q1

November 8, 2019

```
In [29]: import numpy as np
         import pandas as pd
         import math
         from statistics import mean
         from copy import deepcopy
         from itertools import chain
```

```
In [53]: df = pd.read_csv("./train.csv")
         df.head()
```

```
Out[53]:    PassengerId  Survived  Pclass  \
         0            1         0       3
         1            2         1       1
         2            3         1       3
         3            4         1       1
         4            5         0       3

                                                         Name     Sex   Age  SibSp  \
         0                            Braund, Mr. Owen Harris    male  22.0      1
         1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
         2                             Heikkinen, Miss. Laina  female  26.0      0
         3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
         4                           Allen, Mr. William Henry    male  35.0      0

            Parch            Ticket     Fare Cabin Embarked
         0      0         A/5 21171   7.2500   NaN        S
         1      0          PC 17599  71.2833   C85        C
         2      0  STON/O2. 3101282   7.9250   NaN        S
         3      0            113803  53.1000  C123        S
         4      0            373450   8.0500   NaN        S
```

```
In [31]: # drop waste columns
         df = df.drop(columns = ['PassengerId', 'Name', 'Ticket'])
```

```
In [32]: def convert_to_one(str1):
             if str1 != 0:
                 return 1
             return 0
```

```
In [33]: df['Cabin'] = df['Cabin'].replace(np.nan, 0)
         df['Cabin'] = df['Cabin'].apply(convert_to_one)
```

```
In [34]: df.head()
```

```
Out[34]:    Survived  Pclass     Sex   Age  SibSp  Parch     Fare  Cabin Embarked
         0         0       3    male  22.0      1      0   7.2500      0        S
         1         1       1  female  38.0      1      0  71.2833      1        C
         2         1       3  female  26.0      0      0   7.9250      0        S
         3         1       1  female  35.0      1      0  53.1000      1        S
         4         0       3    male  35.0      0      0   8.0500      0        S
```

```
In [35]: dataset = []

         for i, row in df.iterrows():
             vals = row.values
             r = []
             for v in vals:
                 r.append(v)
             dataset.append(r)

         dataset[0]
```

```
Out[35]: [0, 3, 'male', 22.0, 1, 0, 7.25, 0, 'S']
```

```
In [36]: lookup_variable_type = [False, False, False, True, False, False, True, False, False]
```

```
In [37]: header = ['', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Cabin', 'Embarked']
         class Question:
             def __init__(self, column, value):
                 self.column = column
                 self.value = value

             def match(self, example):
                 val = example[self.column]

                 if self.column == 7:
                     return False

                 if lookup_variable_type[self.column]:
                     return val <= self.value

                 return self.value == val

             def __repr__(self):
                 condition = "contains"
                 return "Does %s %s %s?" % (
                     header[self.column], condition, str(self.value))
```

```
In [38]: def getSubset(rules, training_data):
             ans = []

             for d in training_data:
                 flag = True
                 for r in rules:
                     if not r.match(d):
                         flag = False
                         break

                 if flag:
                     ans.append(d)
             return ans
```

```
In [39]: def getMostCommonClass(training_data):
             class_freq_dict = {}

             for d in training_data:
                 c = str(d[0])

                 if c not in class_freq_dict:
                     class_freq_dict[c] = 1
                 else:
                     class_freq_dict[c] += 1

             if class_freq_dict['0'] > class_freq_dict['1'] :
                 return 0
             else:
                 return 1
```

```
In [40]: def getStatSignificance(expression, training_data):
             score = 0
             for d in training_data:
                 score += 1

                 for q in expression:
                     if not q.match(d):
                         score -= 1
                         break

             return score
```

```
In [41]: def getWorstExp(expressionSet, training_data):
             worstExp = []
             worstScore = math.inf

             for e in expressionSet:
                 x = getStatSignificance(e, training_data)
                 if x < worstScore:
                     worstExp = e
                     worstScore = x

             return worstExp
```

```
In [42]: def find_best_condition_exp(training_data, simpleCS, threshExp, maxExpressions):
             conditional_exp_set = [[]]
             best_condition_expression = []
             best_condition_expression_score = 0

             while True:
                 trialCES = []
                 for x in conditional_exp_set:
                     x = set(x)
                     for y in simpleCS:
                         x.add(y)
                         trialCES.append(frozenset(deepcopy(x)))
                         x.remove(y)

         #         print(trialCES)
                 trialCES = set(trialCES)


                 for t in trialCES:
                     if t in conditional_exp_set:
                         trialCES.remove(t)

                 for expression in trialCES:
                     score = getStatSignificance(expression, training_data)
                     if score > best_condition_expression_score:
                         best_condition_expression_score = score
                         best_condition_expression = expression

                 while len(trialCES) > maxExpressions:
                     worst = getWorstExp(trialCES, training_data)
                     trialCES.remove(worst)

                 conditional_exp_set = set(trialCES)

                 if len(conditional_exp_set) > 0:
                     break

             return best_condition_expression
```

```
In [43]: def getRuleList(training_data, simpleCS, threshExp, maxExpressions):
             rule = []
             answers = []
             while len(training_data) > 0:
                 best_condition_exp = find_best_condition_exp(training_data, simpleCS, threshE

                 if len(best_condition_exp) == 0:
                     break
         #         print("x")
                 subset_tr = getSubset(best_condition_exp, training_data)
                 training_data = [x for x in training_data if x not in subset_tr]

                 most_common_class = getMostCommonClass(subset_tr)

                 rule.append(best_condition_exp)
                 answers.append(most_common_class)

             return rule, answers

In [44]: def accuracy(rule, answer, dataset):
             n = 0
             c = 0

             for d in dataset:
                 n += 1
                 flag = True
                 for q in rule:
                     if not q.match(d):
                         n -= 1
                         flag = False
                         break
                 if flag:
                     if answer == d[0]:
                         c += 1
             if n != 0:
                 return c / n
             else:
                 return "NaN"

In [45]: def laplace(rule, answer, dataset):
             n = len(dataset)
             c = 0
             n = 0
             for d in dataset:
                 n += 1
                 flag = True
                 for q in rule:
                     if not q.match(d):
                         n -= 1
                         flag = False
```

```
                break

        if flag:
            if answer == d[0]:
                c += 1

    return (c + 1) / (n + 2)
```

In [46]:

```
def coverage(rule, dataset):
    n = len(dataset)
    c = 0

    for d in dataset:
        c += 1
        for q in rule:
            if not q.match(d):
                c -= 1
                break

    return c / n
```

In [47]:

```
simpleCS = []

for i in range(1, len(dataset[0])):
    if lookup_variable_type[i]:
        #get average
        m = mean([row[i] for row in dataset])
        simpleCS.append(Question(i, m))
    else:
        uniqueVals = set([row[i] for row in dataset])

        for u in uniqueVals:
            simpleCS.append(Question(i, u))

rules, answers = getRuleList(dataset, simpleCS, 0, 400)
rulesX = [list(r) for r in rules]
rulesX
```

Out[47]:

```
[[Does Fare contains 32.204207968574636?],
 [Does Pclass contains 1?],
 [Does Embarked contains S?],
 [Does Pclass contains 2?]]
```

In [48]:

```
answerTable = []
for i in range(len(rules)):
    row = []
    row.append(rulesX[i])
    row.append(answers[i])
    row.append(accuracy(rulesX[i], answers[i], dataset))
```

```
        row.append(laplace(rulesX[i], answers[i], dataset))
        row.append(coverage(rulesX[i], dataset))

        answerTable.append(row)
```

```
In [49]: answerTable
```

```
Out[49]: [[[Does Fare contains 32.204207968574636?],
           0,
           0.6823529411764706,
           0.6818181818181818,
           0.7631874298540965],
          [[Does Pclass contains 1?],
           1,
           0.6296296296296297,
           0.6284403669724771,
           0.24242424242424243],
          [[Does Embarked contains S?],
           0,
           0.6630434782608695,
           0.6625386996904025,
           0.7227833894500562],
          [[Does Pclass contains 2?],
           1,
           0.47282608695652173,
           0.4731182795698925,
           0.20650953984287318]]
```

```
In [50]: answerSortedAccuracy = deepcopy(answerTable)
         answerSortedAccuracy.sort(key = lambda x : x[2], reverse = True)
         answerSortedAccuracy
```

```
Out[50]: [[[Does Fare contains 32.204207968574636?],
           0,
           0.6823529411764706,
           0.6818181818181818,
           0.7631874298540965],
          [[Does Embarked contains S?],
           0,
           0.6630434782608695,
           0.6625386996904025,
           0.7227833894500562],
          [[Does Pclass contains 1?],
           1,
           0.6296296296296297,
           0.6284403669724771,
           0.24242424242424243],
          [[Does Pclass contains 2?],
           1,
           0.47282608695652173,
           0.4731182795698925,
           0.20650953984287318]]
```

```
In [51]: answerSortedLaplace = deepcopy(answerTable)
         answerSortedLaplace.sort(key = lambda x : x[3], reverse = True)
         answerSortedLaplace
```

```
Out[51]: [[[Does Fare contains 32.204207968574636?],
           0,
           0.6823529411764706,
           0.6818181818181818,
           0.7631874298540965],
          [[Does Embarked contains S?],
           0,
           0.6630434782608695,
           0.6625386996904025,
           0.7227833894500562],
          [[Does Pclass contains 1?],
           1,
           0.6296296296296297,
           0.6284403669724771,
           0.24242424242424243],
          [[Does Pclass contains 2?],
           1,
           0.47282608695652173,
           0.4731182795698925,
           0.20650953984287318]]
```

```
In [52]: answerSortedCoverage = deepcopy(answerTable)
         answerSortedCoverage.sort(key = lambda x : x[3], reverse = True)
         answerSortedCoverage
```

```
Out[52]: [[[Does Fare contains 32.204207968574636?],
           0,
           0.6823529411764706,
           0.6818181818181818,
           0.7631874298540965],
          [[Does Embarked contains S?],
           0,
           0.6630434782608695,
           0.6625386996904025,
           0.7227833894500562],
          [[Does Pclass contains 1?],
           1,
           0.6296296296296297,
           0.6284403669724771,
           0.24242424242424243],
          [[Does Pclass contains 2?],
           1,
           0.47282608695652173,
           0.4731182795698925,
           0.20650953984287318]]
```