Homework 4

86-595 Neural Data Analysis
Name: Shashank Singh
Email: sss1@andrew.cmu.edu
Due: Thursday, October 18, 2012

Problem 1

---

a. Since each neuron is independent, by properties of logarithms,

$$\ell(s) = \ln\left(\mathsf{P}\left(\mathbf{r}|\boldsymbol{\lambda}(s)\right)\right) = \ln\left(\prod_i \mathsf{P}\left(r_i|\lambda_i(s)\right)\right) = \sum_i \ln\left(\mathsf{P}\left(r_i|\lambda_i(s)\right)\right)$$
$$= \sum_i \ln\left(\frac{\lambda_i^{r_i}(s)}{r_i!}e^{-\lambda_i(s)}\right) = \sum_i r_i \ln\left(\lambda_i(s)\right) - \lambda_i(s) - \ln(r_i!). \quad \blacksquare$$

b. By the result of part a., since $\sum_i \lambda_i(s)$ and $\sum_i \ln(r_i!)$ are constant with respect to s,

$$\hat{s}_{ML} = \operatorname*{argmax}_s \ell(s) = \operatorname*{argmax}_s \sum_i r_i \ln\left(\lambda_i(s)\right).$$

Then, by definition of $\lambda_i(s)$, since $2\sigma^2$ is constant with respect to s,

$$\hat{s}_{ML} = \operatorname*{argmax}_s \sum_i r_i \ln\left(\exp\left(\frac{(s-\mu_i)^2}{2\sigma^2}\right)\right) = \operatorname*{argmax}_s \sum_i r_i \frac{(s-\mu_i)^2}{2}.$$

Assuming this function achieves its maximum at the zero of its derivative gives

$$0 = \sum_i r_i(\hat{s}_{ML} - \mu_i) = \sum_i r_i \hat{s}_{ML} - \sum_i r_i\mu_i, \quad \text{so that} \quad \hat{s}_{ML} = \frac{\sum_i r_i\mu_i}{\sum_i r_i}. \quad \blacksquare$$

c. As shown in part b.,

$$\hat{s}_{MAP} = \operatorname*{argmax}_s \ell(s)p(s) = \operatorname*{argmax}_s \ln(\ell(s)p(s)) = \operatorname*{argmax}_s \ln\ell(s) + \ln p(s)$$
$$= \operatorname*{argmax}_s \sum_i \frac{r_i(s-\mu_i)^2}{2\sigma^2} + \ln\left(\exp\left(\frac{(s-s_{prior})^2}{2\sigma^2_{prior}}\right)\right)$$
$$= \operatorname*{argmax}_s \sum_i \frac{r_i(s-\mu_i)^2}{2\sigma^2} + \frac{(s-s_{prior})^2}{2\sigma^2_{prior}}.$$

Assuming this function achieves its maximum at the zero of its derivative gives

$$0 = \sum_i \frac{r_i(\hat{s}_{MAP} - \mu_i)}{\sigma^2} + \frac{(\hat{s}_{MAP} - s_{prior})}{\sigma^2_{prior}}, \quad \text{so that} \quad \hat{s}_{MAP} = \frac{\sum_i \frac{r_i\mu_i}{\sigma^2} + \frac{s_{prior}}{\sigma^2_{prior}}}{\sum_i \frac{r_i}{\sigma^2} + \frac{1}{\sigma_{prior}}}. \quad \blacksquare$$

Problem 2

a. The following matlab code computes, for each firing rate, the difference between the likelihoods $\mathsf{P}(f|A) - \mathsf{P}(f|B)$. If this is difference is positive, $\ell(A) > \ell(B)$, so MLE predicts the stimulus as A; otherwise, MLE predicts the stimulus as B.

```
>> f = [2 5 8 11 14 17];
>> ma = 8; sa = 5; mb = 12; sb = 6;
>> L = (1/sa)*exp(-(f - ma).^2/(2*sa^2)) - (1/sb)*exp(-(f - mb).^2/(2*sb^2));
>> [f; L > 0]

ans =
     2     5     8    11    14    17
     1     1     1     1     0     0
```

Thus we have

$f[sp/s]$	2	5	8	11	14	17
MLE	A	A	A	A	B	B

.

b. If, instead, we compute the posteriors by myltiplying the likelihoods by the priors, the matlab code gives:

```
>> P = (1/3)*(1/sa)*exp(-(f - ma).^2/(2*sa^2))...
                                    - (2/3)*(1/sb)*exp(-(f - mb).^2/(2*sb^2));
>> [f; P > 0]

ans =
     2     5     8    11    14    17
     1     0     0     0     0     0
```

Thus we have

$f[sp/s]$	2	5	8	11	14	17
MAP	A	B	B	B	B	B

.

Problem 3

A) i) Since Naïve Bayes assumes that the X_i's are conditionally independent given Y,

$$\mathsf{P}(X|Y) = \mathsf{P}(X_1, X_2, \ldots, X_n|Y) = \boxed{\prod_i \mathsf{P}(X_i|Y)\,.}$$

ii) Since we are only interested in computing $\text{argmax}_Y \mathsf{P}(Y|X)$, and $\mathsf{P}(X)$ is constant with respect to Y, we can ignore the $\mathsf{P}(X)$ in the denominator.

iii) By the result of part a.,

$$\mathsf{P}(X = x_i|Y = y)\,\mathsf{P}(Y = y) = \boxed{\mathsf{P}(Y = y)\prod_i \frac{\lambda_{i,y}^{x_i}}{x_i!}e^{-\lambda_{i,y}}.}$$

iv) Using the result of part a. of Problem 1 gives

$$\begin{aligned}\ln(\mathsf{P}(X|Y)\,\mathsf{P}(Y)) &= \ln \mathsf{P}(Y = y) + \ln\left(\prod_i \frac{\lambda_{i,y}^{x_i}}{x_i!}e^{-\lambda_{i,y}}\right)\\ &= \boxed{\ln \mathsf{P}(Y = y) + \sum_i x_i \ln(\lambda_{i,y}) - \lambda_{i,y} - \ln(x_i!).}\end{aligned}$$

B) i) The following is my implementation of `trainPoissonNBDecoder`:

```
function [classMeans, classPriors] = trainPoissonNBDecoder(trainCounts, trainLabels)

  classes = unique(trainLabels); % this way generalizes to multiple classes

  for i=1:length(classes)
    classMeans(:,i) = mean(trainCounts(:,trainLabels ==  classes(i)),2);
    classPriors(i) = sum(trainLabels == i)./length(trainLabels);
  end
end
```

ii) The following is my implementation of `poissonNBDecode`:

```
function estLabels = poissonNBDecode(testCounts, classMeans, classPriors)

  l = zeros(size(testCounts,2),length(classPriors));

  for trial=1:size(testCounts,2)
    gln = gammaln(testCounts + 1);

    for class=1:length(classPriors)
      l(trial, class) = testCounts(:,trial)'*log(classMeans(:,class)); % xln(lambda)
      l(trial, class) = l(trial,class) - sum(classMeans(:,class)); % - lambda
      l(trial, class) = l(trial,class) - sum(gln(:,trial)); % - ln(x!)
    end
    posterior(trial,:) = log(classPriors) + l(trial,:);
  end

  [~, estLabels] = max(posterior',[],1);
end
```

iii) The decoder is $\boxed{80.2\%}$ accurate.

iv) See Figure 1.

v) Decoding accuracy is maximized when the class 1 prior is 35% (so that the class 2 prior is 65%), which is significantly lower than the prior learned from the training data (50% each for class 1 and class 2). This makes sense since looking at `testLabels` reveals that

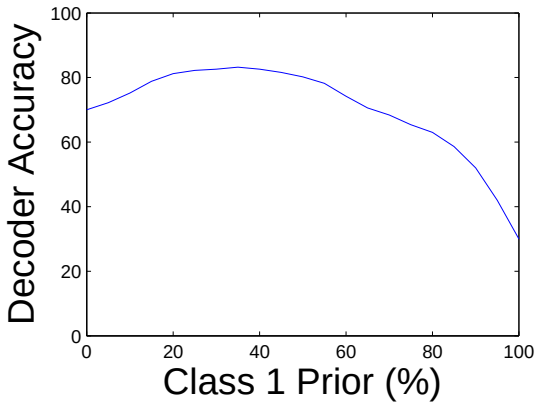

Figure 1: Decoding accuracy for different class 1 priors.

only 150 of the 500 test examples (30%) were from class 1, whereas about 250 of the 500 training examples (50%) were from class 1. In this case, the difference in test accuracies using our learned prior and the optimal prior was only 3%, so it was not very signficant, but, had the learned class 1 prior been much greater, the difference could have been as much 50%, so knowing the true prior may be more important in general.

vi) Since spike data is inevitably noisy, there will always be class 1 trials with spike counts resembling class 2 trials, and vice versa. Furthermore, Naïve Bayes can only fit a linear decision surface, which may not be sufficiently descriptive to classify the data. Approaches to increasing decode accuracy might include reducing noise in both the training and test data (e.g., spike sorting) or using a more powerful classifer (e.g., a nonlinear classifier, or one that does not make Naïve Bayes conditional independence assumptions; relaxing the conditional independence assumptions may be particularly important, since neurons are known to have significant noise correlation).

Problem 4

This homework was pretty short; the theory questions went very quickly, and the only part of the coding that took me a while was remembering that $\Gamma(n) = (n-1)!$, not $n!$, which brought my decoding accuracy down to 64% in part iii).

I'm also taking a Machine Learning class (10-601) alongside, so I've seen and implemented MLE, MAP, and Naïve Bayes before, although I haven't done so with neural data, so deriving the solutions for the Poisson case was nice. But how reasonable is the assumption that the neurons' tuning curves uniformly tile the stimulus space? I do research in visual cortex, where this is almost never the case.

I was a little confused because part iv) of the coding section seemed to have already been implemented in the script; were we supposed to do anything more for that part than attach the plot generated?