Homework 3 CMPUT 466

Zhi Han

November 23, 2019

I was having issues with the provided template, so I wrote the assignment on a blank template.

You need to change the OS directory in utils.dataloader to get the script to work! (Dataloader doesn't work out of the box for windows, so I had to change it manually to load the data.)

The code is also availible at `https://github.com/hanzhihua72/cmput466/tree/master/a3barebones`

1 Question 1

a) Implemented Naive Bayes. The column of ones is on the last column, so I removed the column of ones if add ones is not kept.

```
(phys420) C:\Users\Zhi>C:/Users/Zhi/Anaconda3/envs/phys420/python.exe c:/Users/Zhi/Desktop/cmput466/a3barebones/script_classify.py
NaiveBayes Cross validate parameters : {'usecolumnones': True} error: 26.599999999999994 run 1/5, parameters 1/2
NaiveBayes Cross validate parameters : {'usecolumnones': False} error: 26.599999999999994 run 1/5, parameters 2/2
NaiveBayes Cross validate parameters : {'usecolumnones': True} error: 26.900000000000006 run 2/5, parameters 1/2
NaiveBayes Cross validate parameters : {'usecolumnones': False} error: 26.900000000000006 run 2/5, parameters 2/2
NaiveBayes Cross validate parameters : {'usecolumnones': True} error: 25.799999999999997 run 3/5, parameters 1/2
NaiveBayes Cross validate parameters : {'usecolumnones': False} error: 25.799999999999997 run 3/5, parameters 2/2
NaiveBayes Cross validate parameters : {'usecolumnones': True} error: 25.0 run 4/5, parameters 1/2
NaiveBayes Cross validate parameters : {'usecolumnones': False} error: 25.0 run 4/5, parameters 2/2
NaiveBayes Cross validate parameters : {'usecolumnones': True} error: 22.700000000000003 run 5/5, parameters 1/2
NaiveBayes Cross validate parameters : {'usecolumnones': False} error: 22.700000000000003 run 5/5, parameters 2/2
[{'average_error': 25.4,
  'name': 'NaiveBayes',
  'params': {'usecolumnones': True},
  'standard_error': 0.672309452558864},
 {'average_error': 25.4,
  'name': 'NaiveBayes',
  'params': {'usecolumnones': False},
  'standard_error': 0.672309452558864}]
```

Figure 1: Naive Bayes, Runs = 5

I found no difference between appending ones vs. not appending ones. This makes sense since Naive Bayes algorithm assumes that features are independent. This means no dependence between features.

Since all samples have the same value, and assuming that features are independent meaning that other features cannot affect the values of appending a column of ones, thus there is no information gained from appending a column of ones. So it makes no difference in the algorithm which is exactly what we found.

b) Implemented.

```
LogisticReg Cross validate parameters : {'stepsize': 1} error: 25.200000000000003 run 1/5, parameters 1/3
LogisticReg Cross validate parameters : {'stepsize': 0.1} error: 23.900000000000006 run 1/5, parameters 2/3
LogisticReg Cross validate parameters : {'stepsize': 0.01} error: 32.099999999999994 run 1/5, parameters 3/3
LogisticReg Cross validate parameters : {'stepsize': 1} error: 24.900000000000006 run 2/5, parameters 1/3
LogisticReg Cross validate parameters : {'stepsize': 0.1} error: 25.099999999999994 run 2/5, parameters 2/3
LogisticReg Cross validate parameters : {'stepsize': 0.01} error: 29.700000000000003 run 2/5, parameters 3/3
LogisticReg Cross validate parameters : {'stepsize': 1} error: 23.799999999999997 run 3/5, parameters 1/3
LogisticReg Cross validate parameters : {'stepsize': 0.1} error: 23.5 run 3/5, parameters 2/3
LogisticReg Cross validate parameters : {'stepsize': 0.01} error: 30.5 run 3/5, parameters 3/3
LogisticReg Cross validate parameters : {'stepsize': 1} error: 24.599999999999994 run 4/5, parameters 1/3
LogisticReg Cross validate parameters : {'stepsize': 0.1} error: 23.900000000000006 run 4/5, parameters 2/3
LogisticReg Cross validate parameters : {'stepsize': 0.01} error: 27.0 run 4/5, parameters 3/3
LogisticReg Cross validate parameters : {'stepsize': 1} error: 24.200000000000003 run 5/5, parameters 1/3
LogisticReg Cross validate parameters : {'stepsize': 0.1} error: 23.299999999999997 run 5/5, parameters 2/3
LogisticReg Cross validate parameters : {'stepsize': 0.01} error: 30.900000000000006 run 5/5, parameters 3/3
[{'average_error': 24.54,
  'name': 'LogisticReg',
  'params': {'stepsize': 1},
  'standard_error': 0.22199099080818657},
 {'average_error': 23.94,
  'name': 'LogisticReg',
  'params': {'stepsize': 0.1},
  'standard_error': 0.2794279871451669},
 {'average_error': 30.04,
  'name': 'LogisticReg',
  'params': {'stepsize': 0.01},
  'standard_error': 0.7629416753592632}]
```

Figure 2: Logistic Regression, Runs = 5

c) Implemented.

```
NeuralNet Cross validate parameters : {'epochs': 100, 'nh': 16} error: 20.39999999999999 run 5/5, parameters 3/3
[{'average_error': 23.020000000000003,
  'name': 'NeuralNet',
  'params': {'epochs': 100, 'nh': 4},
  'standard_error': 0.4013975585376676},
 {'average_error': 22.759999999999994,
  'name': 'NeuralNet',
  'params': {'epochs': 100, 'nh': 8},
  'standard_error': 0.4879344218232618},
 {'average_error': 23.299999999999997,
  'name': 'NeuralNet',
  'params': {'epochs': 100, 'nh': 16},
  'standard_error': 0.8148619514985348}]
```

Figure 3: Neural Net 1 hidden layer, Runs = 5

d) Implemented.

```
[[{'average_error': 25.4,
   'name': 'NaiveBayes',
   'params': {'usecolumnones': True},
   'standard_error': 0.672309452558864},
  {'average_error': 25.4,
   'name': 'NaiveBayes',
   'params': {'usecolumnones': False},
   'standard_error': 0.672309452558864}],
 [{'average_error': 24.54,
   'name': 'LogisticReg',
   'params': {'stepsize': 1},
   'standard_error': 0.22199099080818657},
  {'average_error': 23.94,
   'name': 'LogisticReg',
   'params': {'stepsize': 0.1},
   'standard_error': 0.2794279871451669},
  {'average_error': 30.04,
   'name': 'LogisticReg',
   'params': {'stepsize': 0.01},
   'standard_error': 0.7629416753592632}],
 [{'average_error': 23.020000000000003,
   'name': 'NeuralNet',
   'params': {'epochs': 100, 'nh': 4},
   'standard_error': 0.4013975585376676},
  {'average_error': 22.759999999999994,
   'name': 'NeuralNet',
   'params': {'epochs': 100, 'nh': 8},
   'standard_error': 0.4879344218232618},
  {'average_error': 23.299999999999997,
   'name': 'NeuralNet',
   'params': {'epochs': 100, 'nh': 16},
   'standard_error': 0.8148619514985348}],
```

Figure 4: Summary of 1a-1c model performances in cross validation Runs = 5

2 Question 2

a) Implemented. (Random Selection)

```
[{'average_error': 45.12,
  'name': 'KernelLogisticRegression',
  'params': {'centers': 10, 'stepsize': 0.01},
  'standard_error': 0.7966931655286104},
 {'average_error': 43.220000000000006,
  'name': 'KernelLogisticRegression',
  'params': {'centers': 20, 'stepsize': 0.01},
  'standard_error': 0.5962549790148496},
 {'average_error': 41.279999999999994,
  'name': 'KernelLogisticRegression',
  'params': {'centers': 40, 'stepsize': 0.01},
  'standard_error': 1.077552782929912},
 {'average_error': 31.820000000000004,
  'name': 'KernelLogisticRegression',
  'params': {'centers': 80, 'stepsize': 0.01},
  'standard_error': 1.2117425469133276}]
```

Figure 5: Kernel Logistic Regression on Susy over Runs = 5

b) Running the algorithm on census is has significantly less error than Kernel Linear Regression on susy. error 31% $\rightarrow$ 23%.

This is because the centers are selected based on Hamming distance, which I would expect is able to eliminate redundant centers, since the hamming distance of two of the same string is zero.

The algorithm is thus able to fit better, compared to random centers.

Note that I have made a new class called

`KernelLogisticRegressionCensus`

to handle the different census data.

```
[{'average_error': 23.18,
  'name': 'KernelLogisticRegressionCensus',
  'params': {'centers': 10, 'stepsize': 0.01},
  'standard_error': 0.7372380890865592},
 {'average_error': 22.119999999999997,
  'name': 'KernelLogisticRegressionCensus',
  'params': {'centers': 20, 'stepsize': 0.01},
  'standard_error': 0.5458204833093021},
 {'average_error': 21.999999999999996,
  'name': 'KernelLogisticRegressionCensus',
  'params': {'centers': 40, 'stepsize': 0.01},
  'standard_error': 0.4569463863518367},
 {'average_error': 21.419999999999995,
  'name': 'KernelLogisticRegressionCensus',
  'params': {'centers': 80, 'stepsize': 0.01},
  'standard_error': 0.42652080840212303}]
[[{'average_error': 23.18,
   'name': 'KernelLogisticRegressionCensus',
   'params': {'centers': 10, 'stepsize': 0.01},
   'standard_error': 0.7372380890865592},
  {'average_error': 22.119999999999997,
   'name': 'KernelLogisticRegressionCensus',
   'params': {'centers': 20, 'stepsize': 0.01},
   'standard_error': 0.5458204833093021},
  {'average_error': 21.999999999999996,
   'name': 'KernelLogisticRegressionCensus',
   'params': {'centers': 40, 'stepsize': 0.01},
   'standard_error': 0.4569463863518367},
  {'average_error': 21.419999999999995,
   'name': 'KernelLogisticRegressionCensus',
   'params': {'centers': 80, 'stepsize': 0.01},
   'standard_error': 0.42652080840212303}]]
Best parameters for Kernel Logistic Regression Census :
{'centers': 80, 'stepsize': 0.01}
Average error for Kernel Logistic Regression Census: 19.840000000000003+-0.0
```

Figure 6: Kernel Logistic Regression on Census Runs = 5

3 Summary of Models Performance (outside cross validation)

The data is run on susy with the best parameters from cross validation, k = 5 folds.

```
Average error for Naive Bayes: 26.099999999999994+-0.0
Average error for Logistic Regression: 23.760000000000005+-0.0
Average error for Neural Network: 22.439999999999998+-0.0
Average error for Kernel Logistic Regression: 29.879999999999995+-0.0
```

Figure 7: Summary of model performance (outside of cross validation)