УДК 004.93
DOI: 10.15827/2311-6749.17.1.2

ОБ ОДНОМ ОПЫТЕ ПРИМЕНЕНИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ПРИ РАЗРАБОТКЕ WEB-ИНСТРУМЕНТАРИЯ РАСПОЗНАВАНИЯ ОБРАЗОВ

В.В. Фомин, д.т.н., профессор, v_v_fomin@mail.ru;
И.В. Александров, аспирант, chrono555@yandex.ru
(ФГБОУ ВПО «РГПУ им. А.И. Герцена», наб. реки Мойки, 48, г. Санкт-Петербург, 191186, Россия)

Предлагается программное решение по увеличению качества распознавания образов и повышению эффективности инструментария машинного обучения посредством реализации Grid-технологий. Формулируются стратегические направления разработки инструментария распознавания образов в виде программной системы, основанной на принципах распределенных систем, распараллеливания и адаптивной настройки вычислительных ресурсов. Рассматривается структура организации web-инструментария распознавания образов с использованием концепции библиотеки алгоритмов. Даются алгоритмические решения распараллеливания алгоритмов обучения и распознавания на базе методов прецедентов.

Ключевые слова: *интеллектуальные информационные системы, распознавание образов, машинное обучение, web-системы, параллельные вычисления.*

В современном мире увеличение производительности обработки большого количества разнообразной информации достигается только в тех случаях, когда часть интеллектуальной нагрузки берут на себя компьютеры. Одним из направлений исследований в области искусственного интеллекта [1, 2] являются интеллектуальные информационные системы, которые находятся на пике исследовательской активности, практики их прикладного применения и коммерциализации и широко применяются в разных областях промышленности, экономики, медицины и т.д. Интеллектуальные информационные системы сосредоточили в себе наиболее наукоемкие технологии с высоким уровнем автоматизации процессов не только подготовки информации для принятия решений, но и выработки вариантов решений, опирающихся на полученные информационной системой данные.

Сложность проблем эффективного применения, использования, внедрения интеллектуальных систем определяет сложность подходов к классификации задач искусственного интеллекта и решается в рамках различных направлений исследований, основанных на близких моделях, методах и алгоритмах. К таким исследованиям относятся распознавание образов, машинное обучение, машинный перевод, интеллектуальный анализ данных и др. [2].

Технология интеллектуального анализа данных (data-mining) широко применяется в бизнес-аналитике. К концу первого десятилетия XXI в. рынок аналитического ПО, использующего технологии data-mining, достиг объема $7,8 млрд (с ежегодным ростом 12,1 %). А к 2022 г., согласно исследованиям Wise Guy Reports, рынок аналитического ПО и бизнес-аналитики достигнет объема в $35,26 млрд [3].

Одной из базовых апорий интеллектуального анализа данных являются технологии распознавания образов [4]. Распознавание образов – класс задач, связанных с определением принадлежности объекта к одному из классов объектов. Распознавание образов можно рассматривать как классификацию объектов. Сложность и многофакторность проблемы распознавания образов, разнообразие проблемных областей и категорий задач, отсутствие универсальных методов решения задач распознавания приводят к тому, что традиционная методология разработки систем распознавания, как правило, не позволяет достигнуть требуемой эффективности результатов в рамках промышленного применения.

В настоящее время существует значительное противоречие, обусловленное уровнем развития формального аппарата теории распознавания образов, а также сравнительной ограниченностью и узкой направленностью большинства из существующих прикладных систем распознавания. Данное противоречие частично может быть преодолено путем разработки интегрированных многофункциональных автоматизированных информационных систем распознавания, ориентированных на решение широкого класса задач и допускающих адаптацию и настройку на новую предметную область и категорию задач.

Описание программы. В рамках исследований по тематике «web-инструментарий машинного обучения» в качестве прототипа облачного ресурса была разработана программа «Web-инструментарий распознавания образов на основе расширяющейся библиотеки методов машинного обучения» [5]. Инструментарий предназначен для решения обширного круга задач и может быть применен в различных интеллектуальных областях анализа данных: образование, экономика, финансы, медицина, геология и др. Программа может быть использована для принятия решений широким кругом лиц, в том числе студентами, преподавателями, бизнесменами, специалистами в различных областях.

Общий принцип работы программы показан на рисунке 1.

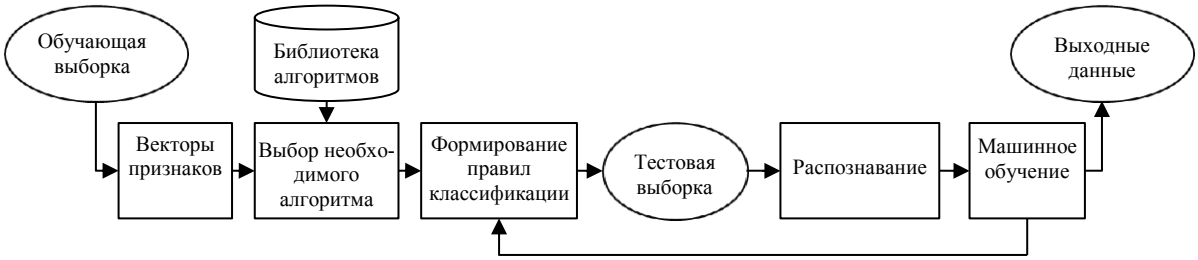

Рис. 1. Общий принцип работы программы

В базовой версии программы был реализован класс методов прецедентов как пользующаяся популярностью классика в области data-mining. Эти методы состоят из следующих циклов: 1) извлечение похожего прецедента из библиотеки; 2) использование выбранного прецедента для построения решения; 3) проверка полученного решения; 4) занесение решения как нового прецедента в библиотеку. Таким образом, системой накапливается опыт (прецеденты) и реализуется машинное обучение, что способствует повышению качества принимаемого решения.

Главным достоинством методов прецедентов является их высокая эффективность для множества задач с точки зрения достоверности результата и компактности алгоритмов анализа, а главным недостатком – требование значительных вычислительных ресурсов. Это особенно актуально при работе с большими объемами данных.

В качестве основных алгоритмов анализа в программе были реализованы такие методы прецедентов, как метод *k* ближайших соседей, метод потенциальных функций, метод эталонов, метод корреционных приращений, метод наименьших среднеквадратических ошибок, наивный байесовский метод (см. таблицу).

Классификация методов прецедентов

Название метода распознавания	**Достоинства**	**Недостатки**
Метод *k* ближайших соседей	Возможность обновления обучающей выборки без переобучения классификатора. Устойчивость к аномальным выбросам	Высокая зависимость результатов классификации от меры расстояния (метрики). Необходимость полного перебора обучающей выборки при распознавании. Вычислительная трудоемкость
Метод потенциальных функций	Хранение лишь части выборки. Нелинейная разбивка множества объектов	Зависимость результата обучения от порядка предъявления объектов. Трудность выбора подходящей потенциальной функции. Трудоемкость вычислений при большом объеме обучающей выборки
Метод эталонов	Хранение лишь части выборки. Низкие вычислительные затраты	Высокая зависимость результатов классификации от меры расстояния (метрики)
Наивный байесовский классификатор	Нечувствительность к размерам обучающей выборки. Высокая скорость обучения и устойчивость к переобучению	Наличие предположения о независимости признаков, что ведет к низкой точности классификации в реальных задачах
Алгоритм корректирующих приращений	Высокая устойчивость к аномальным отклонениям	Трудоемкость вычислений, медленная сходимость, алгоритм сходится только в случае, когда обучающие выборки не перекрываются в пространстве распознающих признаков
Алгоритм наименьшей среднеквадратичной ошибки	Высокая устойчивость к аномальным отклонениям. Возможность оценки разделимости классов в процессе обучения	Трудоемкость вычислений, алгоритм сходится только в случае, когда обучающие выборки не перекрываются в пространстве распознающих признаков

Вышеперечисленные методы основаны на алгоритмах метрической классификации и оценивании сходства объектов. Также для всех методов характерно наличие обучающей выборки.

Программа позволяет произвести распознавание образов на основе методов машинного обучения посредством реализации web-сервиса и использования web-технологий, что дает возможность применить технологию облачных вычислений для повышения скорости распознавания путем распределения вычислительных ресурсов между компьютерами, и технологию облачного хранения для повышения объема обрабатываемых данных и скорости доступа к ним за счет хранения их на серверной стороне, а не на машине клиента, а также для обеспечения резервного копирования.

Компоненты web-технологий в программе реализованы с помощью сервлетов, которые расширяют функциональность web-серверов. Выбор в пользу сервлетов обусловлен их следующими достоинствами: позволяют писать платформенно-независимый код, предоставляют возможность масштабировать систему, могут выполняться параллельно в рамках одного процесса на сервере. Характерной особенностью сервлетов является то, что они не требуют создания новых процессов при каждом новом запросе.

Структура управления программой и ее основные функции. В интерфейсе программы присутствуют несколько информационных блоков. На рисунке 2 показана структура расположения основных информационных блоков.

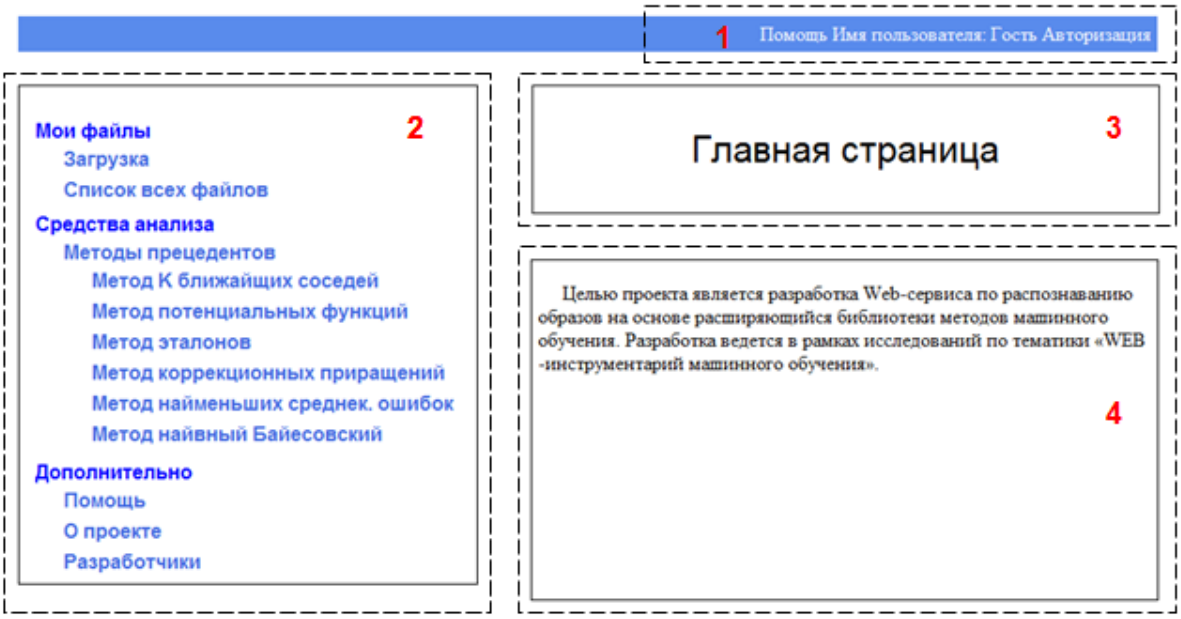

Рис. 2. Структура расположения информационных блоков

1. Системное меню. Содержит ссылки на элементы системы: Помощь и Авторизация. В разделе «Помощь» можно ознакомиться с доступными методами анализа данных и их описанием. В разделе «Авторизация» можно войти в свой профиль. Также в системном меню выводится имя пользователя.

2. Главное меню. Содержит ссылки на модули, обеспечивающие доступ к управлению файлами, алгоритмам распознавания, помощи и организационной информации. Меню структурно разделено на три раздела:

– «Мои файлы», отвечающий за управление файлами, обеспечиваемое модулями «Загрузка» и «Список всех файлов»; модуль «Загрузка» позволяет загрузить на сервер новый файл с возможностью его описания, модуль «Список всех файлов» – просмотреть название и дополнительную информацию о загруженных файлах в области 4;

– «Средства анализа», отвечающий за распознавание образов одним из шести предложенных алгоритмов;

– «Дополнительно», отвечающий за предоставление организационной информации и помощи.

3. Название страницы. Описывает характер производимого действия и тип информационного блока, который отображается в области 4.

4. Рабочая область. Содержит информацию, относящуюся к модулю, с которым происходит взаимодействие. С помощью рабочей области осуществляется поддержка БД загруженных файлов (например, позволяет совершить следующие действия с файлами на сервере: скачать файл на компьютер клиента, просмотреть файл, удалить файл из БД и т.д.). Также в рабочей области выводится дополнительная информация о файлах: описание файла, его размер и дата последнего изменения.

Распознавание образов. Распознавание образов происходит в два этапа.

1. Выбирается один из шести методов распознавания образов. Для этого необходимо воспользоваться одним из пунктов меню раздела «Средства анализа» в «Главном меню». На рисунке 3 представлена страница одного из методов – метода эталонов.

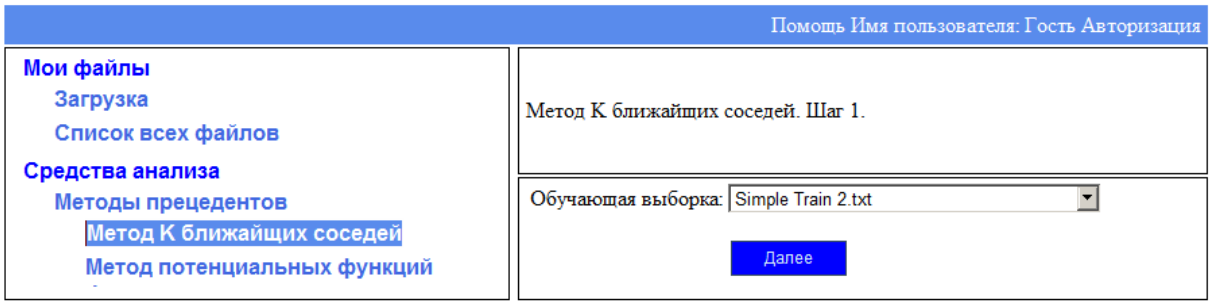

Рис. 3. Страница первого шага метода k ближайших соседей

В списке «Обучающая выборка» необходимо выбрать выборку, по которой будет проходить обучение алгоритма, и нажать кнопку «Далее».

2. После нажатия кнопки «Далее» появится страница второго шага (рис. 4).

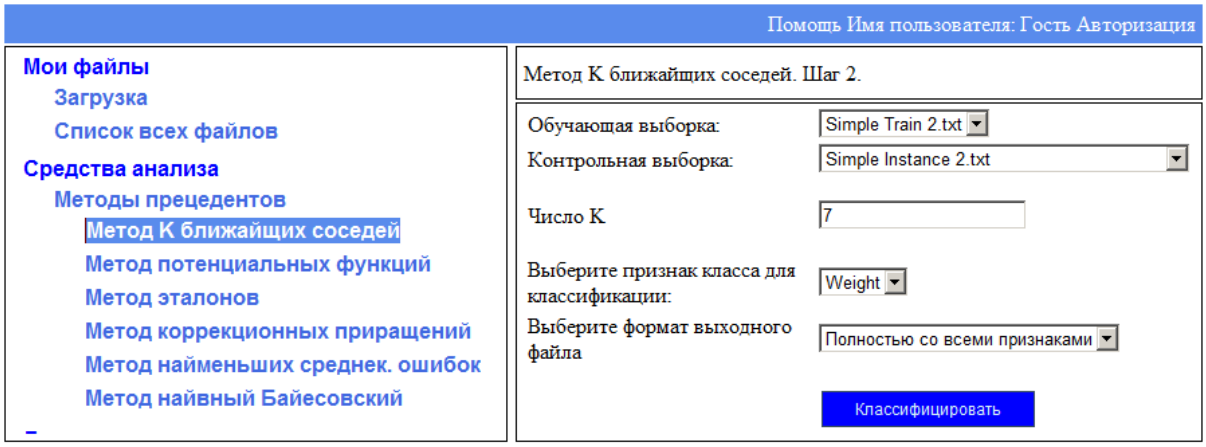

Рис. 4. Страница второго шага метода k ближайших соседей

В поле «Обучающая выборка» можно увидеть выборку, по которой будет происходить обучение алгоритма. (На этом шаге поменять ее уже нельзя, так как она влияет на список признаков класса.)

Для запуска процесса распознавания образов необходимо выполнить следующее:

– в списке «Контрольная выборка» определить выборку для распознавания;

– в списке «Признак класса для классификации» выбрать признак класса, по которому необходимо распознать выборку;

– в списке «Формат выходного файла» выбрать формат выходного файла: а) записать только один атрибут (признак класса), по которому происходило распознавание образов; б) записать все атрибуты выборки;

– с помощью кнопки «Классифицировать» запустить процесс распознавания образов.

В будущем при увеличении количества поступающих объектов трудоемкость процесса распознавания методов прецедентов будет увеличиваться с каждым разом, что отразится на времени их распознавания.

Алгоритмы распараллеливания методов прецедентов. Борьба за эффективное использование вычислительных ресурсов привела к появлению технологий распределенных систем и параллельных вычислений [6], в частности Grid-технологий и Internet-технологий. Grid-система строится по принципу клиент-сервер, состоит из одного или нескольких компьютеров-серверов и множества компьютеров-клиентов свободной конфигурации и представляет собой эффективную для распараллеливания структуру вычислительных ресурсов.

В ходе исследования были разработаны три режима распознавания образов: стандартное распознавание (без распараллеливания), с распараллеливанием алгоритма обучения, с распараллеливанием алгоритма распознавания.

Рассмотрим вышеперечисленные виды распараллеливания на примере метода *k* ближайших соседей, суть которого заключается в нахождении *k* объектов с минимальным расстоянием до нового (поиск соседей) и в определении среди них наиболее встречаемого класса.

Вариант 1. Происходит распараллеливание алгоритма обучения. Схема такого распараллеливания включает три этапа. Первый этап состоит в разбиении файла с обучающими объектами на *N* файлов и в отправке их с полным файлом классифицируемых объектов на *N* компьютеров.

Второй этап отвечает за отбор *k* ближайших соседей из *i*-й части файла с обучающими объектами для каждого классифицируемого объекта (рис. 5).

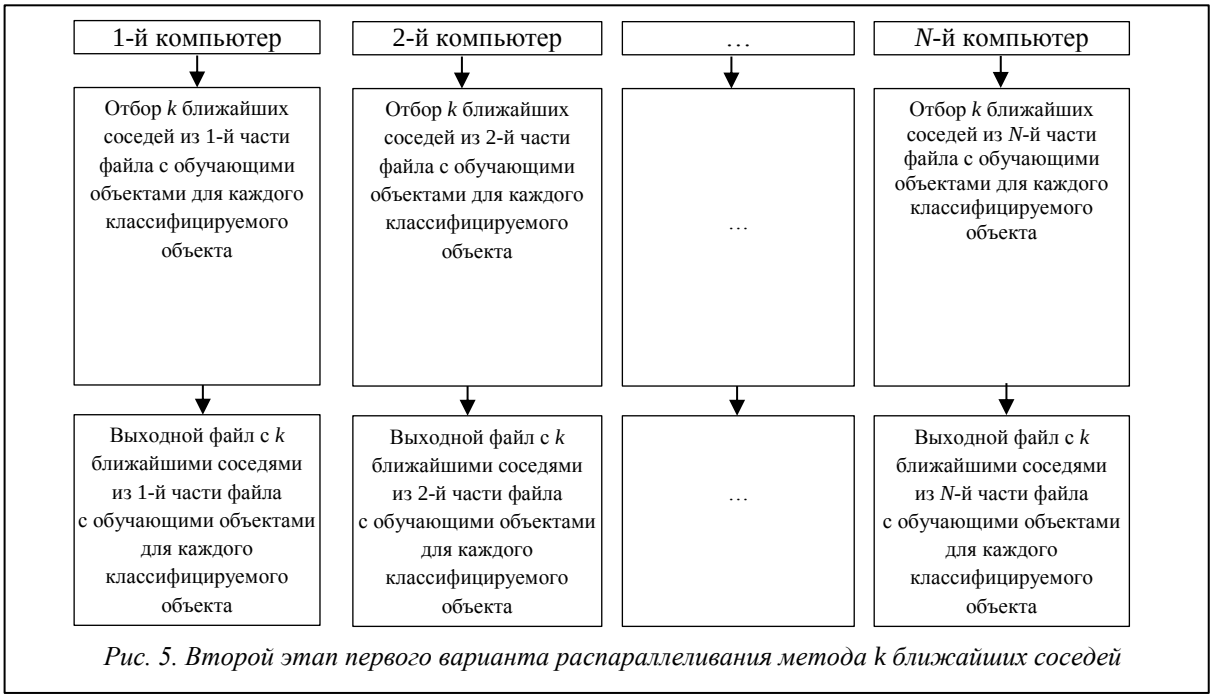

Рис. 5. Второй этап первого варианта распараллеливания метода k ближайших соседей

Третий этап (рис. 6) отвечает за просмотр для каждого классифицируемого объекта всех *N* выходных файлов и отбора *k* ближайших соседей по ним с последующим определением класса классифицируемого объекта по этим соседям.

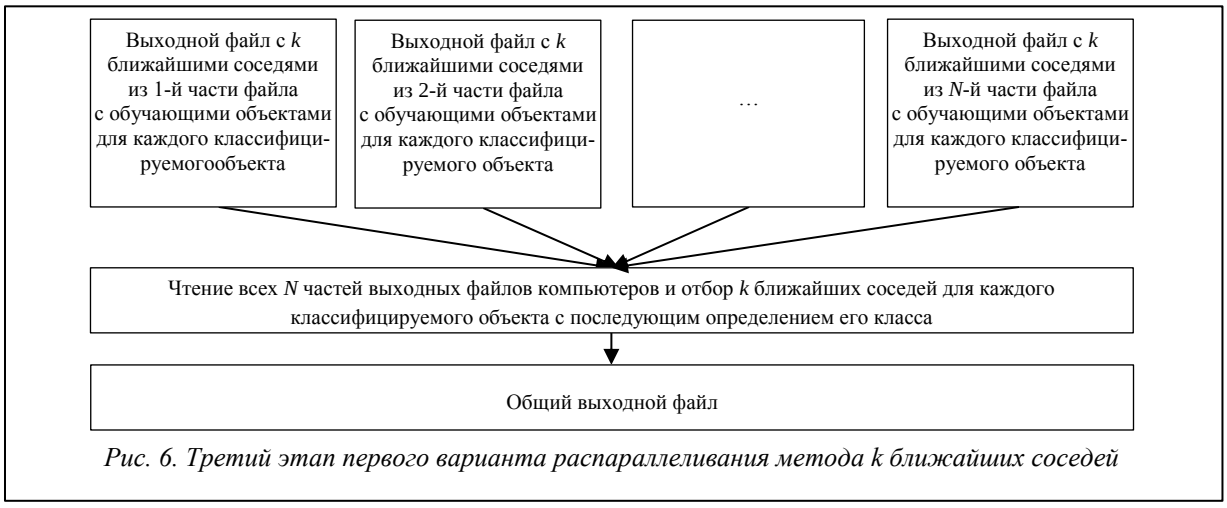

Рис. 6. Третий этап первого варианта распараллеливания метода k ближайших соседей

Приведем часть кода функции, отвечающую за третий этап:

```
// Цикл по количеству классифицируемых объектов
for (int count = 0; count < Number_Data; count ++)
{
   // Цикл по количеству частей разбиения файла с обучающими объектами
   for (int k = 0; k < _Vector_Names_Parts_Files.size(); k++)
   {
```

```
...
  // Открытие текущей части файла с обучающими объектами на чтение и перемещение
  // курсора к месту начала записей в файле
 File_For_Read = new FLS_Class_For_Work_With_Files_Read (Path_Current_Part_File);
 File_For_Read.Open_File_For_Read();
 File_For_Read.Skip_Info_Header();
 // Пропуск необходимого количества записей до k ближайших записей
 // текущего классифицируемого объекта
 int Number_Skip_Records = count*(Number_K_Current_Part_File+1);
 for (int i = 0; i < Number_Skip_Records; i++)
     File_For_Read.Read_Data_Object_With_Class_And_Distance(ob,
     (Data_File_First_Part_File.Get_Size_Info_Header()-1));
...
  // Цикл числа k
  for (int i = 0; i < Number_K_Current_Part_File; i++)
  {
     // Чтение текущего ближайшего соседа
     File_For_Read.Read_Data_Object_With_Class_And_Distance(ob,
     (Data_File_First_Part_File.Get_Size_Info_Header()-1));
     // Если количество в массиве k ближайших соседей меньше числа k
     if (Vector_K_Objects.size() < Number_K)
     {
        // Добавляем в массив считанный объект
        DataObject ob_temp = new DataObject(Data_File_First_Part_File.Get_Size_Info_Header()-1);
        for (int index = 0; index < ob.get_number_attrib(); index ++)
            ob_temp.set_attrib(index, ob.get_attrib(index));
        ob_temp.set_class(ob.get_class());
        ob_temp.set_dist(ob.get_dist());
        Vector_K_Objects.add(ob_temp);
      }
      else
      {
         // Сравниваем расстояние от считанного объекта до текущего классифицируемого
          // объекта и расстояние от самого дальнего объекта в массиве k ближайших соседей
         // до текущего классифицируемого объекта
         double Distance_Last_K_Objects = Vector_K_Objects.get(Number_K - 1).get_dist();
          if (ob.get_dist() < Distance_Last_K_Objects)
          {
             // Если расстояние меньше у считанного объекта, заменяем самый
             // дальний объект в массиве до классифицируемого на считанный объект
             for (int index = 0; index < ob.get_number_attrib(); index ++)
                  Vector_K_Objects.get(Number_K - 1).set_attrib(index, ob.get_attrib(index));
              Vector_K_Objects.get(Number_K - 1).set_class(ob.get_class());
              Vector_K_Objects.get(Number_K - 1).set_dist(ob.get_dist());
           }
       }
       // После записи считанного объекта в массив сортируем массив по расстоянию
       // до текущего классифицируемого объекта
       Collections.sort(Vector_K_Objects);
   ...
}
```

Вариант 2. Происходит распараллеливание алгоритма распознавания, состоящего из трех этапов. Первый этап заключается в разбиении файла с классифицируемыми объектами на *N* файлов и отправке их с полным файлом обучающих объектов на *N* компьютеров (рис. 7).

За разбиение файла на несколько частей отвечает функция Divide_File, принимающая два значения: имя файла, который будет разбиваться на несколько частей, и массив имен этих частей. Часть кода функции представлена далее:

```
...
// Открытие файла на чтение и перемещение
// курсора к месту начала записей в файле
```

```
FLS_Class_For_Work_With_Files_Read File_For_Read;
File_For_Read = new FLS_Class_For_Work_With_Files_Read (_Path_File);
File_For_Read.Open_File_For_Read();
File_For_Read.Skip_Info_Header();
...
// Цикл по количеству файлов на разбиение
for (int k = 0; k < _Vector_Names_Parts_Files.size(); k++)
{
   // Открытие текущего файла на запись
  String Path_File_Part = _Vector_Names_Parts_Files.get(k);
  FLS_Class_For_Work_With_Files_Write File_For_Write = new
  FLS_Class_For_Work_With_Files_Write (Path_File_Part);
        File_For_Write.Open_File_For_Write();
     ...
        // Цикл по количеству записей в читаемом файле
        for (int j = 0; j < number_records && number_current_records < Data_File.Get_Number_Data();
        j++)
        {
           // Цикл по количеству атрибутов в записи
           for(int i = 0; i < Data_File.Get_Size_Info_Header(); i++)
           {
              // Читаем один атрибут из файла для чтения
              str_temp = File_For_Read.Read_Word();
              // Записываем атрибут в текущий файл для записи с форматированием текста
              File_For_Write.Write_Attribute(str_temp, Data_File.Get_Array_Max_Lenght_Attribute()[i]);
              // Если атрибут не последний, записываем пробел
              if (i != Data_File.Get_Size_Info_Header()-1)
                 File_For_Write.Write_Space();
           }
           File_For_Write.Write_Enter();
           // Увеличиваем число прочитанных записей
           number_current_records++;
        }
```

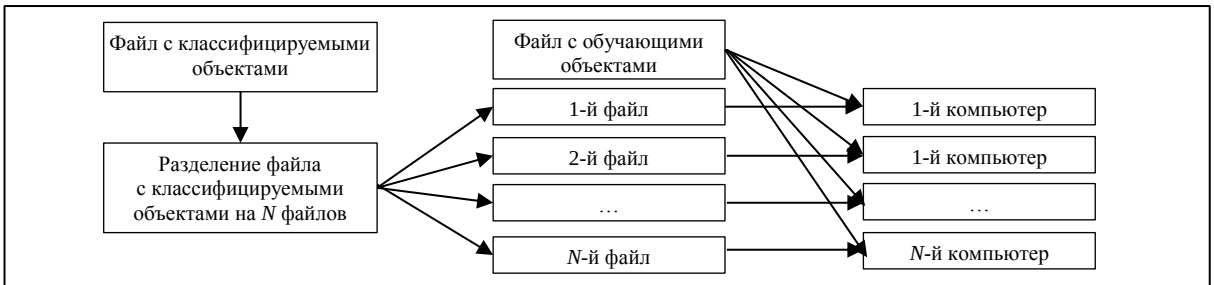

Рис. 7. Первый этап второго варианта распараллеливания метода k ближайших соседей

Второй этап заключается в распознавании *i*-й части классифицируемых объектов на *i*-м компьютере при наличии всех обучающих объектов и в формировании *i*-го выходного файла. Процесс формирования *k* ближайших соседей ничем не отличается от первого варианта распараллеливания. Принципиальное отличие появляется после формирования *k* ближайших соседей. В первом варианте в выходной файл *i*-го компьютера записываются все *k* ближайших соседей из *i*-й части обучающего файла для каждого классифицируемого объекта. В данном же варианте распараллеливания программа обрабатывает все *k* ближайших соседей и по ним определяет класс классифицируемого объекта.

Распознавание состоит из трех функций: функция чтения классифицируемого объекта из файла и записи необходимой информации в выходной файл, функции формирования *k* ближайших соседей классифицируемого объекта из обучающей выборки, функции определения и записи класса классифицируемого объекта по *k* ближайшим соседям. Приведем часть кода функции формирования *k* ближайших соседей классифицируемого объекта из файла с обучающими объектами:

```
// Пока не конец файла
while (File_For_Read.Has_Next())
 {
```

```
 // Считывание объекта из файла с обучающими объектами
 File_For_Read.Read_Data_Object_With_Class (ob,Get_Data_File_Train().Get_Index_Class());
...
 // Вычисление расстояния от считанного объекта до классифицируемого объекта
 Calculation_Distance_Object_X_Object (ob,X_Object);
...
 // Сравнение расстояний до классифицируемого объекта межу считанным объектом и самым
// дальним объектом в массиве ближайших соседей
 if (ob.get_dist() < Vector_K_Objects.get(Number_K - 1).get_dist())
 {
     // Записываем считанный объект в массив вместо самого дальнего объекта до
     // классифицируемого
     for(int j = 0; j < Get_Data_File_Train().Get_Number_Attrib(); j++)
        Vector_K_Objects.get(Number_K - 1).set_attrib(j, ob.get_attrib(j));
     Vector_K_Objects.get(Number_K - 1).set_class(ob.get_class());
     Vector_K_Objects.get(Number_K - 1).set_dist(ob.get_dist());
     Vector_K_Objects.get(Number_K - 1).set_id(ob.get_id());
     // После записи считанного объекта в массив сортируем массив по возрастанию
     // расстояний до классифицируемого объекта
     Collections.sort(Vector_K_Objects);
 }
...
}
```

Третий этап состоит в отправке всех *N* выходных файлов на главный компьютер и в сборке их в единый файл. За сборку общего файла из нескольких частей отвечает функция Gather_Files, которая принимает два значения: имя общего файла и массив имен этих частей. Часть кода функции представлена далее:

```
...
// Цикл по количеству файлов для сборки
for (int k = 0; k < _Vector_Names_Parts_Files.size(); k++)
 {
   // Открытие текущего файла на чтение
   String Path_Current_Part_File = _Vector_Names_Parts_Files.get(k);
   File_For_Read = new FLS_Class_For_Work_With_Files_Read (Path_Current_Part_File);
   File_For_Read.Open_File_For_Read();
   // Перемещение курсора к месту начала записей в файле
   File_For_Read.Skip_Info_Header();

   String str_temp;
   // Пока не конец файла
   while (File_For_Read.Has_Next())
   {
      // Цикл по количеству атрибутов в записи
     for(int j = 0; j < Data_File_First_Part_File.Get_Size_Info_Header(); j++)
     {
        // Чтение атрибута
       str_temp = File_For_Read.Read_Word();
       // Записываем атрибут в файл с форматированием текста
      File_For_Write.Write_Attribute(str_temp,
      Data_File_First_Part_File.Get_Array_Max_Lenght_Attribute()[j]);
      // Если атрибут не последний, записываем пробел
       if (j != Data_File_First_Part_File.Get_Size_Info_Header()-1)
           File_For_Write.Write_Space();
     }
    File_For_Write.Write_Enter();
    }
   // Закрытие файла на чтение
    File_For_Read.Close_File_For_Read();
 }
```

В ходе исследования было разработано web-ориентированное инструментальное средство по распознаванию образов в идее расширяемой библиотеки методов на основе прецедентов, реализованы две схемы распараллеливания на основе Grid-технологий.

Реализуемая архитектура инструментария позволяет решать задачу повышения эффективности вычислительных ресурсов через процедуры настройки структуры подключаемых методов и алгоритмов обработки данных и реконфигурирования подключаемых технических вычислительных средств сети Internet..

Литература

1. Козлов А.Н. Интеллектуальные информационные системы. Пермь: Изд-во Пермская ГСХА, 2013. 278 с.

2. Фомин В.В., Миклуш В.А. Интеллектуальные информационные системы. СПб: Изд-во РГГМУ, 2013. 150 с.

3. Business Intelligence and Analytics Software Market Outlook – Global Trends, Forecast, and Opportunity Assessment (2014-2022). URL: https://www.wiseguyreports.com/reports/231297-business-intelligence-and-analytics-software-market-outlook-global-trends-forecast-and-opportunity-assessment-2014-2022 (дата обращения: 10.11.2016).

4. Ту Дж., Гонсалес Р. Принципы распознавания образов. М.: Мир, 1978. 412 с.

5. Сикулер Д.В., Фомин В.В. Концепция internet-системы интеллектуальной обработки данных // Некоторые актуальные проблемы современной математики и математического образования: материалы LXIV науч. конф. СПб: Изд-во РГПУ, 2011. С. 206–209.

6. Петров Е.В., Фомин В.В. Organization of parallel processing for implementation of web data mining system // Ученые записки РГГМУ. 2014. № 33. С. 149–154.