p3

September 19, 2024

[2]:
```
import math

# open and read in ratings
f = open('./ratings.txt', 'r')
ratings = []
for line in f:
    rating = line.strip().split()
    ratings.append(rating)

# class Node as described in textbook
class Node:
    def __init__(self, f, l, r, s):
        self.feature= f
        self.left = l
        self.right = r
        self.score = s


# class Leaf as described in textbook
class Leaf:
    def __init__(self, guess):
        self.guess = guess
        self.right = None
        self.left = None


# compute entropy uncertainty (tree train helper)
def entropy_uncertainty(p):
    if (p == 0):
        return 0
    else:
        return p*math.log2(1/p)

# encode rating features
rf = [1, 2, 3, 4, 5]

# compute entropy uncertainty for features (tree train helper)
```

```
def feature_uncertainty(uncf, data, positive, p, n):
    for f in rf:
        unc = None
        no = None
        yes = None
        for d in data:
            no = [d for d in data if d[f] == 'n']
            yes = [d for d in data if d[f] == 'y']
            no_pos = [n for n in no if n[0] in positive]
            no_neg = [n for n in no if n[0] not in positive]
            yes_pos = [y for y in yes if y[0] in positive]
            yes_neg = [y for y in yes if y[0] not in positive]

        n_ratio = len(no) / (len(no) + len(yes))
        y_ratio = len(yes) / (len(no) + len(yes))

        no_pos_p = len(no_pos) / len(no)
        no_neg_p = len(no_neg) / len(no)

        yes_pos_p = len(yes_pos) / len(yes)
        yes_neg_p = len(yes_neg) / len(yes)

        ul = entropy_uncertainty(no_pos_p) + entropy_uncertainty(no_neg_p)
        ur = entropy_uncertainty(yes_pos_p) + entropy_uncertainty(yes_neg_p)

        pp = p/(p+n)
        np = n/(p+n)
        us = entropy_uncertainty(pp) + entropy_uncertainty(np)

        score = us - (n_ratio*ul + y_ratio*ur)
        uncf[f] = score

# select highest scoring feature for the new decision node (tree train helper)
def select_best_feature(uncf):
    max = None
    feature = None
    for unc in uncf:
        if unc is not None:
            max = unc
            break
    for unc in uncf:
        if unc is None:
            continue
        if unc > max:
            max = unc
            feature = uncf.index(max)
    return feature, max
```

```
# decision tree train implemented based off model pseudocode in textbook
def decisionTreeTrain(data, rf, maxdepth):
    p = 0
    n = 0
    guess = None
    positive = ['+2', '+1', '0']
    for d in data:
        if d[0] in positive:
            p += 1
        else:
            n += 1
    if p >= n:
        guess = 'positive'
    else:
        guess = 'negative'
    if p == len(data) or n == len(data):
        return Leaf(guess)
    elif maxdepth == 2:
        return Leaf(guess)
    elif not rf:
        return Leaf(guess)
    else:
        uncf = [None, None, None, None, None, None]
        feature_uncertainty(uncf, data, positive, p, n)
        feature, max = select_best_feature(uncf)
        for d in data:
            no = [d for d in data if d[feature] == 'n']
            yes = [d for d in data if d[feature] == 'y']
            rf.remove(feature)
            maxdepth += 1
            left = decisionTreeTrain(no, rf, maxdepth)
            right = decisionTreeTrain(yes, rf, maxdepth)
            return Node(feature, left, right, max)

# train decision tree with max depth 2
n = decisionTreeTrain(ratings, rf, 0)

FEATURE = [None, 'easy?', 'AI?', 'Sys?', 'Thy?', 'Morning?']

print(FEATURE[n.feature] + ' Uncertainty Score: ' + str(n.score))
print(n.left.guess)
print(FEATURE[n.right.feature] + ' Uncertainty Score: ' + str(n.right.score))
print(n.right.left.guess)
print(n.right.right.guess)
```

```
Sys? Uncertainty Score: 0.6099865470109875
positive
```

```
AI? Uncertainty Score: 0.3219280948873623
negative
positive
```