Problem set 2

Brian Sherrill

March 21, 2017

1. Naive Bayes I

a.

Lets redefine sgn function real quick, so it fits better into the problem.

$$sgn(x) = 1 \text{ if } x > 0 \text{ else } -1$$

Now lets define the linear threshold function. let $\vec{x} = [x_1, x_2, \ldots x_n]$, where x_i is the ith observed value. then $\vec{w} = [1, 1, \ldots 1]$ i.e. an array of just the value 1. Then if we set $\theta = (m-1)$, we have everything we need.

$$f(\vec{x}) = sgn(\vec{w} \cdot \vec{x} - (m-1))$$

Proof of concept, let $\vec{x} = [1, 1, 1, 0]$.

Case 1: $m = 4$, $sgn(\vec{w} \cdot \vec{x} - (m-1)) = sgn(3-3) = sgn(0) = -1$

Case 2: $m = 3$, $sgn(\vec{w} \cdot \vec{x} - (m-1)) = sgn(3-2) = sgn(1) = 1$

Case 3: $m = 2$, $sgn(\vec{w} \cdot \vec{x} - (m-1)) = sgn(3-1) = sgn(2) = 1$

b.

Okay lets calculate each portion of the equation. lets do $p(y=1)/p(y=0)$ first.

$$p(y=1) = \frac{\binom{8}{3} + \binom{8}{4} + \ldots + \binom{8}{8}}{2^8}$$

$$p(y=0) = \frac{\binom{8}{0} + \binom{8}{1} + \ldots + \binom{8}{2}}{2^8}$$

Therefore

$$\frac{p(y=1)}{p(y=0)} = \frac{\binom{8}{3} + \binom{8}{4} + \ldots + \binom{8}{8}}{\binom{8}{0} + \binom{8}{1} + \ldots + \binom{2}{8}}$$

Now lets do u_i. I'll enumerate over the possible cases combinations and sum their probabilities. For example case 1 is $[1,1,1,1,1,1,1,1]$ so $p(x_i = 1|[1,1,1,1,1,1,1,1]) = 8/8$. The probability of this case, given $y = 1$, is (combinations / total possible combinations) $= \frac{\binom{8}{8}}{\binom{8}{3}+\binom{8}{4}+\ldots+\binom{8}{8}}$.
Case 2 is seven 1s and one 0, one example is $[0,1,1,1,1,1,1,1]$. so $p(x_i = 1|\text{seven 1 and one 0}) = 7/8$. The probability of this case, given $y = 1$, is (combinations / total possible combinations) $= \frac{\binom{7}{8}}{\binom{8}{3}+\binom{8}{4}+\ldots+\binom{8}{8}}$
This pattern continues until we reach the last valid case, three 1s and five 0s.

$$u_i = p(x_i = 1|y=1) = \frac{8}{8}\frac{\binom{8}{8}}{\binom{8}{3} + \binom{8}{4} + \ldots + \binom{8}{8}} + \frac{7}{8}\frac{\binom{7}{8}}{\binom{8}{3} + \binom{8}{4} + \ldots + \binom{8}{8}} + \ldots$$

Generalizing

$$u_i = \sum_{i=3}^{8} \frac{i}{8} \frac{\binom{i}{8}}{\binom{8}{3} + \binom{8}{4} + ... + \binom{8}{8}}$$

The same process can be applied to χ_i

$$\chi_i = \sum_{i=0}^{2} \frac{i}{8} \frac{\binom{i}{8}}{\binom{8}{0} + \binom{8}{1} + ... + \binom{8}{2}}$$

At this point you can then run the numbers and plug into the classifier.

c.

Ignoring my potentially incorrect logic above, I believe the naive bayes algorithm should be able to learn the target function. The weighted terms, i.e. the terms multiplied by x_i should contribute a constant value to the left side of the equation for every 1 in the passed vector. the value the weighted terms contribute must be of opposite sign of the constant terms. The constant terms, $log(\frac{p(y=1)}{p(y=0)}) + \sum_{i=1}^{n} log(\frac{1-u_i}{1-\chi_i})$ should function the same as θ. The result of the left side of the equation would be greater than 0 if three or more 1s exist in the vector, and less than 0 otherwise. As you can probably tell, I believe naive bayes will work analogous a general linear function.

2. Naive Bayes II

a.

See code.

b.

Used file as is, did not modify algorithm

c.

```
1:  5 1 0 0 0 0 0 0 (5 / 6 = 0.8333333333333333)
2:  1 29 0 0 0 0 0 15 (29 / 45 = 0.6444444444444444)
3:  0 3 0 0 0 0 0 3 (0 / 6 = 0)
4:  0 2 0 0 0 0 0 0 (0 / 2 = 0)
5:  0 5 0 0 1 0 0 6 (1 / 12 = 0.08333333333333333)
6:  0 6 0 0 0 0 0 25 (0 / 31 = 0)
7:  3 2 0 0 0 0 0 12 (0 / 17 = 0)
8:  4 1 0 0 0 0 0 146 (146 / 151 = 0.966887417218543)
181 / 270 = 0.67037037037037)
```

d.

My algorithm only preformed with 67 accuracy, although I heard people in class mention they better performance. It seemed to have trouble identify 3.0 - 7.0 labels. It seems the majority of misclassification where classifying into the 8.0 bucket. I suspect this might be because 8.0 was highly represented in the sample, and as such it not only had a lot of words which could cause overlap, but also just the prior for 8.0, i.e. p(8.0) is much higher than other labels.

2. Logistic Regression

a.

See code.

b.

Used file as is, did not modify algorithm

c.

Accuracy ranged between 85% and 92%. The confusion matrix below is for 92%

```
2:  45 3 (45 / 48 = 0.9375)
6:  3 25 (25 / 28 = 0.8929)
70 / 76 = 0.9211)
```

d.

The algorithm seems to have preformed well. It misclassified 6.0 labels more than 2.0 labels. Despite the large dimensionality, which I think was around 3000, it ran quick. I did not take the time to toy around with hyper parameters, but it's quite possible they could improve the accuracy.