Machine Learning HW5

Ben Zhang, bz957

April 9, 2018

2 From Scores to Conditional Probabilities

2.1

$$E_y[\ell(yf(x))|x] = p(\ell(f(x))|x)\cdot \ell(f(x)|x) + (1-p(\ell(f(x))|x))\cdot \ell(-f(x)|x)$$

$$= p(y=1|x)\cdot \ell(f(x)) + (1-p(y=1|x))\cdot \ell(-f(x))$$

$$= \pi(x)\cdot \ell(f(x)) - \pi(x)\cdot \ell(-f(x)) + \ell(-f(x))$$

2.2 ∵ $f^*(x) \in \arg\min_f R(f)$, $R(f) = E(\ell(f(x), y))$

∴ Bayes prediction function $f^*(x)$ should be a minimizer of $E_y[\ell(yf(x))|x]$

let $p = \pi(x)$, $\hat{y} = f(x)$, and $\ell(y, f(x)) = e^{-yf(x)}$

$$\Rightarrow E_y[\ell(yf(x))|x] = p\cdot \ell(\hat{y}) - p\cdot \ell(-\hat{y}) + \ell(-\hat{y})$$

$$= p\cdot e^{-\hat{y}} - p\cdot e^{\hat{y}} + e^{\hat{y}}$$

differentiate $\hat{y}$ $\Rightarrow$ $E_y'[\ell(y\hat{y})|x] = -p\cdot e^{-\hat{y}} - pe^{\hat{y}} + e^{\hat{y}} = 0$

$$\Rightarrow \frac{p}{1-p} e^{-\hat{y}} = e^{\hat{y}}$$

$$\ln\left(\frac{p}{1-p}\right) + \ln e^{-\hat{y}} = \ln e^{\hat{y}}$$

$$\hat{y} = \frac{1}{2}\ln\left(\frac{p}{1-p}\right)$$

$$\Rightarrow f^*(x) = \frac{1}{2}\ln\left(\frac{\pi(x)}{1-\pi(x)}\right)$$

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

with condition: $\pi(x) \in (0, 1)$

2.3

$$\because \ell(y,f(x)) = \ln(1+e^{-yf(x)}) \Rightarrow E_y[\ell(yf(x))|x] = p\cdot\ell(\hat{y}) - p\cdot\ell(-\hat{y}) + \ell(-\hat{y})$$

$$= p\cdot\ln(1+e^{-\hat{y}}) - p\ln(1+e^{\hat{y}}) + \ln(1+e^{\hat{y}})$$

differentiate $\hat{y}$, $E_y[\ell(yf(x))|x]' = -p\cdot\frac{e^{-\hat{y}}}{1+e^{-\hat{y}}} - p\cdot\frac{e^{\hat{y}}}{1+e^{\hat{y}}} + \frac{e^{\hat{y}}}{1+e^{\hat{y}}} = 0$

$$\Rightarrow \frac{e^{\hat{y}}}{1+e^{\hat{y}}}(1-p) = p\cdot\frac{e^{-\hat{y}}}{1+e^{-\hat{y}}}$$

$$\Rightarrow \frac{1+e^{\hat{y}}}{1+e^{-\hat{y}}} = \frac{p}{1-p}$$

$$\frac{e^{\hat{y}}(e^{-\hat{y}}+1)}{1+e^{-\hat{y}}} = \frac{p}{1-p}$$

$$e^{\hat{y}} = \frac{p}{1-p}$$

$$\hat{y} = \ln\left(\frac{p}{1-p}\right)$$

$$\Rightarrow f^*(x) = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right)$$

$$\Rightarrow \pi(x) = \frac{1}{1+e^{-f^*(x)}}$$

with condition: $\pi(x) \in (0,1)$

2.4. $E_y[\ell(yf(x))|x] = p\cdot\ell(\hat{y}) - p\cdot\ell(-\hat{y}) + \ell(-\hat{y})$

$$= p\cdot\max(0,1-\hat{y}) - p\cdot\max(0,1+\hat{y}) + \max(0,1+\hat{y})$$

3 Logistic Regression

3.1 Equivalence of ERM and probabilistic approaches

3.1

$y_i'=1$, $NLL(w)_{y_i} = -\log\phi(w^T x_i) = -\log\frac{1}{1+e^{-w^T x_i}} = \log(1+e^{-w^T x_i})$ ①

$y_i'=0$, $NLL(w)_{y_i'} = -\log(1-\phi(w^T x_i)) = -\log\left(\frac{e^{-w^T x_i}}{1+e^{-w^T x_i}}\right) = \log(1+e^{w^T x_i})$ ②

$y_i=1$, $n\hat{R}_n(w)_{y_i} = \log(1+e^{-w^T x_i})$ ③

$y_i=-1$, $n\hat{R}_n(w)_{y_i} = \log(1+e^{w^T x_i})$ ④

$\Rightarrow$ when $y_i'=1$, $y_i=1$, ① = ③
whe $y_i'=0$, $y_i=-1$, ② = ④ and the pairs of y_i', y_i are samer

$\Rightarrow$ $n\hat{R}_n(w) = NNL(w)$ for all $w \in \mathbb{R}^d$

3.2 Numerical Overflow and the log-sum-exp trick

3.2.1

$\log\mathrm{SumExp}(x_1, \cdots x_n) = x^* + \log(e^{x_1-x^*} + \cdots + e^{x_n-x^*}) = \log(e^{x_1-x^*} + \cdots + e^{x_n-x^*}) \cdot e^{x^*}$

$= \log(e^{x_1} + \cdots + e^{x_n})$

$\because$ for all $x_i - x^*$ is no more than 709, will not overflow

$\therefore \log\mathrm{SumExp}(x_1, \cdots x_n)$ is valid

3.2.2 $\because$ for any i, $x_i - x^* \leq 0$, $\therefore \exp(x_i - x^*) \in (0, 1)$

thus the exp calculations will not overflow

3.2.3 when $x_i = x^*$, $e^{x_i - x^*} = 1$, $\therefore e^{x_1-x^*} + \cdots e^{x_n - x^*} \geq 1$

$\therefore \log(e^{x_1-x^*} + \cdots e^{x_n-x^*})$ will never be "-inf"

3.2.4 $\mathrm{logaddexp}(0, -s) = \log(e^0 + e^{-s})$

$= \log(1 + e^{-s})$

3.3 Regularized Logistic Regression

3.3.1

$\|w\|^2$ is convex, because every norm on $\mathbb{R}^n$ is convex.

for every i, $\log(1+\exp(-y_i w^T x_i)) = \log(e^0 + \exp(-y_i w^T x_i))$

is convex because Log-Sum-Exp is convex on $\mathbb{R}^n$

$\Rightarrow J = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i)) + \lambda \|w\|^2$ is convex

because nonnegative weighted sums of convex functions is convex.

3.3.2

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from sklearn.preprocessing import scale
import math
import matplotlib.pyplot as plt

def f_objective(theta, X, y, l2_param=1):
    n, num_ftrs = X.shape
    predictions = np.dot(X,theta)
    y = y.reshape(-1)
    margins = y*predictions
    l2_norm_squared = np.sum(theta**2)
    logloss = sum([np.logaddexp(0,-margins[i]) for i in range(n)]) #n, not num_ftrs
    objective = logloss/n + l2_param * l2_norm_squared
    return objective
```

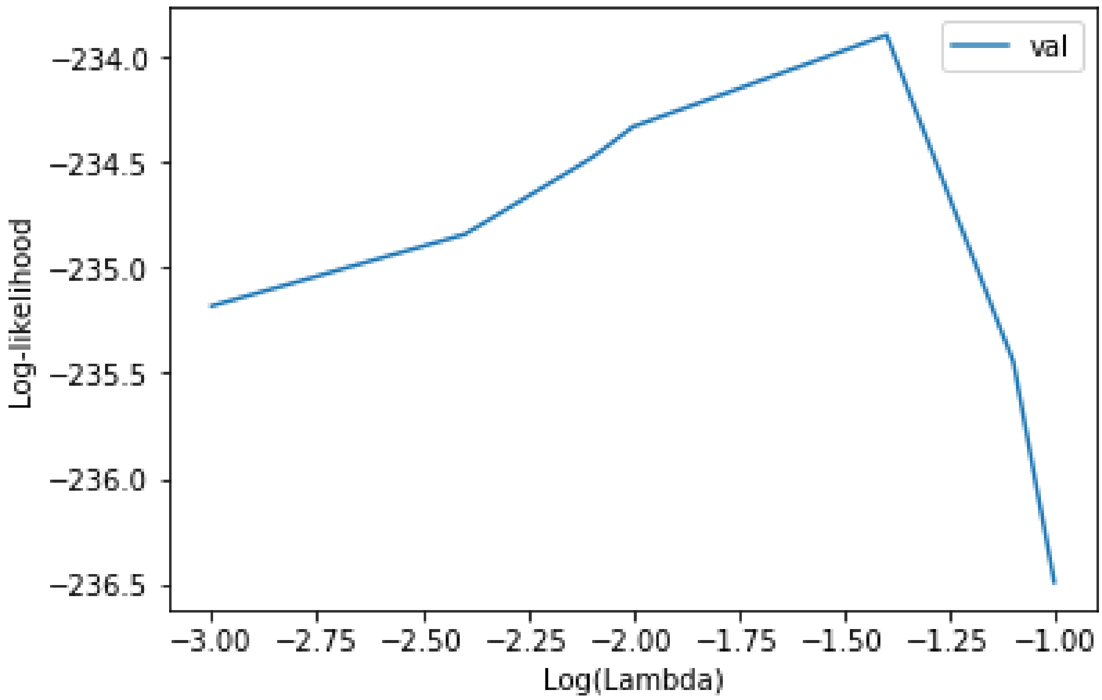
3.3.3

```
def fit_logistic_reg(X, y, objective_function, l2_param=1):
    n, num_ftrs = X.shape
    w_0 = np.zeros(num_ftrs)
    optimal_theta = minimize(objective_function, w_0, args=(X, y, l2_param)).x
    return optimal_theta

Xtrain = pd.read_csv('X_train.txt', sep=",", header=None)
ytrain = pd.read_csv('y_train.txt', sep=",", header=None)
Xtrain_s = scale(Xtrain, axis = 1)
bias = np.ones((Xtrain_s.shape[0],1))
Xtrain_s = np.concatenate((Xtrain_s, bias), axis =1)
ytrain_s = ytrain.replace(to_replace= 0 , value= -1)
ytrain_s = ytrain_s.values

fit_logistic_reg(Xtrain_s, ytrain_s, f_objective, l2_param=1)
```

3.3.4

```
def loglikelihood(X, y, theta):
    n, num_ftrs = X.shape
    predictions = np.dot(X,theta)
    y = y.reshape(-1)
    margins = y*predictions
    logloss = -sum([np.logaddexp(0,-margins[i]) for i in range(n)])
    return logloss

Xval = pd.read_csv('X_val.txt', sep=",", header=None)
yval = pd.read_csv('y_val.txt', sep=",", header=None)
Xval_s = scale(Xval, axis = 1)
bias = np.ones((Xval.shape[0],1))
Xval_s = np.concatenate((Xval_s, bias), axis =1)
yval_s = yval.replace(to_replace= 0 , value= -1)
yval_s = yval_s.values

list_lambda=[0.001,0.004,0.008,0.01,0.04,0.08,0.1]
llist_val = []
losslist_val = []
for l in list_lambda:
    optimal_theta = fit_logistic_reg(Xtrain_s, ytrain_s, f_objective, l)
```

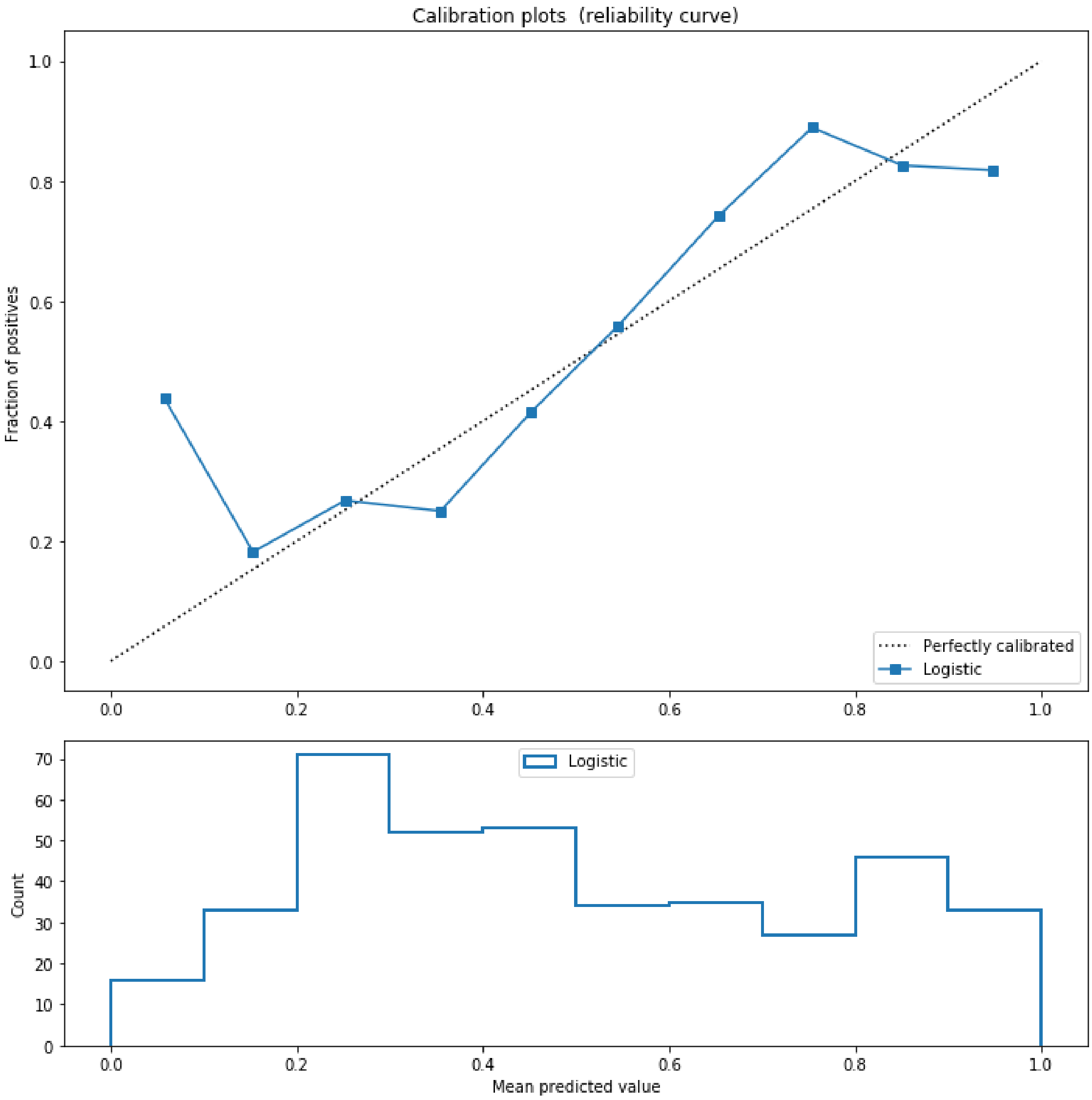
```
    val_loss = loglikelihood(Xval_s, yval_s, optimal_theta)
    llist_val.append(math.log10(l))
    losslist_val.append(val_loss)
    print(val_loss)
# plt.plot(llist_train,losslist_train,label="train")
plt.plot(llist_val,losslist_val,label="val")
plt.legend()
plt.xlabel('Log(Lambda)')
plt.ylabel('Log-likelihood')
plt.show()
```



3.3.5

```
from sklearn import datasets
from sklearn.calibration import calibration_curve


plt.figure(figsize=(10, 10))
ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
ax2 = plt.subplot2grid((3, 1), (2, 0))

ax1.plot([0, 1], [0, 1], "k:", label="Perfectly calibrated")

optimal_theta = fit_logistic_reg(Xtrain_s, ytrain_s, f_objective, 0.04)
n, num_ftrs = Xval_s.shape
prob_pos = np.dot(Xval_s,optimal_theta)
for i in range(n):
    prob_pos[i] = 1/(1+np.exp(-prob_pos[i]))

prob_pos = (prob_pos - prob_pos.min()) / (prob_pos.max() - prob_pos.min())

fraction_of_positives, mean_predicted_value = calibration_curve(yval_s,
                                               prob_pos, n_bins=10

ax1.plot(mean_predicted_value, fraction_of_positives, "s-",
        label="%s" % ('Logistic'))
```

```
ax2.hist(prob_pos, range=(0, 1), bins=10, label='Logistic',
         histtype="step", lw=2)

ax1.set_ylabel("Fraction of positives")
ax1.set_ylim([-0.05, 1.05])
ax1.legend(loc="lower right")
ax1.set_title('Calibration plots  (reliability curve)')

ax2.set_xlabel("Mean predicted value")
ax2.set_ylabel("Count")
ax2.legend(loc="upper center", ncol=2)

plt.tight_layout()
plt.show()
```



The x axis represents the mean predicted probability in each bin, while the y axis represents the true probability in each bin (fraction of positives). For instance, a well calibrated (binary) classifier should classify the samples such that among the samples to which it gave a $predict_proba$ value close to 0.8, approximately 80% actually belong to the positive class.(source:$http://scikit-learn.org/stable/modules/calibration.html$).

For my logistic model, the performance of bins from $0.1-0.9$ performs good, especially $0.1-0.3$, while $0-0.1$ and $0.9-1$ perform not well. The reason may be related to the data sample data amount. Because the best performance bins have higher sample data counts.

4 Bayesian Logistic Regression with Gaussian Priors

4.1

$$p(w|D') = \frac{p(D'|w)\cdot p(w)}{p(D')} \propto p(D'|w)\cdot p(w)$$

$$p(D'|w) \propto \exp(-NLL(w))$$

$$p(w|D') \propto \exp(-NLL(w))\cdot p(w)$$

4.2. $J(w) = \hat{R}_n(w) + \lambda \|w\|^2$

$$p(w|D') \propto p(D'|w)\cdot p(w) = \exp(-NLL(w))\cdot p(w).$$

$$g(w) = \log(p(w|D')) \propto -NLL(w) + \log(p(w))$$

MAP of $g(w)$ is same as the minimizer of $-g(w)$, also the same of the minimizer of $J(w)$.

$$-g(w) = NLL(w) - \log(p(w)),$$

$$nJ(w) = n\cdot\hat{R}_n(w) + n\lambda\|w\|^2$$

$\because$ $n\hat{R}_n(w) = NLL(w)$,

we need to let $-\log(p(w)) + c = n\lambda\|w\|^2$,

where c is an arbitrary constant.

$\because$ $w \sim N(0, \Sigma)$, $p(w) = \frac{1}{\sqrt{2\pi\cdot\Sigma}} e^{-\frac{w^2}{2\Sigma}}$

$$\Rightarrow \frac{1}{2}\cdot\log 2\pi + \frac{1}{2}\log(\Sigma) + \frac{1}{2}w^T\Sigma^{-1}w + c = n\lambda\|w\|^2$$

$\frac{1}{2}\log 2\pi$, $\frac{1}{2}\log(\Sigma)$ is irrelavant with w, $\frac{1}{2}\log 2\pi + \frac{1}{2}\log(\Sigma) + c = 0$

$$\Rightarrow \frac{1}{2}w^T\Sigma^{-1}w = n\lambda\|w\|^2$$

$\Sigma = \frac{1}{2\lambda n} I$ where I is identity matrix.

4.3. $w \sim N(0, I)$.

$J(w) = \hat{R}(w) + \lambda \|w\|^2$

from 4.2 we have $\frac{1}{2\lambda n} I = I$

$\Rightarrow \lambda = \frac{1}{2n}$

5 Bayesian Linear Regression - Implementation

5.1

```
from __future__ import division

import matplotlib.pyplot as plt
import numpy.matlib as matlib
from scipy.stats import multivariate_normal
import numpy as np
import support_code
from numpy.linalg import inv

def likelihood_func(w, X, y_train, likelihood_var):
    n, num_ftrs = X.shape
    likelihood = 1
    for i in range(n):
        coefficient = 1/np.sqrt(likelihood_var * 2 * np.pi)
        r = coefficient * np.exp(-1*((y_train[i]-
              np.dot(X[i,:],w))**2/(2*likelihood_var)))
        print((y_train[i]-np.dot(X[i,:],w))
        likelihood = likelihood*r
    return likelihood
```

5.2

```
def get_posterior_params(X, y_train, prior, likelihood_var = 0.2**2):
    y_train = y_train.reshape(-1)
    post_mean = inv(np.dot(X.T,X) +
                likelihood_var*inv(prior['var'])).dot(X.T).dot(y_train.T)
    post_var = inv(np.dot(X.T,X)/likelihood_var + inv(prior['var']))

    return post_mean, post_var
```

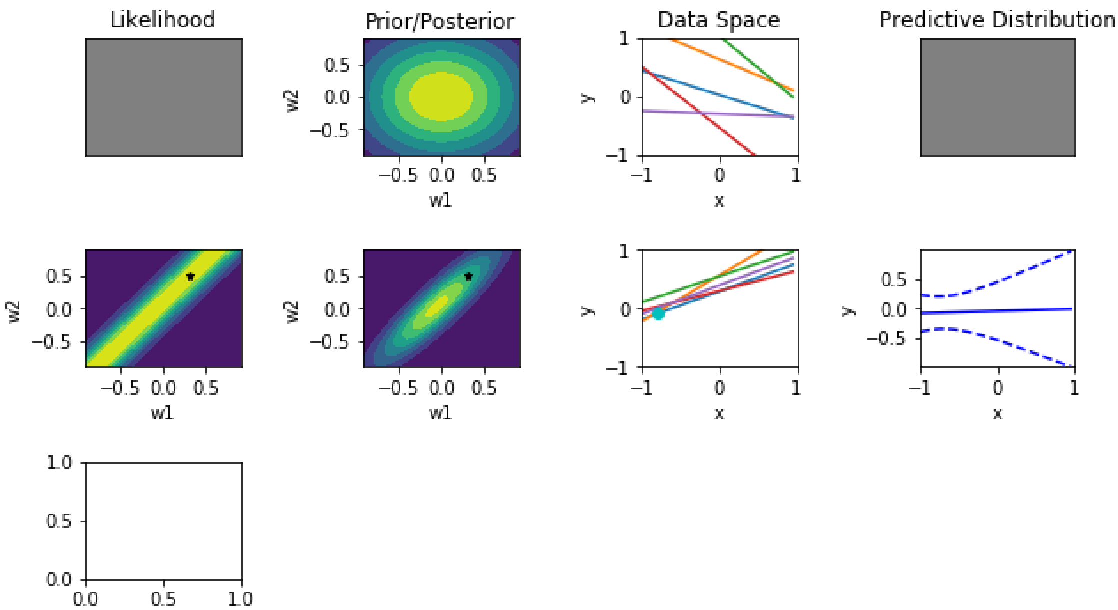
5.3

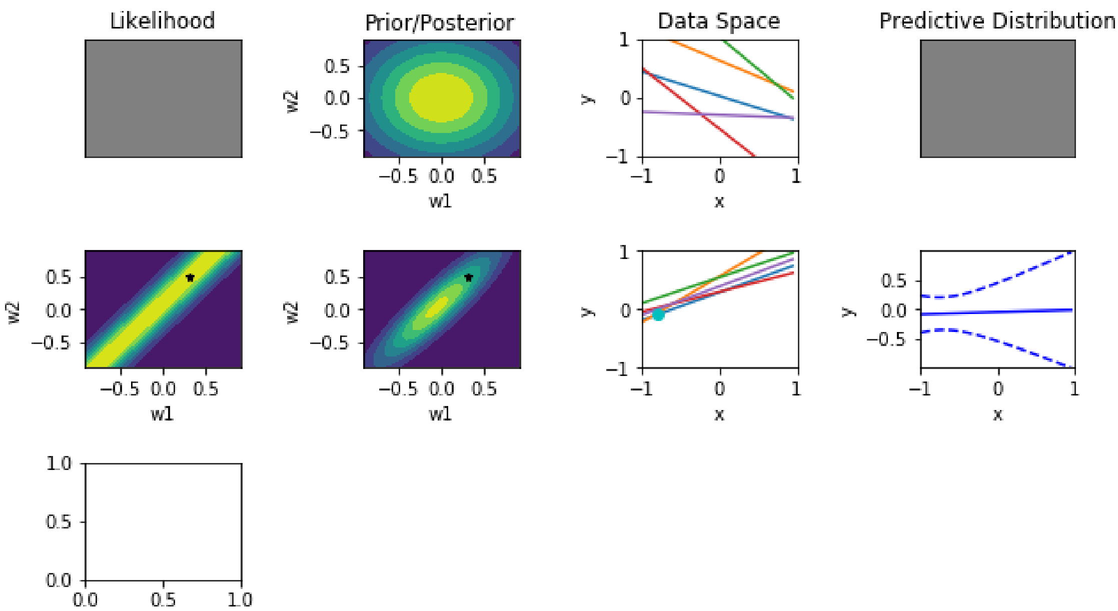
```
def get_predictive_params(X_new, post_mean, post_var, likelihood_var = 0.2**2):
    pred_mean = np.dot(post_mean.T,X_new)
    pred_var = np.dot(X_new.T, post_var).dot(X_new) + likelihood_var

    return pred_mean, pred_var
```

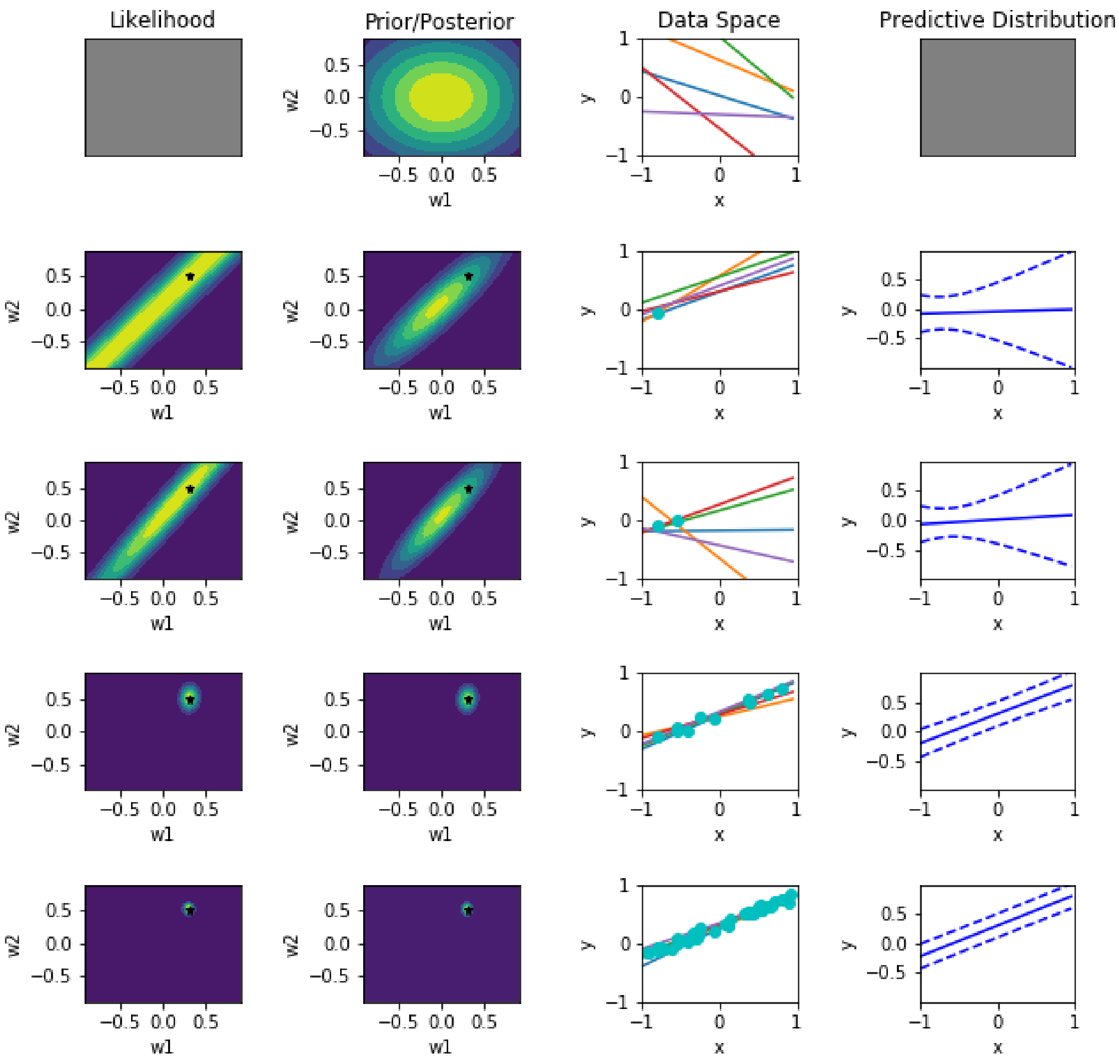
5.4

```
if __name__ == '__main__':

    np.random.seed(46134)
    actual_weights = np.matrix([[0.3], [0.5]])
    data_size = 40
    noise = {"mean":0, "var":0.2 ** 2}
    likelihood_var = noise["var"]
    xtrain, ytrain = support_code.generate_data(data_size, noise, actual_weights)

    #Question (b)
    sigmas_to_test = [1/2, 1/(2**5), 1/(2**10)]
    for sigma_squared in sigmas_to_test:
        prior = {"mean":np.matrix([[0], [0]]),
                 "var":matlib.eye(2) * sigma_squared}
```

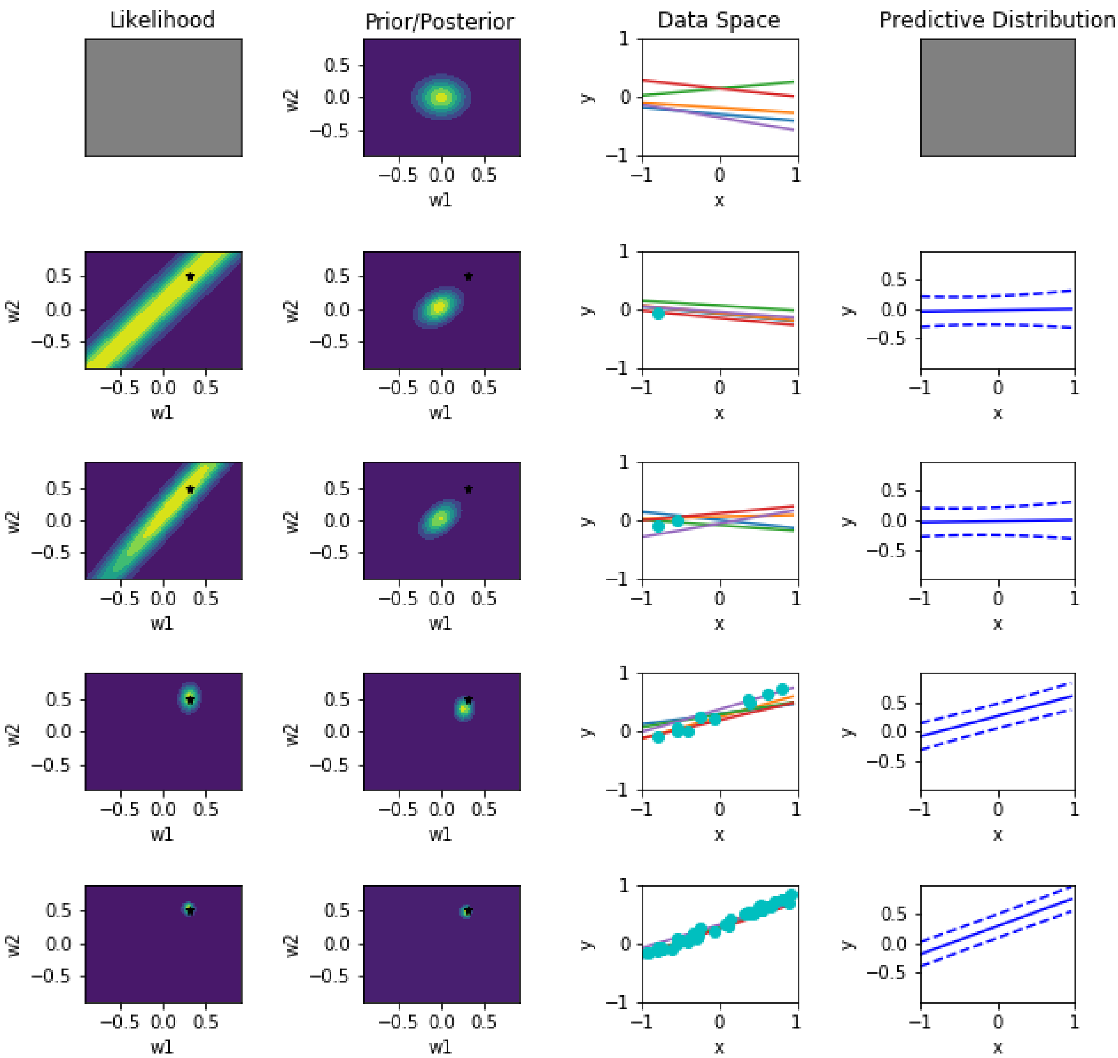
```
support_code.make_plots(actual_weights,
                        xtrain,
                        ytrain,
                        likelihood_var,
                        prior,
                        likelihood_func,
                        get_posterior_params,
                        get_predictive_params)
```



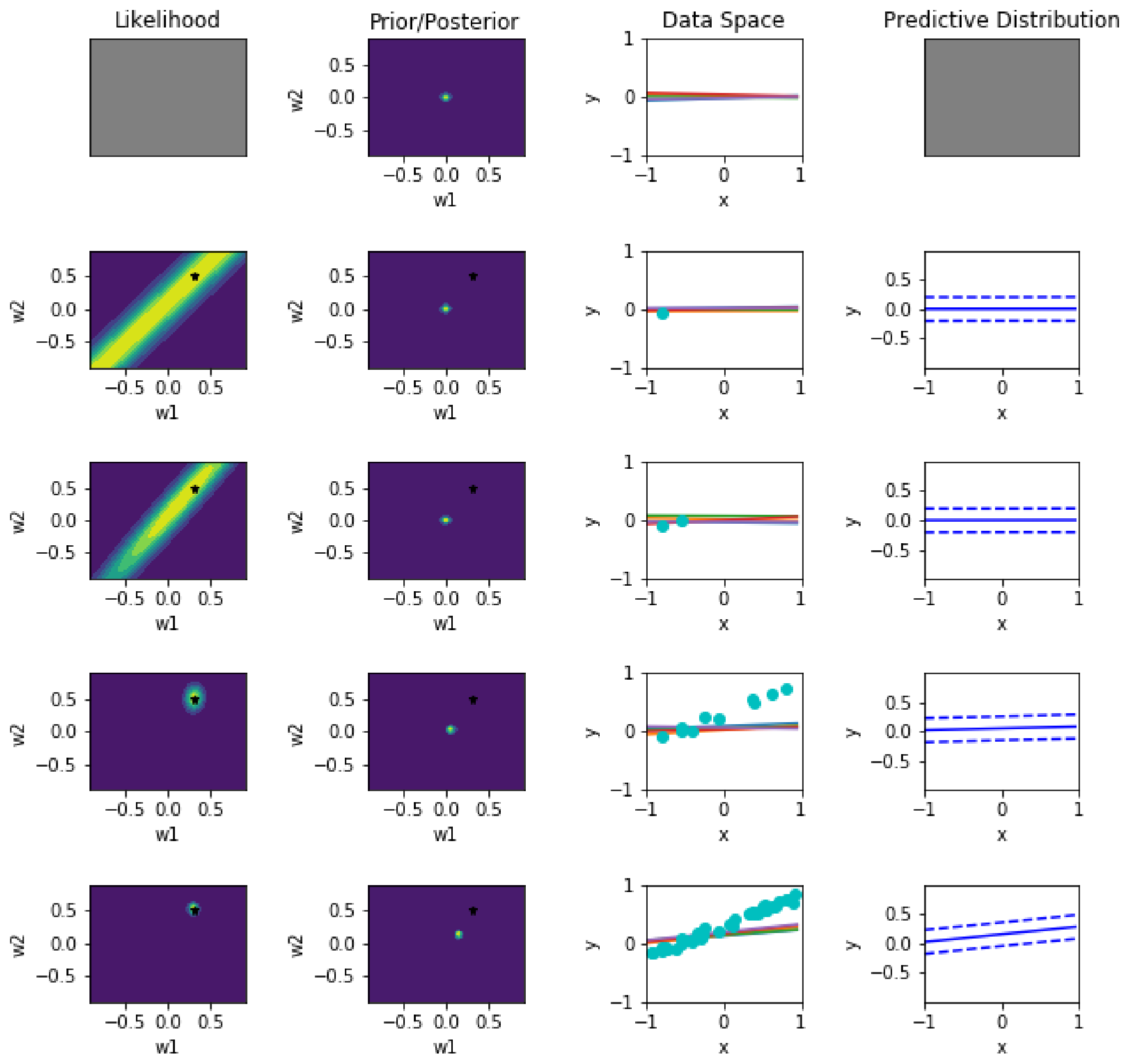




Likelihood
Prior/Posterior
Data Space
Predictive Distribution
w1
w2
x
y



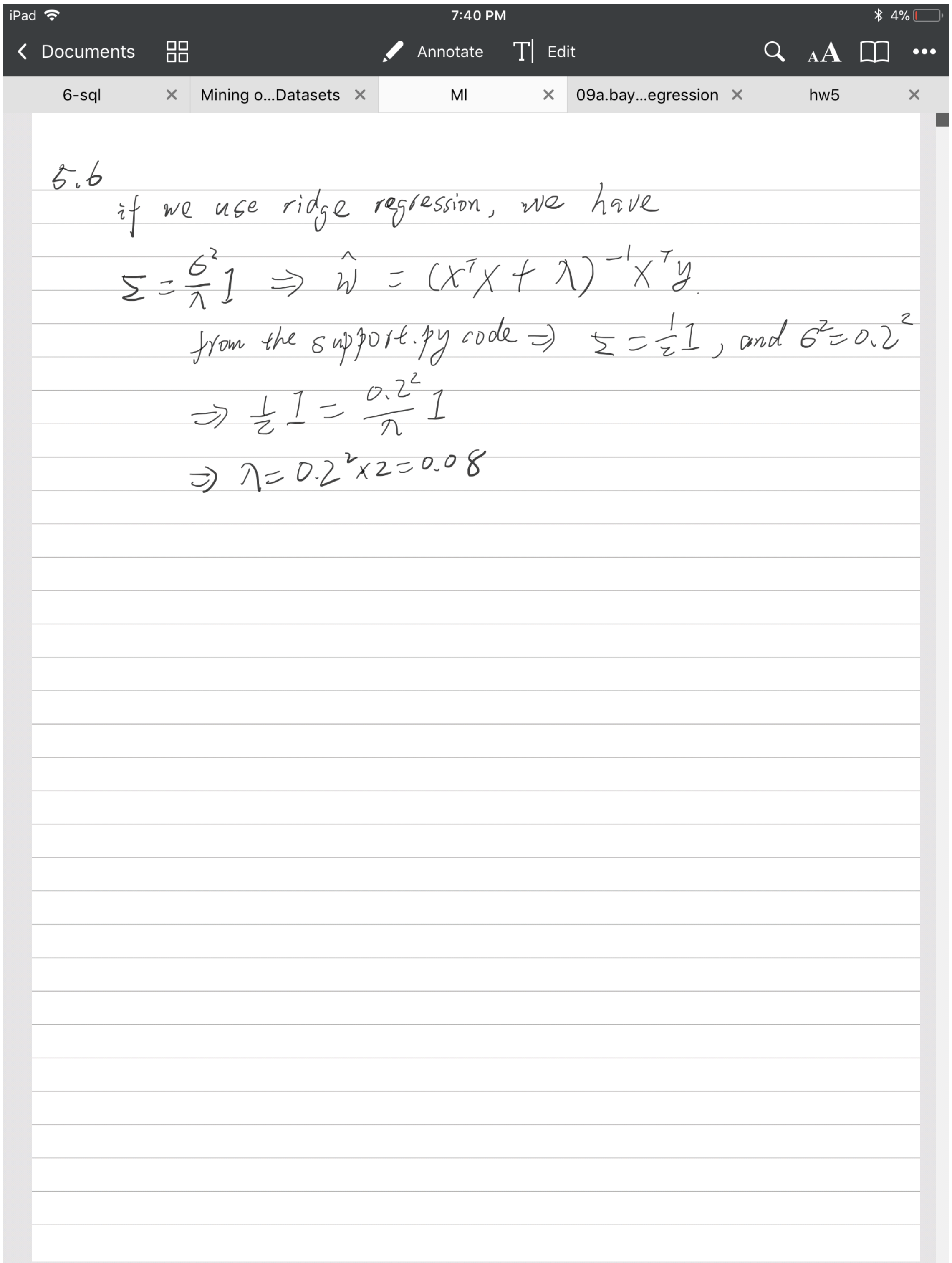
Likelihood
Prior/Posterior
Data Space
Predictive Distribution
w1
w2
x
y



5.5

(i) the larger sample size will limit the distribution of likelihood, making it in a smaller range; the stronger prior means smaller variance, but no influence on the likelihood. (ii) the larger sample size helps to predict more accurately on the posterior distribution; the stronger prior will limit the posterior distribution, making it in a smaller range, but it is also more difficult to hit the true posterior distribution of w. (iii) the larger sample size predicts more accurately on the posterior predictive distribution; the stronger prior will decrease the error bands, and also predicts less accurately on the posterior predictive distribution, because it makes more difficult to hit the true posterior distribution of w.

5.6



5.6

if we use ridge regression, we have

$$\Sigma = \frac{\sigma^2}{\lambda} I \Rightarrow \hat{w} = (X^T X + \lambda)^{-1} X^T y$$

from the support.py code $\Rightarrow \Sigma = \frac{1}{2} I$, and $\sigma^2 = 0.2^2$

$$\Rightarrow \frac{1}{2} I = \frac{0.2^2}{\lambda} I$$

$$\Rightarrow \lambda = 0.2^2 \times 2 = 0.08$$