Список литературы

[1] Сорокин В. Н., Вьюгин В. В., Тананыкин А. А. Распознавание личности по голосу: аналитический обзор //Информационные процессы. – 2012. – Т. 12. – №. 1. – С. 1-30.

[2] Kinnunen T., Li H. An overview of text-independent speaker recognition: From features to supervectors //Speech communication. – 2010. – Т. 52. – №. 1. – С. 12-40.

[3] Федоров Е. Е. Моделирование особенностей речи диктора //Математические машины и системы. – 2008. – Т. 1. – №. 1.

[4] Martinez J. et al. Speaker recognition using Mel frequency Cepstral Coefficients (MFCC) and Vector quantization (VQ) techniques //Electrical Communications and Computers (CONIELECOMP), 2012 22nd International Conference on. – IEEE, 2012. – С. 248-251.

[5] Mariéthoz J., Bengio S., Grandvalet Y. Kernel Based Text-Independnent Speaker Verification. – Idiap, 2008. – №. LIDIAP-REPORT-2008-013.

[6] Pekcan O. Development of machine learning based speaker recognition system. – 2010.

[7] Kamruzzaman S. M. et al. Speaker identification using mfcc-domain support vector machine //arXiv preprint arXiv:1009.4972. – 2010.

[8] Senin P. Dynamic time warping algorithm review //Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA. – 2008. – С. 1-23.

[9] Practical Cryptography, “Mel Frequency Cepstral Coefficient (MFCC) Tutorial” [Электронный ресурс]. URL: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ (дата обращения: май 2015).

[10] Sonkamble B. A., Doye D. D. Speech Recognition Using Vector Quantization through Modified K-meansLBG Algorithm //Computer Engineering and Intelligent Systems. – 2012. – Т. 3. – №. 7. – С. 137-144.

[11] Разинков Е.В., Машинное обучение, КСАИТ, ИВМиИТ КФУ

[12] Criminisi A., Shotton J. Decision forests for computer vision and medical image analysis. – Springer Science & Business Media, 2013.

[13] Сигналов О. А., Моделирование К. Определение пола диктора по голосу. – 2008.

[14] Любимов Н. А. Метод разреженных представлений в задаче автоматической тексто-независимой идентификации и верификации диктора.

Приложение. Листинг программы

main.py

```
# -*- coding: utf-8 -*-
import pylab
import datetime
import sklearn
from sklearn import metrics,ensemble,cluster

import os
import recognition as rec
os.system('cls')

startTime = datetime.datetime.now()
print startTime

CONST_setN=150
CONST_esimators=100
CONST_depth=25
CONST_learner='RandomForestClassifier'
CONST_supvec='Kmeans-2, mfcc = 26'


#Parsing,getting paths, making list
speakerInfoFile=open('speaker-info.txt')
speakerList=rec.parse(speakerInfoFile)
print 'Parsing done',str(datetime.datetime.now()-startTime)

rightTrainList=rec.makeSublist(speakerList,0,CONST_setN)
rightTestList=rec.makeSublist(speakerList,0,10)
print 'RightSpeakerListsdone',str(datetime.datetime.now()-startTime)

directory = 'wav48'
audiofiles=rec.getFiles(directory)
print "Getting files'paths done",str(datetime.datetime.now()-startTime)

trainFiles=rec.createFilesList(audiofiles,0,CONST_setN)
testFiles=rec.createFilesList(audiofiles,CONST_setN,CONST_setN+10)
print 'Files list done',str(datetime.datetime.now()-startTime)

######################################################################
filename='results.txt'
txt = open(filename,'a')

#Supervectors, RandomForestClassifier
trainSupVec=rec.getMfcc(trainFiles)
```

```
      testSupVec=rec.getMfcc(testFiles)
      print 'Make supervectors done',str(datetime.datetime.now()-startTime)

      clf = ensemble.RandomForestClassifier(n_estimators=CONST_esimators,
                                            max_depth=CONST_depth)
      clf = clf.fit(trainSupVec, rightTrainList)
      print 'RandomForestClassifier
done',str(datetime.datetime.now()-startTime)



      #Prediction
      predictTest=clf.predict(testSupVec)
      predictTrain=clf.predict(trainSupVec)
      print 'Prediction done',str(datetime.datetime.now()-startTime)


      #Estimation
      resultTest=rec.intersec(predictTest,rightTestList)
      resultTrain=rec.intersec(predictTrain,rightTrainList)
      errlst=[]
      for x in xrange(len(predictTest)):
       if predictTest[x]!=rightTestList[x]:
            errlst.append((rightTestList[x],predictTest[x]))

      lenTrain_all,lenTrain_res,perTrain=rec.estimate(rightTrainList,
resultTrain)
      lenTest_all,lenTest_res,perTest=rec.estimate(rightTestList, resultTest)



      #Results output
      now=str(datetime.datetime.now().date())+" "+\
            str(datetime.datetime.now().time())
      cond="set = "+str(CONST_setN)+\
          ", trees = "+str(CONST_esimators)+\
          ", max_depth = "+str(CONST_depth)

      resTrain="Train set: size = "+str(lenTrain_all)+\
         ", rigth = "+str(lenTrain_res)+", "+str(perTrain)

      resTest="Test set: size = "+str(lenTest_all)+\
             ", rigth = "+str(lenTest_res)+", "+str(perTest)



      txt.write(now+"\n"+
            CONST_learner+"\n"+
            CONST_supvec+"\n"+
```

```
            cond+"\n"+
            resTrain+"\n"+
            resTest+"\n\n\n")

      txt.close()


      print 'time elapsed =', (datetime.datetime.now() - startTime)
      print '\a'
```

recognition.py

```
      # -*- coding: utf-8 -*-
      import os
      import sklearn
      from sklearn.cluster import KMeans,MiniBatchKMeans
      from features import mfcc
      from features import logfbank
      import scipy.io.wavfile as wav
      from sklearn import mixture
      import numpy as np

      def parse(inputFile):
            """
            Parsing input file into dictionary
            :param inputFile: lines like "225  23  F   English     Southern
England"
            :return outputDict:list of No. of speakers
            temp(dictionary like {id:(age, gender, accents, region)}
            Sometimes there is no region)
            """
            indexes=[]
            genders=[]
            ages=[]
            accents=[]
            regions=[]

            for inputString in inputFile:
            text=''
            symbol = 0
            #Indexes
            while symbol < 3 and inputString[symbol]!=' ' and
inputString[symbol]!='\n':
                  text+=inputString[symbol]
                  symbol+=1
            indexes.append(int(text))

            #Ages
            symbol += 2
```

```
            text = inputString[symbol]
            symbol += 1
            text +=inputString[symbol]
            ages.append(int(text))

            #Genders
            symbol += 3
            text = inputString[symbol]
            genders.append(text)

            #Accents
            symbol +=5
            text=''
            while inputString[symbol]!=' ' and inputString[symbol]!='\n':
                  text+=inputString[symbol]
                  symbol+=1
            accents.append(text)

            #Region
            text=''
            if(inputString[symbol]!='\n'):
                  while inputString[symbol]==' ':
                  symbol+=1
                  while (inputString[symbol]!='\n'):
                  text+=inputString[symbol]
                  symbol+=1
            regions.append(text)
            #zipped_list = zip(ages, genders, accents, regions)
            #liswit = zip(indexes, zipped_list)
            #outputDict = dict(liswit)

            return indexes

      def getFiles(directory):
            """
            Collecting paths to .wav files into one list
            :param directory: path to direcrory
            :return audiofiles: list with paths "wav48/<speaker No>/<speaker
No>_<sentence No>
            """
            audiofiles = []
            lenghs=[]
            for d, dirs, files in os.walk(directory):
            files1=[]
            for onefile in files:
                  files1.append(d+"/"+onefile)
            audiofiles.append(files1)
            audiofiles.pop(0)
            return audiofiles
```

```
def createFilesList(audiofiles, startingFrom, stopAt):
      """
      Divide list into parts
      :param audiofiles: list of paths
      :return trainSet: list of paths
      Sometimes there is no region
      """
      trainSet = []
      for files in audiofiles:
      subSet=[]
      for index in xrange(startingFrom, stopAt):
            subSet.append(files[index])
      trainSet.append(subSet)
      return trainSet

def makeSuperVecAver(path_to_onefile):
      """
      Making supervector using average values
      :param path_to_onefile: path to file
      :return aver_vec: list of numbers-supervector of mfcc (average)
      """
      aver_vec=[]
      (rate,sig)=wav.read(path_to_onefile)
      mfcc_feat=mfcc(sig,rate,numcep=13)
      aver_elem=[0 for x in xrange(len(mfcc_feat[0]))]
      for vec in mfcc_feat:
      aver_elem+=vec
      aver_vec=aver_elem/len(mfcc_feat)
      return aver_vec


def makeSuperVecKMean(path_to_onefile):
      """
      Making supervector using average values
      :param path_to_onefile: path to file
      :return aver_vec: list of numbers-supervector of mfcc (average)
      """
      clf = KMeans(n_clusters=2)
      (rate,sig)=wav.read(path_to_onefile)
      mfcc_feat=mfcc(sig,rate,numcep=26)
      clf.fit(mfcc_feat)
      aver_vec = listmerge(clf.cluster_centers_)
      return aver_vec

def getMfcc(filesArray):
      """
      Making list of supervector
      :param filesArray: array of paths to files
```

```
            :return aver_array: list of numbers-supervector of mfcc
(average)
            """
            aver_array=[]
            for vec in filesArray:
            for onefile in vec:
                  aver_vec=makeSuperVecKMean(onefile)
                  aver_array.append(aver_vec)
            return aver_array

      def intersec(predict, right):
            result=[]
            for i in xrange(len(predict)):
            if(predict[i]==right[i]):
                  result.append(predict[i])
            return result

      def makeSublist(biglst, startingFrom, stopAt):
            reslst=[]
            for speaker in biglst:
            for i in xrange(startingFrom, stopAt):
                  reslst.append(speaker)
            return reslst

      def estimate(right, res):
            len_all=len(right)
            len_res=len(res)
            per=(float(len_res)/len_all)*100
            return len_all,len_res,per

      def listmerge(lstlst):
            all=[]
            for lst in lstlst:
            all.extend(lst)
            return all
```