Лабораторная работа №1

Знакомство с графическим интерфейсом пользователя библиотеки алгоритмов Data Mining

Цель и задача работы

Ознакомиться и получить навыки работы GUI интерфейсом библиотеки data mining алгоритмов Xelopes

Построить задачи поиска ассоциативных правил, кластеризации и классификации для файлов transact.arff и weather-nominal.arff.

Теоретические положения

Библиотека Xelopes

Xelopes свободно распространяемая библиотека, обеспечивающая универсальную основу для стандартного доступа к алгоритмам data mining. Она была разработана немецкой компанией ProdSys в тесном сотрудничестве со специалистами российской фирмы ZSoft. Для удобной работы с библиотекой с ней поставляется GUI интерфейс GUI Xelopes, реализованный в виде отдельного приложения. Он позволяет выполнять следующие основные функции:

- Загрузить данные представленные в виде текстового файла формата arff и просмотреть их в табличном виде.
- Получить информацию об атрибутах данных (полях таблицы)
- Получить статическую информацию об исходных данных:
- Построить модель data mining.
- Для ассоциативных правил, деревьев решений и дейтограмм визуализировать построенную модель.
- Сохранить модель и применить ее в дальнейшем.

Рассмотрим перечисленные функции более подробно.

Загрузка и просмотр исходных данных

Для загрузки исходных данных необходимо открыть диалог представленный на рис. 1.1. Это можно выполнить или нажатием кнопки **Open Mining Data** на панели инструментов или выбором пункта меню **File | Open Mining Data**. Кроме того диалог открывается при запуске программы.

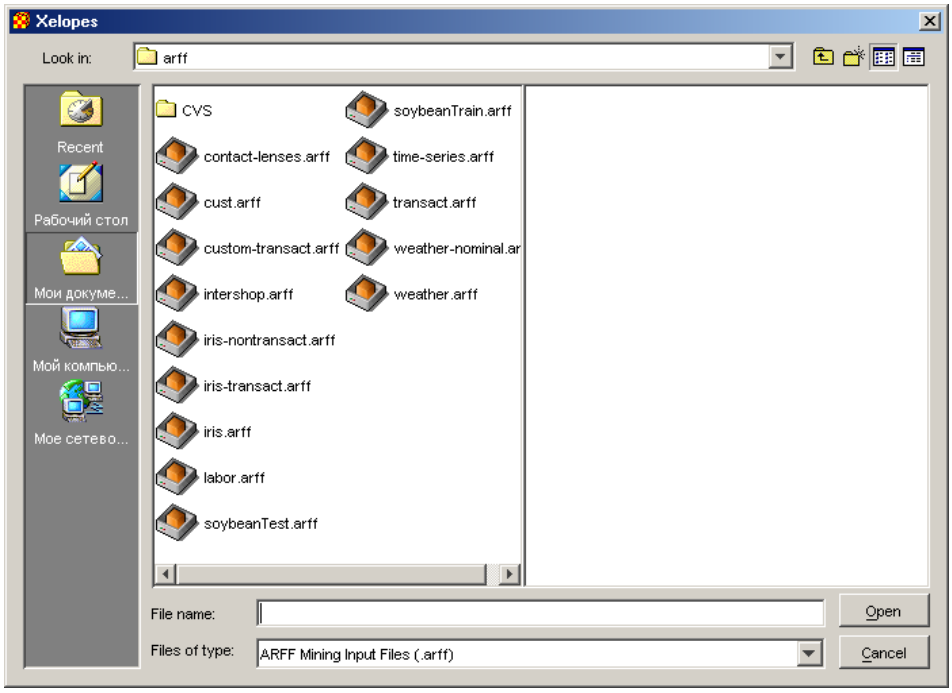


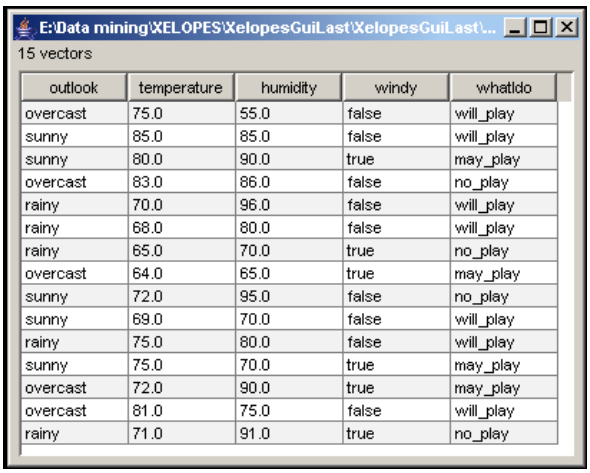
Рисунок 1.1. - Диалог загрузки исходных данных

Используя данный диалог необходимо выбрать текстовый файл с данными, представленными в формате arff. Нажатие на кнопку **Open** приведет к загрузке данных из выбранного файла.

После загрузки данных на панели инструментов становятся доступными следующие кнопки:

- **View Input Data** – отображение исходных данных;
- **Display Data Description** – получение информации о атрибутах исходных данных;
- **Display Descriptive Statistics** – получение статистической информации об исходных данных;
- **Build Mining Model** – генерация mining модели для загруженных исходных данных.

Для просмотра исходных данных в табличном виде необходимо нажать кнопку **View Input Data** на панели инструментов или выбрать пункт меню **File | View Data Source**. При этом открывается окно представленное на рис. 1.2. В заголовке окна отображается полный путь к файлу, из которого были загружены данные. Данные представляются в виде таблицы в которой строки соответствуют исследуемым объектам, а колонки атрибутам характеризующим их. Над таблицей можно заметить информацию об общем количестве объектов (векторов) представленных в таблице.



outlook	temperature	humidity	windy	whatIdo
overcast	75.0	55.0	false	will_play
sunny	85.0	85.0	false	will_play
sunny	80.0	90.0	true	may_play
overcast	83.0	86.0	false	no_play
rainy	70.0	96.0	false	will_play
rainy	68.0	80.0	false	will_play
rainy	65.0	70.0	true	no_play
overcast	64.0	65.0	true	may_play
sunny	72.0	95.0	false	no_play
sunny	69.0	70.0	false	will_play
rainy	75.0	80.0	false	will_play
sunny	75.0	70.0	true	may_play
overcast	72.0	90.0	true	may_play
overcast	81.0	75.0	false	will_play
rainy	71.0	91.0	true	no_play

Рисунок 1.2. - Исходные данные в табличном виде.

Информация об атрибутах данных

Интерфейс GUI Xelopes позволяет получить подробную информацию о атрибутах загруженных данных. Для этого необходимо нажать на кнопку **Display Data Description** на панели инструментов. Информация представляется в диалоговом окне **Variables** (рис 1.3.). В верхней части окна выводится название данных (на рисунке это weather). В правой части окна представлен список атрибутов. В левой части информация о выбранном атрибуте в зависимости от его типа.

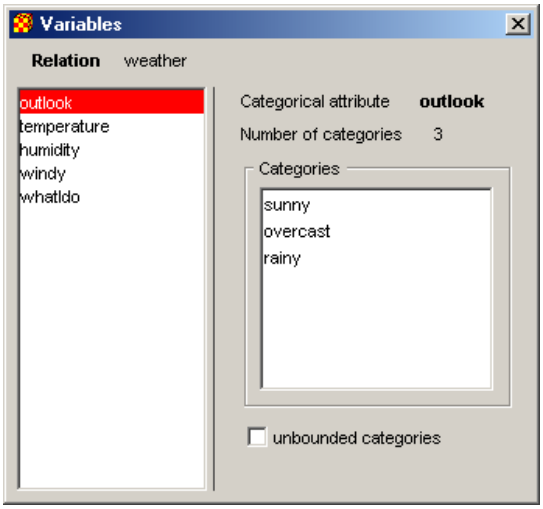


Рисунок 1.3. - Информация о категориальном атрибуте.

В Xelopes различают два основных типа атрибутов: категориальный и числовой. В зависимости от типа меняется и информация об атрибуте. Для любого атрибута выводится его название и тип.

Для категориальных атрибутов (рис. 1.3.) отображается информация о принимаемых им значениях (категориях): количестве (Number of categories) и списке значений (Categories). Если количество категорий не ограничено, то будет отмечен флаг unbounded categories.

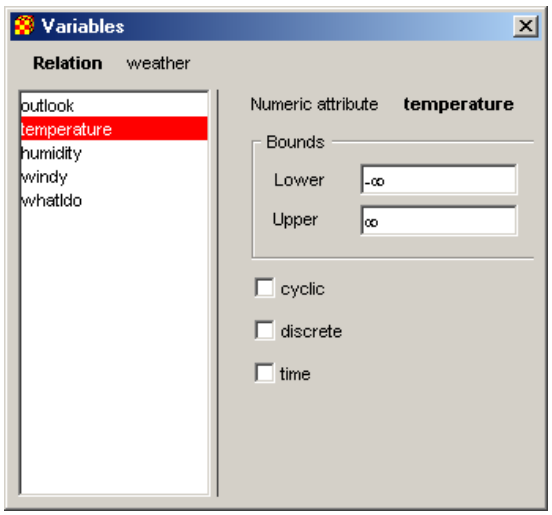


Рисунок 1.4. - Информация о числовом атрибуте

Для числовых атрибутов (рис. 1.4.) отображается информация о наибольшем (Upper) и наименьшем (Lower) значениях. Кроме того, в зависимости от свойств атрибута могут быть установлены следующие флажки:

- Cyclic – если значения атрибута циклические (т. е. может быть определено понятие расстояния)
- Discrete – если значениями атрибута являются дискретные величины
- Time – если атрибут представляет собой время.

Статистическая информация о данных

Для получения статистической информации о данных необходимо нажать кнопку **Display Descriptive Statistics** на панели инструментов или выбрать пункт меню **File | Statistics.** В открывшемся диалоговом окне **Statistics** (рис.1.5.) необходимо выполнить настройку отображаемой информации.

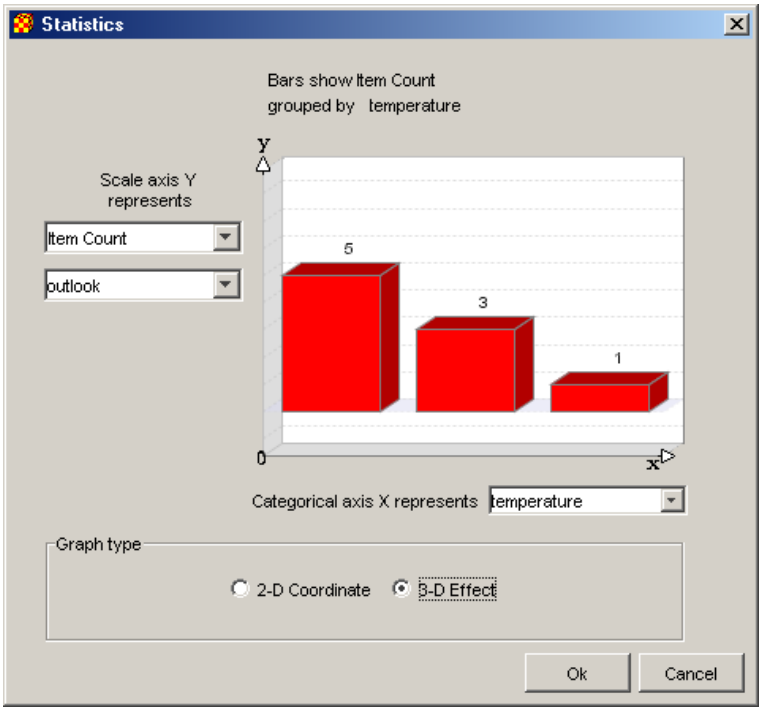


Рисунок 1.5. - Диалог настроек представления статистической информации по исходным данным.

Необходимо настроить следующие параметры:

- Тип отображаемой информации
- Атрибуты, откладываемые по осям X и Y
- Мерность отображаемой информации: в 2-х или 3-х мерном пространстве.

После настройки необходимых параметров нажатием на кнопку ОК можно получит статистическую информацию выбранного типа. Для настроек представленных на рис. 1.5. будет открыто диалоговое окно с информацией изображенное на рис 1.6.

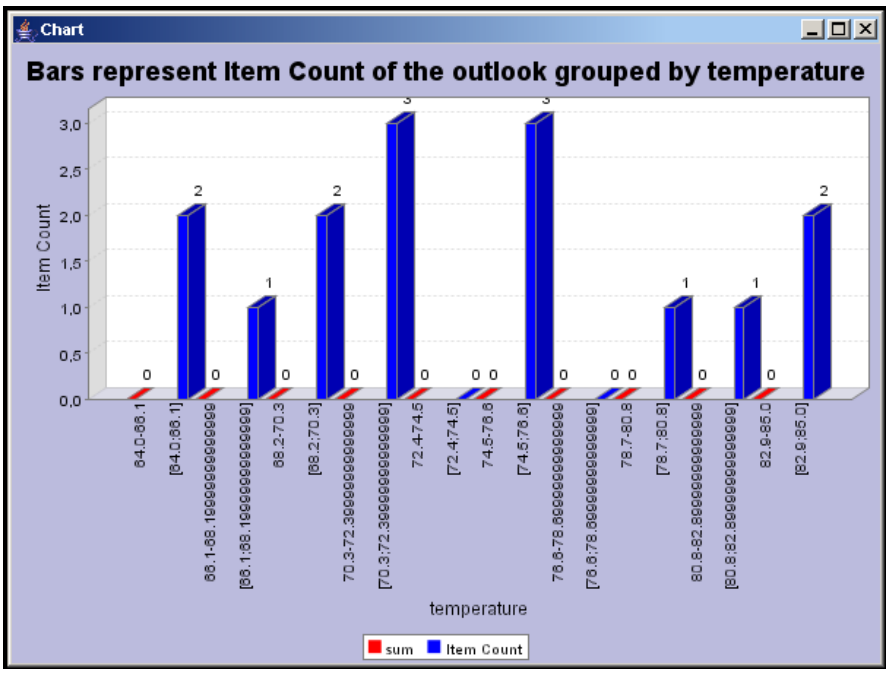


Рисунок 1.6. - Пример статистической информации по исходным данным.

Можно получить следующие типы информации:

- Количество объектов (Item Count)
- Минимальные (Minimal) и максимальные (Maximal) значения
- Предел (Range) значений
- Сумма (Sum) значений
- Среднее значение (Mean) др.

Построение mining модели

В результате применения методов data mining должна быть построена mining модель. Для этого необходимо нажать кнопку **Build Mining Model** на панели инструментов или выбрать пункт меню **Model | Build**. В результате откроется диалоговое окно предлагающее построить один из типов модели для загруженных ранее данных (рис. 1.7).

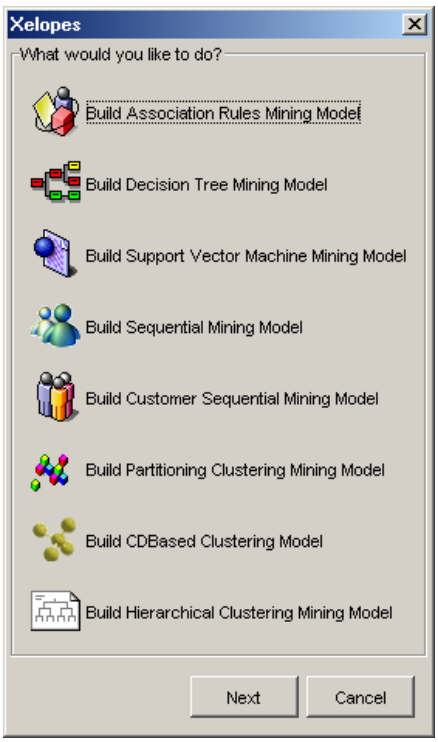


Рисунок 1.7. - Типы моделей создаваемых алгоритмами библиотеки Xelopes.

Для построения доступны следующие модели:

- ассоциативные правила (Association Rules Mining Model);
- деревья решений (Decision Tree Mining Model);
- математическая зависимость, построенная методом SVM (Support Vector Machine Mining Model);
- последовательности (Sequential Mining Model);
- модель сиквенциального анализа (Customer Sequential Mining Model);
- разделяемая кластерная модель (Partition Clustering Mining Model);
- центрированная кластерная модель (CDBased Clustering Mining Model);
- иерархическая кластерная модель (Hierarchical Clustering Mining Model).

После выбора строящейся модели необходимо выполнить: настройку процесса построения и алгоритм построения (рис. 1.7). Настройки процесса зависят от типа строящейся модели и выполняются на закладке **Settings** (Настройки).

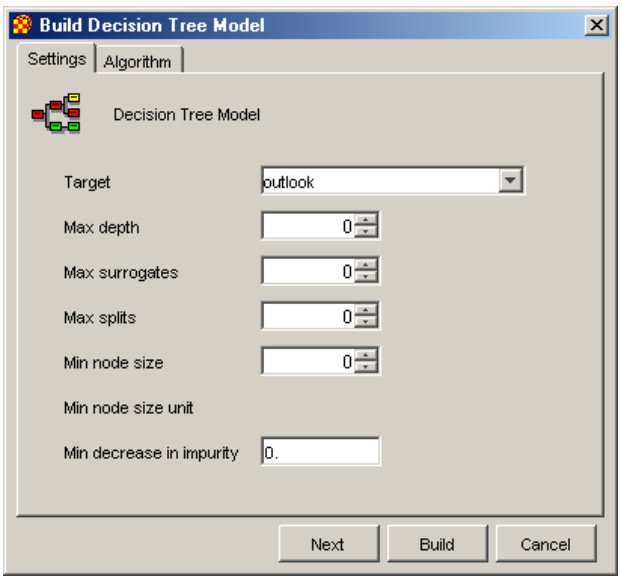


Рисунок 1.8. - Пример настроек для построения деревьев решений.

Выбор алгоритма выполняется на закладке **Algorithm** (алгоритм) (рис. 1.8.). Список доступных для построения модели алгоритмов зависит от типа модели. Кроме того, для некоторых алгоритмов необходимо выполнить дополнительную настройку. При их выборе в поле Algorithm Parameters появляются поля для определения специфичных для алгоритма настроек.

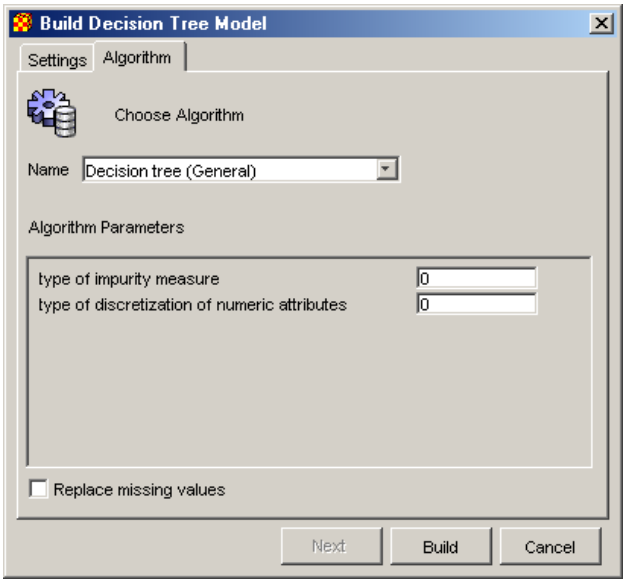


Рисунок 1.9. - Пример настроек алгоритма построения деревьев решений.

Для построения модели после выполнения настроек необходимо нажать на кнопку **Build** в диалоговом окне. После завершения построения модели появится диалоговое окно (рис. 1.9) предлагающее выполнить следующие действия:

- Визуализировать модель (Browse Model)
- Применить модель (Apply Model)
- Показать модель в виде PMML (View PMML Presentation)

- Записать модель в PMML формате (Save Model as PMML)

Для выполнения перечисленных действий необходимо выбрать соответствующую опцию и нажать на кнопку **Next**. Кроме того, после построения модели на панели инструментов становятся доступными соответствующие кнопки.

Рисунок 1.10. - Действия выполняемые с построенной моделью

В данной версии GUI Xelopes визуализируются только три вида моделей:

- Ассоциативные правила
- Деревья решений
- Иерархическая кластерная модель в виде дейтограмм.

Для остальных моделей при попытке визуализации происходит отображение модели в формате PMML. То есть для них действия **Browse Model** и **View PMML Presentation** будут иметь одинаковый результат.

Представление модели в формате PMML

Для представления модели в формате PMML необходимо нажать кнопку **View PMML Presentation** на панели инструментов или выбрать пункт меню **Model | View PMML** или выбрать опцию **View PMML Presentation** в диалоговом окне представленном на рис. 1.9. В результате будет открыто окно в котором будет представлена построенная модель в формате PMML в текстовом виде (1.10).

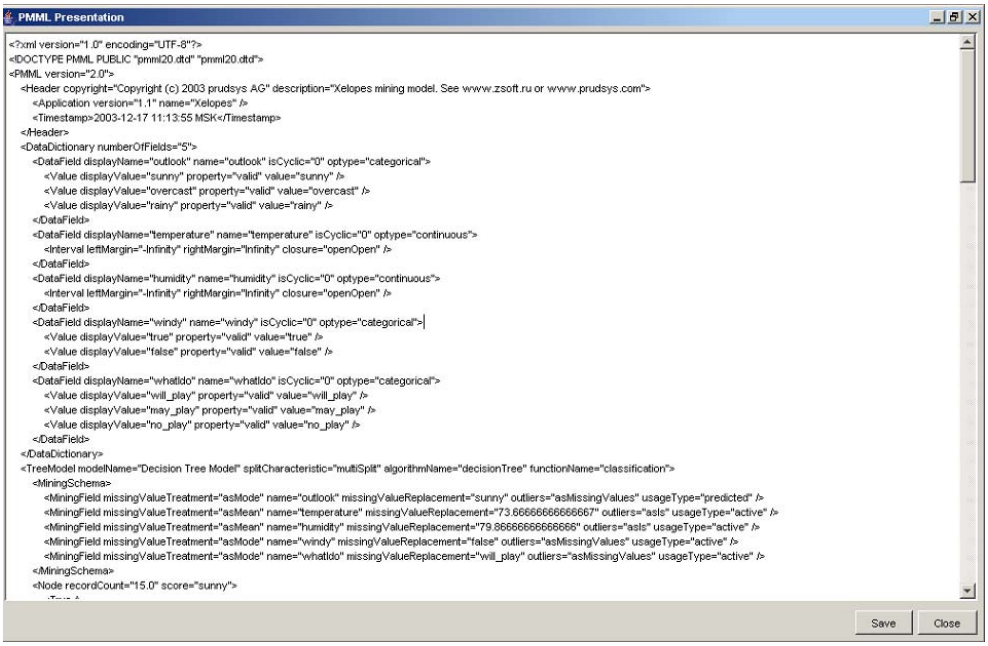
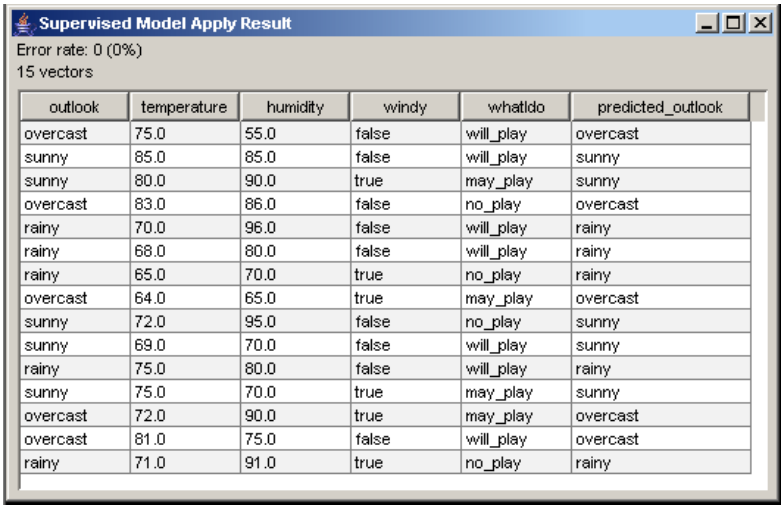


Рисунок 1.10. - Представление модели в PMML формате.

Представленную модель можно сохранить нажав в открытом окне кнопку Save. Кроме того, модель можно сохранить, нажав кнопку **Save Model as PMML** на панели инструментов или выбрав пункт меню **Model | Save** или опцию **Save Model as PMML** в диалоговом окне представленном на рис. 1.9.

Применение модели

Модели строящиеся для задач классификации и регрессии используются для предсказательных целей на новых данных. Следовательно они могут быть применены к другим данным. Для этого необходимо нажать кнопку **Apply Model** на панели инструментов или выбрав пункт меню **Model | Apply** или опцию **Apply Model** в диалоговом окне представленном на рис. 1.9. В результате будет предложено выбрать файл с новыми данными, записанными в формате arff (будет открыт диалоговое окно аналогичное представленному на рис. 1.1.). После выбора файла и применения построенной модели будет отображено окно в котором новые данные будут представлены в табличном виде (рис. 1.11).



outlook	temperature	humidity	windy	whatIdo	predicted_outlook
overcast	75.0	55.0	false	will_play	overcast
sunny	85.0	85.0	false	will_play	sunny
sunny	80.0	90.0	true	may_play	sunny
overcast	83.0	86.0	false	no_play	overcast
rainy	70.0	96.0	false	will_play	rainy
rainy	68.0	80.0	false	will_play	rainy
rainy	65.0	70.0	true	no_play	rainy
overcast	64.0	65.0	true	may_play	overcast
sunny	72.0	95.0	false	no_play	sunny
sunny	69.0	70.0	false	will_play	sunny
rainy	75.0	80.0	false	will_play	rainy
sunny	75.0	70.0	true	may_play	sunny
overcast	72.0	90.0	true	may_play	overcast
overcast	81.0	75.0	false	will_play	overcast
rainy	71.0	91.0	true	no_play	rainy

Рисунок 1.11. - Результат применения модели к новым данным.

В открывшемся окне в виде таблицы будут представлены классифицированные данные. Как видно таблица алогична той же представлена на рис. 1.2. Разница заключается в новой колонке **predicted_*** описывающей результат классификации (* - заменяется на атрибут классификации). В окне также выводится информация о степени ошибки классификации (Error rate).

Порядок выполнения работы

1. Открыть GUI интерфейс библиотеки Xelopes.
2. Загрузить исходные данные из файла transact.arff.
3. Просмотреть загруженные данные.
4. Просмотреть информацию об атрибутах данных.
5. Просмотреть статистическую информацию о данных.
6. Построить модель Association Rules Mining Model.
7. Визуализировать построенную модель.
8. Просмотреть и сохранить модель в формате PMML.
9. Выполнить пункты 2-5 для данных из файла weather-nominal.arff.
10. Выполнить пункты 6 - 8 для модели Decision Tree Mining Model.
11. Применить модель к данным из файла weather-nominal.arff
12. Выполнить пункты 6 - 8 для модели Hierarchical Clustering Mining Model.

Отчет по работе

1. Титульный лист.
2. Цель работы.
3. Данные из файлов transact.arff и weather.arff
4. Информация об атрибутах данных из файлов transact.arff и weather.arff
5. Статистическая информация по данным из файлов transact.arff и weather-nominal.arff

6. Скриншоты визуализации моделей Association Rules Mining Model, Decision Tree Mining Model и Hierarchical Clustering Mining Model.
7. Результат применения модели Decision Tree Mining Model к данным из файла weather-nominal.arff
8. Выводы по работе.

Контрольные вопросы

1. Какую статистическую информацию можно получить средствами GUI Xelopes.
2. Какие существуют типы атрибутов и их характеристики.
3. Какие mining модели можно построить средствами GUI Xelopes.
4. Какие существуют mining модели не реализованные в GUI Xelopes.
5. Какие действия можно выполнить с моделью.
6. Какие модели могут быть применены к другим данным и почему.

Лабораторная работа №2

Выполнение анализа данных методами data mining.

Цель и задача работы

Изучить основные этапы интеллектуального анализа данных с использованием алгоритмов data mining реализованных в библиотеке Xelopes.

Для данных из файла определенных вариантом задания построить модели также в соответствии с вариантом задания с помощью различных алгоритмов и объяснить результаты.

Теоретические положения

Mining модели

Процесс интеллектуального анализа данных состоит из следующих основных этапов:

1. Подготовка данных в виде удобном для применения методов data mining
2. Настройка процесса построения mining модели
3. Построение mining модели
4. Анализ построенной mining модели
5. В случае supervised модели применение ее к новым данным

2.2. Подготовка исходных данных

Процесс подготовки предполагает сбор данных для анализа из разных источников данных и представления их в формате пригодном для применения алгоритмов data mining.

Настоящая версия Xelopes поддерживает ARFF (Attribute-Relation File Format) формат представления данных. Он разработан для библиотеки Weka в университете Waikato. ARFF файл является ASCII текстовым файлом, описывающем список объектов с общими атрибутами.

Структурно такой файл разделяется на две части: заголовок и данные.

В заголовке описывается имя данных и их метаданные (имена атрибутов и их типы). Например,

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {true, false}
@attribute whatIdo {will_play, may_play, no_play}
```

Во второй части представлены сами данные. Например,

```
@data
overcast,75,55,false,will_play
sunny,85,85,false,will_play
sunny,80,90,true,may_play
overcast,83,86,false,no_play
rainy,70,96,false,will_play
rainy,68,80,false,will_play
rainy,65,70,true,no_play
overcast,64,65,true,may_play
sunny,72,95,false,no_play
sunny,69,70,false,will_play
rainy,75,80,false,will_play
sunny,75,70,true,may_play
overcast,72,90,true,may_play
overcast,81,75,false,will_play
rainy,71,91,true,no_play
```

2.3. Заголовок

Заголовок содержит информацию об имени файла и метаданные о представленных в нем данных. Имя описывается в следующем формате

```
@relation <имя>
```

Имя может быть любая последовательность символов. Если имя включает пробелы то оно должно быть заключено в кавычки. Например

```
@relation weather
@relation "weather nominal"
```

Мета данные описывают атрибуты представленных в файле данных. Информация о каждом атрибуте записывается в отдельной строке и включает в себя имя атрибута и его тип. Очевидно, что имена должны быть уникальны. Порядок их описания должен совпадать с порядком колонок описания данных. Общий формат описания атрибута следующий:

```
@attribute <имя атрибута> <тип атрибута>
```

Например,

```
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
```

Имя атрибута должно начинаться с символа. В случае если оно содержит пробелы, то должно быть заключено в кавычки.

Значением поля <тип> может быть одно из следующих пяти типов:

- real
- integer
- <категория>
- string
- date [<формат даты>]

Типы real и integer являются числовыми. Категориальные типы описываются перечислением категорий (возможных значений). Например:

```
@attribute outlook {sunny, overcast, rainy}
```

При описании даты можно указать формат в котором она будет записываться (например, "yyyy-MM-dd").

2.4. Данные

Данные представляются в ARFF формате в виде списка значений атрибутов объектов после тэга `@data`. Каждая строка списка соответствует одному объекту. Каждая колонка соответствует атрибуту описанному в части заголовка. Причем порядок следования колонок должен совпадать с порядком описания атрибутов. Например:

```
@data
overcast,75,55,false,will_play
sunny,85,85,false,will_play
sunny,80,90,true,may_play
```

Часто в терминологии data mining такие строки называют векторами.

Данные могут содержать пропущенные (неизвестные) значения. В ARFF они представляются символом «?», например:

```
@data
4.4,?,1.5,?,Iris-setosa
```

Строковые данные в случае если они содержат разделяющие слова символы, должны заключаться в кавычки. Например,

```
@relation LCCvsLCSH

@attribute LCC string
@attribute LCSH string

@data
AG5, 'Encyclopedias and dictionaries.;Twentieth century.'
AS262, 'Science -- Soviet Union -- History.'
AE5, 'Encyclopedias and dictionaries.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Phases.'
AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Tables.'
```

Даты также должны быть заключены в кавычки. Если при описании соответствующего атрибута бы указан формат даты, то данные должны быть записаны в соответствии с ним:

```
@relation Timestamps

@attribute timestamp DATE "yyyy-MM-dd HH:mm:ss"

@data
"2001-04-03 12:12:12"
"2001-05-03 12:59:55"
```

2.5. Настройка процесса построения mining модели

Результатом анализа данных с помощью методов data mining являются структуры представляющие собой новые знания. Такие структуры называются моделями. Они могут быть разных видов: правила классификации, ассоциативные правила, деревья решений, математические зависимости. Вид модели во многом зависит от метода с помощью которого она была построена. Таким образом, конечный результат зависит от метода и исходных данных. Кроме того, процесс построения моделей можно настроить изменяя тем самым свойства модели (точность, глубина дерева и т.п.). Настраиваемые параметры зависят от конкретной модели.

В GUI Xelopes пользователь имеет возможность выполнить настройки для каждой строящейся модели индивидуально. Этот процесс

осуществляется в диалоговом окне настроек, описанном в предыдущей лабораторной работе на закладке Settings. Далее рассмотрим более подробно настройки для каждой модели.

2.6. Настройки для ассоциативных правил и сиквенциального анализа

Настройки для модели представляющей ассоциативные правила выполняются в диалоговом окне изображенном на рис. 2.1.

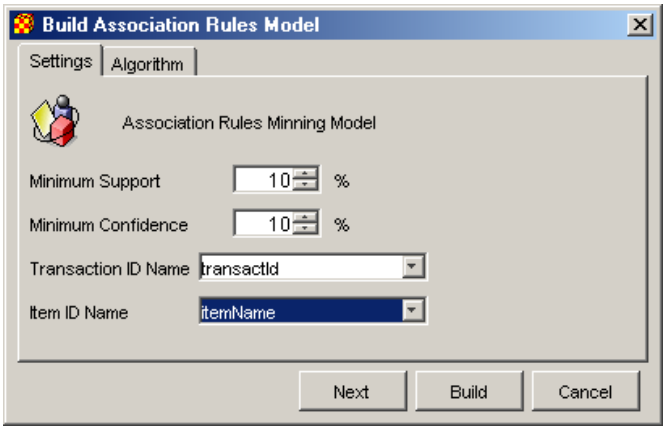


Рисунок 2.1. - Настройки модели ассоциативных правил

В нем выполняется настройка следующих параметров:

- **Minimum Support** – минимальное значение поддержки для искомых частых наборов и строящихся ассоциативных правил. Значение должно быть больше нуля, иначе не будет построено не одного правила.
- **Minimum Confidence** – минимальное значение доверия для строящихся ассоциативных правил. Значение должно быть больше нуля, иначе не будет построено не одного правила.
- **Transaction ID Name** – атрибут уникально идентифицирующий транзакции (ключевое поле).
- **Item ID Name** – атрибут представляющий собой имена объектов. Они используются для построения правил. От его выбора зависит степень понимания полученных результатов.

Настройки для сиквенциальной модели выполняются в диалоговом окне изображенном на рис. 2.2.

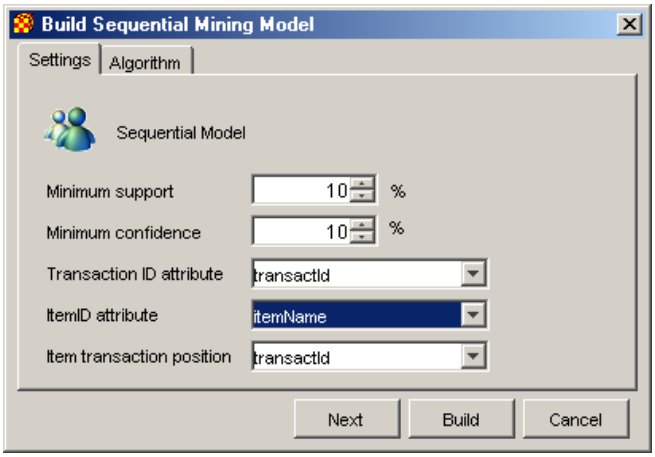


Рисунок 2.2. - Настройки сиквенциальной модели

В нем выполняется настройка аналогичные модели ассоциативных правил. Дополнительно появляется параметр **Item transaction position** представляющий атрибут, идентифицирующий позицию элемента в последовательности.

2.7. Настройки для деревьев решений (Decision Tree Mining Model)

Настройки для модели представляющей деревья решений выполняются в диалоговом окне изображенном на рис. 2.3.

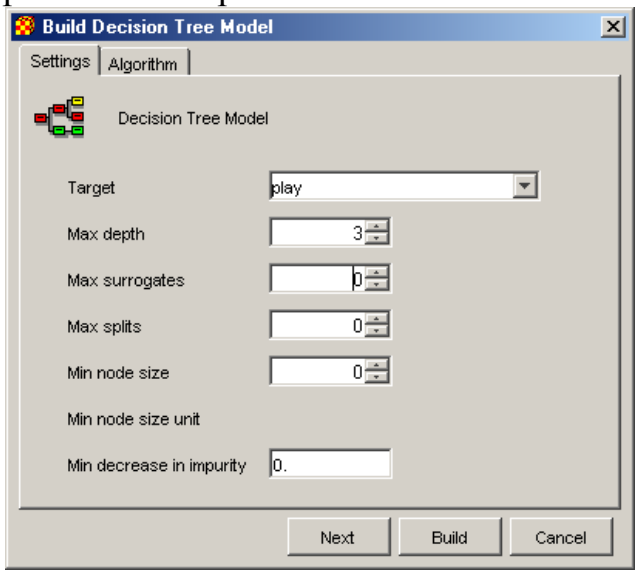


Рисунок 2.3. - Настройки модели деревьев решений

В нем выполняется настройка следующих параметров:

- **Target** – атрибут по которому выполняется классификация данных (независимая переменная).
- **Max depth** – максимально допустимая глубина строящегося дерева
- **Max surrogates** - максимально допустимое число замен
- **Max splits** - максимально допустимое количество расщеплений
- **Min node size** – минимальный размер узла дерева
- **Min decrease in impurity** – минимальная степень примесей

2.8. Настройки для математической зависимости построенной методом SVM

Настройки для модели представляющую математическую зависимость, построенную методом SVM, выполняются в диалоговом окне изображенном на рис. 2.4.

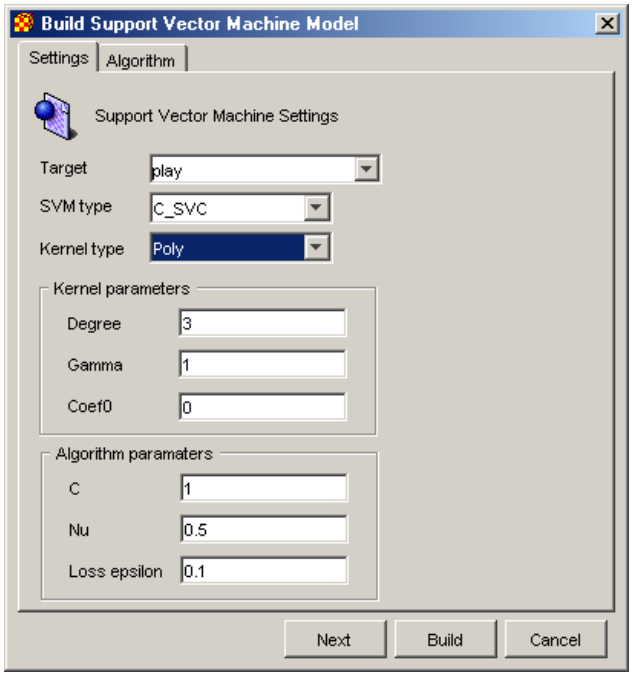


Рисунок 2.4. - Настройки модели SVM

В нем выполняется настройка следующих параметров:

- **Target** – атрибут по которому выполняется классификация данных (независимая переменная).
- **SVM Type** – тип модели SVM. В Xelopes могут быть построены следующие типы: C-SVC (classical SVM), Nu-SVC, one-class SCM, Epsilon-SVR (classic regression SVM), Nu-SVR. Они отличаются классификационнй функции. Так наиболее распространенная SVM для задачи регрессии Epsilon-SVR имеет функцию вида:

$$f(x,\alpha,\alpha^*) = \sum_{i=1}^{M}(\alpha_i^* - \alpha_i)K(x,x_i) + b$$

в то время как SVM для классификации имеет вид

$$f(x,\alpha) = \sum_{i=1}^{M}\alpha_i \; K(x,x_i) + b$$

- **Kernel Type** – вид функции $K(x, x_i)$ в классификационной функции (тип ядра). Может принимать следующие значения:
  - **Linear** - Линейная - $k(x,y) = x*y$
  - **Poly** - Полиномиал степени d - $k(x,y) = (\gamma * x*y+c_0)^d$
  - **RBF**- Базовая радиальная функция Гаусса - $k(x,y) = exp(-\gamma|| x - y||)$
  - **Sigmoid** - Сигмоидальная $k(x,y) = tanh(\gamma * x*y+c_0)$
- **Kernel Parameters** – параметры ядра, зависят от выбранного типа ядра.

- Degree – степень d в ядре poly;
- Gamma – параметр γ в последних трех видах;
- Coef0 – коэффициент c_0 в типах poly и sigmoid.

- **Algorithm Parameters** – общие параметры алгоритмов класса SVM:
  - C – инверсный регулирующий параметр $C = \frac{1}{2\lambda M}$;
  - Nu – параметр ν в типе Nu – SVM;
  - Loss epsilon – ε функция потерь в типе Epsilon-SVR.

2.9. Настройки кластерной модели

Настройки для кластерных центрированной и иерархических моделей выполняются в диалоговом окне изображенном на рис. 2.5.

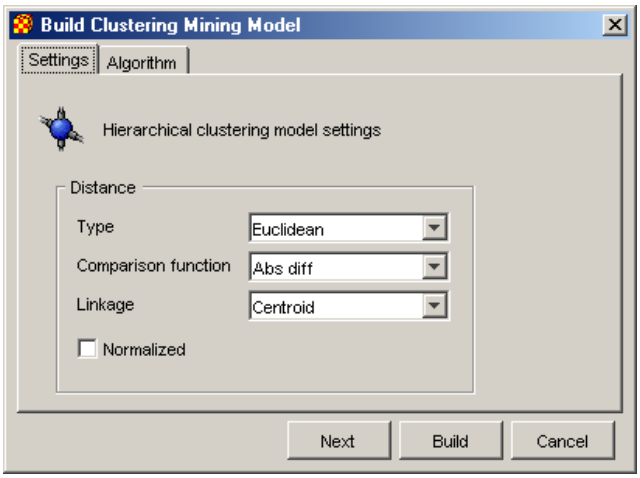


Рисунок 2.5. - Настройки для кластерной модели.

В нем выполняется настройка следующих параметров:

- **Maximum number of** clusters – максимальное количество построенных кластеров. Знчение параметра должно быть больше нуля.
- **Distance** – параметры характеризующие функцию вычисления расстояния между объетами:
- **Type** – тип функции расстояния. Xelopes (Евклидово – Euclidean, Чебышева – Chebyshev и др.)
- **Comparison function** – функция сопоставления.
- **Normalized** – использовать ли нормализацию при расчете расстояний.

Настройки для разделяемой кластерной модели выполняются в диалоговом окне изображенном на рис. 2.6.

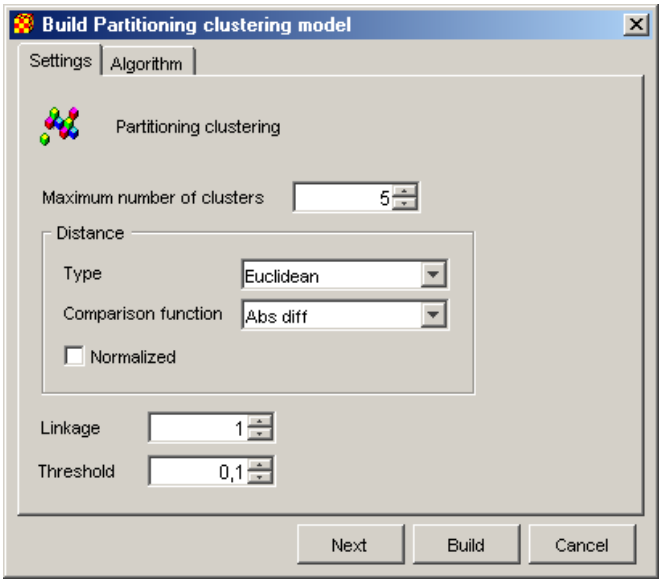


Рисунок 2.6. - Настройки для разделяемой кластерной модели.

В нем выполняется настройка дополнительных параметров параметров:

- **Linkage** – параметр k для алгоритма k-linkage.
- **Threshold** – предел для расстояния.

2.10. Анализ моделей

Для применения полученных с помощью методов data mining знаний необходимо проанализировать построенные модели. При анализе необходимо проверить насколько полученные знания являются логически объяснимыми, не противоречат ли они здравому смыслу, действительно ли они являются новыми и т.п. Кроме того, модели, строящиеся при решении задач ассоциативного анализа и кластеризации, являются описательными, т.е. служат для лучшего понимания самих данных. В связи с этим можно сделать вывод, что важным является представление моделей в виде удобном для их анализа человеком.

В GUI Xelopes любая модель может быть представлена в формате PMML. Это стандартизированных формат основанный на формате XML. К сожалению для визуального анализа данный формат довольно сложен. По этой причине в GUI Xelopes реализованы специальные средства визуализации для трех основных видов моделей:

- ассоциативные правила;
- деревья решений;
- дейтограммы.

2.11. Визуализация ассоциативных правил

Модель представляющая ассоциативные правила в GUI Xelopes представляется в виде 3-х мерных гистограмм (рис. 2.7.). По осям плоскости откладываются подмножества частых наборов. LHS – означает левую часть правил, RHS – правую. На их пересечении рисуется гистограмма. По

умолчанию высота гистограмма отражает уровень поддержки правила включающего в условную и заключительную части данные наборы. Цвет от синего к красному (от меньшего к большему) уровень доверия.

Например, для правила *Если (Nut) то (Coke)* нарисована красная высокая гистограмма означающая что данное правило имеет наибольшую степень поддержки с высокой степенью доверия. Правило Если (Water) то (Cracker, Coke) имеет низкий уровень поддержки и низкую степень доверия.

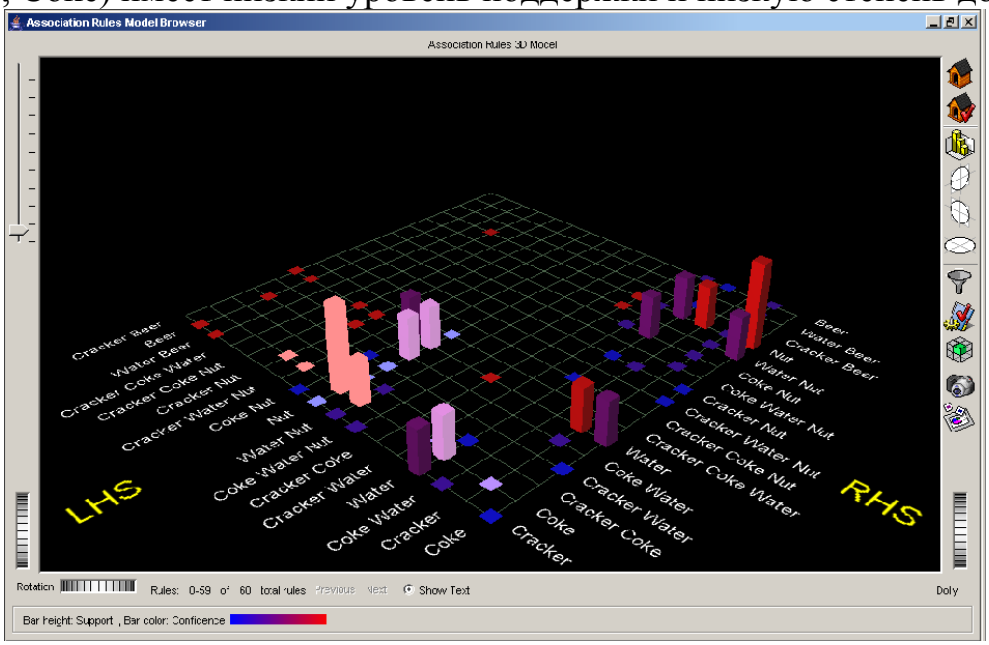


Рисунок 2.7. - Визуальное представление ассоциативных правил.

Для более детального изучения правил необходимо выделить гистограмму на пересечении интересующих наборов. Визуально они подсвечиваются более ярким цветом. На рис. Выделена гистограмма Nut – Coke. Для выделенных наборов можно более детально посмотреть гистограммы их оценок. Для этого необходимо на панели инструментов находящейся слева от диаграмма нажать кнопку  . В результате появится диалог изображенный на рис. 2.8.

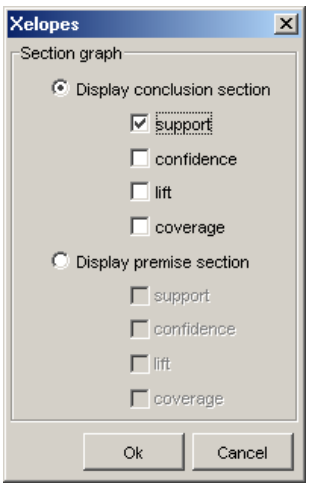


Рисунок 2.8. - Диалог для детализации оценок ассоциативных правил.

В нем можно выбрать какие оценки должны быть детализированы. После нажатие на кнопку ОК, появятся диаграммы (Рис. 2.9) оценок для выделенных наборов.

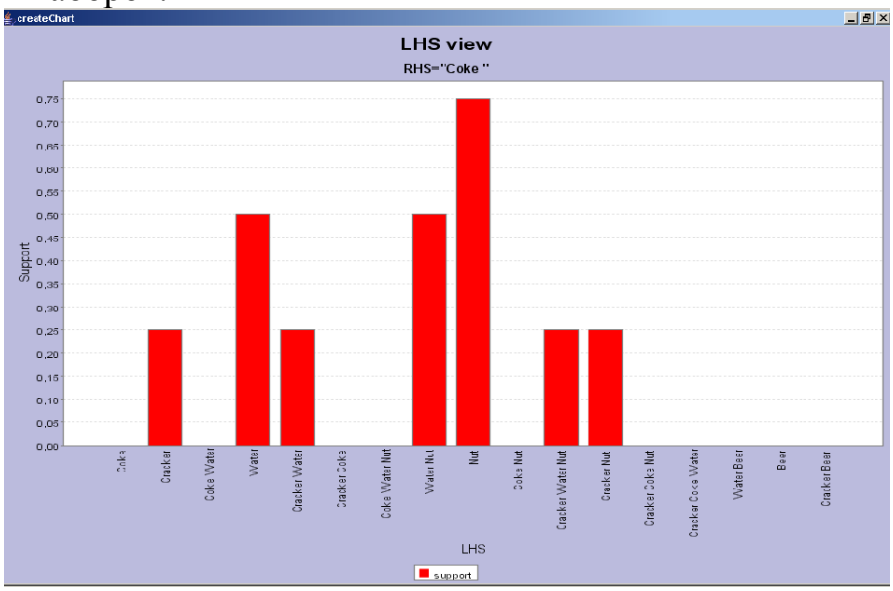


Рисунок 2.9. - Пример диаграммы поддержки

2.12. Визуализация деревьев решений

Модель представляющая деревья решений в GUI Xelopes представляется в дерева (рис. 2.10.). Узлами дерева являются выражения определяющие разбиения множества объектов на подмножества. Нижняя часть в каждом узле отображает уровень вхождения в множество соответствующее узлу объектов относящихся к разным классам. Можно заметить, что листья дерева соответствующие подмножествам содержащим объекты одного класса имеют одноцветную нижнюю часть. Подписи на ветвях дерева отражают условия перехода по этой ветви.

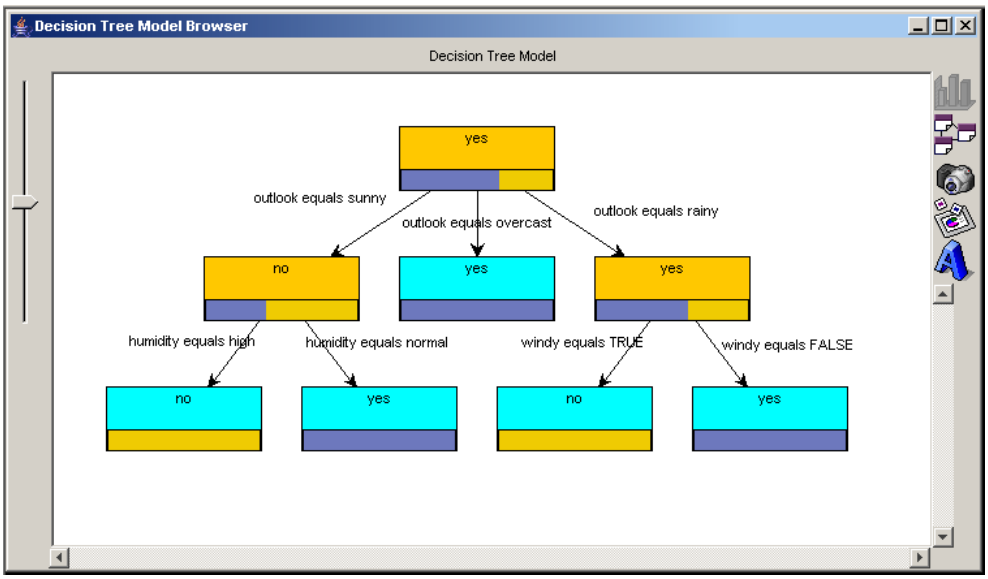


Рисунок 2.10. - Пример визуализации модели дерева решений

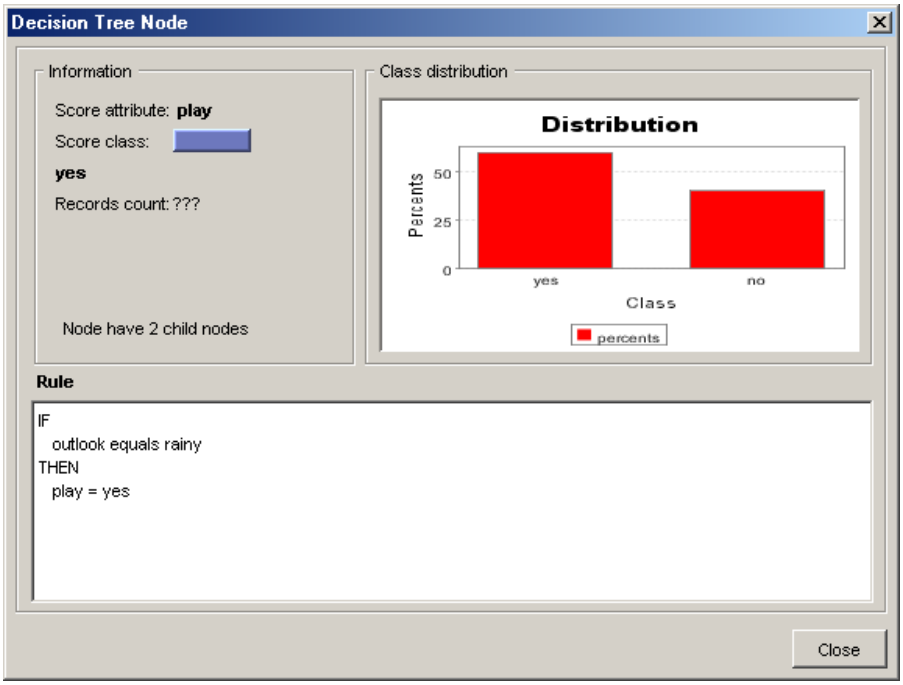


Рисунок 2.11. - Пример визуализации модели дерева решений

По каждому узлу дерева можно получить дополнительную информацию. Для этого необходимо выделить узел и или выбрав в контекстном меню пункт **Node Information** или нажать на кнопку **Node Info** на панели инструментов слева от диаграммы. В результате появится окно (рис. 2.11) представляющую следующую информацию об узле:

- **Information** – информация об узле:
  - **Score attribute** – сравниваемый атрибут (зависимая переменна);
  - **Score class** – значение с которым выполняется сравнение;
  - **Records count** – количество объектов покрывемых узлом;
  - Количество ветвей выходящих из узла.
- **Class distribution** – распределение объектов относящихся к разным классам для данного узла;
- **Rule** – классификационное правило соответствующее данному узлу.

2.13. Визуализация иерархической кластеризации

Модель представляющая иерархическую кластеризацию решений в GUI Xelopes представляется в виде дейтограммы (рис. 2.12.). Верхний узел представляет собой кластер соответствующий всему множеству объектов. Листья соответствуют кластерам содержащим по одному элементу из исходного множества.

С помощью мыши можно задать уровень кластеризации. На дейтаграмме он представляется в виде линии. При этом будет выводиться информация о среднем расстоянии между кластерами. Объединяемые кластеры при заданном уровне выделяются красным цветом.

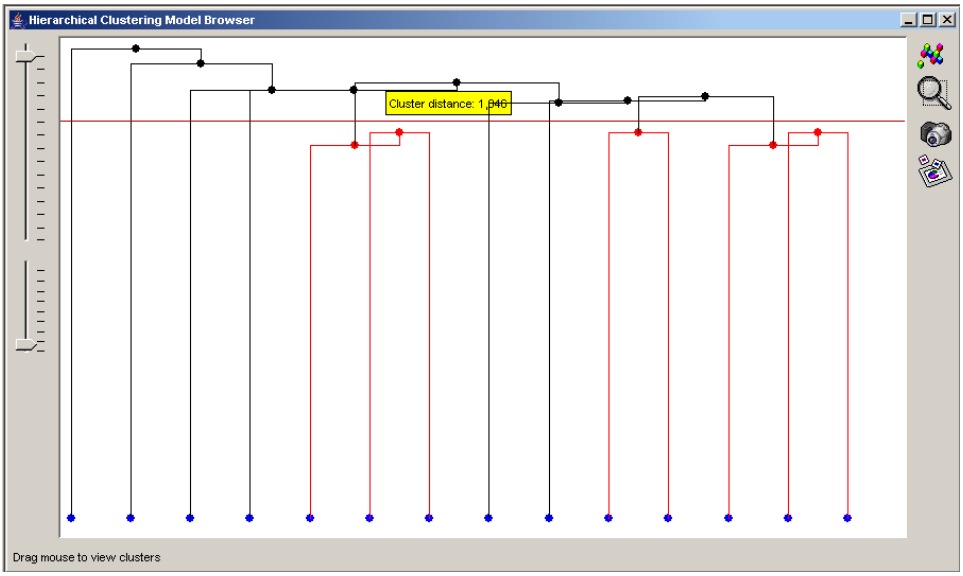


Рисунок 2.12. - Пример визуализации дейтограммы

По кластеризации можно получить более детальную информацию, нажав на кнопку  на панели инструментов слева от диаграммы. В результате появится окно представленное на рис. 2.13.

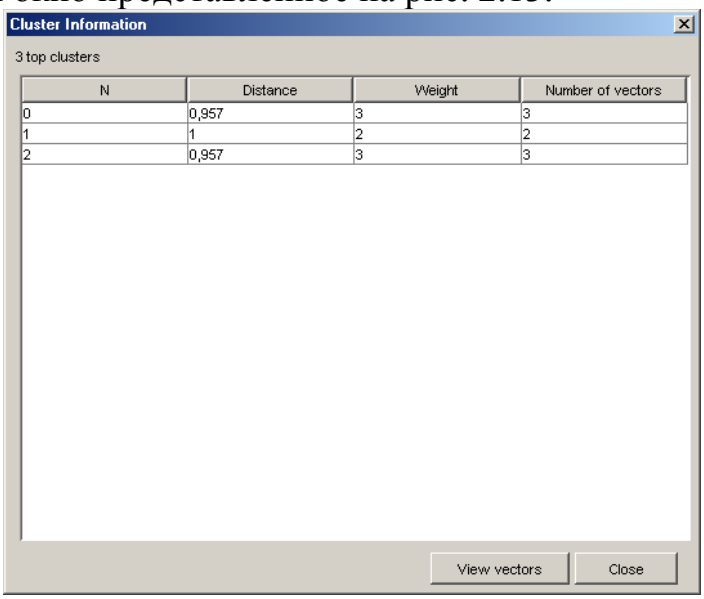


Рисунок 2.13. - Детализированная информация о кластерах.

В окне в табличном виде отображается информация о кластерах для заданного уровня. Над таблицей отображается информация о количестве кластеров. Колонки в таблице содержат следующую информацию:

- **N** – номер кластеров
- **Distance** – расстояние между кластерами.
- **Weight** – вес кластера (в данном случае количество объектов попавших в кластер)
- **Number of vectors** - количество объектов попавших в кластер.

Нажав на кнопку View vectors можно просмотреть информацию об исходных данных.

Порядок выполнения работы

1. Подготовить новые данные в формате ARFF имеющую ту же структуру, что и данные в файле заданном вариантом задания без значений независимой переменной.
2. Открыть GUI интерфейс библиотеки Xelopes.
3. Загрузить исходные данные из файла указанного в варианте задания.
4. Просмотреть загруженные данные.
5. Просмотреть информацию об атрибутах данных.
6. Просмотреть статистическую информацию о данных.
7. Поочередно попытаться построить модели определенные вариантом задания каждым из доступных алгоритмов для разных параметров настройки.
8. Визуализировать и сравнить модели, построенные разными алгоритмами.
9. Просмотреть и сохранить построенные модели в формате PMML.
10. Применить модели типа supervised к данным, подготовленным на шаге 1.

Варианты заданий

Вар-т	Файл	Модели
1	contact-lenses.arff	Sequential Mining Model, Decision Tree Mining Model, Hierarchical Clustering Mining Model
2	weather.arff	Customer Sequential Mining Model Support Vector Machine Mining Model, CDBased Clustering Mining Model
3	weather-nominal.arff	Association Rules Mining Model Decision Tree Mining Model Hierarchical Clustering Mining Model
4	iris.arff	Association Rules Mining Model Support Vector Machine Mining Model Partition Clustering Mining Model
5	iris-transact.arff	Customer Sequential Mining Model, Support Vector Machine Mining Model, Hierarchical Clustering Mining Model
6	iris-nontransact.arff	Association Rules Mining Model Decision Tree Mining Model Partition Clustering Mining Model
7	transact.arff	Association Rules Mining Model, Support Vector Machine Mining Model CDBased Clustering Mining Model

8	custom-transact.arff	Association Rules Mining Model Decision Tree Mining Model CDBased Clustering Mining Model
9	iris.arff	Customer Sequential Mining Model Support Vector Machine Mining Model, CDBased Clustering Mining Model
10	iris-transact.arff	Association Rules Mining Model Decision Tree Mining Model Hierarchical Clustering Mining Model
11	iris-nontransact.arff	Association Rules Mining Model Support Vector Machine Mining Model Partition Clustering Mining Model
12	transact.arff	Customer Sequential Mining Model, Support Vector Machine Mining Model, Hierarchical Clustering Mining Model
13	contact-lenses.arff	Customer Sequential Mining Model, Support Vector Machine Mining Model, Hierarchical Clustering Mining Model
14	weather.arff	Association Rules Mining Model Decision Tree Mining Model Partition Clustering Mining Model
15	weather-nominal.arff	Association Rules Mining Model, Support Vector Machine Mining Model CDBased Clustering Mining Model
16	iris.arff	Association Rules Mining Model Decision Tree Mining Model CDBased Clustering Mining Model
17	iris.arff	Association Rules Mining Model Decision Tree Mining Model Hierarchical Clustering Mining Model
18	iris-transact.arff	Association Rules Mining Model Support Vector Machine Mining Model Partition Clustering Mining Model
19	iris-nontransact.arff	Customer Sequential Mining Model, Support Vector Machine Mining Model, Hierarchical Clustering Mining Model
20	transact.arff	Customer Sequential Mining Model, Support Vector Machine Mining Model, Hierarchical Clustering Mining Model

Отчет по работе

1. Титульный лист.
2. Цель работы.
3. Данные из файла определенного вариантом задания и информация о них.
4. Список моделей, которые не удалось построить для данных с пояснениями почему.
5. Для каждой модели список алгоритмов, которые не построили модель для данных с пояснениями почему.
6. Каждую модель, построенную разными алгоритмами с описанием различий между ними и пояснениями.
7. Модели, построенные одним алгоритмом при разных параметрах настройки с описанием различий и пояснениями.
8. Результат применения модели типа supervised к новым данным.
9. Выводы по работе.

Контрольные вопросы

1. Какие проблемы возникают с исходными данными.
2. Почему для одних и тех же данных не могут быть построены все виды моделей.
3. Какие требования на исходные данные накладывают разные алгоритмы data mining.
4. Какие параметры необходимо настроить для построения ассоциативных правил. Как от них зависит результат (построенная модель).
5. Какие параметры необходимо настроить для построения дерева решений. Как от них зависит результат (построенная модель).
6. Какие параметры необходимо настроить для выполнения кластеризации. Как от них зависит результат (построенная модель).
7. Какие параметры определяются алгоритмами. Привести примеры. Как от них зависит результат (построенная модель).
8. Что такое сиквенциальный анализ и чем он отличается от поиска ассоциативных правил.

Лабораторная работа №3

Создание программ анализа данных с использованием алгоритмов data mining

Цель и задача работы

Изучить основные принципы создания систем интеллектуального анализа данных с использованием алгоритмов data mining.

Реализовать программу, выполняющую анализ данных представленных в формате ARFF с помощью алгоритма data mining и строящую модель, заданную вариантом задания.

Теоретические положения

Mining в СППР

Системы поддержки принятия решений (СППР) используют методы data mining для анализа данных и получения новых знаний. Полученные результаты могут носить как описательный характер, позволяющие лучше понять данные, так и предсказательный, позволяющие в дальнейшем предсказывать значение каких либо параметров на основании найденных закономерностей. К первому виду относятся методы выполняющие поиск ассоциативных правил и кластеризацию. Ко второму методы строящие функции классификации и регрессии.

Библиотека Xelopes в своем составе имеет алгоритмы обоих типов. Она может служить основой для построения СППР. Структура и поддержка основных стандартов библиотекой делают ее использование достаточно простым.

Основной подход решения задач data mining с помощью библиотеки Xelopes не зависит от вида или используемого метода.

Первоначально создается экземпляр класс `MiningInputStream` для загрузки исходных данных, например, из файла формата ARFF.

```
MiningInputStream inputData = new MiningArffStream("data.arff");
```

Затем выделяются метаданные загруженных данных.

```
MiningDataSpecification metaData = inputData.getMetaData();
```

Далее создается экземпляр класса `MiningAlgorithm`. Для настройки процесса построения модели создаются экземпляры классов `MiningSettings` и `MiningAlgorithmSpecification`. У данных экземпляров устанавливаются необходимые параметры. Создание конкретных экземпляров классов зависят от используемого метода и решаемой задачи.

Сделанные настройки должны быть проверены с помощью метода: `verifySettings()`:

```
miningSettings.verifySettings();
```

Необходимо заметить, что настройки специфичные для алгоритма могут быть выполнены как непосредственно в коде, так и загружены из конфигурационного файла `algorithms.xml` по имени алгоритма.

Созданному экземпляру алгоритма передаются исходные данные и настройки.

```
algorithm.setMiningInputStream( inputData );
algorithm.setMiningSettings( miningSettings );
algorithm.setMiningAlgorithmSpecification(
                            miningAlgorithmSpecification );
```

Затем вызывается метод `buildModel()` построения модели, который возвращает построенную модель в виде экземпляра класса `MiningModel`.

```
MiningModel model = algorithm.buildModel();
```

Любая построенная модель может быть сохранена в формате PMML

```
FileWriter writer = new FileWriter("example.xml");
model.writePmml(writer);
```

или в текстовом формате

```
writer = new FileWriter("example.txt");
model.writePlainText(writer);
```

Если построенная модель является экземпляром класса `SupervisedMiningModel`, то следовательно она может быть применена к новы данным с цель определения значения зависимой переменной.

```
MiningVector vector = inputData.read();
double predicted = model.applyModelFunction(vector);
```

Далее рассмотрим особенности решения задач поиска ассоциативных правил, кластеризации и классификации.

Поиск ассоциативных правил

Для настройки процесса поиска ассоциативных правил создается экземпляр класса `AssociationRulesSettings`.

```
AssociationRulesSettings miningSettings = new AssociationRulesSettings();
```

Ему передаются метаданные загруженных данных:

```
miningSettings.setDataSpecification( metaData );
```

Для поиска ассоциативных правил важно указать какие из атрибутов идентифицируют элементы, а какие транзакции, в которые они входят. Например,

```
CategoricalAttribute categoryItemId =
      (CategoricalAttribute) metaData.getMiningAttribute("itemId");
miningSettings.setItemId( categoryItemId );

CategoricalAttribute categoryTransactId =
    (CategoricalAttribute)metaData.getMiningAttribute("transactId");
miningSettings.setTransactionId( categoryTransactId );
```

Также для поиска необходимо определить минимальную поддержку и минимальную степень доверия для искомых правил. Например,

```
miningSettings.setMinimumConfidence( 0.30 );
miningSettings.setMinimumSupport( 0.5 );
```

Для решения задачи поиска ассоциативных правил можно использовать алгоритм Apriori. В библиотеке Xelopes он реализован в классе `com.prudsys.pdm.Models.AssociationRules.Algorithms.AprioriSimple.Apriori`. Для его использования необходимо сделать специфичные для него настройки. Для этого создается экземпляр *класса* `MiningAlgorithmSpecification`:

```
MiningAlgorithmSpecification miningAlgorithmSpecification = new
                        MiningAlgorithmSpecification();
```

Указываются имя, функция, используемый алгоритм, версия алгоритма и класс его реализующий. Например:

```
miningAlgorithmSpecification.setName("AprioriSimple");
miningAlgorithmSpecification.setFunction("AssociationRules");
miningAlgorithmSpecification.setAlgorithm("associationRules");
miningAlgorithmSpecification.setClassname("com.prudsys.pdm.Models.Association
             Rules.Algorithms.AprioriSimple.Apriori");
miningAlgorithmSpecification.setVersion("1.0");
```

Далее устанавливается дополнительные параметры с помощью массива экземпляров класса `MiningAlgorithmParameter`:

```
MiningAlgorithmParameter[] miningAlgorithmParameter =
                       new MiningAlgorithmParameter[3];
```

Устанавливается минимальный размер набора. Например,

```
miningAlgorithmParameter[0] = new MiningAlgorithmParameter();
miningAlgorithmParameter[0].setName("minimumItemSize");
miningAlgorithmParameter[0].setType("int");
miningAlgorithmParameter[0].setValue("1");
miningAlgorithmParameter[0].setMethod("setMinimumItemSize");
miningAlgorithmParameter[0].setDescr("Minimum size for large items");
```

Устанавливается минимальный размер набора. Например,

```
miningAlgorithmParameter[1] = new MiningAlgorithmParameter();
miningAlgorithmParameter[1].setName("maximumItemSize");
miningAlgorithmParameter[1].setType("int");
miningAlgorithmParameter[1].setValue("-1");
miningAlgorithmParameter[1].setMethod("setMaximumItemSize");
miningAlgorithmParameter[1].setDescr("Maximum size for large items");
```

Устанавливается параметр позволяющий генерировать ассоциативные правила, а не только частые наборы. Например,

```
miningAlgorithmParameter[2] = new MiningAlgorithmParameter();
miningAlgorithmParameter[2].setName("generateRules");
miningAlgorithmParameter[2].setType("boolean");
miningAlgorithmParameter[2].setValue("true");
miningAlgorithmParameter[2].setMethod("setGenerateRules");
miningAlgorithmParameter[2].setDescr("Allow to generate
association rules");
```

Все дополнительные параметры должны быть добавлены в спецификацию алгоритма:

```
miningAlgorithmSpecification.setInputAttribute(
                                    miningAlgorithmParameter );
```

Далее должен быть создан экземпляр алгоритма выполняющего построение модели ассоциативных правил:

```
String className = miningAlgorithmSpecification.getClassname();
if( className == null )
      throw new MiningException( "className attribute expected." );

Class algorithmClass = = Class.forName( className );
Object algorithm = algorithmClass.newInstance();
AssociationRulesAlgorithm miningAlgorithm =
                        (AssociationRulesAlgorithm)algorithm;
```

После создания необходимо вызвать метод метод `buildModel()` для построения модели класса `AssociationRulesMiningModel`.

```
AssociationRulesMiningModel model =
             (AssociationRulesMiningModel) algorithm.buildModel();
```

Обработать построенную модель, например, для вывода ассоциативных правил в консоль можно следующим образом.

Получить список ассоциативных правил и частых наборов:

```
Vector rules = ruleModel.getAssociationRules();
Vector LITS  = ruleModel.getLargeItemSets();
```

Получить атрибуты идентифицирующие элементы и транзакции:

```
CategoricalAttribute itemId =
              (CategoricalAttribute)((AssociationRulesSettings)ruleModel.getM
              iningSettings()).getItemId();
```

```
CategoricalAttribute transactId =
              (CategoricalAttribute)((AssociationRulesSettings)
              ruleModel.getMiningSettings()).getTransactionId();
```

Определить количество правил, частых наборов и транзакций

```
int nLITS  = LITS.size();
int nRules = rules.size();
int itemNumber = itemId.getCategoriesNumber();
int transactsNumber = transactId.getCategoriesNumber();
```

Вывести все ассоциативные правила

```
System.out.println();
System.out.println("Number of association rules found: " + nRules);
for (int i = 0; i < nRules; i++) {
          // Новое правило:
        System.out.print(i + ": ");

        // Показать правило:
        RuleSet rs   = (RuleSet) rules.elementAt(i);
        int itemSize = rs.getSize();

        // условная часть правила:
        ItemSet is   = rs.getPremise();
        int nprem    = rs.getPremise().getSize();
        for (int j = 0; j < nprem; j++) {
          int pN        = is.getItemAt(j);
          Category cat  = (Category) itemId.getCategory(pN);
          System.out.print(cat.getValue() + " ");
        };
        System.out.print("=> ");

        // Заключительная часть правила:
        for (int j = nprem; j < itemSize; j++) {
          int pN        = rs.getConclusion().getItemAt(j-nprem);
          Category cat  = (Category) itemId.getCategory(pN);
          System.out.print(cat.getValue() + " ");
}
```

Задача кластеризации

Для настройки процесса кластеризации создается экземпляр класса `ClusteringSettings`.

```
  ClusteringSettings miningSettings = new ClusteringSettings();
```

Ему передаются метаданные загруженных данных:

```
     miningSettings.setDataSpecification( metaData );
```

Для кластеризации можно указать имя атрибута определяющего идентификацию кластера. Например,

```
miningSettings.setClusterIdAttributeName("ItemID");
```

Для решения задачи кластеризации можно использовать алгоритм KMeans. В библиотеке Xelopes он реализован в классе

`com.prudsys.pdm.Models.Clustering.CDBased.Algorithms.KMeans.KMeans`. Для его использования необходимо выполнить настройку специфичных параметров. Покажем как это можно сделать с помощью загрузки из файла algorithms.xml.

```
MiningAlgorithmSpecification miningAlgorithmSpecification =
                  MiningAlgorithmSpecification.getMiningAlgorithmSpecificatio
                  n("KMeans");
```

Далее должен быть создан экземпляр алгоритма выполняющего кластеризацию:

```
String className = miningAlgorithmSpecification.getClassname();
if( className == null )
      throw new MiningException( "className attribute expected." );

Class algorithmClass = = Class.forName( className );
Object algorithm = algorithmClass.newInstance();
ClusteringAlgorithm miningAlgorithm = (ClusteringAlgorithm)algorithm;
```

После создания необходимо вызвать метод метод `buildModel()` для построения модели класса `ClusteringMiningModel`.

```
ClusteringMiningModel model =
             (ClusteringMiningModel) algorithm.buildModel();
```

Обработать построенную модель, например, для вывода кластеров в консоль можно следующим образом.

```
System.out.println("number of clusters: " +
clustModel.getNumberOfClusters());
Cluster[] clust = clustModel.getClusters();
for (int i = 0; i < clust.length; i++)
              System.out.println("Clust["+i+"]: " + clust[i].toString() );
```

Задача классификации

Для настройки процесса классификации например с помощью деревьев решений создается экземпляр класса `SupervisedMiningSettings`.

```
SupervisedMiningSettings miningSettings = new SupervisedMiningSettings();
```

Ему передаются метаданные загруженных данных:

```
      miningSettings.setDataSpecification( metaData );
```

Для классификации необходимо указать атрибут соответствующий независимой переменной. Например,

```
MiningAttribute targetAttribute =
   (MiningAttribute)metaData.getMiningAttribute( "contact-lenses" );
miningSettings.setTarget(targetAttribute);
```

Для построения дерева решений можно использовать алгоритм ID3. В библиотеке Xelopes он реализован в классе `com.prudsys.pdm.Models.Classification.DecisionTree.Algorithms.Id3.ID3Algorithm`. Для его использования необходимо выполнить настройку параметров, которые могут быть загружены из файла algorithms.xml.

```
MiningAlgorithmSpecification miningAlgorithmSpecification =
                  MiningAlgorithmSpecification.getMiningAlgorithmSpecificatio
                  n( "Decision Tree (ID3)" );
```

Далее должен быть создан экземпляр алгоритма выполняющего классификацию:

```
String className = miningAlgorithmSpecification.getClassname();
if( className == null )
       throw new MiningException( "className attribute expected." );
Class algorithmClass = = Class.forName( className );
Object algorithm = algorithmClass.newInstance();
DecisionTreeAlgorithm miningAlgorithm = (DecisionTreeAlgorithm)algorithm;
```

После создания необходимо вызвать метод `buildModel()` для построения модели класса `DecisionTreeMiningModel`.

```
DecisionTreeMiningModel model =
             (DecisionTreeMiningModel) algorithm.buildModel();
```

Обработать построенную модель, например, для вывода в консоль можно с помощью рекурсивной процедуры.

```
  private static void showTreeRecursively(DecisionTreeNode node) {

    // Loop over all childs:
    for (int i = 0; i < node.getChildCount(); i++) {
        DecisionTreeNode child = (DecisionTreeNode) node.getChildAt(i);
      System.out.println( "parent: " + node.toString() + " ==> child: " +
               child.toString());

      // Get child's childs:
      showTreeRecursively(child);
    };
   }
```

Она может быть вызвана следующим образом:

```
DecisionTreeNode root = (DecisionTreeNode) model.getClassifier();
showTreeRecursively(root);
```

Деревья решений используются для классификации данных. Следовательно построенная модель может быть применена к новым данным с целью их классификации. Покажем, как это может быть выполнено на практике:

```
int i = 0;
int wrong = 0;
while (inputData.next()) {
       // классифицировать вектор:
       MiningVector vector = inputData.read();
```

```
        double predicted    = ((SupervisedMiningModel) model).apply(vector);

        double realTarCat   = vector.getValue( targetAttribute.getName() );
        Category tarCat     =
              ((CategoricalAttribute)targetAttribute).getCategory(realTarCat)
              ;
        Category predTarCat =
              ((CategoricalAttribute)targetAttribute).getCategory(predicted);
        System.out.println(" " + ++i +": " + vector + " -> " + predTarCat);
        if (! predTarCat.equals( tarCat) )
                wrong = wrong + 1;
};
System.out.println("classification rate = " + (100.0 - ((double) wrong /
              i)*100.0) );
```

Порядок выполнения работы

1. Изучить основные принципы анализа данных с использованием библиотеки Xelopes.
2. Реализовать программу выполняющую построение модели заданной вариантом задания.
3. Построить модели для всех файлов с исходными данными и разными параметрами.
4. Сохранить построенные модели в текстовом формате и формате PMML.
5. Сравнить полученные модели с моделями, построенными с помощью GUI Xelopes для тех же данных и тех же настройках.
6. Если существует разница объяснить ее.

Варианты заданий

Вариант	Модели
1	*StatisticsMiningModel*
2	*AssociationRulesMiningModel*
3	*SequentialMiningModel*
4	*CustomerSequentialMiningModel*
5	*ClusteringMiningModel*
6	*SupportVectorMiningModel*
7	*SparseGridsMiningModel*
8	*DecisionTreeMiningModel*
9	*CustomerSequentialMiningModel*
10	*ClusteringMiningModel*
11	*SupportVectorMiningModel*
12	*SparseGridsMiningModel*
13	*DecisionTreeMiningModel*
14	*ClusteringMiningModel*
15	*SupportVectorMiningModel*
16	*SparseGridsMiningModel*
17	*DecisionTreeMiningModel*
18	*CustomerSequentialMiningModel*
19	*StatisticsMiningModel*
20	*AssociationRulesMiningModel*

Отчет по работе

1. Титульный лист.
2. Цель работы.
3. Блок схема программы
4. Результаты применения к исходным данным из разных файлов с разными параметрами.
5. Сравнение с моделями, построенными с помощью GUI Xelopes.
6. Выводы по работе.

Контрольные вопросы

1. Что необходимо для построения модели.
2. Какие параметры должны быть установлены для построения ассоциативных правил.
3. Какие параметры должны быть установлены для построения кластеров.
4. Какие параметры должны быть установлены для построения дерева решений
5. Как можно классифицировать алгоритмы в соответствии с иерархией принятой в CWM и Xelopes.
6. Что такое формат PMML.

Лабораторная работа №4

Реализация алгоритмов построения Unsupervised - моделей.

Цель и задача работы

Изучить основные принципы разработки алгоритмов data mining строящих unsupervised модели.

Реализовать алгоритм в соответствии с вариантом задания строящий unsupervised модель.

Теоретические положения

Классы библиотеки Xelopes

Библиотека Xelopes позволяет добавлять новые классы, тем самым расширяя ее. В библиотеке имеется необходимая инфраструктура для работы алгоритмов data mining за счет чего процесс внедрения нового алгоритма не сложен. Совместимость с такими стандартами data mining как CWM и PMML позволяет интегрировать реализованные алгоритмы в существующие системы с минимальными затратами.

В соответствии со стандартом CWM общую концепцию работы алгоритмов библиотеки Xelopes можно представить так как это изображено на рис. 4.1.

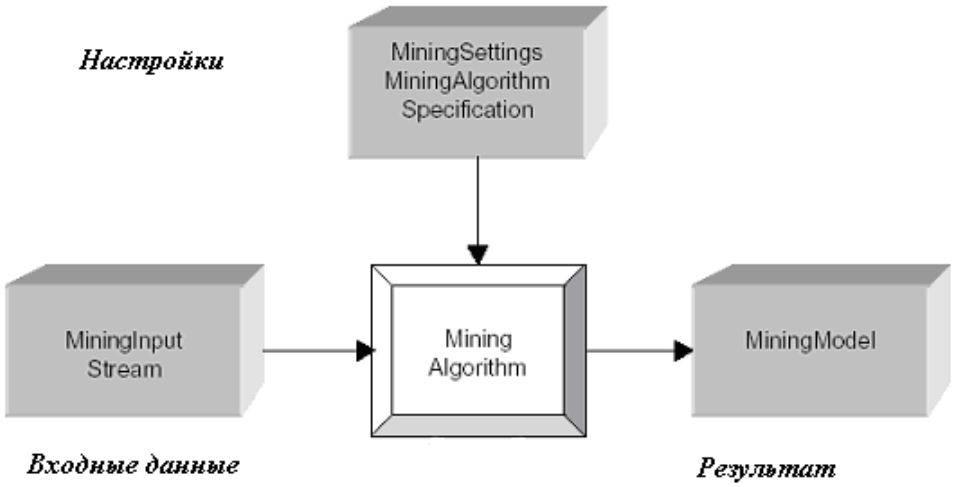


Рисунок 4.1. -Общая концепция работы алгоритмов в библиотеке Xelopes

Согласно данной концепции на вход любого алгоритма подаются исходные данные в виде потока представленного классом `MiningInputStream` . Кроме данных перед работой алгоритма должны быть выполнены соответствующие настройки. Они делятся на два типа: настройки специфичные для решаемой задачи (строящейся модели) и настройки специфичные для конкретного алгоритма. Первые реализуются с помощью наследования от класса `MiningSettings`, второй тип наследование от класса

MiningAlgorithmSpecification. Результат работы алгоритма представляется в виде моделей являющихся экземплярами класса MiningModel. Сами алгоритмы наследуются от класса MiningAlgorithm.

В библиотеке Xelopes реализованы классы настроек, моделей и алгоритмов для основных задач data minig (табл. 4.1). Необходимо заметить, что входные данные не зависят от решаемой задачи и для любого алгоритма представляются в виде экземпляра класса MiningInputStream.

Таблица 4.1. Основные классы библиотеки Xelopes

№	Задача	Класс настроек	Класс модели	Класс алгоритма
1	Статистические задачи	StatisticsMiningSettings	StatisticsMiningModel	StatisticsAlgorithm
2	Поиск ассоциативных правил	AssociationRulesSettings	AssociationRulesMiningModel	AssociationRulesAlgorithm
3	Сиквинциальный анализ	SequentialSettings	SequentialMiningModel	SequentialAlgorithm
4		CustomerSequentialSettings	CustomerSequentialMiningModel	CustomerSequentialAlgorithm
5	Кластеризация	ClusteringMiningSettings	ClusteringMiningModel	ClusteringAlgorithm
6	Регрессии	SupportVectorSettings	SupportVectorMiningModel	SupportVectorAlgorithm
7		SparseGridsSettings	SparseGridsMiningModel	SparseGridsAlgorithm
8	Классификации	DecisionTreeSettings	DecisionTreeMiningModel	DecisionTreeMiningModel

Для реализации собственного алгоритма необходимо реализовать класс, наследуемый от одного из подклассов класса MiningAlgorithm (в зависимости от той задачи, которая решается). В подклассе реализовать метод runAlgorithm(). Данный метод вызывается из метода уже реализованного buildModel(). В этом же методе после после запуска алгоритма выполняется создание модели соответствующего подкласса класса MiningModel. Для корректного построения модели необходимо чтобы класс алгоритма реализовывал методы определенные как абстрактные в классе предке.

Алгоритм должен учитывать настройки. Для этого в классе MiningAlgorithm реализованы методы setMiningSettings(MiningSettings miningSettings) и setApplicationInputSpecification(ApplicationInputSpecification applicationInputSpecification). Входные данные передаются в алгоритм с помощью реализованного в этом же классе метода setMiningInputStream(MiningInputStream miningInputStream).

Таким образом, при для добавления в библиотеку Xelopes нового алгоритма достаточно реализовать абстрактные методы одного из подклассов класса MiningAlgorithm (в зависимости от той задачи, которая решается). Особенности реализации уже зависят от задачи решаемой создаваемым алгоритмом.

4.2. Задача поиска ассоциативных правил

При поиске ассоциативных правил целью является нахождение частых зависимостей (или ассоциаций) между объектами или событиями. Найденные зависимости представляются в виде правил и могут быть использованы как для лучшего понимания природы анализируемых данных, так и для предсказания появления событий.

Рассмотрим реализацию алгоритмов решающих задачу поиска ассоциативных правил он должен наследоваться от класса `com.prudsys.pdm.Models.AssociationRules.AssociationRulesAlgorithm`, который в свою очередь наследуется от класса `com.prudsys.pdm.Core. MiningAlgorithm`. В результате он должен реализовать следующие абстрактные методы вызываемые из метода `buildModel()` класса `AssociationRulesAlgorithm`:

`protected void runAlgorithm()` – реализация алгоритма

`protected Vector getAssociationRules()` – возвращает вектор содержащий построенные ассоциативные правила. Ассоциативное правило представляется экземпляром класса *`RuleSet`*

`protected Vector getLargeItemSets()` – возвращает вектор частых наборов. Частый набор представляется экземпляром класса *`ItemSet`*.

Перед тем как перейти к более подробному описанию этих методов, рассмотрим, как они вызываются из метода `buildModel()` класса `AssociationRulesAlgorithm`. Далее представлена часть кода метода создающая модель класса `AssociationRulesMiningModel` отражающая общий принцип:

```
runAlgorithm();
AssociationRulesMiningModel model = new AssociationRulesMiningModel();

model.setMiningSettings( miningSettings );
model.setInputSpec( applicationInputSpecification );

Vector rules = getAssociationRules();
model.setAssociationRules( rules );

model.setLargeItemSets( getLargeItemSets() );

this.miningModel = model;

return model;
```

Как видно вначале запускается алгоритм, реализуемый подклассом. Далее создается экземпляр модели ассоциативных правил и ей передаются настройки, при которых она была создана. Далее с помощью методов опять же реализуемых подклассом модели присваиваются вектора найденных алгоритмом ассоциативных правил и частых наборов.

Теперь более подробно остановимся на трех методах, которые должны реализовывать подклассы класса `AssociationRulesAlgorithm`.

4.3. Метод runAlgorithm

Метод `runAlgorithm()` реализует алгоритм поиска ассоциативных правил и строит вектор ассоциативных правил и вектор частых наборов. Исходные данные, настройки процесса построения алгоритма и специфичные параметры алгоритм получает через переменные класса `MiningAlgorithm`

```
protected MiningInputStream miningInputStream;
protected MiningSettings miningSettings;
protected MiningAlgorithmSpecification miningAlgorithmSpecification;
```

Доступ к ним осуществляется с помощью соответствующих методов `set/get`.

При решении задачи поиска ассоциативных правил переменная `miningSettings` должна быть экземпляром класса `AssociationRulesSettings`. Она с помощью методов get предоставляет доступ к следующим параметрам процесса построения модели:

`getItemId()` – возвращает категориальный атрибут класса CategoricalAttribute являющийся идентификатором исследуемых объектов (элементов).

`getTransactionId()` – возвращает категориальный атрибут класса `CategoricalAttribute` являющийся идентификатором транзакций.

`getMinimumSupport()` – возвращает значение типа double являющееся значением минимальной поддержки для искомых частых наборов.

`getMinimumConfidence()` - возвращает значение типа double являющееся значением минимальной степени доверия для искомых ассоциативных правил.

4.4. Метод getAssociationRules

Метод `getAssociationRules()` должен возвращать вектор, содержащий построенные ассоциативные правила. Ассоциативное правило представляется экземпляром класса `RuleSet`.

Класс `RuleSet` имеет следующие переменные характеризующее представляемое им правило:

`public ItemSet premise` – переменная представляющая условную часть правила как набор элементов класса `ItemSet`.

`public ItemSet conclusion` – переменная представляющая заключительную часть правила как набор элементов класса `ItemSet`.

`public int size` – размер правила (количество элементов в условной и заключительной части)

`public double support` – поддержка правила;

`public double confidence` – степень доверия правила.

4.5. Метод getLargeItemSets

Метод `getLargeItemSets()` должен возвращать вектор, частых наборов. Частый набор представляется экземпляром класса `ItemSet`.

Класс `ItemSet` имеет следующие переменные, характеризующие представляемый им частый набор:

`private IntVector itemList` – список индексов элементов входящих в набор (реализован как вектор целых чисел).

`private int supportCount` – поддержка набора;

`private int size` – количество элементов входящих в набор.

4.6. Задача кластеризации

Задача кластеризации заключается в поиске независимых групп (кластеров) и их характеристик во всем множестве анализируемых данных. Решение этой задачи помогает лучше понять данные. Кроме того, группировка однородных объектов позволяет сократить их число, а следовательно и облегчить анализ.

Рассмотрим реализацию неиерархических алгоритмов решающих задачу кластеризации. Такие алгоритмы должны наследоваться от класса `com.prudsys.pdm.Models.Clustering.CDBased.CDBasedClusteringAlgorithm`, который наследуется от класса `com.prudsys.pdm.Models.Clustering.ClusteringAlgorithm`, который в свою очередь наследуется от класса `com.prudsys.pdm.Core.MiningAlgorithm`. В результате он должен реализовать следующие методы, вызываемые из метода `buildModel()` класса `CDBasedClusteringAlgorithm`.

`protected void runAlgorithm()` – реализация алгоритма;

`protected Cluster[] getClusters()` – возвращает массив содержащий построенные кластеры. Кластер представляется экземпляром класса `Cluster;`

`public Distances getDistances()` – возвращает используемую меру расстояния. Мера представляется экземпляром класса `Distances`.

Рассмотрим, более подробно метод `buildModel()` класса `CDBasedClusteringAlgorithm`. Далее представлен код метода создающего модель класса `CDBasedClusteringMiningModel`.

```
public MiningModel buildModel() throws MiningException
{
   runAlgorithm();
   CDBasedClusteringMiningModel model = new  CDBasedClusteringMiningModel();
   model.setMiningSettings( miningSettings );
   model.setInputSpec( applicationInputSpecification );
   model.setClusters( getClusters() );
   model.setDistanceType( getDistances() );
   this.miningModel = model;
   return model;
}
```

Так же как и в случае ассоциативных правил вначале запускается алгоритм, реализуемый подклассом. Далее создается экземпляр центрированной/распределенной модели кластеров и ей передаются настройки, при которых она была создана. Далее с помощью метода `getClusters()` модели присваивается массив кластеров. Модель также должна содержать и используемую меру расстояния, которая может быть получена методом `getDistances`.

Теперь более подробно остановимся на методах, которые должны реализовывать подклассы класса `CDBasedClusteringAlgorithm`.

4.7. Метод runAlgorithm

Метод `runAlgorithm()` реализует алгоритм разбиения объектов на группы схожих объектов и строит массив кластеров. Исходные данные, настройки процесса построения алгоритма и специфичные параметры алгоритм получает через переменные класса `MiningAlgorithm`

```
protected MiningInputStream miningInputStream;
protected MiningSettings miningSettings;
protected MiningAlgorithmSpecification miningAlgorithmSpecification;
```

Доступ к ним осуществляется с помощью соответствующих методов set/get.

При решении задачи поиска ассоциативных правил переменная `miningSettings` должна быть экземпляром класса `ClusteringSettings`. Она с помощью методов get предоставляет доступ к следующим параметрам процесса построения модели:

`getClusterIdAttributeName()` – возвращает имя атрибута являющегося идентификатором кластеров.

`getMaxNumberOfClusters()` – максимальное количество строящихся кластеров.

4.8. Метод getClusters

Метод `getClusters ()` должен возвращать массив, содержащий построенные кластеры. Кластер представляется экземпляром класса *Cluster*.

Класс `Cluster` имеет следующие переменные характеризующее представляемый им кластер:

`private String name` – имя кластера;

`private MiningVector centerVec` – вектор определяющий центр данного вектора;

`private Vector centerVectors` – набор векторов представляющих центр данного кластера (несколько векторов возможно в случае преобразования категориальных типов в числовые);

`private Vector containedVectors` – содержит набор векторов (объектов) входящих в данный кластер;

4.9. Метод getDistances

Метод `getDistances()` должен возвращать используемую меру расстояния. Мера представляется экземпляром класса `Distances`.

Класс `Distances` имеет следующие переменные характеризующее представляемую им меру расстояния:

`private int type` - код типа функции используемой для вычисления расстояния (например, Euclidean, Squared Eucliden, City-Block, и т.п.).

`private int measureType` – код типа меры расстояния (distance, similarity)

`private int compareFunction` - код типа функции используемой для сравнения (например, Abs Diff, Gauss-Sim, и т.п.).

`private double[] fieldWeights` – массив весов для каждого атрибута данных ;

`private boolean normalized` – определяет использовать ли при вычислении расстояния нормализацию;

`private double[] minAtt` – массив минимальных значений всех атрибутов (необходимы при нормализации);

`private double[] maxAtt` – массив максимальных значений всех атрибутов (необходимы при нормализации).

Порядок выполнения работы

1. Изучить алгоритм, определенный вариантом задания.
2. Создать новый класс, наследуемый от соответствующего класса библиотеки Xelopes.
3. Реализовать алгоритм, определенный вариантом задания.
4. Добавить алгоритм в библиотеку Xelopes.
5. Отладить алгоритм.
6. Построить с помощью реализованного алгоритма модели для всех файлов с данными.
7. Сравнить полученные результаты с моделями, построенными подобными алгоритмами из библиотеки Xelopes.

Варианты заданий

Вар-т	Алгоритм
1	Apriori TID на основании реализации алгоритма Apriori в Xelopes
2	Алгоритм KMeans
3	Дивизимный алгоритм кластеризации
4	Аггломеративный алгоритм кластеризации
5	Дивизимный алгоритм кластеризации
6	Аггломеративный алгоритм кластеризации
7	Apriori TID на основании реализации алгоритма Apriori в Xelopes

8	Алгоритм KMeans
9	Дивизимный алгоритм кластеризации
10	Аггломеративный алгоритм кластеризации
11	Apriori TID на основании реализации алгоритма Apriori в Xelopes
12	Алгоритм KMeans
13	Дивизимный алгоритм кластеризации
14	Аггломеративный алгоритм кластеризации
15	Дивизимный алгоритм кластеризации
16	Аггломеративный алгоритм кластеризации
17	Apriori TID на основании реализации алгоритма Apriori в Xelopes
18	Алгоритм KMeans
19	Дивизимный алгоритм кластеризации
20	Аггломеративный алгоритм кластеризации

Отчет по работе

1. Титульный лист.
2. Цель работы.
3. Блок схема алгоритма.
4. Результаты применения к исходным данным из разных файлов с пояснениями.
5. Сравнение с моделями, построенными аналогичными алгоритмами, реализованными в библиотеке Xelopes.
6. Выводы по работе.

Контрольные вопросы

1. Что такое unsupervised модели.
2. Что такое описательные модели.
3. Какие модели относятся к типу unsupervised.
4. Какие существуют алгоритмы поиска ассоциативных правил
5. Какие существуют алгоритмы сиквенциального анализа.
6. В чем идея алгоритма KMeans.
7. Какие существуют дивизимные алгоритмы и чем они отличаются друг от друга.
8. Какие существуют аггломеративные алгоритмы и чем они отличаются друг от друга.

Лабораторная работа №5

Реализация алгоритмов построения supervised - моделей.

Цель и задача работы

Изучить основные принципы разработки алгоритмов data mining строящих supervised модели. А также применения этих моделей к новым данным.

Реализовать алгоритм в соответствии с вариантом задания строящий supervised модель, которая могла бы быть использована для предсказания на новых данных.

Теоретические положения

5.1. Supervised модели

Задачи классификации и кластеризации относятся к типу задач с учителем - **supervised learning.** Такие задачи решаются в несколько этапов. Сначала с помощью какого-либо алгоритма data mining, строится модель анализируемых данных - классификатор. Затем, классификатор подвергается "обучению". Другими словами, проверяется качество его работы и, если оно неудовлетворительно, происходит "дополнительное обучение" классификатора. Так происходит до тех пор, пока мы не достигнем требуемого уровня качества или не убедимся, что выбранный алгоритм не работает корректно с данными, либо же сами данные не имеют структуры, которую можно выявить. К этому типу задач относят задачи классификации и регрессии.

В результате описанных отличий в стандарт CWM были введены специальные классы для такого вида задач:

`SupervisedMiningModel` - наследуется от класса `MiningModel` и используется при решении supervised задачах (т.е.задач классификации и регрессии).

`SupervisedMiningSettings` – наследуется от класса `MiningSettings` и используется для определения настроек для задач с учителем.

В соответствии со стандартам библиотека Xelopes также поддерживает данный подход. Более того по аналогии с классами моделей и настроек был добавлен класс `SupervisedMiningAlgorithm`.

В библиотеку был добавлен интерфейс `Classifie`*r,* который описывает метод `applay()` выполняющий классификацию на основе построенной модели новых данных. Данный метод должен реализовываться всеми классами, реализующими интерфейс `Classifier`.

К типу задач с учителем относятся задачи классификации и регрессии. Для строящих соответствующие модели в библиотеке присутствуют следующие классы:

`ClassificationMiningModel` - расширяет `SupervisedMiningModel` для задачи классификации;

`DecisionTreeMiningModel` - расширяет `ClassificationMiningModel` для моделей деревьев решений

- `RegressionMiningModel` - расширяет `SupervisedMiningModel` для задачи регрессии;

`SupportVectorModel` - расширяет `RegressionModel` для моделей построенных методом Support Vector Machines;

`ClassificationMiningSettings` - расширяет `SupervisedMiningSettings` для настройки процесса решения задачи классификации;

`DecisionTreeSettings` - расширяет `ClassificationSettings` для деревьев решений,

- `RegressionMiningSettings` - расширяет `SupervisedMiningSettings` для настройки процесса решения задачи регрессии;
- `SupportVectorSettings` - расширяет `RegressionSettings` для метода `Support` Vector Machines;

`ClassificationAlgorithm` - расширяет `SupervisedMiningAlgorithm` для задачи классификации;

`DecisionTreeMiningAlgorithm` - расширяет `ClassificationMiningAlgorithm` для алгоритмов дерева решений,

`RegressionAlgorithm` - расширяет `SupervisedMiningAlgorithm` для задачи регрессии,

`SupportVectorAlgorithm` - расширяет `RegressionAlgorithm` для алгоритмов Support Vector Machines;

Перечисленные классы и их подклассы необходимы для реализации собственных алгоритмов решающих задачи классификации и регрессии.

Задача классификации

Задача классификации сводится к определению класса объекта по его характеристикам. Необходимо заметить, что в этой задаче множество классов, к которым может быть отнесен объект заранее известно.

Рассмотрим реализацию алгоритмов строящих деревья решений классификации данных. Такие алгоритмы должны наследоваться от класса `com.prudsys.pdm.Models.Classification.DecisionTree.DecisionTreeAlgorithm`. В результате он должен реализовать следующие методы, вызываемые из метода `buildModel()` класса `DecisionTreeAlgorithm`.

`protected void runAlgorithm()`- реализация алгоритма

`protected Classifier getClassifier()` – возвращает классификатор выполняющий классификацию новых данных. Классификатор представляется экземпляром класса реализующего интерфейс `Classifier`

Рассмотрим, более подробно метод `buildModel()` класса `DecisionTreeAlgorithm`. Далее представлен код метода создающего модель класса `DecisionTreeMiningModel`.

```
public MiningModel buildModel() throws MiningException
{
```

```
    runAlgorithm();
    DecisionTreeMiningModel model = new DecisionTreeMiningModel();
    model.setMiningSettings( miningSettings );
    model.setInputSpec( applicationInputSpecification );
    model.setClassifier( getClassifier() );
    ApplicationAttribute target =
                      applicationInputSpecification.getTargetApplicationAttri
                      bute();
    model.setTarget( target );
    this.miningModel = model;
    return model;
}
```

Вначале запускается алгоритм, реализуемый подклассом. Далее создается экземпляр модели представляющей собой дерево решений и ей передаются настройки, при которых она была создана. Далее с помощью метода `getClassifier()` модели присваивается экземпляр класса выполняющего классификацию новых объектов.

Теперь более подробно остановимся на методах, которые должны реализовывать подклассы класса `DecisionTreeAlgorithm`.

5.3. Метод runAlgorithm

Метод `runAlgorithm()` реализует алгоритм построения классификационной функции на основании обучающей выборки. Исходные данные, настройки процесса построения алгоритма и специфичные параметры алгоритм получает через переменные класса `MiningAlgorithm`

```
protected MiningInputStream miningInputStream;
protected MiningSettings miningSettings;
protected MiningAlgorithmSpecification miningAlgorithmSpecification;
```

Доступ к ним осуществляется с помощью соответствующих методов `set/get`.

При построении классифицирующей функции вида дерева решений переменная `miningSettings` должна быть экземпляром класса `DecisionTreeSettings`. Она с помощью методов get предоставляет доступ к следующим параметрам процесса построения модели:

`getCostMatrix()` – матрица стоимости*;*

`getTarget()` – ссылка на экземпляр класса `MiningAttribute` определяющего независимую переменную;

`getPredictedAttributeName()` – имя атрибута значения, которого необходимо предсказать (имя зависимой переменной);

`getConfidenceAttributeName()` – имя атрибута определяющего ступень доверия предсказания.

`getCostFunction()` – функция, определяющая стоимость некорректного предсказания. Значениями могут быть entropy, Gini, costMatrix, pnorm или none

5.4. Метод getClassifier

Метод `getClassifier()` должен возвращать классификатор выполняющий классификацию новых данных. Классификатор представляется экземпляром класса реализующего интерфейс `Classifier`

Интерфейс *Classifier* описывает один единственный метод:

```
public double apply(MiningVector miningVector)
```

В качестве аргумента методу передается вектор описывающий некий объект, который необходимо отнести к одному из известных классов (т.е. классифицировать). В качестве возвращаемого значения передается оценка классификации/регрессии для данного вектора.

Для классификации данных с помощью деревьев решений в библиотеке находится класс
`com.prudsys.pdm.Models.Classification.DecisionTree.DecisionTreeNode`
реализующий интерфейс `Classifier`. Этот класс наследуется от класса `com.prudsys.pdm.Core.MiningTreeNode` который реализует функциональность необходимую для перемещения по дереву от одного узла к другому. Кроме нее класс `DecisionTreeNode` реализует метод `applay()`:

```
public double apply( MiningVector miningVector ) throws MiningException
{
  if(leaf) return score;
  else
  {
    for(int i=0;i<children.length;i++)
    {
      DecisionTreeNode dtn = (DecisionTreeNode)children[i];
      if(dtn.predicate.evaluate(miningVector))
        return dtn.apply(miningVector);
    }
    throw new MiningException("bad tree mode");
  }
}
```

В данном методе новый вектор рекурсивно сравнивается с некоторым узлом дерева. Выбор очередного узла осуществляется в зависимости от результата сравнения с родительским узлом. Вид сравнения узлов может быть самый разный.

Классификатор строится в процессе работы реализуемого алгоритма. В связи с этим он может зависеть от особенностей алгоритма. Следовательно, вместе с реализацией алгоритма в библиотеку должен быть добавлен и корректный классификатор.

Задача регрессии

Задача регрессии подобно задаче классификации позволяет определить по известным характеристикам объекта значение некоторого его параметра. В отличие от задачи классификации значением параметра является не конечное множество классов, а множество действительных чисел. В связи с

близостью этих двух задач подход реализации алгоритмов построения функции регрессии близок к построению классификатора.

Алгоритмы строящие функциональные зависимости должны наследоваться от класса `com.prudsys.pdm.Models.Regression.RegressionAlgorithm` и реализовать те же методы, что и в случае с классификацией.

`protected void runAlgorithm()` – реализация алгоритма

`protected Classifier getClassifier()` – возвращает классификатор вычисляющий значение зависимой переменной. Классификатор представляется экземпляром класса реализующего интерфейс `Classifier`

Метод `buildModel()` класса `RegressionAlgorithm` создает модель класса `RegressionMiningModel` *по аналогии с* классмом `DecisionTreeAlgorithm`.

Настройки, реализуемые классом `RegressionSettings`, те же что и в задаче классификации. Естественно, что строящийся классификатор отличен от рассмотренного при описании задачи классификации. Он во многом определяется используемым методом data mining и должен быть реализован совместно с алгоритмом.

Порядок выполнения работы

1. Изучить алгоритм, определенный вариантом задания.
2. Создать новый класс, наследуемый от соответствующего класса библиотеки Xelopes.
3. Реализовать алгоритм, определенный вариантом задания.
4. Добавить алгоритм в библиотеку Xelopes.
5. Отладить алгоритм.
6. Построить с помощью реализованного алгоритма модели для всех файлов с данными.
7. Применить построенные модели к новым данным.
8. Сравнить полученные результаты с моделями, построенными подобными алгоритмами из библиотеки Xelopes.

Варианты заданий

Вариант	Алгоритм
1	Алгоритм 1R
2	Алгоритм Naïve Bayes
3	Используя реализацию алгоритма ID3 реализовать алгоритм C4.5
4	Алгоритм покрытия
5	Алгоритм наименьших квадратов.
6	Линейную разновидность алгоритма SVM
7	Полиномиальную разновидность алгоритма SVM
8	Гауссовскую разновидность алгоритма SVM
9	Алгоритм покрытия
10	Алгоритм наименьших квадратов.
11	Линейную разновидность алгоритма SVM

12	Полиномиальную разновидность алгоритма SVM
13	Гауссовскую разновидность алгоритма SVM
14	Алгоритм 1R
15	Алгоритм Naïve Bayes
16	Используя реализацию алгоритма ID3 реализовать алгоритм C4.5
17	Алгоритм покрытия
18	Алгоритм наименьших квадратов.
19	Алгоритм Naïve Bayes
20	Используя реализацию алгоритма ID3 реализовать алгоритм C4.5

Отчет по работе

1. Титульный лист.
2. Цель работы.
3. Блок схема алгоритма.
4. Результаты применения к исходным данным из разных файлов с пояснениями.
5. Сравнение с моделями, построенными аналогичными алгоритмами, реализованными в библиотеке Xelopes.
6. Результаты применения к новым данным с пояснениями.
7. Выводы по работе.

Контрольные вопросы

1. Что такое supervised модели?
2. Что такое предсказательные модели?
3. Какие модели относятся к типу supervised?
4. Какие существуют алгоритмы построения классификационных функций?
5. Какие существуют алгоритмы построения функций регрессии?
6. Какие существуют алгоритмы построения классификационных правил?
7. Какие существуют алгоритмы построения деревьев решений?
8. Какие существуют алгоритмы построения математических зависимостей?

Рассмотрено на заседании
кафедры АТМ
Протокол № _____ от _____
Зав. кафедрой

_____ А.А. Фомичев

Тула: ЭУМК ТулГУ, рег.№1023, 2010.