TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Algoritmus AQ11 v jazyku C#

2024 **Ivan Tkachenko**

Obsah

Zoznam tabuliek

1. Teoretický popis algoritmu

Algoritmus AQ realizuje formu učenia s učiteľom. Vzhľadom na množinu pozitívnych udalostí (príkladov) P, množinu negatívnych udalostí N algoritmus generuje pokrytie C pozostávajúce z komplexov, t. j. spojenia atribučných podmienok, ktoré pokrývajú všetky udalosti z P a žiadne udalosti z N. Algoritmus začína zameraním sa na jednu pozitívnu udalosť e, ktorá sa nazýva semienko, ktoré sa potom zovšeobecní vytvorením všetkých maximálne všeobecných komplexov, ktoré pokrývajú semienko a nepokrývajú žiadne udalosti z N. Táto maximálne všeobecná množina komplexov sa nazýva obálka G(e, N). Všetky udalosti pokryté komplexom c sa odstránia z P. Ak je množina príkladov P prázdna, vráti sa obal C; v opačnom prípade sa z P vyberie ďalší semienko a operácia sa opakuje, kým nie je P prázdna. Kvôli jednoduchosti opisu tu môžeme predpokladať, že komplex je ekvivalentný pravidlu.[1]

2. Popis postupu a jednotlivých funkcií

2.1. Data a Preprocessor

Triedy Data a Preprocessor pracujú spoločne na načítaní údajov z daného súboru údajov, ich transformácii do štruktúry, s ktorou je algoritmus schopný pracovať, a ich uložení v inštancii Data pre budúce použitie algoritmom.

2.1.1. Preprocessor

Preprocesor zodpovedný najmä za čítanie a transformáciu údajov zo súboru údajov. Preprocesor je statická trieda, takže nemôže ukladať údaje a inštanciu triedy nemožno vytvoriť.

Metódy:

2.1.1.1. public static List<string> GetHeaders(string datasetPath)

Metóda preberá ako vstup reťazcovú cestu k súboru údajov. Pomocou balíka CsvHelper načíta názvy hlavičiek zo súboru údajov. Uloží ich do zoznamu reťazcov, ktorý sa potom vráti.

2.1.1.2. public static List<List<string>> GetRecordsString(string datasetPath)

Metóda preberá ako vstup reťazcovú cestu k súboru údajov. Pomocou balíka CsvHelper načíta každý stĺpec atribútu zo súboru údajov a na začiatok pridá aj stĺpec Id. Uloží ich do dvojrozmerného zoznamu, v ktorom každý uložený zoznam reťazcov predstavuje stĺpec hodnôt jednotlivých atribútov. Tento dvojrozmerný zoznam je návratovou hodnotou.

2.1.1.3. public static List<List<string>> TransformRecordsIntoRecordsIndividual(List<List<string>> recordsString)

Metóda transformuje zoznam stĺpcov atribútov, ktorý sme získali predchádzajúcou metódou, na dvojrozmerný zoznam, kde každý prvok je zoznam reťazcov, ktorý predstavuje jeden záznam zo súboru údajov.

2.1.1.4. public static Dictionary<string, List<string>> CreateAttributesDictionary(List<string> headers, List<List<string>> recordsString)

Metóda preberá zoznam hlavičiek a zoznam stĺpcov - atribútov zo súboru údajov ako vstupy a vytvára užitočný slovník. Pomocou tohto slovníka je možné previesť akúkoľvek reťazcovú hodnotu atribútu na číselnú hodnotu. Slovník obsahuje názvy jednotlivých atribútov ako kľúčové hodnoty. Pre každý atribút existuje zoznam hodnôt atribútov. Hodnota na indexe 0 je prázdna, takže žiadna hodnota atribútu nemôže mať index 0. Pomocou toho je možné previesť akúkoľvek reťazcovú hodnotu atribútu na číselnú hodnotu indexu atribútu v príslušnom zozname v slovníku a naopak.

A s tým súvisí aj negácia hodnôt atribútov (napríklad Hair : Black -> Hair : Not Black), ak má Black index 2, potom negácia tejto hodnoty bude -2. Výstupom je tento slovník.

2.1.1.5. public static List<List<int>> ConvertRecordsToInt(List<List<string>> recordsIndividual, Dictionary<string, List<string>> attributesDict, List<string> headers)

Metóda používa slovník atribútov a hodnôt na prevod reťazcových hodnôt jednotlivých záznamov na číselné. Ako vstupy uvádzame zoznam jednotlivých záznamov, predtým vytvorený slovník a zoznam hlavičiek. Ako výstup dostaneme zoznam, ktorý obsahuje jednotlivé záznamy(zoznam), ktoré obsahujú číselné hodnoty atribútov.

Napríklad:

4 | Female | 61-80 | Yes | No | High | never smoked | Yes |

Prevedie sa na:

4 | 2 | 1 | 2 | 2 | 3 | 2 | 1 |

2.1.1.6. public static List<List<int>> GetRecordsIndividualNumerical(List<List<string>> recordsString, Dictionary<string, List<string>> attributesDict, List<string> headers)

Užitočná metóda na transformáciu zoznamu, ktorý obsahuje stĺpce - atribúty súboru údajov, na zoznam s jednotlivými záznamami, ktoré obsahujú číselné hodnoty, v jednom volaní. Metóda len využíva predchádzajúce metódy TransformRecordsIntoRecordsIndividual a ConvertRecordsToInt.

Ako výstup dostaneme zoznam, ktorý obsahuje jednotlivé záznamy(zoznam), ktoré obsahujú číselné hodnoty atribútov.

2.1.1.7. public static List<List<int>> GetPositiveRecords(List<List<int>> recordsNumerical, Dictionary<string, List<string>> attributesDict, List<string> headers, string posAttributeName, string posAttributeValue)

Metóda extrahuje všetky pozitívne záznamy z daného zoznamu záznamov. Ako vstup uvádzame zoznam záznamov, slovník hodnôt atribútu, názov reťazca cieľového atribútu a pozitívnu hodnotu reťazca atribútu. Jednoducho len prejde každý záznam a vezme každý, ktorý má cieľovú hodnotu. Návratová hodnota je zoznam pozitívnych záznamov.

2.1.1.8. public static List<List<int>> GetNegativeRecords(List<List<int>> recordsNumerical, Dictionary<string, List<string>> attributesDict, List<string> headers, string posAttributeName, string posAttributeValue)

Funguje takmer identicky ako metóda 2.1.1.7, ale berie len záznamy, ktoré neobsahujú cieľovú hodnotu. Vráti zoznam zaznamenaných záznamov.

2.1.2. Data

Trieda Data je zodpovedná hlavne za volanie špecifických metód Preprocesora na danej dátovej množine a uloženie výsledku do inštancie. V podstate po zavolaní správneho konštruktora alebo metódy ReadDataset musíte mať objekt Data, ktorý je pripravený na prácu s algoritmom. Obsahuje tiež metódy pre formátované zobrazenie uložených hodnôt v konzole.

Súkromné atribúty triedy:

- string datasetPath – Reťazec cesty k súboru údajov.
- List<List<string>> recordsString – Zoznam záznamov s reťazcovymi hodnotami.
- string targetAttributeName – Reťazec názvu cieľového atribútu.
- string targetAttributeValue – Cieľová reťazcova hodnota cieľového atribútu.

Verejné atribúty triedy:

- List<string> Headers – Zoznam hlavičiek súboru údajov.
- Dictionary<string, List<string>> AttributesDict – Slovník hodnôt atribútov.
- List<List<int>> Records – Zoznam záznamov s číselnými hodnotami.
- List<List<int>> PositiveRecords – Zoznam pozitívnych záznamov.
- List<List<int>> NegativeRecords – Zoznam negatívnych záznamov.
- int NumberOfColumns – Počet stĺpcov v množine údajov + stĺpec id.
- int NumberOfRecords – Počet záznamov v množine údajov.

Metódy:

2.1.2.1. public Data(string datasetPath, string targetAttributeName, string targetAttributeValue)

Konštruktor pre trénovacie dáta. Ako vstupy berie cestu k súboru údajov, názov cieľového atribútu a cieľovú hodnotu atribútu. Volá metódu 2.1.2.3 ReadDataset.

2.1.2.2. public Data(string datasetPath, string targetAttributeName, string targetAttributeValue, Dictionary<string, List<string>> attributesDict)

Konštruktor pre vyhodnocovacie dáta. Ako vstupy berie cestu k množine údajov, názov cieľového atribútu, cieľovú hodnotu atribútu a špecifický slovník atribútov, ktorý musí použiť na konverziu údajov z reťazcov na číselné. Volá metódu 2.1.2.4 ReadDatasetWithExistingDictionary.

2.1.2.3. public void ReadDataset()

Volá metódy preprocesora a ukladá výsledky do inštancie Data. Vytvára svoj vlastný slovník atribútov a používa sa na čítanie trenovacich súborov údajov. Tiež ukladá počet atribútov a záznamov súboru údajov.

2.1.2.4. public void ReadDatasetWithExistingDictionary()

Má podobnú funkčnosť ako predchádzajúca metóda 2.1.2.3, ale nevytvára svoj vlastný slovník, ale používa existujúci daný slovník na prevod hodnôt reťazcov na číselné.

2.1.2.5. Metódy zobrazenia údajov do konzoly:

Jednoduché metódy, ktoré sa používaju na zobrazenie rôznych uložených hodnôt inštancie údajov v konzole. Veľmi užitočné pre fázu vývoja a ladenia.

- public void DisplayHeaders() - Metóda zobrazuje zoznam hlavičiek aktuálne uložených v inštancii.
- public void DisplayRecordsString() - Metóda zobrazí zoznam záznamov s reťazcovymi hodnotami aktuálne uloženými v inštancii.
- public void DisplayNumericalRecords(List<List<int>> records) - Metóda zobrazí zoznam záznamov s číselnými hodnotami, zadaný ako parameter.
- public void DisplayNumericalRecords() - Metóda zobrazí zoznam záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.
- public void DisplayPositiveRecords() - Metóda zobrazí zoznam pozitívnych záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.
- public void DisplayNegativeRecords() - Metóda zobrazí zoznam negatívnych záznamov s číselnými hodnotami, ktorý je aktuálne uložený v inštancii.

2.2. AQ11

Trieda AQ11 je trieda, ktorá predstavuje algoritmus.

Verejné atribúty triedy:

- public int NumberOfColumns – Počet atributov v trenovacich datoch.
- int NumberOfRecords – Počet záznamov v trenovacich datoch.
- public Data LocalTrainingData – Inštancia trénovacich Data uložená v inštancii algoritmu.
- public Data EvaluationData – Inštancia Data pre hodnotenie algoritmu uložená v inštancii algoritmu.
- public List<List<List<int?>>> fullStar – Zoznam, ktorý predstavuje celú obálku G, (každý pozitívný záznam voči každemu negatívnemu záznamu). Obálka obsahuje negácie hodnôt atribútov.
- public List<List<List<List<int?>>>> PositiveFullStar – Zoznam, ktorý predstavuje celú obálku G, (každý pozitívný záznam voči každemu negatívnemu záznamu). Obálka obsahuje pozitivne hodnôty atribútov.
- public int NumberOfTP – Atribút obsahuje počet True Positive prípadov, uložený po vyhodnotení.
- public int NumberOfFP – Atribút obsahuje počet False Positive prípadov, uložený po vyhodnotení.
- public int NumberOfFN – Atribút obsahuje počet False Negative prípadov, uložený po vyhodnotení.
- public int NumberOfTN – Atribút obsahuje počet True Negative prípadov, uložený po vyhodnotení.
- public float Precision – Atribút obsahuje vypočítané Precision po vyhodnotení.
- public float Recall – Atribút obsahuje vypočítané Recall po vyhodnotení.
- public float F1 – Atribút obsahuje vypočítané F1 po vyhodnotení.
- public float Accuracy – Atribút obsahuje vypočítané Accuracy po vyhodnotení.

2.2.1. Metódy trénovania a tvorby pravidla

2.2.1.1. public AQ11() a public AQ11(Data data)

Konštruktory inštancie algoritmu. Druhý berie ako vstup trenovacie dáta v inštancii Data. Volá metódu 2.2.1.2 SetData.

2.2.1.2. public void SetData(Data data)

Metóda berie inštanciu Data ako vstup a ukladá ju do inštancie AQ11 ako trenovaciu. Tiež vypočíta počet atribútov a počet záznamov a uloží ich.

2.2.1.3. public void SetEvaluationData(Data data)

Metóda berie inštanciu dát ako vstup a ukladá ju do inštancie AQ11 ako vyhodnocovacie dáta.

2.2.1.4. public void ApplyAlgorithmOnData(bool display=false)

Metóda, ktorá aplikuje algoritmické metódy na uložené trénovacie dáta. Výsledné fullStar a PositiveFullStar sú uložené v inštancii AQ11. Hodnota parametru display určuje, či sa má výsledné pravidlo zobraziť v konzole.

2.2.1.5. public void ApplyAlgorithmOnData(Data data, bool display=false)

Funguje rovnako ako predchádzajúca metóda 2.2.1.4, ale teraz s trénovacimi datmi, ktoré sú uvedené ako parameter. Hodnota parametru display určuje, či sa má výsledné pravidlo zobraziť v konzole.

2.2.1.6. public void DisplayResultingRule()

Metóda používa metódu DisplayPositiveFullStarAsRule a dáva jej uložený PositiveFullStar ako vstupný parameter. Metóda zobrazuje PositiveFullStar ako pravidlo v konzole, ktoré je pre človeka zrozumiteľná.

2.2.1.7. public List<int?> CreatePartialStarDisjunction(List<int> positiveRecord, List<int> negativeRecord)

Metóda na vytvorenie čiastočnej obálky-disjunkcie (obálka G - pozitívny príklad voči kontraprík ladu) z jedného pozitívneho záznamu a jedného negatívneho záznamu. Vstupy sú jeden pozitívny záznam a jeden negatívny záznam. Výstupom je čiastočná obálka-disjunkcia. Disjunkcia neobsahuje žiadne informácie o ID záznamov alebo cieľovej hodnote.

Príklad

Pozitívny záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Negatívny záznam: (| 21 | 1 | 2 | 2 | 2 | 3 | 1 | 2 |)

Výsledná disjunkcia: (| -1 | -2 | -2 | | -3 | |) => (Atribút0: nie 1) ALEBO (Atribút1: nie 2) ALEBO (Atribút2: nie 2) ALEBO (Atribút4: nie 3)

2.2.1.8. public void DisplayPartialStarDisjunction(List<int?> partialStarDisjunction)

Metóda zobrazenia danej ako parameter obálky (obálka G - pozitívny príklad voči kontraprík ladu).

2.2.1.9. public List<List<int?>> CreatePartialStarConjunction(List<int> positiveRecord, List<List<int>> negativeRecords)

Metóda na vytvorenie čiastočnej obálky-konjunkcie (obálka G - pozitívny príklad voči všetkým kontraprík ladom) z jedného pozitívneho záznamu a všetkych negatívnych záznamov. V podstate používa metódu 2.2.1.7 CreatePartialStarDisjunction na každý negatívny záznam a ukladá výsledky do zoznamu. Výstupom je čiastočná obálka-konjunkcia.

Príklad

Pozitívny záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Negatívne záznamy:

(| 21 | 1 | 2 | 2 | 2 | 3 | 1 | 2 |)

(| 29 | 1 | 4 | 1 | 2 | 2 | 3 | 2 |)

Výsledná konjunkcia: (| -1 | -2 | -2 | | -3 | |), (| -1 | | | | | -3 |) == [(Atribút0: nie 1) ALEBO (Atribút1: nie 2) ALEBO (Atribút2: nie 2) ALEBO (Atribút4: nie 3)] A [(Atribút0: nie 1) ALEBO (Atribút5: nie 3)]

2.2.1.10. public void DisplayPartialStarConjunction(List<List<int?>> partialStarConjunction)

Metóda zobrazenia danej ako parameter obálky (obálka G - pozitívny príklad voči všetkým kontraprík ladom).

2.2.1.11. private bool CanBeAbsorbed(List<int?> primaryStar, List<int?> secondaryStar)

Metóda na kontrolu, či sekundárna obláka môže byť absorbovaná pomocou primárnej obálky pri použití absorpčného zákona. Obálky sú uvedené ako parametre. (obálka G - pozitívny príklad voči kontraprík ladu). Výstup je boolovská hodnota.

Príklad 1

Primárna obálka: (| -1 | | | | | -3 |)

Sekundárna obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: False

Príklad 2

Primárna obálka: (| -1 | | -2 | | -3 | |)

Sekundárna obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: True

2.2.1.12. public List<List<int?>> ApplyAbsorptionLawOnConjunction(List<List<int?>> partialStarConjucntion)

Metóda aplikácie absorpčného zákona na čiastočnú obálku-konjunkciu. Metóda odstraňuje akúkoľvek čiastočnú obálku-disjunkciu, ktorá môže byť absorbovaná z čiastočnej obálky-konjunkcie. Používa metódu 2.2.1.11 CanBeAbsorbed s každou kombináciou disjunkcií na kontrolu, či je možné sekundárnu disjunkciu absorbovať. Výstupom je zmenená čiastočná obálka-konjunkcia.

Príklad

Obálka-konjunkcia: (| -1 | -2 | -2 | | -3 | |), (| -1 | | | | | -3 |), (| -1 | | -2 | | -3 | |)

Výsledná obálka-konjunkcia: (| -1 | | | | | -3 |), (| -1 | | -2 | | -3 | |)

Vysvetlenie: obálka (| -1 | -2 | -2 | | -3 | |) bola absorbovana obálkou (| -1 | | -2 | | -3 | |) ako bolo preukázané v príklade 2 metódy 2.2.1.11.

2.2.1.13. public bool IsRecordCoveredByDisjunction(List<int> record, List<int?> disjunction)

Metóda kontroly, či konkrétny daný záznam je pokrytý danou čiastočnou obálkou-disjunkciou. Vstupy sú špecifický záznam a disjunkcia, ktorú chceme skontrolovať. Výstup je boolovská hodnota.

Príklad 1

Záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Obálka: (| -1 | -2 | -2 | | -3 | |)

Výsledok: True

Príklad 2

Záznam: (| 15 | 2 | 4 | 1 | 2 | 2 | 1 | 1 |)

Obálka: (| -2 | -4 | -2 | | -3 | |)

Výsledok: False

2.2.1.14. public void DisplayNotCoverage(List<int> record, List<int?> disjunction)

Metóda zobrazenia konkrétneho daného záznamu, ktorý nebol pokrytý danou obálkovou disjunkciou. Ako vstupy berie jeden záznam a jednu disjunkciu. Hlavne veľmi užitočné pri ladení.

2.2.1.15. public bool IsRecordCoveredByConjunction(List<int> record, List<List<int?>> conjunction)

Metóda kontroly, či konkrétny daný záznam je pokrytý danou čiastkovou obálkou-konjunkciou. Metóda v podstate iteruje konjunkciou (zoznam disjunkcii) a pomocou metódy 2.2.1.13 IsRecordCoveredByDisjunction kontroluje, či záznam pokrýva každá disjunkcia. Vstupy sú jeden záznam a obálka-konjunkcia. Výstup je boolovská hodnota.

2.2.1.16. public List<List<int>> SelectCoveredRecords(List<List<int>> records, List<List<int?>> conjunction)

Metóda na výber záznamov, ktoré sú pokryté obálkou-konjunkciou, zo zoznamu záznamov. Pokryté záznamy sú uložené v druhom zozname. Vstupmi sú zoznam záznamov a obálka-konjunkcia. Výstupom je zoznam pokrytých záznamov. V podstate používa metódu 2.2.1.15 IsRecordCoveredByConjunction na každý záznam v zozname a berie len tie, ktoré sú pokryté.

2.2.1.17. public List<List<List<int?>>> CreateFullStarDisjunction(List<List<int>> positiveRecords, List<List<int>> negativeRecords)

Metóda na vytvorenie plnej obálky-disjunkcie. (obálka G - všetky pozitívne príklady voči všetkym kontraprikladom). Vstupmi sú zoznam pozitívnych záznamov a zoznam negatívnych záznamov. Výstup je plna obálka-disjunkcia. Metóda vytvorí spojku pre každý záznam, ktorý nie je pokrytý žiadnou zo obálok v zozname, kým nie sú všetky záznamy pokryté zoznamom obálok (zoznam obálok v tomto pripade je obálka G - všetky pozitívne príklady voči všetkym kontraprikladom).

2.2.1.18. public void DisplayFullStarDisjunction(List<List<List<int?>>> fullStar)

Metóda zobrazenia plnej obálky-disjunkcie (vystupnej z methody 2.2.1.17).

2.2.1.19. public List<List<int?>> TransformNegationsIntoDisjunctions(List<int?> negations)

Metóda transformuje každú negáciu v čiastočnej obál-disjunkcii na disjunkcie obsahujúce pozitivne hodnoty. Vstup je disjunkcia (zoznam), ktorá obsahuje negatívne hodnoty. Výstupom je zoznam samostatných pozitívnych disjunkcií pre každý atribút.

Príklad

Obálka: (| -2 | | -3 |)

A napríklad pre atribút 0 sú jediné možné hodnoty 1,2. A pre atribút 2 možné hodnoty sú 1,2,3,4.

To znamená, že pre atribút 0 sa hodnota -2 premení na disjunkciu (zoznam), ktorá bude obsahovať jednu hodnotu: (| 1 |).

Pre atribút 1 disjunkcia (zoznam) bude prázdna: ( ).

Pre atribút 2 disjunkcia (zoznam) bude obsahovať všetky kladné hodnoty okrem tej, ktorá bola negovaná. Disjunkcia (zoznam) bude vyzerať takto: (| 1 | 2 | 4 |).

2.2.1.20. public void DisplayPositiveDisjunctions(List<List<int?>> disjunctions)

Metóda na zobrazenie zoznamu pozitívnych disjunkcií. Vstup je disjunkcia (zoznam), pričom každý prvok je disjunkcia (zoznam) s pozitivnymi prvkami.

2.2.1.21. public List<List<List<int?>>> TransformNegativeConjunctionIntoPositive(List<List<int?>> conjucntion)

Metóda, ktorá používa metódu 2.2.1.19 TransformNegationsIntoDisjunctions na transformáciu všetkych disjunkcii s negativnymi hodnotami v spojke na disjunkcie s pozitivnymi hodnotami. Vstup je konjunkcia (zoznam zoznamov), ktoré chceme transformovať.

2.2.1.22. public void DisplayPositiveConjunction(List<List<List<int?>>> positiveConjucntion)

Metóda zobrazenia konjunkcie s pozitivnymi disjunkciami v konzole. Ako vstup používa konjunkciu (zoznam disjunkcií s pozitívnymi prvkami).

2.2.1.23. public List<List<List<List<int?>>>> TransformFullStarToNumericalPositiveFullStar(List<List<List<int?>>> fullStar)

Metóda transformácie prvkov plnej obálky-disjunkcie na pozitívne. Výstup je plná obálka-disjunkcia iba s pozitivnymi hodnotami. Vstup je fullStar (plna obálka ako vystup z metódy 2.2.1.17 CreateFullStarDisjunction) ktorá obsahuje iba negatívne hodnoty.

2.2.1.24. public void DisplayPositiveFullStar(List<List<List<List<int?>>>> positiveStar)

Metóda zobrazenia plnej obálky-disjunkcie ktoru sme získali z metódy 2.2.1.23.

2.2.1.25. public void DisplayPositiveFullStarAsRule(List<List<List<List<int?>>>> positiveStar)

Metóda na zobrazenie úplnej obálky-disjunkcie, ktorá obsahuje iba pozitivne hodnoty ako formátované pravidlo napísané v texte, ktorému človek rozumie.

2.2.2. Metódy hodnotenia

2.2.2.1. public bool IsRecordCoveredByPositiveDisjunction(List<int> record, List<List<int?>> disjunction)

Metóda kontroly či je záznam pokrytý jedinou disjunkciou s pozitívnymi hodnotami. Vráti boolovskú hodnotu.

2.2.2.2. public bool IsRecordCoveredByPositiveConjunction(List<int> record, List<List<List<int?>>> conjunction)

Metóda kontroly, či je záznam pokrytý konjunkciou, ktorá obsahuje disjunkcie s pozitívnymi hodnotami. V podstate používa metódu 2.2.2.1 na každu disjunkciu. Vráti boolovskú hodnotu.

2.2.2.3. public bool IsRecordCoveredByPositiveFullStar(List<int> record, List<List<List<List<int?>>>> positiveStar)

Metóda kontroly, či je záznam pokrytý celou pozitívnou obálkou. Vráti boolovskú hodnotu. V zásade používa metódu 2.2.2.2 na každu konjunkciu a ak aspoň jedna konjunkcia vráti hodnotu true, záznam je pokrytý plnou obálkou.

2.2.2.4. public bool IsRecordCoveredByPositiveFullStar(List<int> record)

Metóda funguje rovnako ako metóda 2.2.2.3, ale používa iba PositiveFullStar, ktorá je uložená v inštancii algoritmu.

2.2.2.5. public int GetNumberOfTruePositives(List<List<int>> positiveRecords, List<List<List<List<int?>>>> positiveStar)

Metóda vypočíta počet True Positive výsledkov vzhľadom na vstupné pozitívne záznamy a pozitívnu celu obálku. Používa metódu 2.2.2.3 pre každý záznam. Na začiatku sa číslo TP rovná 0. Ak algoritmus klasifikuje záznam ako pozitívny, potom sa počet TP zvýši.

2.2.2.6. public int GetNumberOfTrueNegatives(List<List<int>> negativeRecords, List<List<List<List<int?>>>> positiveStar)

Metóda vypočíta počet True Neagtive výsledkov vzhľadom na vstupné negatívne záznamy a pozitívnu celu obálku. Používa metódu 2.2.2.3 pre každý záznam. Na začiatku sa číslo TN rovná počtu negatívnych záznamov. Ak algoritmus klasifikuje záznam ako pozitívny, potom sa počet TN zníži.

2.2.2.7. public int GetNumberOfFalsePositives(List<List<int>> negativeRecords, List<List<List<List<int?>>>> positiveStar)

Metóda vypočíta počet False Positive výsledkov vzhľadom na vstupné negatívne záznamy a pozitívnu celu obálku. Používa metódu 2.2.2.3 pre každý záznam. Na začiatku sa číslo FP rovná 0. Ak algoritmus klasifikuje záznam ako pozitívny, potom sa počet FP zvýši.

2.2.2.8. public int GetNumberOfFalseNegatives(List<List<int>> positiveRecords, List<List<List<List<int?>>>> positiveStar)

Metóda vypočíta počet False Negative výsledkov vzhľadom na vstupné pozitívne záznamy a pozitívnu celu obálku. Používa metódu 2.2.2.3 pre každý záznam. Na začiatku sa číslo FN rovná počtu pozitívnych záznamov. Ak algoritmus klasifikuje záznam ako pozitívny, potom sa počet FN zníži.

2.2.2.9. public void EvaluateAlgorithm(bool display = false)

Metóda počíta počet TP, FP, FN, TN a Precision, Recall, F1, Accuracy pre uložené EvaluationData v inštancii algoritmu. A uloží výsledky do inštancie algoritmu. Parameter "display" je zodpovedný za rozhodnutie, či sa výsledky zobrazia v konzole.

Vzorce pre metriky hodnotenia:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

2.2.2.10. public void EvaluateAlgorithm(Data evalData, bool display=false)

Metóda ukladá nové hodnotiace dáta zadané ako vstup. Potom zavolá metódu 2.2.2.9.

2.2.2.11. public void DisplayEvaluationResults()

Metóda na zobrazenie v konzole výsledkov hodnotenia, ktoré sú uložené v inštancie algoritmu.

2.3. Postup fungovania algoritmu

Keď je zavolaná metóda ApplyAlgorithmOnData, algoritmus najprv vytvorí plnú obálku G(P, N), ktorá pokrýva všetky pozitívne záznamy a nepokrýva žiadne negatívne. Na túto tvorbu používa metódy, ktoré sú popísané vyššie v texte. Najprv metódu 2.2.1.17 CreateFullStarDisjunction, ktorá používa metódy 2.2.1.9 CreatePartialStarConjunction a 2.2.1.12 ApplyAbsorptionLawOnConjunction na vytvorenie samostatných konjunkčných obálok G(p, N), ktoré pokrývajú jeden (alebo viac) pozitívnych záznamov a nepokrývajú žiadne negatívne záznamy. Metóda ApplyAbsorptionLawOnConjunction iba aplikuje absorpčný zákon na konjunkciu. Metóda CreatePartialStarConjunction používa metódu 2.2.1.7 CreatePartialStarDisjunction na vytvorenie samostatných disjunkčných obálok G(p, n), ktoré pokrývajú iba jeden pozitívny záznam a nepokrývajú jeden negatívny záznam.

Obálka vytvorená skôr obsahuje iba negacie hodnot. Teraz musíme premeniť negatívne hodnoty na pozitívne. Na tento účel používa metódu 2.2.1.23 TransformFullStarToNumericalPositiveFullStar, ktorá používa metódu 2.2.1.21 TransformNegativeConjunctionIntoPositive na transformáciu negatívnych hodnôt v spojkách na pozitívne hodnoty. Táto metóda používa metódu 2.2.1.19 TransformNegationsIntoDisjunctions na transformáciu každej disjunkcie s negatívnymi hodnotami na disjunkciu pozitívnych hodnôt.

Výsledná plná obálka je potom uložená v inštancii algoritmu a môže byť použitá metódou 2.2.1.6 DisplayResultingRule na zobrazenie výsledného pravidla v konzole. Uloženú celú obálku možno použiť aj na predpovedanie triedy záznamov (dany záznam je pozitívny alebo negatívny). Toto je popísané v bode 2.4.4.

2.4. Ako používať algoritmus

2.4.1. Čítanie a predspracovanie údajov

Napríklad máme 2 súbory údajov:

- stroke_dataset_sample.csv – na trenovanie.
- stroke_dataset_sample_evaluation.csv – na vyhodnotenie.

Najprv musíme načítať súbor údajov na trenovanie. Môžeme to urobiť pomocou tohto príkazu:

Data trainingData = new Data(datasetPath, "stroke", "Yes");

Ako parametre uvádzame úplnú cestu k súboru údajov ako reťazec, názov cieľového atribútu ako reťazec (v tomto prípade je to "stroke"), cieľovú hodnotu atribútu ako reťazec(v tomto prípade je to "Yes"). To znamená, že pomocou tejto inštancie údajov, algoritmus vytvorí pravidlo, ktoré bude pokrývať prípady, ktoré majú v cieľovom atribúte „stroke“ hodnotu „Yes“.

Potom môžeme použiť tieto metódy na zobrazenie hlavičiek súboru údajov a údajov v konzole:

trainingData.DisplayHeaders();

trainingData.DisplayRecordsString();

Dostaneme niečo také, ale ako úplný súbor údajov:

Id | gender | age | hypertension | heart_disease | avg_glucose_level | smoking_status | stroke |

0 | Male | 61-80 | No | Yes | Very High | formerly smoked | Yes |

1 | Female | 61-80 | No | No | Very High | never smoked | Yes |

2 | Male | 61-80 | No | Yes | Normal | never smoked | Yes |

3 | Female | 41-60 | No | No | High | smokes | Yes |

4 | Female | 61-80 | Yes | No | High | never smoked | Yes |

A ďalej...

2.4.2. Trenovanie algoritmu

Keď máme inštanciu trénovacích údajov, musíme vytvoriť inštanciu algoritmu AQ11.

AQ11 aq = new AQ11();

A použiť hlavnú trénovaciu metódu na trénovacich údajoch.

aq.ApplyAlgorithmOnData(trainingData, true);

Prvým parametrom sú trénovacie dáta. Druhý parameter je true, pretože chcem, aby sa výsledné pravidlo, ktoré algoritmus vytvorí, zobrazovalo v konzole (bližšie informácie nájdete v bode 2.2.1.4 a 2.2.1.5).

Výsledné pravidlo bude vyzerať takto, ale s najväčšou pravdepodobnosťou bude dlhšie:

==========

FINAL RULE:

==========

(age : 61-80) or (age : 80+) or (age : 21-40) or (age : 0-20) or

(hypertension : No) or

(heart_disease : Yes)

AND

(age : 61-80) or (age : 41-60) or (age : 80+) or (age : 21-40) or

(heart_disease : Yes) or

(avg_glucose_level : Very High) or (avg_glucose_level : Normal) or (avg_glucose_level : Low) or

(smoking_status : formerly smoked) or (smoking_status : never smoked) or (smoking_status : smokes)

AND

A ďalej...

2.4.3. Vyhodnotenie algoritmu

Keď je algoritmus natrénovaný, musíme načítať vyhodnocovacie dáta.

Data evaluationData =

new Data(evaluationPath, "stroke", "Yes", aq.LocalTrainingData.AttributesDict);

Prvé 3 parametre fungujú úplne rovnako ako v bode 2.3.1. Parameter 4 je atribútový slovník z existujúcich tréningových dát uložených v inštancii AQ11. Používa sa takto, aby preprocesor mohol konvertovať reťazcové dáta na numerické rovnakým spôsobom, ako to bolo urobené s trénovacími dátami (ďalšie informácie nájdete v bodoch 2.1.2.3 a 2.1.2.4.).

Údaje používame na vyhodnotenie algoritmu.

aq.EvaluateAlgorithm(evaluationData, true);

Prvým parametrom je inštancia údajov. Druhý parameter je true, pretože chcem, aby sa výsledná kofuzna matica a vyhodnocovacie metriky zobrazovali v konzole (bližšie informácie nájdete v bode 2.2.2.8 a 2.2.2.9).

Dostaneme približne nasledujúci výsledok:

```
====================
EVALUATION RESULTS:
 TP: 28  |  FP: 16
 FN: 2  |  TN: 14
--------------------
 Precision: 0,6363636
 Recall: 0,9333333
 F1: 0,7567567
 Accuracy: 0,7
====================
```

2.4.4. Ako používať na jednom zázname.

Pomocou tohto príkazu môžeme získať jeden záznam z dátovej inštancie:

evaluationData.Records[38]

Potom pomocou tejto metódy z inštancie AQ11 môžeme skontrolovať, či je záznam pokrytý výsledným pravidlom:

aq.IsRecordCoveredByPositiveFullStar(evaluationData.Records[38])

Metóda teda vráti hodnotu true, ak pravidlo povie, že záznam je pozitívny (pokrytý). Ak pravidlo hovorí, že záznam je negatívny (nie je pokrytý), dostaneme false.

3. Popis dát

Ako počiatočný súbor údajov som použil súbor údajov o mozgovej príhode, ktorý som našiel na Kaggle. Tento súbor údajov sa používa na predpovedanie toho, či pacient pravdepodobne dostane mozgovú príhodu, na základe rôznych vstupných parametrov. Každý riadok v údajoch je jednym záznamom.

Odkaz na súbor údajov:

https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset?resource=download

Z počiatočného súboru údajov som odstránil stĺpce id, ever_married, work_type, residence_type a bmi.

Tiež som zmenil číselné hodnoty pre age a avg_glucose_level na reťazcové kategórie (napríklad ak je vek nižší ako 20, potom je to kategória 0-20, ak je vek medzi 21 a 40, potom je to kategória 21-40 atď.). Urobil som to, pretože implementácia AQ11 môže pracovať iba s kategorickými údajmi.

Z počiatočného súboru údajov som náhodne vybral 35 záznamov (16 pozitívnych záznamov a 19 negatívnych), ktoré majú byť v súbore údajov na trenovanie. Uložil som ich do súboru stroke_dataset_sample.csv.

Rovnakým spôsobom som vybral 60 záznamov (30 pozitívnych záznamov a 30 negatívnych), ktoré majú byť v hodnotiacom súbore údajov. Uložil som ich do súboru stroke_dataset_sample_evaluation.csv.

Takže dáta majú teraz atribúty:

- „gender“ - Možné hodnoty: (Male, Female)
- „age“ - Možné hodnoty: (0-20, 21-40, 41-60, 61-80, 80+)
- „hypertention“ - Možné hodnoty: (Yes, No)
- „heart_disease“ - Možné hodnoty: (Yes, No)
- „avg_glucose_level“ - Možné hodnoty: (Low, Normal, High, Very High)
- „smoking_status“ - Možné hodnoty: (Unknown, never smoked, formerly smoked, smokes)
- „stroke“ - Možné hodnoty: (Yes, No)

V tomto prípade je cieľovým atribútom "stroke".

Na možných hodnotách atribútov nezáleží, pretože preprocesor číta všetky dáta univerzálne. Tento algoritmus teda môžete použiť takmer na akýkoľvek súbor údajov s akýmikoľvek atribútmi a hodnotami.

Jediný problematický prípad je, keď existujú dva záznamy s rovnakými atribútmi, ale jeden je pozitívny a jeden negatívny. V tomto prípade algoritmus nemôže vytvoriť pravidlo na klasifikáciu záznamov. V tomto prípade sa v konzole zobrazí správa, že pozitívny záznam s id: ... a negatívny záznam s id: ... sú rovnaké.

4. Vyhodnotenie

Na trénovanie sa použili údaje v súbore stroke_dataset_sample.csv. Na vyhodnotenie sa použili údaje v súbore stoke_dataset_sample_evaluation.

4.1. Konfuzna matica

		Actual Values	
		Positive	Negative
Predicted Values	Positive	28	16
	Negative	2	14

Tab. 1 Konfuzna matica

4.2. Výsledné metriky hodnotenia

Precision	0.63636
Recall	0,93333
F1	0.75675
Accuracy	0.7

Tab. 2 Metriky hodnotenia

Záver

Algoritmus nie je na dané dáta veľmi presný. Algoritmus dosahuje úroveň Accuracy 0,7 a úroveň Precision 0,63. Možno, že atribúty nie sú tak relevantné pre cieľový atribút. Pretože v počiatočnom súbore údajov je veľa presne rovnakých záznamov, ktoré sú aj pozitívne aj negatívne. Myslím si, že pre dataset tejto obtiažnosti bol by možno dobrou voľbou pokročilejší algoritmus. Algoritmus však stále počíta užitočné pravidlo pri klasifikácii a poskytuje dostatočne dobré výsledky. Aj v tomto prípade.

Vzhľadom na to, že algoritmus používaný na lekárske predpovede, je lepšie mať viac False Positive ako viac False Negative.

Zoznam použitej literatúry

[1]. Wojtusiak, J. (2012). AQ Learning. In: Seel, N.M. (eds) Encyclopedia of the Sciences of Learning. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_845 ISO 690-2: 1997, Information and documentation – Bibliographic references - Part 2: Electronic documents or parts thereof.