Размита логика с Python

В темата ще бъдат разгледани следните основни въпроси:

- Същност на размитата логика
- Размити множества
- Реализиране на размити множества с Python

1. Размита логика

Терминът размит се отнася до състояния, които не са точно изяснени или са неопределени. В реалния свят често съществуват ситуации, когато не може да се определи дали състоянието е вярно или невярно. В тези случаи размитата логика осигурява много ценна гъвкавост за разсъждение, като позволява по този начин да се опишат неточностите и несигурността на всяка реална ситуация.

В противовес на булевата алгебра, в която стойността на истината е 1, а стойността на лъжата е 0, в размитата система няма логика за такива понятия като абсолютна истина и абсолютна лъжа, но в размитата логика присъства някаква междинна стойност, която е частично вярна или частично невярна.

Размитата логика е в основата на изграждането на системите за размито управление. Това е техника за прилагане на човешкото мислене в системите за управление, която е предназначена да дава приемлива аргументация в случаите, когато не може да се даде точна такава. Характерно за техниката е, че може да имитира човешкото дедуктивно мислене, т.е. процесът, който хората използват, за да правят изводи от това, което знаят.

Размитата логика има множество приложения: в автомобилните системи за управление на скоростта и трафика; в аерокосмическата промишленост за управление на височината на космически кораби и сателити; в съвременните системи за управление като експертни системи, основно за вземане на решения и оценка в големи бизнес предприятия; в химическата промишленост за управление на процеси на сушене, дестилация, поддръжка на киселинност и др.

Размитата логика се използва в множество приложения на изкуствения интелект, в това число и при обработка на естествен език. Основно тя се прилага с невронни мрежи, тъй като имитира начина, по който човек взема решения, което спомага за създаване на много по-бързи реализации. Това се извършва чрез обработка на данните и преобразуването им в по-смислени такива чрез формиране на частични истини като размити множества.

2. Размити множества

Размито множество е множество, в което всеки елемент има дефирана степен на принадлежност между 1 и 0. Размитите множества се представят със знака тилда (~).

Нека е дадено универсално множество X и изображение $m_{\tilde{A}}(x)$, което съпоставя на всеки елемент x от множеството X число в интервала [0, 1]. Тогава размитото множество $\tilde{A}$ може да се дефинира като множество от подредени двойки, което се задава с формулата

$$\tilde{A} = \{(x, m_{\tilde{A}}(x)) \mid x \in X\}$$

Когато универсалното множество X е дискретно и крайно, то размитото множество $\tilde{A}$ може да се представи с формулата:

$$\tilde{A} = \sum_{i=0}^{n} \frac{m_{\tilde{A}}(x_i)}{x_i}$$

Размитите множества имат всички свойства, като класическите множества:

1. **Комутативен закон**: За всеки две размити множества $\tilde{A}$ и $\tilde{B}$ е валидно:

$$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A}$$
$$\tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A}$$

2. **Асоциативен закон**: За всеки три размити множества $\tilde{A}$, $\tilde{B}$ и $\tilde{C}$ е валидно:

$$\tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C}$$
$$\tilde{A} \cap (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cap \tilde{C}$$

3. **Дистрибутивен закон**: За всеки три размити множества $\tilde{A}$, $\tilde{B}$ и $\tilde{C}$ е валидно:

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C})$$
$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})$$

4. **Идентичност:** За всяко размито множество $\tilde{A}$ е валидно:

$$\tilde{A} \cup \tilde{A} = \tilde{A}$$
$$\tilde{A} \cap \tilde{A} = \tilde{A}$$
$$\tilde{A} \cap \tilde{\emptyset} = \tilde{\emptyset}$$
$$\tilde{A} \cup \tilde{\emptyset} = \tilde{A}$$

5. **Транзитивен закон:** За всеки три размити множества $\widetilde{\mathrm{A}}$, $\widetilde{\mathrm{B}}$ и $\widetilde{\mathrm{C}}$ е валидно:

$$\text{Ако } \widetilde{\mathrm{A}} \subseteq \tilde{B} \text{ и } \tilde{B} \subseteq \tilde{C}, \text{то } \tilde{A} \subseteq \tilde{C}$$

Основните операции над размитите множества се дефират по следния начин:

1. **Обединение**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхното обединение $\widetilde{\mathrm{C}}$ е

$$\widetilde{\mathrm{A}} = \widetilde{\mathrm{A}} \cup \tilde{B}$$

$$m_{\tilde{C}}(x) = \max\left(m_{\tilde{A}}(x), m_{\tilde{B}}(x)\right)$$

2. **Сечение**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхното обединение $\widetilde{\mathrm{C}}$ е

$$\widetilde{\mathrm{A}} = \widetilde{\mathrm{A}} \cap \tilde{B}$$

$$m_{\tilde{C}}(x) = \min\left(m_{\tilde{A}}(x), m_{\tilde{B}}(x)\right)$$

3. **Отрицание**: За размитото множество $\widetilde{\mathrm{A}}$, неговото отрицание $\widetilde{N}$ е

$$\widetilde{\mathrm{N}} = C_{\widetilde{\mathrm{A}}}\, X$$

$$m_{\widetilde{N}}(x) = 1 - m_{\tilde{A}}(x)$$

Други често използвани опреции с размитите множества са

1. **Алгебрична сума**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхната алгебрична сума $\widetilde{\mathrm{A}} + \tilde{B}$ е размито множество с изображение $m_{\widetilde{\mathrm{A}}+\tilde{B}}(x)$, което се дефенира с формулата

$$m_{\widetilde{\mathrm{A}}+\tilde{B}}(x) = m_{\tilde{A}}(x) + m_{\tilde{B}}(x) - m_{\tilde{A}}(x).m_{\tilde{B}}(x)$$

2. **Алгебрично произведение**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхното алгебрично произведение $\widetilde{\mathrm{A}}.\tilde{B}$ е размито множество с изображение $m_{\widetilde{\mathrm{A}}.\tilde{B}}(x)$, което се дефенира с формулата

$$m_{\widetilde{\mathrm{A}}.\tilde{B}}(x) = m_{\tilde{A}}(x).m_{\tilde{B}}(x)$$

3. **Алгебрична разлика**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхната алгебрична разлика $\widetilde{\mathrm{C}}$ е

$$\widetilde{\mathrm{C}} = \widetilde{\mathrm{A}} - \tilde{B} = \widetilde{\mathrm{A}} \cap \overline{\widetilde{\mathrm{B}}}$$

$$m_{\tilde{C}}(x) = \min\left(m_{\tilde{A}}(x), 1 - m_{\tilde{B}}(x)\right)$$

4. **Ограничена сума**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхната ограничена сума $\widetilde{\mathrm{A}} \oplus \tilde{B}$ е размито множество с изображение $m_{\widetilde{\mathrm{A}}+\tilde{B}}(x)$, което се дефенира с формулата

$$m_{\widetilde{\mathrm{A}}\oplus\tilde{B}}(x) = min\big(1, m_{\tilde{A}}(x) + m_{\tilde{B}}(x)\big)$$

5. **Ограничена разлика**: За всеки две размити множества $\widetilde{\mathrm{A}}$ и $\widetilde{\mathrm{B}}$, тяхната ограничена разлика $\widetilde{\mathrm{A}}\odot\tilde{B}$ е размито множество с изображение $m_{\widetilde{\mathrm{A}}\odot\tilde{B}}(x)$, което се дефенира с формулата

$$m_{\widetilde{\mathrm{A}}\odot\tilde{B}}(x) = min\big(0, m_{\tilde{A}}(x) - m_{\tilde{B}}(x)\big)$$

3. Реализиране на размити множества с Python

Най-подходящият начин за представяне на размитите пножества в Python е чрез използването на речник, защото по определение размитото множество е множество от подредени двойки. В следващите 3 примера ще бъдат представени последователно операциите обединение (фиг. 4.1), сечение (фиг. 4.2), отрицание (фиг. 4.3) и алгебрична разлика (фиг. 4.4) над размити множества.

В примерите е представен кодът на Python и изхода от него. Входни са размитите множества *A* и *B*, а изходни - обединението *C*, сечението *C*, отрицанието N и алгебричната разлика *C*.

```
A = dict()
B = dict()
C = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is ', A)
print('The Second Fuzzy Set is ', B)


for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

    if A_value > B_value:
        C[A_key] = A_value
    else:
        C[B_key] = B_value

print('Fuzzy Set Union is ', C)
```

а)

```
The First Fuzzy Set is {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union is {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
```

б)

Фиг. 4.1. Пример за пресмятане на обединение на две размити множества
а. Код на Python **б.** Изход от програмата

Примерите използват функцията zip(), която връща обединен обект, който е итератор от n-торки. Всички първият елементи от всеки предаден итератор се комбинират, след това вторите елементи във всеки предаден итератор се съединяват и т.н.

Ако предадените итератори имат различни дължини, итераторът с най-малък брой елементи определя дължината на новия итератор.

Синтаксис

zip(iterator1, iterator2, iterator3 ...)

Реализацията на изображението на обединението на двете размити множества $m_{\tilde{C}}(x) = \max\left(m_{\tilde{A}}(x), m_{\tilde{B}}(x)\right)$ се осъществява с оператора *if else* (фиг. 4.1.a).

```
A = dict()
B = dict()
C = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is', A)
print('The Second Fuzzy Set is', B)

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

    if A_value < B_value:
        C[A_key] = A_value
    else:
        C[B_key] = B_value

print('Fuzzy Set Intersection is', C)
```

а)

```
The First Fuzzy Set is {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Intersection is {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd':
0.5}
```

б)

Фиг. 4.2. Пример за пресмятане на сечение на две размити множества

а. Код на Python **б.** Изход от програмата

Реализацията на изображението на сечението на двете размити множества $m_{\tilde{C}}(x) = \min\left(m_{\tilde{A}}(x), m_{\tilde{B}}(x)\right)$ се осъществява с оператора *if else* (фиг. 4.2.а).

Реализацията на изображението на отрицанието на размитото множество $m_{\tilde{N}}(x) = 1 - m_{\tilde{A}}(x)$ се осъществява в тялото на цикъла *for A_key in A:* (фиг. 4.3.а).

```
A = dict()
N = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

print('The Fuzzy Set is', A)


for A_key in A:
   N[A_key]= 1-A[A_key]

print('Fuzzy Set Complement is', N)
```

а)

```
The Fuzzy Set is {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
```

б)

Фиг. 4.3. Пример за пресмятане на отрицание на размито множество

а. Код на Python **б.** Изход от програмата

Реализацията на изображението на алгебрична разлика на две размити множества на двете размити множества $m_{\tilde{C}}(x) = \min\left(m_{\tilde{A}}(x), 1 - m_{\tilde{B}}(x)\right)$ се осъществява с оператора *if else* (фиг. 4.4.а).

```
A = dict()
B = dict()
C = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is', A)
print('The Second Fuzzy Set is', B)

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]
    B_value = 1 - B_value

    if A_value < B_value:
        C[A_key] = A_value
    else:
        C[B_key] = B_value

print('Fuzzy Set Difference is', C)
```

а)

```
The Fuzzy Set is {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
```

б)

Фиг. 4.4. Пример за пресмятане на алгебрична разлика на две размити множества

а. Код на Python **б.** Изход от програмата

4. Дефиниране на клас, описващ размити множества

Разгледаните до тук примери за пресмятане на основните операции с размити множества могат да бъдат обединени в един клас `FuzzySets`.

В дефиницията на класа `FuzzySets` са дефинирани два метода за инициализация на класа:

1. `__init__(self)`, който инициализира празни речници за размитите множества `A` и `B`, отрицанието на множеството `A` – `complement_A`, отрицанието на множеството `B` – `complement_B`, обединението на множествата `A` и `B` – `union_AB`, сечението на множествата `A` и `B` – `intersection_AB`, разликата на множествата `A - B` `differenceAB`, разликата на множествата `B - A` – `differenceBA`. Установява флаговете за промяна `change_union`, `change_intersection` и `change_complement` в лъжа.

2. `__init__(self,A,nA,B,nB)`, който освен инициализациите в метода `__init__(self)` задава размитите множества `A` и `B` и техните имена `nA` и `nB`.

За реализиране на основните операции са дефинирани методите:

1. `unionOp(self)` – пресмята и извежда обединението на двете размити множества `A` и `B`.

2. `intersectionOp(self)` – пресмята и извежда сеченението на двете размити множества `A` и `B`.

3. `complementOp(self)` – пресмята и извежда отрицанията на двете размити множества `A` и `B`.

4. `__oneMinustwo(self,L,R)` – реализира основните операции за пресмятане на разлика на две две размити множества `L - R`.

5. `AminusB(self)` – пресмята и извежда разликата на двете размити множества `A - B`.

6. `BminusA(self)` – пресмята и извежда разликата на двете размити множества `B - A`.

7. `change_Setz(self,A,B)` – промена и извежда двете размити множества `A` и `B`.

8. `displaySets(self)` – извежда двете размити множества `A` и `B`.

Така дефинираният клас `FuzzySets` с неговите методи е показан на фиг. 4.5.

```
class FuzzySets:

  def __init__(self):
    self.A = dict()
    self.B = dict()

    self.complement_A = dict()
    self.complement_B = dict()
    self.union_AB = dict()
    self.intersection_AB = dict()
    self.differenceAB = dict()
    self.differenceBA = dict()

    self.change_union = False
    self.change_intersection = False
    self.change_complement = False

  def __init__(self,A,nA,B,nB):
    self.A = A
    self.B = B
    self.Aname = nA
    self.Bname = nB

    self.complement_A = dict()
    self.complement_B = dict()
    self.union_AB = dict()
    self.intersection_AB = dict()
    self.differenceAB = dict()
    self.differenceBA = dict()

    self.change_union = False
    self.change_intersection = False
    self.change_complement = False

  def unionOp(self):
    if self.change_union:
      print('Result of UNION operation :',self.union_AB)
    else:

      #unionSet = set(self.A.keys()).union(self.B.keys())
      sa = set(self.A.keys())
      sb = set(self.B.keys())
      intersectionSet = set(self.A.keys()).intersection(self.B.keys())

      for i in intersectionSet:
        self.union_AB[i] = max(self.A[i],self.B[i])
      for i in sa-intersectionSet:
        self.union_AB[i] = self.A[i]
      for i in sb-intersectionSet:
        self.union_AB[i] = self.B[i]

      print('Result of UNION operation :',self.union_AB)
```

Фиг. 4.5. Дефиниране на клас `FuzzySets`

```
  def intersectionOp(self):
    if self.change_intersection:
      print('Result of INTERSECTION operation
:\n\t\t',self.intersection_AB)
    else:
      #unionSet = set(self.A.keys()).union(self.B.keys())
      sa = set(self.A.keys())
      sb = set(self.B.keys())
      intersectionSet = set(self.A.keys()).intersection(self.B.keys())
      for i in intersectionSet:
        self.intersection_AB[i] = min(self.A[i],self.B[i])
      for i in sa-intersectionSet:
        self.intersection_AB[i] = 0.0
      for i in sb-intersectionSet:
        self.intersection_AB[i] = 0.0
      print('Result of INTERSECTION operation
:\n\t\t',self.intersection_AB)
      self.change_intersection = True

  def complementOp(self):
    if self.change_complement:
      print('Result of COMPLEMENT on ',self.Aname,' operation
:',self.complement_A)
      print('Result of COMPLEMENT on ',self.Bname,' operation
:',self.complement_B)
    else:

      for i in self.A:
        self.complement_A[i] = 1 - A[i]
      for i in self.B:
        self.complement_B[i] = 1 - B[i]

      print('Result of COMPLEMENT on ',self.Aname,' operation
:',self.complement_A)
      print('Result of COMPLEMENT on ',self.Aname,' operation
:',self.complement_B)

      self.change_complement = True

  def __oneMinustwo(self,L,R):
    minus_d = dict()
    Rcomp = dict()
    for i in R:
      Rcomp[i] = 1 - R[i]
    sa = set(L.keys())
    sb = set(R.keys())
    intersectionSet = sa.intersection(sb)    # min( A , complement(B) )

    # l - r OR a - b
    for i in intersectionSet:
      minus_d[i] = min(L[i],Rcomp[i])
    for i in sa-intersectionSet:
      minus_d[i] = 0.0
    for i in sb-intersectionSet:
      minus_d[i] = 0.0
    return minus_d
```

Фиг. 4.5. Дефиниране на клас `FuzzySets` – продължение 1

```
  def AminusB(self):
    self.differenceAB = self.__oneMinustwo(self.A,self.B)
    print('Result of DIFFERENCE ',self.Aname,' | ',self.Bname,'
operation :\n\t\t',self.differenceAB)

  def BminusA(self):
    self.differenceBA = self.__oneMinustwo(self.B,self.A)
    print('Result of DIFFERENCE ',self.Bname,' | ',self.Aname,'
operation :\n\t\t',self.differenceBA)

  def change_Setz(self,A,B):
    self.A = A
    self.B = B

    print('\nSet ',self.Aname,' :',self.A)
    print('Set ',self.Bname,' :',self.B,end='')

    self.change_union = True
    self.change_intersection = True
    self.change_complement = True
    print('\t\t\t Cache Reset')

  def displaySets(self):
    print('\nSet ',self.Aname,' :',self.A)
    print('Set ',self.Bname,' :' ,self.B)
```

Фиг. 4.5. Дефиниране на клас `FuzzySets` – продължение 2