Luke Jaffe
CS6140 Machine Learning
10/06/15

HW2 – Gradient Descent Learning

Problem 1.

A) Linear Ridge Regression (Normal Equations)
Usage: python regression.py -d [dataset]* -m 'normal' -r [ridge size]
*dataset can be 'data' or 'housing'

B) Linear Regression (Gradient Descent)
Usage: python regression.py -d [dataset] -m 'descent'

C) Logistic Regression (Gradient Descent)
Usage: python regression.py -d 'spam' -m 'logistic'

	Decision or Regression Tree	**Linear Regression (Normal Equations)**	**Linear Ridge Regression (Normal Equations)**	**Linear Regression (Gradient Descent)**	**Logistic Regression (Gradient Descent)**
Spambase	Train ACC: 0.9135 Test ACC: 0.9000	Train ACC: 0.8986 Test ACC: 0.8799	Train ACC: 0.9018 Test ACC: 0.8838	Train ACC: 0.9116 Test ACC: 0.9080	Train ACC: 0.9276 Test ACC: 0.9151
Housing	Train MSE: 15.8076 Test MSE: 25.3198	Train MSE: 22.08127319 Test MSE: 22.6382563	Train MSE: 22.24998403 Test MSE: 22.29217039	(10M iterations) Train MSE: 24.4922 Test MSE: 24.2013	N/A *

*We cannot apply logistic regression gradient descent to the housing dataset because it is not a classification problem

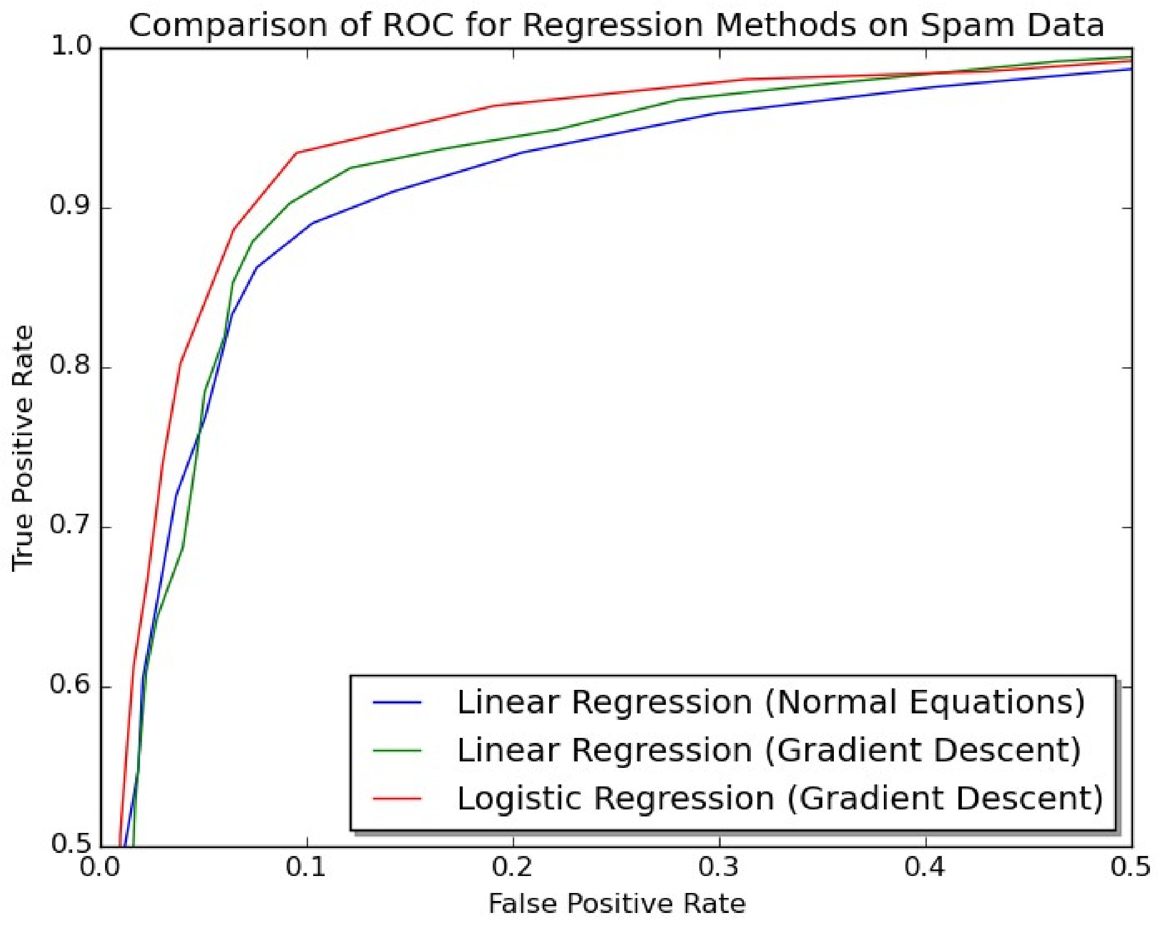
D) Confusion Matrices for Spambase Dataset
Usage: python regression.py -d 'spam' -m 'all'
kfolds = 10

a) Linear Ridge Regression (Normal Equations)
```
Fold 1: TPR = 0.821621621622 , FPR = 0.0436363636364 , Accuracy = 0.902173913043
Fold 2: TPR = 0.818652849741 , FPR = 0.0674157303371 , Accuracy = 0.884782608696
Fold 3: TPR = 0.786982248521 , FPR = 0.0309278350515 , Accuracy = 0.902173913043
Fold 4: TPR = 0.780748663102 , FPR = 0.040293040293 , Accuracy = 0.886956521739
Fold 5: TPR = 0.850267379679 , FPR = 0.043956043956 , Accuracy = 0.913043478261
Fold 6: TPR = 0.79792746114 , FPR = 0.0486891385768 , Accuracy = 0.886956521739
Fold 7: TPR = 0.786585365854 , FPR = 0.0337837837838 , Accuracy = 0.902173913043
Fold 8: TPR = 0.782608695652 , FPR = 0.036231884058 , Accuracy = 0.891304347826
Fold 9: TPR = 0.781818181818 , FPR = 0.0406779661017 , Accuracy = 0.895652173913
Fold 10: TPR = 0.795698924731 , FPR = 0.0620437956204 , Accuracy = 0.880434782609
Total: TPR = 0.800291139186 , FPR = 0.0447655581415 , Accuracy = 0.0880434782609
```

b) Linear Regression (Gradient Descent)
```
Fold 1: TPR = 0.919786096257 , FPR = 0.0659340659341 , Accuracy = 0.928260869565
Fold 2: TPR = 0.901162790698 , FPR = 0.100694444444 , Accuracy = 0.9
Fold 3: TPR = 0.854651162791 , FPR = 0.0763888888889 , Accuracy = 0.897826086957
Fold 4: TPR = 0.85393258427 , FPR = 0.0709219858156 , Accuracy = 0.9
Fold 5: TPR = 0.814814814815 , FPR = 0.0570469798658 , Accuracy = 0.897826086957
Fold 6: TPR = 0.877659574468 , FPR = 0.0367647058824 , Accuracy = 0.928260869565
Fold 7: TPR = 0.888372093023 , FPR = 0.0571428571429 , Accuracy = 0.917391304348
Fold 8: TPR = 0.890710382514 , FPR = 0.072202166065 , Accuracy = 0.913043478261
Fold 9: TPR = 0.905555555556 , FPR = 0.0571428571429 , Accuracy = 0.928260869565
Fold 10: TPR = 0.845714285714 , FPR = 0.0561403508772 , Accuracy = 0.90652173913
Total: TPR = 0.87523593401 , FPR = 0.0650379302059 , Accuracy = 0.090652173913
```

c) Logistic Regression (Gradient Descent)
```
Fold 1: TPR = 0.875706214689 , FPR = 0.0318021201413 , Accuracy = 0.932608695652
Fold 2: TPR = 0.915254237288 , FPR = 0.0494699646643 , Accuracy = 0.936956521739
Fold 3: TPR = 0.927374301676 , FPR = 0.0533807829181 , Accuracy = 0.939130434783
Fold 4: TPR = 0.927083333333 , FPR = 0.0559701492537 , Accuracy = 0.936956521739
Fold 5: TPR = 0.875 , FPR = 0.0674603174603 , Accuracy = 0.90652173913
Fold 6: TPR = 0.90625 , FPR = 0.0833333333333 , Accuracy = 0.913043478261
Fold 7: TPR = 0.887640449438 , FPR = 0.0567375886525 , Accuracy = 0.921739130435
Fold 8: TPR = 0.880681818182 , FPR = 0.0422535211268 , Accuracy = 0.928260869565
Fold 9: TPR = 0.931818181818 , FPR = 0.0492957746479 , Accuracy = 0.94347826087
Fold 10: TPR = 0.862433862434 , FPR = 0.0553505535055 , Accuracy = 0.910869565217
Total: TPR = 0.898924239886 , FPR = 0.0545054105704 , Accuracy = 0.0910869565217
```

E) ROC Plots for Spambase Dataset
Usage: python regression.py -d 'spam' -m 'all'



*Full plot truncated to upper-left quadrant to distinguish methods

a) Linear Ridge Regression (Normal Equations) AUC = 0.9495

b) Linear Regression (Gradient Descent) AUC = 0.9544

c) Logistic Regression (Gradient Descent) AUC = 0.9643

Problem 2.
Usage: python perceptron.py [learning rate]

With learning rate of .001:

```
Iteration 1, total_mistake 523
Iteration 2, total_mistake 264
Iteration 3, total_mistake 476
...
Iteration 68, total_mistake 1
Iteration 69, total_mistake 1
Iteration 70, total_mistake 0
```

```
Classifier weights: [-1.13972064,  0.19749955,  0.44465487,  0.6965424,   0.93514716]
```

```
Normalized with threshold: [-1.,  0.17328768,  0.39014373,  0.61115187,  0.82050559]
```

Problem 3.
Usage: python autoencoder.py

a) I used the posted algorithm for my implementation. Weight vectors and biases were randomly initialized between 0 and 1. The program regularly attains convergence on the correct output with learning rate of .02 in less than 20,000 iterations.

b) The purpose of the autoencoder training algorithm is to create a compressed, distributed representation of the data [1]. In this problem, the 8 inputs are compressed into 3 hidden nodes. Autoencoders are commonly used for dimensionality reduction; they condense a dataset into a feature vector representation, which can be returned to its original form with little loss. They have been used for unsupervised deep neural nets, by recursively training on the hidden layer output of previous autoencoders for the input data [2]. When used on images, autoencoders can extract useful features for object recognition and other computer vision tasks. Similarly, autoencoders have been used for phoneme recognition [3].

c) The number of autoencoder inputs and outputs is limited by 2 to the power of the number of hidden units. Let us recall that each hidden and output unit is a perceptron, the function of which is to perform a binary classification in a pointset. This means a single perceptron can only “encode” two values. Let us use the given problem as an example. The input/output is an inefficient representation of the numbers 0-7. We know that we can do better in binary, since 8 different values can be encoded with just 3 bits. Further, 2^n values can be encoded with n bits. As seen in the output to the problem, the correct hidden layer outputs can be rounded to the 8 different binary numbers which can be represented with 3 bits {{000}, {001}, {010}, {011}, {100}, {101}, {110}, {111}}. Now, let us answer the original questions. No, this encoder-decoder scheme would not work with one unit, because one unit could only be used to represent two unique values. With two hidden units, only four unique values could be respresented. Similarly, if there are more than 8 inputs and outputs, the autoencoder could converge to representing at most 8 of them, with 3 hidden units. ceiling($\log_2 n$) hidden units are needed for the autoencoder with n inputs/outputs.

Problem 4.

a) Let us consider a three-layer neural network with linear units. This network would have two sets of weights, one between the input and hidden layer, and one between the hidden and output layer. If the hidden vector is the product of the input and the first weight set, and the output vector is the product of the hidden vector and the second weight set, then clearly the output can also be written as the product of the two weight sets multiplied by the input. This weight set product could simply be described as a third, combined weight set. In this representation, we could eliminate the hidden layer altogether. Therefore, if the transfer function of the hidden units is linear, a three-layer network is equivalent to a two-layer one.

b) Let us assume that a two-layer neural network can solve the problem. In this case, the problem must be linearly separable by design. But the problem statement supposes problems which are non-linearly separable. Since we have shown that a two-layer neural network with linear hidden units has an equivalent representation in a three-layer one, by contradiction, we have proved that a three-layer network cannot solve a non-linearly separable problem such as XOR or n-parity.

Problem 5.

a) Andrew Ng Summary:

- Debugging diagnostics
    - Try a variety of methods to determine the issue
        - Try getting more training examples
        - Try smaller/larger feature set
        - Try changing features
        - Use more iterations in gradient descent
        - Try Newton's method
        - Use a different learning rate
        - Try a support vector machine
    - Better approach is to run diagnostics to find the problem, and fix it
    - Getting more training samples, or trying a smaller set of features fixes high variance
    - Trying a larger set of features of different features fixes high bias
    - Run gradient descent for more iterations of try newton's method to fix optimization algorithm
    - Use a different learning rate or try a support vector machine to fix optimization objective
    - Even if a learning algorithm is working well, you might also run diagnostics to make sure you understand how the learner is functioning
- Error analysis
    - Tries to explain the difference between current performance and perfect performance
    - Determine how much error is attributable to each of the components of the system
    - Plug in ground-truth for each component and see how this affects accuracy
- Ablative Analysis
    - Tries to explain the difference between some baseline and current performance
    - Determine how much each element of your learning system really helps the overall performance
- Getting started
    - Multiple approaches: careful design vs. build & fix
    - Careful design spend a long time feature engineering, collecting the right data, and designing effective algorithm architecture
        - Potentially more scalable, may come up with something neat or contribute to the field
    - Build and fix: Implement a learner quickly, then run error analyses and diagnostics
        - Gets a working solution more quickly
        - One way to find out what needs work is to implement quickly, and find out what parts break
    - Over-theorizing is dangerous, think Occam's razor

b) Pedro Domingos Summary:

- Machine learning is a developing field with a large body of heuristic methods
- Many of these heuristics, needed to successfully develop machine learning applications, are not readily available in the major textbooks
- Deciding which machine learning algorithm to use should be based on representation, evaluation, and optimization
  - Representation: A classifier must be represented in formal language workable with a computer
  - Evaluation: Mechanism to distinguish good classifiers from bad ones
  - Optimization: Method to search among the classifiers for the highest-scoring one
- The fundamental goal of machine learning is to generalize beyond examples in the training set
- No learning can beat random guessing over all possible functions to be learned
- When a learner outputs a classifier which has much higher classification rate on the training data than on the testing data, it has overfit
  - Bias is a learner's tendency to consistently learn the same thing wrong
  - Variance is the tendency to learn random things irrespective of the real signal
  - A linear learner has high bias and a decision tree has high variance
- One of the biggest problems in machine learning is the curse of dimensionality, in which tuition fails for high-dimensional input
- Machine learning papers are filled with heuristic guarantees that meant not as criteria for practical decisions, but as a source of understanding for algorithm design
- Feature engineering is essential for machine learning
  - If you have many independent features that each correlate well with the class, learning is easy
  - Building a strong learner is an iterative task, in which a developer runs the learner, analyzes the results, and modifies the data/learner
  - Automation of feature engineering is one of the holy grails of machine learning
- It is better to have more data than to have a more clever algorithm
- Model ensembles can produce a much more effective model at little cost
  - In bagging, random variations of the training set are generated and learned upon, and the results are combined by voting
  - In boosting, training examples have weights, and these are varied so that each new classifier focuses on the examples the previous ones tended to get wrong
- Simplicity does not imply accuracy, as shown by the efficacy of ensemble models
- Just because a function can be represented does not mean it can be learned
- Correlation does not imply causation

Works Cited

[1] Modeling word perception using the Elman network, Liou, C.-Y., Huang, J.-C. and Yang, W.-C., Neurocomputing, Volume 71, 3150–3157 (2008), doi:10.1016/j.neucom.2008.04.030

[2] Yoshua Bengio (2009), "Learning Deep Architectures for AI", Foundations and Trends® in Machine Learning: Vol. 2: No. 1, pp 1-127.

[3] Ng, Andrew (2011), “CS294A Lecture notes”, web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf