2.1 $E_y[l(yf(x))\mid x] = l(1\cdot f(x))\cdot P(y=1\mid x) + l(-1\cdot f(x))\cdot P(y=-1\mid x)$

$= l(1f(x))\cdot \pi(x) + l(-f(x))\cdot(1-\pi(x))$

2.2 we can write $p=\pi(x)$ $\quad \hat{y}=f(x)$

then $E_y[l\mid x] = e^{-\hat{y}}\cdot p + e^{\hat{y}}\cdot(1-p)$

$\frac{\partial E_y[l\mid x]}{\partial \hat{y}} = -e^{-\hat{y}}\cdot p + e^{\hat{y}}(1-p) = 0 \Rightarrow \quad e^{2\hat{y}^*} = \frac{p}{1-p}, \; 2\hat{y}^* = \ln\left(\frac{p}{1-p}\right)$

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

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

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

2.3 $E_y[l\mid x] = \ln(1+e^{-\hat{y}})\cdot p + \ln(1+e^{\hat{y}})\cdot(1-p)$

$\frac{\partial E_y[l\mid x]}{\partial \hat{y}} = p\cdot\frac{-e^{-\hat{y}}}{1+e^{-\hat{y}}} + (1-p)\cdot\frac{e^{\hat{y}}}{1+e^{\hat{y}}} = 0 \Rightarrow [(1-p)e^{\hat{y}} - p][e^{\hat{y}}+1] = 0$

$\because e^{\hat{y}}+1 > 0$

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

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

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

2.4

3.1 $n\hat{R}_n(w) = \sum^n \log(1+e^{-y_i w^T x_i})$

$NLL(w) = \sum^n [-y_i' \log \frac{1}{1+e^{-w^T x_i}}] + (y_i'-1)\log \frac{e^{-w^T x_i}}{1+e^{-w^T x_i}}$

$= \sum^n -y_i' [\log 1 - \log(1+e^{-w^T x_i})] + (y_i'-1)(\log e^{-w^T x_i} - \log(1+e^{-w^T x_i}))$

$= \sum^n (y_i'-1)\log e^{-w^T x_i} + \log(1+e^{-w^T x_i})$

$= \sum^n y_i' \log e^{-w^T x_i} + \log \frac{1+e^{-w^T x_i}}{e^{-w^T x_i}} = \sum^n \log (e^{-w^T x_i})^{y_i'} \cdot \frac{1+e^{-w^T x_i}}{e^{-w^T x_i}}$

$\because y_i' = \begin{cases} 1, & y_i = 1 \\ 0, & y_i = -1 \end{cases}$

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

$NLL(w) = \sum^n \log(1+e^{-w^T x_i}) = n\hat{R}_n(w)$

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

$NLL(w) = \sum^n \log(1+\frac{1}{e^{-w^T x_i}}) = \sum^n \log(1+e^{w^T x_i}) = n\hat{R}_n(w)$

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

3.2 1. $LogSumExp(x_1, \cdots x_n) = \log[e^{x^*}(e^{x_1-x^*} + \cdots + e^{x_n-x^*})] = \log e^{x^*} + \log[e^{x_1-x^*} + \cdots + e^{x_n-x^*}]$

$= e^{x^*} + \log[e^{x_1-x^*} + \cdots + e^{x_n-x^*}]$

2. $\because x^* = \max(x_1, \cdots, x_n)$

$\therefore x_i - x^* \leq 0$ for any $i \in 1, \cdots, n$.

$\therefore \exp(x_i - x^*) \in (0, 1]$

3. because $x^* = \max(x_1, \cdots x_n)$

So there must exist one term among "$x_1-x^*, \cdots, x_n-x^*$" that satisfies $x_i - x^* = 0$

so $e^{x_1-x^*} + e^{x_2-x^*} + \cdots + e^{x_n-x^*} > 1$

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

4. np.logaddexp(0, -s)

3.3.1 $J'(w) = \frac{1}{n}\sum \frac{e^{-y_i w^T x_i}(-y_i x_i)}{1+e^{-y_i w^T x_i}}$

first put $\lambda\|w\|^2$ aside

$J''(w) = \frac{(-y_i x_i)^2 e^{-y_i w^T x_i}(1+e^{-y_i w^T x_i}) - e^{-y_i w^T x_i}(-y_i x_i) e^{-y_i w^T x_i}(-y_i x_i)}{(1+e^{-y_i w^T x_i})^2}$

$= \frac{(y_i x_i)^2 e^{-y_i w^T x_i} \cdot 1}{(1+e^{-y_i w^T x_i})^2} > 0$

$\therefore J(w)$ is convex

also norm-2 is convex so for $\lambda > 0$, $J_{logistic}(w)$ is convex.

In [1]:
```
import numpy as np
import pandas as pd
import logreg_skeleton as l
```

In [24]:
```
#load data
X_train = np.loadtxt('X_train.txt',delimiter=',')
X_val = np.loadtxt('X_val.txt',delimiter=',')
y_train = np.loadtxt('y_train.txt',delimiter=',')
y_val = np.loadtxt('y_val.txt',delimiter=',')
```

In [26]:
```
MIN = np.amin(X_train,axis=0)
MAX = np.amax(X_train,axis=0)
X_normalized = (X_val - MIN)/(MAX - MIN)
bias = np.ones(X_val.shape[0])
X_val = np.c_[X_normalized, bias]
y_val[y_val==0] = -1
```

In [19]:
```
optimal_theta = l.fit_logistic_reg(X_train, y_train, l.f_objective, l2_param=1
).x
```

In [21]:
```
optimal_theta
```

Out[21]:
```
array([ 0.00098726,  0.00086963,  0.00030947,  0.02207397,  0.00024182,
        0.00074032,  0.00011984,  0.00085947,  0.0009058 , -0.01224862,
        0.00195631,  0.00023594,  0.00459852,  0.00012709,  0.00047898,
        0.00117242,  0.00037251, -0.00044843, -0.00044867, -0.00069619,
        0.00178109])
```

In [27]:
```
l.f_objective(optimal_theta,X_val,y_val)
```

Out[27]:
```
0.6925989795847987
```

In [28]:

```
l.fit_logistic_reg(X_train, y_train, l.f_objective, l2_param=1)
```

Out[28]:
```
      fun: 0.6924566759413383
 hess_inv: array([[ 9.74191918e-01, -2.87436028e-02, -3.10097113e-02,
        -2.74374966e-02, -2.58121396e-02, -2.86268776e-02,
        -2.54308372e-02, -2.75374736e-02, -3.08876812e-02,
        -2.92462592e-02, -2.56223249e-02, -3.18055373e-02,
        -2.38610895e-02, -2.92923633e-02, -2.82271810e-02,
        -2.89203278e-02, -2.56657268e-02, -3.14126209e-02,
        -2.40546588e-02, -2.53970063e-02, -5.48122290e-02],
       [-2.87436028e-02,  9.67946460e-01, -3.47079621e-02,
        -2.68344191e-02, -2.88792216e-02, -3.19450828e-02,
        -2.84739527e-02, -3.07026910e-02, -3.44483459e-02,
        -3.49497012e-02, -2.83639326e-02, -3.55968387e-02,
        -2.59055142e-02, -3.28011918e-02, -3.15435785e-02,
        -3.21982411e-02, -2.86925963e-02, -3.52766817e-02,
        -2.70328081e-02, -2.85791969e-02, -6.11016639e-02],
       [-3.10097113e-02, -3.47079621e-02,  9.62041531e-01,
        -1.72713516e-02, -3.15685286e-02, -3.46593241e-02,
        -3.11916223e-02, -3.32286385e-02, -3.73148549e-02,
        -4.51461869e-02, -3.00580727e-02, -3.89478452e-02,
        -2.58716021e-02, -3.59408073e-02, -3.43629693e-02,
        -3.47001630e-02, -3.12924834e-02, -3.89723522e-02,
        -2.99237454e-02, -3.17553486e-02, -6.60923825e-02],
       [-2.74374966e-02, -2.68344191e-02, -1.72713516e-02,
         6.28604523e-01, -1.53922799e-02, -2.47294592e-02,
        -1.32042744e-02, -2.62422317e-02, -2.84822124e-02,
         1.87322443e-01, -4.31188388e-02, -1.78592832e-02,
        -8.68793523e-02, -1.48780872e-02, -2.02927479e-02,
        -3.18130540e-02, -1.74025975e-02, -6.64611568e-03,
        -3.32669069e-03, -3.15784211e-05, -5.32295123e-02],
       [-2.58121396e-02, -2.88792216e-02, -3.15685286e-02,
        -1.53922799e-02,  9.73759626e-01, -2.88321767e-02,
        -2.59218084e-02, -2.76485106e-02, -3.10457733e-02,
        -3.69418534e-02, -2.50672800e-02, -3.23722691e-02,
        -2.16976792e-02, -2.98696848e-02, -2.85736752e-02,
        -2.88873207e-02, -2.60177236e-02, -3.23593006e-02,
        -2.48416990e-02, -2.63503939e-02, -5.49983316e-02],
       [-2.86268776e-02, -3.19450828e-02, -3.46593241e-02,
        -2.47294592e-02, -2.88321767e-02,  9.68151408e-01,
        -2.84389689e-02, -3.05956389e-02, -3.43337865e-02,
        -3.60820944e-02, -2.81561438e-02, -3.55453296e-02,
        -2.54407435e-02, -3.27628945e-02, -3.14721199e-02,
        -3.20612699e-02, -2.86337045e-02, -3.52893824e-02,
        -2.70527626e-02, -2.86203371e-02, -6.08828993e-02],
       [-2.54308372e-02, -2.84739527e-02, -3.11916223e-02,
        -1.32042744e-02, -2.59218084e-02, -2.84389689e-02,
         9.74381851e-01, -2.72575046e-02, -3.06121581e-02,
        -3.76471653e-02, -2.46060065e-02, -3.19854656e-02,
        -2.10252780e-02, -2.95215867e-02, -2.82072674e-02,
        -2.84544760e-02, -2.56900195e-02, -3.20345708e-02,
        -2.46021965e-02, -2.61157079e-02, -5.42148111e-02],
       [-2.75374736e-02, -3.07026910e-02, -3.32286385e-02,
        -2.62422317e-02, -2.76485106e-02, -3.05956389e-02,
        -2.72575046e-02,  9.70590512e-01, -3.29958958e-02,
        -3.31433609e-02, -2.71985159e-02, -3.40782391e-02,
        -2.49127613e-02, -3.13995319e-02, -3.02047318e-02,
        -3.08488240e-02, -2.74731526e-02, -3.37545673e-02,
        -2.58637265e-02, -2.73377035e-02, -5.85297655e-02],
       [-3.08876812e-02, -3.44483459e-02, -3.73148549e-02,
        -2.84822124e-02, -3.10457733e-02, -3.43337865e-02,
        -3.06121581e-02, -3.29958958e-02,  9.62977694e-01,
        -3.77835378e-02, -3.04632837e-02, -3.82686081e-02,
        -2.77724931e-02, -3.52648656e-02, -3.39064889e-02,
        -3.45989201e-02, -3.08430493e-02, -3.79356145e-02,
        -2.90722355e-02, -3.07386888e-02, -6.56645374e-02],
       [-2.92462592e-02, -3.49497012e-02, -4.51461869e-02,
```

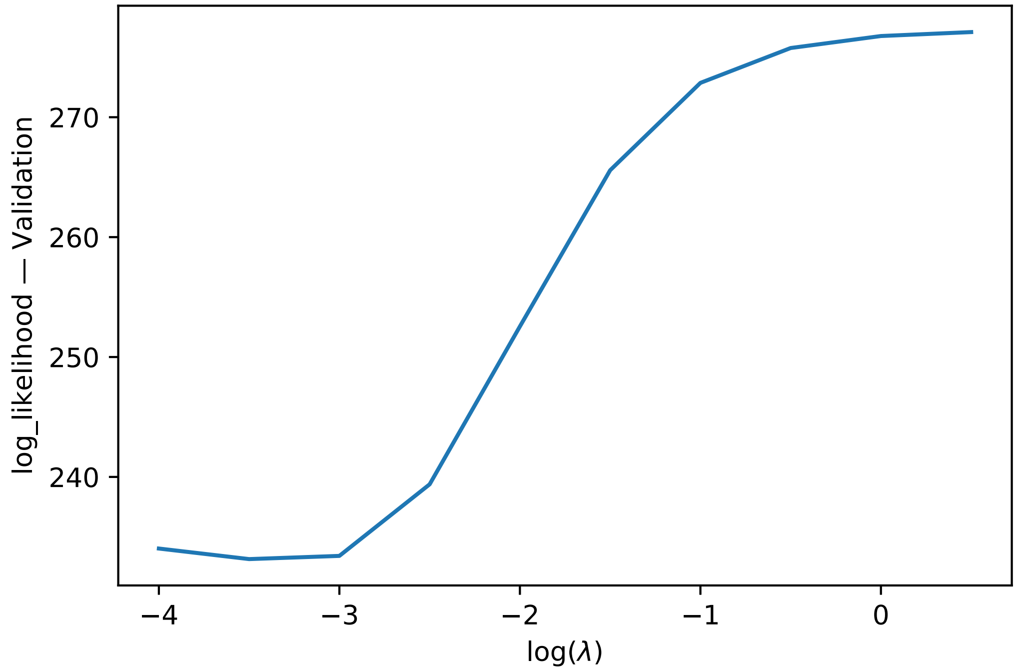
```
 1.87322443e-01, -3.69418534e-02, -3.60820944e-02,
-3.76471653e-02, -3.31433609e-02, -3.77835378e-02,
 8.27467769e-01, -1.89014813e-02, -4.62251478e-02,
 1.22050655e-02, -4.35737176e-02, -3.81894889e-02,
-3.20921761e-02, -3.53931149e-02, -5.26706122e-02,
-4.14578118e-02, -4.59940261e-02, -6.53339396e-02],
[-2.56223249e-02, -2.83639326e-02, -3.00580727e-02,
-4.31188388e-02, -2.50672800e-02, -2.81561438e-02,
-2.46060065e-02, -2.71985159e-02, -3.04632837e-02,
-1.89014813e-02,  9.73825340e-01, -3.08363533e-02,
-2.65435814e-02, -2.83269034e-02, -2.75731819e-02,
-2.87616672e-02, -2.50224134e-02, -2.99453815e-02,
-2.28493086e-02, -2.39629059e-02, -5.41837163e-02],
[-3.18055373e-02, -3.55968387e-02, -3.89478452e-02,
-1.78592832e-02, -3.23722691e-02, -3.55453296e-02,
-3.19854656e-02, -3.40782391e-02, -3.82686081e-02,
-4.62251478e-02, -3.08363533e-02,  9.60059501e-01,
-2.65382991e-02, -3.68577290e-02, -3.52397758e-02,
-3.55912936e-02, -3.20908610e-02, -3.99595401e-02,
-3.06817624e-02, -3.25561852e-02, -6.77849683e-02],
[-2.38610895e-02, -2.59055142e-02, -2.58716021e-02,
-8.68793523e-02, -2.16976792e-02, -2.54407435e-02,
-2.10252780e-02, -2.49127613e-02, -2.77724931e-02,
 1.22050655e-02, -2.65435814e-02, -2.65382991e-02,
 9.66906870e-01, -2.41607388e-02, -2.43502276e-02,
-2.69304406e-02, -2.19524559e-02, -2.42330688e-02,
-1.82407695e-02, -1.86308186e-02, -4.97691307e-02],
[-2.92923633e-02, -3.28011918e-02, -3.59408073e-02,
-1.48780872e-02, -2.98696848e-02, -3.27628945e-02,
-2.95215867e-02, -3.13995319e-02, -3.52648656e-02,
-4.35737176e-02, -2.83269034e-02, -3.68577290e-02,
-2.41607388e-02,  9.65980140e-01, -3.24998821e-02,
-3.27740769e-02, -2.96005202e-02, -3.69248969e-02,
-2.83594823e-02, -3.01076092e-02, -6.24520502e-02],
[-2.82271810e-02, -3.15435785e-02, -3.43629693e-02,
-2.02927479e-02, -2.85736752e-02, -3.14721199e-02,
-2.82072674e-02, -3.02047318e-02, -3.39064889e-02,
-3.81894889e-02, -2.75731819e-02, -3.52397758e-02,
-2.43502276e-02, -3.24998821e-02,  9.68851134e-01,
-3.16008666e-02, -2.83520760e-02, -3.51164716e-02,
-2.69410134e-02, -2.85431651e-02, -6.00931354e-02],
[-2.89203278e-02, -3.21982411e-02, -3.47001630e-02,
-3.18130540e-02, -2.88873207e-02, -3.20612699e-02,
-2.84544760e-02, -3.08488240e-02, -3.45989201e-02,
-3.20921761e-02, -2.87616672e-02, -3.55912936e-02,
-2.69304406e-02, -3.27740769e-02, -3.16008666e-02,
 9.67588791e-01, -2.87299985e-02, -3.51173517e-02,
-2.68861094e-02, -2.83756264e-02, -6.14064266e-02],
[-2.56657268e-02, -2.86925963e-02, -3.12924834e-02,
-1.74025975e-02, -2.60177236e-02, -2.86337045e-02,
-2.56900195e-02, -2.74731526e-02, -3.08430493e-02,
-3.53931149e-02, -2.50224134e-02, -3.20908610e-02,
-2.19524559e-02, -2.96005202e-02, -2.83520760e-02,
-2.87299985e-02,  9.74190417e-01, -3.20118510e-02,
-2.45644515e-02, -2.60357221e-02, -5.46555067e-02],
[-3.14126209e-02, -3.52766817e-02, -3.89723522e-02,
-6.64611568e-03, -3.23593006e-02, -3.52893824e-02,
-3.20345708e-02, -3.37545673e-02, -3.79356145e-02,
-5.26706122e-02, -2.99453815e-02, -3.99595401e-02,
-2.42330688e-02, -3.69248969e-02, -3.51164716e-02,
-3.51173517e-02, -3.20118510e-02,  9.59675001e-01,
-3.10172720e-02, -3.30200709e-02, -6.71094686e-02],
[-2.40546588e-02, -2.70328081e-02, -2.99237454e-02,
-3.32669069e-03, -2.48416990e-02, -2.70527626e-02,
-2.46021965e-02, -2.58637265e-02, -2.90722355e-02,
```

```
       -4.14578118e-02, -2.28493086e-02, -3.06817624e-02,
       -1.82407695e-02, -2.83594823e-02, -2.69410134e-02,
       -2.68861094e-02, -2.45644515e-02, -3.10172720e-02,
        9.76133484e-01, -2.54245831e-02, -5.14160499e-02],
      [-2.53970063e-02, -2.85791969e-02, -3.17553486e-02,
       -3.15784211e-05, -2.63503939e-02, -2.86203371e-02,
       -2.61157079e-02, -2.73377035e-02, -3.07386888e-02,
       -4.59940261e-02, -2.39629059e-02, -3.25561852e-02,
       -1.86308186e-02, -3.01076092e-02, -2.85431651e-02,
       -2.83756264e-02, -2.60357221e-02, -3.30200709e-02,
       -2.54245831e-02,  9.72882584e-01, -5.43364294e-02],
      [-5.48122290e-02, -6.11016639e-02, -6.60923825e-02,
       -5.32295123e-02, -5.49983316e-02, -6.08828993e-02,
       -5.42148111e-02, -5.85297655e-02, -6.56645374e-02,
       -6.53339396e-02, -5.41837163e-02, -6.77849683e-02,
       -4.97691307e-02, -6.24520502e-02, -6.00931354e-02,
       -6.14064266e-02, -5.46555067e-02, -6.71094686e-02,
       -5.14160499e-02, -5.43364294e-02,  8.83513386e-01]])
     jac: array([ 7.45058060e-08,  1.49011612e-08,  1.49011612e-08, -1.71363
354e-07,
        1.34110451e-07,  0.00000000e+00, -6.70552254e-08, -7.45058060e-09,
        7.45058060e-09,  2.30967999e-07, -8.94069672e-08, -7.45058060e-08,
        3.72529030e-08,  2.23517418e-08,  2.23517418e-08,  4.47034836e-08,
        0.00000000e+00,  0.00000000e+00,  5.96046448e-08,  6.70552254e-08,
        1.49011612e-08])
 message: 'Optimization terminated successfully.'
    nfev: 161
     nit: 3
    njev: 7
  status: 0
 success: True
       x: array([ 0.00098726,  0.00086963,  0.00030947,  0.02207397,  0.0002
4182,
        0.00074032,  0.00011984,  0.00085947,  0.0009058 , -0.01224862,
        0.00195631,  0.00023594,  0.00459852,  0.00012709,  0.00047898,
        0.00117242,  0.00037251, -0.00044843, -0.00044867, -0.00069619,
        0.00178109])
```

3.3.4

In [43]:
```
l2_param_list = np.arange(-4,1,0.5)
loss_list = []
for l2_param in l2_param_list:
    l2 = 10 **l2_param
    opt_theta = l.fit_logistic_reg(X_train, y_train, l.f_objective, l2_param=l
2).x
    loss = l.f_objective(opt_theta,X_val,y_val,l2_param=0) * len(y_val)
    loss_list.append(loss)
```

In [717]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
plt.plot(l2_param_list,loss_list)
plt.xlabel('log($\lambda$)')
plt.ylabel("log_likelihood — Validation")
plt.savefig('1.png')
plt.show()
```



- l2_param = 0.001 minimizes the log_likelihood

3.3.5

In [49]:
```
opt_theta = l.fit_logistic_reg(X_train, y_train, l.f_objective, l2_param=0.001
).x
```

In [697]:
```
y_pred = 1/(1+ np.exp(-np.dot(X_val,opt_theta)))
```

In [303]:
```
y_pred = np.zeros(len(y_val))
for i in range(len(y_pred)):
    y_pred[i] = 1/(1+ np.exp(-np.dot(opt_theta,X_val[i])))
```

In [702]:
```
#original table
import pandas as pd
df = pd.DataFrame(columns=['y_val','y_pred'])

for idx in range(len(y_val)):
    df.loc[idx] = [y_val[idx],y_pred[idx]]
```

In [703]:
```python
bins1 = np.arange(0,1.1,0.1)
labels1 = np.arange(0.05,1,0.1)
bins2 = np.arange(-0.05,1.1,0.1)
labels2 =np.arange(0.1,1.1,0.1)
labels2 = np.append(0.025,labels2)
labels2
```

Out[703]:
```
array([0.025, 0.1  , 0.2  , 0.3  , 0.4  , 0.5  , 0.6  , 0.7  , 0.8  ,
       0.9  , 1.   ])
```

In [704]:
```python
# pd.value_counts(pd.cut(df['y_bin'],bins1))
df['y_bin'] = pd.cut(df['y_pred'], bins=bins1, labels=labels)
```

In [705]:
```python
g = df.groupby(["y_bin", "y_val"]).count()
g = g.fillna(0)
```

In [706]:
```python
g1 = g.groupby(level=[0]).apply(lambda g: g/g.sum())
g1 = g1.iloc[g1.index.get_level_values('y_val')==1]
g1 = g1.reset_index()

g1['y_bin'] = g1['y_bin'].astype('float')
g1.rename(index=str, columns={"y_pred": "percentage"})
```

Out[706]:

	y_bin	y_val	percentage
0	0.05	1.0	0.000000
1	0.15	1.0	0.454545
2	0.25	1.0	0.254902
3	0.35	1.0	0.241379
4	0.45	1.0	0.350649
5	0.55	1.0	0.666667
6	0.65	1.0	0.840909
7	0.75	1.0	0.818182
8	0.85	1.0	0.812500
9	0.95	1.0	0.800000

In [707]:
```python
df['y_bin'] = pd.cut(df['y_pred'], bins=bins2, labels=labels2)
```

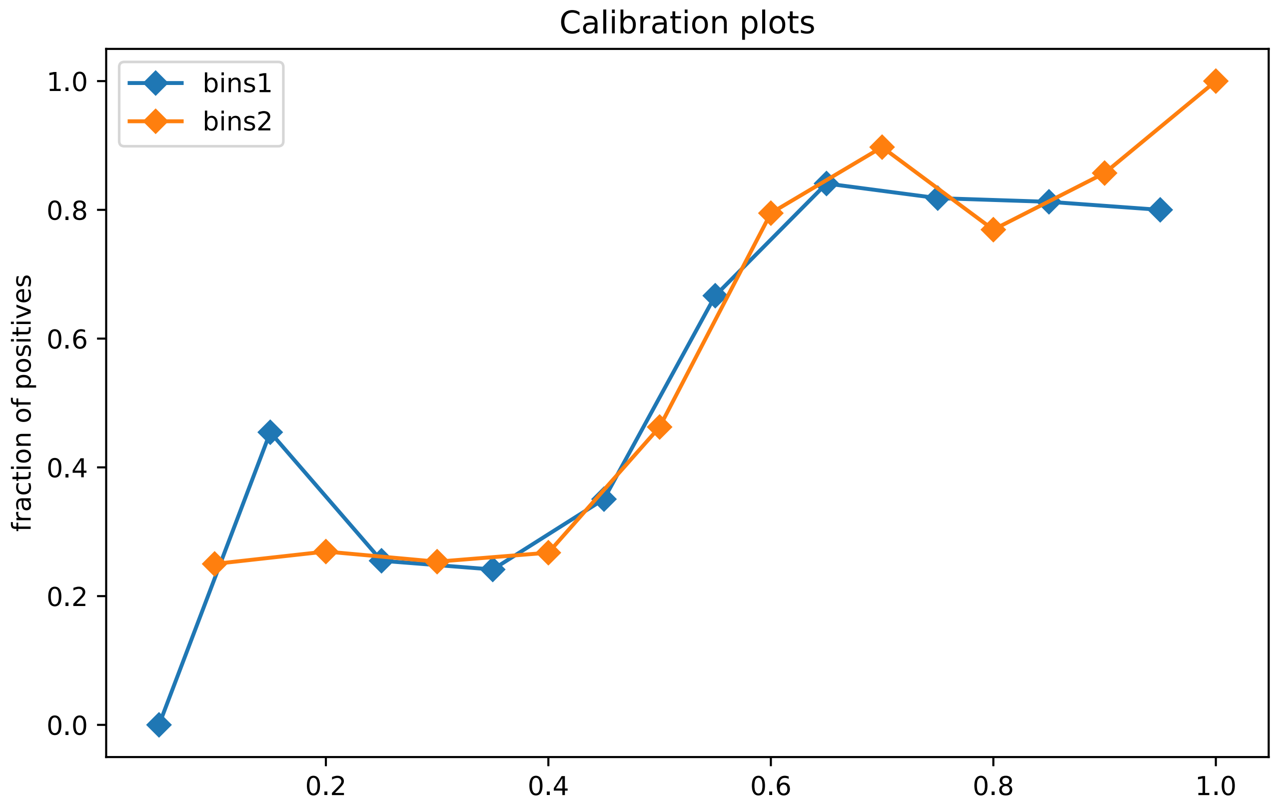
In [708]:
```python
g = df.groupby(["y_bin", "y_val"]).count()
g = g.fillna(0)
```

In [709]:
```
g2 = g.groupby(level=[0]).apply(lambda g: g/g.sum())
g2 = g2.iloc[g2.index.get_level_values('y_val')==1]
g2 = g2.reset_index()

g2['y_bin'] = g2['y_bin'].astype('float')
g2.rename(index=str, columns={"y_pred": "percentage"})
```

Out[709]:

	y_bin	y_val	percentage
0	0.025	1.0	NaN
1	0.100	1.0	0.250000
2	0.200	1.0	0.269231
3	0.300	1.0	0.253521
4	0.400	1.0	0.267442
5	0.500	1.0	0.462687
6	0.600	1.0	0.794872
7	0.700	1.0	0.897436
8	0.800	1.0	0.769231
9	0.900	1.0	0.857143
10	1.000	1.0	1.000000

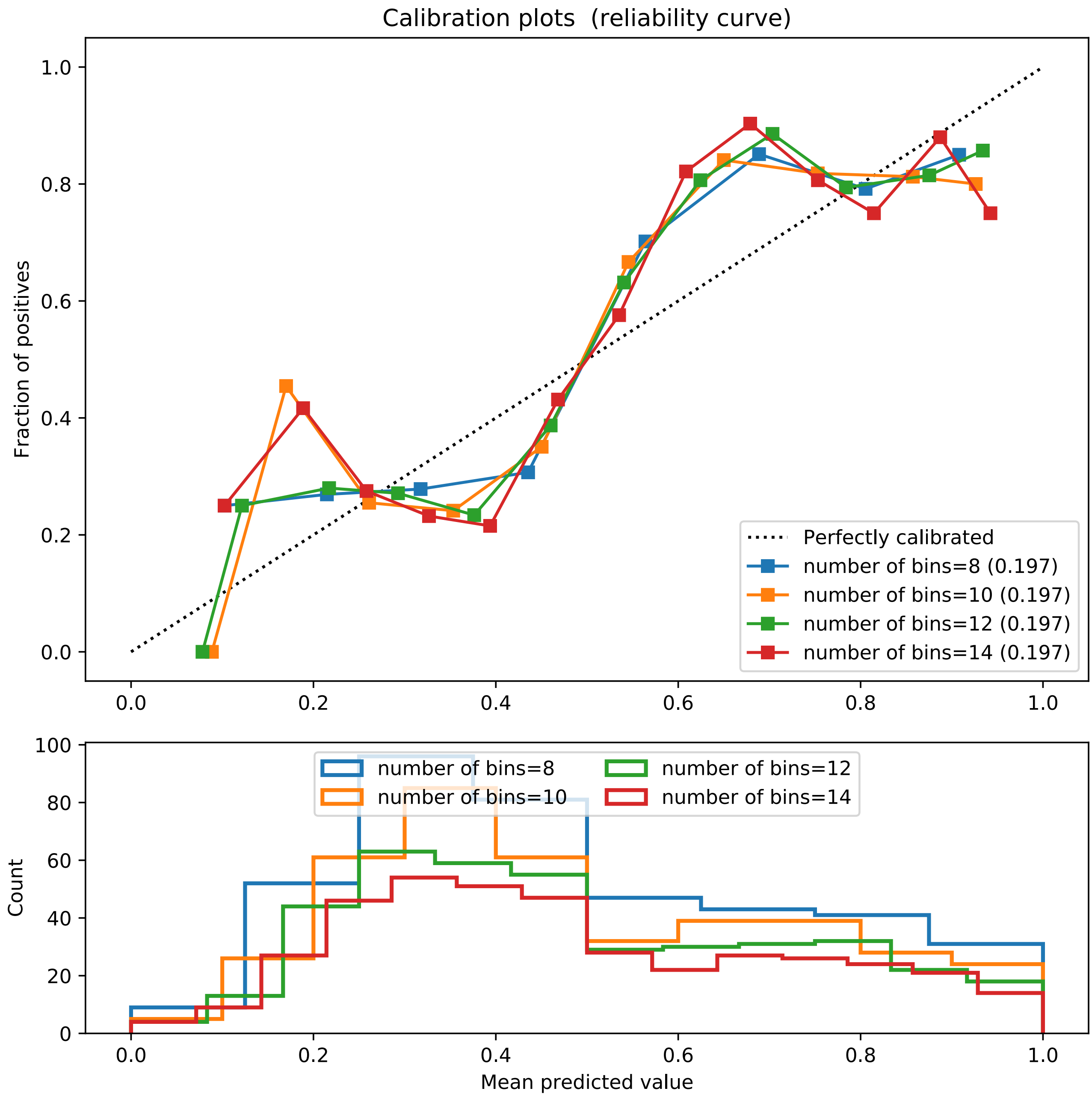
In [716]:
```
plt.figure(figsize=(8, 5))

plt.plot(g1['y_bin'],g1['y_pred'],marker='D', label ='bins1')
plt.plot(g2['y_bin'],g2['y_pred'],marker='D', label ='bins2')
x = np.linspace(0, 1, 1000)

plt.legend()
plt.ylabel('fraction of positives')
plt.title('Calibration plots')
plt.savefig('2.png')
plt.show()
```



In [660]:
```
prob_pos = (y_pred - y_pred.min()) / (y_pred.max() - y_pred.min())
```

In [718]:

```
from sklearn.calibration import calibration_curve
plt.figure(figsize=(8, 8))
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")
bins = [8,10,12,14]
for b in bins:
    fraction_of_positives, mean_predicted_value = calibration_curve(y_val, y_p
red, n_bins=b)
    ax1.plot(mean_predicted_value, fraction_of_positives, "s-", label="number
 of bins=%s (%1.3f)" % (b, clf_score))

    ax2.hist(prob_pos, range=(0, 1), bins=b,histtype="step", lw=2,label="numbe
r of bins=%s" % (b, ))

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.savefig('3.png')
plt.show()
```


Calibration plots (reliability curve)
Fraction of positives
Perfectly calibrated
number of bins=8 (0.197)
number of bins=10 (0.197)
number of bins=12 (0.197)
number of bins=14 (0.197)
Count
number of bins=8
number of bins=10
number of bins=12
number of bins=14
Mean predicted value

```
In [ ]:
```

4.1 $P(w|D') = \dfrac{P(D'|w)\cdot P(w)}{P(D')} = e^{-NLL_{D'}(w} \cdot p(w)$

4.2 $p(w) = \dfrac{1}{\sqrt{(2\pi)^d|\Sigma|}} e^{-\frac{1}{2}w^T\Sigma^{-1}w}$

negative log posterior $= -\sum_{i=1}^{n} y_i' \log\phi(w^Tx_i) + (1-y_i')\log(1-\phi(w^Tx_i)) + \left(\log\frac{1}{\sqrt{(2\pi)^d|\Sigma|}} + (-\frac{1}{2}w^T\Sigma^{-1}w)\right)$

$$\begin{cases} \dfrac{\partial\ \text{posterior}}{\partial w} = \left(-\sum^{n} y_i' \cdot \dfrac{\phi'\cdot x_i}{\phi(w^Tx_i)} + (1-y_i')\dfrac{-\phi'\cdot x_i}{1-\phi(w^Tx_i)}\right) - \Sigma^{-1}w = 0 & ① \\ \dfrac{\partial J_{logistic}(w)}{\partial w} = \left(\dfrac{1}{n}\sum^{n}\dfrac{e^{-y_iw^Tx_i}\cdot(-y_iX_i)}{1+e^{-y_iw^Tx_i}}\right) + 2\lambda w = 0 & ② \end{cases}$$

we can rewrite ② as $\left(\frac{1}{n}\sum y_i' \cdot \dfrac{e^{-w^Tx_i}(-X_i)}{1+e^{-w^Tx_i}} + (1-y_i')\cdot\dfrac{e^{w^Tx_i}\cdot(X_i)}{1+e^{w^Tx_i}}\right) + 2\lambda w = 0$

$= \dfrac{X_i}{e^{-w^Tx_i}+1}$

$\because\ \phi(\eta) = \dfrac{1}{1+e^{-\eta}}$

$\therefore$ we can rewrite ① as $\left(-\sum y_i' \cdot \dfrac{e^{-w^Tx_i}\cdot X_i}{1+e^{-w^Tx_i}} + (1-y_i')\dfrac{X_i}{e^{-w^Tx_i}+1}\right) - \Sigma^{-1}w = 0$

$\therefore$ if the above two equations give the same w, we should have

$\Sigma^{-1}w = 2\lambda w n$

then we must have $\Sigma^{-1} = 2\lambda n I \Rightarrow \Sigma = \frac{1}{2\lambda n} I$

4.3 $p(w) = \dfrac{1}{\sqrt{(2\pi)^d|I|}} e^{-\frac{1}{2}w^TI^{-1}w}$

mode $= \underset{w}{\text{argmax}}\ \text{posterior}(w) =$ solution of $\dfrac{\partial\ \text{posterior}(w)}{\partial w} = 0$

according to 4.2, $2\lambda n I = I^{-1} \Rightarrow \lambda = \frac{1}{2n}$

5.6 alpha $= 0.08$

$w \sim N(0, \frac{1}{2}I)$

$J(w) = \sum^{n}(w^Tx_i - y_i)^2 + \lambda\|w\|^2$

$J'(w) = \sum 2(w^Tx_i - y_i)\cdot x_i + 2\lambda w = 0.$ ①

$p(w|D) \propto e^{-\frac{1}{2\sigma^2}\left[\sum^{n}(y_i - w^Tx_i)^2\right]} \times e^{-\frac{1}{2}w^T(\frac{1}{2}I)^{-1}w} \triangleq A$

$\log(A) = -\frac{1}{2\sigma^2}\left[\sum^{n}(y_i - w^Tx_i)^2\right] + \left[-\frac{1}{2}w^T\cdot 2I\cdot w\right]$

$\dfrac{\partial\log(A)}{\partial w} = -\frac{1}{2\sigma^2}\sum^{n} 2(y_i - w^Tx_i)\cdot x_i - 2w = 0$ ②

① and ② should give the same w, then $2\lambda w = 2w\cdot 2\sigma^2 \Rightarrow \lambda = 2\sigma^2 = 2\times 0.2^2 = 0.08$

Recreating figure 3.7 from Bishop's "Pattern Recognition and Machine Learning."

This notebook provides scaffolding for your exploration Bayesian Linear Gaussian Regression, as described in Lecture. In particular, through this notebook you will reproduce several variants of figure 3.7 from Bishop's book.

Instructions:

5.1-3:

Implement the functions in `problem` -- completed implementations of these functions are needed to generate the plots.

In [1]:
```
from support_code import *
from problem import *
```

Instructions (continued):

5.4:

If your implementations are correct, then the next few code blocks in this notebook will generate the required variants of Bishop's figure. These are the same figures that you would obtain if you ran `python problem.py` from the command line -- this notebook is just provided as additional support.

In [2]:
```
# Generate our simulated dataset
# Note we are using sigma == 0.2

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 = generate_data(data_size,
                               noise,
                               actual_weights)
```

Next, we generate the plots using 3 different prior covariance matrix. In the main call to `problem.py` , this is done in a loop -- here we wrap the loop body in a short helper function.

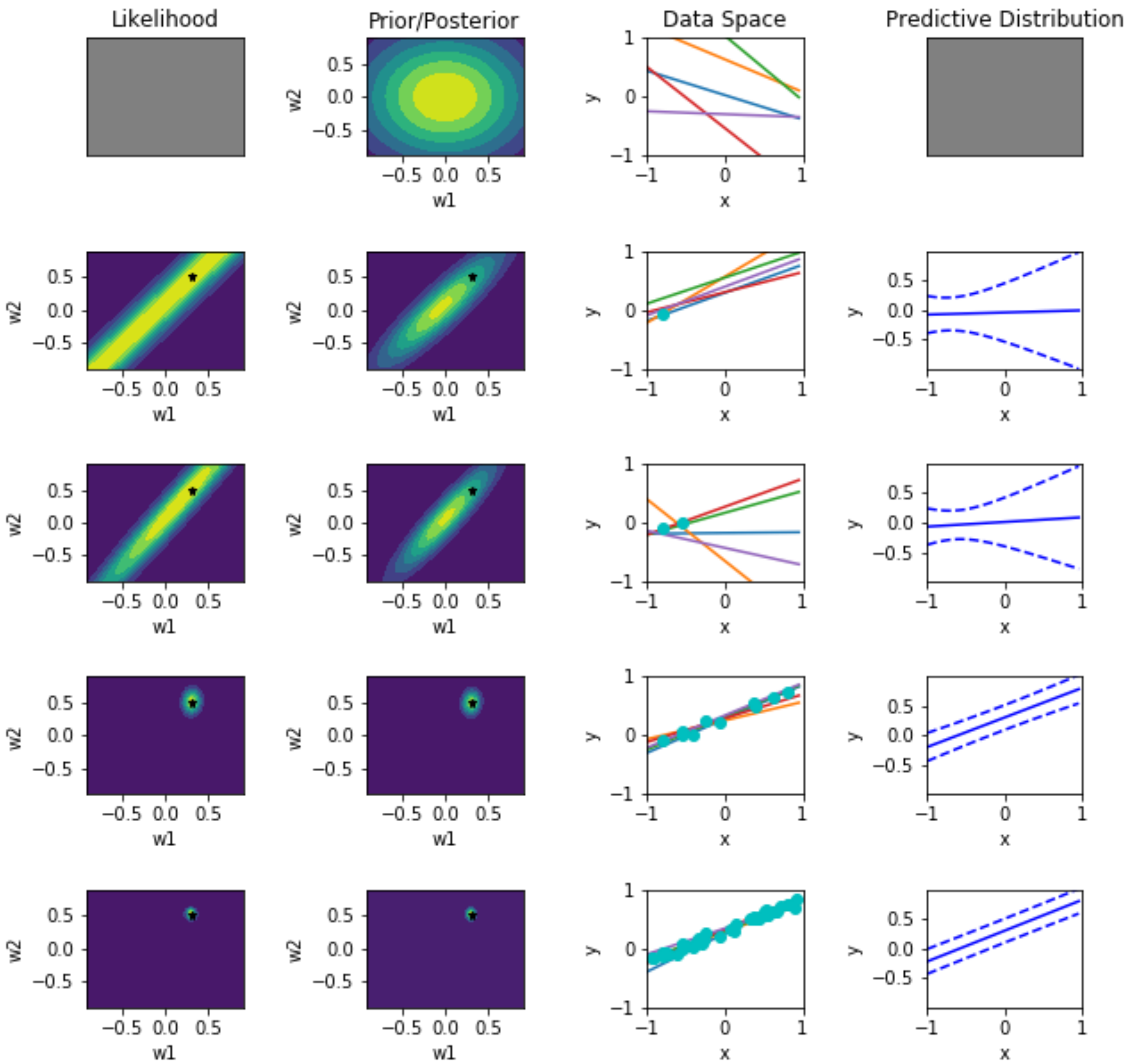
In [3]:
```
def make_plot_given_sigma(sigma_squared):
    prior = {"mean":np.matrix([[0], [0]]),
             "var":matlib.eye(2) * sigma_squared}

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

```
In [4]: sigmas = [1/2, 1/(2**5), 1/(2**10)]
```

First covariance matrix:

$$\Sigma_0 = \frac{1}{2}I, \qquad I \in \mathbb{R}^{2\times 2}$$

```
In [5]: make_plot_given_sigma(sigmas[0])
```

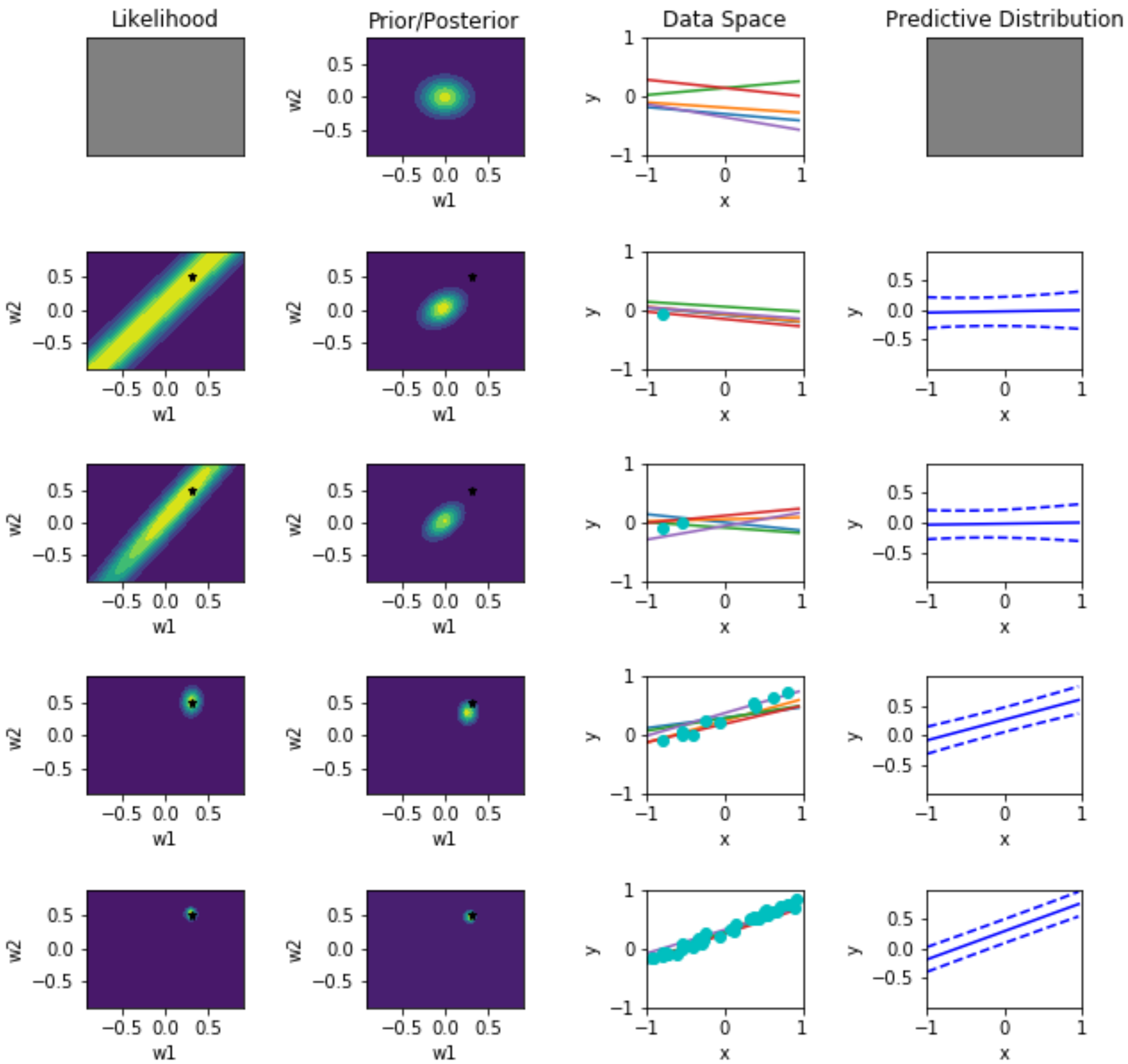


Second covariance matrix:

$$\Sigma_0 = \frac{1}{2^5}I, \qquad I \in \mathbb{R}^{2\times 2}$$

In [6]:

```python
try:
    make_plot_given_sigma(sigmas[1])
except NameError:
    print('If not yet implemented, implement functions in problem.py.')
    print('If you have implemented, remove this try/except.')
```

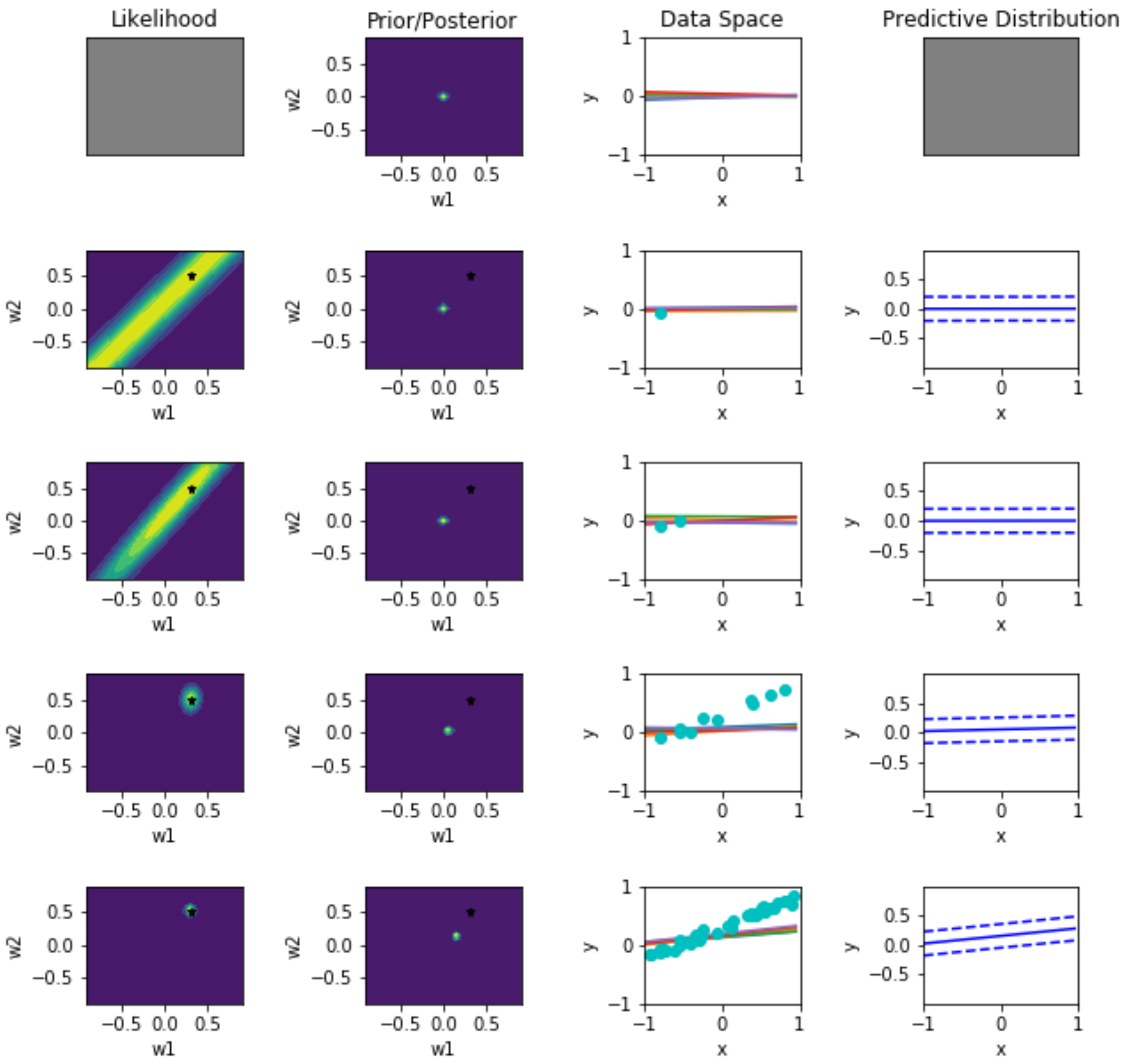


Third covariance matrix:

$$\Sigma_0 = \frac{1}{2^{10}} I, \qquad I \in \mathbb{R}^{2\times 2}$$

In [7]:

```
try:
    make_plot_given_sigma(sigmas[2])
except NameError:
    print('If not yet implemented, implement functions in problem.py.')
    print('If you have implemented, remove this try/except.')
```



Instructions (continued):

5.5:

For questiion (5) (Comment on your results ...) no code is required -- instead please answer with a written description.

Answer:

(1)Sample size: For each covariance matrix, we have five rows of plots, from 1st row to fifth row, it shows the result as the number of the observed data points increases. The first row corresponds to the situation before any data points are observed so it shows the prior distribution in w. And from 2nd row to fifth, as sample size increases, (i)the likelihood bocomes more and more compact to the actual parameters, (ii)and the posterior distribution(which is represented by the light circle in the 2nd column) gets closer to actual parameters(represented by the dark circle in the 2nd column). (iii)And for the predictive distribution, the line becomes sharper, and closer to the actual linear regression model.

(2)strength of the prior: We can compare plots with different covariance matrix, we can see as the variance of prior gets smaller, (i) the likelihood function remains the same as likelihood is not related to variance of

w; (ii) the posterior distribution becomes more compact(for the same sample size), and when variance is really small(e.g.the third covaiance matrix), the posterior distribution may not reach the actual weight(it's because the prior distribution is of very small range(concentrated around mean), so it may dominates the posterior); (iii) the posterior predictive distribution becomes more and more compact, specifically the error bands get narrower and narrower as the variance in w gets smaller(which is apparant when sample size is small), and when variance is really small, the predictive distribution cannot get close to the actual linear regression model.

Instructions (continued):

5.6:

For question (6), find the MAP solution for the first prior covariance $\left(\frac{1}{2}I\right)$ by completing the implementation below. In addition, be sure to justify the value for the regularization coefficient (in `sklearn` named `alpha`) in your written work.

In [8]:
```
from sklearn.linear_model import Ridge
```

In [9]:
```
alpha = 0.08 # Change to the correct value
ridge = Ridge(alpha=alpha,
              fit_intercept=False,
              solver='cholesky')

ridge.fit(xtrain, ytrain)
```

Out[9]:
```
Ridge(alpha=0.08, copy_X=True, fit_intercept=False, max_iter=None,
   normalize=False, random_state=None, solver='cholesky', tol=0.001)
```

If alpha is set correctly, ridge.coef_ will equal the prior mean/MAP estimate returned by the next two cells.

In [10]:
```
ridge.coef_
```

Out[10]:
```
array([[0.30052135, 0.52406189]])
```

In [11]:
```
prior = {"mean":np.matrix([[0], [0]]),
         "var":matlib.eye(2) * sigmas[0]}

post = get_posterior_params(xtrain, ytrain, prior,
                            likelihood_var = 0.2**2)
post[0].ravel()
```

Out[11]:
```
matrix([[0.30052135, 0.52406189]])
```

Homework 5: Conditional Probability Models

Rui Jiang NetID: rj1407

3.3Regularized-Logistic-Regression

For a dataset $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n))$ drawn from $\mathbf{R}^d \times \{-1, 1\}$, the regularized logistic regression objective function can be defined as

$$\begin{aligned} J_{\text{logistic}}(w) &= \hat{R}_n(w) + \lambda\|w\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i w^T x_i\right)\right) + \lambda\|w\|^2. \end{aligned}$$

1. Prove that the objective function $J_{\text{logistic}}(w)$ is convex. You may use any facts mentioned in the convex optimization notes.

2. Complete the `f_objective` function in the skeleton code, which computes the objective function for $J_{\text{logistic}}(w)$. Make sure to use the log-sum-exp trick to get accurate calculations and to prevent overflow.

```
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
    '''
    import numpy as np
    res = 0
    for i in range(len(y)):
        res += np.logaddexp(0,(-y[i]*np.dot(theta,X[i])))

    return res/len(y) + l2_param * np.dot(theta,theta)
```

3. Complete the `fit_logistic_regression_function` in the skeleton code using the `minimize` function from `scipy.optimize`. `ridge_regression.py` from Homework 2 gives an example of how to use the `minimize` function. Use this function to train a model on the provided data. Make sure to take the appropriate preprocessing steps, such as standardizing the data and adding a column for the bias term.

```
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
    '''
    import numpy as np
    from scipy.optimize import minimize
    MIN = np.amin(X, axis=0)
    MAX = np.amax(X, axis=0)
    X_normalized = (X - MIN)/(MAX - MIN)
    bias = np.ones(X.shape[0])
    X = np.c_[X_normalized, bias]
    y[y==0] = -1

    theta = np.zeros(X.shape[1])

    def transform_func(theta):   #func in minimize only contain one attribute theta
        return objective_function(theta, X, y, l2_param)

    optimal_theta = minimize(transform_func,theta)
    return optimal_theta
```

5.Bayesian Linear Regression - Implementation

In this problem, we will implement Bayesian Gaussian linear regression, essentially reproducing the example from lecture, which in turn is based on the example in Figure 3.7 of Bishop's *Pattern Recognition and Machine Learning* (page 155). We've provided plotting functionality in "support_code.py". Your task is to complete "problem.py". The implementation uses np.matrix objects, and you are welcome to use[1] the np.matrix.getI method.

(a) Implement likelihood_func.

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

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:
```

[1] However, in practice we are usually interested in computing the product of a matrix inverse and a vector, i.e. $X^{-1}b$. In this case, it's usually faster and more accurate to use a library's algorithms for solving a system of linear equations. Note that $y = X^{-1}b$ is just the solution to the linear system $Xy = b$. See for example John Cook's blog post for discussion.

```
        likelihood: Data likelihood (float)
    '''
    #TO DO
    n = y_train.shape[0]
    factor = 0
    for i in range(n):
        factor+=(y_train[i]-np.dot(X[i],w))**2
    likelihood = (1/(np.sqrt(2*np.pi*likelihood_var))**n) *(np.exp(-1/(2*
    likelihood_var) * factor))
    return likelihood
```

(b) Implement get_posterior_params.

```
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
    m0 = prior['mean']
    var0 = prior['var']
    post_mean=np.matmul((np.matmul(X.T,X)+likelihood_var*var0.getI()).getI
    (),X.T).dot(y_train)+np.matmul((likelihood_var**(-1)*np.matmul(X.T,X)).
    getI(),var0.getI()).dot(m0)+m0
    post_var = (likelihood_var**(-1)*np.matmul(X.T,X)+var0.getI()).getI()
    return post_mean, post_var
```

(c) Implement get_predictive_params.

```
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.matmul(post_mean.T,X_new)
    pred_var = np.matmul(np.matmul(X_new.T,post_var),X_new)+likelihood_var

    return pred_mean, pred_var
```

6.1 $p(D|\theta) = \theta^2(1-\theta)$

6.2 (H,H,T) (H,T,H) (T,H,H) $\Rightarrow$ 3 different sequences

probability $= \theta^2(1-\theta) \times 3 = 3\theta^2(1-\theta)$

6.3 $p(D|\theta) = \theta^{n_h}(1-\theta)^{n_t} \cdot C_{n_h+n_t}^{n_h}$

6.4 $\log p(D|\theta) = n_h \log\theta + n_t \log(1-\theta) + \log(C_{n_h+n_t}^{n_h})$

when $\dfrac{\partial \log p(D|\theta)}{\partial\theta} = n_h \cdot \dfrac{1}{\theta} - \dfrac{n_t}{1-\theta} = 0$, $\theta = \hat{\theta}_{MLE}$

$$\frac{n_h}{\theta} = \frac{n_t}{1-\theta}$$

$$n_t \cdot \theta = n_h - n_h \cdot \theta$$

$$\therefore \hat{\theta}_{MLE} = \frac{n_h}{n_h + n_t}$$