assignment3_sol

chandan u

11/16/2016

Question 1

a

Naive bayes has been implemented including logic of "usecolumnofones".

'Naive Bayes': algs.NaiveBayes({'usecolumnones': False}),

'Naive Bayes Ones': # # # algs.NaiveBayes({'usecolumnones': True})

When we include the column with all ones an error is produced: because for the columnofones , the variance is zero (since all values are the same).

As per the gaussian formula if you substitute zero variance in the forumula it will give you :

```
"""/Users/chandanuppuluri/classwork/machine_learning/Assignment3/code/classalgorithms.py:102: RuntimeWa
```

return (1/(np.sqrt(2 * np.pi * variance))) * np.exp(- ((x - mean)**2 / (2 * variance)))""

So this will affect the computing probabilities of target zero, target one for a given data input. Hence the error spikes when we include the columnn of ones from 22(without ones) to 46 to 50 (when ones are inluded.)

And moreover I am using a different function then utils.calculateprobability and hence i am getting the error. If i use the function from utils it handles situations when variance is less $1e^{-3}$. And I am getting same output values.

b

The logsitic regression has been implemented as:

'Logistic Regression': algs.LogitReg(),

c

Neural network has been implemented with sigmoid transfer:

'Neural Network': algs.NeuralNet({'epochs': 1})

d

Naive bayes with column of ones:
Error for Naive Bayes Ones: 46.52

Random: Average error for Random: 49.44 +- 0.720683231385 Best parameters for Naive Bayes: {'usecolumnones': False}

Naive Bayes(without ones):
Average error for Naive Bayes: 25.34 +- 0.77612040303 Best parameters for Linear Regression: {'regwgt': 0.0}

Linear Regression:
Average error for Linear Regression: 24.54 +- 0.388060201515

Logistic Regression:
Best parameters for Logistic Regression: {'regwgt': 0.0, 'regularizer': 'None'} Average error for Logistic Regression: 23.89 +- 0.873135453409

NeuralNetwork:
Average error for Neural Network: 50.42 +- 0.77612040303

Question 3

a

The regularizers are directly applied to the first order gradients.

for l1 regularizer we change the gradient as follows:

```
First_Grad= np.dot(Xtrain.T, np.subtract(ytrain,p))- self.params['regwgt'] * utils.dl1(weights)
```

for l2 regularizer we change the gradient as follows: (we directly use the weights in prodcut with the regularizer value of 0.01 etc)

First_Grad= np.dot(Xtrain.T, np.subtract(ytrain,p))- 2 * self.params['regwgt'] * utils.dl2(weights)

b

A third regularizer would be to use a combination of the l1 and l2 regularizer together to change the gradient.

First_Grad= np.dot(Xtrain.T, np.subtract(ytrain,p)) - 2 *self.params['regwgt']* utils.dl2(weights) - self.params['regwgt'] * utils.dl1(weights)

c

Your model might overfit sometimes , so thats where regularizers come in . Overfitting is when your model starts learning too much from the data i.e the noise too and it will not generalize well.

To pervent the such ovefitting we use regularizers, where we penalize the function , in our case we penalize the gradient using the l1 and l2 and our own third approach.

The behaviour of the regularizers: (l1(lasso), l2(ridge), l3(using l1 + l2 at the same time))

vanilla vs l1(lasso):

Average error for Logistic Regression: 23.6 +- 1.66311514935 Average error for L1 logit: 22.88 +- 0.249467272403

The perfromance of the l1 increases in compare to the vanila logistic

vanilla vs l2(ridge):

Average error for Logistic Regression: 23.6 +- 1.66311514935 Average error for L2 logit: 22.88 +- 0.249467272403

The performance of the l2 increases compare to the vanilla logistic

vanilla vs l3(l1 + l2):

Average error for Logistic Regression: 23.6 +- 1.66311514935 Average error for L3 logit: 22.88 +- 0.249467272403

The performance of vanilla is not better than the combination of l1 and l2

So you can clearly with increased features using the susy complete dataset, we are able to observe that the performance of the l1 and l2 and the l3 are better than the vanilla regularizer.

question 2

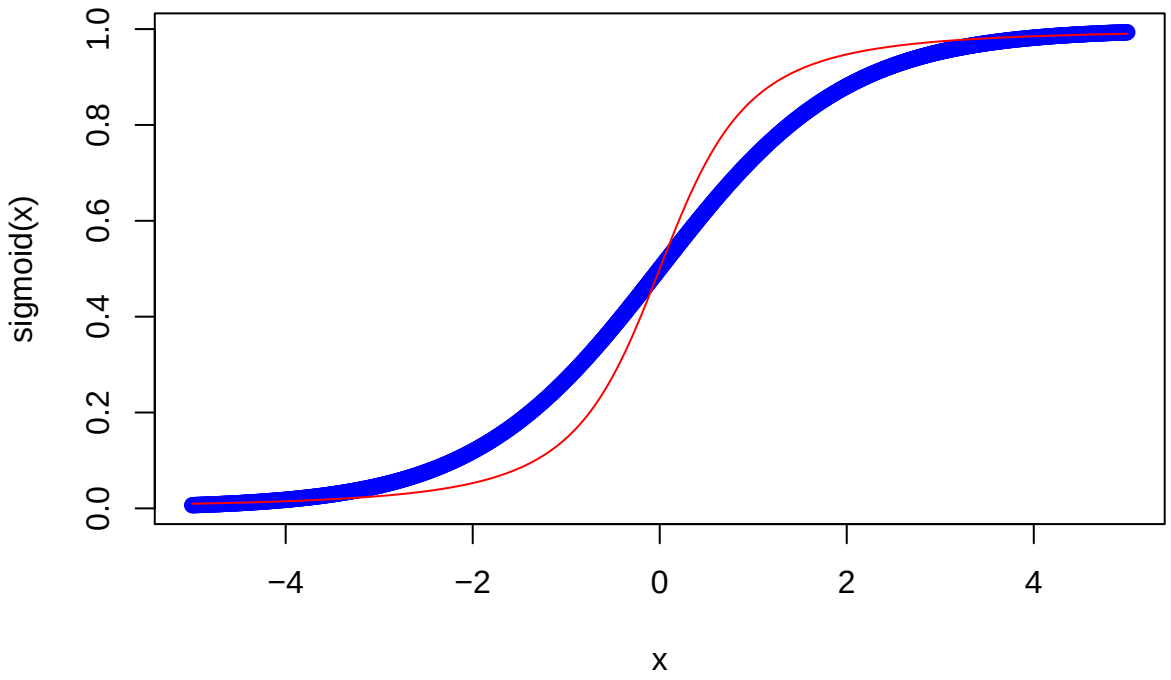
This given function behaves somewhat similar to the sigmoid functon used earlier. Lets plot a graph to see how these two functions behave:

```
sigmoid = function(x) {
   1 / (1 + exp(-x))
}

similar = function(x) {
  (0.5) * ( 1+ (x)/sqrt(1 + x^2))
}


x <- seq(-5, 5, 0.01)

plot(x, sigmoid(x), col='blue')
lines(x, similar(x), col='red')
```



As you can see both the functions sigmoid as well as the given function maintain the value between zero and one and hence behave similiarly.

So the error of the logistic regression with sigmoid is similar to the error when we use the given function.

The derivation of MLE: