Cs441 Writeup

Yiming Lin

yl6@pdx.edu

Code of Naïve Bayes Classifier

1.1 Bayes' Rule

According to Bayes' rule:

$$P(H|E) = \frac{p(H)\prod_i P(E_i|H)}{\prod_i P(E_i)}$$

We compare the probability of $P(H_1|E)$ and $P(H_0|E)$ where H_0 is the event that patient's heart is abnormal (0), and H_1 is the event that patient's heart is normal (1). Thus, we are comparing the following:

$$P(H_0|E) = \frac{P(H_0)\prod_i P(E_i|H_0)}{\prod_i P(E_i)} \text{ and } P(H_1|E) = \frac{P(H_0)\prod_i P(E_i|H_1)}{\prod_i P(E_i)}$$

The denominators between those two are same, then comparing

$$P(H_0)\prod_i P(E_i|H_0) \text{ and } P(H_1)\prod_i P(E_i|H_1)$$

Log is used to avoid underflow, then comparing

$$\log P(H_0) + \sum_i \log P(E_i|H_0) \text{ and } \log P(H_1) + \sum_i \log P(E_i|H_1)$$

The following section describes coding for obtaining $\log P(H_0)$, $\log P(H_1)$, $\log P(E_i|H_0)$, and $\log P(E_i|H_1)$, where i is the index of feature.

1.2 Code representation of Bayes' rule

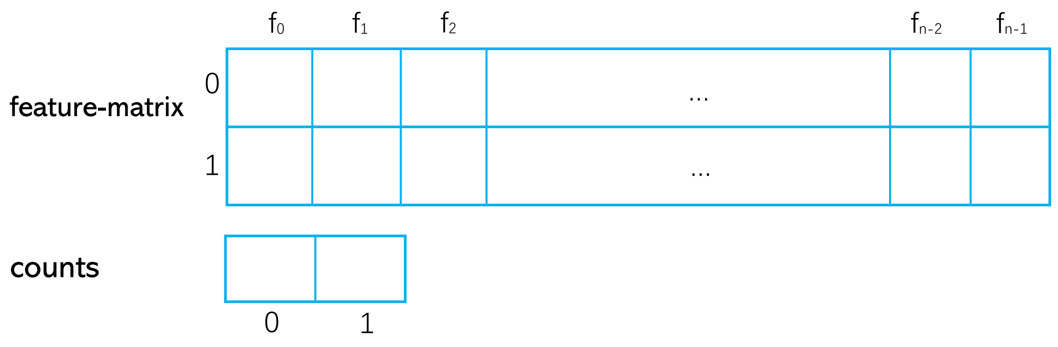

Figure 1 Two 2-d arrays used in this program

The feature matrix is a 2-d array such that the left dimension is the classes of dataentreis, and the top dimension is the feature indices. Each element in this 2-d array stores the number of "1"s appears in the training dataentries, given the feature index, and given the class of that data entries.

The counts matrix is a 1-d array whose length is equal to the number of classes of dataentries. In this problem, there are 2 classes (normal, abnormal), then the length is 2. The element of counts matrix stores the number of dataentries of that certain class in the training set.

During the training phrase, all the dataentries in training set is traversed, and the total number of “1”s in training set for each feature and each class, is obtained, so the feature matrix is fully populated. Counting happens in the same time, so the counts matrix is fully populated.

In addition, to avoid multiplying zeros in the Bayes’ rule, m-estimation is used (choose m = 0.5).

$$\log P(H_0) = \log \frac{\# \, of \ dataentries \ with \ class \ 0 \ + 0.5}{total \ \# \, of \ dataentries \ in \ training \ set + 0.5}$$

$$\log P(H_1) = \log \frac{\# \, of \ dataentries \ with \ class \ 1 + 0.5}{total \ \# \, of \ dataentries \ in \ training \ set + 0.5}$$

$$\log P(E_i = 1|H_0) = \log \frac{\# \, of \ dataentries \ whose \ feature \ i \ is \ 1 \ given \ class \ 0 + 0.5}{total \ \# \, of \ dataentries \ with \ class \ 0 \ in \ training \ set + 0.5}$$

$$\log P(E_i = 1|H_1) = \log \frac{\# \, of \ dataentries \ whose \ feature \ i \ is \ 1 \ given \ class \ 1 + 0.5}{total \ \# \, of \ dataentries \ with \ class \ 1 \ in \ training \ set + 0.5}$$

$$\log P(E_i = 0|H_0) = \log\,[1 - P(E_i = 1|H_0)]$$

$$\log P(E_i = 0|H_1) = \log\,[1 - P(E_i = 1|H_1)]$$

Next, translate the above 6 formula into 2-d array’s pseudo code.

```
log-p0 ← log(counts[0] + 0.5) - log(counts[0] + counts[1] + 0.5)
log-p1 ← log(counts[1] + 0.5) - log(counts[0] + counts[1] + 0.5)
```

```
function log-p-given-class(class, feature-index, evidence):
    // count is the # of dataentries with feature “arg: feature-index” is 1 given class “arg: class”
    count ← feature-matrix[class][feature-index]
    // when evidence is 0 given feature “arg: feature-index”, count the complemental part.
    if evidence == 0:
        count ← counts[class] – count
    return log(count + 0.5) – log(counts[class] + 0.5)
```

```
function classify(dataentry, label):
    likelihood-0 ← log-p0
    loop feature in 0..n-1:
        likelihood-0 ← likelihood-0 + log-p-given-class(0, feature, dataentry[feature])
    likelihood-1 ← log-p1
    loop feature in 0..n-1:
        likelihood-1 ← likelihood-1 + log-p-given-class(1, feature, dataentry[feature])
    if label == 0:
        if likelihood-0 > likelihood-1:
            return "true negative"
        else:
            return "false negative"
    else:
        if likelihood-1 > likelihood-0:
            return "true positive"
        else:
            return "false positive"
```

```
function training(entries, labels):
    feature-matrix ← [zeros(num_of_features), zeros(num_of_features)]
    counts ← [0,0]
    loop e in 0..num_of_entries-1:
        loop f in 0..num_of_features-1:
            if entries[e][f] == 1:
                feature-matrix[labels[e]][f] ← feature-matrix[labels[e]][f] + 1
        counts[labels[e]] ← counts[labels[e]] + 1
```

Results and Discussion

2.1 Results

orig 142/187(0.7593582887700535) 10/15(0.6666666666666666) 132/172(0.7674418604651163)

itg 145/187(0.7754010695187166) 15/15(1.0) 130/172(0.7558139534883721)

resplit-itg 63/90(0.7) 17/19(0.8947368421052632) 46/71(0.647887323943662)

resplit 78/90(0.8666666666666667) 17/19(0.8947368421052632) 61/71(0.8591549295774648)

2.2 Discussion

The results can be interpreted as the following:

- true positive: the heart is normal, and the classifier indicates it is normal
- true negative: the heart is abnormal, and the classifier indicates it is abnormal
- false positive: the heart is abnormal, but the classifier indicates it is normal
- false negative: the heart is normal, but the classifier indicates it is abnormal

In this application, the accuracy on abnormal instances is more important. Because if the classifier classifies an abnormal instance to be normal, the patient can never know that his heart is abnormal, which is dangerous. The “resplit” dataset gives the best result on accuracy. The “itg” dataset gives the equally best result on true negative rate.