# type: np.ndarray***class** ThresholdMixin:  
 @classmethod  
 **def** threshold\_filter(cls, audio\_data, threshold):  
 threshold\_nonnorm = np.abs(audio\_data).max() \* (threshold / 100)  
 mask = audio\_data.copy()  
 mask[np.abs(audio\_data) <= threshold\_nonnorm] = 0  
 mask[np.abs(audio\_data) > threshold\_nonnorm] = 1  
 **return** audio\_data \* mask  
  
 @classmethod  
 **def** window\_step(cls, n, window\_len, window\_overlap):  
 **return** np.arange(1, n - window\_len,   
 window\_len - (window\_overlap \* window\_len / 100),   
 dtype=np.int32)  
  
  
**class** SpecPropRecordStatsMixin(BaseRecordStats):  
 @cached\_property  
 **def** amp(self): *# amplitude - normalized* **return** self.fft / self.fft.sum()  
  
 @cached\_property  
 **def** amp\_cumsum(self): *# cumulative sum of the amp* **return** self.amp.cumsum()  
  
 @cached\_property  
 **def** amp\_standard\_deviation(self): *# w* **return** self.amp.std(ddof=1)  
  
 @cached\_property  
 **def** mean(self): *# same as the centroid* **return** (self.amp \* self.freq).sum()  
  
 @cached\_property  
 **def** mode(self):  
 **return** self.freq[self.amp.argmax()]  
  
 @cached\_property  
 **def** spectral\_entropy(self): *# sh, sp.ent  
 # https://github.com/cran/seewave/blob/master/R/seewave.r  
 # function `sh`  
 # https://en.wiktionary.org/wiki/Shannon\_entropy* **assert** self.amp.min() >= 0, **'Amplitude must be >=0'  
 if** self.amp.sum() == 0:  
 **return** 0  
  
 amp = self.amp.copy()  
 amp[amp == 0] = 1e-7 *# prevent log(0) warning (division by zero).   
 #* ***todo maybe try np.nan?*** amp = amp / amp.sum()  
 **return** -(amp \* np.log(amp)).sum() / np.log(len(amp))  
  
 @cached\_property  
 **def** spectral\_flatness(self): *# sfm  
 # https://github.com/cran/seewave/blob/master/R/seewave.r  
 # function `sfm`  
 # https://en.wikipedia.org/wiki/Spectral\_flatness* **if** self.amp.sum() == 0:  
 **return** np.nan  
  
 amp = self.amp.copy()  
  
 **if** len(amp) > 4000:  
 *# undersample* amp = amp[::np.round(len(amp) / 256)] *# why 256?* amp[amp == 0] = 1e-5  
  
 geometric\_mean = np.prod(amp \*\* (1 / len(amp)))  
 arithmetic\_mean = amp.mean()  
 flatness = geometric\_mean / arithmetic\_mean  
 **return** flatness  
  
 @cached\_property  
 **def** standard\_deviation(self): *# sd* **return** np.sqrt((self.amp \* ((self.freq - self.mean) \*\* 2)).sum())  
  
 @cached\_property  
 **def** median(self):  
 **return** self.freq[len(self.amp\_cumsum[self.amp\_cumsum <= 0.5])]  
  
 @cached\_property  
 **def** Q25(self): *# first quartile* **return** self.freq[len(self.amp\_cumsum[self.amp\_cumsum <= 0.25])]  
  
 @cached\_property  
 **def** Q75(self): *# third quartile* **return** self.freq[len(self.amp\_cumsum[self.amp\_cumsum <= 0.75])]  
  
 @cached\_property  
 **def** IQR(self): *# interquantile range* **return** self.Q75 - self.Q25  
  
 @cached\_property  
 **def** skewness(self):  
 **return** (((self.amp - self.amp.mean()) \*\* 3).sum() / (len(self.fft) - 1)) / self.amp\_standard\_deviation \*\* 3  
  
 @cached\_property  
 **def** kurtosis(self):  
 **return** (((self.amp - self.amp.mean()) \*\* 4).sum() / (len(self.fft) - 1)) / self.amp\_standard\_deviation \*\* 4  
  
 @cached\_property  
 **def** fft(self):  
 window = np.hanning(len(self.audio\_data\_normalized))  
 audio\_data\_w = self.audio\_data\_normalized \* window  
  
 fft = FFT.real\_fft(audio\_data\_w)  
  
 *# take half of the FFT  
 # multiply by 2 and by a scaling factor related to the  
 # window shape, this saves the total energy  
 # see http://www.ni.com/white-paper/4278/en/,  
 # section Converting from a Two-Sided Power Spectrum to  
 # a Single-Sided Power Spectrum  
 # https://github.com/cran/seewave/blob/97501b30e1475710511d4a147d14c17b2d4ca96c/R/seewave.r#L5838* fft = 2 \* fft[0:len(self.audio\_data\_normalized) // 2] *# type: np.ndarray* fft = fft / fft.max() *# normalize* els\_per\_hz = len(fft) / (self.sample\_rate / 2)  
 lowlimit, uplimit = (els\_per\_hz \* self.frequency\_limit)\  
 .astype(dtype=np.int)  
 **return** fft[lowlimit:uplimit]  
  
 @cached\_property  
 **def** freq(self):  
 **return** np.linspace(\*self.frequency\_limit, num=len(self.fft))  
  
  
**class** FundamentalRecordStatsMixin(ThresholdMixin, BaseRecordStats):  
 @cached\_property  
 **def** mean\_fundamental(self): *# meanfun* **return** np.nanmean(self.fundamental\_frequency)  
  
 @cached\_property  
 **def** min\_fundamental(self): *# minfun* **return** np.nanmin(self.fundamental\_frequency)  
  
 @cached\_property  
 **def** max\_fundamental(self): *# maxfun* **return** np.nanmax(self.fundamental\_frequency)  
  
 @cached\_property  
 **def** fundamental\_frequency(self):  
 *# https://github.com/cran/seewave/blob/master/R/seewave.r#L3156  
 # function `fund`  
 # https://en.wikipedia.org/wiki/Fundamental\_frequency  
  
 # ff <- seewave::fund(r, f = r@samp.rate, ovlp = 50, threshold = threshold,  
 # fmax = 280, ylim=c(0, 280/1000), plot = FALSE, wl = wl)[, 2]* window\_len = 512 *# wl* threshold = 15  
 window\_overlap = 50 *# ovlp* half\_window\_len = window\_len // 2  
  
 audio\_data = self.audio\_data\_normalized.copy()  
  
 *# https://github.com/cran/tuneR/blob/master/R/readWave.R* audio\_data = audio\_data \* (2 \*\* 15)  
  
 **if** threshold **is not None**:  
 audio\_data = self.threshold\_filter(audio\_data, threshold)  
  
 audio\_data[audio\_data == 0] = 1e-6  
  
 step = self.window\_step(len(audio\_data), window\_len, window\_overlap)  
 step\_len = len(step) *# N* z1 = np.zeros((window\_len, step\_len), dtype=np.float64)  
  
 **for** idx **in** range(step\_len):  
 i = step[idx]  
  
 l = FFT.real\_fft(audio\_data[i - 1: (window\_len + i - 1)])  
 l[l == 0] = np.nan *# fix division by 0 warning* l = l.astype(dtype=np.float64)  
  
 *# https://stat.ethz.ch/R-manual/R-devel/library/stats/html/fft.html* z1[:, idx] = FFT.real\_ifft(np.log(l))  
  
 z = z1[0: half\_window\_len]  
 z[np.isnan(z) | np.isneginf(z)] = 0  
 fmaxi = self.sample\_rate // self.frequency\_limit[1]  
 tfund = np.zeros(step\_len)  
 **for** k **in** range(step\_len):  
 *#* ***todo loss of precision in the z.*** *# argmax yields wrong values sometimes  
 # I think that the problem is in the R:  
 # because raising precision in numpy yields  
 # the same answer.  
 # Though there's no way to change precision in the R  
 # but they say that it is the double (float64).* tfund[k] = z[fmaxi:, k].argmax() + 1  
  
 tfund[tfund == 1] = np.nan  
 ffund = self.sample\_rate / (tfund + fmaxi - 1)  
  
 **return** ffund  
  
  
**class** DominantRecordStatsMixin(ThresholdMixin, BaseRecordStats):  
 @cached\_property  
 **def** mean\_dominant(self): *# meandom* **return** np.nanmean(self.dominant\_frequency)  
  
 @cached\_property  
 **def** min\_dominant(self): *# mindom* **return** np.nanmin(self.dominant\_frequency)  
  
 @cached\_property  
 **def** max\_dominant(self): *# maxdom* **return** np.nanmax(self.dominant\_frequency)  
  
 @cached\_property  
 **def** range\_dominant(self): *# dfrange* **return** self.max\_dominant - self.min\_dominant  
  
 @cached\_property  
 **def** \_notnan\_dominant\_frequency(self):  
 **return** self.dominant\_frequency[np.negative(np.isnan(  
 self.dominant\_frequency))]  
  
 @cached\_property  
 **def** start\_dominant(self): *# startdom* **return** self.\_notnan\_dominant\_frequency[0]  
  
 @cached\_property  
 **def** end\_dominant(self): *# enddom* **return** self.\_notnan\_dominant\_frequency[-1]  
  
 @cached\_property  
 **def** slope\_dominant(self): *# dfslope* **return** (self.end\_dominant - self.start\_dominant) / self.duration  
  
 @cached\_property  
 **def** modulation\_index(self): *# modindx  
 # Calculated as the accumulated absolute difference between   
 # adjacent measurements of fundamental frequencies divided   
 # by the frequency range* **if** self.min\_dominant == self.max\_dominant:  
 **return** 0  
  
 changes = np.abs(np.diff(self.dominant\_frequency))  
  
 **return** np.nanmean(changes) / self.range\_dominant  
  
 @cached\_property  
 **def** dominant\_frequency(self):  
 *# https://github.com/cran/seewave/blob/master/R/seewave.r#L1620  
 # function `dfreq`  
  
 # y <- seewave::dfreq(r, f = r@samp.rate, wl = wl, ylim=c(0, 280/1000),  
 # ovlp = 0, plot = FALSE, threshold = threshold, bandpass = b \* 1000,  
 # fftw = TRUE)[, 2]* bandpass = np.array([0, 22000]) *# b. 0, 22* threshold = 15  
 window\_len = 512 *# wl* window\_overlap = 0 *# ovlp* **if** bandpass[1] > np.ceil(self.sample\_rate / 2) - 1:  
 bandpass[1] = np.ceil(self.sample\_rate / 2) - 1   
  
 **assert** bandpass.shape == (2,), \  
 **'The argument "bandpass" should be a numeric vector of length 2'** **assert** bandpass[0] <= bandpass[1], \  
 **'The first element of "bandpass" has to be inferior to the'** \  
 + **' second element, i.e. bandpass[0] < bandpass[1]'  
 assert** bandpass[0] != bandpass[1], \  
 **'The limits of the bandpass have to be different'** audio\_data = self.audio\_data\_normalized.copy()  
  
 *# https://github.com/cran/tuneR/blob/master/R/readWave.R* **if** threshold **is not None**:  
 audio\_data = self.threshold\_filter(audio\_data, threshold)  
  
 step = self.window\_step(len(audio\_data), window\_len, window\_overlap)  
  
 step = np.round(step)  
 y1 = self.\_stft(audio\_data, window\_len, step)  
  
 lowlimit, upperlimit = np.round(  
 window\_len \* bandpass / self.sample\_rate).astype(dtype=np.int)  
  
 y1[:max(lowlimit - 1, 0)] = 0  
 y1[upperlimit:] = 0  
  
 maxi = np.max(y1, axis=0)  
  
 *#* ***todo loss in precision here*** y2 = np.argmax(y1, axis=0).astype(dtype=np.float64) + 1  
 y2[maxi == 0] = np.nan *# NA* y = (self.sample\_rate \* y2 - self.sample\_rate) / window\_len  
 **return** y  
  
 @classmethod  
 **def** \_stft(cls, wave, wl, step):  
 *# https://github.com/cran/seewave/blob/master/R/seewave.r#L7782* window = np.hanning(wl)  
  
 *#* ***todo ??? optimization fftw::planFFT(wl+zp)*** z = np.empty((len(step), len(window)), dtype=np.float64)  
 **for** i, x **in** enumerate(step):  
 z[i] = FFT.real\_fft(wave[x - 1:(wl + x - 1), ] \* window)  
  
 z = z.transpose()  
 z = z[0:wl // 2] *# to keep only the relevant frequencies (half of the FFT)* z = z / wl *# scaling by the original number of fft points  
  
 # multiplied by 2 to save the total energy see  
 # http://www.ni.com/white-paper/4278/en/  
 # section Converting from a Two-Sided Power Spectrum to a   
 # Single-Sided Power Spectrum* z = 2 \* z *# np.abs(z)* z = z / z.max()  
 **return** z  
  
  
**class** VoiceTractLengthRecordStatsMixin(BaseRecordStats):  
  
 @cached\_property  
 **def** vocal\_tract\_length\_vector(self):  
 formants = self.formants[self.formants != 0]  
 idx = np.arange(0, len(formants))  
 **return** 35000 \* ((idx + 1) / 2 - 0.25) / formants  
  
 @cached\_property  
 **def** mean\_vtl(self):  
 **return** np.nanmean(self.vocal\_tract\_length\_vector)  
  
 @cached\_property  
 **def** min\_vtl(self):  
 **return** np.nanmin(self.vocal\_tract\_length\_vector)  
  
 @cached\_property  
 **def** max\_vtl(self):  
 **return** np.nanmax(self.vocal\_tract\_length\_vector)  
  
 @cached\_property  
 **def** formants(self):  
 *# http://stackoverflow.com/questions/25107806/estimate-formants-using-lpc-in-python  
 # https://github.com/cran/phonTools/blob/master/R/findformants.R  
  
 # https://github.com/danilobellini/audiolazy* minformant = 200  
 maxbw = 600  
 maxformant = self.sample\_rate / 2  
  
 w = np.hanning(len(self.audio\_data\_normalized))  
  
 *# Apply window and high pass filter.* x1 = self.audio\_data\_normalized \* w  
 *# x1 = signal.lfilter([1.], [1., 0.63], x1)* x1 = signal.lfilter(  
 [1., -np.exp(-2 \* np.math.pi \* 50 / self.sample\_rate)], [1.], x1)  
  
 *# Get LPC.* A, \_ = lpc(x1, round(self.sample\_rate / 1000) + 3)  
 A = np.insert(A, 0, 1, axis=0) *# spectrum's LPC skips first '1' value.  
  
 # https://github.com/cran/phonTools/blob/master/R/findformants.R#L16* **assert** len(A) != 1  
  
 roots = np.roots(A)  
 roots = roots[np.imag(roots) >= 0]  
  
 angles = np.arctan2(np.imag(roots), np.real(roots))  
  
 bws = -(self.sample\_rate / np.math.pi) \* np.log(np.abs(roots))  
  
 frqs = angles \* (self.sample\_rate / (2 \* np.math.pi))  
 frqs = frqs[np.all(np.array(  
 [frqs > minformant, frqs < maxformant, bws < maxbw]), axis=0)]  
 frqs.sort()  
 **return** frqs  
  
  
**class** RecordStats(SpecPropRecordStatsMixin, FundamentalRecordStatsMixin,   
 DominantRecordStatsMixin, VoiceTractLengthRecordStatsMixin,   
 BaseRecordStats):  
 frequency\_limit = np.array((0, 280)) *# range in hz to analyze.   
 # That is close to the human vocal range.* audio\_sample\_max\_length\_seconds = 20  
  
 @classmethod  
 **def** from\_file\_path(cls, recording\_file\_path) -> **'RecordStats'**:  
 audio\_data, sample\_rate = sf.read(recording\_file\_path)  
 **return** cls(audio\_data, sample\_rate, recording\_file\_path)  
  
 **def** \_\_init\_\_(self, audio\_data, sample\_rate, audio\_name):  
 self.audio\_name = audio\_name  
 self.audio\_data\_normalized = \  
 audio\_data[:sample\_rate \* self.audio\_sample\_max\_length\_seconds]  
 self.sample\_rate = sample\_rate  
  
 @cached\_property  
 **def** duration(self):  
 **return** len(self.audio\_data\_normalized) / self.sample\_rate  
  
  
**class** ThreadSafePrinter:  
 \_lock = multiprocessing.Lock()  
  
 @classmethod  
 **def** print(cls, s):  
 **with** cls.\_lock:  
 sys.stdout.write(str(s))  
  
 @classmethod  
 **def** print\_newline(cls, s):  
 **with** cls.\_lock:  
 sys.stdout.write(str(s))  
 sys.stdout.write(**'\n'**)  
  
  
**class** TablePrinter:  
 *"""  
 Thread-safe table printer  
 """* fields = [  
 (**''**, **lambda** \_: **'""'**),  
 (**'sound.files'**, **lambda** stats: **'"%s"'** % stats.audio\_name),  
 (**'selec'**, **lambda** stats: **'-1'**),  
 (**'duration'**, **lambda** stats: **'-1'**),  
 (**'meanfreq'**, **lambda** stats: **'{:.15f}'**.format(stats.mean / 1000)),  
 (**'sd'**, **lambda** stats: **'{:.15f}'**.format(stats.standard\_deviation / 1000)),  
 (**'median'**, **lambda** stats: **'{:.15f}'**.format(stats.median / 1000)),  
 (**'Q25'**, **lambda** stats: **'{:.15f}'**.format(stats.Q25 / 1000)),  
 (**'Q75'**, **lambda** stats: **'{:.15f}'**.format(stats.Q75 / 1000)),  
 (**'IQR'**, **lambda** stats: **'{:.15f}'**.format(stats.IQR / 1000)),  
 (**'skew'**, **lambda** stats: **'{:.15f}'**.format(stats.skewness)),  
 (**'kurt'**, **lambda** stats: **'{:.15f}'**.format(stats.kurtosis)),  
 (**'sp.ent'**, **lambda** stats: **'{:.15f}'**.format(stats.spectral\_entropy)),  
 (**'sfm'**, **lambda** stats: **'{:.15f}'**.format(stats.spectral\_flatness)),  
 (**'mode'**, **lambda** stats: **'{:.15f}'**.format(stats.mode / 1000)),  
 (**'meanfun'**, **lambda** stats: **'{:.15f}'**.format(stats.mean\_fundamental / 1000)),  
 (**'minfun'**, **lambda** stats: **'{:.15f}'**.format(stats.min\_fundamental / 1000)),  
 (**'maxfun'**, **lambda** stats: **'{:.15f}'**.format(stats.max\_fundamental / 1000)),  
 (**'meandom'**, **lambda** stats: **'{:.15f}'**.format(stats.mean\_dominant / 1000)),  
 (**'mindom'**, **lambda** stats: **'{:.15f}'**.format(stats.min\_dominant / 1000)),  
 (**'maxdom'**, **lambda** stats: **'{:.15f}'**.format(stats.max\_dominant / 1000)),  
 (**'dfrange'**, **lambda** stats: **'{:.15f}'**.format(stats.range\_dominant / 1000)),  
 (**'modindx'**, **lambda** stats: **'{:.15f}'**.format(stats.modulation\_index)),  
 (**'startdom'**, **lambda** stats: **'{:.15f}'**.format(stats.start\_dominant / 1000)),  
 (**'enddom'**, **lambda** stats: **'{:.15f}'**.format(stats.end\_dominant / 1000)),  
 (**'dfslope'**, **lambda** stats: **'{:.15f}'**.format(stats.slope\_dominant)),  
 (**'meanvtl'**, **lambda** stats: **'{:.15f}'**.format(stats.mean\_vtl)),  
 (**'minvtl'**, **lambda** stats: **'{:.15f}'**.format(stats.min\_vtl)),  
 (**'maxvtl'**, **lambda** stats: **'{:.15f}'**.format(stats.max\_vtl)),  
 ]  
  
 **def** print\_table\_header(self):  
 ThreadSafePrinter.print\_newline(**','**.join(**'"%s"'** % field  
 **for** field, \_ **in** self.fields))  
  
 **def** print\_record\_stats(self, record\_stats: RecordStats):  
 ThreadSafePrinter.print\_newline(**','**.join(func(record\_stats)  
 **for** \_, func **in** self.fields))  
  
  
**class** FFT:  
 @classmethod  
 **def** real\_fft(cls, data):  
 *# http://stackoverflow.com/questions/8158632/is-it-worth-offloading-fft-computation-to-an-embedded-gpu* fft = np.fft.fft(data)  
 fft = np.abs(fft) *# strip imaginary part* **return** fft  
  
 @classmethod  
 **def** real\_ifft(cls, data):  
 fft = np.fft.ifft(data)  
 fft = np.real(fft)  
 **return** fft  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

## Приложение 3

make\_dataset.py:

*#!/usr/bin/env python3  
"""  
  
Makes a usable for learning dataset from two csv files:  
voxforge\_readme.csv - see pipe\_voxforge\_readme.py  
voxforge\_processed\_wavs.csv - see voxforge\_sound\_processing.R  
  
Provides csv which looks like:  
"sound.files","meanfreq","sd","median","Q25","Q75","IQR","skew","kurt","sp.ent","sfm","mode","centroid","meanfun","minfun","maxfun","meandom","mindom","maxdom","dfrange","modindx","startdom","enddom","dfslope","sample\_name","gender","age\_range","readme\_file\_path"  
"Main/16kHz\_16bit/anonymous-20110323-mcm/wav/b0138.wav",0.14312527541694298,0.0967057486545892,0.133138856476079,0.0488448074679113,0.24781796966161,0.19897316219369895,23.0228048857441,705.807544913565,0.922600382655861,0.594745423492673,0,0.14312527541694298,0.142019123548646,0.0627450980392157,0.27586206896551696,0.29866803278688503,0,2.34375,2.34375,0.120251572327044,0.0,0.0,0.0,"Main/16kHz\_16bit/anonymous-20110323-mcm","m","y","./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README"  
"Main/16kHz\_16bit/anonymous-20110323-mcm/wav/b0143.wav",0.13953980371761499,0.0945213018704719,0.13869484151647,0.0492479801118707,0.235798632691113,0.186550652579242,26.698632565131103,846.4815850080311,0.9291249489556341,0.677109790956615,0,0.13953980371761499,0.147606502834263,0.064,0.27586206896551696,0.5710227272727271,0,4.09375,4.09375,0.144924233792868,3.53125,0.0,-0.1765625,"Main/16kHz\_16bit/anonymous-20110323-mcm","m","y","./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README"  
  
  
Undetermined values of gender/age range columns are stripped.  
Resulting table is an INNER JOIN of two given ones.  
  
  
---  
  
USAGE  
  
./make\_dataset.py ../input/voxforge\_readme.csv ../input/voxforge\_processed\_wavs.csv > ../input/voxforge\_dataset.csv  
  
---  
  
  
voxforge\_readme.csv:  
"gender","age\_range","readme\_file\_path"  
"m","y","./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README"  
"m","s","./Main/16kHz\_16bit/ColinBeckingham-20100128-gzl/etc/README"  
"m","a","./Main/16kHz\_16bit/JohanLingen-20080306-nov/etc/README"  
  
  
voxforge\_processed\_wavs.csv:  
"","sound.files","selec","duration","meanfreq","sd","median","Q25","Q75","IQR","skew","kurt","sp.ent","sfm","mode","centroid","meanfun","minfun","maxfun","meandom","mindom","maxdom","dfrange","modindx","startdom","enddom","dfslope"  
"1","Main/16kHz\_16bit/anonymous-20110323-mcm/wav/b0138.wav",0,20,0.143125275416943,0.0967057486545892,0.133138856476079,0.0488448074679113,0.24781796966161,0.198973162193699,23.0228048857441,705.807544913565,0.922600382655861,0.594745423492673,0,0.143125275416943,0.142019123548646,0.0627450980392157,0.275862068965517,0.298668032786885,0,2.34375,2.34375,0.120251572327044,0,0,0  
"2","Main/16kHz\_16bit/anonymous-20110323-mcm/wav/b0143.wav",0,20,0.139539803717615,0.0945213018704719,0.13869484151647,0.0492479801118707,0.235798632691113,0.186550652579242,26.6986325651311,846.481585008031,0.929124948955634,0.677109790956615,0,0.139539803717615,0.147606502834263,0.064,0.275862068965517,0.571022727272727,0,4.09375,4.09375,0.144924233792868,3.53125,0,-0.1765625  
  
  
"""***import** sys  
**import** csv  
  
**import** pandas **as** pd  
  
  
PROCESSED\_WAVS\_STRIP\_COLUMNS = (  
 **"Unnamed: 0"**,  
 **"selec"**,  
 **"duration"**,  
)  
  
  
**def** main():  
 *# read data frames* df\_voxforge\_readme = pd.read\_csv(sys.argv[1])  
 df\_voxforge\_processed\_wavs = pd.read\_csv(sys.argv[2])  
  
 **for** col **in** PROCESSED\_WAVS\_STRIP\_COLUMNS:  
 **del** df\_voxforge\_processed\_wavs[col]  
  
 *# filter out samples with incomplete/undetermined values* df\_voxforge\_readme = \  
 df\_voxforge\_readme[df\_voxforge\_readme[**'gender'**].isin([**'m'**, **'f'**])]  
 df\_voxforge\_readme = \  
 df\_voxforge\_readme[df\_voxforge\_readme[**'age\_range'**].isin([**'y'**, **'a'**, **'s'**])]  
 df\_voxforge\_readme[**'gender\_and\_age\_range'**] = \  
 df\_voxforge\_readme[**'gender'**].combine(df\_voxforge\_readme[**'age\_range'**],  
 **lambda** g, a: **"%s%s"** % (g, a))  
  
 *# create common column  
 # Example value: Main/16kHz\_16bit/anonymous-20110323-mcm  
 # Main/16kHz\_16bit/anonymous-20110323-mcm/wav/b0138.wav* df\_voxforge\_processed\_wavs[**'sample\_name'**] = \  
 df\_voxforge\_processed\_wavs[**'sound.files'**].map(  
 **lambda** s: s.strip()[0:s.index(**'/wav/'**)].strip(**'./'**))  
  
 *# ./Main/16kHz\_16bit/anonymous-20110323-mcm/etc/README* df\_voxforge\_readme[**'sample\_name'**] = \  
 df\_voxforge\_readme[**'readme\_file\_path'**].map(  
 **lambda** s: s.strip()[0:s.index(**'/etc/README'**)].strip(**'./'**))  
  
 *# inner join data frames* merged\_df = pd.merge(df\_voxforge\_processed\_wavs, df\_voxforge\_readme,  
 on=**'sample\_name'**)  
  
 sys.stdout.write(merged\_df.to\_csv(index=**None**, quoting=csv.QUOTE\_NONNUMERIC))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

## Приложение 4

voxforge\_classifiers\_overview.py:

*"""  
Usage:  
python voxforge\_classifiers\_overview.py 1 '../input/voxforge\_dataset\_py\_short.csv'  
  
1 - union adult and senior. 0 - leave split.  
  
"""***import** sys  
  
**from** sklearn **import** metrics  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.linear\_model **import** SGDClassifier  
**from** sklearn.model\_selection **import** GridSearchCV, train\_test\_split, \  
 cross\_val\_score  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.svm **import** SVC  
**from** sklearn.pipeline **import** Pipeline  
  
**from** learn.dataset **import** load\_dataset  
  
  
**def** train(data\_bunch, name, pipeline, parameters):  
 print()  
 print(**'Name: %s'** % name)  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 data\_bunch.data, data\_bunch.target, test\_size=0.2, random\_state=1)  
  
 clf = GridSearchCV(pipeline, parameters, verbose=1, n\_jobs=3)  
  
 clf.fit(X\_train, y\_train)  
  
 print(**'Best score: %0.3f'** % clf.best\_score\_)  
 print(**'Best parameters set: %s'** % clf.best\_params\_)  
  
 predicted = clf.predict(X\_test)  
  
 print(**'Cross validation scores: %s'** % cross\_val\_score(  
 estimator=clf.best\_estimator\_,  
 X=X\_train,  
 y=y\_train))  
  
 print(**'Classification report for classifier %s:\n%s\n'** % (clf.best\_estimator\_,  
 metrics.classification\_report(y\_test, predicted)),  
 **'Confusion matrix:\n%s'** % metrics.confusion\_matrix(y\_test, predicted))  
  
 **return** clf.best\_score\_  
  
  
**def** train\_all\_data\_bunches(dataset, name, pipeline, parameters):  
 print(**'---'**)  
  
 scores = [  
 train(dataset.gender, **'%s: gender'** % name, pipeline, parameters),  
 train(dataset.male, **'%s: male age'** % name, pipeline, parameters),  
 train(dataset.female, **'%s: female age'** % name, pipeline, parameters)  
 ]  
  
 print(**'Scores: %s'** % scores)  
 print(**'---'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = load\_dataset(sys.argv[2],  
 union\_age\_classes=[**'s'**, **'a'**] **if** bool(  
 int(sys.argv[1])) **else None**)  
 data.print\_info()  
  
 train\_all\_data\_bunches(data,  
 **'SVM'**,  
 Pipeline([(**'std\_scl'**, StandardScaler()),  
 (**'svc'**, SVC(random\_state=1))]),  
 [  
 {  
 **'svc\_\_C'**: [1, 10],  
 **'svc\_\_kernel'**: [**'linear'**]  
 },  
 {  
 **'svc\_\_C'**: [1, 5, 10, 50],  
 **'svc\_\_gamma'**: [0.1, 0.05, 0.03, 0.01, 0.005,  
 0.001],  
 **'svc\_\_kernel'**: [**'rbf'**],  
 },  
 ])  
  
 train\_all\_data\_bunches(data,  
 **'RF'**,  
 Pipeline([(**'std\_scl'**, StandardScaler()),  
 (**'rf'**,  
 RandomForestClassifier(random\_state=1))]),  
 [  
 {  
 **'rf\_\_n\_estimators'**: [5, 7, 10, 20, 50],  
 }  
 ])  
  
 train\_all\_data\_bunches(data,  
 **'SGD'**,  
 Pipeline([(**'std\_scl'**, StandardScaler()),  
 (**'sgd'**, SGDClassifier(random\_state=1))]),  
 [  
 {  
 **'sgd\_\_loss'**: [  
 **'hinge'**, **'log'**, **'modified\_huber'**,  
 **'perceptron'** ],  
 **'sgd\_\_penalty'**: [**'l1'**, **'l2'**, **'elasticnet'**],  
 **'sgd\_\_alpha'**: [0.001, 0.0001, 0.00001,  
 0.000001],  
 }  
 ])  
  
 train\_all\_data\_bunches(data,  
 **'KNN'**,  
 Pipeline([(**'std\_scl'**, StandardScaler()),  
 (**'knn'**, KNeighborsClassifier())]),  
 [  
 {  
 **'knn\_\_n\_neighbors'**: [5, 7, 10, 15],  
 **'knn\_\_algorithm'**: [**'ball\_tree'**],  
 **'knn\_\_leaf\_size'**: [18, 20, 22],  
 **'knn\_\_p'**: [1, 3, 4],  
 }  
 ])

## Приложение 5

dataset.py:

**import** pandas **as** pd  
**from** sklearn.datasets.base **import** Bunch  
**from** sklearn.preprocessing **import** LabelEncoder  
  
  
**class** load\_dataset:  
 *"""Load voxforge dataset from csv"""* **def** \_\_init\_\_(self, path,  
 gender\_data\_cols=**None**,  
 male\_data\_cols=**None**, female\_data\_cols=**None**,  
 union\_age\_classes=**None**):  
  
 df = pd.read\_csv(path)  
  
 count = df.shape[0]  
 df = df.dropna()  
 self.na\_count = count - df.shape[0]  
  
 **del** df[**'sound.files'**]  
 **del** df[**'sample\_name'**]  
 **del** df[**'readme\_file\_path'**]  
 **del** df[**'gender\_and\_age\_range'**]  
  
 cols = set(df.columns.values)  
 cols.discard(**'age\_range'**)  
 cols.discard(**'gender'**)  
  
 self.gender\_data\_cols = tuple(gender\_data\_cols **or** cols)  
 self.male\_data\_cols = tuple(male\_data\_cols **or** cols)  
 self.female\_data\_cols = tuple(female\_data\_cols **or** cols)  
  
 df\_gender = df[list(self.gender\_data\_cols) + [**'gender'**]]  
  
 df\_male = df[df.gender == **'m'**][  
 list(self.male\_data\_cols) + [**'age\_range'**]]  
 **if** union\_age\_classes:  
 df\_male.loc[df\_male.age\_range == union\_age\_classes[0],  
 **'age\_range'**] = union\_age\_classes[1]  
  
 df\_female = df[df.gender == **'f'**][  
 list(self.female\_data\_cols) + [**'age\_range'**]]  
 **if** union\_age\_classes:  
 df\_female.loc[df\_female.age\_range == union\_age\_classes[0],  
 **'age\_range'**] = union\_age\_classes[1]  
  
 self.gender = Bunch(data=df\_gender.iloc[:, :-1].values,  
 target=self.\_encode\_labels(  
 df\_gender.iloc[:, -1].values),  
 feature\_names=df\_gender.columns.values[:-1],  
 target\_names=df\_gender.iloc[:, -1].values)  
  
 self.male = Bunch(data=df\_male.iloc[:, :-1].values,  
 target=self.\_encode\_labels(  
 df\_male.iloc[:, -1].values),  
 feature\_names=df\_male.columns.values[:-1],  
 target\_names=df\_male.iloc[:, -1].values)  
  
 self.female = Bunch(data=df\_female.iloc[:, :-1].values,  
 target=self.\_encode\_labels(  
 df\_female.iloc[:, -1].values),  
 feature\_names=df\_female.columns.values[:-1],  
 target\_names=df\_female.iloc[:, -1].values)  
  
 **def** print\_info(self):  
 print(**"Number of dropped NA rows: %s"** % self.na\_count)  
 print(**"Number of gender samples: %s"** % len(self.gender.data))  
 print(**"Number of male age samples: %s"** % len(self.male.data))  
 print(**"Number of female age samples: %s"** % len(self.female.data))  
  
 print(**"Gender cols: %s"** % (self.gender\_data\_cols,))  
 print(**"Male age cols: %s"** % (self.male\_data\_cols,))  
 print(**"Female age cols: %s"** % (self.female\_data\_cols,))  
  
 @classmethod  
 **def** \_encode\_labels(cls, labels):  
 encoder = LabelEncoder()  
 **return** encoder.fit\_transform(labels)

## Приложение 6

Вывод voxforge\_classifiers\_overview.py:

PYTHONPATH=. python learn/voxforge\_classifiers\_overview.py 1 'input/voxforge\_dataset\_py\_short.csv'

Number of dropped NA rows: 2

Number of gender samples: 2526

Number of male age samples: 1498

Number of female age samples: 1028

Gender cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Male age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Female age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

---

Name: SVM: gender

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 4.1s finished

Best score: 0.933

Best parameters set: {'svc\_\_C': 10, 'svc\_\_kernel': 'rbf', 'svc\_\_gamma': 0.01}

Cross validation scores: [ 0.9272997 0.93016345 0.94056464]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=10, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.01, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.94 0.91 0.92 227

1 0.93 0.95 0.94 279

avg / total 0.93 0.93 0.93 506

Confusion matrix:

[[207 20]

[ 14 265]]

Name: SVM: male age

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 3.6s finished

Best score: 0.714

Best parameters set: {'svc\_\_C': 1, 'svc\_\_kernel': 'rbf', 'svc\_\_gamma': 0.1}

Cross validation scores: [ 0.7275 0.70676692 0.70676692]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=1, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.1, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.68 0.94 0.79 191

1 0.66 0.21 0.32 109

avg / total 0.67 0.67 0.62 300

Confusion matrix:

[[179 12]

[ 86 23]]

Name: SVM: female age

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 1.8s finished

Best score: 0.766

Best parameters set: {'svc\_\_C': 5, 'svc\_\_kernel': 'rbf', 'svc\_\_gamma': 0.03}

Cross validation scores: [ 0.80363636 0.74087591 0.75457875]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=5, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.03, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.77 0.78 0.77 117

1 0.70 0.70 0.70 89

avg / total 0.74 0.74 0.74 206

Confusion matrix:

[[91 26]

[27 62]]

Scores: [0.93267326732673272, 0.71368948247078468, 0.76642335766423353]

---

---

Name: RF: gender

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 1.0s finished

Best score: 0.930

Best parameters set: {'rf\_\_n\_estimators': 50}

Cross validation scores: [ 0.92878338 0.92421991 0.93759287]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.91 0.91 0.91 227

1 0.93 0.92 0.93 279

avg / total 0.92 0.92 0.92 506

Confusion matrix:

[[207 20]

[ 21 258]]

Name: RF: male age

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 0.9s finished

Best score: 0.712

Best parameters set: {'rf\_\_n\_estimators': 50}

Cross validation scores: [ 0.7 0.70676692 0.72932331]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.68 0.91 0.77 191

1 0.59 0.24 0.34 109

avg / total 0.64 0.66 0.62 300

Confusion matrix:

[[173 18]

[ 83 26]]

Name: RF: female age

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 0.8s finished

Best score: 0.760

Best parameters set: {'rf\_\_n\_estimators': 50}

Cross validation scores: [ 0.76363636 0.75912409 0.75824176]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.82 0.81 0.82 117

1 0.76 0.76 0.76 89

avg / total 0.79 0.79 0.79 206

Confusion matrix:

[[95 22]

[21 68]]

Scores: [0.93019801980198025, 0.71202003338898168, 0.76034063260340634]

---

---

Name: SGD: gender

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.5s finished

Best score: 0.913

Best parameters set: {'sgd\_\_penalty': 'l1', 'sgd\_\_loss': 'log', 'sgd\_\_alpha': 0.001}

Cross validation scores: [ 0.89465875 0.91679049 0.92867756]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=0.001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='log', n\_iter=5, n\_jobs=1,

penalty='l1', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.90 0.89 0.89 227

1 0.91 0.92 0.91 279

avg / total 0.90 0.90 0.90 506

Confusion matrix:

[[201 26]

[ 23 256]]

Name: SGD: male age

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.4s finished

Best score: 0.637

Best parameters set: {'sgd\_\_penalty': 'l1', 'sgd\_\_loss': 'log', 'sgd\_\_alpha': 0.001}

Cross validation scores: [ 0.6375 0.62155388 0.65162907]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=0.001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='log', n\_iter=5, n\_jobs=1,

penalty='l1', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.67 0.80 0.73 191

1 0.47 0.32 0.38 109

avg / total 0.60 0.62 0.60 300

Confusion matrix:

[[152 39]

[ 74 35]]

Name: SGD: female age

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.4s finished

Best score: 0.661

Best parameters set: {'sgd\_\_penalty': 'l2', 'sgd\_\_loss': 'perceptron', 'sgd\_\_alpha': 1e-05}

Cross validation scores: [ 0.63636364 0.65693431 0.68864469]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=1e-05, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='perceptron', n\_iter=5, n\_jobs=1,

penalty='l2', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.68 0.79 0.73 117

1 0.64 0.51 0.57 89

avg / total 0.66 0.67 0.66 206

Confusion matrix:

[[92 25]

[44 45]]

Scores: [0.9133663366336634, 0.63689482470784642, 0.66058394160583944]

---

---

Name: KNN: gender

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 50 tasks | elapsed: 37.6s

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 1.3min finished

Best score: 0.899

Best parameters set: {'knn\_\_leaf\_size': 18, 'knn\_\_n\_neighbors': 5, 'knn\_\_p': 1, 'knn\_\_algorithm': 'ball\_tree'}

Cross validation scores: [ 0.89465875 0.90193165 0.89895988]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=5, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.90 0.85 0.88 227

1 0.89 0.92 0.90 279

avg / total 0.89 0.89 0.89 506

Confusion matrix:

[[194 33]

[ 22 257]]

Name: KNN: male age

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 67 tasks | elapsed: 17.8s

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 28.6s finished

Best score: 0.709

Best parameters set: {'knn\_\_leaf\_size': 18, 'knn\_\_n\_neighbors': 10, 'knn\_\_p': 1, 'knn\_\_algorithm': 'ball\_tree'}

Cross validation scores: [ 0.6925 0.72681704 0.70676692]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=10, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.68 0.96 0.80 191

1 0.76 0.20 0.32 109

avg / total 0.71 0.69 0.62 300

Confusion matrix:

[[184 7]

[ 87 22]]

Name: KNN: female age

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 13.3s finished

Best score: 0.751

Best parameters set: {'knn\_\_leaf\_size': 18, 'knn\_\_n\_neighbors': 5, 'knn\_\_p': 1, 'knn\_\_algorithm': 'ball\_tree'}

Cross validation scores: [ 0.74545455 0.74087591 0.76556777]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=5, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.79 0.76 0.78 117

1 0.70 0.74 0.72 89

avg / total 0.75 0.75 0.75 206

Confusion matrix:

[[89 28]

[23 66]]

Scores: [0.89851485148514854, 0.70868113522537568, 0.75060827250608275]

---

PYTHONPATH=. python learn/voxforge\_classifiers\_overview.py 0 'input/voxforge\_dataset\_py\_short.csv'

Number of dropped NA rows: 2

Number of gender samples: 2526

Number of male age samples: 1498

Number of female age samples: 1028

Gender cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Male age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Female age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

---

Name: SVM: gender

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 4.1s finished

Best score: 0.933

Best parameters set: {'svc\_\_kernel': 'rbf', 'svc\_\_C': 10, 'svc\_\_gamma': 0.01}

Cross validation scores: [ 0.9272997 0.93016345 0.94056464]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=10, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.01, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.94 0.91 0.92 227

1 0.93 0.95 0.94 279

avg / total 0.93 0.93 0.93 506

Confusion matrix:

[[207 20]

[ 14 265]]

Name: SVM: male age

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 4.6s finished

Best score: 0.537

Best parameters set: {'svc\_\_kernel': 'rbf', 'svc\_\_C': 5, 'svc\_\_gamma': 0.1}

Cross validation scores: [ 0.5325 0.56140351 0.51629073]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=5, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.1, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.42 0.47 0.44 95

1 0.58 0.55 0.56 96

2 0.46 0.42 0.44 109

avg / total 0.48 0.48 0.48 300

Confusion matrix:

[[45 14 36]

[25 53 18]

[38 25 46]]

Name: SVM: female age

Fitting 3 folds for each of 26 candidates, totalling 78 fits

[Parallel(n\_jobs=3)]: Done 78 out of 78 | elapsed: 1.8s finished

Best score: 0.768

Best parameters set: {'svc\_\_kernel': 'rbf', 'svc\_\_C': 5, 'svc\_\_gamma': 0.03}

Cross validation scores: [ 0.80363636 0.73818182 0.76102941]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('svc', SVC(C=5, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape=None, degree=3, gamma=0.03, kernel='rbf',

max\_iter=-1, probability=False, random\_state=1, shrinking=True,

tol=0.001, verbose=False))]):

precision recall f1-score support

0 0.76 0.76 0.76 110

1 0.71 0.71 0.71 7

2 0.69 0.69 0.69 89

avg / total 0.73 0.73 0.73 206

Confusion matrix:

[[84 0 26]

[ 0 5 2]

[26 2 61]]

Scores: [0.93267326732673272, 0.53672787979966607, 0.76763990267639903]

---

---

Name: RF: gender

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 1.1s finished

Best score: 0.930

Best parameters set: {'rf\_\_n\_estimators': 20}

Cross validation scores: [ 0.92878338 0.92570579 0.93610698]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=20, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.91 0.90 0.91 227

1 0.92 0.93 0.92 279

avg / total 0.92 0.92 0.92 506

Confusion matrix:

[[205 22]

[ 20 259]]

Name: RF: male age

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 0.9s finished

Best score: 0.544

Best parameters set: {'rf\_\_n\_estimators': 50}

Cross validation scores: [ 0.565 0.55137845 0.51629073]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.44 0.49 0.46 95

1 0.59 0.58 0.59 96

2 0.49 0.44 0.47 109

avg / total 0.51 0.50 0.50 300

Confusion matrix:

[[47 18 30]

[21 56 19]

[40 21 48]]

Name: RF: female age

Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n\_jobs=3)]: Done 15 out of 15 | elapsed: 0.9s finished

Best score: 0.770

Best parameters set: {'rf\_\_n\_estimators': 50}

Cross validation scores: [ 0.77090909 0.74909091 0.79044118]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('rf', RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=1,

verbose=0, warm\_start=False))]):

precision recall f1-score support

0 0.83 0.77 0.80 110

1 0.83 0.71 0.77 7

2 0.72 0.79 0.75 89

avg / total 0.78 0.78 0.78 206

Confusion matrix:

[[85 0 25]

[ 0 5 2]

[18 1 70]]

Scores: [0.93019801980198025, 0.54424040066777968, 0.77007299270072993]

---

---

Name: SGD: gender

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.7s finished

Best score: 0.913

Best parameters set: {'sgd\_\_alpha': 0.001, 'sgd\_\_penalty': 'l1', 'sgd\_\_loss': 'log'}

Cross validation scores: [ 0.89465875 0.91679049 0.92867756]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=0.001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='log', n\_iter=5, n\_jobs=1,

penalty='l1', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.90 0.89 0.89 227

1 0.91 0.92 0.91 279

avg / total 0.90 0.90 0.90 506

Confusion matrix:

[[201 26]

[ 23 256]]

Name: SGD: male age

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.7s finished

Best score: 0.441

Best parameters set: {'sgd\_\_alpha': 0.001, 'sgd\_\_penalty': 'l1', 'sgd\_\_loss': 'hinge'}

Cross validation scores: [ 0.445 0.42857143 0.44862155]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=0.001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='hinge', n\_iter=5, n\_jobs=1,

penalty='l1', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.34 0.54 0.42 95

1 0.45 0.27 0.34 96

2 0.42 0.36 0.39 109

avg / total 0.41 0.39 0.38 300

Confusion matrix:

[[51 16 28]

[45 26 25]

[54 16 39]]

Name: SGD: female age

Fitting 3 folds for each of 48 candidates, totalling 144 fits

[Parallel(n\_jobs=3)]: Done 144 out of 144 | elapsed: 0.6s finished

Best score: 0.644

Best parameters set: {'sgd\_\_alpha': 0.001, 'sgd\_\_penalty': 'l1', 'sgd\_\_loss': 'modified\_huber'}

Cross validation scores: [ 0.64363636 0.62909091 0.65808824]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('sgd', SGDClassifier(alpha=0.001, average=False, class\_weight=None, epsilon=0.1,

eta0=0.0, fit\_intercept=True, l1\_ratio=0.15,

learning\_rate='optimal', loss='modified\_huber', n\_iter=5, n\_jobs=1,

penalty='l1', power\_t=0.5, random\_state=1, shuffle=True, verbose=0,

warm\_start=False))]):

precision recall f1-score support

0 0.74 0.82 0.78 110

1 0.55 0.86 0.67 7

2 0.71 0.58 0.64 89

avg / total 0.72 0.72 0.71 206

Confusion matrix:

[[90 0 20]

[ 0 6 1]

[32 5 52]]

Scores: [0.9133663366336634, 0.44073455759599334, 0.64355231143552316]

---

---

Name: KNN: gender

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 35.1s

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 1.4min finished

Best score: 0.899

Best parameters set: {'knn\_\_algorithm': 'ball\_tree', 'knn\_\_n\_neighbors': 5, 'knn\_\_p': 1, 'knn\_\_leaf\_size': 18}

Cross validation scores: [ 0.89465875 0.90193165 0.89895988]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=5, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.90 0.85 0.88 227

1 0.89 0.92 0.90 279

avg / total 0.89 0.89 0.89 506

Confusion matrix:

[[194 33]

[ 22 257]]

Name: KNN: male age

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 67 tasks | elapsed: 17.8s

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 29.4s finished

Best score: 0.517

Best parameters set: {'knn\_\_algorithm': 'ball\_tree', 'knn\_\_n\_neighbors': 7, 'knn\_\_p': 1, 'knn\_\_leaf\_size': 18}

Cross validation scores: [ 0.4925 0.55137845 0.50626566]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=7, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.40 0.46 0.43 95

1 0.51 0.69 0.59 96

2 0.48 0.27 0.34 109

avg / total 0.46 0.46 0.45 300

Confusion matrix:

[[44 29 22]

[20 66 10]

[46 34 29]]

Name: KNN: female age

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n\_jobs=3)]: Done 108 out of 108 | elapsed: 15.0s finished

Best score: 0.747

Best parameters set: {'knn\_\_algorithm': 'ball\_tree', 'knn\_\_n\_neighbors': 5, 'knn\_\_p': 1, 'knn\_\_leaf\_size': 18}

Cross validation scores: [ 0.73818182 0.73818182 0.76470588]

Classification report for classifier Pipeline(steps=[('std\_scl', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('knn', KNeighborsClassifier(algorithm='ball\_tree', leaf\_size=18, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=5, p=1,

weights='uniform'))]):

precision recall f1-score support

0 0.79 0.77 0.78 110

1 0.80 0.57 0.67 7

2 0.70 0.74 0.72 89

avg / total 0.75 0.75 0.75 206

Confusion matrix:

[[85 0 25]

[ 0 4 3]

[22 1 66]]

Scores: [0.89851485148514854, 0.51669449081803009, 0.74695863746958635]

---

## Приложение 7

voxforge\_rf.py:

*"""  
Usage:  
python voxforge\_rf.py '../input/voxforge\_dataset\_py\_short.csv'  
"""***import** sys  
  
**from** sklearn **import** metrics  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.model\_selection **import** cross\_val\_score, train\_test\_split  
  
**from** learn.dataset **import** load\_dataset  
  
  
**def** train(data\_bunch, graph\_name):  
 print()  
 print(graph\_name)  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 data\_bunch.data, data\_bunch.target,  
 test\_size=0.2, random\_state=1)  
  
 clf = RandomForestClassifier(n\_estimators=50, n\_jobs=3, verbose=1)  
  
 clf.fit(X\_train, y\_train)  
  
 predicted = clf.predict(X\_test)  
  
 print(**'Score: %.3f'** % clf.score(X\_test, y\_test))  
  
 print(**'Cross validation scores: %s'** % cross\_val\_score(estimator=clf,  
 X=X\_train,  
 y=y\_train))  
  
 print(**'Classification report for classifier %s:\n%s\n'** % (clf, metrics.classification\_report(y\_test, predicted)),  
 **'Confusion matrix:\n%s'** % metrics.confusion\_matrix(y\_test, predicted))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = load\_dataset(sys.argv[1], union\_age\_classes=[**'s'**, **'a'**])  
 data.print\_info()  
  
 train(data.gender, **"gender"**)  
 train(data.male, **"male"**)  
 train(data.female, **"female"**)

## Приложение 8

Вывод voxforge\_rf.py:

PYTHONPATH=. python learn/voxforge\_rf.py 'input/voxforge\_dataset\_py\_short.csv'

Number of dropped NA rows: 2

Number of gender samples: 2526

Number of male age samples: 1498

Number of female age samples: 1028

Gender cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Male age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

Female age cols: ('skew', 'sfm', 'minvtl', 'centroid', 'meanfun', 'dfrange', 'maxvtl', 'meanvtl', 'minfun', 'maxdom', 'startdom', 'sp.ent', 'mode', 'IQR', 'mindom', 'median', 'dfslope', 'sd', 'meanfreq', 'meandom', 'Q75', 'maxfun', 'Q25', 'modindx', 'enddom', 'kurt')

gender

Score: 0.939

Cross validation scores: [ 0.93620178 0.9346211 0.93164933]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False):

precision recall f1-score support

0 0.93 0.94 0.93 227

1 0.95 0.94 0.94 279

avg / total 0.94 0.94 0.94 506

Confusion matrix:

[[213 14]

[ 17 262]]

male

Score: 0.693

Cross validation scores: [ 0.6975 0.70175439 0.71679198]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False):

precision recall f1-score support

0 0.69 0.93 0.79 191

1 0.70 0.28 0.39 109

avg / total 0.69 0.69 0.65 300

Confusion matrix:

[[178 13]

[ 79 30]]

female

Score: 0.752

Cross validation scores: [ 0.76 0.74817518 0.78754579]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=1, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False):

precision recall f1-score support

0 0.81 0.74 0.77 117

1 0.69 0.76 0.73 89

avg / total 0.76 0.75 0.75 206

Confusion matrix:

[[87 30]

[21 68]]

PYTHONPATH=. python learn/voxforge\_rf.py 'input/voxforge\_dataset\_py\_full.csv'

Number of gender samples: 177159

Number of male samples: 156242

Number of female samples: 20917

gender

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 25.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 28.4s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

Score: 0.972

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 15.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 17.0s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.2s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 15.5s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 17.7s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.2s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 16.4s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 18.3s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.2s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

Cross validation scores: [ 0.97072582 0.97246095 0.97178358]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=3, oob\_score=False, random\_state=None,

verbose=1, warm\_start=False):

precision recall f1-score support

0 0.89 0.87 0.88 4176

1 0.98 0.99 0.98 31256

avg / total 0.97 0.97 0.97 35432

Confusion matrix:

[[ 3622 554]

[ 451 30805]]

male

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 23.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 25.9s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.2s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.2s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.2s finished

Score: 0.890

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 13.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 15.0s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.3s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 13.5s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 15.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.3s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 15.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 17.1s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.3s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.3s finished

Cross validation scores: [ 0.88892356 0.89021697 0.89093702]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=3, oob\_score=False, random\_state=None,

verbose=1, warm\_start=False):

precision recall f1-score support

0 0.89 1.00 0.94 27372

1 0.87 0.14 0.24 3877

avg / total 0.89 0.89 0.85 31249

Confusion matrix:

[[27291 81]

[ 3345 532]]

female

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 2.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 2.4s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.0s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.0s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.0s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.0s finished

Score: 0.948

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 1.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 1.3s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.0s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.0s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 1.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 1.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.0s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.0s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 1.1s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 1.2s finished

[Parallel(n\_jobs=3)]: Done 44 tasks | elapsed: 0.0s

[Parallel(n\_jobs=3)]: Done 50 out of 50 | elapsed: 0.0s finished

Cross validation scores: [ 0.94765149 0.95069918 0.94925587]

Classification report for classifier RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=50, n\_jobs=3, oob\_score=False, random\_state=None,

verbose=1, warm\_start=False):

precision recall f1-score support

0 0.95 1.00 0.97 3937

1 0.80 0.16 0.27 247

avg / total 0.94 0.95 0.93 4184

Confusion matrix:

[[3927 10]

[ 207 40]]

## Приложение 9

voxforge\_cart.py:

*"""  
Usage:  
python voxforge\_cart.py '../input/voxforge\_dataset\_py\_short.csv'  
  
"""***import** sys  
  
**import** pandas **as** pd  
**from** sklearn **import** tree  
**import** pydotplus  
  
**from** learn.dataset **import** load\_dataset  
  
  
**def** train(data\_bunch, graph\_name):  
 print()  
 print(graph\_name)  
  
 clf = tree.DecisionTreeClassifier(  
 *# max\_depth=4* )  
  
 clf.fit(data\_bunch.data, data\_bunch.target)  
  
 print(pd.Series(data=clf.feature\_importances\_,  
 index=data\_bunch.feature\_names).sort\_values(ascending=**False**))  
  
 dot\_data = tree.export\_graphviz(clf, out\_file=**None**,  
 feature\_names=data\_bunch.feature\_names,  
 class\_names=data\_bunch.target\_names,  
 filled=**True**, rounded=**True**,  
 special\_characters=**True** )  
 graph = pydotplus.graph\_from\_dot\_data(dot\_data)  
 graph.write\_png(**"%s.png"** % graph\_name)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = load\_dataset(sys.argv[1], union\_age\_classes=[**'s'**, **'a'**])  
 data.print\_info()  
  
 train(data.gender, **"gender"**)  
 train(data.male, **"male"**)  
 train(data.female, **"female"**)

## Приложение 10

voxforge\_rf\_model\_validation.py:

*"""  
Usage:  
python voxforge\_rf\_model\_validation.py '../input/voxforge\_dataset\_py\_short.csv'  
"""***import** sys  
  
**import** numpy **as** np  
**import** scipy  
**from** sklearn.cross\_validation **import** StratifiedKFold, ShuffleSplit  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.metrics **import** auc, roc\_curve  
**from** sklearn.model\_selection **import** validation\_curve, learning\_curve  
**import** matplotlib.pyplot **as** plt  
  
**from** learn.dataset **import** load\_dataset  
  
  
**def** plot\_validation\_curve(data\_bunch, name):  
 param\_range = np.linspace(1, 200, 50, dtype=np.int) *# 50* train\_scores, test\_scores = validation\_curve(  
 RandomForestClassifier(), data\_bunch.data, data\_bunch.target,  
 param\_name=**"n\_estimators"**, param\_range=param\_range, n\_jobs=3  
 )  
 train\_scores\_mean = np.mean(train\_scores, axis=1)  
 train\_scores\_std = np.std(train\_scores, axis=1)  
 test\_scores\_mean = np.mean(test\_scores, axis=1)  
 test\_scores\_std = np.std(test\_scores, axis=1)  
  
 plt.title(**"Validation Curve with RF: %s"** % name)  
 plt.xlabel(**"n\_estimators"**)  
 plt.ylabel(**"Score"**)  
 plt.ylim(0.0, 1.1)  
  
 plt.semilogx(param\_range, train\_scores\_mean, label=**"Training score"**, color=**"r"**)  
 plt.fill\_between(param\_range, train\_scores\_mean - train\_scores\_std,  
 train\_scores\_mean + train\_scores\_std, alpha=0.2, color=**"r"**)  
  
 plt.semilogx(param\_range, test\_scores\_mean, label=**"Cross-validation score"**,  
 color=**"g"**)  
 plt.fill\_between(param\_range, test\_scores\_mean - test\_scores\_std,  
 test\_scores\_mean + test\_scores\_std, alpha=0.2, color=**"g"**)  
  
 plt.legend(loc=**"best"**)  
 plt.show()  
  
  
**def** plot\_roc(data\_bunch, name):  
 X = data\_bunch.data  
 y = data\_bunch.target  
  
 cv = StratifiedKFold(y, n\_folds=6)  
  
 clf = RandomForestClassifier(n\_estimators=50)  
  
 mean\_tpr = 0  
 mean\_fpr = np.linspace(0, 1, 100)  
 **for** i, (train, test) **in** enumerate(cv):  
 probas\_ = clf.fit(X[train], y[train]).predict\_proba(X[test])  
  
 *# Compute ROC curve and area the curve* fpr, tpr, thresholds = roc\_curve(y[test], probas\_[:, 1])  
 mean\_tpr += scipy.interp(mean\_fpr, fpr, tpr)  
 mean\_tpr[0] = 0.0  
 roc\_auc = auc(fpr, tpr)  
 plt.plot(fpr, tpr, lw=1, label=**'ROC fold %d (area = %0.2f)'** % (i, roc\_auc))  
  
 plt.plot([0, 1], [0, 1], **'--'**, color=(0.6, 0.6, 0.6), label=**'Luck'**)  
 mean\_tpr /= len(cv)  
 mean\_tpr[-1] = 1.0  
 mean\_auc = auc(mean\_fpr, mean\_tpr)  
 plt.plot(mean\_fpr, mean\_tpr, **'k--'**,  
 label=**'Mean ROC (area = %0.2f)'** % mean\_auc, lw=2)  
  
 plt.xlim([-0.05, 1.05])  
 plt.ylim([-0.05, 1.05])  
 plt.xlabel(**'False Positive Rate'**)  
 plt.ylabel(**'True Positive Rate'**)  
 plt.title(**'Receiver operating characteristic: %s'** % name)  
 plt.legend(loc=**"lower right"**)  
 plt.show()  
  
  
**def** plot\_learning\_curve(data\_bunch, name):  
 train\_sizes = np.linspace(.1, 1.0, 10)  
  
 X = data\_bunch.data  
 y = data\_bunch.target  
  
 estimator = RandomForestClassifier(n\_estimators=50)  
  
 cv = ShuffleSplit(X.shape[0], n\_iter=100,  
 test\_size=0.2, random\_state=0)  
  
 plt.figure()  
 plt.title(**"Learning Curve: %s"** % name)  
 plt.ylim(0.5, 1.1)  
 plt.xlabel(**"Training examples"**)  
 plt.ylabel(**"Score"**)  
 train\_sizes, train\_scores, test\_scores = learning\_curve(  
 estimator, X, y, cv=cv, n\_jobs=3, train\_sizes=train\_sizes)  
 train\_scores\_mean = np.mean(train\_scores, axis=1)  
 train\_scores\_std = np.std(train\_scores, axis=1)  
 test\_scores\_mean = np.mean(test\_scores, axis=1)  
 test\_scores\_std = np.std(test\_scores, axis=1)  
 plt.grid()  
 plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,  
 train\_scores\_mean + train\_scores\_std, alpha=0.1,  
 color=**"r"**)  
 plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,  
 test\_scores\_mean + test\_scores\_std, alpha=0.1, color=**"g"**)  
 plt.plot(train\_sizes, train\_scores\_mean, **'o-'**, color=**"r"**,  
 label=**"Training score"**)  
 plt.plot(train\_sizes, test\_scores\_mean, **'o-'**, color=**"g"**,  
 label=**"Cross-validation score"**)  
 plt.legend(loc=**"best"**)  
 plt.show()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = load\_dataset(sys.argv[1], union\_age\_classes=[**'s'**, **'a'**])  
 data.print\_info()  
  
 *# 7 40 40* plot\_validation\_curve(data.gender, **"gender"**)  
 plot\_validation\_curve(data.male, **"male"**)  
 plot\_validation\_curve(data.female, **"female"**)  
  
 plot\_roc(data.gender, **"gender"**)  
 plot\_roc(data.male, **"male"**)  
 plot\_roc(data.female, **"female"**)  
  
 plot\_learning\_curve(data.gender, **"gender"**)  
 plot\_learning\_curve(data.male, **"male"**)  
 plot\_learning\_curve(data.female, **"female"**)