6.867: Homework 3

Essentially all problems in machine learning involve some form of optimization, whether to fit a model to data (estimation) or to select a prediction based on a model (decision). In a few cases we can find the optimum analytically; in many others we do it numerically. In this assignment we will begin to explore numerical methods for some of the basic estimation problems that we have been learning about.

You will find a zip file with some useful code and data in the Resources section of the Piazza course page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (Easy Chair) to have these papers peer reviewed (by the other students). This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions. The details of this process will be posted on Piazza.

Grading process and due dates

- We *strongly* encourage you to do this assignment in groups of two students, though you may do it individually if you wish. Post on Piazza or email a TA or instructor if you'd like help finding a partner.
- You submit your paper via the Easy Chair site by 11:59 PM on Tuesday, November 10.
- Each student who was an author or co-author on a submission will be assigned 3 papers to review.
- Reviews must be entered on the Easy Chair site by 11:59PM on Tuesday, November 17.
- All reviews will be made visible shortly thereafter, and students will have a 2–3 day period in which to enter rebuttals of their reviews into EasyChair if they wish.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the four parts of the assigment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?
- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%**: The average of all 8 scores on your assignment given by all three reviewers.
- **20%**: A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process
- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The following questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing.

1 Optional–Multi-Class SVM–Optional

Doing the implementation described in this section will give you a nice basis for comparison between SVMs and Neural Networks on an interesting data set. But if it's too much to get done, you can skip directly to section 2.

In the last homework, we focused on the task of binary classification where the prediction is either one class or the other. But in real world applications the number of classes in consideration may go far beyond two. The classification in this case is called *multi-class classification*. In this homework, we will extend the binary SVM from the previous homework to make predictions for problems with multiple classes.

Since SVM is only defined for binary classification we need to play tricks in order to make it work on K-class classification. The *one-versus-the-rest* approach involves defining K different SVM's. For each, the label of class k is set to $+1$, while all other class labels are set to -1. Once all of the classification functions have been created, new points can be classified. Then, if classifier k returns $+1$, then we say that this new point has class k. However, if multiple classifiers return $+1$, then you have to break the tie. In an SVM, you might pick the classifier that assigned the test point the largest margin.

You can read more about this technique in the included `multi_svm.pdf` excerpt from Bishop.

1. Multi-Class SVM:

   Use your dual-form SVM classifier from the last assignment to implement the *one-versus-the-rest* approach for SVM. Be sure that your code is capable of using different kernel functions, so as to perform linear/nonlinear classification.

2. Toy Data:

   We have included two toy data sets for you to test the performance of your classifier. Each data set includes three groups of data, with labels $c = \{1, 2, 3\}$. Use the linear kernel, $K(x, z) = x^T z$, and the 2-degree polynomial kernel, $K(x, z) = (x^T z + 1)^2$, to classify the data and use cross validation to determine the the values of the cost parameter C. Comment on the performance and decision boundaries of each (remember that you will have multiple decision boundaries).

3. MNIST-data:

   With the multi-class SVM written in the previous problem, you are now ready to enter the arena to solve real world AI problems! The MNIST database is a well-known set of handwritten digits (0-9) that is often used as a benchmark for classification algorithms. We have included `mnist_train/test/validate.csv` data sets for this purpose. You will notice that, since the data corresponds to a 28×28 image, each point has 784 dimensions! To save on time, we have only given you points with classes (1-6).

   Use the multi-class SVM code you implemented above on the included MNIST data using a linear kernel, $K(x, z) = x^T z$ and a polynomial kernel of degree 10, $K(x, z) = (x^T z + 1)^{10}$.

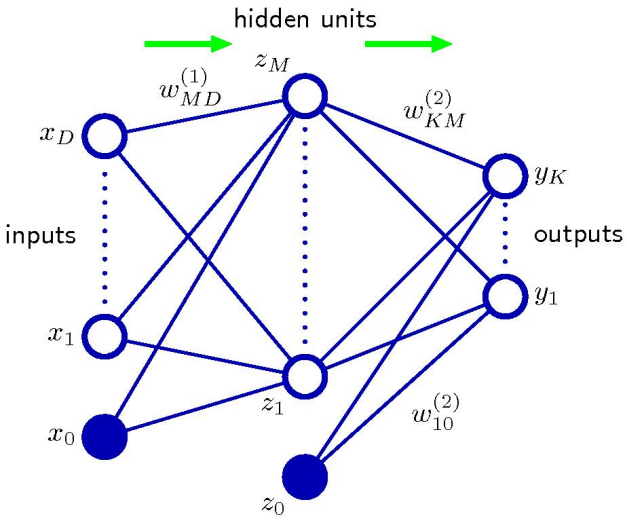
Since running the SVM solver may take a long time, you can just use a value of $C = 100$, rather than searching for a good one. Also, we found that it was not necessary to standardize the features.

2 Neural Networks

Now, we'll ask you to implement a simple 2-layer regularized neural network trained using gradient descent. The network will have weights $w^{(1)}$ between input and hidden layers and $w^{(2)}$ between hidden and output layers. Let the number of samples be N, the number of classes be K, and the activation function for both the hidden and output layers be the sigmoid

$$g(z) = \frac{1}{1 + e^{-z}}$$

The predicted value for output k given parameters $w = \{w^{(1)}, w^{(2)}\}$ and input x is written $h_k(x, w)$. The structure of the network is shown below (Bishop figure 5.1):



In the output layer we use one-of-many encoding of the labels; so, for any sample $x^{(i)}$ in class k, the corresponding one-of-many encoding has the output target be a k-dimensional vector $y^{(i)}$ that contains all -1 values, except for a 1 value in the kth dimension. The loss function, equal to the negative log likelihood, given training data and parameters w will be

$$\ell(w) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_k^{(i)} \log(h_k(x^{(i)}, w)) \quad .$$

It's usually the case in practice that optimizing this directly will result in over-fitting. To avoid overfitting, we add a regularization term to the loss function, and the final cost function becomes[1]

$$J(w) = \ell(w) + \lambda(\|w^{(1)}\|_F^2 + \|w^{(2)}\|_F^2)$$

where $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$ is the matrix Frobenius norm.

1. Gradient Calculation:

   Calculate analytic expressions for $\nabla_{w_1} J(w)$ and $\nabla_{w_2} J(w)$, the gradients of J with respect to the different sets of parameters (or use the gradient as we calculated it in lecture).

2. Implement 2-Layer Neural Network:

   Use the analytic gradient expressions together with your simple gradient descent procedure to train the neural network. Remember to do your implementation in sufficient generality so that you can vary the number of nodes in the hidden layer.

3. Stochastic gradient descent:

   Make a small change to your code so that it does *stochastic* gradient descent, doing separate updates based on individual training examples.

4. Test your Neural Network code:

   Use your neural network code on the 3-class toy data introduced in the previous question. Use cross-validation to determine the number of hidden nodes necessary for classification. Compare the results from batch and from stochastic gradient descent. Compare these results to those from your SVM classifier, discussing the feature spaces constructed in each case.

5. MNIST Data:

   Use your neural network to classify the included subset of the MNIST dataset. Be sure to use the analytic formula for the gradient in your optimization, as numerically calculating the gradient in a nearly 800 dimensional space will be far too expensive.

6. Apply your neural network on the provided MNIST dataset. Use `mnist_validate.csv` as validation dataset and change the number of hidden nodes, the trade-off parameter in regularization, the step-size, and batch/stochastic training to see how well you can perform on it. Discuss your choice of parameters.

[1] It's not really necessary to divide by N in ℓ; however, doing so means that the scale of the first term of J is independent of N and so you can use the same value of λ for batch as for stochastic gradient descent.