ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
КАФЕДРА ЕОМ

МЕТОДИ ГОЛОСОВОЇ ІДЕНТИФІКАЦІЇ В КОМП'ЮТЕРНИХ СИСТЕМАХ

КВАЛІФІКАЦІЙНА РОБОТА
ДРУГИЙ (МАГІСТЕРСЬКИЙ) РІВЕНЬ

Автор:
Коваль Д.І.
ст. гр. СПм-20-1

Керівник:
Іващенко Г.С.
доц. каф. ЕОМ

АКТУАЛЬНІСТЬ ТЕМИ

Актуальною технічною задачею в галузі телекомунікаційних систем, інтернет-технологій тощо, є забезпечення функцій контролю доступу, що полягають у формуванні дозволу до певних даних або його забороні. Такий контроль ґрунтується на ідентифікації суб'єктів, яким потрібен доступ, і об'єкта даних, що є метою доступу.

В галузі інформаційної безпеки під ідентифікацією розуміється процедура розпізнавання користувача в системі шляхом сприйняття системою ідентифікаторів користувача, які формуються на основі апріорної інформації про нього.

МЕТОДИ БІОМЕТРИЧНОЇ ІДЕНТИФІКАЦІЇ

Методи біометричної ідентифікації можна поділити на дві великі групи:

- статичні методи, які ґрунтуються на фізіологічних характеристиках людини;
- динамічні методи, які ґрунтуються на особливостях поведінки людини.

Серед статичних методів можна виділити, розпізнання за формою кисті руки, розпізнавання за відбитком пальців, розпізнавання за сітківкою ока, розпізнавання за портретом.

Серед динамічних методів біометричної ідентифікації розрізняють такі, як ідентифікація особи за особливостями голосу, ідентифікація за почерком миші, ідентифікація за клавіатурним почерком.

3

МЕТА РОБОТИ

Метою кваліфікаційної роботи є дослідження методів ідентифікації людини за голосом.

Огляд існуючих досліджень по біометричній ідентифікації.

Розробка тестового середовища, яке дозволятиме дослідити можливості голосової ідентифікації та особливості існуючих методів.

Аналіз отриманих результатів дослідження.

4

ВИДІЛЕННЯ ХАРАКТЕРНИХ ОЗНАК ГОЛОСУ

При постановці завдання ідентифікації особистості по її біометричним даними, одним з головних етапів є виділення цінної інформації з вхідних сигналів.

У розпізнаванні диктора важливо підкреслити ознаки, що відповідають за індивідуальність висловлювань. Для розпізнання особистості по голосу використовуються два основних алгоритми: це MFCC (Мелчастотні кепстральні коефіцієнти) і LPCC (Коефіцієнти лінійного передбачення).

5

МЕЛ-ЧАСТОТНІ КЕПСТРАЛЬНІ КОЕФІЦІЄНТИ

- вхідний сигнал розбивається на фрейми, щоб вони перекривали наступні і попередні за ними. Довжина фреймів безпосередньо впливає на роботу алгоритму.
- для кожного фрейма обчислюється його спектр за допомогою дискретного перетворення Фур'є.
- отримані спектральні коефіцієнти фреймів накладаються на мелчастотні вікна.
- застосовується дискретне косинусне перетворення, яке дає на виході багатовимірний вектор ознак сигналу. Вони і є мелчастотними кепстральними коефіцієнтами.

6

КОЕФІЦІЄНТИ ЛІНІЙНОГО ПЕРЕДБАЧЕННЯ

- обчислення коефіцієнтів авторегресивної моделі для кожного фрейму.
- обчислюються кепстральні LPCC-коефіцієнти по рекурсивній функції.

LPCC використовує лінійну шкалу переведення частоти звуку в залежності від його висоти. Цей спосіб добре працює в області низьких частот, так як в цій зоні залежність висоти звуку від його частоти практично лінійна.

7

МЕТОДИ ІДЕНТИФІКАЦІЇ

Основні етапи, характерні для кожного з методів:

- отримання ознак із вхідного мовного сигналу;
- побудова шаблону диктора на основі отриманих на попередньому етапі векторів ознак.

Поширені методи:

- векторне квантування (Vector Quantization);
- метод опорних векторів (Support-vector Machine);
- алгоритм динамічної трансформації часової шкали (Dynamic Time Warping);
- прихована марковська модель (Hidden Markov Model);
- модель суміші гауса (Gaussian Mixture Model);
- штучні нейронні мережі.

8

ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

- Python.

Простота та швидкість написання коду. Часто використовується для роботи з нейронними мережами.

- TensorFlow.

Великий обсяг документації та посібників. Забезпечує обслуговування моделей та підтримує розподілене навчання.

- Visual Studio Code.

У редакторі присутні інструменти для роботи з Git та засоби рефакторингу, навігації за кодом.

9

АЛГОРИТМ ПРОЦЕСУ ІДЕНТИФІКАЦІЇ

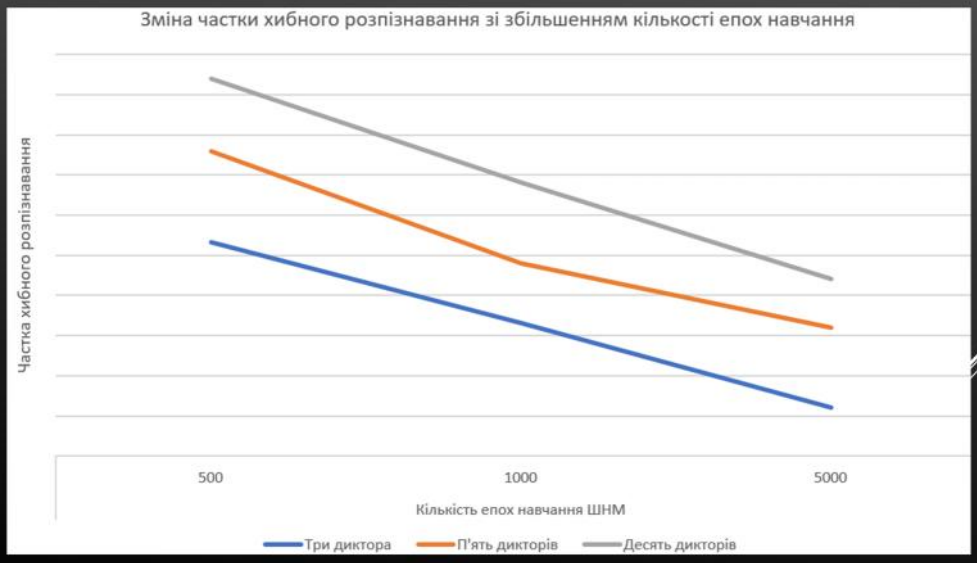
10

ЗАЛЕЖНІСТЬ ТОЧНОСТІ РОЗПІЗНАВАННЯ ВІД КІЛЬКОСТІ ЕПОХ НАВЧАННЯ

		Кількість епох навчання ШНМ		
		500	1000	5000
Частка хибного розпізнавання	Три диктора	26.6%	16.6%	6%
	П'ять дикторів	38%	24%	16%
	Десять дикторів	47%	34%	22%

11

ЗАЛЕЖНІСТЬ ТОЧНОСТІ РОЗПІЗНАВАННЯ ВІД КІЛЬКОСТІ ЕПОХ НАВЧАННЯ



12

ВИСНОВКИ

В ході виконання роботи проведений аналіз методів ідентифікації дикторів. Досліджені існуючі підходи щодо вирішення задачі ідентифікації користувача по голосу.

Мел-частотні кепстральні коефіцієнти не пов'язані зі слів, які вимовляє диктор, вони використовують «значущі» для людського вуха частоти, які характеризують людський голос.

При тексто-незалежній ідентифікації дикторів точність розпізнавання зменшується. Пов'язано це з тим, що для такої ідентифікації необхідно більше епох навчання нейронної мережі, ніж для тексто-залежної з таким же набором дикторів.

При збільшенні кількості дикторів точність ідентифікації зменшується, а при збільшенні кількості епох навчання зростає.

13

ДОДАТОК Б

Формат WAV файлу

Позиція	Розмір	Назва	Пояснення
RIFF Заголовок			
0	4 байти	ChunkID	Містить літери "RIFF" в ASCII кодуванні (0x52494646 у порядку байт big-endian)
4	4 байти	ChunkSize	36 + SubChunk2Size, або більш точно: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) Це розмір всього файлу в байтах, не враховуючи 8 байтів перших двох полів: ChunkID і ChunkSize.
8	4 байти	Format	Містить літери "WAVE" (0x57415645 у порядку байт big-endian).
Підсекція «fmt»			
12	4 байти	Subchunk1ID	Містить літери «fmt» (0x666d7420 у порядку байт big-endian).
16	4 байти	Subchunk1Size	Приймає значення 16 для формату кодування PCM. Це розмір області цієї підсекції, що слідує після цього поля.
20	2 байта	AudioFormat	PCM = 1 (Лінійне квантування) Значення відмнінні від 1 вказують на наявність конкретного кодування аудіо даних.
22	2 байта	NumChannels	Кількість звукових каналів. Моно = 1, Стерео = 2, і т.д.

24	4 байти	SampleRate	Частота дискретизації у Герцах. Наприклад, 8000, 44100, інші.
28	4 байти	ByteRate	= SampleRate * NumChannels * BitsPerSample/8
32	2 байта	BlockAlign	= NumChannels * BitsPerSample/8 Кількість байт, яка міститься в одному семплі враховуючи кількість каналів.
34	2 байта	BitsPerSample	Кількість біт в одному семплі. Так звана "глибина" чи точність звучання. 8, 16, 32, і так далі.
	2 байта	ExtraParamSize	Розмір поля з параметрами. Якщо PCM, ці два поля ExtraParamSize і ExtraParams не записуються.
	X	ExtraParams	Місце для запису додаткових параметрів
Підсекція «data»			
36	4 байти	Subchunk2ID	Містить літери «data» (0x64617461 у порядку байт big-endian).
40	4 байти	Subchunk2Size	Розмір даних звукозапису в байтах, які містяться в наступному полі. Кількість семплів * NumChannels * BitsPerSample/8
44	X	Data	Фактичні дані звукозапису.

ДОДАТОК В

Фрагменти вихідного коду

```
def wavefile_to_waveform(wav_file, features_type):
    data, sr = sf.read(wav_file)
    if features_type == 'vggish':
        tmp_name = str(int(np.random.rand(1)*1000000)) + '.wav'
        sf.write(tmp_name, data, sr, subtype='PCM_16')
        sr, wav_data = wavfile.read(tmp_name)
        os.remove(tmp_name)
        assert wav_data.dtype == np.int16, 'Bad sample type: %r'
% wav_data.dtype
        data = wav_data / 32768.0
        src_repeat = data
    while (src_repeat.shape[0] < sr):
        src_repeat = np.concatenate((src_repeat, data), axis=0)
        data = src_repeat[:sr]
    return data, sr

def binm(rr,ra):
  ih=len(ra)-1
  il=0
  if rr<ra[il]: return il
  while (ih-il>1):
    ie=(ih+il)/2
    if rr<ra[ie]:
      ih=ie
    else:
      il=ie
  return ih

bins = [20,30,40]
results = [0,0,0,0]

for _ in range(iterations):
  x = somefunction()
  ib=binm(x,bins)
  results[ib]+=1

from pydub import AudioSegment
import math
class SplitWavAudioMubin():
    def __init__(self, folder, filename):
        self.folder = folder
        self.filename = filename
        self.filepath = folder + '\\' + filename

        self.audio = AudioSegment.from_wav(self.filepath)

    def get_duration(self):
        return self.audio.duration_seconds
```

```
    def single_split(self, from_min, to_min, split_filename):
        t1 = from_min * 60 * 1000
        t2 = to_min * 60 * 1000
        split_audio = self.audio[t1:t2]
        split_audio.export(self.folder + '\\' + split_filename,
format="wav")
    def multiple_split(self, min_per_split):
        total_mins = math.ceil(self.get_duration() / 60)
        for i in range(0, total_mins, min_per_split):
            split_fn = str(i) + '_' + self.filename
            self.single_split(i, i+min_per_split, split_fn)
            print(str(i) + ' Done')
            if i == total_mins - min_per_split:
                print('All splited successfully')


from pydub import AudioSegment
from pydub.silence import split_on_silence

sound_file = AudioSegment.from_wav("a-z.wav")
audio_chunks = split_on_silence(sound_file,
    # must be silent for at least half a second
    min_silence_len=500,

    # consider it silent if quieter than -16 dBFS
    silence_thresh=-16
)

for i, chunk in enumerate(audio_chunks):

    out_file = ".//splitAudio//chunk{0}.wav".format(i)
    print "exporting", out_file
    chunk.export(out_file, format="wav")


def entropy1(labels, base=None):
  value,counts = np.unique(labels, return_counts=True)
  return entropy(counts, base=base)
def entropy2(labels, base=None):
  n_labels = len(labels)
  if n_labels <= 1:
    return 0
  value,counts = np.unique(labels, return_counts=True)
  probs = counts / n_labels
  n_classes = np.count_nonzero(probs)

  if n_classes <= 1:
    return 0

  ent = 0.
  base = e if base is None else base
  for i in probs:
```

```
    ent -= i * log(i, base)
  return ent

def entropy3(labels, base=None):
  vc = pd.Series(labels).value_counts(normalize=True,
sort=False)
  base = e if base is None else base
  return -(vc * np.log(vc)/np.log(base)).sum()

def entropy4(labels, base=None):
  value,counts = np.unique(labels, return_counts=True)
  norm_counts = counts / counts.sum()
  base = e if base is None else base
  return -(norm_counts * np.log(norm_counts)/np.log(base)).sum()


def extract_features(file_name):    audio, sample_rate =
librosa.load(file_name, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=40)
    mfccs_processed = np.mean(mfccs.T,axis=0)

    return mfccs_processed
features = []
for index, row in metadata.iterrows():

    file_name =
os.path.join(os.path.abspath(fulldatasetpath),'fold'+str(row["fo
ld"])+'/',str(row["slice_file_name"]))

    class_label = row["class"]
    data = extract_features(file_name)
    features.append([data, class_label])
featuresdf = pd.DataFrame(features, columns=['feature', 'fold'
,'class_label'])
featuresdf.head()


def _mfcc_and_labels(audio, labels):
  mfcc_sample_rate = 100.0
  winfunc = lambda x: np.hamming(x)
  mfcc_features = python_speech_features.mfcc(audio,
samplerate=timit.SAMPLE_RATE, winlen=0.025,

winstep=1.0/mfcc_sample_rate, lowfreq=85.0,

highfreq=timit.SAMPLE_RATE/2, winfunc=winfunc)
  t_audio = np.linspace(0.0, audio.shape[0] * 1.0 /
timit.SAMPLE_RATE, audio.size, endpoint=False)
  t_mfcc = np.linspace(0.0, mfcc_features.shape[0] * 1.0 /
mfcc_sample_rate, mfcc_features.shape[0], endpoint=False)
  interp_func = scipy.interpolate.interp1d(t_audio, labels,
kind='nearest')
```

```
  mfcc_labels = interp_func(t_mfcc)
  return mfcc_features, mfcc_labels


low_freq_mel = 0
high_freq_mel = (2595 * numpy.log10(1 + (sample_rate / 2) /
700))  # Convert Hz to Mel
mel_points = numpy.linspace(low_freq_mel, high_freq_mel, nfilt +
2)  # Equally spaced in Mel scale
hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel
to Hz
bin = numpy.floor((NFFT + 1) * hz_points / sample_rate)

fbank = numpy.zeros((nfilt, int(numpy.floor(NFFT / 2 + 1))))
for m in range(1, nfilt + 1):
    f_m_minus = int(bin[m - 1])   # left
    f_m = int(bin[m])             # center
    f_m_plus = int(bin[m + 1])    # right

    for k in range(f_m_minus, f_m):
        fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m -
1])
    for k in range(f_m, f_m_plus):
        fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] -
bin[m])
filter_banks = numpy.dot(pow_frames, fbank.T)
filter_banks = numpy.where(filter_banks == 0,
numpy.finfo(float).eps, filter_banks)  # Numerical Stability
filter_banks = 20 * numpy.log10(filter_banks)

(nframes, ncoeff) = mfcc.shape
n = numpy.arange(ncoeff)
lift = 1 + (cep_lifter / 2) * numpy.sin(numpy.pi * n /
cep_lifter)
mfcc *= lift


def amplitudes_at_frequencies(freqInds, timeseries, times=None,
transform='dct'):
    amplitudes = {}
    for o in timeseries.keys():

        if transform == 'dct':
            temp = _dct(timeseries[o], norm='ortho')[freqInds] /
_np.sqrt(len(timeseries[o]) / 2)
            if 0. in freqInds:
                temp[0] = temp[0] / _np.sqrt(2)
            amplitudes[o] = list(temp)

        else:
            raise NotImplementedError("This function only
currently works for the DCT!")
```

```
    return amplitudes

data = pd.read_csv('dataset.csv')
data.head()# Dropping unneccesary columns
data = data.drop(['filename'],axis=1)#Encoding the Labels
genre_list = data.iloc[:, -1]
encoder = LabelEncoder()
y = encoder.fit_transform(genre_list)#Scaling the Feature
columns
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype =
float))#Dividing data into training and Testing set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)


model = Sequential()
model.add(layers.Dense(256, activation='relu',
input_shape=(X_train.shape[1],)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,
activation='softmax'))model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


classifier = model.fit(X_train,
                   y_train,
                   epochs=100,
                   batch_size=128)


for i in range(8):
    result = round()
    if result > x:
        x = result
if x == 5:
return True


def viterbi(y, A, B, Pi=None):
    K = A.shape[0]
    Pi = Pi if Pi is not None else np.full(K, 1 / K)
    T = len(y)
    T1 = np.empty((K, T), 'd')
    T2 = np.empty((K, T), 'B')
    T1[:, 0] = Pi * B[:, y[0]]
    T2[:, 0] = 0
    for i in range(1, T):
        T1[:, i] = np.max(T1[:, i - 1] * A.T * B[np.newaxis, :,
y[i]].T, 1)
        T2[:, i] = np.argmax(T1[:, i - 1] * A.T, 1)
```

```
    x = np.empty(T, 'B')
    x[-1] = np.argmax(T1[:, T - 1])
    for i in reversed(range(1, T)):
        x[i - 1] = T2[x[i], i]
    return x, T1, T2


def BaumWelch(T, S, H, A, B, pi, X):
    alpha = cal_alpha(T, S, H, A, B, pi)
    beta = cal_beta(T, S, H, A, B, pi)
    gamma = cal_gamma(T, S, H, A, B, pi, alpha, beta)
    xi = cal_xi(T, S, H, A, B, pi, alpha, beta)
    N = len(H)
    M = len(S)
    for i in range(N):
        pi[i] = gamma[0][i]
        for j in range(N):
            a = 0
            b = 0
            for t in range(T-1):
                a += xi[t][i][j]
                b += gamma[t][i]
            A[i][j] = a / b
    for j in range(N):
        for k in range(M):
            c = 0
            d = 0
            for t in range(T):
                if X[t] == S[k]: c += gamma[t][j]
                d += gamma[t][j]
            B[j][k] = c / d
```