CSCI 447/547: Assignment 2

February 15, 2018

Due March 9th. Problems marked with an asterisk are only required for grad students. However, undergrads may undertake them at no penalty, and will receive extra credit for successful completion. Turn in responses and plots, as well as code.

1 Logistic Regression

1A 10 pts

Implement binary (two class) logistic regression for arbitrary input variables.

1B The Titanic: 20 pts

Use your logistic regression classifier to predict survival rates for the Titanic based on a number of features, both binary and continuous.

The training and test datasets for this problem are packaged with this document as titanic_train.csv and titanic_test.csv. It is a mix of different feature types. The features 'Sex' and 'Embark' (embarkation location) are both categorical, and should be converted to a 'one-hot' encoding (add a column to the data for each category and put a one in the column if a data point corresponds to that category). Counts (number of parents/children aboard 'Parch', number or siblings/spouses aboard 'SibSp'), and (almost) real values ('Age', 'Fare') do not need to be encoded differently, but must be normalized for the classifier to work properly (either $z-$normalize or scale between 0 and 1). There are also a number of categories that are either too sparse to be useful, or aren't likely to be predictive ('Cabin', 'Ticket', 'Name'). You should drop these from the dataset.

Note that logistic regression is sensitive to missing data, so if any of your data points are missing values drop the point.

Perform 10^4 training iterations (more than enough if your model is working correctly). When you believe the model has converged, supply the following:

1. The prediction error in the test and training sets.
2. A plot of the cost function as a function of iteration during the training phase.

Additionally, compute the test error you would have achieved if the training data had no features (only the labels: survival or not). Do the features add additional predictive power?

1C (*) 10 pts

So far, we have not discussed when to terminate learning. Instead, we've simply run gradient descent for a fixed number of iterations and evaluated whether the graph of the cost function looked like it had gotten flat. Brainstorm and implement some quantitative criterion for when the training phase should stop.

2 Neural Networks

2A 0 pts

Packaged with this homework is the same neural network code I applied to the Iris problem in class, broken into two files. neural_network.py contains the Network class, which implements a multilayer perceptron network with arbitrary number of layers, and several activation functions. iris_driver.py imports the Network class and runs the neural network. Have a look at both files, run the classifier, and ensure that it works.

2B 10 pts

The final layer and cost function for this problem are designed for softmax classification. Amend the code so that it can also be used for regression, i.e. change the final activation function to the identity, and implement the least squares cost function and its gradient. To test your implementation, generate some synthetic data ($m = 100$) according to the function

$$x \in [0, 1], \tag{1}$$

$$y = \exp^{-\sin(4\pi x^3)}. \tag{2}$$

Train your model on this data. A learning rate of $\eta = 10^{-3}$ worked for me, but depending on the details of your implementation, you may have to search around for a good value. It took me 10^5 iterations of gradient descent to get a good fit. You may wish to test your model on a simpler function to ensure that your scheme works.

2C 10 pts

Implement L_1 and L_2 regularization. Recall that L_1 regularization penalizes the sum of absolute values

$$\mathcal{J}_{L_1} = \gamma \sum_{ijl} |w_{ij}^{(l)}|, \tag{3}$$

and has gradient

$$\frac{\partial J_{L_1}}{\partial w_{ij}^{(l)}} = \gamma \operatorname{sign}(w_{ij}^{(l)}. \tag{4}$$

For L_2 regularization, we penalize the sum of squares

$$\mathcal{J}_{L_2} = \frac{\gamma}{2} \sum_{ijl} (w_{ij}^{(l)})^2, \tag{5}$$

and has gradient

$$\frac{\partial J_{L_2}}{\partial w_{ij}^{(l)}} = \gamma w_{ij}^{(l)}. \tag{6}$$

Try a few values of γ for each regularization method, and comment on the resulting classification accuracy for the iris dataset. How does regularization change the convergence rate?

2D 10 pts

Implement mini-batch stochastic gradient descent (i.e., at each step of gradient descent, compute gradients based on a random subset of the data). Compare the results of training with a sample size $s = 10$, versus a sample size $s = m$ (the entire dataset at once). Which method converges faster? Which method provides better test set error?

2E (*) 10 pts

Apply the neural network classifier to the MNIST dataset, which is a much larger and richer dataset than the 'digits' dataset we worked on last time. Note that MNIST comes as an array of integers between 0 and 255. You'll want to scale these to the unit interval. Using a single hidden layer with 300 nodes, I found a learning rate of $\eta = 10^{-1}$ and a sample size of $s = 10$ to provide better than 97% test accuracy. Report your test accuracy and a plot of your training error through time.

Repeat the above, but with the training data downsampled to 10% of its original size (training should go much faster). What is the test accuracy now? Comment on what could be causing the difference.

3 TensorFlow

3A 5 pts: Getting it working

Find the file tf_softmax.py in the homework archive. This is a TensorFlow implementation of softmax regression, configured to operate on the MNIST dataset. Verify that this code is working. It should provide classification accuracy of around 90%. Examine the optimal weights by reshaping them into 10 28x28 arrays (the same shape as the input images). Display one or two of these arrays, and comment on the spatial pattern that the optimization procedure is producing.

3B 15 pts

The above classifier is pretty plain. Turn it into a real neural network by adding a hidden layer with 300 sigmoid nodes. A useful command will be tf.sigmoid. After 2×10^5 iterations with a minibatch size of 10, I get around 97% accuracy. Add a second hidden layer, also with 300 sigmoid nodes. Is the extra model commplexity worth it? Turn in both your code and comments.