DS-GA 1003 HW5

Yiyan Chen

April 2018

1 Introduction

2 From Scores to Conditional Probabilities

2.1

$$\begin{aligned}
E_y[l(y(f(x))|x] &= \sum l(yf(x))p(y|x) \\
&= l(f(x))p(y=1|x) + l(-f(x))p(y=-1|x) \\
&= l(f(x))\pi(x) + l(-f(x))(1-\pi(x))
\end{aligned}$$

2.2

$$\begin{aligned}
E_y[l(y(f(x))|x] &= e^{-f(x)}\pi(x) + e^{f(x)}(\pi(x)) \\
&= e^{-\hat{y}}\pi(x) + e^{\hat{y}}(1-\pi(x)) \\
\frac{\partial E_y}{\partial \hat{y}} &= -e^{-\hat{y}}\pi(x) + e^{\hat{y}}(1-\pi(x)) = 0 \\
e^{\hat{y}}(1-\pi(x)) &= e^{-\hat{y}}\pi(x) \\
e^{2\hat{y}} &= \frac{\pi(x)}{1-\pi(x)} \\
\hat{y}^* &= \frac{1}{2}\ln\frac{\pi(x)}{1-\pi(x)}
\end{aligned}$$

$$
\begin{aligned}
e^{f^*(x)} &= \frac{\pi(x)}{1-\pi(x)} \\
(1-\pi(x))e^{2f^*(x)} &= \pi(x) \\
e^{2f^*(x)} - \pi(x)e^{2f^*(x)} &= \pi(x) \\
(1+e^{2f^*(x)})\pi(x) &= e^{2f^*(x)} \\
\pi(x) &= \frac{e^{2f^*(x)}}{1+e^{2f^*(x)}} \\
&= \frac{1}{e^{-2f^*(x)}+1}
\end{aligned}
$$

2.3

$$
\begin{gathered}
E_y = \ln(1+e^{-f(x)})\pi(x) + \ln(1+e^{f(x)})(1-\pi(x)) \\
\frac{\partial E_y}{\partial \hat{y}} = \frac{-e^{-\hat{y}}}{1+e^{-\hat{y}}}\pi(x) + \frac{e^{\hat{y}}}{1+e^{\hat{y}}}(1-\pi(x)) \\
= \frac{-1}{e^{\hat{y}}+1}\pi(x) + \frac{e^{\hat{y}}}{1+e^{\hat{y}}}(1-\pi(x)) = 0 \\
-\pi(x) + e^{\hat{y}}(1-\pi(x)) = 0 \\
e^{\hat{y}}(1-\pi(x)) = \pi(x) \\
e^{\hat{y}} = \frac{\pi(x)}{1-\pi(x)} \\
\hat{y}^* = \ln\frac{\pi(x)}{1-\pi(x)}
\end{gathered}
$$

$$
\begin{aligned}
e^{f^*(x)} &= \frac{\pi(x)}{1-\pi(x)} \\
(1-\pi(x))e^{f^*(x)} &= \pi(x) \\
(1+e^{f^*(x)})\pi(x) &= e^{f^*(x)} \\
\pi(x) &= \frac{e^{f^*(x)}}{1+e^{f^*(x)}} \\
\pi(x) &= \frac{1}{e^{-f^*(x)}+1}
\end{aligned}
$$

2.4 [Optional]

$[-1, 1]:$

$$\begin{aligned} E_y &= (1 - f(x))\pi(x) + (1 + f(x))(1 - \pi(x)) \\ &= (1 - 2\pi(x))f(x) + 1 \end{aligned}$$

$$\begin{cases} 0 < \pi(x) < \frac{1}{2}, & f^*(x) = 1 \\ \frac{1}{2} < \pi(x) < 1, & f^*(x) = -1 \end{cases} \tag{1}$$

$$\begin{cases} \pi(x) - \frac{1}{2} < 0, & f^*(x) = 1 \\ \pi(x) - \frac{1}{2} > 0, & f^*(x) = -1 \end{cases} \tag{2}$$

$$f(x) \in (1, \infty):$$
$$E_y = (1 + f(X))(1 - \pi(x)) \text{ The minimizer doesn't exist}$$
$$f(x) \in (-\infty, -1):$$
$$E_y = (1 - f(X))\pi(x) \text{ The minimizer doesn't exist}$$

$$f^*(x) = sign(\pi(x) - \frac{1}{2})$$

3 Logistic Regression

3.1 Equivalence of ERM and probabilistic approaches

When $i \in P, y_i = 1, y'_i = 1$
When $i \in N, y_i = -1, y'_i = 0$

$$
\begin{aligned}
-\log \phi(w^T x_i) &= -\log \frac{1}{1+e^{-w^T x_i}} \\
&= \log(1+e^{-w^T x_i} \\
-\log(1-\phi(w^T x_i)) &= \log(1-\frac{1}{1+e^{-w^T x_i}}) \\
&= \log(1-\frac{1}{1+e^{-w^T x_i}})^{-1} \\
&= \log \frac{e^{-w^T x_i}+1}{e^{-w^T x_i}} \\
&= \log(1+e^{w^T x_i})
\end{aligned}
$$

$$
n\hat{R}_n(w) = \sum_{i \in P} \log(1+e^{-w^T x_i}) + \sum_{i \in N} \log(1+e^{w^T x_i})
$$

$$
\begin{aligned}
NLL(w) &= \sum_{i \in P} [-\log \phi(w^T x_i)] + \sum_{i \in N} -\log(1-\phi(w^T x_i)) \\
&= \sum_{i \in P} \log(1+e^{-w^T x_i}) + \sum_{i \in N} \log(1+e^{w^T x_i}) \\
&= n\hat{R}_n(w)
\end{aligned}
$$

3.2 Numerical Overflow and the log-sum-exp trick

3.2.1

$$\begin{aligned}\log(e^{x_1} + \ldots + e^{x_n}) &= \log(e^{x^*}(e^{x_1-x^*} + \ldots + e^{x_n-x^*})) \\ &= x^* + \log(e^{x_1-x^*} + \ldots + e^{x_n-x^*})\end{aligned}$$

3.2.2

$x_i - x^* \in (\infty, 0)$, thus, $exp(x_i - x^*) \in (0, 1)$

3.2.3

Since at least one $x_i - x^*$ is zero when $x_i = x^*$, $e^{x_1-x^*} + \ldots + e^{x_n-x^*}$ will always bigger than 1. So the $\log(e^{x_1-x^*} + \ldots + e^{x_n-x^*})$ will never goes to negative infinity.

3.2.4

part1 = 0
part2 = -s
np.logaddexp(part1,part2)

3.3 Regularized Logistic Regression

3.3.1

According to Log-sum-exp rule, $log(1+exp(-y_i w^T x_i)) = log(exp(0)+exp(-y_i w^T x_i))$ is convex. And the sum of all convex functions is convex as well. Also, norm is convex. $\frac{1}{n}\sum_{i=1}^{n}\log(1+exp(-y_i w^T x_i))$ is convex, and $\lambda||w||^2$ is convex too. So, $J_{logistic(w)}$ is convex.

3.3.2

```
def f_objective(theta, X, y, l2_param=1):
    '''
    Args:
        theta: 1D numpy array of size num_features
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        l2_param: regularization parameter

    Returns:
        objective: scalar value of objective function
    '''
    n = X.shape[0]
    #x_max = np.max(np.dot(-y, np.dot(X, theta)))
    summ = 0
    for i in range(n):
        summ+= np.logaddexp(0,-y[i]*np.dot(theta, X[i]))
    return summ/n + l2_param*np.sum(theta**2)
```

3.3.3

```
def fit_logistic_reg(X, y, objective_function, l2_param=1):
    '''
    Args:
        X: 2D numpy array of size (num_instances, num_features)
        y: 1D numpy array of size num_instances
        objective_function: function returning the value of the objective
        l2_param: regularization parameter

    Returns:
        optimal_theta: 1D numpy array of size num_features
    '''
    theta = np.zeros(X.shape[1])
    optimal_theta = minimize(objective_function, theta, args=(X,y,l2_param)).x
```

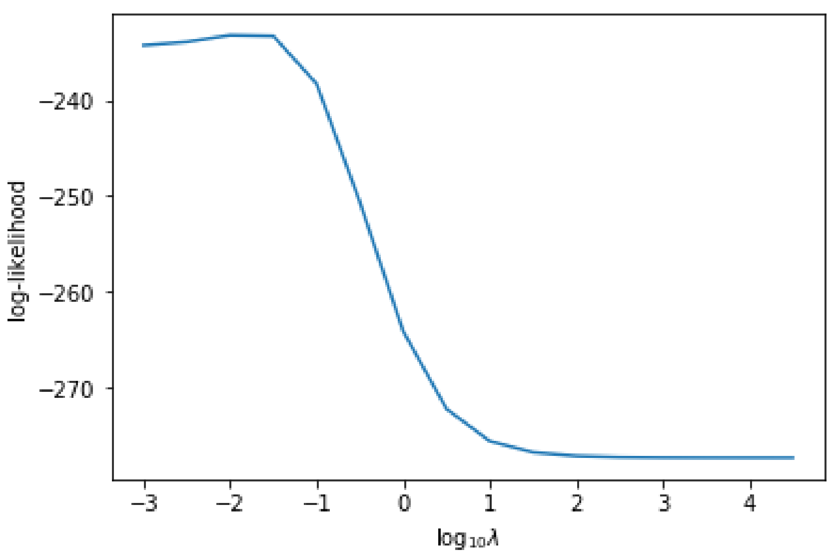
```python
    return optimal_theta

def load_file(file_name):
    f_myfile = open(file_name, 'r')
    d = StringIO(f_myfile.read())
    data = np.loadtxt(d,delimiter=",")
    f_myfile.close()
    return data

X_train = load_file("/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw5-probabilistic/logisti
X_val = load_file("/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw5-probabilistic/logistic-
y_train = load_file("/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw5-probabilistic/logisti
y_val = load_file("/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw5-probabilistic/logistic-
train_bias = np.ones((X_train.shape[0],1))
val_bias = np.ones((X_val.shape[0],1))
X_train_bias = np.hstack((X_train, train_bias))
X_val_bias = np.hstack((X_val, val_bias))
X_train_bias_std = preprocessing.scale(X_train_bias)
X_val_bias_std = preprocessing.scale(X_val_bias)

w_1 = fit_logistic_reg(X_train_bias_std, y_train_j, f_objective)
```

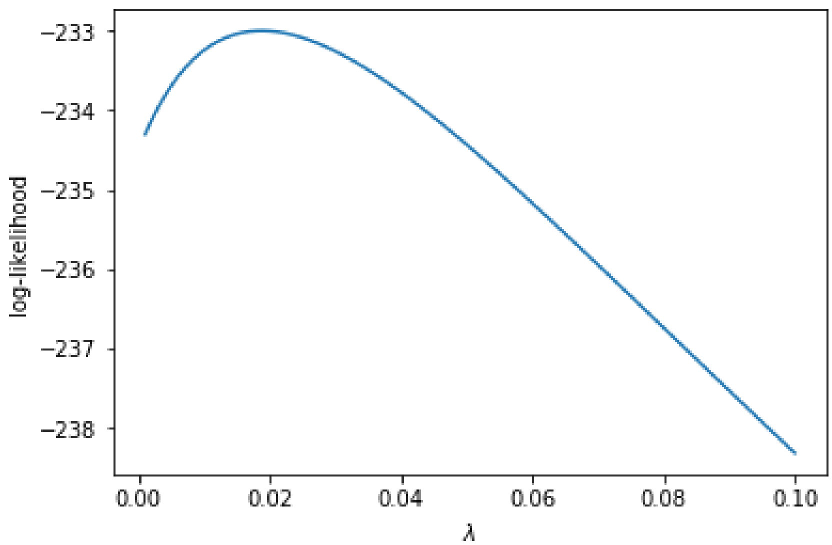
```
array([  9.55682759e-04,  -2.98411854e-04,   3.02812767e-03,
         1.05326700e-01,  -3.58837262e-03,  -1.35879681e-03,
        -3.85259502e-03,  -7.90123362e-04,  -1.14392118e-03,
        -7.17819733e-02,   6.54800235e-03,  -4.51121904e-03,
         1.12491086e-02,  -3.86491439e-03,  -2.71224635e-03,
         1.50343327e-03,  -2.78428667e-03,  -9.19058606e-03,
        -6.82319847e-03,  -1.02758826e-02,  -1.32052354e-08])
```

3.3.4

```python
def get_loglikelihood(X, y, w, l2_param):
    n = X.shape[0]
    obj_val = f_objective(w,X,y, l2_param)
    return -n*(obj_val-l2_param*np.sum(w**2))
l2_val = np.arange(-3, 5, 0.5, dtype=float)
loglike = []
for i in l2_val:
    w = fit_logistic_reg(X_train_bias_std, y_train_j, f_objective, 10**i)
    loglike.append(get_loglikelihood(X_val_bias_std, y_val_j, w, 10**i))

import matplotlib.pyplot as plt
plt.plot(l2_val,loglike)
plt.xlabel('$\log_{10}\lambda$')
```

```
plt.ylabel("log-likelihood")
plt.show()

l2_val2 = np.linspace(0.001, 0.1, 100)
loglike2 = []
for i in l2_val2:
    w = fit_logistic_reg(X_train_bias_std, y_train_j, f_objective, i)
    loglike2.append(get_loglikelihood(X_val_bias_std, y_val_j, w, i))
plt.plot(l2_val2,loglike2)
plt.xlabel('$\lambda$')
plt.ylabel("log-likelihood")
plt.show()

from operator import itemgetter
zipped = zip(l2_val2, loglike2)
best_lambda = max(zipped, key = itemgetter(1))[0]
best_lambda
```



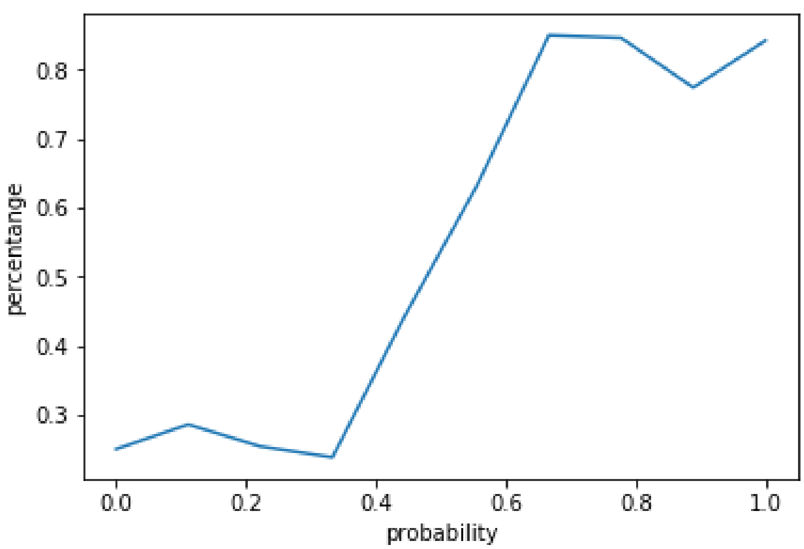


The l2 regularization parameter is 0.019000000000000003.

3.3.5

```
def slice_tuple(min_val, max_val, lst):
    pos = []
    for i in lst:
        if i[1] >min_val and i[1]<=max_val:
            pos.append(i)
    return pos

pos1 = slice_tuple(0,0.1,lst)
pos2 = slice_tuple(0.1,0.2,lst)
pos3 = slice_tuple(0.2,0.3,lst)
pos4 = slice_tuple(0.3,0.4,lst)
pos5 = slice_tuple(0.4,0.5,lst)
pos6 = slice_tuple(0.5,0.6,lst)
pos7 = slice_tuple(0.6,0.7,lst)
pos8 = slice_tuple(0.7,0.8,lst)
pos9 = slice_tuple(0.8,0.9,lst)
pos10 = slice_tuple(0.9,1,lst)

def percent_tuple(lst):
    n = len(lst)
    count = 0
    for i in lst:
        if i[0] == 1:
```

```
            count +=1
    return count/n

per1 = percent_tuple(pos1)
per2 = percent_tuple(pos2)
per3 = percent_tuple(pos3)
per4 = percent_tuple(pos4)
per5 = percent_tuple(pos5)
per6 = percent_tuple(pos6)
per7 = percent_tuple(pos7)
per8 = percent_tuple(pos8)
per9 = percent_tuple(pos9)
per10 = percent_tuple(pos10)
percent = [per1, per2, per3, per4, per5, per6, per7, per8, per9, per10]
plt.plot(np.linspace(0,1,10), percent)
plt.xlabel("probability")
plt.ylabel("percentange")
plt.show()
```



Summarize the result: When there are 10 bins, the percentage rate with respect to probability is very close to y=x.

4 Bayesian Logistic Regression with Gaussian Priors

4.1

$$
\begin{aligned}
P(w|D') &= \frac{P(D'|w)P(w)}{P(D')} \\
&= k \cdot P(D'|w)P(w) \\
&= k \cdot exp(-NLL_{D'}(w)) \cdot P(w)
\end{aligned}
$$

4.2

$$
\begin{aligned}
-\log(p(w|D')) &= -\log k + NLL_{D'}(w) - \log P(w) \\
&= NLL_{D'}(w) - \log P(w) + c \\
&= -\log((2\pi^k|\Sigma|)^{\frac{1}{2}} exp(-\frac{1}{2}w^T\Sigma^{-1}w)) + n\hat{R}_n(w) + c \\
&= -\frac{1}{2}\log(2\pi^k|\Sigma|) + \frac{1}{2}w^T\Sigma^{-1}w + n\hat{R}_n(w) + c \\
&= \frac{1}{2}w^T\Sigma^{-1}w + n\hat{R}_n(w) + c'
\end{aligned}
$$

In order to make MAP estimator is same as the regularized logistic regression:

$$
min(\frac{1}{2}w^T\Sigma^{-1}w + n\hat{R}_n(w)) \sim min(n\hat{R}_n(w) + \lambda n||w||^2)
$$

Then the covariance matrix is I

4.3

$$
\begin{aligned}
\frac{1}{2}w^Tw + n\hat{R}_n(w) &= n\hat{R}_n(w) + \lambda n||w||^2 \\
\frac{1}{2}||w||^2 + n\hat{R}_n(w) &= n\hat{R}_n(w) + \lambda n||w||^2 \\
\frac{1}{2} &= \lambda n \\
\lambda &= \frac{1}{2n}
\end{aligned}
$$

5 Bayesian Linear Regression - Implementation

5.1

```python
def likelihood_func(w, X, y_train, likelihood_var):
    '''
    Implement likelihood_func. This function returns the data likelihood
    given f(y_train | X; w) ~ Normal(Xw, likelihood_var).

    Args:
        w: Weights
        X: Training design matrix with first col all ones (np.matrix)
        y_train: Training response vector (np.matrix)
        likelihood_var: likelihood variance

    Returns:
        likelihood: Data likelihood (float)
    '''

    #TO DO
    likelihood = 1
    k = X.shape[0]
    for i in range(k):
        likelihood *= (2*math.pi*likelihood_var)**0.5*np.exp(-0.5*(y_train[i]-
        np.dot(X[i], w))**2/likelihood_var)

    return likelihood
```

5.2

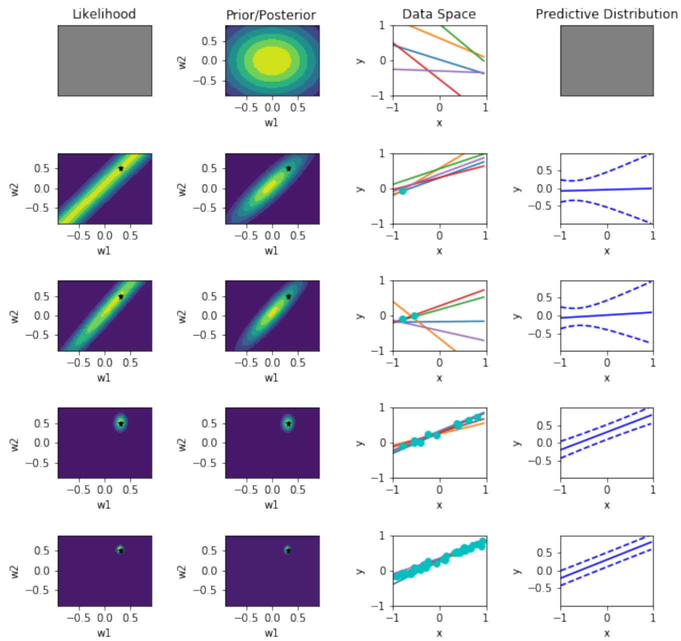
```python
def get_posterior_params(X, y_train, prior, likelihood_var = 0.2**2):
    '''
    Implement get_posterior_params. This function returns the posterior
    mean vector \mu_p and posterior covariance matrix \Sigma_p for
    Bayesian regression (normal likelihood and prior).

    Note support_code.make_plots takes this completed function as an argument.

    Args:
        X: Training design matrix with first col all ones (np.matrix)
        y_train: Training response vector (np.matrix)
        prior: Prior parameters; dict with 'mean' (prior mean np.matrix)
               and 'var' (prior covariance np.matrix)
        likelihood_var: likelihood variance- default (0.2**2) per the lecture slides
```

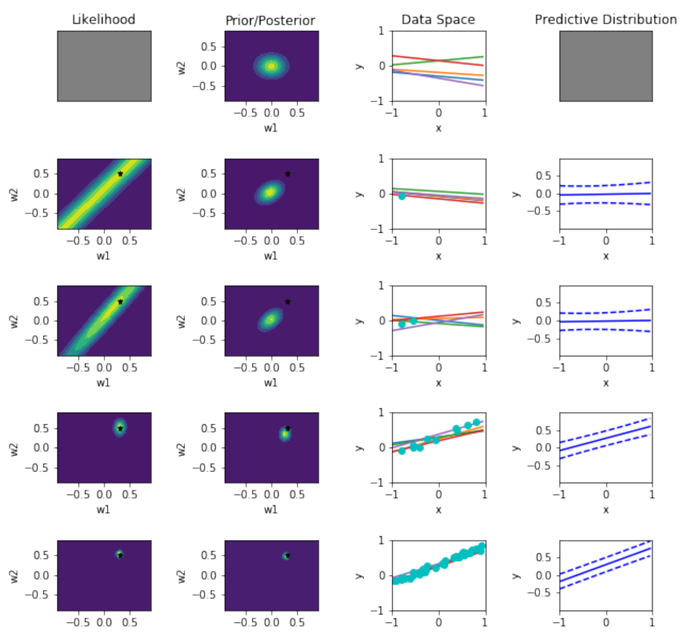
```

    Returns:
        post_mean: Posterior mean (np.matrix)
        post_var: Posterior mean (np.matrix)
    '''

    # TO DO
    prior_var = prior['var']
    post_mean = np.dot(np.dot((np.dot(X.T,X)+ likelihood_var*
        prior_var.getI()).getI(), X.T), y_train)
    post_var = (1./likelihood_var*np.dot(X.T,X)+prior_var.getI()).getI()
    return post_mean, post_var
```

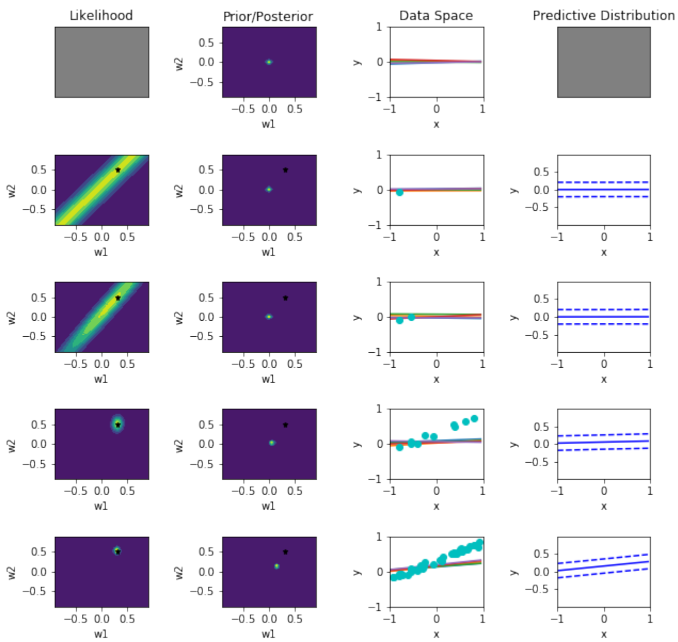
5.3

```
def get_predictive_params(X_new, post_mean, post_var, likelihood_var = 0.2**2):
    '''
    Implement get_predictive_params. This function returns the predictive
    distribution parameters (mean and variance) given the posterior mean
    and covariance matrix (returned from get_posterior_params) and the
    likelihood variance (default value from lecture).

    Args:
        X_new: New observation (np.matrix object)
        post_mean, post_var: Returned from get_posterior_params
        likelihood_var: likelihood variance (0.2**2) per the lecture slides

    Returns:
        - pred_mean: Mean of predictive distribution
        - pred_var: Variance of predictive distribution
    '''

    # TO DO
    pred_mean = np.dot(post_mean.T, X_new)
    pred_var = np.dot(X_new.T, np.dot(post_var, X_new))+likelihood_var
    return pred_mean, pred_var
```

5.4


Likelihood
Prior/Posterior
Data Space
Predictive Distribution
w1
w2
x
y


Likelihood
Prior/Posterior
Data Space
Predictive Distribution
w1
w2
x
y



5.5

(i): the likelihood function shrinks as the sample size increases; when the strength of prior decreases, the likelihood function doesn't seem to get affected much (ii): the posterior distribution shrinks as either the case that sample size increases or the strength of prior decreases. (iii): As the sample size increases and the strength of prior decreases, the confidence interval get closer to the predicted value.

5.6

```
lamb = 0.2**2/(1/2)
from sklearn.linear_model import Ridge
ridge = Ridge(alpha = lamb)
ridge.fit(xtrain[:,1], ytrain)
```

first prior covariance is $\Sigma = \frac{1}{2}I$, $\lambda = \frac{\sigma^2}{\Sigma} = 0.08$.

```
ridge.coef_
```

```
array([[ 0.52399695]])
```

```
ridge.intercept_
```

```
array([ 0.30112479])
```

6 [Optional] Coin Flipping: Maximum Likelihood

6.1

$$p(D|\theta) = \theta^2(1-\theta)$$

6.2

There are 3 different sequences of 3 coin tosses have 2 heads and 1 tail. And the probability of 2 heads and 1 tail is $C_3^2\theta^2(1-\theta)$.

6.3

$$P(D|\theta) = \theta^{n_h}(1-\theta)^{n_t}$$

6.4

$$\begin{aligned}
L(\theta) &= n_h \log\theta + n_t \log(1-\theta) \\
max(L(\theta)) &= \frac{\partial \log(p(D|\theta))}{\partial\theta} \\
n_h\frac{1}{\theta} - n_t\frac{1}{1-\theta} &= 0 \\
n_h - n_h\theta &= n_t\theta \\
(n_t + n_h)\theta &= n_h \\
\theta &= \frac{n_h}{n_t + n_h}
\end{aligned}$$