**Класс Recognizer**

***Описание констант***

|  |  |
| --- | --- |
| **Константа** | **Описание** |
| DTW\_AVERAGE | Усреднить новый эталон по DTW с уже существующим в базе (не используется в данной версии класса) |
| DTW\_REPLACE | Заменить старый эталон в базе новым(в данной версии не используется) |
| DTW\_INSERT | Добавить новый эталон в базу( используется в функции addEtalon) |
| UP | в данной версии не используется |
| RIGHT | в данной версии не используется |
| UPRIGHT | в данной версии не используется |

***Структура EtalonStruct –*** запись с данными об эталоне в базе данных.

***TemplateData templ –*** объект содержащий данные овекторах признаков, составляющие эталон.

***string name –*** строковое название эталона.

***int numOfMean –*** количество усреднений для данного эталона (в данной версии не используется).

***Описание функций***

***Функция setWord(unsigned int Num, string sWord)*** – изменяет имя эталона с номером *Num* в базе эталонов, на имя, заданное параметром *sWord*.

***Функция void addEtalon(EtalonStruct newEt,unsigned int Num,int iMode=DTW\_AVERAGE)*** – добавление нового эталона *newEt* в базу эталонов. Если *iMode =* DTW\_INSERT то новый эталон добавляется к базе, иначе новый эталон заменяет эталон с номером *Num.*

***Функция string getWord(unsigned int i)*** – возвращает в строке название эталона, заданное параметром *i*.

***Функция unsigned int getEtalonCount() –*** возвращает количество эталонов в базе.

***Функция*** ***void saveEtalons(string path)*** – сохраняет базу эталонов на диске в файл, заданный параметром *path.*

***Функция void loadEtalons(string path)*** – загружает базу эталонов из файла, заданного параметром *path.*

***Функция*** ***void delEtalon(unsigned int Num)*** – удаляет из базы эталон с номером *Num.*

***Функция void delAllEtalons() –*** удаляет все эталоны из базы.

***Функция*** ***int getN(unsigned int Num)*** – количество усреднений для эталона с номером *Num*.

***Функция void getEtalon(EtalonStruct\* &v, unsigned int Num)*** – возвращает данные об эталоне с номером *Num* из базы данных в структуре *v*.

***Функция*** ***float dist2Vectors(float x[SV\_SIZE\_VECTOR],float y[SV\_SIZE\_VECTOR]) –*** вычисляет расстояние между двумя векторами признаков *x* и *y.*  Расстояние вычисляется как корень из суммы квадратов разностей соответствующих координат: 

***Функция*** ***float distDTW(TemplateData E, TemplateData A)* –**вычисление DTW-расстояния между двумя эталонами *E* и *A*. Вычисляется матрица C размера n на m, где n – количество векторов распознаваемого образца речи и m – количество векторов эталона. Элементы матрицы определяются как

,,,

C = new float\*[J\_max];

for(j = 0;j < J\_max;j++)

C[j] = new float[I\_max];

C[0][0] = 2 \* dist2Vectors(E.m\_vec[0].V,A.m\_vec[0].V);

for(i = 1; i < I\_max;i++)

C[0][i] = C[0][i-1] + dist2Vectors(A.m\_vec[0].V,E.m\_vec[i].V);

for(j = 1;j < J\_max;j++)

C[j][0] = C[j-1][0] + dist2Vectors(A.m\_vec[j].V,E.m\_vec[0].V);

for(j = 1;j < J\_max;j++)

{

for(i = 1;i < I\_max;i++)

{

d = dist2Vectors(A.m\_vec[j].V,E.m\_vec[i].V);

g1 = C[j-1][i] + d;

g2 = C[j-1][i-1] + 2\*d;

g3 = C[j][i-1] + d;

C[j][i] = MIN(g1,g2,g3);

}

}

Расстояние между эталонами определяется как:



res = C[J\_max - 1][I\_max - 1];

float Norm = J\_max + I\_max;

res /= Norm;

***Функция*** ***recognize(TemplateData RecVec, string &res)*** – вычисляет DTW-расстояния между набором векторов признаков исследуемого сигнала *RecVec* и всеми эталонами базы и определяет эталон с наименьшим расстоянием. Строковое имя эталона возвращает в параметре *res*.

**Класс TemplateData**

***Описание констант***

|  |  |
| --- | --- |
| **Константа** | **Описание** |
| SV\_FEAT\_MFCC\_1 | Указывает тип вектора признаков - мел-кепстральные коэффициенты (используется в функции createVectors) |
| SV\_SIZE\_WINDOW | Длина окна анализа сигнала в отсчетах |
| SV\_SIZE\_VECTOR | Размер вектора признаков |
| M\_PI | Число π |

***Структура FeatureVector –*** описывает вектор признаков.

***float V[SV\_SIZE\_VECTOR] –*** вектор признаков размера SV\_SIZE\_VECTOR***.***

***Описание функций***

***Функция void fft(float \*fftBuffer, long fftFrameSize, long sign)*** *–* вычисляет коэффициенты быстрого преобразования Фурье для фрагмента сигнала в *fftBuffer* длиной *fftFrameSize* отсчетов. Параметр *sign* определяет направление преобразования Фурье: -1 – прямое, 1 – обратное. Коэффициенты преобразования возвращаются в параметре *fftBuffer.*

***Функция*** ***void transformFFTReIm(float \*dSignal,float \*fft\_re,float \*fft\_im, int fftFrameSize)*** – применяет весовую функцию и затем прямое быстрое преобразование Фурье для фрагмента сигнала в *dSignal* длиной *fftFrameSize* отсчетов. В параметре *fft\_re* функция возвращает вещественные части коэффициентов, в параметре *fft\_im*– комплексные. В качестве весовой функции (для уменьшения искажений в Фурье-анализе, вызванных конечностью выборки) используется окно Хэмминга, которое имеет следующий вид:



где *N* – длина окна, выраженная в отсчетах.

***Функция*** ***void MFCC1(float \*fft\_re,float \*fft\_im,int N,float \*C1, int NumBand) –***

вычисляет вектор признаков мел-кепстральные коэффициенты по коэффициентам БПФ, заданным в параметрах *fft\_re, fft\_im.* Количество коэффициентов определяется параметром *NumBand.* В параметре *C1* функция возвращает вектор рассчитанных коэффициентов.

***Функция*** ***void createVectors(unsigned char \* str,int iSize,int feat\_type = SV\_FEAT\_MFCC\_1) –*** преобразует сигнал, заданный в параметре str в последовательность векторов признаков. Параметр *iSize* определяетдлину сигнала в отсчетах, *feat\_type* – тип вектора признаков (по умолчанию MFCC);

***Функция*** ***float fround(float x) –*** округление вещественного числа до ближайшего целого.