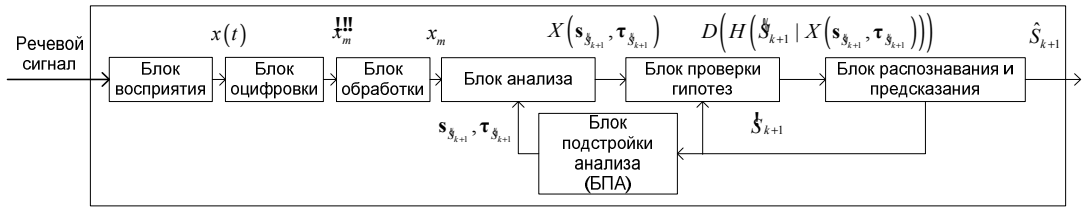

Рисунок 1. Обобщенная схема подобной системы распознавания речи

Где $x(t)$ – аналоговый электрический сигнал; $\overset{!!!}{x}_m$ – оцифрованный дискретный сигнал; x_m – сигнал после предварительной обработки (фильтрации от шумов, выравнивания энергии); $\overset{!}{S}_{k+1}$ – фонема предсказанная стохастической частью модели $\mathbf{s}_{\overset{!}{S}_{k+1}}, \boldsymbol{\tau}_{\overset{!}{S}_{k+1}}$ – соответственно множество анализируемых масштабов и смещений по времени; необходимых для поиска признаков предсказанной фонемы; $X\left(\mathbf{s}_{\overset{!}{S}_{k+1}}, \boldsymbol{\tau}_{\overset{!}{S}_{k+1}}\right)$ – результат вейвлет-анализа на заданных масштабах и смещениях по времени; $D\left(H\left(\overset{!}{S}_{k+1} \mid X\left(\mathbf{s}_{\overset{!}{S}_{k+1}}, \boldsymbol{\tau}_{\overset{!}{S}_{k+1}}\right)\right)\right)$ – решение относительно гипотезы о наличии в сигнале предсказанной фонемы при условии $X\left(\mathbf{s}_{\overset{!}{S}_{k+1}}, \boldsymbol{\tau}_{\overset{!}{S}_{k+1}}\right)$; $\hat{S}_{k+1}$ – распознанная фонема. Дикторонезависимые признаки закладываются в блок проверки гипотез в виде соответствующих предсказываемым фонемам алгоритмов анализа частотно-временной картины сигнала $X\left(\mathbf{s}_{\overset{!}{S}_{k+1}}, \boldsymbol{\tau}_{\overset{!}{S}_{k+1}}\right)$. Предсказание осуществляется на основе вычисления вектора вероятностей фонем (состояний системы) на шаге $k+1$, в соответствии с теорией Марковских процессов. При этом сначала выбирается состояние с максимальной вероятностью. Если оно не подтверждается, то далее в порядке убывания вероятности. Частотно временной анализ при этом выполняется для тех масштабов и смещений преобразования по времени, вычислении по которым еще не проводились. Таким образом, обратная связь от блока предсказания и распознавания на блок частотно-временного анализа позволяет сократить вычислительные затраты на непрерывное вейвлет-преобразование.

Эксперименты с вейвлет-портретами множества дикторов выявили существенно - значимые дикторонезависимые признаки фонем русской речи. Определены минимальные множества масштабов и смещений преобразования по времени необходимые для поиска данных признаков в частотно-временной картине сигнала. Выявлена зависимость положения формантных частот и периода элементарных повторяющихся частей вокализованных фонем от частоты основного тона. Предложен алгоритм определения частоты основного тона и определения вокализованности/невокализованности участка речевого сигнала на основе непрерывного вейвлет-преобразования с минимизацией множества масштабов и смещений преобразования по времени. Разработан алгоритм ускоренного вычисления непрерывного вейвлет-преобразования на основе БПФ.

USING .NET COMPONENTS IN THE DEVELOPMENT OF FAULT-TOLERANCE SOFTWARE

Maymistov D.S.
Siberian State Aerospace University named after academician M.F. Reshetnev

Multi-version programming (MVP) or N-version programming (NVP) concept was first introduced by Algidras Avizhenis in 1977. The main idea of MVP is that several versions of the same algorithm are used simultaneously to deal with the same system tasks. Than the algorithms' results are analyzed, and the one, which best meets the system objectives at the current situation, is selected. The selection is carried out according to the inner logics of the system, thus the system reliability is increased. MVP can be used in dealing with a large number of various tasks in complex systems. It is obvious that the building of such systems requires a general concept and a general approach to the development of algorithms, which execute separate tasks. Component-oriented programming methodology perfectly meets these requirements.

Component-oriented programming has been introduced relatively recently. According to this methodology software is built from ready components similarly to the way a building is built of blocks by construction workers. Component-oriented software development implies adding components to the project during the design time. At this time the components are adjusted. Components do offer any user interface neither to a programmer nor to an end user, so special wizards and designers of an integrated development system have to be used to do this. The first component-oriented development environment was created by Microsoft at the very beginning of their life. Later many other development environments based on the same principles were created. So by the end of 20th century the majority of development environment manufacturers had introduced the support for component-oriented programming in their products.

At present, the most advanced component model is offered by Microsoft in their .NET platform.

Component, as Microsoft perceive it, is binary code and data, joined together in an alienable form and able to be used later in the development of software systems. Alienability implies the possibility to use a component without any additional knowledge about it. In practice, it means that a component has to contain all the necessary information about itself. Component also must have a public interface to allow executing the code inside itself. Alienability also means that a component instance can be

created dynamically and it is not necessary to use any compilers and interpreter.

In other words, a component is a class, which provides information about itself (metainformation) and which instances can be created dynamically without any static information about itself.

Almost every class in .NET meets these requirements: metainformation is created for any class member, any class instance can be created dynamically and all classes can be put into assemblies (one or more executable modules), which can be distributed independently. However, it is not true that any class in .NET is a component. The reason is that .NET has a special class *Conponent*. Any class which must interact with the development environment must be derived from *Component* class.

NET remoting enables to build widely distributed applications easily, whether application components are all on one computer or spread out across the entire world. You can build client applications that use objects in other processes on the same computer or on any other computer that is reachable over its network. .NET remoting also is used to communicate with other application domains in the same process.

NET remoting provides an abstract approach to interprocess communication that separates the object from a specific client or server application domain and from a specific mechanism of communication. As a result, it is flexible and easily customizable. One communication protocol can be replaced with another, or one serialization format with another without recompiling the client or the server. In addition, the remoting system assumes no particular application model. Connection can be established from a Web application, a console application, a Windows Service – from almost anything you want to use. Remoting servers can also be any type of application domain. Any application can host remoting objects and provide its services to any client on its computer or network.

Basing on the foregoing description of the .NET component model basic concepts, it can be inferred that .NET components have the following advantages over the components, which are based on other concepts and technologies:

- Ability to integrate a component into any development environment, which supports appropriate Microsoft standards;
- Ability to create and distribute components by third-party developers;
- Ability to create components in any language and development environment, which supports appropriate Microsoft standards.

Consequently, it is obvious to choose .NET component technology for the development of multi-version components.

Sources:

1. Vladislav Chistyakov ".NET — classes, components and controls" RSDN Magazine №3 2003.

2. A.V. Kotenok "Development of the environment for multi-version execution of program modules" Vestnik NII SUVPT: proceedings collection; Krasnoyarsk: NII SUVPT. — 2003 №14. — p. 13–21.

ЗАВИСИМОСТЬ ПРИЗНАКОВ В ЗАДАЧАХ МНОГОМЕРНОЙ КЛАССИФИКАЦИИ ОБЪЕКТОВ

Романовская Т.С.

Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева

Рассмотрены вопросы определения функции зависимости многомерного признакового пространства при решении задач классификации. Вводится понятие эталонного состояния признака, в котором он не поддается влиянию других признаков. Предлагается процедура определения функции зависимости признаков, имеющих эталонное состояние.

В задачах многомерной классификации объектов признаковое пространство может быть достаточно сложным для его использования без предварительной обработки. Такая сложность, как правило, обусловлена тем, что признаки измерены в разных шкалах, часть признаков неинформативна или признаки зависимы, т.е. для признаков $(x_1, x_2, ..., x_i, x_j)$ существует функция f, такая что:

$x_i = f(x_1, x_2, ..., x_j) + x_{i0}$, $\quad i \neq j$; $j = 1, ..., M-1$, где M – размерность признакового пространства.

Задача классификации еще более усложнится, если предположить, что часть признаков характеризуется эталонным состоянием, в котором они условно не влияют друг на друга. То есть существует порог влияния группы признаков $(x_1, x_2, ..., x_j)$ на признак x_i такой, что $f(x_1, x_2, ..., x_j) = 0$, причем признаки $(x_1, x_2, ..., x_j)$ могут принимать значение отличное от Null.

Таким образом, каждый признак объекта, зависящий от других признаков, имеет матрицу состояний $F_o = (x_{io}, f_{io}, x_{i\max})$, где x_{io} - эталонное состояние (минимальное значение), определяемое отсутствием влияния на признак других признаков; f_{io} - функция зависимости признака; $x_{i\max}$ - максимальное значение признака, которое можно достичь путем изменения других признаков. Причем в эталонном состоянии признак может принимать сколько угодно разных значений. Функция зависимости может быть получена двумя способами: 1. Задана экспертно; 2. Получена с помощью аппарата математической статистики на основе ряда наблюдений.

В этом случае признаковое пространство $X = \{x_1, x_2, ..., x_M\}$ состоит их трех подпространств: $X_h^* = \{x_1^*, x_2^*, ..., x_h^*\}$ - подпространство признаков, независимых от других признаков; $X_g^* = \{x_{h+1}^*, x_{h+2}^*, ..., x_g^*\}$ - подпространство признаков, зависимых от других, не имеющих эталонного состояния в котором они условно не зависят от других признаков или не оказывают влияния на другие