q2

November 8, 2019

In [79]:
```
import numpy as np
import pandas as pd
import math
from statistics import mean
from copy import deepcopy
from itertools import chain
from collections import Counter
from sklearn.model_selection import train_test_split
```

In [80]:
```
df = pd.read_csv("./train.csv")
```

In [81]:
```
# drop waste columns
df = df.drop(columns = ['PassengerId', 'Name', 'Ticket'])
```

In [82]:
```
def convert_to_one(str1):
    if str1 != 0:
        return 1
    return 0
```

In [83]:
```
df['Cabin'] = df['Cabin'].replace(np.nan, 0)
df['Cabin'] = df['Cabin'].apply(convert_to_one)
```

In [84]:
```
df.head()
```

Out[84]:

	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 [85]:
```
dataset = []

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

```
            dataset.append(r)

        dataset[0]
```

Out[85]: [0, 3, 'male', 22.0, 1, 0, 7.25, 0, 'S']

```
In [86]: lookup_variable_type = [False, False, False, True, False, False, True, False, False]

In [87]: 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"
                 if lookup_variable_type[self.column]:
                     condition = "less than equal to"

                 return "Does %s %s %s?" % (
                     header[self.column], condition, str(self.value))

In [88]: def get_min_class(df):
             col_survived = df['Survived']
             survived_count = Counter(col_survived)
             print(survived_count)
             if survived_count[0] < survived_count[1]:
                 return 0
             else:
                 return 1

In [89]: def split_pos_neg(dataset, pos_class):
             pos = []
             neg = []

             for d in dataset:
                 if d[0] == pos_class:
```

```
                pos.append(d)
            else:
                neg.append(d)

        return pos, neg
```

In [90]:
```
def split_grow_prune(pos, neg, ratio=0.7):
    indices_pos = list(range(len(pos)))
    np.random.shuffle(indices_pos)
    split_pos = int(np.floor(ratio * len(pos)))
    grow_pos_idx, prune_pos_idx = indices_pos[:split_pos], indices_pos[split_pos:]

    indices_neg = list(range(len(neg)))
    np.random.shuffle(indices_neg)
    split_neg = int(np.floor(ratio * len(neg)))
    grow_neg_idx, prune_neg_idx = indices_neg[:split_neg], indices_neg[split_neg:]

    grow_pos = []
    grow_neg = []
    prune_pos = []
    prune_neg = []

    for i in grow_pos_idx:
        grow_pos.append(pos[i])

    for i in grow_neg_idx:
        grow_neg.append(neg[i])

    for i in prune_pos_idx:
        prune_pos.append(pos[i])

    for i in prune_neg_idx:
        prune_neg.append(neg[i])

    return grow_pos, grow_neg, prune_pos, prune_neg
```

In [91]:
```
min_class = get_min_class(df)
pos, neg = split_pos_neg(dataset, min_class)
```

```
Counter({0: 549, 1: 342})
```

In [92]:
```
def get_info_gain(exp, grow_pos, grow_neg):
    count_pos = 0
    count_neg = 0

    for d in grow_pos:
        count_pos += 1
```

```
            for c in exp:
                if not c.match(d):
                    count_pos -= 1
                    break

        for d in grow_neg:
            count_neg += 1

            for c in exp:
                if not c.match(d):
                    count_neg -= 1
                    break

        if count_pos == 0 and count_neg == 0 or count_pos == 0:
#           print("Zero encountered")
            return -1
        return count_pos * (math.log(count_pos/(count_pos + count_neg)) - math.log(len(gr
```

In [93]:
```
def grow_rule(grow_pos, grow_neg, basic_conditions):
    exp = []

    while True:
        max_exp_gain = get_info_gain(exp, grow_pos, grow_neg)
        cond_to_add = None

        for cond in basic_conditions:
            exp.append(cond)
            new_exp_info_gain = get_info_gain(exp,grow_pos, grow_neg)

            if new_exp_info_gain > 0 and new_exp_info_gain > max_exp_gain:
                max_exp_gain = new_exp_info_gain
                cond_to_add = cond
            exp.pop()

        if cond_to_add is not None:
            exp.append(cond_to_add)
            basic_conditions.remove(cond_to_add)
        else:
            break

    return exp
```

In [94]:
```
def getScorePrune(rule, prune_pos, prune_neg):
    p = 0
    n = 0
    P = len(prune_pos)
    N = len(prune_neg)
```

```
            for d in prune_pos:
                p += 1

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

            for d in prune_neg:
                n += 1

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

            return (p + N - n) / (P + N)
```

In [95]:
```
def getAccuracy(rule, prune_pos, prune_neg):
    count_wrong_pos = 0
    count_wrong_neg = 0

    for d in prune_pos:
        for c in rule:
            if not c.match(d):
                count_wrong_pos += 1
                break

    for d in prune_neg:
        for c in rule:
            if not c.match(d):
                count_wrong_neg += 1
                break

    return 1 - (count_wrong_neg + count_wrong_pos) / (len(prune_pos) + len(prune_neg))
```

In [96]:
```
def getErrorRate(rule, prune_pos, prune_neg):
    return 1 - getAccuracy(rule, prune_pos, prune_neg)
```

In [97]:
```
def custom_deepcopy(rule):
    rule_temp = []

    for d in rule:
        rule_temp.append(d)

    return rule_temp
```

In [98]:
```
def prune_rule(rule, prune_pos, prune_neg):
```

```
            while True:
                if getErrorRate(rule, prune_pos, prune_neg) > 0.5:
                    break

                ruleCopy = custom_deepcopy(rule)
                cur_score = getScorePrune(rule, prune_pos, prune_neg)
                cond_to_remove = None
                for cond in ruleCopy:
                    rule.remove(cond)
                    score = getScorePrune(rule, prune_pos, prune_neg)

                    if score > cur_score:
                        cond_to_remove = cond
                        cur_score = score

                    rule.append(cond)

                if cond_to_remove is not None:
                    rule.remove(cond_to_remove)
                else:
                    break

            return rule
```

```
In [99]: def get_subset(data, rules):
            ans = []

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

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

```
In [100]: def IREP(pos, neg, simple_condtions):
             ruleset = []

             while(len(pos)>0):
                 grow_pos, grow_neg, prune_pos, prune_neg = split_grow_prune(pos,neg)
                 rule = grow_rule(grow_pos, grow_neg, deepcopy(simple_condtions))
                 rule = prune_rule(rule, prune_pos, prune_neg)

                 if len(rule) == 0:
                     return ruleset
```

```
            else:
                ruleset.append(rule)
                pos_subset = get_subset(pos,rule)
                neg_subset = get_subset(neg,rule)

                pos = [x for x in pos if x not in pos_subset]
                neg = [x for x in neg if x not in neg_subset]

        return ruleset
```

```
In [101]: 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 [102]: 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 [103]: 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 [104]: 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))
```

```
In [105]: rules = IREP(pos,neg,simpleCS)
          rules
```

```
Out[105]: [[Does Sex contains female?],
           [Does Pclass contains 1?],
           [Does Parch contains 1?, Does SibSp contains 1?],
           [Does Embarked contains C?, Does Pclass contains 3?],
           [Does SibSp contains 0?, Does Parch contains 2?],
           [Does SibSp contains 0?, Does Parch contains 0?],
           [Does SibSp contains 2?, Does Embarked contains Q?],
           [Does Parch contains 2?, Does Fare less than equal to 32.2042079685746367],
           [Does SibSp contains 0?, Does Pclass contains 3?],
           [Does Pclass contains 2?, Does Embarked contains S?],
           [Does SibSp contains 1?, Does Embarked contains S?, Does Parch contains 0?]]
```

```
In [106]: answerTable = []
          for i in range(len(rules)):
              row = []
              row.append(rules[i])
              row.append(min_class)
              row.append(accuracy(rules[i], min_class, dataset))
```

```
            row.append(laplace(rules[i], min_class, dataset))
            row.append(coverage(rules[i], dataset))

            answerTable.append(row)

        answerTable
```

Out[106]:
```
[[[Does Sex contains female?],
  1,
  0.7420382165605095,
  0.740506329113924,
  0.35241301907968575],
 [[Does Pclass contains 1?],
  1,
  0.6296296296296297,
  0.6284403669724771,
  0.24242424242424243],
 [[Does Parch contains 1?, Does SibSp contains 1?],
  1,
  0.5964912280701754,
  0.5932203389830508,
  0.06397306397306397],
 [[Does Embarked contains C?, Does Pclass contains 3?],
  1,
  0.3787878787878788,
  0.38235294117647056,
  0.07407407407407407],
 [[Does SibSp contains 0?, Does Parch contains 2?],
  1,
  0.7241379310344828,
  0.7096774193548387,
  0.03254769921436588],
 [[Does SibSp contains 0?, Does Parch contains 0?],
  1,
  0.30353817504655495,
  0.3042671614100185,
  0.6026936026936027],
 [[Does SibSp contains 2?, Does Embarked contains Q?],
  1,
  0.6666666666666666,
  0.6,
  0.003367003367003367],
 [[Does Parch contains 2?, Does Fare less than equal to 32.2042079685746367],
  1,
  0.5,
  0.5,
  0.04489337822671156],
 [[Does SibSp contains 0?, Does Pclass contains 3?],
```

```
 1,
 0.23646723646723647,
 0.23796033994334279,
 0.3939393939393939],
[[Does Pclass contains 2?, Does Embarked contains S?],
 1,
 0.4634146341463415,
 0.463855421686747,
 0.1840628507295174],
[[Does SibSp contains 1?, Does Embarked contains S?, Does Parch contains 0?],
 1,
 0.42857142857142855,
 0.43037974683544306,
 0.08641975308641975]]
```

In [107]:
```
answerSortedAccuracy = deepcopy(answerTable)
answerSortedAccuracy.sort(key = lambda x : x[2], reverse = True)
answerSortedAccuracy
```

Out[107]:
```
[[[Does Sex contains female?],
 1,
 0.7420382165605095,
 0.740506329113924,
 0.35241301907968575],
[[Does SibSp contains 0?, Does Parch contains 2?],
 1,
 0.7241379310344828,
 0.7096774193548387,
 0.03254769921436588],
[[Does SibSp contains 2?, Does Embarked contains Q?],
 1,
 0.6666666666666666,
 0.6,
 0.003367003367003367],
[[Does Pclass contains 1?],
 1,
 0.6296296296296297,
 0.6284403669724771,
 0.24242424242424243],
[[Does Parch contains 1?, Does SibSp contains 1?],
 1,
 0.5964912280701754,
 0.5932203389830508,
 0.06397306397306397],
[[Does Parch contains 2?, Does Fare less than equal to 32.2042079685746 36?],
 1,
 0.5,
 0.5,
```

```
 0.04489337822671156],
[[Does Pclass contains 2?, Does Embarked contains S?],
 1,
 0.4634146341463415,
 0.463855421686747,
 0.1840628507295174],
[[Does SibSp contains 1?, Does Embarked contains S?, Does Parch contains 0?],
 1,
 0.42857142857142855,
 0.43037974683544306,
 0.08641975308641975],
[[Does Embarked contains C?, Does Pclass contains 3?],
 1,
 0.3787878787878788,
 0.38235294117647056,
 0.07407407407407407],
[[Does SibSp contains 0?, Does Parch contains 0?],
 1,
 0.30353817504655495,
 0.3042671614100185,
 0.6026936026936027],
[[Does SibSp contains 0?, Does Pclass contains 3?],
 1,
 0.23646723646723647,
 0.23796033994334279,
 0.39393939393939393]]
```

In [108]:
```
answerSortedLaplace = deepcopy(answerTable)
answerSortedLaplace.sort(key = lambda x : x[3], reverse = True)
answerSortedLaplace
```

Out[108]:
```
[[[Does Sex contains female?],
  1,
  0.7420382165605095,
  0.740506329113924,
  0.35241301907968575],
 [[Does SibSp contains 0?, Does Parch contains 2?],
  1,
  0.7241379310344828,
  0.7096774193548387,
  0.03254769921436588],
 [[Does Pclass contains 1?],
  1,
  0.6296296296296297,
  0.6284403669724771,
  0.24242424242424243],
 [[Does SibSp contains 2?, Does Embarked contains Q?],
  1,
```

```
 0.6666666666666666,
 0.6,
 0.003367003367003367],
[[Does Parch contains 1?, Does SibSp contains 1?],
 1,
 0.5964912280701754,
 0.5932203389830508,
 0.06397306397306397],
[[Does Parch contains 2?, Does Fare less than equal to 32.204207968574636?],
 1,
 0.5,
 0.5,
 0.04489337822671156],
[[Does Pclass contains 2?, Does Embarked contains S?],
 1,
 0.4634146341463415,
 0.463855421686747,
 0.1840628507295174],
[[Does SibSp contains 1?, Does Embarked contains S?, Does Parch contains 0?],
 1,
 0.42857142857142855,
 0.43037974683544306,
 0.08641975308641975],
[[Does Embarked contains C?, Does Pclass contains 3?],
 1,
 0.3787878787878788,
 0.38235294117647056,
 0.07407407407407407],
[[Does SibSp contains 0?, Does Parch contains 0?],
 1,
 0.30353817504655495,
 0.3042671614100185,
 0.6026936026936027],
[[Does SibSp contains 0?, Does Pclass contains 3?],
 1,
 0.23646723646723647,
 0.23796033994334279,
 0.3939393939393939]]
```

In [109]:
```
answerSortedCoverage = deepcopy(answerTable)
answerSortedCoverage.sort(key = lambda x : x[3], reverse = True)
answerSortedCoverage
```

Out[109]:
```
[[[Does Sex contains female?],
  1,
  0.7420382165605095,
  0.740506329113924,
  0.35241301907968575],
```

```
[[Does SibSp contains 0?, Does Parch contains 2?],
 1,
 0.7241379310344828,
 0.7096774193548387,
 0.03254769921436588],
[[Does Pclass contains 1?],
 1,
 0.6296296296296297,
 0.6284403669724771,
 0.24242424242424243],
[[Does SibSp contains 2?, Does Embarked contains Q?],
 1,
 0.6666666666666666,
 0.6,
 0.003367003367003367],
[[Does Parch contains 1?, Does SibSp contains 1?],
 1,
 0.5964912280701754,
 0.5932203389830508,
 0.06397306397306397],
[[Does Parch contains 2?, Does Fare less than equal to 32.2042079685746367],
 1,
 0.5,
 0.5,
 0.04489337822671156],
[[Does Pclass contains 2?, Does Embarked contains S?],
 1,
 0.4634146341463415,
 0.463855421686747,
 0.1840628507295174],
[[Does SibSp contains 1?, Does Embarked contains S?, Does Parch contains 0?],
 1,
 0.42857142857142855,
 0.43037974683544306,
 0.08641975308641975],
[[Does Embarked contains C?, Does Pclass contains 3?],
 1,
 0.3787878787878788,
 0.38235294117647056,
 0.07407407407407407],
[[Does SibSp contains 0?, Does Parch contains 0?],
 1,
 0.30353817504655495,
 0.3042671614100185,
 0.6026936026936027],
[[Does SibSp contains 0?, Does Pclass contains 3?],
 1,
 0.23646723646723647,
```

```
0.23796033994334279,
0.39393939393939]]
```