6.891 Machine learning and neural networks

Problem set 2

Deadline: October 12, in class

Note: You will need to use MATLAB in this problem set. Information about Matlab can be found on the course web site[1]. Data files for the problems will be made available via the course Athena locker: `/mit/6.891`.

Reading: Lecture notes 1-6, Chapters 1, 2.1-2.6, 5.1–5.8

Problem 1

Softmax model is a generalization of the logistic model for multi-way classification problems. Let $\mathbf{x} = [x_1, \ldots, x_d]^T$ be the input vector and suppose that the label y can take k distinct values $1, \ldots, k$. The softmax is then defined as follows

$$\begin{aligned} z_1 &= w_{10} + w_{11}x_1 + \ldots + w_{1d}x_d && (1) \\ &\cdots && (2) \\ z_k &= w_{k0} + w_{k1}x_1 + \ldots + w_{kd}x_d && (3) \\ P(y = i|\mathbf{x}, \mathbf{w}) &= \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} && (4) \end{aligned}$$

You should first convince yourself that when $k = 2$, softmax reduces to a logistic model.

a) Write the gradient ascent learning rule for softmax, i.e.,

$$w_{ij} \leftarrow w_{ij} + \epsilon \frac{\partial}{\partial w_{ij}} \log P(y|\mathbf{x}, \mathbf{w}) \qquad (5)$$

as a *delta rule*. In a delta rule the update is made on the basis of the difference between the target and the predictions. See the solutions to the previous problem set for an example.

[1] http://www.ai.mit.edu/courses/6.891

b) Consider the 3-way classification problem given in figure 1. We'd like to solve this classification problem with the softmax model. You can either construct a solution by hand or use the data in "threeway_x.dat" and "threeway_y.dat" and the delta rule. **Turn in the solution weights w and the MATLAB code (if any)**.

c) What type of classification problems can be solved with the softmax model? In other words, can you characterize the type of training sets that can be perfectly classified with the softmax model? **2 sentences suffices**.

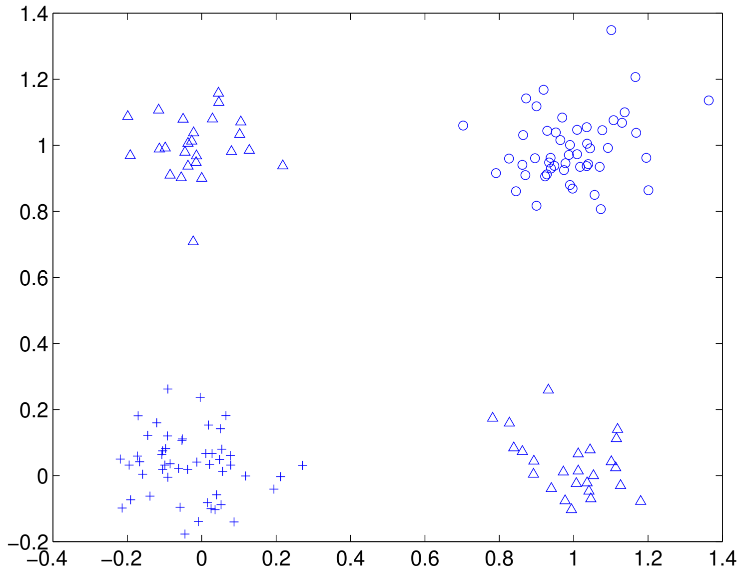

Figure 1: A 3-way classification problem

Problem 2

For this problem you can reuse most of the code you wrote for problem 2 of the previous problem set. You can also use the MATLAB code from the solutions if you prefer.

Please look over problem 2 in problem set #1 to remind yourself of the notation and details. You should be able to get by with small changes to the MATLAB code. There's no need to turn in your MATLAB code for this problem.

a) Set the learning rate to $0.1/\sqrt{nd}$ where n is the number of training examples and d is the dimensionality of the examples ($d = 64$). Rerun the gradient ascent learning rule for 1000 iterations and after each update compute and store the resulting classification error (number of mistakes) on the test set. **Plot the classification error as a function of the number of iterations and give us the figure. Provide a brief explanation for the figure.**

b) How would the figure change if we had a large number of training examples? **One sentence**.

c) Now add a quadratic regularizaton penalty to the log-likelihood criterion. In other words, we maximize

$$J(\mathbf{w}) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}, \mathbf{w}) - C \sum_{i=1}^{64} w_i^2/2 \tag{6}$$

Note that we ask that you DO NOT regularize the bias term w_0.

Modify the gradient ascent learning algorithm for this case and repeat part a) for $C = 1, 10, 100$. (*This shouldn't require much new code...*).

Give us the resulting figures and a brief account for for any differences between the figures.

d) Rerun part a) and store the squared norm $\sum_{i=1}^{64} w_i^2$ after each iteration. Plot this norm as a function of the number of iterations. What can you say about performing only a few gradient ascent iterations as a regularization method in comparison to adding a regularization penalty and not limiting the number of iterations? **2 sentences suffices**.

Problem 3

Here we continue the digit recognition problem but try to reduce the number of pixels that we have to include in the logistic regression model. There is no need to turn in your MATLAB code for this problem.

a) Let y,ϕ_1, and ϕ_2 be binary variables. We assume we have already computed $P(y, \phi_1, \phi_2)$. Suppose that

$$P(y|\phi_1, \phi_2) = P(y|\phi_1), \quad \text{for all } y, \phi_1, \phi_2 \in \{0, 1\} \tag{7}$$

Show that this implies that

$$I(y; \phi_1) = I(y; \phi_1, \phi_2) \tag{8}$$

where

$$I(y; \phi_1, \phi_2) = \sum_{y,\phi_1,\phi_2=0,1} P(y, \phi_1, \phi_2) \log \frac{P(y, \phi_1, \phi_2)}{P(y)P(\phi_1, \phi_2)} \tag{9}$$

You should not assume that ϕ_1 and ϕ_2 are independent.

b) We will use a simple filter method here. We have provided you with a script that finds the indices of the top 10 most informative components of the input vector (digit) according to the simple mutual information criterion $I(y; x_i)$, where x_i is a component of $\mathbf{x}$. The probabilities are estimated from the training set ("digit_x.dat" and "digit_y.dat").

   To get the relevant indices, try

```
I = topk(digit_x,digit_y,10)
```

   Run the logistic regression model (without regularization) as in problem 2a) but using only the 10 "best" components as the input vector. **Give us the resulting plot**. No explanation is needed.

c) For comparison, you should try a wrapper feature selection method. For this, we can add a 1-norm regularization penalty to the log-likelihood

$$J(\mathbf{w}) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}, \mathbf{w}) - C \sum_{i=1}^{64} |w_i| \tag{10}$$

   as described in the lectures. Note that you should include all the input components for this, not just those selected in part b). Generate an analogous plot of classification error as a function of the number of iterations. You can vary the coefficient C but we found that $C = 5$ gives approximately the same number of features (10) as in part b).

   If you like, you can use

```
% [w] = abs_reg_update(w,learning_rate,C)

function [w] = abs_reg_update(w,learning_rate,C)

  w_update = learning_rate*C*sign(w);
  index = find( (w-w_update).*sign(w) < 0 );
  w = w - w_update;
  w(index) = 0;
```

   to make the gradient updates with the new regularization penalty. This can be applied after the usual log-likelihood gradient update. The script avoids "overshooting" that often occurs with simple gradient updates.

   Give us the resulting figure

d) Explain the differences in the results of part b) and c). **A brief 2 sentence explanation suffices**.