Math 189r
Homework 5
November 28, 2016

There are 3 problems in this set. Feel free to work with other students, but make sure you write up the homework and code on your own (no copying homework *or* code; no pair programming). Feel free to ask students or instructors for help debugging code or whatever else, though. When implementing algorithms you may not use any library (such as `sklearn`) that already implements the algorithms but you may use any other library for data cleaning and numeric purposes (`numpy` or `pandas`). Use common sense. Problems are in no specific order.

1 (Laplace Approximation) Reference Section 8.4 of Murphy on Bayesian Logstic Regression. We will use the Laplace Approximation to approximate the posterior distribution over $\mathbf{w}$ when we have a prior of the form $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{V}_0)$. With the energy function $E(\mathbf{w}) = -\log \mathbb{P}(\mathcal{D}|\mathbf{w}) - \log \mathbb{P}(\mathbf{w})$,

(a) Compute the gradient of the energy ∇E, and

(b) Compute the Hessian of the energy $\nabla^2 E$.

(c) Using (a) and (b), what is the Laplace approximate posterior over $\mathbf{w}$? Assume we have the mode of the posterior $\mathbf{w}^\star$ such that $\nabla E(\mathbf{w}^\star) = 0$.

(a) We can see that

$$E = \sum_i -y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) - (1 - y_i) \log \Big(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)\Big) + \frac{1}{2}\mathbf{w}^\top \mathbf{V}_0^{-1} \mathbf{w}. \tag{1}$$

This gives

$$\begin{aligned}\nabla E &= \sum_i \Big[\sigma(\mathbf{w}^\top \mathbf{x}_i) - y_i\Big] \mathbf{x}_i + \mathbf{V}_0^{-1}\mathbf{w} &(2)\\ &= \mathbf{X}^\top (\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y}) + \mathbf{V}_0^{-1}\mathbf{w}. &(3)\end{aligned}$$

(b) Using part (a) we have

$$\begin{aligned}\nabla^2 E &= \sum_i \nabla \sigma(\mathbf{w}^\top \mathbf{x}_i)\mathbf{x}_i^\top + \mathbf{V}_0^{-1} &(4)\\ &= \mathbf{X}^\top \mathrm{diag}[\sigma(\mathbf{X}\mathbf{w})(1 - \sigma(\mathbf{X}\mathbf{w}))]\mathbf{X} + \mathbf{V}_0^{-1}. &(5)\end{aligned}$$

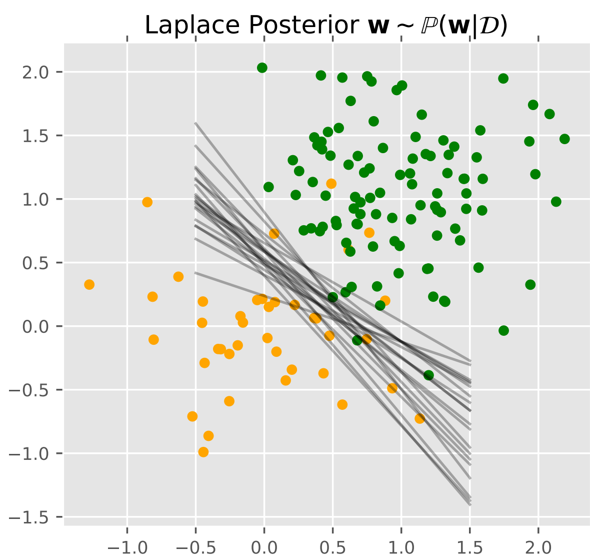
(c) With $\mathbf{H} = \nabla^2 E(\mathbf{w}^\star)$ the Laplace Approximation gives $\mathbb{P}(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}^\star, \mathbf{H}^{-1})$.

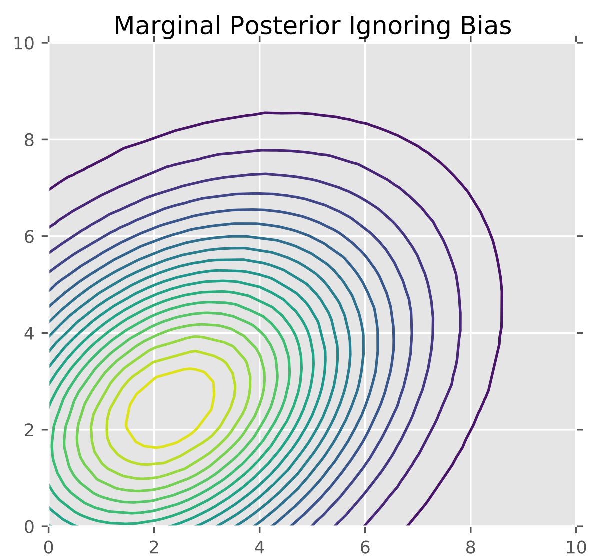
2 (Logistic Regression) Download the data at `https://math189r.github.io/hw/data/classification.csv`. Consider the Laplace Approximated Bayesian Logistic Regression from Problem 1. Calculate the posterior distribution over $\mathbf{w}$.

Code is below. We find the approximate posterior $\mathbf{w} \sim \mathcal{N}(\mathbf{w}^\star, \mathbf{H}^{-1})$ where

$$\mathbf{w}^\star = \begin{bmatrix} -3.052 \\ 4.012 \\ 3.721 \end{bmatrix}, \qquad \mathbf{H}^{-1} = \begin{bmatrix} 0.597 & -0.600 & -0.342 \\ -0.600 & 0.953 & 0.222 \\ -0.342 & 0.221 & 0.578 \end{bmatrix}. \tag{6}$$

We can see the distribution over $\mathbf{w}$ graphically below:





Note that much of the code comes from problem 2 from Problem Set 1.

```
import numpy as np
from scipy import sparse
from scipy import linalg

plt.style.use("ggplot")

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def log_likelihood(X, y_bool, w, reg=1e-6):
    mu = sigmoid(X @ w)
    mu[~y_bool] = 1 - mu[~y_bool]
```

```
    return np.log(mu).sum() - reg*np.inner(w, w)/2

def grad_log_likelihood(X, y, w, reg=1e-6):
    return X.T @ (sigmoid(X @ w) - y) + reg * w

def hess_log_likelihood(X, w, reg=1e-6):
    mu = sigmoid(X @ w)
    return X.T @ sparse.diags(mu * (1 - mu)) @ X + reg * sparse.eye(X.shape[1])

def newton_step(X, y, w, reg=1e-6):
    return linalg.cho_solve(
        linalg.cho_factor(hess_log_likelihood(X, w, reg=reg)),
        grad_log_likelihood(X, y, w, reg=reg),
    )

def lr_newton(X, y, reg=1e-6, tol=1e-6, max_iters=300, verbose=False, print_freq=5):
    y = y.astype(bool)
    w = np.zeros(X.shape[1])
    objective = [log_likelihood(X, y, w, reg=reg)]
    step = newton_step(X, y, w, reg=reg)

    while len(objective)-1 <= max_iters and \
          np.linalg.norm(step) > tol:
        if verbose and (len(objective)-1) % print_freq == 0:
            print("[i={}] likelihood: {}. step norm: {}".format(
                len(objective)-1, objective[-1], np.linalg.norm(step)),
            )

        step = newton_step(X, y, w, reg=reg)
        w = w - step
        objective.append(log_likelihood(X, y, w, reg=reg))

    if verbose:
        print("[i={}] done. step norm = {:0.2f}".format(
            len(objective)-1, np.linalg.norm(step)),
        )
    return w, objective

def sample_posterior(X, w, S=5000, reg=1e-6):
    H = hess_log_likelihood(X, w, reg=reg)
    w = np.random.multivariate_normal(w, np.linalg.inv(H), size=S)
    return H, w

def predict(X, w, S=5000, reg=1e-6):
    H, w = sample_posterior(X, w, S=S, reg=reg)
```

```
    return (X @ w.T).mean(axis=1)

def plot_line(w, x0, xf):
    """ plot_line
    Plots w.T @ x == 0 where w[0] is
    a bias term
    """
    x = np.array([x0, xf])
    plt.plot(x, -(w[1]*x + w[0])/w[2], alpha=0.3, color="black")

def plot_line(w, x0, xf):
    """ plot_line
    Plots w.T @ x == 0 where w[0] is
    a bias term
    """
    x = np.array([x0, xf])
    plt.plot(x, -(w[1]*x + w[0])/w[2], alpha=0.3, color="black")

## CALL WITH DATA ##

X = np.loadtxt(
    "classification.csv",
    delimiter=",",
)
y = X[:,2]
X = np.hstack((np.ones((X.shape[0],1)),X[:,:2]))

reg = 1.
w, obj = lr_newton(X,y, reg=reg)

np.random.seed(0)
H, ws = sample_posterior(X, w, S=20, reg=reg)
for w_ in ws:
    plot_line(w_, -0.5,1.5)

plt.scatter(
    X[:,1], X[:,2],
    color=["green" if y_==1. else "orange" for y_ in y],
)
plt.title(r"Laplace Posterior $\mathbf{w}\sim\mathbb{P}(\mathbf{w}|\mathcal{D})$")
plt.savefig("nov_28/w_posterior.pdf")

print("Laplace Approximate Posterior: w ~ N(w_, H^{-1})")
print("w_ = {}".format(w))
print("H^-1 = {}".format(np.linalg.inv(H)))
```

3 (Monte-Carlo Predictive Posterior) From Problem 2 we have the distribution $\mathbb{P}(\mathbf{w}|\mathcal{D})$. Now suppose we want to compute the probability that a test point $\mathbf{x}$ belongs to class 1. Analytically, we marginalize out $\mathbf{w}$ as

$$\mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathcal{D}) = \int \mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathbf{w})\mathbb{P}(\mathbf{w}|\mathcal{D})\, d\mathbf{w}.$$

Unfortunately, this integral cannot be computed in closed form (we say the integral is intractable). On the other hand, a Simple Monte Carlo approximation of the integral is

$$\mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} \mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathbf{w}^{(s)}). \qquad (\mathbf{w}^{(s)} \sim \mathbb{P}(\mathbf{w}|\mathcal{D}))$$

This is an unbiased estimate of the true predictive probability in the sense that its expectation is $\mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathcal{D})$. This is also easy to compute since we approximated $\mathbb{P}(\mathbf{w}|\mathcal{D})$ as a Gaussian, so we can sample from it easily.

(a) Given a function $f(\mathbf{x})$ where $\mathbf{x} \sim \mathbb{P}(\mathbf{x})$, show that

$$\mathbb{E}_{\mathbb{P}(\{\mathbf{x}^{(s)}\})}[\hat{f}] = \mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^{(s)})\right] = \mathbb{E}[f(\mathbf{x})]. \qquad (\mathbf{x}^{(s)} \sim \mathbb{P}(\mathbf{x}))$$

Put in other terms, show that our Monte Carlo estimator is unbiased.

(b) Show that the variance of the Monte Carlo estimate is proportional to $1/S$. That is, show

$$\mathbb{V}_{\mathbb{P}(\{\mathbf{x}^{(s)}\})}[\hat{f}] = \mathbb{V}[f(\mathbf{x})]/S.$$

Note that this means that standard deviation error bars shrink like $1/\sqrt{S}$.

(c) Plot the posterior predictive distribution $\mathbb{P}(\mathbf{y}=1|\mathbf{x},\mathcal{D})$ overlaying your data using this Monte Carlo approximation. You plot should look similar to Figure 8.6 in Murphy.

(a) We can see by linearity of expectation that

$$\mathbb{E}\left[\frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^{(s)})\right] = \frac{1}{S}\sum_{s=1}^{S}\mathbb{E}\left[f(\mathbf{x}^{(s)})\right] \tag{7}$$

$$= \frac{1}{S}\sum_{s=1}^{S}\mathbb{E}\left[f(\mathbf{x})\right] = \mathbb{E}[f(\mathbf{x})]. \tag{8}$$

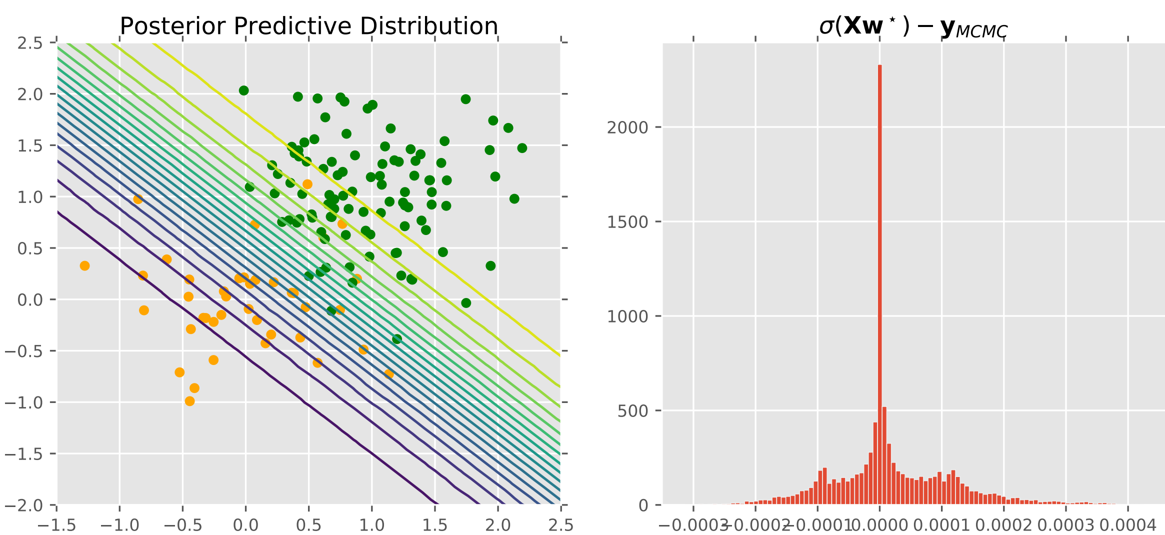
(b) Since each sample $\mathbf{x}^{(s)}$ is independently drawn from $\mathbb{P}(\mathbf{x})$ the variance can be pulled into the sum. It follows that

$$\mathbb{V}_{\mathbb{P}(\{\mathbf{x}^{(s)}\})}[\hat{f}] = \mathbb{V}\left[\frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^{(s)})\right] \tag{9}$$

$$= \frac{1}{S^2}\sum_{s=1}^{S}\mathbb{V}\left[f(\mathbf{x}^{(s)})\right] \tag{10}$$

$$= \frac{1}{S^2}\sum_{s=1}^{S}\mathbb{V}\left[f(\mathbf{x})\right] = \frac{1}{S}\mathbb{V}[f(\mathbf{x})] \tag{11}$$

(c) Code is below. It relies on the code from problem 2.




```
N = 15000
X_ = 7*np.random.rand(N,2) - 3
X_bias = np.hstack((np.ones((X_.shape[0],1)),X_))
y_ = predict(X_bias, w, S=10000, reg=reg)

n_levels=20
levels = np.linspace(y_.min(), y_.max(), n_levels)

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.tricontour(X_[:,0], X_[:,1], y_, levels=levels)
plt.scatter(
    X[:,1], X[:,2],
    color=["green" if y_==1. else "orange" for y_ in y],
)
```

```
plt.xlim(-1.5,2.5)
plt.ylim(-2,2.5)
plt.title("Posterior Predictive Distribution")

plt.subplot(1,2,2)
plt.hist(sigmoid(X_bias @ w) - y_, bins=100)
plt.title(r"$\sigma(\mathbf{X}\mathbf{w}^\star) - \mathbf{y}_{MCMC}$")
plt.savefig("nov_28/posterior_predictive.pdf")
```