406.426B Data Management and Analysis

2017 Fall

Project #3

DB Mining

팀 #10

이명제 (2012 - 13324)

백강희 (2014 – 11564)

권순빈 (2016 – 17330)

오설아 (2016 – 13415)

1. 문제 분석

**1.1 문제 정의**

프로젝트 #2에서 진행 했던 사이트 A의 데이터 및 추가 제공 데이터를 사용하여 프로젝트 #3을 진행한다. 본 프로젝트는 크게 두 부분으로 나뉘게 된다.

Part 1. 연관 분석 - (R1), (R2), (R3)

Part 2. 의사 결정 나무 - (R4), (R5), (R6)

본 프로젝트의 목표이자 방향성은 다음의 3가지로 요약된다.

(1) 주어진 데이터를 이용하여 연관 분석을 진행 한다.

(2) 주어진 데이터를 이용하여 의사결정 나무 모델을 생성한다.

(3) 위 (1), (2)의 결과를 분석한다.

**1.2 요구 조건**

구체적인 요구 사항은 다음과 같으며, 해당 프로젝트는 python, MySQL, Orange3와 scikit-learn을 사용하여 구현하여야 한다.

* + 1. R1

questionPosts.csv와 tagname.csv를 이용하여 questionPost의 Tags field에 태그 언급 횟수가 내림차순으로 정렬된 tagname의 상위 100개의 Tagname이 존재하는지를 boolean 값으로 가지는 100개의 column을 가진 연관 분석을 위한 테이블 TagMatrix view를 만든다.

* + 1. R2

(R1)을 통해 얻은 TagMatrix 데이터로부터 Tag 간의 연관 분석을 실시하고 결과를 출력한다. 연관 분석은 다음 조건을 만족해야 한다.

|  |  |
| --- | --- |
| 사용 알고리즘 | FP-Growth |
| 평가척도 | Confidence |
| 사용 Tag 개수 | 인용 수 상위 100개 |
| Min\_support | 0.01 |
| Min\_confidence | 0.05 |

* + 1. R3

분석 모델의 결과를 정량적, 정성적 관점에서 분석하시오. 이때, 정량적 분석에서는 confidence와 lift를 활용한 분석이 포함되어야 한다.

* + 1. R4

다양한 feature를 바탕으로 사이트 A 사용자의 평판 수치를 추론 할 수 있는 의사결정 나무를 그려야한다. 이를 위해 다음과 같은 column이 포함되는 ReputStatMatrix 뷰를 생성한다.

* UserId(from userInfo.csv / Id)
* Reputation (from user Info.csv / Reputation > 110)
* NumOfPosts (from posts.csv) = 게시물 작성수
* NumOfComments (from comments.csv) = 댓글 작성 수
* NumOfBadges (from badges.csv) = 획득한 뱃지 수
  + 1. R5

(R4)에서 구현한 ReputStatMatrix 뷰 데이터로부터 Reputation이 180을 초과하는지 판단하는 의사결정나무를 생성한다. 본 문제에서 만드는 의사결정나무의 속성들은 다음과 같다.

A. 사용 library : sklearn.tree.DecisionTreeClassifier

B. Binary Classifier 사용

C. Criterion : gini / entropy

D. min\_samples\_split = 10

E. graphviz 출력 format = ‘png’

F. 나머지 속성은 Default

생성된 의사결정 나무 모델을 간단히 분석하고 아래 제시된 3명의 사용자는 위의 각 모델에서 어떤 class로 분류되고, 각 class로 분류될 확률을 얼마인지 구한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UserId | Reputation | NumOfPosts | NumOfComments | NumOfBadges |
| 1000000 | ? | 5 | 5 | 5 |
| 1000001 | ? | 2 | 6 | 18 |
| 1000002 | ? | 6 | 3 | 10 |

* + 1. R6

(R5)에서 만든 의사결정나무를 발전시키기 위해 input feature를 추가하고자 한다. 기존의 view에 최대 2개의 다른 feature를 추가하여 ReputStatMatrix2 view를 생성하고, 이를 활용하여 (R5)와 같은 속성의 의사결정 나무를 만든다. 마찬가지로 생성된 모델을 간략히 분석하고, (R5)의 모델과 간단히 비교한다.

1. 소스 프로그램 설명

구체적인 내용은 코드를 통해 쉽게 알 수 있으므로 보고서에는 핵심적인 프로그램에 대한 설명만을 포함한다.

**2-1. association**

* Pandas의 read\_csv 함수를 활용하여 qeustionPosts.csv와 tagname.csv 파일을 읽어 왔다.
* Pandas의 열을 뽑아오는 함수를 사용해 questionPosts.csv의 Id column만 뽑아 question\_post\_id에 저장하고, 마찬가지로 Tags column만 뽑아 question\_post\_tags에 저장한다.
* 상위 100개의 tag들만을 tagname.csv의 Tag열에서 뽑아 tag\_name1에 저장한다.
* tagname.csv의 상위 100개의 tag\_name1을 dataframe의 column index에 “Tags에 <tag\_name1>이 포함되어 있음 (0 혹은 1)” 꼴로 저장하기 위해 다시 tag\_name2로 저장한다.
* Questionpost.csv의 Id 들을 모두 row index로 가지고, tag\_name2를 각각의 colum index로 갖는 dataframe type의 df를 만든다.
* Df에 for문을 중첩하여 각 Id의 question\_post\_tags에 대해 column의 tag\_name1이 있다면 해당 칸에 있다면 1, 없다면 0 값을 넣는다. 이 과정을 모든 Id에 대해 진행한다.
* 위의 df를 pandas의 dataframe.to\_csv 함수를 이용해 csv 파일로 만든 후 sqlalchemy에서 import해온 create\_engine과 df.to\_sql을 이용해 만들어진 csv 파일을 MySQL Workbench 상에 Table로 넣는다.
* 커서를 이용해 파이썬 상에 VIEW를 생성하는 쿼리를 실행시켜 VIEW를 생성한다.
* Pandas의 기능으로 생성된 TagMarix View의 ‘SELECT \* FROM TagMatix’ SQL문 을 이용해 모든 record들을 갖는 Dataframe을 만들고 이를 연관성 분석하기 위해 numpy의 array에 넣고, 이 array의 1열만을 뽑은 것을 X라 한다.
* 이 X에 대해 연관성 분석을 실시한다. 이를 위해 Orangecontrib.associate.fp growth 전체를 import해온다.
* X 중 min\_support =0.01이상인 itemset들을 Fp growth의 itemsets 함수를 사용하여 뽑아 dict construct하여 itemsets로 저장했다.
* Fp growth의 association\_rules에 앞서 정의된 itemsets와 min\_confidence=0.05를 입력하여 결과를 rules에 저장했다.
* Fp growth의 rules\_stats함수를 사용하여 실행 결과 연관 규칙의 instance 수, confidence, coverage, strength, lift, leverage 등의 통계 결과를 얻었다. 이때 정의된 rules, itemsets 그리고 row의 수를 numpy에 포함된 shape 함수를 이용한 data.shape[0]로 입력했다.

**2-2. decisiontree1**

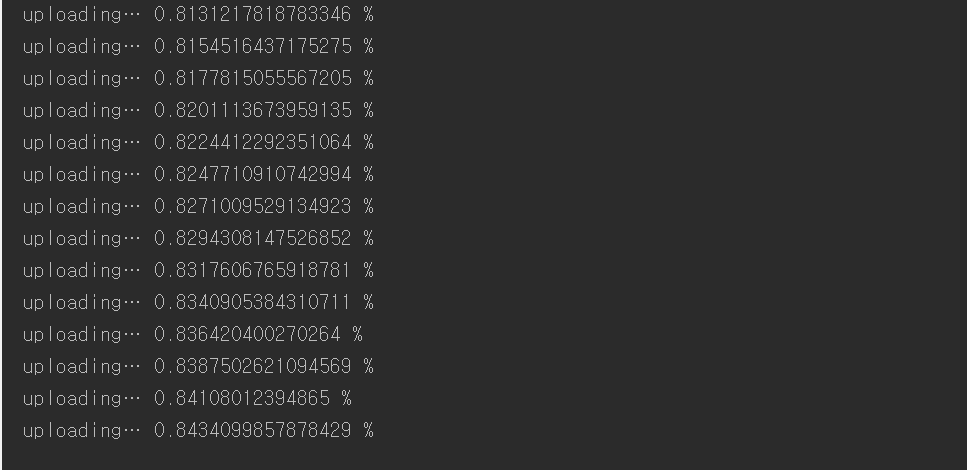
* IF NOT EXISTS 를 사용하여 이미 db2017\_10 database가 존재하는 경우 다시 생성하지 않도록 했다.
* IF NOT EXISTS 를 사용하여 이미 table이 존재하는 경우 다시 생성하지 않도록 했다.
* INSERT IGNORE 를 사용하여 데이터를 insert할 때 중복되는 데이터를 넣지 않도록 했다. 이를 위해서 테이블 생성 시 각 테이블에 PRIMARY KEY를 설정한다. INSERT IGNORE은 PRIMARY KEY가 중복된 데이터는 중복 저장하지 않는 명령이다.
* Userinfo, posts, comments, badges table에서 UserId가 같은 것끼리 개수를 COUNT하여 JOIN하도록 했다. 이 때, NumOfPosts, NumOfComments, NumOfBadges가 NULL일 경우 0으로 저장하도록 IFNULL을 사용했다. 이렇게 생성된 결과를 VIEW를 생성하여 나타내도록 했다.
* Pandas의 read\_sql 함수를 이용하여 ReputStatMatrix view로부터 모든 row를 읽어와 dataframe의 형태로 저장했다.
* Reputation이 180을 초과하는지에 대해 decisiontree를 생성하기 위해 주어진 dataset에 대해 (df[‘Reputation’]>180).astype(int) 로 Reputation이 180을 초과하는 경우 1 class로, 이하인 경우 0 class으로 dataframe의 Reputation 값을 바꾸었다.
* 이후 decisiontree를 fitting하고 visualize하여 gini criterion에 대해 decisiontree1\_gini의 이름으로 png 파일을 저장하고, entropy criterion에 대해 decisiontree1\_entropy의 이름으로 png 파일을 저장하였다. 이 때, 가시성을 높이기 위해 tree.export\_graphviz 함수에 filled 값을 True로 설정하여 decisiontree가 색을 띄도록 했다.
* 주어진 feature를 가지는 사용자에 대해 class를 추론하고 그 확률을 계산하였다.

**2-3. decisiontree2**

* 2-2의 decisiontree1 함수에 이어 추가로 UserInfoId를 column으로 갖는 postHistory, votes table을 생성하고 데이터를 넣었다.
* postHistory와 votes에서 각 UserInfoId 에 대해 개수를 COUNT하여 decisiontree1의 view에 2개의 column NumOfPostHistorys, NumOfVotes를 추가하였다.
* 마찬가지로 Reputation에 대해 decisiontree를 fitting하고 visualize하였다.

1. 프로그램 실행 결과

**3-1. association**



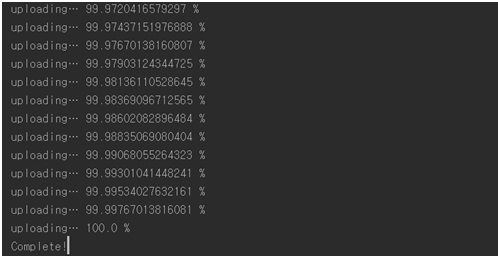


Figure association의 실행 결과1

Mysql 상에 TagMatrix VIEW가 생성된 모습

Figure 2 association의 실행 결과2

연관성 분석 결과

Figure 3 association의 실행 결과3 - 연관성 분석 결과

**3-2. decisiontree1**

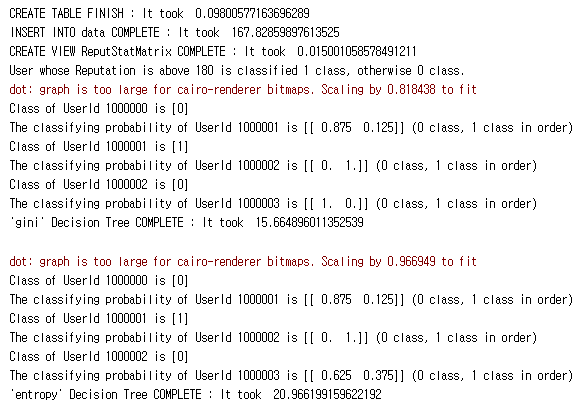


Figure decisiontree1의 실행 결과

**3-3. decisiontree2**

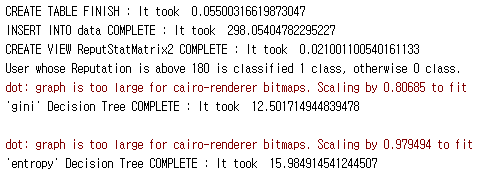


Figure decisiontree2의 실행 결과

1. 모델의 결과 및 분석

**4-1. association**

Rules, itemsets과 itemsets 총 record 의 수에 대해 Fpgrowth.rule\_stats를 적용하고 이를 list에 넣어 반환한 결과 다음과 같다.

먼저 각 리스트 원소의 구성은 순서대로 다음과 같다.

* **atecedent** (*frozenset*) – The LHS of the association rule.
* **consequent** (*frozenset*) – The RHS of the association rule.
* **support** (*int*) – Support as an absolute number of instances.
* **confidence** (*float*) – The confidence percent, calculated as: total\_support / lhs\_support.
* **coverage** (*float*) – Calculated as: lhs\_support / n\_examples
* **strength** (*float*) – Calculated as: rhs\_support / lhs\_examples
* **lift** (*float*) – Calculated as: n\_examples \* total\_support / lhs\_support / rhs\_support
* **leverage** (*float*) – Calculated as: (total\_support \* n\_examples - lhs\_support \* rhs\_support) / n\_examples\*\*2

[(frozenset({1}), frozenset({0}), 1059, 0.19564012562349897, 0.12611542135551362, 1.3382597450581932, 1.1591758464779403, 0.003388082470134884),

(frozenset({0}), frozenset({1}), 1059, 0.1461899503036996, 0.16877519163113627, 0.7472390944229708, 1.1591758464779403, 0.003388082470134884),

(frozenset({2}), frozenset({0}), 607, 0.2217756667884545, 0.06376831853871065, 2.646693459992693, 1.3140300102467224, 0.0033797511825348903),

(frozenset({0}), frozenset({2}), 607, 0.08379348426283821, 0.16877519163113627, 0.377829928216455, 1.3140300102467224, 0.0033797511825348903),

(frozenset({1}), frozenset({8}), 576, 0.10641049325697395, 0.12611542135551362, 0.3000184740439682, 2.8123428454941988, 0.008648180511140139),

(frozenset({8}), frozenset({1}), 576, 0.35467980295566504, 0.03783695626849328, 3.333128078817734, 2.812342845494199, 0.008648180511140139),

(frozenset({12}), frozenset({3}), 431, 0.3221225710014948, 0.031173551408401484, 1.9162929745889388, 5.3922866107469405, 0.008179469588053833),

(frozenset({3}), frozenset({12}), 431, 0.16809672386895474, 0.059737657556906874, 0.5218408736349454, 5.3922866107469405, 0.008179469588053833)]

* 코드 실행 결과 0 : r , 1 : regression, 2 : time-series 3 : machine=learning, 8 : distributions, 12 : statistical-significance 이 6개의 tag에 대한 8개의 association rules를 얻었다.
* 이때 0 : r과 1 : regression은 각 두개의 association rules에 antecedent로 포함되었다.
* tagname은 많이 언급된 tagname을 내림차순으로 정렬해 놓은 것이므로, 역시 연관성 분석 결과 상위 순서에 있는 tagname들 끼리 높은 연관성을 가졌다. 12를 제외하고는 모두 상위 10개 이내의 tagname 들이다.
* 자동으로 support 기준으로 내림차순 정렬된 결과가 나온 것인지 모르겠으나, support의 순서도 tagname의 내림차순 정렬과 거의 일치한다.
* 모든 association rule들은 antecedent와 consequent 들이 대칭을 이루었다. 즉 일방향 적인 관계가 아니라 하나가 있으면 다른 하나가 있을 확률이 높은 양방향적인 관계인 것이다.
* 엑셀의 함수를 이용한 결과 Confidence의 평균은 0.19983… 과 같았고, Confidence의 표준편차는 0.09683… 과 같았다.
* 마찬가지로 엑셀 함수를 이용한 결과 Lift의 평균은 2.66945… 과 같았고, 표준편차는 1.81677… 과 같았다.
* Lift는 대칭인 두 association rules에 대해 같은 값을 갖는데 lift의 정의상 이는 당연하다. Lift는 consequent의 support를 고려한 confidence 인데 이 값들이 8개의 모든 association rules에서 1보다 크므로 모두 positively corelated되어 있다고 할 수 있다.
* Confidence를 기준으로 association rule들을 내림차순 정렬하면 6-> 7-> 3-> 1 -> 8 -> 2 -> 5 -> 4 순이다.
* Lift를 기준으로 association rule들을 내림차순 정렬하면 8=7 -> 6=5 -> 4=3 -> 2=1 인데, 이는 confidence 값보다는 consequent의 support 값의 영향이 크기 때문으로 보인다.
* 대칭인 두 association rule들은 Lift 값은 같으나 confidence 값은 다르다. Confidence만으로는 그 association rule의 실질적 의미를 파악하는데에 한계가 있기 때문에 lift를 고려해야 한다. 즉 confidence는 6번 rule이 가장 높으나 실제로는 7,8 번 rule이 가장 큰 양의 상관관계가 있다고 할 수 있다.

**4-2. decisiontree1**

Decisiontree의 경우 크기가 너무 크기 때문에 처음 분류한 7개 node만을 뽑아보았다.

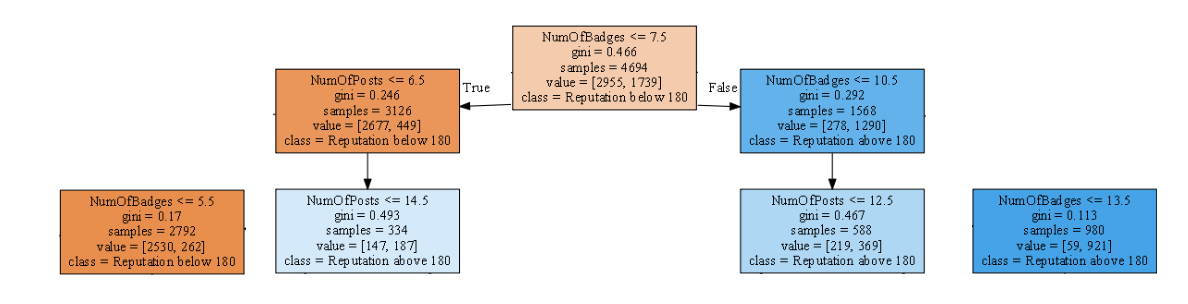


Figure gini 기준으로 분류한 결과

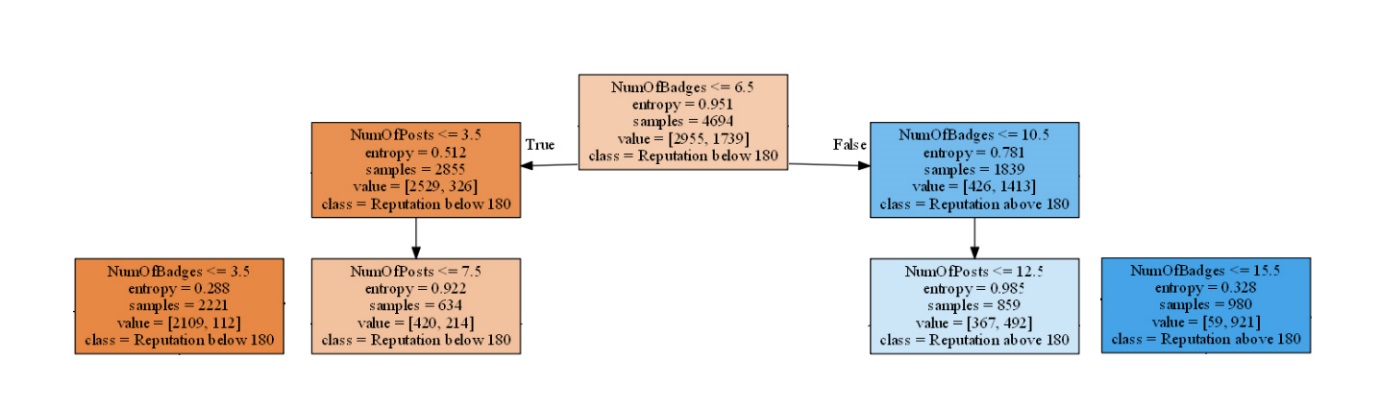


Figure entropy 기준으로 분류한 결과

class가 0과 1 두 개이기 때문에 0<=gini<=0.5 이고 0<=entropy<=1 이고, gini와 entropy 둘 다 작을수록 좋은 분류 기준이라 할 수 있다. Gini와 entropy 둘 다 첫번째 분류 기준은 NumOfBadges로, gini의 경우 <=7.5, entropy는 <=6.5 이다. 이 때 Gini=0.466 이고 entropy=0.951 이다. 최댓값을 고려했을 때 gini는 0.932, entropy는 0.951의 비율을 가지므로 비슷한 수준이라 할 수 있다. 다음 분류를 보면 첫번째 분류가 True일 때 gini는 NumOfPosts<=6.5, gini=0.246 이고, entropy는 NumofPosts<=3.5, entropy=0.512 이다. 최댓값과의 비율을 계산하면 gini는 0.492, entropy는 0.512이다. False일 때 gini는 NumOfBadges<=10.5, gini=0.292, entropy는 NumOfBadges<=10.5, entropy=0.781이다. 최댓값과의 비율을 고려했을 때 gini는 0.584, entropy는 0.781이다. 여기까지 보았을 때 gini의 비율이 대부분 다 작은 수준으로 나타나 entropy에 비해 더 좋은 분류 기준이라 생각할 수 있다. 하지만 전체 tree의 크기를 보았을 때 gini가 entropy에 비해 더 많은 node를 가지고 큰 tree이다. Tree가 작을 경우 해석이 용이하기 때문에 이런 점을 고려했을 때는 entropy가 더 좋은 분류 기준이라 판단할 수 있다.

**4-3. decisiontree2**

Decisiontree1을 발전시키기 위해 input feature에 NumOfPostHistorys와 NumOfVotes를 추가했다. 이를 바탕으로 decisiontree를 생성해본 결과를 분석해 보겠다. 위 4-2의 decisiontree1에서 분석한 것과 같이 상위 7개 node에 대해서는 똑 같은 결과가 나타났다.

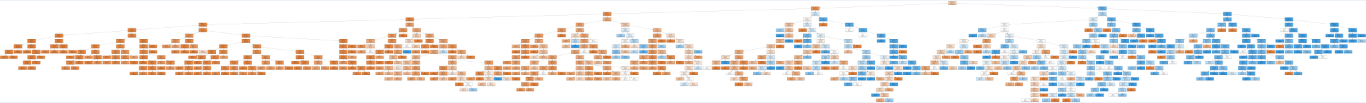


Figure decisiontree1\_gini



Figure decisiontree2\_gini

위 두 그림은 1과 2 decisiontree에서 gini 기준으로 분류한 전체 decisiontree인데, 전체적으로 보았을 때 decisiontree1의 결과에 비해 2의 결과가 node가 좀 더 혼합된 결과가 나타난다고 할 수 있다. 따라서 NumOfPostHistorys와 NumOfVotes를 추가한 결과 기존의 NumOfPosts, NumOfComments, NumOfBadges만 있었던 것에 비해 feature 간의 관계가 흐려졌다고 볼 수 있다.

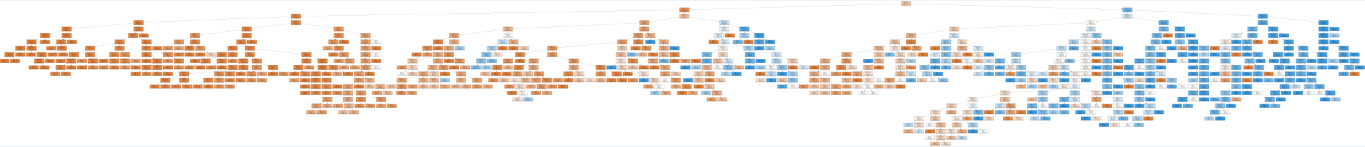


Figure decisiontree1\_entropy

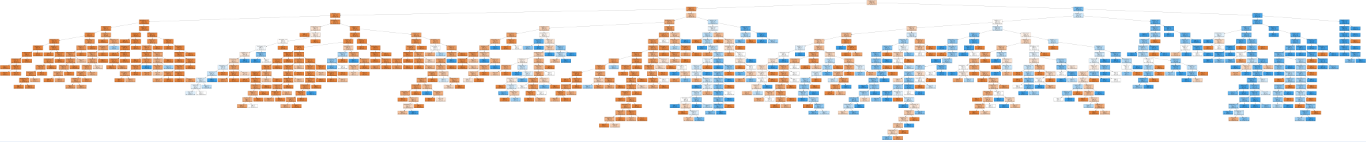


Figure decisiontree2\_entropy

마찬가지로 entropy 기준에서도 decisiontree2의 결과가 더 혼합된 node의 모습이 나타난다고 볼 수 있다.

결과적으로, Posts, Comments, Badges 사이에 비해 PostHistory와 Votes를 추가하면 좀 더 무의미한 관계가 존재한다고 분석할 수 있다. 즉, Posts, Comments, Badges가 많다고 해서 PostHistory와 Votes가 꼭 많은 것은 아니라고 할 수 있다.