CS 573 Homework 2

Yifan Fei

Computer Science
Purdue University

Instructor: Prof. Bruno Ribeiro

Spring 2018

Section 0:

1

Yes, I discussed the homework with others but came up with my own answers. Their name(s) are Yupeng Han and Meng Liu.

2

I have used online resources to help me answer this question, but I came up with my own answers. Here is a list of the websites I have used in this homework:

logistic regression is linear classifer: https://stats.stackexchange.com/questions/93569/why-is-logistic-regression-a-linear-classifier

Naive Bayes classifier is linear classifer: https://stackoverflow.com/questions/41682781/why-does-naive-bayes-fail-to-solve-xorl

Naive Bayes classifier is linear classifer: https://svivek.com/teaching/machine-learning/fall2017/slides/prob-learning/naive-bayes-linear.pdf

Q1: Ensembles

1

Model M_2 will have better performance in practice.

Since bagging attempts to reduce the chance overfitting complex models, the drawback of Model M_2 will be prevented to some degree. M_1 model is too simple so it may have big bias.

2

$$e_{single,k} = E[\sum_i \epsilon_{k,i}]$$

$$f^*(x) = \frac{1}{K}\sum_{k=1}^{K} f_k(x)$$

$$e_{bagging} = E[\sum_i \epsilon'_{k,i}]$$

$$\epsilon'_{k,i} = (y_i - f^*(x_i))^2$$

So,

$$e_{bagging} = E[\sum_i (y_i - f^*(x_i))^2]$$

$$e_{bagging} = E[\sum_i (y_i - \frac{1}{K}\sum_{k=1}^{K} f_k(x))^2]$$

$$e_{bagging} = E[\sum_i \frac{1}{K^2}(\sum_{k=1}^{K}(y_i - f_k(x)))^2]$$

$$e_{bagging} = \frac{1}{K^2}\sum_{k=1}^{K} E[(\sum_i (y_i - f_k(x)))^2]$$

from Cauchy inequality:

$$K\sum_{k=1}^{K}(y_i - f(x_i))^2 \geq (\sum_{k=1}^{K}(y_i - f(x_i)))^2$$

So,

$$e_{bagging} = \frac{1}{K^2}\sum_{k=1}^{K} E[(\sum_i (y_i - f_k(x)))^2] \leq \frac{1}{K^2}\sum_i E[K\sum_{k=1}^{K}(y_i - f^*(x_i))^2] = \frac{1}{K}\sum_{k=1}^{K} e_{single,k}$$

i.e:

$$e_{bagging} \leq \frac{1}{K}\sum_{k=1}^{K} e_{single,k}$$

Assume i.i.d. for single errors, then:

$$E[e_{bagging}] \leq E[e_{single}]$$

$$Var[e_{bagging}] \leq \frac{1}{K}Var[e_{single}]$$

We can see that the mean of error after bagging is not bigger than before, and the variance of the error after bagging is smaller.

Q2: Classification Tasks

a: training a decision tree

Yes. See the example below.

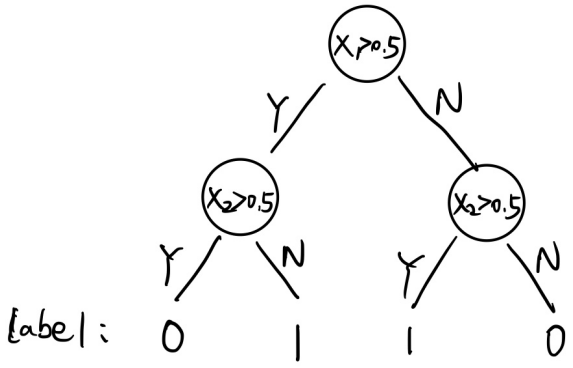

Figure 1: Decision tree with depth 2

b: training a logistic regression classifier

No.

Because logistic regression is a linear classifier with decision boundary as $\mathbf{w}^T\mathbf{x} = 0$ and XOR cannot be solved by a linear classifier.

There is no way you can draw a straight line that separates the two different outcomes. So linear classifiers, which model the class separation using straight lines, cannot solve problems of this nature.

Improvement to solve this issue: logistic regression can handle with the XOR problem by introducing a simple transformation to the feature space. Add the feature x_1x_2 to the data and you will get perfect accuracy. Adding this nonlinear feature allows logistic regression to learn a decision boundary that is linear in the features, but not in the original data space.

c: training a Naive Bayes classifier

No.

The provement of this binary naive Bayes classifier being a linear classifier is shown in problem 4.

There is no way you can draw a straight line that separates the two different outcomes. So linear classifiers, which model the class separation using straight lines, cannot solve problems of this nature.

Naive Bayes models independent events. Given only X and Y, it can model the distribution of x and it can model the y, but it does not model any relation between the two variables. To model the XOR function, the classifier would have to observe both variables at the same time. Only making a prediction based on the state of X without taking into account Y's state cannot lead to a proper solution for XOR problem.

Improvement to solve this issue: Similarly, add the feature x_1x_2 to the data and you will get perfect accuracy. Adding this nonlinear feature allows Naive Bayes to learn a decision boundary that is linear in the features, but not in the original data space.

Q3

a

Yes.

A good example is the following: assume your data is positive in $x \in [0, 1]$, and negative outside. A good tree would be the following:

1. If x > 1: return negative
2. If x <= 1:

   If x >= 0: return positive

   If x < 0: return negative

In this case, it can make sense to use x twice.

b

Wrong.

The information gain at the root node will always be greater than or equal to the weighted average information gain of next lower level nodes in the tree. It is possible that the information gain at the root node will be smaller than the information gain at some lower node in the tree. An example is shown below. In this case, $H_R > H(Y)$.

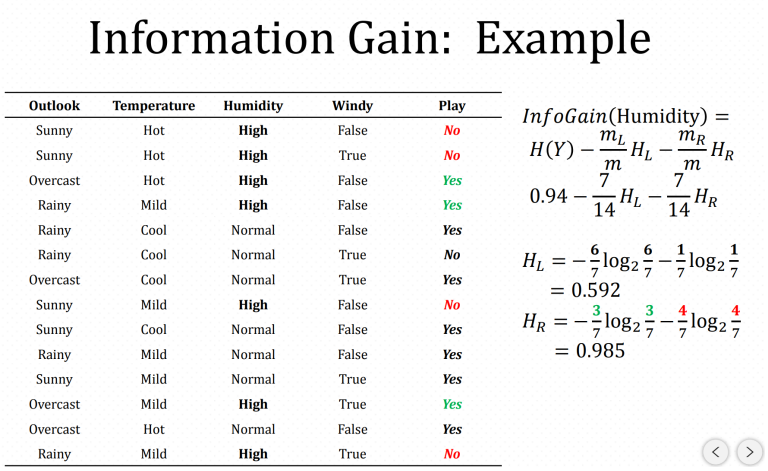

Figure 2: Counter example of information gain relationship

c: Random Forests

The probability that a particular feature does not get considered for splitting even once is:

$$p = ((\frac{m-1}{m})^Q)^P$$

$$p = (\frac{m-1}{m})^{QP}$$

d: Find the labels

Two missing labels are zero.

Assume the first missing label with (2,1) is a, and the second missing label with (1,0) is b.

$$H_{parent} = -\frac{3+a+b}{7}log(\frac{3+a+b}{7}) - \frac{4-(a+b)}{7}log(\frac{4-(a+b)}{7})$$

$H[X_1 = 0] = -\frac{1}{2}log(\frac{1}{2}) - \frac{1}{2}log(\frac{1}{2}) = -log(\frac{1}{2}) = 1$, $H[X_1 = 1] = -\frac{b}{2}log(\frac{b}{2}) - \frac{2-b}{2}log(\frac{2-b}{2})$,
$H[X_1 = 2] = -\frac{2+a}{3}log(\frac{2+a}{3}) - \frac{1-a}{3}log(\frac{1-a}{3})$
$H[X_2 = 0] = -\frac{1+b}{2}log(\frac{1+b}{2}) - \frac{1-b}{2}log(\frac{1-b}{2})$, $H[X_2 = 1] = -\frac{a}{3}log(\frac{a}{3}) - \frac{3-a}{3}log(\frac{3-a}{3}))$, $H[X_2 = 2] = 0$

$$IG(X_1) = H_{parent} - \frac{2}{7}H[X_1 = 0] - \frac{2}{7}H[X_1 = 1] - \frac{3}{7}H[X_1 = 2]$$

$$IG(X_2) = H_{parent} - \frac{2}{7}H[X_2 = 0] - \frac{3}{7}H[X_2 = 1] - \frac{2}{7}H[X_2 = 2]$$

,

Using Python to calculate 4 possible label combinations, we finally found when a = b = 1, $IG_1 = 0.29169$ and $IG_2 = 0.4695$.

Q4

a

$$P(y=1 \mid x) = \frac{\prod_{j=1}^{3} P(x_j|y=1)P(y=1)}{\prod_{j=1}^{3} P(x_j|y=1)P(y=1) + \prod_{j=1}^{3} P(x_j|y=0)P(y=0)} = \frac{7}{32}$$

$$P(y=0 \mid x) = \frac{\prod_{j=1}^{3} P(x_j|y=0)P(y=0)}{\prod_{j=1}^{3} P(x_j|y=1)P(y=1) + \prod_{j=1}^{3} P(x_j|y=0)P(y=0)} = \frac{25}{32} > P(y=1 \mid x)$$

So class 0 has highest posterior probability.

b: the decision boundary for the above Naive Bayes classifier

This binary naive Bayes classifier is a linear classifier. It will predict the label 1 if $P(y=1 \mid x) \geq P(y=0 \mid x)$. i.e.:

$$\frac{P(x \mid y=1)P(y=1)}{P(x \mid y=0)P(y=0)} \geq 1$$

By the naive Bayes assumption of conditional independence, we have $P(x|y) = \prod_{j=1}^{3} P(x_j|y)$. So:

$$\frac{P(y=1)}{P(y=0)} \prod_{j=1}^{3} \frac{P(x_j|y=1)}{P(x_j|y=0)} \geq 1$$

To simplify notation, let us denote $P(y=1)$ by p, $P(x_j=1|y=1)$ by a_j and $P(x_j=1|y=0)$ by b_j . Then:

$$P(x_j|y=1) = a_j^{x_j}(1-a_j)^{1-x_j}$$

$$P(x_j|y=0) = b_j^{x_j}(1-b_j)^{1-x_j}$$

Plugging into the above formula we get the following equivalent condition for predicting y = 1:

$$\frac{p}{1-p} \prod_{j=1}^{3} \frac{a_j^{x_j}(1-a_j)^{1-x_j}}{b_j^{x_j}(1-b_j)^{1-x_j}} \geq 1$$

Collecting the constants, we get:

$$(\frac{p}{1-p} \prod_{j=1}^{3} \frac{1-a_j}{1-b_j}) \prod_{j=1}^{3} (\frac{a_j}{b_j} \frac{1-b_j}{1-a_j})^{x_j} \geq 1$$

By taking log:

$$log(\frac{p}{1-p} \prod_{j=1}^{3} \frac{1-a_j}{1-b_j}) + \sum_{j=1}^{3} x_j log(\frac{a_j}{b_j} \frac{1-b_j}{1-a_j}) \geq 0$$

For this problem, $p = 4/10$, $a_1 = 1/2$, $a_2 = 3/10$, $a_3 = 1/5$, $b_1 = 1/3$, $b_2 = 1/4$, $b_3 = 2/3$ So the decision boundary is:

$$log(\frac{28}{25}) + log(2)x_1 + log(\frac{9}{7})x_2 + log(\frac{1}{8})x_3 = 0$$

When the label is 1:

$$log(\frac{28}{25}) + log(2)x_1 + log(\frac{9}{7})x_2 + log(\frac{1}{8})x_3 \geq 0$$

c

False. There will still be unavoidable error.

First, in Naive Bayes, Y is probabilistic, so it is often impossible to predict Y even if the model's estimate of P(Y) is perfect.

Another reason is that if two training examples, with the same features but different labels, then there is no way you can get 100% accuracy .

Q5

a

The label 1 example in the middle will have their weights increased since the decision stump with least error in first iteration is constant over the whole domain. Notice this decision stump only predicts incorrectly on the label 1 example, whereas any other decision stump predicts incorrectly on more training examples. So the label 1 example will have their weights increased.

b

i: decision stumps

We use sklearn package to help us calculate the minimum iterations to achieve zero error. In the parameters, we used SAMME discrete boosting algorithm, learning rate is 1.0, base estimator is DecisionTreeClassifier with depth 1. The minimum number of iterations to achieve zero training error with Adaboost and decision stumps over this data is 9.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import numpy as np

X = [[i,j] for i in range(6) for j in range(3)]
Y = [0 for i in range(9)]+ [1 for i in range(9)]
Y[4]=1

Ada = AdaBoostClassifier(base_estimator=None,
        n_estimators=1, learning_rate=1.0, algorithm='SAMME', random_state=None)
Ada.fit(X,Y)

iter = 1
while sum(Ada.predict(X)-Y) != 0:
        iter += 1
        Ada = AdaBoostClassifier(base_estimator=None,
        n_estimators=iter, learning_rate=1.0, algorithm='SAMME', random_state=None)
        Ada.fit(X,Y)
print(iter)
```

ii: depth two

We use sklearn package to help us calculate the minimum iterations to achieve zero error. In the parameters, we used SAMME discrete boosting algorithm, learning rate is 1.0, base estimator is DecisionTreeClassifier with depth 2. The

minimum number of iterations to achieve zero training error with Adaboost using decision trees with depth two over this data is 5.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import numpy as np

X = [[i,j] for i in range(6) for j in range(3)]
Y = [0 for i in range(9)]+ [1 for i in range(9)]
Y[4]=1

Ada = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
        n_estimators=1, learning_rate=1.0, algorithm='SAMME', random_state=None)
Ada.fit(X,Y)

iter = 1
while sum(Ada.predict(X)-Y) != 0:
        iter += 1
        Ada = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
        n_estimators=iter, learning_rate=1.0, algorithm='SAMME', random_state=None)
        Ada.fit(X,Y)
print(iter)
```

iii: advantage of depth two

According to the comparison between (i) and (ii), the advantage of using decision stumps against a depth-two tree in this dataset is to prevent overfitting since the tree with depth two achieve zero error faster even if the label is mislabeled.