**Project Machine Learning**

**Đề tài:** Giải quyết bài toán phân loại dân cư dựa trên việc xây dựng cây quyết định và thuật toán Naïve Bayes.

**Nhóm thực hiện:**

* Lê Minh Nghĩa.
* Bùi Trung Hiếu.
* Phạm Tuấn Long.

**Người viết document:** Lê Minh Nghĩa.

**Ngày viết document:** 5/4/2010.

**Mục Lục.**

[I. Các Interface. 4](#_Toc258978317)

[1. IC4.5. 4](#_Toc258978318)

[2. IProbabilityTable 4](#_Toc258978319)

[3. IProbability<T>. 4](#_Toc258978320)

[4. IStatisticObject. 4](#_Toc258978321)

[5. INode. 5](#_Toc258978322)

[6. IFile. 5](#_Toc258978323)

[7. IData. 6](#_Toc258978324)

[8. IComparer. 7](#_Toc258978325)

[9. IEnumerable<T>. 7](#_Toc258978326)

[10. IDiscreteIndex. 7](#_Toc258978327)

[11. IClassification. 8](#_Toc258978328)

[II. Các Class. 8](#_Toc258978329)

[1. Cac Enum. 8](#_Toc258978330)

[2. Case. 9](#_Toc258978331)

[3. FileProcessing. 11](#_Toc258978332)

[4. Condition. 12](#_Toc258978333)

[5. Data. 13](#_Toc258978334)

[6. StatisticObject. 15](#_Toc258978335)

[7. NodeIEnumerator. 16](#_Toc258978336)

[8. Node. 16](#_Toc258978337)

[9. C45. 19](#_Toc258978338)

[10. Knowledge. 22](#_Toc258978339)

[11. Probability. 23](#_Toc258978340)

[12. ProbabilityTable. 23](#_Toc258978341)

[13. NodeID3 23](#_Toc258978342)

[14. TreeID3. 23](#_Toc258978343)

[15. AttributeDetail. 23](#_Toc258978344)

[16. NaiveBayes. 23](#_Toc258978345)

[17. HoursPWComparer. 23](#_Toc258978346)

[18. FnlwgtComparer. 23](#_Toc258978347)

[19. EduNumComparer. 23](#_Toc258978348)

[20. CapitalLossComparer. 23](#_Toc258978349)

[21. CapitalGainComparer. 23](#_Toc258978350)

[22. AgeComparer. 24](#_Toc258978351)

## Các Interface.

## IC4.5.

Interface IC4.5 dùng để định nghĩa các phương thức cho lớp thuật toán C4.5.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | void ConstructTree(); | Kích hoạt thuật toán xây dựng cây quyết định |
| 2 | bool IsConstructed  {  get;  } | Xác định xem cây đã được xây dựng chưa? |
| 3 | Node Root  {  get;  } | Lấy gốc của cây quyết định sau khi phân loại xong. |

## IProbabilityTable

Dùng để định nghĩa các phương thức cho lớp ProbabilityTable dùng để lưu các xác suất tìm được.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | Void AddProbality (AttributeCase attr, string Value, params double[] probality); | Đưa mảng hai xác suất của một value của một attribute nào đó.(xác suất nhỏ hơn và xác suất lớn 50K$) |
| 2 | double[] GetProbality(AttributeCase attr, string Value) | Lấy một mảng hai xác suất thành phần ứng với một value của một attribute nào đó. |

## IProbability<T>.

Dùng để định nghĩa các phương thức cho lớp Probability ứng với một thuộc tính nào đó là kiểu dữ liệu T.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | void AddProbality(T keyValue, params double[] probality); | Đưa một mảng hai xác suất thành phần ứng với một value nào đó của Attribute T |
| 2 | double[] GetProbality(T keyValue) | Lấy một mảng hai xác suất thành phần nào đó ứng với một giá trị của thuộc tính T nào đó. |

## IStatisticObject.

Dùng để định nghĩa thông tin cho lớp StatisticObject dùng để lưu trữ thông tin thống kê.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | double NumberGreater  { get;} | Lấy số trường hợp có thu nhập lớn 50k$ |
| 2 | double NumberLessEqua  { get;} | Lấy số phần tử có thu nhập nhỏ hơn hoặc bằng 50k$ |
| 3 | double Total  { get;} | Lấy tổng số phần tử. |

## INode.

Dùng để định nghĩa phương thức cho lớp Node

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | void MakeLabel(AttributeCase attr, double? threshold, double numberCase, double greaterNum, double lessEqualNum); | Đánh nhãn cho một node là một attribute nào đó. Attr: thuộc tính cần đánh, threshold: khoảng ứng với thuộc tính liên tục, numberCase số phần tử xét khi đánh nhãn, greaterNum số trường hợp lớn 50K$, lessEquaNum số phần tử nhỏ hơn hoặc bằng 50K$. |
| 2 | void SetLeaf(bool result, double numberCase, double numberError); | Đặt trạng thái một node thành lá với trạng thái result: true- lớn hơn 50K$, false-Nhỏ hơn 50K$; Numbercase: số trường hợp lớn hơn 50K$, numberError: số trường hợp lỗi. |
| 3 | bool Classify(Case ca); | Phân loại một trường hợp sau khi cây được xây dựng. |
| 4 | Node GetChildNode(AttributeCase attr, object value); | Lấy một node con của một attribute nào đó có value tương ứng. |
| 5 | AttributeCase Label  {get;set;} | Lấy nhãn và đặt nhãn của môt node |
| 6 | bool IsLeaf  {get;set;} | Lấy trạng thái của một node có phải là lá hay không? |
| 7 | double NumberCase  {get;set;} | Số trường hợp được xét khi node được đánh nhãn. |
| 8 | double NumberError  {get;set;} | Số trường hợp bị lỗi. |
| 9 | bool Result  {get;set;} | Trạng thái của một node lá: true – lớn hơn 50K$, và false-nhỏ hơn 50K$ |
| 10 | int NumberChilds  {get;} | Số con của một node |
| 11 | Node Parent  {get;set;} | Node cha của một node |
| 12 | Node[] ListChild  {get;set;} | List các tham chiếu tới node con |
| 13 | double GreaterNumber  {get;set;} | Số phần tử lớn hơn 50K$ |
| 14 | double LessEquaNumber  {get;set;} | Số phần tử nhỏ hơn hoặc bằng 50K$ |

## IFile.

Interface IFile quy định các phương thức cho lớp xử lý file File Process.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | Case[] ReadFile(string path) | Đọc một file dữ liệu từ đường dẫn đầu vào. Dữ liệu trả ra là một mảng các case. |
| 2 | void SaveCase(string path, params Case[] datas); | Ghi một mảng các case ra, một file có đường dẫn cho trước. |
| 3 | void TreeSerialization(string path, Node root); | Ghi một cây quyết định ra file. |
| 4 | Node TreeDeserialization(string path); | Đọc một cây quyết định từ file và trả ra node gốc. |

## IData.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | Data Filter(params Condition[] conditions); | Lọc dữ liệu theo một list các condition đầu vào. Trả ra một subset của dataset. |
| 2 | double Entropy(); | Tính entropy của bộ dữ liệu. |
| 3 | double[] GetThreshold(AttributeCase continuousAttr, IComparer<Case> comparer); | Lấy một mảng các thresholde của một attribute liên tục nào đó. Chọn bộ Comparer tương ứng với mỗi attribute liên tục. |
| 4 | double NumberMissingValues(AttributeCase attr); | Trả lại số trường hợp missing ứng với một attribute nào đó. |
| 5 | double NumberGreaterCase(); | Trả lại số trường hợp có thu nhập lớn hơn 50k$ |
| 6 | Case GetRandomCase(); | Lấy random một case bất kỳ trong dataset |
| 7 | StatictisObject[] Statistic(AttributeCase attr, double? threshold); | Thống kê số trường hợp lớn hơn và nhỏ hơn hoặc bằng 50K$ ứng với một attribute nào đó. Với Attribute rời rạc trả ra một mảng tương ứng với các valude của Attribute đó. Còn Attribute liên tục thì trả lại mảng hai phần tử ứng với nhánh lớn hơn và nhánh nhỏ hơn. |
| 8 | void InitialInfo(); | Khởi tạo các lượng tin trung bình của các attribute(dùng trong trường hợp có mising value) |
| 9 | double TotalMissingCase(); | Tổng số phần tử missing value của data set |
| 10 | void CleanData(); | Chuẩn hóa dữ liệu đối với các dữ liệu bị missing. |
| 11 | int NumberCase  {get;} | Tổng số phần tử của dataset |
| 12 | Case this[int i]  {get;} | Toán tử index để duyệt các phần tử trong đối tượng data |
| 13 | Case[] Cases  {get;} | Trả lại một mảng các trường hợp. |
| 14 | Dictionary<AttributeCase, object> MostCommonLabelGreatr{get;set;} | Lấy một tập các nhãn phổ biến nhất cho mỗi attribute ứng với trường hợp có thu nhập lớn hơn 50K$ |
| 15 | Dictionary<AttributeCase, object> MostCommonLabelLessEqua  {get;set;} | Lấy một tập các nhãn phổ biến nhất cho mỗi attribute ứng với trường hợp có thu nhập nhỏ hơn hoặc bằng 50K$ |

## IComparer.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | int IComparer<T>.Compare(T c1, T c2) | So sánh hai phần tử kiểu dữ liệu T: trả lại 1 nếu hơn và trả lại 0 nếu nhỏ hoặc bằng. |

## IEnumerable<T>.

Dùng để quy định các phương thức mà một lớp phải có để có thể duyệt các phần tử trong lớp đó như duyệt mảng.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | T IEnumerator<T>.Current{get;} | Trả lại phần tử hiện tại trong quá trình duyệt là kiểu dữ liệu T |
| 2 | void IDisposable.Dispose() | Giải phóng tài nguyên |
| 3 | object System.Collections.IEnumerator.Current  {get;} | Trả lại phần tử hiện tại, dữ liệu trả ra là kiểu Object |
| 4 | bool System.Collections.IEnumerator.MoveNext() | Trạng thái duyệt tiếp theo, nếu là duyệt đến phần tử cuối cùng thì là false. |
| 5 | void System.Collections.IEnumerator.Reset() | Reset về trạng thái ban đầu |

## IDiscreteIndex.

Quy định các phương thức của lớp DiscreteIndex nhằm xác định vị trí của value trong một attribute Enum

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | public int IndexValues(AttributeCase attr, object discreteValue) | Xác định vị trí một giá trị rời rạc trong một kiểu enum của một attribute nào đó. |
| 2 | int WorkClassValues  {get;} | Số value của kiểu WorkClass |
| 3 | int EducationValues  {get;} | Số value của kiểu Education |
| 4 | int MaritalStatusValues  {get;} | Số value của kiểu Marital Status |
| 5 | int OccupationValues  {get;} | Số value của kiểu Occupation |
| 6 | int RelationshipValues  {get;} | Số value của kiểu Relationship |
| 7 | int RaceValues  {get;} | Số value của kiểu Relationship |
| 8 | int SexValues  {get;} | Số value của kiểu Sex |
| 9 | int NaitiveCountryValues  {get;} | Số value của kiểu Sex |

## IClassification.

Quy định các phương thức mà một lớp phân loại như C4.5, ID3… phải có.

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | bool Classify(Case ca); | Phân loại một trường hợp |
| 2 | bool IsConstructed  {get;} | Xác định trạng thái xem đã được xây dựng hay chưa. |
|  |  |  |

## Các Class.

## Cac Enum.

Các Enum là các cấu trúc dữ liệu dùng để lưu thông tin bài toán. Chủ yếu là tập giá trị của các attribute rời rạc. Các Enum này đều được đặt trong file code **EnumdataType.cs**. Cụ thể các Enum.

1. **AttributeCase.**

Gồm 15 giá trị. Trong đó có 16 giá trị tương ứng với 16 thuộc tính của mỗi đối tượng case. Giá trị thứ 16 là unknow. Cụ thể xem thêm trong tài liệu mô tả bộ dữ liệu.

1. **Sex.**

Lưu trữ thông tin giới tính của mỗi case. Có 3 giá trị: Male, Female, Unknow.

1. **WorkClass.**

Dùng để lưu các giá trị của attribute WorkClass. Cụ thể xem thêm trong tài liệu mô tả bộ dữ liệu.

1. **Education.**

Dùng để lưu các giá trị của attribute Education. Cụ thể xem thêm trong tài liệu mô tả bộ dữ liệu.

1. **Maritalstatus.**

Dùng để lưu các giá trị của attribute MaritalSatatus. Cụ thể xem trong tài liệu mô tả bộ dữ liệu.

1. **Occupation.**

Dùng để lưu các giá trị của attribute Occupation. Cụ thể xem trong tài liệu mô tả bộ dữ liệu.

1. **Relationship.**

Dùng để lưu các giá trị của attribute Relationship. Cụ thể xem trong tài liệu mô tả bộ dữ liệu.

1. **Race.**

Dùng để lưu các giá trị của attribute Race. Cụ thể xem trong tài liệu mô tả bộ dữ liệu.

1. **NativeCountry.**

Dùng để lưu các giá trị của attribute NativeCountry.. Cụ thể xem trong tài liệu mô tả bộ dữ liệu.

1. **StateCompare.**

Dùng để lưu các trạng thái so sánh đối với các thuộc tính rời rạc. Enum này được sử dụng trong quá trình lọc và tìm kiếm dữ liệu.

1. **Algorithm.**

Chứa giá trị tương ứng với ba thuật toán cài đặt: ID3, C4.5, NaiveBayes.

## Case.

Lớp Case dùng để lưu thông tin mỗi trường hợp của bài toán. Tuy thuộc tính liên tục của mỗi trường hợp là kiểu interger nhưng khi lưu trữ phải lưu dạng kiểu số thực kiểu double. Vì để thuận tiện cho việc tính threshold khi cộng chia đôi hai thuộc tính liên tiếp. Các phương thức và thuộc tính:

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | public Case(double age, WorkClass wcl, double fnlwgt, Education edu, double eduNum,MaritalStatus mts, Occupation occ, Relationship re, Race ra,Sex sexCase, double capitalGain, double capitalLoss, double hoursPerWeek,NativeCountry nc, bool isGreater) | Đây là phương thức khởi tạo cho mỗi trường hợp. Tham số IsGreater dùng để xác định có thu nhập lớn hơn 50K$ hay nhỏ hơn 50K$. |

1. **Các Private Fields.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | age | Age |
| 2 | wcl | WorkClass |
| 3 | fnlwgt | FnlWgt |
| 4 | education | Education |
| 5 | eduNum | EduNum |
| 6 | mariSt | MaritalStatus |
| 7 | occ | Occupation |
| 8 | reltsh | Relasionship |
| 9 | ra | Race |
| 10 | sexCase | Sex |
| 11 | capitalGain | CapitalGain |
| 12 | capitalLoss | CapitalLoss |
| 13 | hoursPerWeek | HoursPerWeek |
| 14 | ntvCountry | NativeCountry |
| 15 | isMissingValue | Xác định xem có bị missing value ở trường hợp nào không. |
| 16 | isGreater | Xác định mức thu nhập |

1. **Các Property.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Properties** | **Ý Nghĩa.** |
| 1 | public double Age  {  get { return age; }  set { age = value; }  } |  |
| 2 | public WorkClass Wcl  {  get { return this.wcl; }  set { this.wcl = value; }  } |  |
| 3 | public double Fnlwgt  {  get { return fnlwgt; }  set { fnlwgt = value; }  } |  |
| 4 | public Education Edu  {  get { return education; }  set { education = value; }  } |  |
| 5 | public double EduNum  {  get { return eduNum; }  set { eduNum = value; }  } |  |
| 6 | public MaritalStatus MariSt  {  get { return mariSt; }  set { mariSt = value; }  } |  |
| 7 | public Occupation Occ  {  get { return occ; }  set { occ = value; }  } |  |
| 8 | public Relationship Reltsh  {  get { return reltsh; }  set { reltsh = value; }  } |  |
| 9 | public Race Ra  {  get { return ra; }  set { ra = value; }  } |  |
| 10 | public Sex SexCase  {  get { return sexCase; }  set { sexCase = value; }  } |  |
| 11 | public double CapitalGain  {  get { return capitalGain; }  set { capitalGain = value; }  } |  |
| 12 | public double CapitalLoss  {  get { return capitalLoss; }  set { capitalLoss = value; }  } |  |
| 13 | public double HoursPerWeek  {  get { return hoursPerWeek; }  set { hoursPerWeek = value; }  } |  |
| 14 | public NativeCountry NtvCountry  {  get { return ntvCountry; }  set { ntvCountry = value; }  } |  |
| 15 | public bool IsMissingValue  {  get { return isMissingValue; }  set { isMissingValue = value; }  } |  |
| 16 | public bool IsGreater  {  get { return isGreater; }  set { isGreater = value; }  } |  |

## FileProcessing.

Lớp dùng để đọc thông tin dữ liệu từ file, lưu các dữ liệu cần thiết trong quá trình xử lý ra file.

1. **Các private fields**
2. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public Case[] ReadFile(string path) | Đọc dữ liều từ đường dẫn, trả lại một mảng các case. |
| 2 | public void SaveCase(string path, params Case[] datas) | Lưu một mảng các case ra một file có đường dẫn path. |
| 3 | public void TreeSerialization(string path, Node root) | Lưu trữ một cây quyết định ra file. Tham số đầu vào là gốc và đường dẫn. Sử dụng cơ chế Serializable. |
| 4 | public Node TreeDeserialization(string path) | Đọc dữ liệu tạo thành một node từ đường dẫn của một file. |
| 5 | public void CreateValidateData(string originalFilePath, string validateFilePath, string newTrainningDataPath, double percent) | Tách bộ dữ liệu training gốc thành dữ liệu dữ liệu training và dữ liệu validate, với tỉ lệ phần trăm nào đó. Tham số đầu vào là đường dẫn file dữ liệu gốc và đường dẫn để lưu các file dữ liệu tạo thành, cùng với tỉ lệ percent |
| 6 | public void SaveKnowledge(Knowledge knowledge, string path) | Lưu trữ tri thức khai phá được từ bộ dữ liệu ra file |
| 7 | public Knowledge ReadKnowledge(string path) | Đọc dữ liệu tri thức từ file để phục vụ việc phân loại. |

1. **Các thuộc tính.**

## Condition.

Lớp condition dùng để lưu các thông tin điều kiện dùng để lọc dữ liệu hay để tìm kiếm. Ví dụ tìm kiếm các phần tử có attribute WorkClass là một giá trị nào đó. Đối với thuộc tính liên tục thì cần phải so sánh là lớn hơn, nhỏ hơn…

1. **Các private fields**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private AttributeCase attr | Thuộc tính điều kiện |
| 2 | private string val | Giá trị ứng với thuộc tính đó |
| 3 | private StateCompare compare | Thông tin so sánh. |

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public Condition()  {} | Phương thức khởi tạo mặc định. |
| 2 | public Condition(AttributeCase attr, string value, StateCompare compareState) | Khởi tạo với các tham số về: thuộc tính điều kiện, giá trị, và trạng thái so sánh ứng với thuộc tính liên tục. |

1. **Các property.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public AttributeCase Attr  {  get { return this.attr; }  set { this.attr = value; }  } | Thuộc tính điều kiện |
| 2 | public string Value  {  get { return this.val; }  set { this.val = value; }  } | Giá trị để đánh giá |
| 3 | public StateCompare Compare  {  get { return this.compare; }  set { this.compare = value; }  } | Trạng thái so sánh ứng với thuộc tính liên tục. |

## Data.

Lớp Data đóng vài trò là nơi lưu trữ tập trung dữ liệu để tiến hành xử lý phục vụ các thuật toán khác nhau. Để sử dụng được lớp Data cần thực hiện ba bước: khởi tạo, gọi hàm clean data, sau đó gọi hàm IntialInfoX để khởi tạo các thông số trong trường hợp xử lý missing value bằng xác suất.

1. **Các private fields.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | Case[] \_listCase | Mảng lưu trữ các case |
| 2 | private int \_numberCase | Số case trong data set |
| 3 | private IFile \_file | Đối tượng dùng để đọc file dữ liệu |
| 4 | private Dictionary<AttributeCase, double> \_info | Dùng để lưu thông entropy trung bình ứng với một attribute nào đó. Chỉ có ý nghĩa trong trường hợp xử lý missing value bằng xác suất. |
| 5 | private Dictionary<AttributeCase, double> \_missingNum | Dùng để lưu số lượng missing ứng với một attribute nào đó. |
| 6 | private Dictionary<AttributeCase, double[]> \_thresholdArr | Mảng các threshold ứng với một attribute nào đó. |
| 7 | private Dictionary<AttributeCase, object> \_mostCommonLabelGreatr | Nhãn phổ biến nhất ứng với trường hợp lớn hơn 50K$ ứng với một value của một attribute nào đó. |
| 8 | private Dictionary<AttributeCase, object> \_mostCommonLabelLessEqua | Nhãn phổ biến nhất ứng với trường hợp nhỏ hơn hoặc bằng 50K$ ứng với một value của một attribute nào đó. |

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 0 | public Data(IFile fileProcessing) | Bộ khởi tạo, tham số là một đối tượng fileProcessing để phục vụ việc đọc. |
|  | public Data(params Case[] arrayCase) | Bộ khởi tạo, tham số mảng các case |
| 1 | public void InitialInfo() | Khởi tạo tính toán trước lượng tin trung bình trong trường hợp xủ lý missing value bằng xác suất. |
| 2 | public Data Filter(params Condition[] conditions) | Lọc dữ liệu với một mảng các điều kiện. Dữ liệu trả ra là một subset |
| 3 | public double Entropy() | Tính Entropy |
| 4 | public double[] GetThreshold(AttributeCase continuousAttr, IComparer<Case> comparer) | Lấy một mảng các threshold ứng với một attribute liên tục nào đó, với một so sánh tương ứng. |
| 5 | public double NumberMissingValues(AttributeCase attr) | Tính số missing value ứng với một attribute nào đó. |
| 6 | public double TotalMissingCase() | Tính toán tổng số tất cả các trường hợp bị missing value. |
| 7 | public double NumberGreaterCase() | Đếm tổng số trường hợp có thu nhập lớn hơn 50K$ |
| 8 | public StatictisObject[] Statistic(AttributeCase attr, double? threshold) | Thống kê số lượng trường hợp có thu nhập nhỏ hơn và lớn hơn 50K$ ứng một attribute nào đó. Đối với thuộc tính rời rạc thì trả lại ra một mảng n phần tử ứng với n value của attribute đó, còn với thuộc tính liên tục là hai phần tử ứng với trường hợp lớn hơn và nhỏ hơn threshold. |
| 9 | public Case GetRandomCase() | Lấy random một trường hợp trong bộ dữ liệu. |
| 10 | public void CleanData() | Làm sách dữ liệu, thay thế mọi missing value bằng nhãn phổ biến nhất. |
| 11 | public double CountCase(AttributeCase attr, object value, StateCompare compare) | Đếm số case của một thuộc tính nào đó, với value và trạng thái so sáng tương ứng. Trạng thái so sánh chỉ có tác dụng với attribute liên tục. |
| 12 | public void LoadData(string path) | Load dữ liệu từ một đường dẫn tương ứng. |
|  |  |  |

1. **Các thuộc tính.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public int NumberCase  {  get { return this.\_numberCase; }  } | Lấy số lượng phần tử của tập dữ liệu. |
| 2 | public Case this[int i]  {  get { return this.\_listCase[i]; }  } | Truy cập đến phần tử thứ I qua chỉ số index |
| 3 | public Case[] Cases  {  get { return this.\_listCase; }  } | Lấy mảng các phần tử dữ liệu. |
| 4 | public Dictionary<AttributeCase, double> Info  {  get { return \_info; }  } | Lấy lượng tin trung bình ứng với một attribute nào đó. Chỉ dùng trong trường hợp xử lý missing value bằng xác suất. |
| 5 | public Dictionary<AttributeCase, double> MissingNum | Lấy số lượng phần tử bị missing value ứng với một attribute nào đó. |
| 6 | public Dictionary<AttributeCase, double[]> ThresholdArr  {  get { return \_thresholdArr; }  } | Lấy mảng threshold ứng với một attribute liên tục nào đó. Chỉ sử dụng được với tập data set gốc, các tập subset trong quá trình tách khi xây dựng cây quyết định không khởi tạo thuộc tính này. |
| 7 | public Dictionary<AttributeCase, object> MostCommonLabelGreatr  {  get { return \_mostCommonLabelGreatr; }  } | Lấy tập các nhãn phổ biến nhất ứng với các attribute trong trường hợp có thu nhập lớn hơn 50K$ |
| 8 | public Dictionary<AttributeCase, object> MostCommonLabelLessEqua  {  get { return \_mostCommonLabelLessEqua; }  } | Lấy tập các nhãn phổ biến nhất ứng với các attribute trong trường hợp có thu nhập nhỏ hơn hoặc bằng 50K$ |

## StatisticObject.

Dùng để lưu trữ các thông tin thống kê dữ liệu: các trường hợp lớn hơn và nhỏ hơn 50K$.

1. **Các private fields.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private double \_numberGreater | Số phần tử lớn hơn 50K$ |
| 2 | private double \_numberLessEqua | Số phần tử nhỏ hơn hoặc bằng 50K$ |
| 3 | private double \_total | Tổng số phần tử. |

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public StatictisObject(double numberGreater, double numberLessEqua) | Bộ khởi tạo. Tham số là số phần tử lớn hơn và nhỏ hơn 50K$ |

1. **Các thuộc tính.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public double NumberGreater  {  get { return this.\_numberGreater;}  } | Lấy số phần tử lớn hơn 50K$ |
| 2 | public double NumberLessEqua  {  get {return this.\_numberLessEqua;}  } | Lấy số phần tử nhỏ hơn 50K$ |
| 3 | public double Total  {  get { return this.\_total; }  } | Lấy tổng số phần tử. |

## NodeIEnumerator.

Cài đặt interface IEnumerator<T> dùng để duyệt cây. Khi đó cây sẽ có vai trò như một mảng có thể sử dụng câu lệnh foreach để duyệt qua tất cả các phần tử trong cây. Cây dược duyệt theo thứ tự sau.

1. **Private Fields**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private Queue<Node> queueData | Hàng đợi dùng để lưu các node trong quá trình duyệt. |
| 2 | private Node currentTree | Gốc của cây. |
| 3 | private Node currentNode | Node tại thời điểm hiện tại đang duyệt. |

1. **Phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public NodeIENumerator(Node tree) | Bộ khởi tạo |
| 2 | private void Iterator(Queue<Node> enumData, Node subTree) | Duyệt cây đệ quy theo thứ tự sau. |
| 3 | bool System.Collections.IEnumerator.MoveNext() | Dịch chuyển tới node tiếp theo |
| 4 | void System.Collections.IEnumerator.Reset() | Reset về vị trị đầu. trong trường hợp này không cài. |

1. **Properties**

|  |  |  |
| --- | --- | --- |
| **STT** | **Properties** | **Ý nghĩa** |
| 1 | Node IEnumerator<Node>.Current  {  Get;  } | Lấy node hiện tại trong quá trình duyệt. |
| 2 |  |  |

## Node.

Dùng để xây dựng cây quyết định. Mỗi node có một mảng các tham chiếu tới các node con mà nó có thể có. Đây là một cây nhiều nhánh. Số nhánh con phụ thuộc việc node được đánh nhãn ứng với thuộc tính nào. Với thuộc tính liên tục thì đó sẽ là hai nhánh ứng với lớn hơn và nhỏ hơn threshold của thuộc tính liên tục được đánh nhãn cho node đó. Còn đối với thuộc tính rời rạc thì số nhánh con bằng số value mà nó có thể có. Mỗi node con có một tham chiếu tới node cha để thực hiện duyệt ngược khi cần.

1. **Private Fields**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private AttributeCase \_label | Nhãn của node là một thuộc tính nào đó. |
| 2 | private Node \_parent | Con trỏ trỏ tới node cha. |
| 3 | private Node[] \_listChild | Mảng các tham chiếu tới các node con. |
| 4 | private DiscreteIndex \_indexOfChild | Dùng để xác định vị trị của các node con trong mảng ứng với thuộc tính rời rạc nào đó. |
| 5 | private double \_threshold | Threshold ứng với trường hợp node được đánh nhãn là thuộc tính liên tục. |
| 6 | private bool \_isLeaf | Xác định xem node đó có là lá hay không. |
| 7 | private double \_numberCase | Số trường hợp của data set được xét khi xác định nhãn cho lớp |
| 8 | private double \_numberError; | Số trường hợp bị lỗi khi xét kết quả của một node lá là lớn hơn hay nhỏ hoặc bằng 50K$ |
| 9 | private double \_greaterNumber | Số lượng phần tử lớn 50K$ tại lúc đánh nhãn cho lá. |
| 10 | private double \_lessEquaNumber | Số lượng phần tử nhỏ hơn hoặc bằng 50K$ tại lúc đánh nhãn cho lá. |
| 11 | private bool \_result | Kết quả phân loại node lá. True: lớn hơn 50K$, false: nhỏ hơn 50K$ |
| 12 | private int \_pos | Vị trí node con trong mảng các tham chiếu của node cha. |
| 13 | private bool \_isPruned | Xác định trạng thái node đó đã được tỉa hay chưa. |
| 14 | private int \_greaterBranchNum | Số nhánh con lớn hơn 50K$ của một node |
| 15 | private int \_lessEqualBranchNum | Số nhánh con nhỏ hơn hoặc bằng 50K$ của một node |

1. **Phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public Node(DiscreteIndex indexOfChild) | Phương thức khởi tạo. Tham số đầu vào là một object discerete index dùng để xác định vị trí node con ứng với thuộc tính rời rạc. |
| 2 | public void MakeLabel(AttributeCase attr, double? threshold, double numberCase, double greaterNum, double lessEqualNum) | Đánh nhãn cho một node. Tham số đầu vào: Nhãn là một attribute nào đó, giá trị threshold ứng với thuộc tính liên tục, numbercase: số lượng phần tử xét, số lượng phần tử lớn hơn, số lượng phẩn tử nhỏ hơn. |
| 3 | public void SetLeaf(bool result, double numberCase, double numberError, double greaterNum, double lessEqualNum) | Xét trạng thái một node thành trạng thái node lá. |
| 4 | public bool Classify(Case ca) | Phân loại một trường hợp sau khi cây được dựng xong. |
| 5 | public Node GetChildNode(AttributeCase attr, object value) | Lấy một node con ứng với một attribute nào đó có value cụ thể. |

1. **Thuộc tính.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public AttributeCase Label  {  get { return this.\_label; }  set { this.\_label = value; }  } |  |
| 2 | public bool IsLeaf  {  get { return this.\_isLeaf; }  set { this.\_isLeaf = value; }  } |  |
| 3 | public double NumberCase  {  get { return this.\_numberCase; }  set { this.\_numberCase = value; }  } |  |
| 4 | public double NumberError  {  get { return this.\_numberError; }  set { this.\_numberError = value;}  } |  |
| 5 | public double GreaterNumber  {  get { return \_greaterNumber; }  set { \_greaterNumber = value; }  } |  |
| 6 | public double LessEquaNumber  {  get { return \_lessEquaNumber; }  set { \_lessEquaNumber = value; }  } |  |
| 7 | public bool Result  {  get  } | Lấy trạng thái phân loại của một node xem nó thuộc nhóm lớn hơn hay nhỏ hơn 50K$. Chỉ áp dụng với node lá, nếu một node không phải là node lá thì sẽ trả lại excepition |
| 8 | public int NumberChilds  {  get  } | Lấy số lượng node con. |
| 9 | public Node Parent  {  get { return this.\_parent; }  set { this.\_parent = value; }  } | Lấy node cha của một node |
| 10 | public Node[] ListChild  {  get { return this.\_listChild; }  set { this.\_listChild = value; }  } | Lấy mảng các node con |
| 11 | public int Pos  {  get { return this.\_pos; }  set { this.\_pos = value; }  } |  |
| 12 | public bool IsPruned  {  get { return \_isPruned; }  set { \_isPruned = value; }  } |  |
| 13 | public int GreaterBranchNum  {  get { return \_greaterBranchNum; }  set { \_greaterBranchNum = value;}  } | Số nhánh có kết quả lớn hơn 50K$ |
| 14 | public int LessEqualBranchNum  {  get { return \_lessEqualBranchNum; }  set { \_lessEqualBranchNum = value;}  } | Số nhánh có kết quà nhỏ hơn hoặc bằng. |

## C45.

Lớp C45 được cài đặt để thực hiện xây dựng cầy quyết định. Phương thức có vài trò quan trọng nhất là phương thức makeTree. Trong C45 còn chứa phương thức tỉa cây nhằm tăng độ chính xác của cây quyết định. Việc cài đặt phương thức make tree với bộ dữ liệu Adult có một số chú ý đặc biệt:

* Các thuộc tính liên tục như Fnlwgt, CapitalGain…luôn phải tính threshold do đó luôn cần sắp xếp và tính toán. Điều này làm ảnh hưởng rất lớn tới hiệu năng của chương trình.
* Cách lựa chọn threshold có thể ảnh hưởng nghiêm trọng tới hình dạng của cây quyết định và thậm chí có thể làm chương trình bị tràn bộ đệm do lời gọi đệ quy quá lớn, cây quyết định có quá nhiều node.

Các vấn đề trên được giải quyết như sau:

* Đối với vấn đề thứ nhất: việc sắp xếp lại để tìm threshold sau mỗi lần data set bị chia tách thành các subset là hoàn toàn không tránh khỏi. Nhưng dựa trên kinh nghiệm để làm giảm khối lượng tính toán đi phần nào. Cụ thể qua quá trình quan sát thì các thuộc tính liên tục như CapitalGain hay Fnlwgt luôn có có Information Gain lớn do đó thường được chọn làm node có độ sâu nông, ở gần gốc. Đó cũng là giai đoạn tính toán phức tạp nhất do lúc đó bộ dữ liệu còn rất lớn. Nhưng trong thực tế thì luôn có hiện tượng, nếu giả dụ tại một node CapitalGian được lựa chọn làm thuốc tính phân nhánh thì nhiều nhánh con ngay sau vẫn chọn CapitalGian làm thuộc tính phân nhánh. Trong trường hợp đó khi xét các thuộc tính ở nhánh con thì có thể không cần sắp xếp lại để tính threshold ứng với thuộc tính CapitalGian đó. Như thế làm giảm đi phần nào số lượng tính toán. Tuy không cải thiện đáng kể, nhưng việc xử lý với dữ liệu lớn thì nên tính kiệm tối đa tài nguyên tính toán của hệ thống.
* Đối với vấn đề thứ hai: việc sử dụng GainRatio được nhiều nhà khoa học thống kê và cho kết quả tích cực hơn là so với Gain. Cây dựng nên có số lượng node ít hơn và có độ chính xác cao hơn. Nhưng đối với thuộc tính liên tục, sau khi chia khoảng rồi sử dụng GainRatio để chọn threshold tốt nhất trong bộ dữ liệu Adult lại không phù hợp. Với thuộc tính có độ biến thiên mạnh như Fnlwgt thì việc dùng Gain Ratio dẫn đến mỗi lần lựa chọn được một threshold thì bộ dữ liệu bị chia thành hai bộ dữ liệu lệch. Một susbset thậm chí chỉ có một phần tử và subset còn lại thì có n-1 phần tử. Làm như thế cây quyết định sẽ lệch rất mạnh. Và do Fnlwgt có sự khác biệt rất lớn nên số lần gọi đệ quy sẽ là rất nhiều. Điều này sẽ dẫn tới tràn bộ đệm. Cách giải quyết ở đây là sử dụng Gain khi đó bộ dữ liệu sẽ được chia cân bằng hơn và do đó tránh được hiện tượng bị tràn bộ đệm.

1. **Private Fields**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private DiscreteIndex \_indexChild | Chứa chỉ số index của các giá trị rời rạc. |
| 2 | private Data \_dataSet | Data set của để phân tích. |
| 3 | private Node \_root | Node gốc của cây quyết định. |
| 4 | private bool \_isContructed | Xác định trạng thái cây đã được xây dựng hay chưa |
| 5 | private FileProcessing \_file | Đối tượng file để ghi dữ liệu ra nhằm phục vụ kiểm tra khi cần. |
| 6 | private double \_infoT | Lượng tin trung bình. |
| 7 | private double \_totalCase | Tổng số trường hợp |
| 8 | private double \_totalMissingCase | Tổng số trường hợp bị missing |
| 9 | private bool \_firtAgeThreshold | Lần đầu tính threshold của thuộc tính Age |
| 10 | private bool \_firtEduNumThreshold | Lần đầu tính threshold của thuộc tính EduNum |
| 11 | private bool \_firtFnlwgtThreshold | Lần đầu tính threshold của thuộc tính Fnlwgt |
| 12 | private bool  \_firtCapitalGainThreshold | Lần đầu tính threshold của thuộc tính CapitalGain |
| 13 | private bool  \_firtCapitalLossThreshold | Lần đầu tính threshold của thuộc tính CapitalLoss |
| 14 | private bool  \_firtHourPerWeekThreshold | Lần đầu tính threshold của thuộc tính HoursPerWeek |
| 15 | private Case[] \_validateCases | Các dữ liệu validate |
| 16 | private double \_maxAccurate | Độ chính xác cực đại của cây khi tiến hành tỉa với dữ liệu validate |

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý Nghĩa** |
| 1 | Public C45(Data dataSet,  DiscreteIndex indexChild) | Bộ khơi tạo |
| 2 | public void ConstructTree() | Xây dựng cây |
| 3 | public bool Classify(Case ca) | Phân loại sau khi cây xây dựng xong |
| 4 | public int  ComputingTree(Node root) | Đâu là thủ tục đệ quy qua tất cả các node để tiến nhánh tính số nhánh con của một node có kết quả phân loại lớn hơn 50K$ và số nhánh con có kết quả nhỏ hơn 50K$ |
| 5 | public void OptimizeTree(Case[]  validateData) | Thủ tục tiến hành tối ưu cây bằng phương pháp tỉa cây sử dụng dữ liệu validate. |
| 7 | public double Accuracy(Case[]  testCases) | Tính toán độ chính xác của cây. |
| 8 | private void makeTree(Node node, List<AttributeCase>listRemainAttr, int numberRemainningDiscreteAttr, Data subset, double[]thresholdArr,  params Condition[] listCondition) | Phương thức xây dựng cây. Các tham số đầu vào: Node để xét tách nhánh, các thuộc tính rời rạc còn lại chưa xét, số lượng thuộc tính rời rạc còn, subset, mảng các threshold của thuộc tính liên tục mà lớp cha được chọn đánh nhãn(nhằm tối ưu), các điều kiện để lọc dữ liệu. |
| 9 | private double  chooseThreshold(Data subset,  AttributeCase attr,  double[] threshold, int n, ref  double infox, ref double  splitInfo, ref int posThres) | Chọn lựa ra một threshold tốt nhất ứng với một attribute liên tục. |
| 10 | private double  computeGainRation(StatictisObject[] so, double missingNum) | Tính toán GainRatio cho một thuộc tính rời rạc1 |
| 11 | private void deleteSubTree(Node  root) | Delete xóa sub tree của các node sau khi bị prunning |
| 12 | private void prunTree(Node root) | Tỉa cây. Thực hiện vừa tỉa vừa đánh giá độ chính xác của cây sau khi tỉa. |
| 13 | private double accuracy() | Tính toán độ chính xác. |

1. **Các thuộc tính.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public bool IsConstructed  {  get { return this.\_isContructed; }  } | Trạng thái cây được xây dựng hay chưa |
| 2 | public Node Root  { get { return \_root; }  set { \_root = value; }  } |  |

## Knowledge.

Lớp knowledge dùng để lưu các tri thức khai phá được từ dữ liệu trong các thuật toán C4.5, ID3, và NaiveBayes.

1. **Private Fields.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Fields** | **Ý nghĩa** |
| 1 | private Dictionary<AttributeCase,object>  \_mostCommonLabelGreatr | Dùng để lưu các nhãn phổ biến nhất ứng vỡi mỗi attribute cho trường hợp lớn hơn 50K$ |
| 2 | private Dictionary<AttributeCase, object> \_mostCommonLabelLessEqua | Dùng để lưu các nhãn phổ biến nhất ứng vỡi mỗi attribute cho trường hợp nhỏ hơn hoặc bằng 50K$ |
| 3 | private double \_numberTrainingCase | Số trường hợp training |
| 4 | private double \_numberValidateCase | Số trường hợp validate |
| 5 | private Node \_c45Root | Gốc trogn tường hợp lưu tri thức của C45 |
| 6 | private Algorithm \_algorithm | Loại thuật toán |

1. **Các phương thức.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Ý nghĩa** |
| 1 | public Knowledge(){ } | Khởi tạo mặc định |
| 2 | public Knowledge(Algorithm algorithm, Node root, Dictionary<AttributeCase, object> mostCommonLabelGreatr, Dictionary<AttributeCase, object> mostCommonLabelLessEqua) | Khởi tạo để lưu tri thức của thuật toán C45 |
| 3 | public void CleanData(params Case[] caseData) | Làm sạch dữ liệu |
| 4 | public double TestAccuracy(BackgroundWorker worker, params Case[] testCases) | Test độ chính xác. |
| 5 | public bool makeDecisionC45(Case ca) | Phân loại theo cây quyết định tạo ra từ thuật toán C4.5 |

1. **Các thuộc tính.**

|  |  |  |
| --- | --- | --- |
| **STT** | **Thuộc tính** | **Ý nghĩa** |
| 1 | public Dictionary<AttributeCase, object> MostCommonLabelGreatr  { get; set; }  } |  |
| 2 | public Dictionary<AttributeCase, object> MostCommonLabelLessEqua  {get;set;} |  |
| 3 | public Node C45Root  {  get { return this.\_c45Root; }  set { this.\_c45Root = value; }  } |  |
| 4 | public double NumberTrainingCase  {  get { return \_numberTrainingCase; }  set { \_numberTrainingCase = value;}  } |  |
| 5 | public double NumberValidateCase  {  get { return \_numberValidateCase; }  set { \_numberValidateCase = value; }  } |  |

## Probability.

## ProbabilityTable.

## ContinuousHandler

Sử dụng để rời rạc hóa các thuộc tính liên tục.

## NodeID3

Sử dụng để lưu Node của TreeID3.

## TreeID3.

Sử dụng để tạo cây và đánh nhãn cho 1 trường hợp đưa vào.

## AttributeDetail.

## NaiveBayes.

## HoursPWComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Hours Per Week.

## FnlwgtComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Fnlwgt.

## EduNumComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Edu Num

## CapitalLossComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Capital Loss.

## CapitalGainComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Capital Gain.

## AgeComparer.

Dùng để so sánh các thuộc tính HoursPerWeek của các case khi áp dụng thuật toán Quick Sort sắp xếp mảng các case theo thuộc tính Age.