6.867 Machine learning and neural networks

Problem set 2

Deadline: October 12, 11am in recitation

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to “think”, “consider”, “try” or “experiment” (except when specifically instructed).

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. **Staple each “Problem” separately**, and be sure to write your name on the top of every page.

Problem 1: Active Learning

Reference: Lecture three; Chapter 5-5.4.1

Our goal here is to get a better understanding of the sequential active learning method described in lecture 3.

Let’s start by defining our model and assumptions. We use additive regression models with fixed basis functions $\{\phi_i(\mathbf{x})\}_{i=1,\ldots,m}$ to model outputs as a function of inputs $\mathbf{x}$:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + \ldots + w_m\phi_m(\mathbf{x}) \tag{1}$$

The basis functions could simply return a specific component of the input vector as in $\phi_i(\mathbf{x}) = x_i$ which would reduce the above model to linear regression. Alternatively, the basis functions could measure similarity to “prototypes” as in radial basis functions:

$$\phi_i(\mathbf{x}) = \exp\left\{ \frac{1}{2s^2} \|\mathbf{x} - \mu_i\|^2 \right\} \tag{2}$$

where μ_i specifies a location in the input space and the parameter s gives the spread around μ_i.

The observed outputs y are assumed to be related to the inputs x through some "*true*" weights $\mathbf{w}^*$, and independently corrupted with zero mean Gaussian noise. Thus, for any n input points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we model the observed outputs $\{y_1, \ldots, y_n\}$ as

$$\begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_m(\mathbf{x}_1) \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \phi_1(\mathbf{x}_n) & \cdots & \phi_m(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} w_0^* \\ \cdots \\ w_m^* \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \cdots \\ \epsilon_m \end{bmatrix} \quad (3)$$

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon \quad (4)$$

where $\epsilon \sim N(\mathbf{0}, \sigma^2 I)$. The overall noise variance σ^2 is unknown but insignificant for our purposes here.

It will be helpful to change our notation slightly. We define $\phi(\mathbf{x}) = [1, \phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^T$ as the *feature vector* (column vector) corresponding to an input point $\mathbf{x}$. We can therefore write the matrix $\mathbf{X}$ as

$$\mathbf{X} = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \cdots \\ \phi(\mathbf{x}_n)^T \end{bmatrix} \quad (5)$$

Moreover, we already know from lectures that given $\mathbf{X}$, the noise in the observed outputs cause our parameter estimates $\hat{\mathbf{w}}$ to be Gaussian random variables: $\hat{\mathbf{w}} \sim N(\mathbf{w}^*, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$. For simplicity (and with no real loss of generality), we will assume hereafter that $\sigma^2 = 1$. It will be also helpful to define the inverse covariance matrix $A = (\mathbf{X}^T\mathbf{X})$.

Now, given $\mathbf{x}$, the variance in our predictions $\hat{y}(\mathbf{x}) = f(\mathbf{x}, \hat{\mathbf{w}}) = \phi(\mathbf{x})^T\hat{\mathbf{w}}$ due to the noise in the training outputs is given by

$$\mathrm{Var}(\,\hat{y}(\mathbf{x})\,) = \phi(\mathbf{x})^T(\mathbf{X}^T\mathbf{X})^{-1}\phi(\mathbf{x}) = \phi(\mathbf{x})^T A^{-1}\phi(\mathbf{x}) \quad (6)$$

We are finally ready to turn to our problem...

1. Let A_n be the inverse covariance matrix based on n training examples. Show that $A_{n+1} = A_n + \phi(\mathbf{x}_{n+1})\phi(\mathbf{x}_{n+1})^T$. Is A_n symmetric for all n? Positive definite?

2. Find the *positive* coefficient λ such that $A_{n+1}^{-1} = A_n^{-1} - \lambda A_n^{-1}\phi(\mathbf{x}_{n+1})\phi(\mathbf{x}_{n+1})^T A_n^{-1}$. Express λ as a function of $\mathrm{Var}_n(\hat{y}(\mathbf{x}_{n+1}))$, the expected prediction error (after training on the first n samples) at $\mathbf{x}_{n+1}$. (Hint: In order to check that a matrix B is the inverse of a matrix A, check that $AB = I$).

3. Based on 1) and 2), show that the variance in our predictions at any point $\mathbf{x}$ cannot increase as a result of including *any* new point $\mathbf{x}_{n+1}$ in the training set.

   What we show here is that adding new data, regardless of how we choose the point $\mathbf{x}_{n+1}$, and even if we do not get to choose it, cannot hurt the predictions. This is true in expectation. That is, for any point $\mathbf{x}_{n+1}$, the expected error at another arbitrary point $\mathbf{x}$ is lower than the error at $\mathbf{x}$ before incorperating $(\mathbf{x}_{n+1}, y_{n+1})$ into the prediction. The expectations here are with respect to all the observed outputs.

This result is independent of the type of feature vectors we use. This is good since the feature vectors are typically rather constrained (e.g., $\phi(x) = [1, x, x^2]^T$ traces one dimensional set of points in three dimensions).

4. Show that if we include *any* $\mathbf{x}_{n+1}$ in the training set, then the variance at $\mathbf{x}_{n+1}$ comes down by a multiplicative factor that depends on the current variance at $\mathbf{x}_{n+1}$. Indeed, the larger the variance, the faster it will come down.

   This result guarantees that we can drive the prediction variance to zero at any desired input point so long as we are willing to query the same point multiple times.

Problem 2: Gaussian Mixtures, Logistic, and Softmax models

The following MATLAB functions can be found in the course Athena locker, and on the course web page: `multigaussian.m`, `gaussian2d` and `plotgaussians.m`.

1. Use these functions to investigate different possible Gaussian decision boundaries. For each of the following specifications, find an appropriate pair of Gaussians and prior class probabilities. Turn in the appropriate plot, and the full specification of the Gaussians (means and covariance matrices) and prior class probabilities.

   (a) A linear decision boundary.

   (b) A linear decision boundary, where both means are on the same side of the decision boundary.

   (c) A parabolic decision boundary.

   (d) A non-continues decision boundary (i.e. one of the classes is represented by two disconnected regions).

   (e) A circular decision boundary.

   (f) A skewed (non-circular) ellipsoid decision boundary, with only one of the means inside the ellipsoid.

   (g) No decision boundary– the entire plane is one decision region.

2. Which of the above decision boundaries can represent the decision boundary for a logistic model ?

In many cases, it is necessary to classify into more than two classes. A natural extension of the Gaussian mixture approach is to fit a Gaussian distribution for each class, and classify each input vector to the class with the highest posterior probability for it.

3. We would like to modify the function `plotgaussians()` to plot the decision boundaries between three Gaussian. A partial modification can be found in the MATLAB file `plotgaussians3()`, but it is missing the core instructions for plotting the decision boundaries. Complete this function (turn in the MATLAB code) and use it to plot decision boundaries on several examples. Turn in plots (and associated parameters) for settings where

   (a) All decision boundaries are linear.

   (b) Some decision boundaries are linear, while others are quadratic.

   (c) All decision boundaries are quadratic.

   (d) There are only two decision regions (one class never gets selected).

   ALSO: mark each decision region in the plots with an appropriate label.

Another possible approach is to generalize the the logistic model. Let $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ be the input vector, and suppose we would like to classify to k classes, that is the output y can take a value in $1, \ldots, k$. The *softmax* generalization of the logistic model uses $k(d+1)$ weights $\mathbf{w} = (w_{ij})$, $i = 1, \ldots, k$, $j = 0, \ldots, d$, which define the following k intermediate values:

$$\begin{aligned} z_1 &= w_{10} + \sum_j w_{1j} x_j \\ &\cdots \\ z_i &= w_{i0} + \sum_j w_{ij} x_j \\ &\cdots \\ z_k &= w_{k0} + \sum_j w_{kj} x_j \end{aligned}$$

The classification probabilities under the softmax model are:

$$\Pr(y = i | \mathbf{x}; \mathbf{w}) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

4. Show that when $k = 2$ the softmax model reduces to the logistic model. That is, show how both give rise to the same classification probabilities $\Pr(y|\mathbf{x})$. Do this by constructing an explicit transformation between the weights: for any given set of $2(d+1)$ softmax weights, show an equivalent set of $(d+1)$ logistic weights.

5. Which of the decision regions from question 3 can represent decision boundaries for a softmax model ?

6. Show that the softmax model, for any k, can always be represented by a Gaussian mixture model. What type of Gaussian mixture models are equivalent to a softmax models ?

   The stochastic gradient ascent learning rule for softmax is given by:

$$w_{ij} \leftarrow w_{ij} + \epsilon \sum_t \frac{\partial}{\partial w_{ij}} \log \Pr\left(y^t | \mathbf{x}^t, \mathbf{w}\right)$$

   where $(\mathbf{x}^t, y^t)$ are the training examples. We would like to rewrite this rule as a *delta rule*. In a delta rule the update is specified as a function of the difference between the target and the prediction. In our case, our *target* for each example will actually be a vector $\underline{y}^t = (y_1^t, \ldots, y_k^t)$ where:

$$y_i^t = \begin{cases} 1 & \text{if } y^t = i \\ 0 & \text{if } y^t \neq i \end{cases}$$

   Our prediction will be a corresponding vector of probabilities:

$$\underline{\hat{y}}^t = (\Pr\left(y = 1 | \mathbf{x}^t\right), \ldots, (\Pr\left(y = i | \mathbf{x}^t\right))).$$

7. Calculate the derivative above, and rewrite the update rule as a function of $\underline{y}^t - \underline{\hat{y}}^t$.

Problem 3: Regularization

Reference: Lecture five

We elaborate here a bit on the relation between the "effective" number of parameter choices and regularization discussed in lecture 5. We do this in the context of a simple 1-dim logistic regression model

$$P(y = 1 | x, \mathbf{w}) = g\left(w_0 + w_1 x \right) \tag{7}$$

where $g(z) = (1 + \exp\{-z\})^{-1}$. We assume that $x \in [-1, 1]$.

To understand regularization in this context, we'll try to carve up our parameter space $\mathbf{w} = [w_0, w_1]^T \in \mathcal{R}^2$ into regions such that our loss (or log-probability) is roughly constant within each region. This will help us determine how many "effective" parameter choices we really have. Ideally, the regularization that we impose would directly limit (our estimate of) the number of parameter choices.

A bit more precisely, we want the log-probability $\log P(y|x, \mathbf{w})$ to vary by no more than ϵ within each region in the parameter space. To find such regions, we first examine how the log of the logistic function varies as a function of its input: (this result will be useful to you later on)

$$\frac{\partial}{\partial z}\log g(z) = \frac{1}{g(z)}\frac{\partial}{\partial z}g(z) = \frac{1}{g(z)}\,g(z)(1-g(z)) = 1-g(z) \tag{8}$$

In other words, since $g(z) \in [0,1]$ (probability), the derivative here is also bounded by 1. The function $\log g(z)$ therefore varies at most with slope 1 as a function of z, or

$$|\log g(z) - \log g(z')| \leq |z - z'| \tag{9}$$

for any two points z and z'. To use this result, we define $z = w_0 + w_1 x$ and $z' = w'_0 + w'_1 x$ for any $x \in [-1,1]$. This gives

$$\begin{aligned}
|\log P(y=1|x,\mathbf{w}) - \log P(y=1|x,\mathbf{w}')| &= |\log g(z) - \log g(z')| \leq |z-z'| && (10)\\
&= |(w_0 - w'_0) + (w_1 - w'_1)x| && (11)\\
&\leq |(w_0 - w'_0)| + |(w_1 - w'_1)x| && (12)\\
&\leq |(w_0 - w'_0)| + |(w_1 - w'_1)| && (13)
\end{aligned}$$

since $|x| \leq 1$ by assumption. So, whenever $|w_0 - w'_0| + |w_1 - w'_1| \leq \epsilon$, the corresponding losses or (negative) log-probabilities are also bounded by ϵ. We can therefore carve up the parameter space by finding discrete points $\mathbf{w}^{(i)}$ and regions around them such that

$$|w_0^{(i)} - w'_0| + |w_1^{(i)} - w'_1| \leq \epsilon \tag{14}$$

whenever $\mathbf{w}'$ belongs to the i^{th} region. These regions are shown in Figure 1 for $\epsilon = 0.4$. We have also included in the Figure the area limited by the Euclidean norm of the parameter vector, $\|\mathbf{w}\|^2$. Increasing the the limit $\|\mathbf{w}\|^2$ clearly incorporates more "choices" and it makes sense to use this type of norm in regularization.

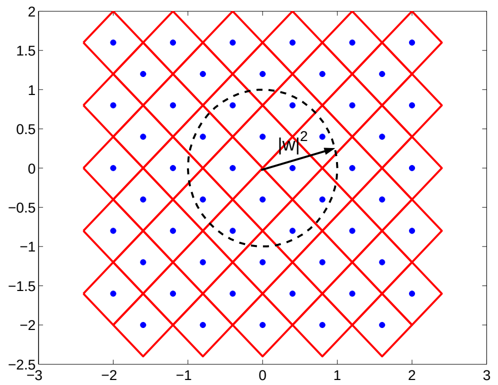

Figure 1: Regions in the parameter space corresponding to roughly constant losses for simple logistic regression model.

Note: Explicitly finding the regions as we have done here is merely a conceptual device in understanding (and analyzing) estimation methods. We never have to identify such regions nor the discrete "effective" parameter choices in practice.

The problem:

The goal here is to understand regularization a bit better in the context of a simple logistic regression model

$$P(y=1|\mathbf{x},\mathbf{w}) = g\left(w_0 + w_1 x_1 + \ldots + w_d x_d\right) \tag{15}$$

We have provided you with a few helpful MATLAB functions for this purpose:

`w = logisticreg(X,y,c) % (logisticreg.m)` performs fast optimization of logistic regression coefficients with squared norm regularization (regularization parameter c).

`[X,y] = sample(n) % (sample.m)` generates training or test examples from a very simple classification problem

`EcvlogP = crossvalid(X,y) % (crossvalid.m)` computes leave-one-out cross-validation estimate of the expected log-probability of labels for logistic regression models.

1. Generate $n = 20$ training and $n = 200$ (or more) test examples using the `sample` function. The resulting examples will be 10 dimensional vectors. We ask you to plot training and test performance for different levels of regularization. More precisely, for each $c = \{1, \ldots, 20\}$, use the `logisticreg` function to get the corresponding parameter estimate $\hat{\mathbf{w}}$. Evaluate and plot the average training and test log-probabilities of labels as a function of the regularization parameter c. Keep the training and test sets fixed while varying c. Return the resulting figure.

2. Explain the *relevant* qualitative properties of the two curves in your figure.

3. Use the `crossvalid` function to generate leave-one-out cross-validation estimates for each $c = \{1, \ldots, 20\}$ and include these results in your figure. How useful is cross-validation in approximating test performance? For setting the regularization parameter c?

4. Explain the changes in the *three* curves when you add 30 examples to the existing training set. (Note: it may be helpful to plot the new results using the same axis. You can store the previous axis with `prevaxis = axis` and recover the settings later on with `axis(prevaxis)`).

Problem 4: Stochastic Gradient Ascent

Reference: Lectures four, five; Chapter 6

Here you will solve a digit classification problem with logistic regression models. We have made available the following training and test sets:

`digit_x.dat, digit_y.dat, digit_x_test.dat, digit_y_test.dat`.

1. Derive the stochastic gradient ascent learning rule for a logistic regression model starting from the regularized likelihood objective

   $$J(\mathbf{w}; c) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2}\|\mathbf{w}\|^2 \tag{16}$$

   where $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$, or by modifying your derivation of the delta rule for the softmax model.

   (Normally we would not include w_0 in the regularization penalty but have done so here for simplicity of the resulting update rule).

2. Write a matlab function `w = SGlogisticreg(X,y,c,epsilon)` that takes inputs simmilar to `logisticreg` from the previous section, and a learning rate parameter ϵ, and uses stochastic gradient ascent to learn the weights. You may include additional parammeters to control when to stop, or hard-code it into the function.

3. Provide a rationale for setting the learning rate and the stopping criterion in the context of the digit classification task. You should assume that the regularization parameter c remains fixed at 1. (You might wish to experiment with different learning rates and stopping criterion but do **NOT** use the test set. Your justification should be based on the available information before seeing the test set.)

4. Set $c = 1$ and apply your procedure for setting the learning rate and the stopping criterion to evaluate the average log-probability of labels in the training and test sets. Compare the results to those obtained with `logisticreg`. For each optimization method, report the average log-probabilities for the labels in the training and test sets as well as the corresponding mean classification errors (estimates of the miss-classification probabilities). (Please include all MATLAB code you used for these calculations)

5. Are the train/test differences between the optimization methods reasonable? Why? (Repeat the gradient ascent procedure a couple of times to ensure that you are indeed looking at a "typical" outcome)

6. The classifiers we found above are both linear classifiers, as are all logistic regression classifiers. In fact, if we set c to a different value, we are still searching the same set

of linear classifiers. Try using `logisticreg` with different values of c, to see that you get different classifications. Why are the resulting classifiers different, even though the same set of classifiers is being searched ? Contrast the reason with the reason for the differences you explained in the previous question.

7. Gaussian mixture models with identical covariance matrices also lead to linear classifiers. Is there a value of c such that training a Gaussian mixture model necessarily leads to the same classification as training a logistic regression model using this value of c ? Why ?